Projektovanje baze podataka

Gavrilo Jovanović

Predgovor

Ova skripta je namenjena kao pomoćno sredstvo za spremanje teorije iz predmeta *Projektovanje baza podataka* koje se održalo 2020/2021 godine. Ovo gradivo je nedovoljno za celokupno znanje, jer ne sadrži direktne odgovore na sva isptna pitanja, ali služi kao sveokupni skup gradiva pretvorenu u literaturu. Za više informacija koje nedostaju mogu da se nađu na sajtu *profesora Saše Malkova* koji je održavao ovaj kurs ili na internetu generalno. Želeo bih i da se zahvalim koleginici Mini Milošević koja iskucala zbirku pitanja i odgovora čiji su materijali pomogli pri pravljenju ove skripte.

Srećno sa učenjem.

Uvod

Moderne baze podataka(BP) imaju sledeće karakteristike:

- *predefinisane operacije sa podacima i strukturama podataka* - veoma su lako koristljivi.
- *sigurnost* - operacije koje su namenjene za čuvanje privatnosti su jako dobre.
- *transakcije* - atomične operacije koje zahtevaju prenos podataka su čvrsto definisane.
- *mogućnost višeprocesnog rada* - povezano sa distribuiranim programiranjem
- *deklarativno programiranje* - postavljanje upita
- *integritet podataka* - baze podataka su uvek konzistentni sa svojom logikom
- *programski interfejsi* - razdvojenost aplikativnog sloja dovodi do dobrog definisanog interfejsa
- *udaljeni pristup podataka*

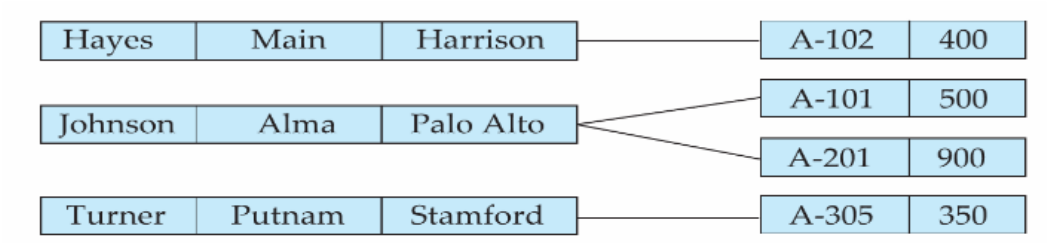
Sve ove karakteristike bi trebale da odgovaraju na ciljeve upravljanja podataka:

1. *Raspoloživost podataka* - razumna cena, smisleni format, jednostavni pristup, aplikativni interfejsi...
2. *Integritet podataka* - konzistentnost, atomičnost transakcije, višekorisnički rad...
3. *Sigurnost i zaštita podataka* - privatnost podataka, autorizacija...
4. *Održavanje sistema* - alat za upravljanje, automizacija upravljanja, obezbedjenje od kvarova...
5. *Nezavisnost od aplikacija* - skrivenost fizičke i logičke strukture

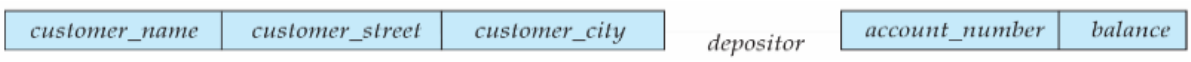
Istorijski model podataka se razvijao:

1. **mrežni model** - Sličan je dijagramskom modelu u programiranju. *Slogovi* su instance jednog *entiteta*, a slog ima više polja koje se nazivaju *atributi*. Ti slogovi se povezuju *link*-ovima koje služe kao pokazivači.

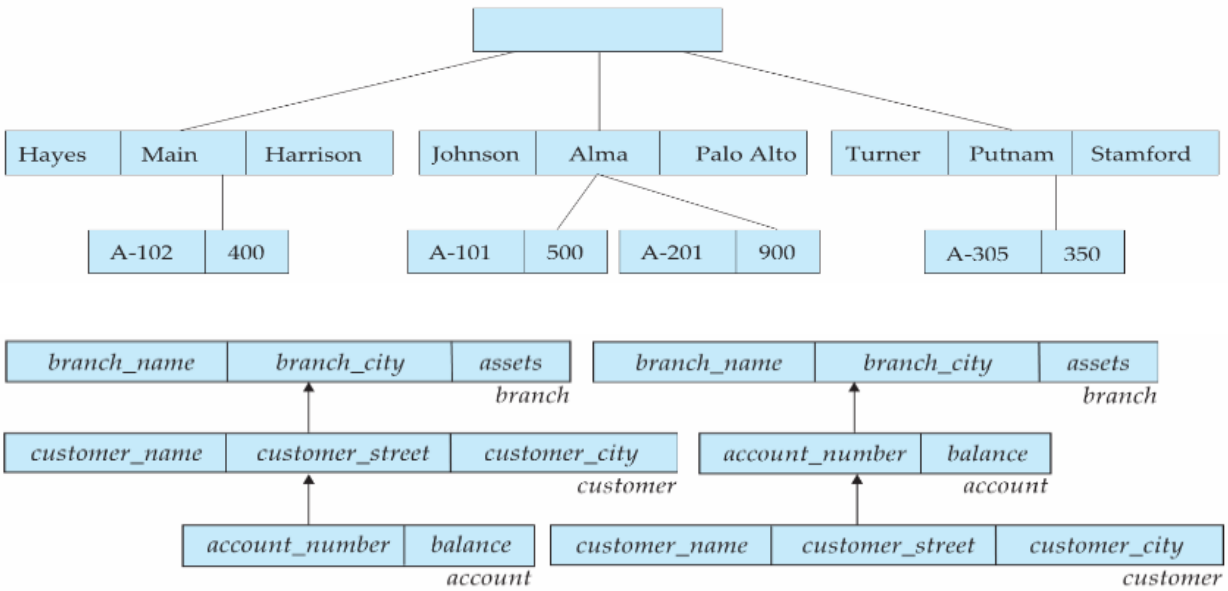
Пример података:



Пример модела:



2. **hijerarhijski** - Struktuiran je kao mešavina mrežnog i stablastom strukturom. Postoji *dummy* čvor i od njega se vezuju entiteti. Ne postoje veze koje su višestruke pošto treba da postoji jedinstveni put ka korenu. Hijerarhija je nivo entiteta, kao u B-stablu.



Sledeći modeli će kasnije biti objašnjeni:

3. **relacioni model**
4. **prošireni relacioni model**
5. **objektni model**
6. **objektno relacioni model**

Arhitektura

U opštem smislu mi apstahujemo bazu podataka na 4 dela:

- *Spoljašnji nivo* - ono što korisnik vidi
- *Konceptualno(logički) nivo* - logička struktura podataka
- *Nivo fizičkih podataka* - fizička organizacija podataka u softverskom sistemu.
- *Nivo fizičkih uredjaja* - fizička organizacija podataka na fizičkim uredjajima

Pored ovoga moramo i da imamo dobar pogled na arhitekturu baze. Ne postoji univerzalni čist pogled

Prvi pogled koji gledamo je ugao **komponenti** sistema. Naš *sistem za upravljanjem baze podataka*(skraćeno SUBP) se sastoji od komponenata koje imaju funkcije koje na neki način predstavlja interakciju izmedju njih. Putem definisanih interakcija imamo punu fukncionalnost sistema. Ovo je neophodni pogled pri implementaciji, ali ne i dovoljno jer moraju da se definišu i funkcionalnosti komponenti.

Drugi ugao je pristup **funkciji** sistema. Svaku gorenavedenu komponentu grupišemo za različitu klasu korisnika i funkcija koje sistem za njih obavlja. Pomoću ovog pogleda imamo jasni uvid na funkciju i cilja našeg sistema, ali ne i uvidu ostvarenja istog.

Treči pogled je ugao **podataka** sistema. Prepoznajemo različite vrste podataka tako da ih razeljujemo u funkcionalne jedinice koje operišu sa njima. Pošto su podaci centralna tema SUBP-a ovaj pristup deluje kao najpovoljniji, iako ne opisuje funkcionalne celine.

Kao kompromis je 1975. godine se predložila ***ANSI/SPARC arhitektura***(*American National Standards Institute / Standard Planning and Requirements*). Načelno počiva na pogledu nad podacima iako je to objedinjen pogled na arhitekturu baze podataka. Prepoznajemo tri nivoa pogleda na podatke:

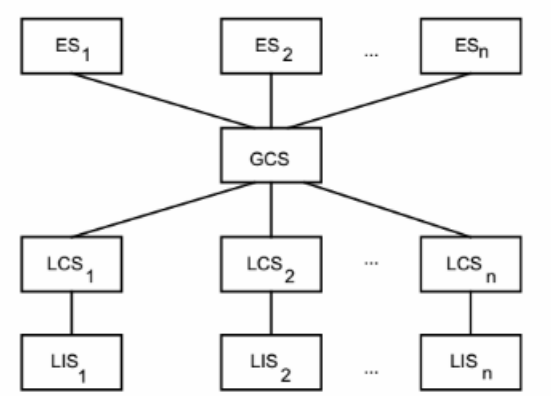
- *Spoljašnja shema* - Ovo je najviši nivo apstrakcije. Odvojeno je i od implementacije i od ograničenja modela podataka. Drugačije gledamo kao pogled na bazu iz ugla korisnika. Za različite korisnike imamo i više spoljašnjih shema.
- *Konceptualna shema* - Opisuje medjusloj, tj. kako su podaci implementirani van fizilčkog sloja,a prilagodjeni odredjenom pogledu.
- *Interna shema* - Opisuje neposrednu implementaciju podataka u SUBP-u. Ovo je fizička implementacija i najniži sloj.

Dobra stvar sa ANSI/SPARC arhitekturom je da ima dobar odnos izmedju relacionih baza podataka:

ANSI/SPARC	Relacione baze podataka
spoljašnja shema	eksterni nivo čine pogledi koje pružaju denormalizovanu sliku domena
konceptualna shema	obuhvata opise atributa, ključeva i ograničenja, uključujući i strane ključeve
interna shema	interni nivo čine fizički aspekti, indeksi, prostori za tabele i razne optimizacije

Deo ANSI/SPARC-a pripada i **arhitektura klijent/server**. Ova arhitektura generalno gleda da *server* pruža usluge dok ih *klijent* koristi. Server je zadužen za upravljanje podacima - obrada upita, optimizavije, transakcije...dok klijent ostvaruje komunikaciju izmedju aplikacije i servera.

Bitan deo arhitekutre je i **distribuiranost**. Ona nam govori da je arhitektura načinenja sa više servera i deljivim uslugama.



Gore vidimo arhitekturu sa ravnomernim čvorovima, gde prepoznajemo sledeće delove:

- GSC(global conceptual scheme) - koja objedinjuje sve lokalne koncepte
- LCS(local conceptual scheme) - koje opisuje koncept za jednog korisnika
- LIS(local internal scheme) - koja opisuje implementaciju za svakog korisnika
- ES(external scheme) - eksterna shema za svaku lokalnu shemu.

Odnos ANSI/SPARC arhitekture i modeliranje podataka:

Modeliranje podataka	ANSI/SPARC arhitektura
Konceptualno projektovanje	Odgovara spoljašnjoj shemi i delu konceptualne sheme. Pravi se apstraktan konceptualni model koji objedinjuje sve spoljašnje sheme.
Logičko projektovanje	Uglavnom odgovara konceptualnoj shemi. Prilagođava se konceptualni model konkretnom modelu podataka i pravi se konkretan logički model.
Fizičko projektovanje	Odnosi se na internu shemu i neke aspekte konceptualne sheme. Pravi se fizički model.

Relacione baze podataka

Da bismo razumeli relacione baze podataka moramo da razumemo njegov koncept *objedinjenog modeliranja*. I **entiteti** i **odnosi** se modeliraju **relacijama**. Relacije se sastoji od **atributa** koje imaju *imena* i *domene*. Skup imena i domena relacije predstavljaju **shemu relacije**.

Torka je skup imenovanih vrednosti. Torka koja ima isti broj vrednosti, njihove nazive i domene kao atributi jedne relacije predstavlja *instancu domena(sheme) relacije*. Vrednost relacije je skup instanci njenog domena.

Relacioni model prepoznaje sledeće delove:

- *Strukturni deo* - Način modeliranja podataka. Osnovna ideja je da se sve modelira relacijama. Sve znači i entiti i odnosi. *Entiteti su svi različiti postojeći elementi sistema koje modeliramo bazom podataka. Odnosi su značajni medjusobni odnosi dvaju ili više entiteta*.
- *Manipulativni deo* - Deo rukovanje modeliranim podacima. Ključan pojam je ovde **upit**. To je relacija koja je dobijena iz već postojaće relacije RBP-a. Pored toga imamo i ažuiranje baze podataka.
- *Integritetni deo* - Način obezbedjivanje valjanosti podataka, koje se vrše automatizacijom pomoću odredjenih mehanizama. Stanje baze je konzistentno ako BP ispunjava sve uslove integriteta. Promena baze podataka je dopušteno akko prevodi bazu iz jednog u drugo ispravno stanje.

Entitet ke objekat sistema koje modeliramo u bazama podataka. Neka je skup entiteta E koji gledamo, a ***atributi*** $A(E)$. Atributi su valjani akko:

- A_i je funkcija slike i-tog elementa u domen D_i
- A_i ima svoj jedinstven naziv t_i

$\forall e \in E, A_i(e) \in D_i$(slika A_i atributa). Slika skupa entiteta funkcijom α je skup $\alpha(E) = R$. Ona je ***relacija*** akko je α "1-1" preslikavanje i $Dom(R) = D_1 \times \ldots \times D_n$ i $Kol(R) = t_1, \ldots, t_n$.

Natključ K relacije R je podskup atributa koje jednoznačno odredjuju torku relacije. ***Ključ*** K je minimalni natključ kojeg poseduje svaka relacija. Natključ ima bar jedan ključ.

Relaciona baza podataka je skup relacije. Opis relacije čine domen relacije i nazivi atributa. ***Relaciona shema*** je skup opisa relacija koje čine bazu podataka.

Manipulativni deo relacionog modela

Manipulativni deo relacionog modela stavlja u prvi plan pojam upita. Upit je definicija nove relacije na osnovu već poznatih relacija baze podataka. Pored upita, važno mesto zauzima ažuriranje baze podataka. Ažuriranje baze podataka je zamenjivanje vrednosti promenljive baze podataka novom vrednošću baze podataka. Da bismo ovo korektno radili moramo da definišemo algebru baza podataka.

Relaciona algebra je proširenje skupovne algebre, koju čine uobičajne skupovne operacije kao unija,presek i proizvod. Pored toga imamo i neke osnovne operacije relacione algebre:

- *Projekcija* - izvlačenje torki relacije R po izboru kolona. Rezultat projekcije može da ima manje elemenata nego polazna relacija. Projekcija je vertikalno "sečenje" tabele.
- *Restrikcija* - izbor torki relacije R u kojima neki od atributa zadovoljava odredjene uslove. Ako radimo restrikciju po ključu sa operacijom jednakosti, onda sigurno dobijamo samo jednu torku. Restrikciju nazivamo i horizontalnim "sečenjem" tabele.
- *Prirodno spajanje* - Spajaju se torke jedne relacije sa torkama druge relacije koje imaju jednake vrednosti zajedničkih atributa. Rezulzaz sadrži samo jednu kopiju zajedničkih atributa. Može sadržati najmanje nula torki, a najviše $M \times N$ torki, gde su M i N brojevi torki relacija R i Q koje spajamo.
- *Slobodno spajanje* - Relacije R i Q se spajaju po nekom binarnom predikatu koja mora da ispunjava uslov. Ovo je restrikcija nad prirodnim spajanjem i opštija je od prirodnog spajanja.
- *Spoljašnje operacije*:
  - *Unija* - relacije R i Q se spajaju tako da svaki njihovi elementi, pa i oni koji nemaju zajedničke atribute imaju red u relaciji. Ti nezajedničke unije će da imaju torku sa svim nedefinisanim vrednostima.
  - *Presek* - će uvek biti prazan jer se porede nedefinisane vrednosti po tačnim pravilima.
  - *Razlika* - če uvek biti jednaka prvoj relaciji jer je presek prazan.

Pored relacione algebre postoji i *relacioni račun*. To je tip predikatskog računa na relacijama. Kod je definisao relacioni račun n-torki i relacioni račun domena i on je ekvivalentan relacionoj algebri. Koristimo kvantifikatore *forall* i *exists*.

Izraz relacionog računa n-torki je $\{(t_1, \ldots, t_n) : f\}$, za koje važi:

- promenljive $t_1, \ldots, t_n$ su n-torne promenljive ili indeksirane n-torne promenljive oblika $t[i]$ gde je i redni broj atributa torke t.
- skup n-tornih promenljivih čine ciljnu listu
- formula f je kvalifikacioni izraz.
- slobodne promenljive u kvalifikacionom izrazu su one i samo one koje čine ciljnu listu.

Relacioni račun domena liči na račun n-torki. Razlika je da se promenljive vezuju za domene atributa a ne za domene torki. Izraz relacionog domena je $\{(x_1, \ldots, x_k)|f\}$ za koje važi:

- promenljive $x_1, \ldots, x_k$ su domenske promenljive
- f je formula relacionog domena računa čije su slobodne promenljive $x_1, \ldots, x_k$
- formule se definišu slično kao u relacionom računu n-torki.

Iz ugla teorije, ključno je da se utvrdi da li je upitni jezik dobar da se sa njima može iskazati bilo koji upit. U praksi nam je potrebna i efikasnost. Optimizacija upita obuhvata poslove analize upita i pronalaženje najefikasnijeg puta za izračunavanje i tu su nam upitni jezici veoma jednostavni za modifikaciju.

Ažuriranje baze podataka se definiše kao zamenjivanje vrednosti promenljive baze podataka novom vrednošću baze podataka. Promenljivost baze podataka se na taj način apstrahuje sekvencom vrednosti promenljive baze podataka koje ona dobija u različitim vremenskim tačkama.

Integritet i konzistentnost u relacionim bazama podataka

Konzistentnost je saglasnost sadržaja baze podataka sa stvarnim prostorom u kojim modeliramo našu bazu. Predstavlja tačnost ili ispravnost sadržaja baze podataka.

Integritet baze podataka podrazumeva logičku ispravnost baze. *Integritetni deo relacionog modela* čine koncepti i mehanizmi koji omogućavaju da se automatizuje proveravanje zadovoljenosti određenih uslova.

Neke osnovne vrste opšteg integriteta su:

- *Integritet domena*:
  - Relacija ima tačan domen.
  - Svaki atribut odgovara odgovarajućem vrednosti iz domena.
  - **Domen** = dužina podataka, opciona jedinstvenost/podrazumevana vrednost.
- *Integritet primarnog ključa*:
  - Svaka relacija ima bar jedan ključ.
  - Domen ključa ne sme biti nedefinisan.
  - Može da se sastoji od jedne ili više kolona.
- *Integritet jedinstvenosti*:
  - Dve torke jedne relacije ne smeju imati iste vrednosti primarnog ključa.
  - Pored primarnog ključa imamo i jedinstvene ključeve, koji su slični kao i primarni, sa namenom implementacije dodatnih uslova jedinstvenosti toriki.
- *Referencijalni integritet* - Mora da važi sledeće:

1. Ne briše se torka koja ima isti ključ kao i druga torka druge relacije
2. Ne dodaje se torka koja ima loš referentni ključ
3. Referenca ako ima nedefinisanu vrednost, mora i na drugim atributima da ima istu

- *Integritet stranog ključa* - Neka je FK strani ključ relacije R, primaran ključ u relaciji B:
  1. FK u R je primarni ključ(PK) u relaciji B.
  2. domen FK-a je isti kao domen PK-a.
  3. svaka vrednost torke FK-a je identična barem jednoj torki iz PK-a
- *Pravila brisanja i ažuiranja* - Ako se briše ili ažuira neka torka iz relacije A zavisne od B tada možemo imati:
  - aktivnu zabranu - RESTRICT
  - pasivnu zabranu - NO ACTION
  - kaskadno brisanje - CASCADE
  - postavljanje nedefinisane vrednosti - SET NULL
  - postavljanje podrazumevane vrednosti - SET DEFAULT

Specifični uslovi integriteta u bazi su:

- *Na atributu* - Prvenstveno gledamo domen svakog atributa u zasebnim torkama. Možemo da ga sužavamo, gledamo ispravnost.
- *Na torki* - Isto je lokalnog karaktera i gledamo da li se atributi torke zajedno logički slažu u okviru nje.
- *Na relaciji* - Globalnog je karaktera i gledamo da li skup torki održavaju saglasnost u okviru date relacije u kojoj se nalaze.
- *Na bazi podataka* - Globalnog je karatkera i gledamo da li torke iz jedne relacije utiču na valjanost torke druge relacije.

Uslovi integriteta se proveravaju pri svakoj promeni sadržaja baze podataka. Nekada je potrebno da se uslovi proveravaju tek na kraju transakcije. Većina RSUBP to omogućava na da se menja podrazumevani način rada, a isto tako može da se menja način rada u tekućoj transakciji.

Aktivno održavanje integriteta podrazumeva da se integritet održava tako što se na određene promene reaguje pokretanjem *eksplicitno definisanog programskog koda*. Mehanizam aktivnog održavanja integriteta su *okidači*. Okidači su programi koji reaguju na svaku promenu u bazama podataka. To su upiti koji proveravaju ispravnost relacije gde je definisana. Može da proverava uslov pre ili posle svakog pokušaja promene(nekada je svejedno). Na jednoj relaciji možemo da stavimo i više okidača u zavisnosti od problema. Pored običnih okidača imamo i *okidače nad pogledima*. Izvršavaju se umesto naredbe za dodavanje, menjanje ili brisanje torki iz pogleda. Omogućavaju preusmeravanje izmena na relacije na kojima počiva pogled, ali i dalje od toga, na sasvim druge relacije. Za razliku od običnih okidača, ovi bolje kriju konceptualnu shemu od spoljašnje.

Dijagrami

E-R dijagram

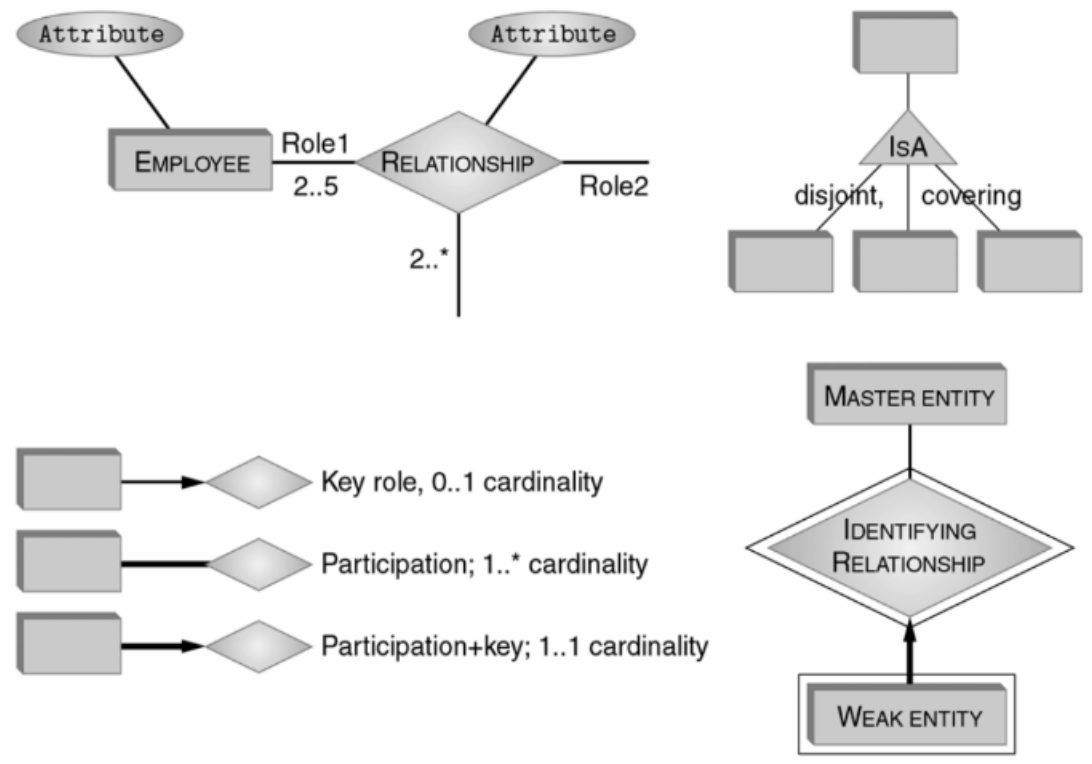
Prvi dijagram koji ćemo videti je *E-R dijagram*. On je veoma bitan u konceptualnom modeliranju i oslanja se na **model entiteta i odnosa(ili skraćeno ERM)**. Ovaj model je uveo Piter Čen 1976. godine kao alternativa prethodnim modelima jer je smatrao da najprirodnije gleda na stvarnim svet, tj. apstraktniji je. Pošto se model sastoji od *entiteta* i *odnosa* on čuva bitne semantičke infmormacije o domenu koji modeliramo. Prepoznajemo četiri nivoa pogleda na podatke:

1. *"nivo informacije - šta imamo u glavi"* - koceptualni nivo
2. *"strukturne informacije - kako se predstavljaju"* - logički nivo
3. *"strukture podataka nezavisne od prstupnog puta - kako se organizuje u memoriji"* - viši fizički nivo
4. *"strukture podataka zavisne pd pristupnog puta - kako se organizuje na hardveru"* - niži fizički nivo

ERM se naviše bavi sa prva dva nivoa.

U prvom nivou indetifikujemo entitete. **Entiteti** su stvari koji mogu jednoznačno da indetifikujemo, dok su **odnosi** medjusobno pridruživanje entiteta. Entiteti se klasifikuju u skup entiteta gde proveravamo da li se neko tu nalazi. Skup odnosa je matematička relacija izmedju *N* entiteta koji pripadaju skupovima entiteta. Skup entiteta i odnosa čine **relaciju**.

Na drugom nivou ustanovljavamo integritet. Imamo **primarne ključeve** i **atribute** koje služe da opisuje odnose izmedju entiteta. Postoji *osnovni atribut*(npr ime,prezime,...), *definisane atribute*(npr. *surogat ključ* - veštački dodat ID) i *izvedene atribute*.



Osnovni pojmovi ER dijagrama

Gore vidimo dva tipa entiteta. *Jak entitet*(*master entity*) je onaj koji indetifikuje sam sebe i nezavaisan je od drugih entiteta u bazi podataka. *Slab entitet*(*weak entity*) je onaj čija je egzistencija uslovljena postojanjem jačeg entiteta. Npr. fakultet može da egzistira bez studenta, dok pojedinačni student ne može bez fakulteta. Ovaj odnos je jedan više u kome više može da bude slabijeg entiteta. *Ključevi* se označavaju kao podvučeni atributi entiteta. *Specijalizacija/generalizacija* se isto definiše kao hijerarhija u objektnom modelu, a ovde se označava kao trougao. Kardinalnost se označava sa druge strane entiteta kao (*MinOccurance,MaxOccurance*).

Pojmovi:

- *Uslov ključa* - Ako na osnovu entiteta jednoznačno možemo da odredimo odnos u kome učestvuje.
- *Puno učešće* - Svaki entitet iz skupa entiteta učestvuje u bar jednom odnosu(puna linija). Inače je *parcijalno učešće*.
- *Agregacija* - odnosi u koje ulaze relacije(znači skup entiteta i odnosa) sa drugim entitetima i odnosima. Označavaj se izprekidanom linijom. a zajedno se označava kao dijamant-pravougaonik.

ER je doprineo svetu u kome je stvoren. Dobro je prilagođen koceptualnom i semantičkom modeliranju. Uvodjenjem dijagramske tehnike lako se predstavljaju konompleksne veze a da se pri tome ne gube semantičke informacije. Kada je napravimo ER dijagram se lako prevodi u SQL shemu kao je bitna u logičkom modelu.

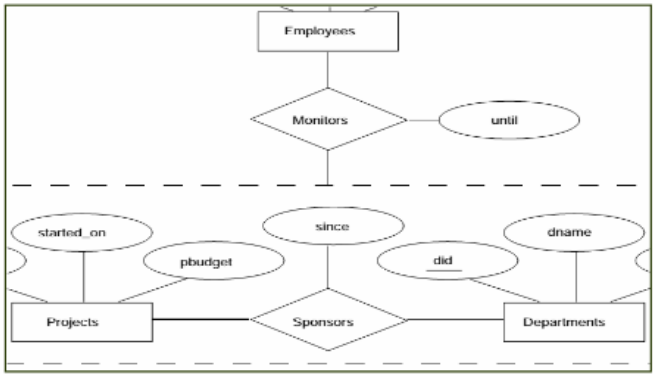
Sa druge strane ERM nije savršen. Postoje neke nedoumice u njemu:

- Složeni atributi ili postojanje novog entiteta?
- Koja je realna razlika između entiteta i odnosa? Da li je brak enititet ili odnos?
- Kako da se suočimo sa dijagramima koje imaju preko sedam entiteta? Preobimni su.

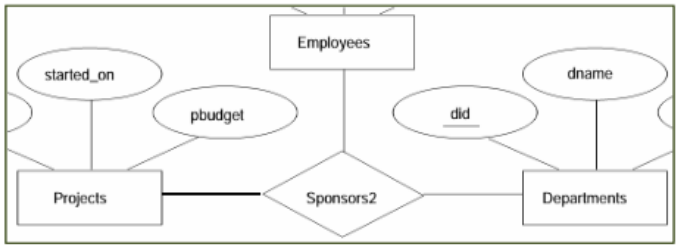
Idemo sada na neke najbitnije dileme:

1. *Entitet ili atribut* - Ako imamo složeni atribut, da li da ga: ostavimo takvog, razložimo na više atributa ili ga pretvorimo u novi entitet.
   - Argument za običan atribut - Jednom entitetu odgovara jedna vresnost koja je skalarna najčešće. U više entiteta dat atribut može da ima nezavisne vrednosti.
   - Složen atribut ima smisla - Čuvamo internu strukturu podataka koji je bitan. Ovaj atribut isto ima nezavisne vrednosti.
   - Entitet ima smisla - Ako nekom entitetu odgovara istovremeno više vrednosti tog atributa ili medjuzavisnost atributa.
2. *Entitet ili odnos* - Odnos je "odnos" ako se svi njegovi atributi odnose na instancu odnosa. Odnos je "entitet" ako važi suprotno.
3. *Složeni odnos ili više binarnih odnosa* - Složeni odnos uvek može da se podeli u više binarnih odnosa i oni su uvek poželjniji. Ostavljamo složene odnose samo kada su nam semantički neophodni.
4. *Agregacija ili ternarni model* - Agregacija je generalno uvek bolja kada:
   - Ako nekada postoje samo deo odnosa, ali ne i ceo
   - Ako se deo atributa odnosa samo na tu agregaciju

   Inače koristimo ternarne odnose.

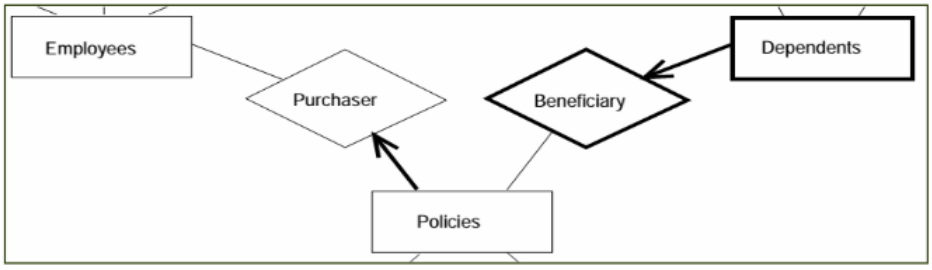


•Агрегирани однос



•Тернарни однос

•Више бинарних односа



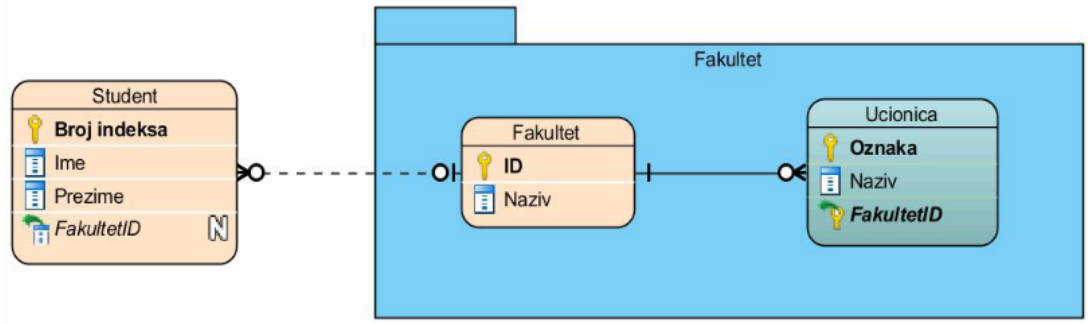
Prevođenje ERM u relacioni nije komplikovan postupak. Svaki entitet se prevodi u relaciju. Atributi entiteta se prevode u atribute relacije. Ključni atributi se prevode u primarni ključ. Većina odnosa postaju relacije dodaju se i kljulni atributi povezanih entiteta. Odnosi se ne prave u relacije ako je odnos izmedju entiteta 0..1. Slabi entitet i odnos sa jačim entitetom se modeliraju jednom relacijom u kojem imamo strani ključ ka jačem entitetom koji je deo primarnog ključa.

Prevodjenje hijerahije ostavlja više načina razreševanja. Pokazaćemo tri metode, ali one ne moraju striktno da se koriste, može i da se meša:

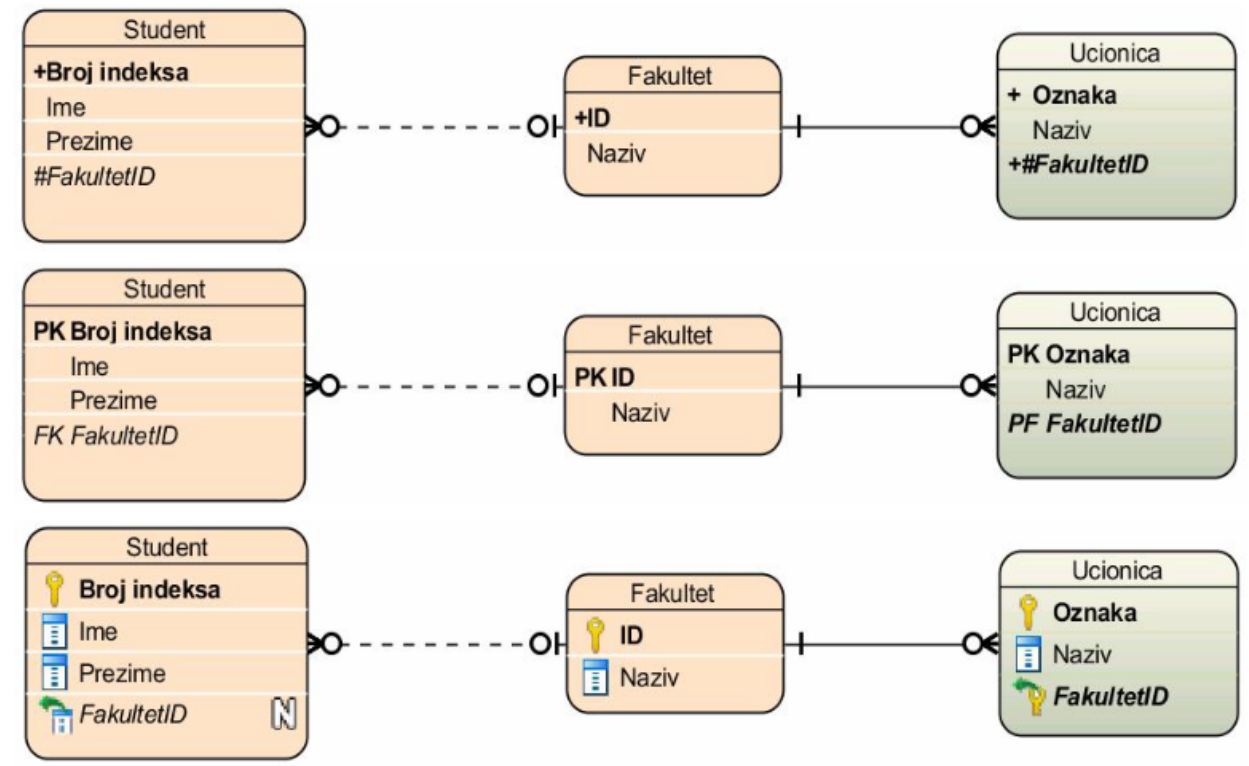
1. *Cela hijerarhija u jednu relaciju* - Sve entitete i atribute stavljamo u jednu relaciju. Efikasna je ali zazuma dosta prostora.
2. *Svaki entitet u zasebnu relaciju* - Pravimo relacije sa stranim ključem u neposrednu baznu relaciju. Zahteva dosta spajanja pri čitanju, i zbog toga je neefikasna.
3. *Svaki entitet-list u zasebnu relaciju* - Dok je za sve ove entitete efikasno, za sve koje nisu se ili pravi relacija sa primarnim ključem ili se izostavlja.

Dijagram tabela

Dijagrami tabela su na jakom niskom nivou. Više govore o implementaciji nego o semantici, za razliku od ER-a. Ovaj model se malo koristi, tako da ćemo ga ukratko preći.



Primer predstavljanja grupisanja tabela



Primer nekoliko načina predstavljanja dijagrama tabela

Osnovni elementi:

- *Tabele* - Predstavljene zaobljenim pravougaonicima, gornji deo sadrži naslov, donji atribute
- *Kolone* - Označene simbolima, malim slovom ili crtežima.
- *Odnosi* - Prikazani su isprekidanim linijama, a mogu da se predstave i stranim ključem
- *Grupisanje* - Predstavimo velikim "folderima" koje sadrže više tabela.
- *Primerni ključ* - Prikazano kao +, PK ili ključ.
- *Strani ključ* - Prikazano kao #, FK ili ključ sa strelicom.

Običan odnos obično prikazujemo isprekidanim linijama. Kardinalnosti se pokazuju na linijama i sa njihove strane entiteta. Postoje simboli: **o** - 0, | - 1, ⋞ - više. Ako prikazujemo vezu jakog entiteta prema slabim, onda koristimo punu liniju.

Postoje još dodatnih elemenata dijagrama tabela:

- *Pogledi* - Prikazuju se kao pravougaonici sa dodanim simbolima i sa vezanim linijama prema tabeli koje gledaju.
- *Sekvence* - Prikazujemo kao pravougaonik sa desno zaobljenom ivicom i vezivnom linijom prema odgovarajućom tabelom.
- *Okidači* - Prikazujemo kao pravougaonik sa levo zaobljenom ivicom koja sadrži nazive indeksa i vezivnom linijom prema odgovarajućom tabelom.

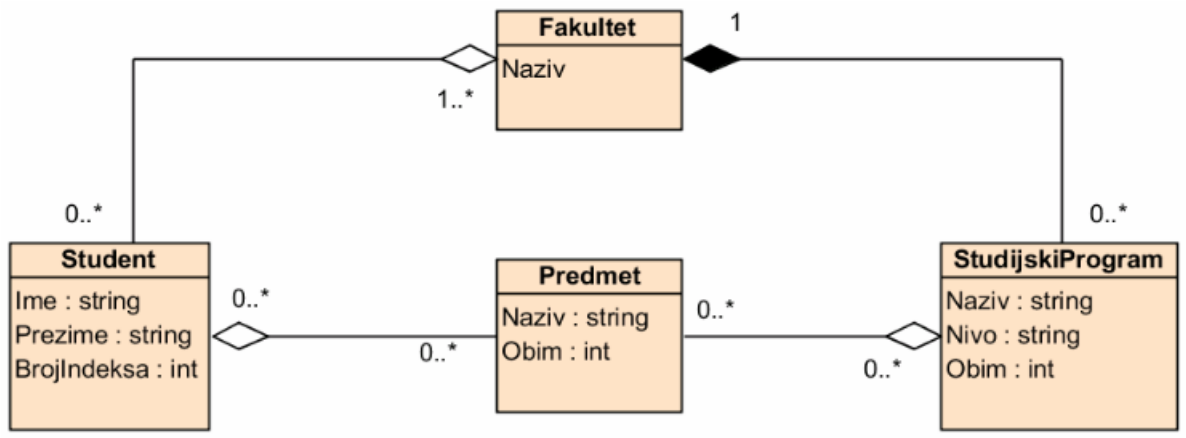
Ovakav dijagram kakav smo prikazali je na logičkom nivou. Na konceptualnom nivou bismo izostavili nepotrebne informacije za taj nivo, kao kolonu surogat ključa/ključa, tip kolone...Na fizičkom nivou bismo sve ovo morali da znamo u detalj.

Dijagram klasa

Dijagrami klasa su slični UML dijagramima koje služe da predstave bazu na niskom nivou i pri tom koriste "jezik" sličan objektnoj paradigmi. **UML dijagram klasa** opisuje strukturu podataka klasa. Ideja je da se isti dijagram koristi za modeliranje podataka, jer bolje predstavlja semantiku odnosa od dijagrama tabela i ER-a. Modeli klasa i relacija se konceptualno razlikuju, prvi je tip, a drugo su skupovi podataka.

Za projektovanje modela baze podataka koristi se tzv. *konceptualni dijagram klasa* koji se u UML-u naziva *dijagram klasa domena* a u radu sa bazama *dijagram klasa podataka*. Za razliku od običnih dijagrama klasa:

- u prvom planu su atributi i odnosi
- ponašanje se skoro potpuno zanemaruje
- enkapsulacija se stavlja u drugi plan



Dijagram klasa, ključeve označavamo tekstom, dok integritet opisujemo u komentarima

Dopune UML dijagrama:

- *Stereotipovi* - Predstavlja šablon. Apstrakcija opšteg slučaja nečega, neki predefinisani skup osobina. Koristi se za navodjenje dodatnih deklaracija ili napomena. Označavaju se kao u klasama: << ... >> u nazivu klase.
- *Označene vrednosti* - Opšti slučaj stereotipa. Navodi se u uglastim zagradama kao par ime-vrednost.
- *Proširenja* - Element koji se dodaje sa osnovnim entitetom strelicom sa popunjenim trouglom. Koristimo za dodavanje osobina itd.

Osnovni odnosi koje imaju dijagrami klasa podataka su:

- *Asocijacija* - Odnos izmedju dve klase. Označava da bar jedna klasa "zna" za objekte iz druge klase i komunicira sa njima. Asocijacija može da bude *funkcionalna* i *strukturna*, koja je važnija.
- *Agregacija* - Predstavlja slabiji odnos strukturne asocijacije. Jedna klasa se sadrži od objekata druge klase.
- *Kompozicija* - Predstavlja jači oblik strukturne asocijacije. On je:
  - binarni odnos
  - odnos celina-deo
  - jedan deo može da pripada samo jednom složenom objektu
  - ako se složeni objekat briše, onda se i delovi brišu
- *Nasledjivanje* - Isto je kao u OO modeliranju
- *Navigacija* - model klasa podataka načelno podrazumeva da se za referisanje drugih objekata koriste neki vidovi jedinstvenih identifikatora tzv. OID. Takvi ključevi identifikuju promenljive, a ne podatke.

Dijagrami klasa se veoma slično koriste na nivoima kao i dijagrami tabela. Na svakom nivou važi sledeće:

- *Konceptualni nivo* - Želimo da očuvamo semantiku odnosa iz domena problema.
- *Logički nivo* - Ostvarujemo stabilnu i jednoznačnu strukturu.
- *Fizički nivo* - Omogućavamo dobre performanse.

Proces projektovanja baze podataka

Da bismo imali dobru definisanu bazu podataka koja će da ima optimalan, konzistentan i pravilan rad, moramo da imamo dobru viziju za to kako će nam data baza izgledati. Prepoznajemo četiri faze koje svaka ima svoje korake.

I. Koncepualno projektovanje:

Ova faza odgovara spoljašnjoj shemi i delu konceptualnoj shemi. Pravimo pojedinačan konceptualni model za svaku od spoljašnjih shema, a zatim pomoću njih pravimo objedinjeni konceptualni model. Imamo četiri koraka:

1. **Analiza zahteva** - Veoma je važna i zahtevna. Tesno je povezana sa procesom analize zahteva u konkretnom projektovanju informacionih sistema. Moramo detaljno razumeti procese. Ciljevi su nam razumevanje:
   - podataka - strukture podataka, obima podataka, ...
   - aplikacije - potrebe. učestalost upita, performanse, ...
2. **Konceptualno projektovanje pojedinačnih baze podataka** - Pravljenje modela podataka za svaku spoljašnju shemu zasebno na visokom nivou. Sve se opisuje iz ugla korisnika. Važno je odrediti semantiku podataka i odnosa. Trebamo da opisujemo strukturu podataka, odnose izmedju struktura podatka, uslove integriteta...
3. **Objedinjeni pogledi** - Pravljenje objedinjenog apstraktnog modela podataka gde ujednačujemo rečnik sa opisom domena. Integrišemo pogled u jedinstveni model.
4. **Grupisanje entiteta** - Pravljenje više preglednijih dijagrama modela, grupisanje jače povezanih entiteta u celine i uočavanje centralnih entiteta za odgovarajuće grupe.

II. Logičko projektovanje:

Ova faza uglavnom odgovara konceptualnoj shemi. Prilagodjavamo konceptualni model konkretnom modelu podataka i pravimo logički model. Sastoji se iz dva koraka:

1. **Prevodjenje konceptualnog modela u logički nivo** - Svi opisi konceptualnog modela se preovodi na jezik logičkog modela i inicijalno ga pravimo. Prilagodjavamo odredjeni konceptualni model konkretnom implementacionom modelu podataka.
2. **Prečišćavanje sheme** - Dodatno prilagođavanje modela jeziku i pravilima konkretnom modelu podataka. Rešavamo potencionalne probleme, kao što su redundantnosti u relacionim bazama podataka.

III. Fizičko projektovanje:

Ova faza odgovara konceptualnoj i fizičkoj shemi. Pravimo logički model iz dva koraka:

1. **Fizičko projektovanje implementacije** - Optimizujemo logički model prema SUBP-u i modelu upotrebe. Određujemo tačnu internu shemu baze i mehanizme preslikavanja fizičke u konceptualnu shemu.
2. **Fizičko projektovanje spoljašnjih shema** - Projektujemo implementaciju spoljašnje sheme, aplikativnih interfejsa za pristupanje podacima, tj. šta korisnik sme da vidi.

IV. Projektovanje bezbednosti:

Ova faza je uporedna na svaku ostalu fazu projektovanja baze podataka. U najvećoj meri je odvajamo nakon ili u toku fizičkog projektovanja. Ova faza nam služi da prepoznamo ulogu i vrstu svakog korisnika. Po tome prepoznajemo vrstu aplikacije i korisničkih grupa tako da možemo da vidimo sve skupove privilegija koje korisnici moraju da imaju. Ovo nam je bitno zbog privatnosti odrdjenih delova podataka kao i implementaciju bezbednostnih mehanizama.

Kao što smo videli mi u prethodna četiri koraka dva puta prolazimo kroz celu arhitektutu. Do fizičkog projektovanje se spuštamo na najniži nivo, pa kada želimo da implementiramo sistem i uvedemo bezbednosne mere, vrazaćamo se na najviši nivo.

Konceptualno modeliranje

Kao što smo rekli, ***konceptualno modeliranje*** je faza koja odgovara spoljašnjoj shemi i delu konceptualnoj shemi. Pravimo pojedinačan konceptualni model za svaku od spoljašnjih shema, a zatim pomoću njih pravimo objedinjeni konceptualni model.

Analiza zahteva nam je prvi korak konceptualnog modeliranja. Ovde obuhvatamo sve bitne aspekte analize sistema. Detaljno izučavamo i modeliramo sistem. To može da obuhvati i modeliranje poslovnih i implementacionih aspekata sistema. Ova faza se po pravilu prepliće sa projektovanjem informacionih sistema i ne može da se prosto izdvoji iz nje. Glavni celjievi u okviru analize zahteva su nam:

- *Prevodjenje fukncionalnih zahteva u kontekst trajnih podataka* - ako razumemo funkcije, razumemo i podatke.
- *Fundamentalno razumevanje podataka i njihovih odnosa* - odnosi se na strukturu potrebnih podataka, svih pojedinačnih transakcija koje se medjusobno izvršavaju, domen...
- *Definisanje nefunkcionalnih zahteva* - kao što su integritet, performanse, bezbednosti i razni administrativni aspekti.
- *Odredjivanje i oblikovanje dodatnih uslova implementacije* - kao što su tehnologije, hardverske i softverske predpostavke, interfejsi,...
- *Izrada iscrpne dokumentacije za sve navedeno*
- *Dekompozicija sistema na skup slabijih povetanih segmenata* - ideja je da se projektovanje sistema podeli na manje celine koje je lakše izračunati i modelirati. Kasnije se ti delovi spajaju u fazi grupisanja.

Konceptualno projektovanje pojedinačnih spoljašnjih shema je sledeći korak i on se sastoji iz raznih *klasifikacija skupova podataka*. Ovo obuhvata prepoznavanje *klasa/entiteta* kao i *atributa*.

Opšte pravilo je da je svaka imenica kandidat za entitet ili atribut. Entiteti bi trebalo da sadrže opisne informacije. Ako nešto ima višestruku ili složenu vrednost, onda je verovatno entitet. Atributi bi trebalo da pripadaju onim entitetima koje najneposrednije opisuju.

Da bi nam baza bila funkcionalna, entiteti moraju da budu u nekoj vrsti odnosa. Pre nego što ih ustanovimo, moramo da prepoznamo:

- učesnike - koji entiteti ulaze u odnos
- kardinalnost - koliko instanci entiteta ulaze u odnos
- dodatne atribute koje opisuju odnos
- jasan naziv i semnatiku odnosa.

Ako između dva entiteta imamo više odnosa, onda su oni *redundantni*. Ovakvi odnosi su veoma opani jer imaju za rezultat teško otklonjivu redundantnost u modelu i implementaciji i ometaju konzistentnost i normalizaciju. Potrebno je eliminisati redundantne odnose ili bar naglasiti da su redundantni. Sa druge strane, ako u jednom odnosu učestvuju više od dva entiteta, onda je ovo *odnos višeg reda*. Vrlo se teško ovi odnosi debaguju, tako da ih samo ostavljamo ako su neophodni. Uvek je bolje imati više binarnih odnosa.

Konceptualno modeliranje se često odvija po delovima odvojeno za različite aplikacije ili delove aplikacije. Osnovna motivacija je da je analiza i modeliranje manjih delova domena se izvode jednostavnije i efikasnije. Manji modeli se lakše oblikuju i razumeju. Svaki pojedinačan model se naziva pogledom ili lokalnom shemom. Globalna shema se dobija integrisanjem lokalnih shema , ili pogleda.

Sa ovim dolazimo do treće faze, iliti **integrisanje pogleda**. Ovde spajamo više lokalnih shema u jednu globalnu. Cilj je kroz te sheme nalazimo informaciju za konstrukciju objedinjenog rečnika i domena našeg modela. Da bismo to uradili moramo da *prepoznamo konflikte u modelu i da ih razrešavamo*, pa onda da ih *spajamo i restruktuiramo našu lokolnu shemu*.

Konflikti su problemi u integrisanju pogleda i možemo da ih razvrstamo na:

1. *Konflikt imena* - Neujednačavanje imenovanje entiteta i odnosa. Postoji:
   - konflikt sinonima - Različitim imenima nazivamo iste kocepte. Prvi je i lako rešiv problem.
   - konflikt hononima - Isti naziv se koristi za različite koncepte. Stvaraju veliki problem oko specifikacije zadataka i njih prvo rešavamo.
2. *Konflikt struktura* - Različiti strukturni elementi se upotrebljavaju za modeliranje istih koncepata. Da bi pogledi mogli da se spoje, odgovarajući koncepti moraju da imaju identične strukutre.
3. *Konflik ključeva* - Istom elementu se u različitim pogledima dodeljuju različiti ključevi. Neophodno je da se ključevi ujednače, jer u suprotnom može doći do reduntantnih ključeva.
4. *Konflikt zavisnosti* - Različito su prepoznate funkcionalne zavisnosti.

Rešavanje gorenavedinih problema uahteva najčešće dodatnu analizu zahteva. Pritom možemo da zahtevamo ozbiljnu modifikaciju pogleda. Ode učestvuju projektanti konfliktnih pogleda.

Postoje tri principa nakon razrešenja konflikta, kojim se rukovodimo pri spajanju i restruktuiranju lokalnih shema:

- *Potpunost* - Svi koncepti iz lokalnih shema moraju da se očuvaju u potpunosti u globalnoj shemi. Nakon razrešavanja konflikta, spajamo koncepte i ostvarujemo potpunost.
- *Minimalnost* - Svi redundantni koncepti moraju da se uklone iz globalne sheme.
- Razumljivost - Globalna shema mora da bude razumljiva svim korisnicima. Po potrebi, uredjujemo model da bi bio razumljiviji.

Nakon integrisanje pogleda, poslednji korak u konceptualnom modeliranju nam je **grupisanje entiteta**. Trenutno je naša globalna shema veoma kompleksna. Teško je pregledliva i moramo je "dekomponovano gledati". To radimo uz pomoć grupacija delova modela radi predstavljanja jednog velikog modela pomoću više manjih dijagrama. Ovo nam olakšava sagledavanje sheme. Postoje četiri principa grupisanja entiteta pri pravljenju konceptualnog modela:

1. *Grpisanje prema dominantnosti* - Entiteti koji imaju veliki broj odnosa su veoma bitni i njih izdvajamo sa entitetima sa kojima je spojen i koji imaju manje veza.
2. *Grupisanje prema apstraknosti* - Ako u modelu postoji hijerarhija, koja se predstavlja jednim baznim entitetom. Ukoliko postoje bitni specifični odnosi nekih elemenata hijerarhije, onda se cela hijerarhija deli na više grupa.
3. *Grupisanje prema uslovima* - Ukoliko postoji neki složeni uslovi koji moraju da važe u nekim odnosima, grupišemo odgovarajuće entitete. Cilje je sklanjanje složenih uslova iz velikog dijagrama.
4. *Grupisanje prema odnosima* - Odnosi višeg reda i odgovarajući entiteti mogu da se grupišu i predstavljaju kao jedna celina.

Da bismo lepo grupisali entitete, moramo prvo da prepoznamo elemente koji se grupišu u okviru funkcionalne oblasti. Potom se oni grupišu tako da svaka grupa pripada funkcionalnoj oblasti. Rekurzivno potom pravimo grupe višeg reda, da bismo na kraju proveravali konzistentnost krajnjih interfejsa koje smo dobili.

Logičko modeliranje

Logički model podataka spušta konkretni model bliži implementacionom modelu tako što uzima sve potrebne informacije. Pre nego što radimo moramo da znamo konkretnu vrstu baze podataka koju koristimo. Ulaz za logičko modeliranje je konceptualni model, a izlaz je detaljni logički model. Razlika izmedju ova dva modela je da se konceptualni model brine o semantici i odnosima, dok se logički model stara da sve podatke približi implementacionom modelu.

Šta je to detaljni logički model? To je model u kome su nam poznati svi uslovi ograničenja, imamo logičku shemu trajnih objekata, poznati su svi ključevi i surogat atributi. Kada bismo preskočili ovo modeliranje izgubili bi sve ove dobre atribute, što bi rezultovalo gubljenju integriteta, kompletnosti, fleksibilnosti, efikasnosti i upotrebljivosti.

Logički model se pravi iterativno. Prva iteracija se pravi na osnovu konceptualnog modela. Svaka naredna se pravi menjanjem prethodne (*flexing*). Da bismo znali da li prelazimo sa jedne iteracije na drugu, naš logički model mora da ispunjava sledeće kriterijume:

- *Da li je upotrebljiv?*
- *Da li (ne)ispunjava funkcionalne zahteve?*
- *Da li ispunjava kompletnost?*
- *Da li pruža fleksibilnost?*
- *Da li omogućava efikasni rad?*

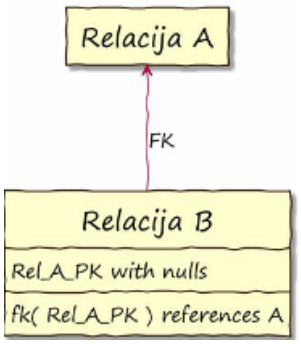
Prevođenje konceptualnog modela u logički

Iako se teorijski, logički model opisuje dijagramima klasa, praktično se služi relacioni model koji je intuitivan prelaz sa konceptualnog modela. Da bismo dobro preveli, moramo da idemo postepeno:

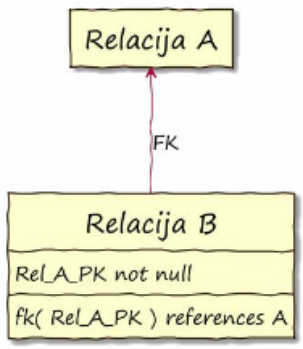
1. Sve entitete prevodimo u relaciju, pa i sve iz hijerarhije
2. Prevodimo odnose koje se tiču direktnih veza entiteta, bilo binarne ili složene
3. Sve druge odnose prevodimo
4. Opisujemo odnose stranim ključevima i njohovom kardinalnošću u konceptualnom modelu. Gledamo da li su strani ključevi obavezni ili ne, ili možda pravimo novu relaciju koja opisuje odnos.

Sada ćemo da prodjemo kroz modeliranje svih mogućih odnosa.

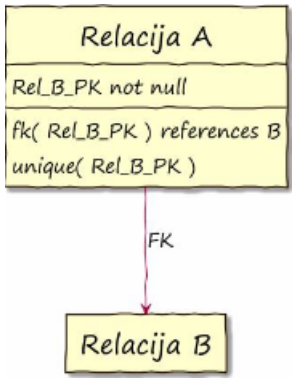
Odnos: 0..1 - 0..N - Opisuje odnos roditelja A i potomka B. Čest slučaj kod agregacije.



Odnos: 1 - 0..N - Opisuje odnos roditelja A i potomka B. Čest slučaj kod kompozicije.

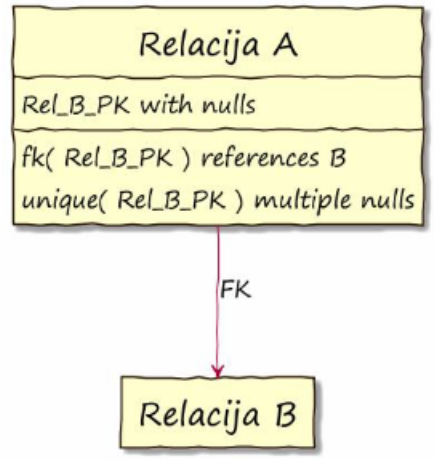


Odnos: 0..1 - 1 - Opisuje odnos nadodređenog A i podređenog B.

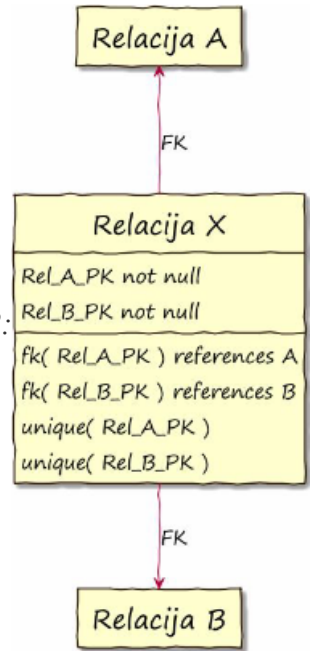


Odnos: 0..1 - 0..1 - Uobičajno je za dvosmerne opcione odnose. Najbolje rešenje je pravljenje *vezne* relacije.

Alternativa 1:

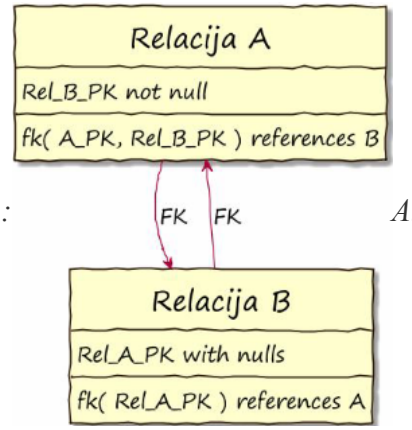


Alternativa 2:

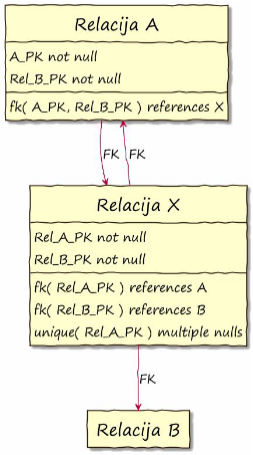


<u>Odnos: 0..1 - 1..N</u> - Uobičajeno za odnose agregacije sa obaveznim delovima.

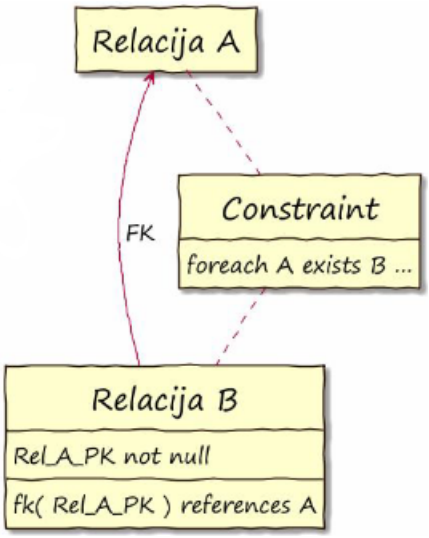
Alternativa 1:



Alternativa 2:

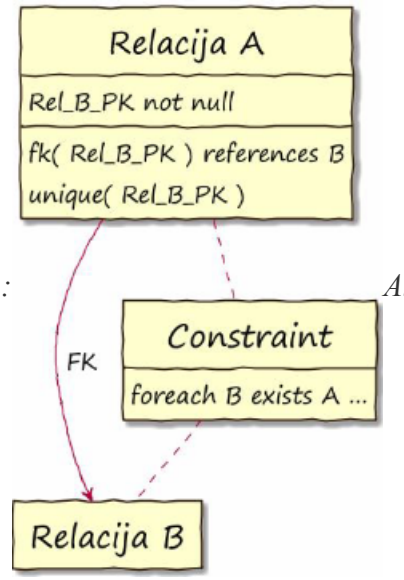


<u>Odnos: 1 - 1..N</u> - Uobičajeno za odnose kompozicije sa obaveznim delovima.

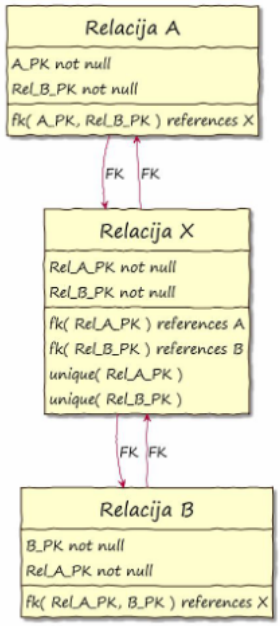


Odnos: 1 - 1 - Uobičajeno za odnose obostranog ekskluzivnog pridruživanja.

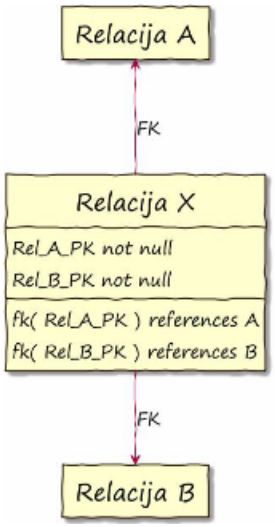
Alternativa 1:



Alternativa 2:

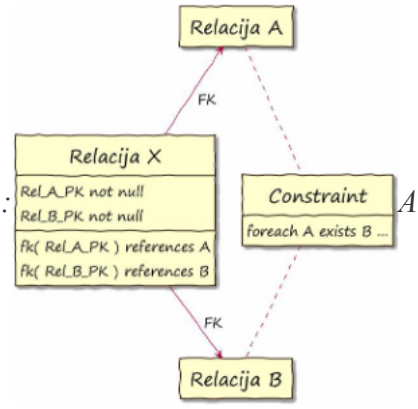


Odnos: 0..N - 0..N - Uobičajno za asocijacije bez posebnih ograničenja.

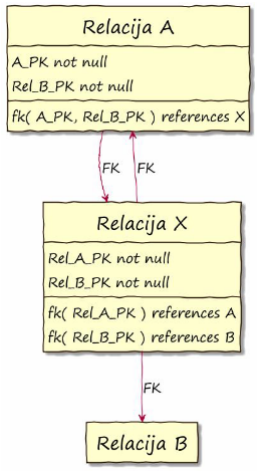


Odnos: 0..N - 1..N - Uobičajeno za odnose asocijacije slabih i jakih entiteta ili kod agregacije kod kojih deo može da čini više celina, ali ne može da postoji samostalno.

Alternativa 1:

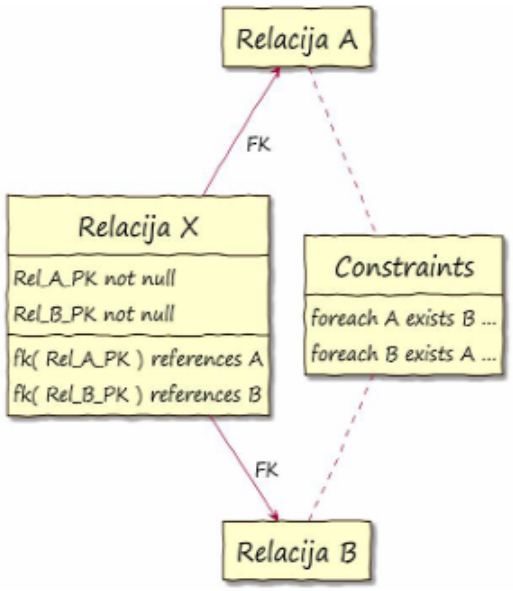


Alternativa 2:

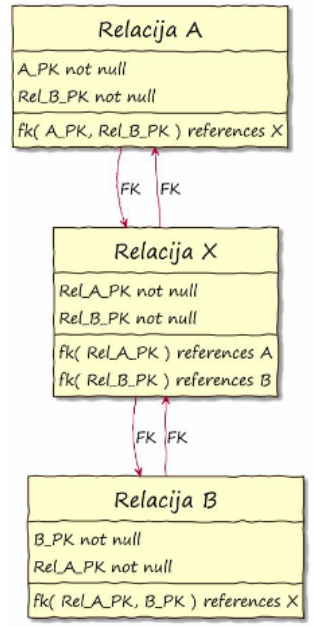


Odnos: 1..N - 1..N - Uobičajeno za odnose agregacije kod kojih deo može da čini više celina, ali ne može da postoji samostalno. Celina ne može da postoji bez delova.

Alternativa 1:



Alternativa 2:



Sada ćemo da vidimo neke binarne ciklične odnose:

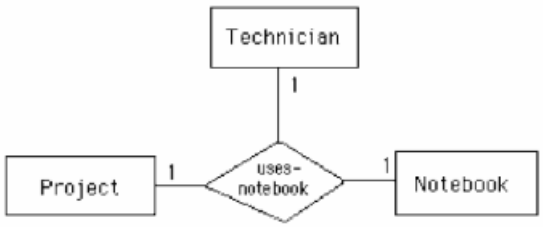
Odnos: 1 - 1 - Bilo da su opcioni ili ne, predstavljaju se dodatnim atributima stranog ključa. Ako je opcioni, sme da bude NULL. Tipičan primer za ovaj ciklični odnos je da svaki zaposleni može da bude u braku samo sa jednom osobom.

Odnos: 1 - * - Strani ključ se ovde uvodi na strani "*". Tipičan primer je da da se inžinjeri grupišu u timove, gde svaki tim ima tačno jednog lidera inžinjera.

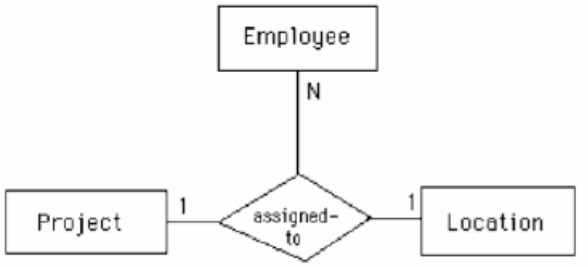
Odnos: * - * - Ovaj odnos se predstavlja novom relacijom, kao običan binarni odnos koji je *-*. Tipičan primer je da svaki službenik može da piše izveštaj sam ili sa koautorima.

Još su nam ostali odnosi sa više učesnika:

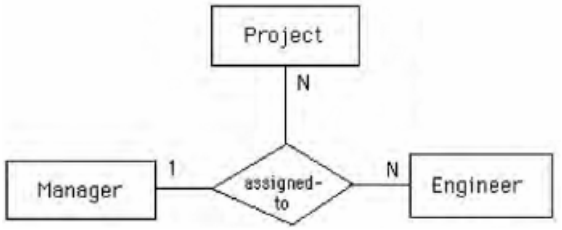
Odnos: 1 - 1 - 1 - Pravimo novu relaciju sa stranim ključevima. Zavisnosti se uređuju dopuštanjem NULL i jedinstvenim ključevima. Primer dijagrama, nova relacije je *uses-notebook*:



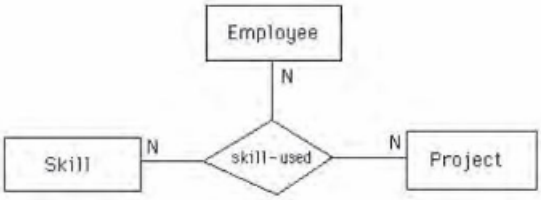
Odnos: 1 - 1 - * - Pravimo novu relaciju sa stranim ključevima. Zavisnosti se uređuju dopuštanjem NULL i jedinstvenim ključevima. Primer dijagrama, nova relacija je *assigned-to*:



Odnos: 1 - * - * - Pravimo novu relaciju sa stranim ključevima. Zavisnosti se uređuju izborom primarnog ključa. Primer dijagrama, opet pravimo novu relaciju *assigned-to*:



Odnos: * - * - * - Pravimo samo novu relaciju sa stranim ključevima. Ovde nema zavisnosti. Primer dijagrama, pravimo novu relaciju *skill-used*:



Pored svega ovoga, postoji četiri načina na koji može u logičkom modelu da se predstavi hijerhijski odnos:

1. *Generalizacija* – Cela hijerarhija se predstavlja u jednu relaciju, dok svaki entitet i entitet-list idu u posebnu relaciju, ali tako da uključi sve nasleđene atribute.
2. *Agregacija* – Svodi se na obične odnose, najčešće 1-*.
3. *Odnosi sa više od tri učesnika* – Slično je kao odnosi sa 3 učesnika. Mora više da se vodi računa o funkcionalnim zavisnostima.
4. *Slabi entiteti* – Obično ne zahteva dodatno postupanje. Prevođenje odnosa sa jakim entitetima rešava problem.

Prečišćavanje sheme

Prečišćavanje sheme je prebacivanje iz konceptualnog u logički model gde se više pažnje prebacuje na funkcionalne zavisnosti(skraćeno FZ), tačnije otklanjivanje redundantnosti. Detaljno gledamo logički model koji smo dobili do sada da bismo uskladili sve elemente sa postojaćim modelom. Posle toga ga optimtimizujemo zarad boljeg rada.

Redundantnosti mogu da nam prave dosta problema pri implementaciji kasnije. Najviše se to ogleda u:

- *Redundantnim čuvanjem podataka* - Nešto što ne moramo da držimo u bazi zauzima memoriju.
- *Anomalije ažuiranja/dodavanja/brisanja* - Ako čuvamo kopiju na više mesta moramo takom ažuiranja na svim mestima da pazimo da bismo održavali konzistentnost. Slično je i za brisanje i dodavanje.

Jedan od načina uklanjanja redundantnosti u relaciji je *dekompozicija baze*. Jednu relaciju delimo na više njih i time dobijamo veći broj relacija sa "manjim brojem atributa", čime umanjujemo redundatnost.

Definicija: Neformalno, **funkcionalna zavisnost** je odnos izmedju dva skupa atributa u okviru torki jedne relacije. Neka je relacija $R = \{X, Y\}$, gde su X i Y atributi. $X \to Y$ je FZ ako za $\forall t_1, t_2 \in R$ važi:

$$X \to Y \iff t_1.X = t_2.X \implies t_1.Y = t_2.Y$$

Relacija je ispravna ako zadovoljava FZ kao integritet. Preko nje možemo da uvedemo reduntantnosti u bazi, a postojanje FZ garantuje da menjanjem baze ne diramo semantiku početne relacije. Neke aksiome koje ima FZ su:

- *Refleksivnost* - $X \to X, X \subseteq Y \implies X \to Y$
- *Proširivost* - $X \to Y \implies (\forall Z) : XZ \to Y \land XZ \to YZ$
- *Tranzitivnost* - $X \to Y \land Y \to Z \implies X \to Z$

Jedna od dobrih osobina FZ je to da možemo da definišemo preko njih i ključeve relacija.

Definicija: Skup atributa X je **nadključ** relacije R ako $X+ = Att(R)$, to jest možemo da dobijemo zatvorenje cele relacije pomoću FZ iz X. Skup atributa je **kandidat za primarni ključ** ako je natljuč i ako za $\forall Y, Y \subseteq X \implies X = Y$, to jest važi minimalnost. Svi atributi na koje referiše ključ su funkcionalno zavisni od njega, isti ključ daje istu vrednost atributa.

Kao što smo rekli FZ garantuje da menjanjem baze ne menjamo semantiku iste. Ovo nam je veoma bitno kod dekompozicije baze. Ako dekomponujemo našu bazu, pravilima FZ bismo trebali da se vratimo u početnu relaciju.

Teorema: Ako dekompozicija čuva funkcionalne zavisnosti, onda je ona **potpuna**.

Reduntantnosti nisu očigledne u konceptualnom modelu. U E-R i UML dijagramima se ne uočavaju lako. Dodavanjem ključeva možemo da uočimo neke nepravilnosti, ali je i dalje nepotpuno. Da bismo mogli jasno da vidimo redundanosti, moramo konceptualni model korak po korak da prevodimo u relacioni. Alat koji nam pomaže pri slanjanju redundantnosti su **normalne forme**.

Normalne forme(ili NF skraćeno) su skupovi pravila vezana za relaciju koja omogućavaju da one u sebi nemaju redundantnosti. Što budemo išli dublje kroz njih ili nekih "medju-formi" dobijaćemo jače uslove za našu bazu.

1NF:

U ovoj normalnoj formi naša relacija može samo da ima atribute atomičkih vrednosti.

2NF:

Relacija je u drugoj normalnoj formi ako je u 1NF i ako nijedan neključan atribut nije funkcionalno zavisan od nekog pravog podskupa atributa nekog kandidata za ključ. Suština je da u slučaju složenog ključa svi ostali atributi zavise od celog ključa a ne od nekog njegovog dela. Narušavanje 2NF ukazuje na neispravno grupisanje atributa u relacije. Dopušta da u relaciji postoje dva ključa kandidata, kao i tranzitivne zavisnosti.

3NF:

Ako je R relacija i X neki podskup njenih atributa i A neki atribut relacije R, onda je R u trećoj normalnoj formi akko za svaku FZ $X \to A$ na relaciji R važi jedno od: $A \in X \Longrightarrow X$ je natključ ili je A deo nekog ključa relacije. To jest, relacija je u 3NF akko je u 2NF i svaki neključan atribut je netranzitivno zavisan od svakog ključa. Narušavanje 3NF ukazuje da je u relaciji modelirano više relativno nezavisnih FZ.

Normalna forma elementarnih ključeva (NFEK):

Relacija je u NFEK ako je u 3NF i akko za svaku elementarnu FZ $X \to Y$ važi sledeće: ili je X ključ ili je Z elementarni ključ ili deo elementarnog ključa. FZ $X \to Y$ je elementarna akko ne postoji FZ $Z \to Y$, ni za koji pravi podskup atributa $Z \in X$. Ključ je elementaran ako je polazište bar jedne elementarne funkcionalne zavisnosti

Bojs-Kodova normalna forma (BCNF):

Relacija je u 3NF akko $\forall$ FZ $X \to Y$, Y je atribut ili je X natključ relacije. Svaki atribut relacije je ili deo ključa ili zavisi od celog ključa. Ne postoje zavisnosti izmedju neključnim odnosima atributa niti tranzitivne zavisnosti. Relacija koja je u 3NF je najčešće i u BCNF.

Sve prethodne normalne forme smo izveli iz funkcionalnih zavisnosti. Iako je BCNF veoma jak uslov (do njega u praksi i normiramo bazu), ona nije dovoljna. Baze podataka se ne ponašaju kao matematičke funkcije. Zbog toga koristimo **višeznačne zavisnosti** da bismo mogli da još dekomponujemo bazu.

Definicja: U relaciji $R = \{X, Y, Z\}$, izmedju X i Z važi **višeznačna zavisnost**(skraćeno VZ) u oznaci $X \to\to Y$ akko za $\forall t_1, t_2$ torke R važi:

$$t_1[X] = t_2[X] \iff \exists t_3 : t_3[X] = t_1[X] = t_2[X] \wedge t_3[Y] = t_1[Y] \wedge t_3[Z] = t_2[Z]$$

VZ ima sledeća pravila, za svaki skup atributa X, Y, Z, W tako da je $W = Att(R)/XY$ važi:

- *Komplementarnost* - $X \to\to Y \implies X \to\to W$
- *Refleksivnost* - $Y \subseteq X \implies X \to\to Y$
- *Proširivost* - $X \to\to Y \implies ZX \to\to ZY$
- *Tranzitivnost* - $X \to\to Y \wedge Y \to\to Z \implies X \to\to Z$

Teorema potpunosti dekompozicije i višeznačnih zavisnosti : Neka je $Att(R) = \{X, Y, Z\}$ gde je $Z = Att(R)/XY$. Tada je: $X \to\to Y \iff R = R[XY] \times R[XZ]$

Definicija: U relaciji $R, Att(R) = \{X_1, \ldots, X_n\}$ važi **zavisnost spajanja**(ili ZS skraćeno) akko: $R = R[X_1] \times \ldots \times R[X_n]$, u oznaci $*\{X_1, \ldots, X_n\}$. Neophodno je za potpunu dekompoziciju.

Sledeće normalne forme se oslanjaju na VZ i ZS

4NF:

Relacija je u 4NF akko za svaku VZ $X \to\to Y$ važi da je X ključ ili natključ. Ako relacija ne zadovoljava 4NF, to znači da se može definisati kao rezultat spajanja dveju jednostavnijih relacija. Nezadovoljavanje 4NF može da dovede do drastičnog narušavanja performansi, zbog značajnog povećavanja broja redova u konkretnim tabelama.

5NF:

Relacija R je u 5NF akko je svaka netrivijalna ZS implicirana ključevima kandidata. ZS je implicirana ključevima akko svaka komponenta zavisnosti spajanja predstavlja natključ relacije. Poznata je i pod imenom Normalna forma projektivnog spajanja. Izmedju ove i 4NF postoje još tri normalne forme:

- *Normalna formu esencijalnih torki*(ETNF) - Je zadovoljava akko svaka torka ove relacije mora da bude esencijalna. Tj. torka nije uopšte reduntantna.

- *Normalna formu bez redundantnosti*(RFNF) - Relacija *R* zadovoljava normalnu formu bez redundantnosti akko ne može da postoji instanca relacije koja sadrži neku torku *t* i neki atribut *A* tako da je par (*t*,*A*) redundantan.
- *Normalna formu natključeva*(RFNF)- Relacija *R* zadovoljava normalnu formu natključeva akko je svaka komponenta svake nereducibilne zavisnosti spajanja natključ. Zavisnost spajanja je nereducibilna ako nepostoji podskup njenih komponenti koji određuje zavisnost spajanja.

Normalna forma domena ključa(DKNF):

Relacija zadovoljava DKNF akko u njoj ne postoje drugi uslovi i ograničenja osim uslova domena i uslova ključa. Uslovi domena propisuju dopuštene vrednosti atributa, a uslovi ključa predstavljaju jedine dopuštene FZ.

6NF:

Relacija je u 6NF akko ne zadovoljava nijednu ZS tj. akko ne postoji potpuna dekompozicija. To znači da je u 5NF i sadrži primarni ključ i najviše još jedan dodatni atribut koji je FZ od ključa. 6NF nije sasvim u skladu sa NFDK. Problem je u razbijanju na trivijalne relacije, jer se gubi mogućnost elementarne provere potpunosti entiteta.

Fizički model baza podatka

Fizički model baza podataka je najniži oblik modela. Opisuje konkretnu implementaciju i uzima u aspekte tehničke implementacije. Fizički model baze podataka čine:

- *Strukture podataka* - Na fizičkom nivou se govori o tabelama i kolonama
- *Interna organizacija* - Prostori za tabele, kontejneri, stranice, baferi podataka
- *Pomoćne komponente* - Indeksi

Dobro poznavanje fizičkog modela nam pomaže da otklanjamo sva moguća opterećenja baza podataka, da bismo poboljšali vremensku i memorjsku složenost aplikacije. Opterećenja baze procenjujemo preko:

- *Modela obrade podataka*
- *Nestrukturnih zahteva*
- *Matrica entiteta i odnosa*
- *Zahtevane performanse*
- *Prepoznate SUBP implementacije*
- *Prepoznavanje ograničenja prostora*

Šta su nama neke od modela obrade podataka koje nam služe radi procene opterećenja?

- *Okolnosti dodavanja novih redova* – Koliko redova se dodaju u proseku ili koliko redova se dodaju pri najvećem opterećenju. Da li su primarni ključevi slični i da li zavise od vremena?
- *Okolnosti ažuiranja redova* – Koliko redova se ažuira u proseku ili koliko redova se ažuira pri najvećem opterećenju. Kolika je verovatnoća da se redovi sa sličnim PK istovremeno koriste.
- *Okolnosti brisanja redova* – Koliko redova se brišu u proseku ili koliko redova se brišu pri najvećem opterećenju. Da li se brišu pojedinačno ili u grupama?
- *Okolnosti čitanja redova* - Kolika je učestalost čitanja redova. Koliko njih se čitaju sa jednim upitom, a kako se kolone koriste za njihove čitanje? Koje druge tabele se često koriste zajedno sa
  posmatranom?

Nestrukturni zahtevi su isto veoma bitni pri pravljenju fizičkog modela baze podataka:

- *Trajanje podataka* - Koliko dugo se podaci zadržavaju u tabeli pri brisanju ili arhiviranju?
- *Obim podataka* - Koliko će redova biti u tabeli pri puštanju u rad i kako će se broj redova menjati tokom vremena?
- *Raspoloživost podataka* - Da li su podaci potrebni stalno ili privremeno, koliko često i dugo podaci mogu ili smeju da budu nedostupni korisnicima?

- *Ažurnost podataka* - Koliko ažurni moraju da budu podaci koji se koriste?
- *Bezbednosni zahtevi*

Ako izgleda da naša baza ima neka opterećenja, moramo da ih otklonimo. Imamo neka od rešenja:

- *Optimizacija na nivou interne organizacije podataka* - Ostvaruje se kroz upravljanje internom organizacijom podataka, pomoćnim komponentama i resursima. Fizički model se ne razlikuje u odnosu na logički.
- *Optimizacija na nivou upita* - Vrši se pisanje upita na način koji omogućava njihovo efikasnije izvršavanje. Ni ovde se fizički model ne razlikuje u odnosu na logički.
- *Optimizacija na nivou strukture podataka* - Fizička struktura podataka se menja u odnosu na logički model.

U nastavku ćemo da uvedemo neke fizičke organizacije podataka koje nam služe za optimizaciju baze. Ove strukutre se koriste u SUBP DB2(pored *particionisanih tabela* i *kompresije podataka*).

Prostor za tabele

Prostor za tabele je osnovni skladišni prostor. Svaka baza podataka mora da ima jedan ili više prostora za tabele. Prostor za tabele pripada jednoj bazi podataka, a jedan prostor za tabele može da sadrži više tabela. U particionisanim bazama podataka jedna tabela može da bude i u više prostora za tabele.

Prostor za tabele predstavlja *logički nivo fizičke organizacije*. Određuju se osobine i način upotrebe prostora za tabele, ne određuje se detaljno gde se podaci nalaze. Ovu organizaciju skladištenja radimo preko skup kontejnera, particija diska, disk, prealociran fajl ali i skladišna grupa.

Stranice baze podataka

Stranica baze podataka je osnovni element fizičkog zapisa tabele. Sve operacije se fizički odvijaju nad stranicama. Uvek se čita sa diska ili zapisuje na disk cela stranica. Svaka tabela i svaki indeks se sastoji od stranica, tj. svaka tabela se zapisuje u skupu stranica jednog prostora za tabele.

Sve stranice jednog prostora za tabele su iste veličine, a one mogu biti:

- *Male* - Veća je iskorišćenost, manja su nešptrebna čitanja/pisanja i uobičajno se koriste za transakcione baze podataka.
- *Velike* - Sadrži više redova, manje je dubina indeksa i koristimo je za skladištenje podataka baze.

Bafer za stranice

Bafer za stranice je memorijski prostor predviđen za čuvanje kopija jednog broja stranica radi omogućavanja bržeg pristupa podacima. Određuje se na nivou prostora za tabele. Što je bafer za stranice veći, to je broj pristupa disku manji.

Materijalizovani pogled

Materijalizovani pogledi su tabele čija struktura i sadržaj su definisani datim upitom. Osnovna namesna im je ubrzavanje složenih upita nad podacima koji se relativno retko menjaju i koji se relativno lako ažuriraju u slučaju promene podataka.

Transakcione baze podataka

To su baze podataka koje podržavaju transakcioni rad. Cilj transakcija je medjusobno izolovanje. Rad jedne ne bsme da utiče na rad druge na bilo koji način. Postoje dve strategije izolovanja transakcije:

- *Optimističko* - Pretpostavimo da neće biti mešanja transakcija. Pamtimo sve što je transakcija uradila i kada budemo čitali podatke proveravamo njemnu validnost. Uobičajno je za dugotrajne transakcije.
- *Pesimističko* - Pretpostavimo da više transakcija izvršava u isto vreme nad istim podacima. Održavamo ovu strategiju *katancima*.

Katanci su mehanizmi izolovanje transakcija. Svaki katanac ima objekat koji zaključava, svoj tip(ekspluzivan ili deljivi) i trajanje. Veliki broj katanaca može da dovede do usporenja provera, pa mora da se na neki način ograniči postavljanje. Ako predjemo granicu načinili smo *eskalaciju katanaca*. Tada više manjih katanaca menjamo jednim ili stavljamo jedan katanac na tabelu.

Indeksi

Indeksi su pomoćne strukture podataka koje omogućavaju brz pristup podacima, preko ključa. Svaka tabela može da ima više indeksa sa različitim ključevima. Ona je fizička struktura podataka koja se zapisuje u izabranom prostoru za tabele. Imamo:

1. **Jedinstveni indeksi** - Ne dozvoljavaju ponavljanje više redova sa istim ključem. Koristi se kao sredstvo za integrisanje podataka.
2. **Grupisajući indeksi** - Obezbeđuju da su redovi u tabeli poređani u odgovarajućem poretku. Najviše jedan ovakav indeks može da postoji na tabeli. Kandidati za grupišuće indekse su kolone relacija. Prednost im je jer ubrzavaju izdvajanje “podsekvence” redova u datom poretku, ali zato usporavaju održavanje, ne menja se samo indeks nego i fizički zapis redova.
3. **Particionisani indeksi** - Particionisana tabela može da ima: neparticionisani indeks (ceo indeks u jednoj particiji) ili particionisani indeks (indeks se nalazi u više prostora za tabele).
4. **Dvosmerni indeks** - Omogućavaju pretraživanje u oba smera. Za neke indekse to ne predstavlja veliku promenu u implementaciji.
5. **B-stabla indeksi** - Podaci se čuvaju u balansiranom drvetu. Svaki red tabele je referisan iz lista jednake dubine. Čvorovi i listovi sadrže po više podataka. Jedinica organizacije indeksa je stranica. Veličina čvorova odgovara stranici prostora za tabele. Prednosti B stabla je da su jednostavni i efikasni algoritmi za održavanje. Slabosti su da nije posebno efikasan ako je mnogo redova a malo različitih vrednosti ključa.

6. **Bit mapirani indeksi** - Indeks je organizovan kao niz vrednosti ključa. Uz svaku vrednost ključa sledi niz bitova, za svaki red tabele po jedan. Vrednost bita je jedan akko odgovarajući red ima baš tu vrednost ključa. Dobre strane su da ako imamo relativno malo različitih vrednosti ključa, onda je ovakva struktura efikasnija od B-stabla. Mana je da u slučaju mnogo različitih vrednosti ključeva, indeks postaje veoma veliki i slabo efikasan i onda je održavanje indeksa često značajno skuplje nego u slučaju B-stabla.
7. **Heš indeksi** - Indeks se implementira kao heš tabela sa datim ključevima. Računaju se heš vrednosti na osnovu ključnih atributa, a onda se pomoću dobijene vrednosti neposredno pristupa podacima. Prednosti su da je alternativa klasičnim indeksima B-stabla je efikasnija za pristupanje pojedinačnim redovima sa tačno zadatim ključem. Loša strana je da nisu dobri za sekvencijalno pristupanje većem broju redova ili ako operator poređenja nije jednakost.
8. **Indeksi sa dodatnim kolonama** - Savremeni RSUBP omogućavaju da se indeksu osim kolona ključa dodaju još i neke kolone, koje ne čine uslov uređenja. Ako upit zahteva samo kolone ključa i te dodatne kolone, onda se čitanje završava na indeksu.

Nije uvek pogodno koristiti indekse. U slučaju malih tabela, može da bude brže da se uvek pretraže svi redovi nego da se koriste indeksi. Indeksi omogućavaju brži pristup konkretnim redovima tabele, naročito onima koje su velike. Ako čitanje zahteva samo kolone sadržane u indeksu, onda tabeli ne mora ni da se pristupa. Moramo da pazimo na indekse, one se održavaju i povećavaju granularnost katanca. Zbog toga idealan broj i vrsta indeksa zavise od vrste, namene, strukture tabele i baze podataka. Za transakcione tabele, konvencija je tri do pet indeksa, dok za analitičke tabele nema ograničenja.

Distribuirane baze podataka

Distribuirano izvršavanje je kada se dva posla izvršavaju u fizičkim razdvojenim adresnim prostorima, posredstvom računarske mreže.

CBP - Centralizovana baza podataka je BP koja u celosti počiva na jednom računaru.

CSUBP - Centralizovan sistem za upravljanje bazom podataka je softverski sistem koja omogućava rad nad CBP-om.

DBP - Distribuirana baza podataka je skup više *logičkih nezavisnih* BP koje su distribuirane na računarskoj mreži, koji je njihov jedini deljeni resurs.

DSUBP - Distribuiran sistem za upravljanjem bazom podatka je softverski sistem koja omogućava upravljanje DBP-om tako da je *distribuiranost transparentna za korisnika*.

Šta je predmet distribuiranja u konteksu BP? Imamo nekoliko:

- logika obrade - distribuira se medju jedinicama obrade i podrazumeva visok nivo konceptualne homogenosti
- funkcija - različite funkcije se izvršavaju na različitim čvorovima
- podaci - podaci se nalaze na različitim čvorovima
- upravljanje - poslovi upravljanje izvršavanjem različitih funkcija je distribuirano

Kriterijumi za posmatranje distribuiranosti

1. Stepen spregnutosti - mera koja opisuje koliko su čvrsto medjusobno povezani elementi sistema(numerički ili opisana rečima).
2. Struktura medjusobne povezanosti - razlikuje dva osnovna slučaja: *neposredno povezivanje čvorova(P2P - peer to peer)* i *upotreba deljivih kanala za komunikaciju medju svim sistemima*.
3. Medjuzavisnost komponenti - opisuje u kojoj meri komponenti zavise jedno od druge.
4. Sinhronizacija medju komponenti - koje može biti *sinhrono* i *asinhrono*.

Osnovni doprinosi DSUBP su:

1. ***Transparentno upravljanje distribuiranim i repliciranim podacima***:
   - Podrazumeva razdvajanje semantike na visokom nivou problema koji nastaju pri implementaciji na visokom nivou.
   - Aspekti koje ima transparentnost su:
     - **Nezavisnost podataka** - Koja može biti: *logička nezavisnost podataka*(otpornost aplikacije na logičke strukture podataka) ili *fizička nezavisnost podataka*(potpuno

skrivanje fizičke strukture podataka od aplikacije)
    - **Mrežna transparentnost** - U centralizovanim BP jedini resurs o kome se stara transparentnost jesu podaci. U distribuiranim sistemima, mrežna struktura je potrebna da budu sklonjena od očiju korisnika i tranparentno upravljena.
    - **Transparentnost replikacije** - korisnici ne bi trebali da budu svesni činjenice da se podaci repliciraju (**Replikacija** - čuvanje istih podataka na više lokacija).
    - **Transparentnost fragmentacije** - Korisnici ne bi trebali da budu svesni činjenice da su podaci fragmentisani (**Fragmentacija** - čuvanje različitih delova podataka na više lokacija)
  - Nosilac transparentosti su:
    - upitni jezik - koji se stalno prevodi u odnosu na distribuiranost i organizacije
    - operativni sistem - stara se o povezivanju modula
    - DSUBP - logički se stara o distribuiranju i strikturizaciji
2. ***Pouzdanost distribuiranih transakcija***:
    - Nosilac pouzdanosti rada SUBP-a je transakcija, koja može da donese u distribuiranom okruženju nove probleme.
3. ***Unapredjenje performansi***:
    - **Fragmentisani podaci se čuvaju bliže mestu upotrebe** - Ravnometno je rasporedjeno opterećenje na više servera na različitim lokacijama i manje se vremena troši na prenos podataka.
    - **Globalni upiti se paralelizuju** - Podaci su distribuirani pa se globalni upiti dele na manje, koje se odnose na lokalne podatke.
4. ***Lakše proširivanje sistema***
    - Distribuiranom SUBP-u je mnogo lakše povečati kapacitet od centralizovog jer su računari veoma skupi ako se povećava njihov kapacitet, čak eksponencijalno.

Najvažniji otežavajući faktori su:

- **Složenost** - Distribuirani sistemi su složeni jer moraju da rade iste probleme koja radi i CSUBP a još mnoge druge.
- **Cena** - Distribuirani sistemi zahtevaju dodatni hardver, a softverski je komplikovaniji pa samim tim i teži za pravljenje i održavanje.
- **Distribuirana kontrola** - Otežani su sinhronizacija i koordinacija komponenti.
- **Bezbednost** - Na CSUBP ovde imamo i aspekte bezbednosti kod računarskih mreža.

Teme koje se istražuju u oblasti DSUBP: *projektovanje DBP, distribuirano izvršavanje upitam, dist. upravljanje metapodacima, dist. kontrola konkurentnosti, dist. upravljanje mrtvim petljama, pouzdanost dist. SUBP, podrška OS-u* i *heterogene BP*.

Heterogene baze podataka je spajanje DBP-a sa više postojećih rešenja. Sa ovim narušavamo *homogenost softvera*(različite BP rade pod različitim SUBP) i *homogensot strukture*(strukture BP nije usaglašena). Povodom toga rešavamo probleme *distribuirano izvršavanje upita*, *upravljanje metapodacima i konkurentnošću*.

Replikacije

U distribuiranim sistemima je realnost da se neke od distribuiranih komponenti ne ponašaju funkcionalno uvek. Neispravnosti obuhvataju: bagove, ljudske greške koje upravljaju sistemima, preopterećenost, zagušenje,...

Postoje dva odnosa prema neispravnostima:

- izolovanje neispravnosti - Princip je obezbeđivanje da neispravnost jedne komponente ne utiču na funkcionalnost ostalih komponenti. Posledica ovoga je da funkcija koju je obezbeđivala jedna komponenta nije raspoloživa dok se problemi na njoj ne otklone.
- tolerancija neispravnosti - U slučaju neispravnosti neki od komponenti, druge komponente preuzimaju njene funkcije. Zbog toga imamo povećanu fleksibilnost i pouzdanost sistemu, ali isto tako i složenost i cenu.

Koje su aspekti pouzdanosti našeg sistema?

- Raspoloživost sistema - "Sposobnost sistema da primi zahtev". Relativno nisko trajanje perioda kada sistem "ne radi" ili retko dolazi do neispravnosti.
- Odgovornost sistema - "Sposobnost sistema da odgovori na zahtev". Ili retko dolaže do neispravnosti ili se u hodu lako ispravljaju da ne bude prekida operativnosti.

  "*Sistem može biti visoko odgovoran, a da nije visoko raspoloživ i sistem može biti visoko rasploživ a da nije visoko odgovoran*"

Sistem je u potpunosti odgovoran ako nikada ne trpi oštećena tokom obrade zahteva. Mera odgovornosti je verovatnoća da će sistem odgovoriti na zahteve koje je primio:

- $R(t) = 1 - F(t)$, ili ako prekazujemo preko gustine $R(t) = 1 - \int_0^t f(x)dx$

Uobičajna mera je "srednje vreme do neuspeha", *mean time to failure*(MTTF):

- $MTTF = R^{-1}(0.5)$

Pored nje koristi se i "srednje vreme izmedju neuspeha", *mean time between failures*(MTBF) i "srednje vreme oporavka", *mean time to repair*(MTTR).

U praksi se koristi granična raspoloživost: $\lim_{t\to\infty} A(t) = \frac{MTTF}{MTTF+MTTR}$

Pored toga se upotrebljava i metod eksperimentalnog merenja, u intervalu [0,t] registruju intervali u_i u kojima sistem nije raspoloživ.

$$A(t) = 1 - \frac{\sum_i u_i}{t}$$

Motivacijom mere odgovornosti uvodimo **replikacije**.

Replikacije je svako udvajanje neke komponente računarskog sistema koje donosi neki nivo redundantnosti. Pored pouzdanosti, ovo uvodimo i zarad performansi. U slučaju distribuiranih BP replikacija je često neophodno sredstvo za dostizanje potrebnog nivoa goronavedenih stvari.

Podizanje performansi se ostvaruje kroz *raspodelu opterećenja na replike*. Ovo je jako značajno u slučaju čitanja, ali može da ima negativan uticaj na performanse pisanja(očekuje se da sve replike izvrše ažuiranja u toku transakcije, te ih odloženo repliciramo nekada).

Podizanje pouzdanosti dobijamo implementacijom tolerancije neispravnosti, dok prepoznavanje neispravnosti vidimo kroz *dupliranja* komponenti RS-a.

Predmet replikacije mogu biti:

- **podaci** - BP, tabele, fragmenti tabele, globalni katalog BP, sistem datoteka...
- procesi
- objekti
- poruke

Ograničavanja replikacije predstavljaju:

- cena replikacije - dodatna cena prostora zbog redundantnog čuvanja podataka, cena pisanja kada se menjaju podaci, obim prenosa....
- ograničenje efikasnosti - ako je ograničena raspoloživost jedne kopije A, onda je ograničena i raspoloživost repliciranog sistema.

Postoje tri načina replikacije:

I. Čitaj jedan piši sve (ROWA):

Koncept ROWA protokola je da se samo primarna kopija čita, a sve kopije se menjaju. Tolerišu se otkaze lokacije kada je u pitanju čitanje, dok se ne tolerišu otkaze lokacije kada je u pitanju pisanje i komunikacije. Četiri varijante koje ćemo spomenuti su: osnovni ROWA protokol, ROWA sa menjanjem raspoloživih kopije, ROWA-A sa primarnom kopijom i ROWA sa tokenima pravih kopija.

Osnovni ROWA protokol prevodi svaku operaciju čitanja u jednu operaciju čitanja, na jednoj od kopija, dok obrnuto radi za pisanje. Prevodi svaku operaciju pisanja u *N* operacija pisanja, po jednu na svakoj kopiji. Kontroler konkurentnosti na svakoj lokaciji sinhronizuje pristup kopijama. U slučaju otkazivanja neke lokacije je čitanje i dalje moguće, a pisanje nije do popravke.

Protokol ROWA-A(*Read One Write All Available*) se razlikuje od osnovnog protokola jer se pisanje izvodi ne na svim, nego na "svim raspoloživim" kopijama. U slučaju otkazivanja neke lokacije čitanje i pisanje je i dalje moguće(što je bolje od osnovnog protokola). Neispravne lokacije mogu da postanu raspoložive tek nakon oporavka uz puno prepisivanja svih izmena nastalih tokom neoperativnog perioda. Problemi koji ima ROWA-A je da u svakoj neispravnoj lokaciji se šalje bar dva zahteva na koje se čeka dopušteno vreme. Dvofazna validacija nije efikasna, bolje je prepoznati oporavak lokacija u što kraćem intervalu.

ROWA sa primarnom kopijom proglašava jednu kopiju za **"primarnu"**, dok su ostale kopije "rezervne". Čitanje se izvodi samo sa primarne kopije, dok se pisanje izvodi na primarnoj i svim raspoloživim rezervnim kopijama. Ovime ne podižemo performanse čitanja, nego samo pouzdanost sistema. Ako primarna kopija postane neoperativna, izabrana rezervna kopija(u slučaju da je operativna) postaje nova primerna. Inače čekamo da se popravi ona. Ovaj algoritam je jako jednostavan i široko upotrebljiv.

ROWA sa tokenima "pravih" kopija daje podacima dva tokena, *eksluzivan* ili skup *deljivih* tokena. Operacija pisanja mora da dobije eksluzivan token, dok operacija čitanja mora da dobije bar deljivi token. Konceptualno je slično distribuiranom zaključavanju.

Pri pisanju, kada kopija mora da izvrši operaciju pisanja, ona locira i uzima eksluzivni token za konkretni podatak. Ako on ne postoji, ona locira i proglašava za neispravne sve deljive tokene za konkretni podatak i pravi novi ekslušivan umesto njih.

Pri čitanju kada kopija mora da izvrši operaciju čitanja, kopija locira deljivi token, kopira njegovu vrednosti i čuva novi deljivi token. Ako ne postoji deljivi token, ona locira eksluzivan token i konvertuje ga u deljivi i zatim postupa kao u prethodnom slučaju.

Token ima objedinjenu semantiku nalik na "lokalni katanac" na "distribuiranom podatku". Predstavlja mehanizam prepoznavanja konflikta izmedju više pisanja ili izmedju pisanja i čitanja podataka.

II. Koncenzus kvoruma (KK) (glasanje):

Kvorumi su podskpuvi operativnih lokacija, kao npr za pisanje, čitanje,... Na primer neki čvorovi mogu biti samo za pisanje, neki samo za čitanje, a neki su oba tipa, te pripadaju u preseku kvoruma. Pravilo preseka kvoruma obezbedjuje da će svako čitanje moći da se odvija najsvežije pisanim vrednostima. Za operaciju koja učestvuje u operaciju se kaže da je *glasala* za operaciju.

Razlika u odnosu na ROWA je da KK dopušta pisanje na podskupu operativnih lokacija. Čitanje se izvode sa podskupa lokacija koji mora da ima neprazan presek sa kvorumom pisanja.

Opšte karakteristike kvoruma su maskiranje neispravnosti, bez dodatih zahteva pri oprovaku neispravnih lokacija. Oni mogu biti:

- statički - odredjeni glasanjem pre podizanja sistema
- dinamički - ako lokacije mogu da se rekonfigurišu.

Prikažimo neke vrste kvoruma:

i) KK sa uniformnom većinom

Operacija čitanja ili pisanja uspeva akko većina lokacija odobri izvršavanje. Ne moraju sve lokacije koje su glasale da izvrše operaciju na svojim kopijama podataka. Čitanje je dovoljno da se izvrši na jednoj kopiji, a pisanje mora da se izvrši na većini kopija.Cena je relativno visoka i pri čitanju i pri pisanju. Bar polovina lokacija mora učestvovati u svakoj operaciji kroz glasanje.

ii) KK sa težinskom većinom

Predstavlja uopštenje algoritma KK sa uniformnom većinom. Svaka kopija d_s podatka d dobija nenegativnu težinu tako da suma svih težina na jednom
podatku bude u. Sam podatak d dobija prag čitanja r i prag pisanja w, tako da važi: $r+w>u$ i $w>u/2$. Kvorum čitanja (ili pisanja) podataka je tamo gde je svaki skup kopija čija je težina bar r (ili w). Zbog većinskog glasanja, u svakom kvorumu čitanja sigurno postoji tekuća kopija podatka. Načelno se ne zahteva složen algoritam oporavka. Algoritam je fleksibilan.

iii) KK za direktorijume i apstraktne tipove podataka:

Direktorijum je preslikavanje prostora ključeva u prostor vrednosti. Primenjuje se i na nerelacione baze zasnovane na katalozima. Menja se granularnost operacija tako da se ne izvršavaju na „velikim“ direktorijumima nego na njihovim manjim delovima. Apstraktni tipovi podataka je sličan problem kao za direktorijume.

iv) Hibridni protokol ROWA/KK:

Osnovna slabost metoda KK je neopravdano visoka cena u slučaju retkih otkazivanja komunikacija. Hibridni pristup redukuje cenu kroz primenu metoda ROWA tokom pouzdanih operativnih perioda . Dok se primenjuje metoda KK u uslovima postojanja otkaza lokacija ili komunikacija.Transakcije mogu da započnu rad u normalnom režimu i da tokom rada pređu u režim otkaza. Ako transakcija uputi zahtev za pisanje i dođe do izostajanja pisanja na nekoj kopiji, samo ista ta transakcija to lako uočava i menja režim. Da bi druge transakcije to doznale potrebni su dodatni mehanizmi.

III. Koncenzus kvoruma u uređenim mrežama:

Savremeni algoritmi replikacije pored tolerisanja neispravnosti stavljaju u prvi plan i optimizaciju troškova. Pokušava se postizanje smanjivanja broja lokacija koje odlučuju u glasanju uvođenjem logičke strukture u skup lokacija. Ta "statička" logička struktura je **koncenzus kvoruma sa uredjenim mrežama**. Neki poznatiji altoritmi su:

- Algoritam $\sqrt{n}$
- Protokol matrice (GRID)
- Asimptotska visoka raspoloživost
- Drvo kvoruma
- KK sa hijerarhijskom težinskom većinom
- KK sa višedimenzionalnom težinskom većinom

Algoritam $\sqrt{n}$ se zove tako jer se u osnovi svodi na to da veličina kvoruma može biti reda $\sqrt{n}$. Uvodi se pretpostavka da neće biti otkazivanja komunikacije. Svakoj lokaciji se dodeljuje jedan kvorum koji ima neprazan presek sa svim kvorumima koji su dodeljeni drugim lokacijama. Transakcija mora da obezbedi konsenzus svih lokacija u kvorumu dodeljenom njegovoj matičnoj lokaciji.

GRID algoritam počiva na skupove lokacija koje se uređuju u obliku matrice sa N kolona i M vrsta, tako da je $M * N \leq n$. Jedan kvorum čitanja obuhvata lokacije sa po tačno jednom lokacijom iz svake kolone. Kvorum pisanja se sastoji od svih lokacija jednog kvoruma čitanja i jedne kolone. Cilj je ravnomerno raspoređivanje opterećenja.

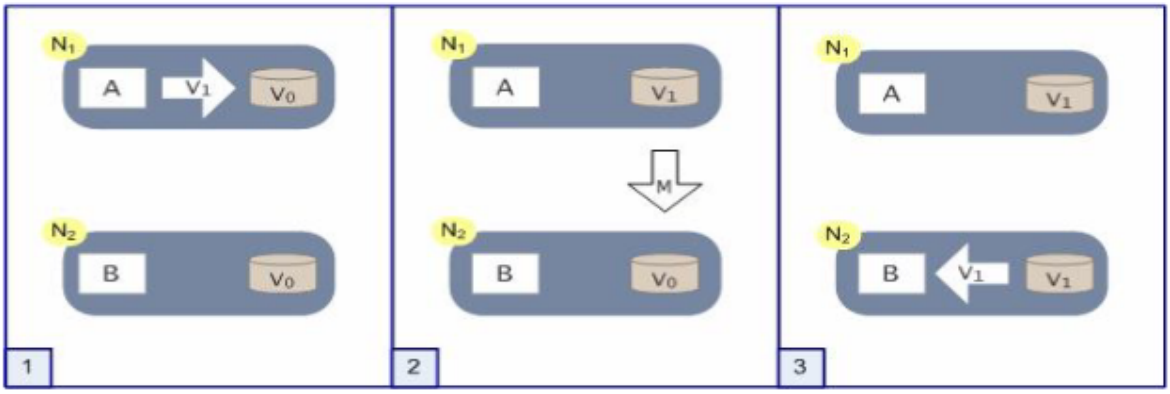
Teorema CAP

Pri upravljanju distribuiranih sistema kao najvažniji ciljevi se ističu

- **C(Konzistentnost)**:
  - "Konzistentan sistem funkcioniše kao celina ili ne funkcioniše uopšte"
  - "Sva čitanja osim na švorovima, moraju da daju isti rezultat"
  - **"Rezulzaz operacije čitanja nikada ne zavisi od čvora na kome se izvršava"**
- **A(Raspoloživost)**:
  - **"Sistem je uvek raspoloživ"** tj. odzivnost sistema u garantovanim granicama
  - Marfijev zakon: "Sistem obično nije raspoloživ kada je najpotrebniji"
- **P(Prihvatanje razdvojenosti)**:
  - **Razdvojenost** je stanje komunikacije mreže u kome su delovi sistema podeljeni na particije izmedju kojih ne postoji komunikacija. Sistem i u ovim okolnostima ima očekivanje da radi ispravno.
  - **"Nijedan skup problema, osim potpunog otkazivanja, ne sme da proizvede neispravan odziv sistema"**

Teorema CAP: Nije moguće definisati sistem koje zadovoljava sve CAP uslove.

Dokazaćemo ovo samo idejno. Neka imamo dva čvora koja žele da urade ažuiranja podatka V pomoću transakcija i poruka izmedju sebe.



Pretpostavimo da poruka M nije stigla na odredište, zbog razdvojenosti delova sistema.

U osnovi imamo tri moguća ponašanja sistema:

- transakcija A se poništava - što znači da sistem ne podržava razdvojenost svojih delova
- transakcija A se uspešno nastavlja - što znači da sistem nije konzistentan
- transakcija A čeka na uspešno slanje poruke M - to znači da sistem nije raspoloživ.

Što je skica dokaza da u svakom od slučaja nije ispunjen neki od uslova. □

Zbog prethodnog problema neophodno je napraviti kompromis:

- odbacivanje tolerancije razdvojenosti
- odbacivanje raspoloživosti
- odbacivanje konzistentnosti
- ublaživanje uslova
- zasnivanje sistema na drugačijem skupu uslova
- projektovanje zaobilaznih puteva

Odbacivanje tolerancije razdvojenosti:

- "pristajemo da sistem ne radi u slučaju razdvojenosti"
- jedan način prevazilaženja je da sve bude na jednoj mašini
- alternativa je da se razdvojenost svede na najmanju moguću meru pomoću višestrukog umrežavanja
- pošto su obe alternative skupe, ovaj vid kompromisa se retko koristi

Odbacivanje raspoloživosti:

- u slučaju razdvojenosti ne garantuje se vreme odziva
- zbog toga treba da se uspostavljaju što niže sprege medju čvorovima

- sistem koji nije raspoloživ je praktično neupotrebljiv, i zbog toga se ovaj pristup koristi prilično retko

Odbacivanje konzistentnosti:

- "dopuštamo da isti upit daje različite rezultate na različitim čvorovima", pod uslovima da je cena nekonzistentnosti prihvatljiva
- iako je konzistentnost jedna od osnovnih uslova za rad BP, nije svuda neophodna. Primera radi, veb pregledači, prodavnice...

Izmenjen skup uslova - BASE:

- Šta je BASE:
  - *Basically Available* - Pristup koji ne garantuje raspoloživost odgovora, nego samo sistema. Ako ne može da dobije odgovor, bar će dobiti obaveštenje od tome.
  - *Soft state* - Stanje sistema može da se menja čak i kada nije u toku nijedna transakcija, na primer radi ostvarivanja konzistentnosti.
  - *Eventually consistant* - Žrtvuju se garantovana stalna konzistentnost i izolovanost transakcija zarad raspoloživosti. Sistem će u **nekom trenutku** postati konzistentan, ali će pre tog trenutka raditi i davati odgovore.
- Projektovanje sa BASE uslovima se svodi na dve nove aktivnosti:
  - *funkcionalna dekompozicija podataka u fragmente* - Malo preciznije odredjujemo uslove za definisanje fragmenata(unutar kojih važe ACID uslovi, medju kojima veže BASE uslovi)
  - *implementacija transakcija prema BASE uslovima* - operacije se ne menjaju, ali se menja vreme njihovih izvršavanja.Time se i menja način implementacije, na sinnhroni i asinhroni deo.
  - *Određivanje uslova replikacije* - Pretpostavlja se da medju replikama važe BASE uslovi. Sinhronizicija replika se odvija asinhrono, van transakcija.
- *Konflikti* su osnovni vid problema kod vidova implementacija sa BASE uslovima. Postoje dva oblika konflikta. Osnovni oblik, kada dve replike istog podatka se nezavisno menjaju, koja je verzija "ispravna" i kako izvršiti usklađivanje. Složeni oblik je kada se koriste reduntantni podaci pored replika. Konflikte rešavamo na više načina:
  - zabrana menjanja podataka u slučaju particionisanja - ili umanjena rasploživost
  - definisanje hijerarhije među redundantnim kpijama konkretnih vrsta podatka - ne umanjuju upotrebljivost ali komplikuju implementaciju
  - višestruke verzije podataka - alternativa zaključavanju. Konflikti mogu automatski da se prepoznaju, ali ne i otklone.

Nerelacione baze podataka

U savremenom razvoju softvera termin **neralacione baze podataka** se odnosi na sistema za upravljanje kolekcijama podataka koje:

- nemaju strogu statičku strukturu podataka
- nemaju iscrpnu proveru uslova integriteta
- ne koristi upitni jezik SQL

Gorenavedenosti mogu da budu i slabosti i prednosti neralacionih baza podataka u odnosu na relacione.

Motivacija kod korišćenje ovakvih BP je slabost RBP-a koje se manifestuju u:

- visokoj ceni čitanja - neredundantnsoti i stroge strukute podataka
- otežano distribuiranje - zbog stroge konzistentnosti podataka
- skupa promena strukture
- distribuirani RBP ima nedovoljnu poboljšanje performani
- normalizobana shema - stroga operacija spajanja, pažljivo fragmentisanje i repliciranje, otežavanje menjanja

Neke osnovne vrste neralacionih baza podataka su:

- parovi ključevi vrednost
- skladišta širokih kolona
- skladišta dokumenata
- grafovske baze podataka
- objektne baze podataka
- tabelarne baze podataka
- XML baze podataka...

Tipični problemi RBP-a i alternativna rešenja:

- složene strukture podataka u specifičnom domenu - objektne baze podataka
- visok nivo distribuiranja - različite nerelacione baze podataka
- slobodna strukutra podataka - -||-
- neophodne izuzetne visoke performanse - memorijska baza podataka
- ogromne količinje podataka niske složenosti - različite vrste matičnih baza podataka

Proćićemo kroz najčešće nerelacione baze podataka:

1. Baze parova ključeva i vrednosti

Svaka kolekcija je heš tabela. Podacima se pristupa isključivo na osnovu poznatog ključa. Zbog toga podržavaju velike skupove podataka i veoma su brze. Podržavaju automatsko repliciranje i horizontalno particionisanje kolekcija.

Slabosti baze parova ključ vrednost je podrazumevana visoka redundantnost. Složene strukture podataka se implementiraju velikim brojem kolekcija. Ako su podaci gusto povezani, efikasnost za očuvanje integriteta. U osnovi nemaju mehanizme za očuvanje integriteta.

Primeri su *Redis, Memcached, Oracle NoSQL...*

2. Baze sa proširivim slogovima

U osnovi liči na bazu parova ključeva i vrednosti. Vrednost predstavlja kolekciju velikog broja parova imena i vrednosti. Podržavaju velike skupove podataka i veoma su brze. Obično podržavaju bar poluatomske repliciranje i horizontalno particionisanje kolekcija.

Mane povlači od baze parova ključeva ivrednosti.

Primeri su *Cassandra, BigTable, Druid...*

3. Baze dokumenata

U osnovi liči na bazu parova ključeva i vrednosti. Svaki dokument predstavlja vrednost kome se dodeljuje ključ. Dokumenti se zapisuju u struktuiranim ili nestruktuiranim oblicima, i ti se čuvaju metapodaci. Sa ovim dobijamo veoma jednostavan i efikasan rad. Obično podržavaju bar poluatomske repliciranje i horizontalno particionisanje kolekcija.

Teškoća korišćenja je u relativnom ograničenom domenu primena. Mnoge implementacije ne omogućavaju menjanje podataka ili velike učestalosti istog.

Primeri su *MongoDB, Google Cloud, Datastore...*

4. Grafovske baze podazaka

Bazu čine čvorovi i veze izmedju njih. Nema čvrste šeme, a podaci imaju slobodnu strukturu. Akcenat je na odnosima izmedju podataka, a ne na strukturi podataka. Imaju efikasne operacije sa grafovima, i podržavaju transakcije i ACID režim uslova.

Kao i prethodna, ima ograničen domen primena i nisu pogodne van njih.

Primeri su *Allegro, Virtuoso, OrientDB...*

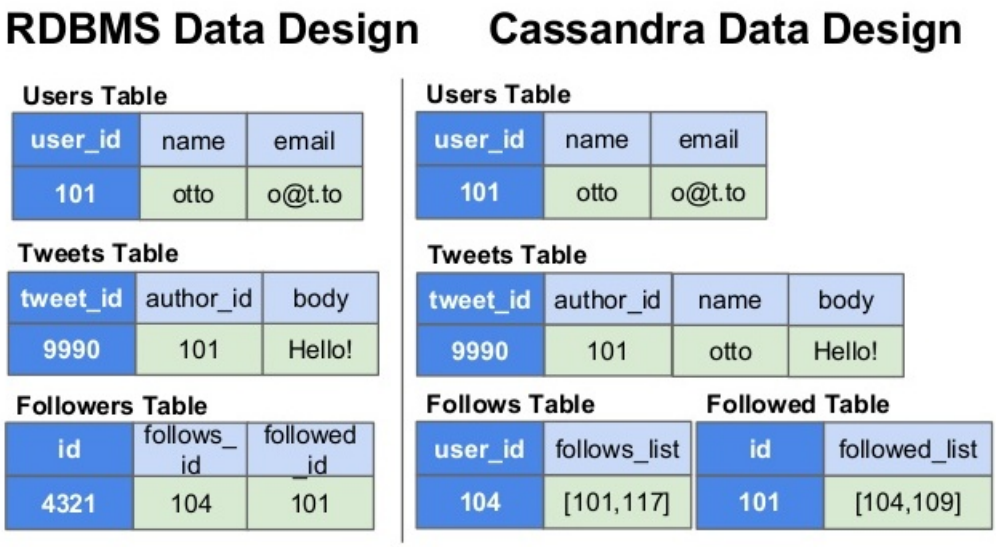
Od primera ćemo za kraj lekcije da se osvrnemo na **Apache Cassandra** kao alat za nerelacione baze podataka. Primarno namenjena za distribuirane baze podataka, tj. dimaničke sheme. Počiva na proširenom modelu kataloga izgrađen uz pomoć upitnog jezika CQL. Osnovni pojmovi kojim se bavimo su:

- *kolona* - trojka *name, value* i *timestamp*(vreme izmene)
- *familija kolona* - struktura koja može da sadrži veći broj redova, konceptualno predstavlja katalog redova
- *superkolona* - sadrži jednu ili više kolona
- *familija superkolona* - struktura koja može da sadrži veći broj superkolona
- *ključ* - kao u RBP-u
- *prostor ključeva* - struktura koja sadrži više familija kolona ili superkolona. Odgovara pojmu baze podataka ili sheme kod RBP-a.

Cassandra ima dva nivoa ugnježdanosti:

1. nivo(obavezan) - čini ga par ime kolone i kolona. Jedan red, iliti slog, se sastoji od proizvoljnog broja parova.
2. nivo(opcioni) - osim da imamo kolone, možemo da imamo kolekciju parova - superkolone.

Tipove podataka koje podržava su: tekstualni, numerički, logički, kolekcije, univerzalni ID...



Sistemi za podršku odlučivanja

Sistemi za podršku odlučivanja su sistemi koji služe za pružanje informacija od strateškog značaja na osnovu analiza prikupljenih informacija. Osnovne teme u okviru ovoga su: prikupljanje i održavanje velikih kolekcija podataka i analiziranje istih. Pravljenje složenih izveštaja je komplikovana obrada koja zahteva da podaci budu na jednom mestu. Osnone pojmove koje ćemo da koristimo pri podršci odlučivanja su:

- *Skladišta podataka* - baze podataka koje su projektovane specifično za podršku odlučivanju i ne koriste se za transakcionu upotrebu.
- *Onlajn analitička obrada* - postavljanje složenih analitičkih upita.
- *Istraživanje podataka* - primena različitih metoda radi izvodjenje složenijih zaključaka i obrazaca iz postojećih podataka

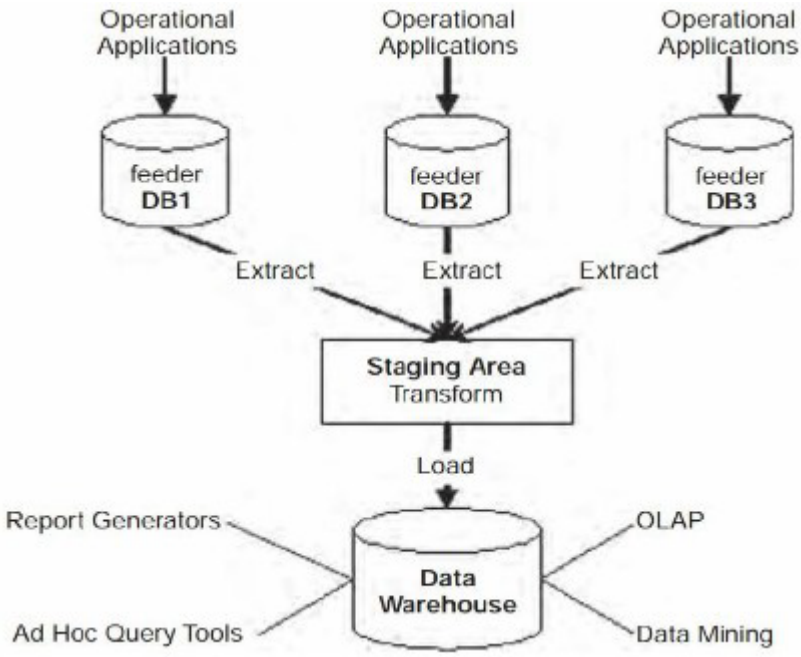
Skladišta podataka su BP koje su projektovane specifično za podršku odlučivanju i ne koriste se za transakcionu upotrebu. Osnovna namena je analitička obrada podataka, nad podacima koje nisu "sasvim" ažurni. Prilagodjene su čitanju i analizi podataka, ali ne i za redovno ažuiranje.

ОЛТП базе података	Складишта података
транскационо оријентисане	оријентисане према теми
хиљаде корисника	неколицина корисника
релативно мале (до неколико *GB*)	релативно велике (од неколико GB до неколико PB)
фокус је на текућим подацима	фокус је на историјским подацима
нормализовани подаци (нема редундантности, обично више табела са по мало колона)	денормализовани подаци (има редундантности, обично мало табела са по више колона)
непрекидно појединачно ажурирање	периодично пакетно ажурирање
једноставни и сложени упити, најчешће мањег обима	веома сложени упити, често веома великог обима

Zahteve koje ćemo da izdvojimo pri projekovanju skladišta podataka su:

1. *Podaci se organizuju tematski* - teme odgovaraju grupama i vrstama podataka. Teme su obično međusobno nezavisne u odnosu na transakcije. Svaka tematska oblast se posebno projektuje i implementira u skladištu.

2. *Mogućnost integrisanja podataka* - iako se svaka tematska celina projektuje i implementira nezavisno, korisno je da postoji mogućnost integrisanja. Obično je radimo radi izvodjenja složenijih uporednih analiza.
3. *Podaci su nepromenljivi*(tokom obrade) - podaci se ažuiraju periodično i masovno.
4. *Ažuiranje se odvaja periodično, u masovnim paketima* - obično se svodi na dodavanje novih pripremljenih podataka. Podaci se prečišćavaju da bi se izbacili nepouzdani i nerelevantni podaci. Mehanizmi za osvežavanje prate izmene i po potrebi iniciraju novi ciklus ažuiranja.
5. *Podaci su često potrebni na različitim nivoima granularnosti* - analize su efikasnije ako su podaci pripremljeni za različite nivoe. Viši nivoi se izvode iz nižih.
6. *Fleksibilbist u odnosu na zahteve i ciljeve* - radi omogućavanja što jednostavnijeg proširivanja skupa posmatranih podataka.
7. *Mogućnost menjanje istorije/budućnosti* - preko horizontalnog particionisanja.
8. *Specifične korisničke aplikacije sa odgovarajućim korisničkim interfejsom*



Arhitektura skladišta podataka

Prodjimo sada kroz cikluse koje imaju skladišta podataka:

1. *Prikupljanje i analiza zahteva* - analiza krajnjih ciljeva i zahteva, pravljenje specifikacij istih, definisanje arhitekture i kapaciteta...
2. *Logičko projektovanje* - proektovanje tabela i pogleda na SP. Kao osnova se koristi "*dimenziono modeliranje*" gde je osnovni model zvezdasta shema.
3. *Fizičko modeliranje* - projektovanje indeks, materijalizovanih pogleda, particija...
4. *Distribuiranje podataka* - obično se teži da SP bude centralizovana, ako je neophodno onda se distribuiraju na više računara u jednom centru, zbog performansi.
5. *Implementacija, praćenje i menjanje* - pravljenje tabela, pogleda, skriptova, metapodataka...

Zvezdasta shema je ona shema kojoj je u centru pažnje tabela činjenica koja se primenjuje u logičkom projektovanju skladišta podataka. Sve dodatne informacije se zapisuju u dimenzionim tabelama. Na sve podatke u dimenzionim tabelama se referišu neposredno iz tabela činjenica i one opisuju pojedinosti dimenzija koje se navode tamo. Nisu normalizovane. Postoje isto tako i **shema pahuljica** koja je slična kao i zvezdasta shema samo što su dimenzione tabele obično normalizovane.

Zagledajmo detaljnije u specifičnosti fizičkog modeliranja:

- **Projektovanje indeksa**:
  - *Indeksi spajanja* - namenjeni su za efikasno povezivanje tabela činjenica i dimenzionih tabela. Prave se na dimenzionim tabelama kao heš ili b-stabla.
  - *Indeksi pretraživanja* - za efikasno pretraživanja tabela činjenica pomoću bit-mapiranih indeksa
  - *I razni drugi indeksi* - za generalno ubrzavanje upita
- **Projektovanje materijalizovanih pogleda**: Koji logički odgovaraju pogledima, a fizički tabelama. Nakon prvog izračunavanja rezultati se čuvaju sve dok se podaci ne promene. Jedne od vrsta materijalizovanog pogleda je *automatske zbirne tabele*. Njihova namena je da predstavljaju alternativne „tabele“ činjenica za grublje nivoe granularnosti. Koriste se za ubrzavanje izračunavanja često upotrebljavanih statistika
- **Projektovanje particija**:
  - horizontalno - po grupama redova
  - vertikalno - po grupama kolona

Onlajn analitička obrada(OLAP) je u širem smislu svaka analitička obrada podataka. Obično se počinje od opštijeg upita i šitanja pa se nastavlja postepenim fokusiranjem na odredjene grupe podataka i detaljisanjem upita. U užem smislu, to je analitička obrada koja se izvodi nad "živim" podacima. U elemente OLAP-a spadaju i odgovarajući elementi upitnih jezika i tehnike njihove upotrebe. Analiza zahteva se odvija istovremeno sa transakcionom obradom.

Istraživanje podataka je izvođenje kvalitativno novih informacija iz velikih skupova podataka. Teorija i primena tehnika - razvoj i primena algoritama skoro uvek nad skladištima podataka. Obrada je složena i traje dugo tako da podaci ne bi trebalo da se menjaju pritom.

Metode optimizacije baze podataka

Našu bazu možemo da optimizujemo na tri načina:

- **Optimizacija na nivou interne organizacije podataia** - Ne uvodimo razliku izmedju fizičkog i logičkog modela. Ostvaruje se kroz upravljanje internom organizacijom podataka, pomoćnim komponentama i resursima.
- **Optimizacija na nivou upita** - Ne uvodi se razlika izmedju fizičkog i logičkog modela. Vrši se pisanje upita na način koji omogućava njihovo efikasnije izvršavanje
- **Optimizacija na nivou strukture podataka** - Uvodi se razlika izmedju fizičkog i logičkog modela. Organizacija tabela se menjaju odnosu na logički model. Ako imamo uzorak radnog opterećenja i odgovarajuće planove izvršavanja, onda možemo da sprovedemo optimizaciju strukture baze podataka. Prepoznajemo dva cilja pri optimizacije strukute:
  - prepoznavanje indeksa koje moramo da napravimo
  - prepoznavanje potrebnih promena u fizičkoj shemi

Optimizacija na nivou upita

Da bismo imali efikasan upit moramo da imamo i **plan izvršavanja upita**. On obuhvata:

- redosled korak a pri izvršavanju
- operacije koje se izvršavaju u pojedinim koracima
- strukture podataka koje se upotrebljavaju
- način svakog pojedinačnog pristupa podacima
- procenjuju cenu svakog od koraka i celog posla

Većina savremenih SUBP-a omogućava korisniku da sagleda plan izvrpavanja pre nego što pokrene izračunavanje upita. Prema trenutku pravljenja izvršavanja upita, način izvršavanja može biti:

- *Statički* - Plan izvšavanja se pravi u napred, pri prevodjenju programa(koji su isto statički). Pri izvršavanju upita nekog programa evidira se u BP, pri tome korisnik mora da ima autorizaciju za izvršavanje. Optimizovani su upiti prema stanju BP u trenutku ugradnja paketa.
- *Dinamički* - Plan izvršavanja se pravi pri izvršavanju upita. Korisnik koji izvršava program mora da ima autorizaciju da sme da izvršava dinamčke upite i naedbi. Optimizovani su na

najbolji način.

Da bismo imali dobru optimizaciju upita moramo da imamo i cenu pojedinačnih koraka u planu izvršavanja upita. Uzimamo u obzir sledeće informacije: cene uredjaja, broj redova u tabeli, popunjenost stranica, selektivnost upita,...Većina tih informacija je tačna **ako su ažurni statistički podaci** o sadržaju baze podataka. Neke operacije koje se često pojavljuju su:

- *TBSCAN* - čitanje redova neposredno it tabele
- *IXSCAN* - pretraživanje indeksa i čitanje pokazivača na redove
- *FETCH* - čitanje sadržaja kolona iz datog reda
- razni *JOIN*-ovi - može biti heš, ugnježđeno ili *merge*.
- *SORT* - uredjivanje redova

Optimizacija upita predstavlja optimizaciju svih vidova upotrebe baze podataka, što uključuje i čitanje i menjanje sadržaja baze podataka. Faktori na optimizaciju upita su raspoloživi podaci o bazi podataka: strukutra tabela i ključeva, strukutra uita, postojaći indeksi, ...

Optimizacija upita može biti *manuelno* ili *automatsko* odabiranje najpovoljnijeg plana izvršavanja upita radi postizanja boljih performansi. Uloga manuelne optimizacije upita je važnija u fazi pravljenja fizičkog modela nego u eksploataciji, jer se ne optimizuje upit, nego se analiziraju planovi izvršavanja radi sagledavanja i prevazilaženja eventualnih slabosti predloženog modela.

Uzrok radnog opterećenja je skup tipičnih upita za koje se zna da će činiti najčešći oblik pristupanja bazi podataka. Dobar uzorak se sastoji od skupa upita i promene baze podataka, kao i učestalost izvršavanja i posebno opterećenje. Za upite se prepoznaje koje se relacije koriste, koji atribute učestvuju u uslovima spajanja i restrikcije.

Optmizacija na nivou strukture baze podataka

Trebamo da prepoznamo potrebne promene u fizičkoj shemi:

- *Uvodjenje pogleda da bi se sakrile načinjene promene* - Ako se promeni fizička strukutra baze podataka, onda na konceptualnom nivou mogu da se uvedu novi pogledi koji sakrivaju načinjene izmene
- *Alternativna normalizacija* - Pri normalizaciji ne postoji jedino moguće rešenje. Ako jedno rešenje ne daje zadovoljavajuće rezultate, možda nešto drugo daje.
- *Denormalizacija* - Promena fizičke strukture BP tako da se svesno narušava dostignuta normalna forma. Optimizacija baze podataka često predstavlja ovaj pristup uz tehnike: *dekompozicije*, *kompozicije* i *dupliranih podataka*(replikacija).

Uloga *dekompozicije* je veoma važna u optimizaciji baze podataka. Ako imamo tabelu sa mnogo redova ili kolona, koja se često menja, onda promene mogu da budu neefikasne. Jedan od načina da se ubrza rad je podela na dva načina:

- *horizontalna* - redovi tabele se podele u dve tabele sa istom strukturom. Jedna tabela obično ima aktivne podatke, dok druga ima arhivirane. Particionisanje tabela je vid horizontalne podele.
- *vertikalna* - ako se većina kolona ne koriste u svim upitima, već samo relativno retko, onda se česte operacije mogu ubrzazi vertikalnom podelom. Prave se dve ili više tabela - svaka sadrži kolone primarnog ključa, a ostale se grupišu prema potrebi.

"Dogma o denormalizaciji": Uvažen je stav da je normalizovana baza neefikasna. Međutim, to važi samo za specifične okolnosti i nije opšte pravilo. Denormalizacija vrlo često doprinosi efikasnosti ili sasvim malo ili nimalo, to je samo jedan od mogućih koraka.

Kompozicija, iliti spajanje tabela, je isto jedna od tehnika optimizacija. Tabele koje se često spajaju u upitima se mogu korisno spojiti u jednu tabelu. Spajanje je veoma skupa operacija i ovakva promena može da ima dugotrajne efekte. Dobijena tabela narušava obično normalne forme i sadrži neke redundantnosti. To dodatno otežava održavanje podataka i staranje o integritetu.

Optimizacija hijerarjhija entiteta

Pri pravljenju logičkog modela hijerarhija može da se prevede u relacije na sledeće načine:

1. jedna integrisana tabela
2. za svaki tip zasebna tabela, koja ima samo kolone ključa i kolone novih atributa
3. za svaki neapstraktan tip zasebna tabela, koja ima sve potrebne kolone
4. različite kombinacije

Čest problem sa hijerarhijama je da pri pravljenju logičkog modela se razmatraju kriterijumi logičkih veza medju podacima. Radi toga je bolje zbog efikasnosti da pripremimo drugo rešenje.

Rešenje za hijerarhije je da se sve operacije čitanja odvijaju kroz poglede koje zaokružuju sve potrebne podatke o elementima hijerarhije. Da se “zabrane” direktne operacije menjanja podataka u tabelama koje pripadaju hijerarhiji i sva pisanja izvode kroz okidače na pogledima ili predefinisane procedure.

Sprecifičnosti fizičkog modela mogu da se sakriju pogledima. Problem klasičnih pogleda je da se iole složeniji pogledi ne mogu koristiti za menjanje podataka, već samo za čitanje. *Okidači nad pogledima* se definišu kao zamena za operacije neposrednog menjanja podataka.