# PODATKOVNE BAZE

## 1.1 O PODATKIH

Človek si svoj **miselni model sveta** gradi z vtisi, ki jih vrednoti in sprejema s svojimi čutili iz sveta, v katerem živi. Nanaša se na **fizično obstoječi svet** in na **svet idej**.

Posameznikov ali skupinski pogled na svet so ljudje že od nekdaj zapisovali in s tem omogočili dostop do lastnega, tujega oziroma **skupnega znanja**. Če se pri zapisovanju uporablja konvencija, ki omogoča jedrnato predstavitev dejstev, pravimo takim zapisom podatki.

**Podatek** je predstavitev nekega dejstva na **formaliziran** način, ki je dogovorjen s **konvencijo** in primeren za komunikacijo, interpretacijo in **obdelavo** s strani človeka ali stroja. Podatek je **diskreten**, če se pri njegovi predstavitvi uporabljajo simboli ali **analogen**, če se za predstavitev uporablja kakšna fizikalna veličina. Podatku se lahko pripiše **pomen le v določenem kontekstu**.

**Informacija** je pomen (spoznanje), ki ga človek pripiše podatkom s pomočjo znanih konvencij, ki so uporabljene pri njegovi predstavitvi. **Podatki posredujejo informacijo prejemniku**, katerega sprejemna struktura je konsistentna z izbrano predstavitvijo podatkov in modelom sveta, na katerega se nanašajo.

Osnovne **komponente podatkovnega sistema** so:
- Človek
- Program
- Podatki
- Računalnik

Z razvojem strojne in programske opreme se je **odnos** med njimi **spreminjal**. V prvem obdobju je bil v središču pozornosti **računalnik**, ki so se mu prilagajale ostale tri komponente. S programsko revolucijo je prešla pozornost na **program** (programski jezik), ki se je približal človeku in postal **neodvisnejši od računalnika**. Po podatkovni revoluciji je največ pozornosti posvečene **podatkom**, ki so se približali človeku in postali razmeroma **neodvisni od strojne in programske opreme**.

## 1.2 PODATKOVNA BAZA

Podatkovna baza je **model okolja**, ki služi kot osnova za sprejemanje odločitev in izvajanje akcij. Je zbirka podatkov z lastnostmi:
- Mehanizirana
- Večuporabniška
- Formalno definirana
- Centralno nadzorovana

**Upravljanje PB**, ki ga izvaja sistem za upravljanje podatkovne baze (SUPB) zajema:
- Zagovaljanje **razpoložljivosti** podatkov (sem sodi tudi skrb za celovitost podatkov)
- Uporabo podatkov v skladu z njihovim **namenom**
- Uporabnost podatkov tudi v **prihodnje**

**Razpoložljivost** podatkov pomeni:
- Zagotovljen učinkovit dostop
- Vsem uporabnikom
- Sočasno
- Do vseh vrst podatkov
- Ves čas

**Celovitost** podatkov pomeni:
- Preverjanje vhodnih podatkov ob ažuriranjih PB
- Obnavljanje PB v primeru nesreč
- Nadzor nad sočasnim dostopom do podatkov
- Sočasno ažuriranje vseh kopij podatkov v PB, če le-te obstajajo

Za zagotovitev pravilne in dovoljene uporabe podatkov v skladu z njihovim namenom mora v okviru PB obstajati uporabnikom **razumljiv opis** njihovega pomena in **sistem nadzorovanih dostopov** do njih.

Razen podatkovnih baz za shranjevanje formatiranih podatkov obstajajo tudi **druge vrste podatkovnih sistemov**:
- Referenčni sistemi (desktiptorji, fizična lokacija dokumenta)
- Sistemi za obdelavo besedil (kreiranje besedil)
- Bibliografski sistemi (kombinacija zgornjih dveh)
- Specializirane podatkovne baze (veliko podatkov ali veliko različnih vrst podatkov)
- Baze znanja (kvalitativna narava)
- Objektne podatkovne baze (podatki + znanja, kako podatke uporabiti)

**Podatkovno bazo sestavljajo**:
- Podatki
- Uporabniki in uporabniški programi
- Upravitelj podatkovne baze
- SUPB

**Podatkovni del** PB sestavljata **fizična** podatkovna baza (FPB) in **metapodatkovna** baza (MPB).

V skladu s trinivojsko arhitekturo PB je **razdeljena MPB na**:
- Notranjo shemo
- Konceptualno shemo
- Zunanje sheme, ki predstavljajo tri vrste opisov FPB

**FPB se kaže kot**:
- Na nivoju **operacijskega sistema** se **FPB** kaže kot zbirka **fizičnih datote**k shranjenih v zunanjem pomnilniku računalniškega sistema
- Na **notranjem** nivoju je FPB predstavljena kot zbirka **logičnih zapisov** različnih tipov in njihovih medsebojnih povezav
- Na **konceptualnem** nivoju jo dojemamo kot imena, lastnosti in povezave **entitet** iz modeliranega okolja
- Na **zunanjem** nivoju se FPB pokaže kot **uporabnikov model okolja**

**Trinivojska arhitektura** PB omogoča in zagotavlja **podatkovno neodvisnost**, ki jo delimo na fizično in logično podatkovno neodvisnost;

- **Fizična podatkovna neodvisnost** je mera za vpliv sprememb z notranjega nivoja na konceptualni nivo
- **Logična podatkovna neodvisnost** pa je mera za vpliv sprememb s konceptualnega nivoja na zunanji nivo

**Uporabnike PB delimo** na:

- Posredne
- Neposredne

**Neposredne uprabnike** delimo na:

- Končne uporabnike (parametrični, menijsko vodeni, uporabniki povpraševalnega jezika)
- Uporabniške in sistemske programerje, ki so v pomoč končnim uporabnikom
- Upravitelja PB

**Naloge upravitelja PB**:

- Definiranje in ažuriranje vseh treh shem
- Kreiranje in inicializacija FPB
- Obnavljanje PB po nesrečah
- Vzdrževanje pristopnih dovoljenj in gesel
- Nadzor performans
- Izvajanje reorganizacij in prilagoditev
- Pomoč uporabnikom

**SUPB** izvaja **dostopne in kontrolne funkcije** pri upravljanju PB

**Dostopne funkcije** omogočajo:
Upravitelju PB:

- Definiranje shem
- Kreiranje in reorganiziranje PB

Splošnim uporabnikom:

- Zajemanje in ažuriranje podatkov v PB

Delovanje **kontrolnih funkcij** je prikrito uporabnikom, aktivirajo se samodejno pri uporabi dostopnih funkcij. Služijo UPB-ju. Delimo jih na:

- Zaščitne (celovitost PB, uporabniške pravice)
- Nadzorne (zbiranje podatkov o uporabi, odkrivanje zlorab)

Poglavini **moduli**, iz katerih je sestavljen **SUPB**:

- Kontrolni sistemi
- Povpraševalni procesor
- Menijski procesor
- Metapodatkovni procesor
- Predprevajalniki

**SUPB** se lahko nahaja v **istem računalniku** kot uporabniški programi, lahko pa se nahaja v samostojnem **podatkovnem strežniku**, ki je preko hitre komunikacijske poti povezan z enim ali več računalniškimi sistemi, v katerih se izvajajo uporabniški programi.

**Podatkovna baza** je lahko shranjena zgolj **v enem strežniku**, lahko pa je porazdeljena preko **več strežnikov**, ki jih upravljajo PSUPB. Od stopnje porazdelitvene **transparentnosti** je odvisno, ali dojema uporabnik podatkovno bazo kot **enotno – globalno bazo**, ali pa se zaveda njene **porazdelitve**, ki jo mora potem pri povpraševanju tudi upoštevati (poleg ukaza še ime strežnika, ki ga naslavlja).

## 1.3 FIZIČNA PODATKOVNA BAZA

Podatkovni del PB sestavljata FPB in MPB. V obliki **fizičnih datotek** se shranjujeta v **zunanjem pomnilniku**, ki je danes najpogosteje **diskovni pomnilnik**. Za dostop do fizičnih datotek v zunanjem pomnilniku skrbi **operacijski sistem**, za preslikavo logičnih datotek v fizične pa SUPB – neodvisnost od fizičnega pomnilnega medija.
Lastnosti pomnilnika:

- Shraniti razmeroma velike količine podatkov
- Trajno pomnenje
- Hitro ažuriranje in dostop do podatkov
- Nizka cena na enoto shranjenih podatkov

**Elementarne pomnilniške enote** v diskovnem pomnilniku so:

- Sektorji (vsak izmed koncentričnih krogov – **sledi**, je razdeljen **na sektorje**)
- Cilindri (sledi na vseh magnetnih ploščah, ki so dostopne bralno-pisalnim glavam v določenem položaju)

Bere / piše se lahko **najmanj 1 sektor**. Najprej se bralno-pisalne glave premakne nad cilinder (iskalni čas), nato se aktiviro eno izmed njih, s čemer se izbere površina v cilindru (magnetna plošča). Sledi zasuk plošče do začetka iskanega sektorja (rotacijska zakasnitev) in branje/pisanje podatkov (odvisno od št. sektorjev na prošči).

**Diskovni pomnilnik** je razdeljen na:

- Podatke o diskovni enoti (na prvem sektorju (0,0,0) – OS prepozna napravo)
- Dodelitveno tabelo (podatki o legi fiz. datotek – vektor)
- Datotečni seznam (podatki o fiz. datotekah – ime, atributi, lastnik, velikost,...)
- Dodelitvene enote (najmanjši del pomnilnika za fiz. datoteko, sestavljena iz sektorjev)

Sektorji, ki sestavljajo dodelitveno enoto, so vedno na istem cilindru, saj to omogoča zaporedno branje enot brez premikanja bralno-pisalnih glav.
**S stališča SUPB** je fizična datoteka sestavljena iz enako velikih **fizičnih blokov**, katerih vsak obsega enega ali več sektorjev oziroma eno ali več dodelitvenih enot. V notranjem pomnilniku je vsaki aktivni **fizični datoteki** prirejenih eden ali več **datotečnih vmesnikov** v velikosti fizičnega bloka, preko katerih se izvaja dostop do datoteke. Za korespondenco med fizičnimi bloki in sektorji na disku skrbi operacijski sistem.

**Logična datoteka** je množica **logičnih zapisov** istega tipa, ki s pomočjo podatkovnih elementov opisujejo enakovrstna dejstva. **Tip zapisa je formalizem**, s pomočjo katerega se lahko oblikujejo in berejo zapisi pripadajočega tipa. S tipom zapisa je predstavljena struktura zapisov ter vrsta in oblika podatkovnih elementov, ki sestavljajo zapise. Zapisi so lahko spremenljive ali nespremenljive dolžine.

tip Rojen = (Ime: char(10),

Kraj: char(10) )

Podatkovne elemente, na katerih je možna identifikacija posameznih zapisov, imenujemo **ključ zapisa**, ki **obsega enega ali več** podatkovnih **elementov** navedenih v tipu zapisa. Ključ je lahko razločevalen (v datoteki ne smeta obstajati 2 zapisa z isto vrednostjo ključa) ali nerazločevalen, služi pa tudi za urejanje zapisov v fizičnih datotekah.

Logični zapisi se s pomočjo **fizičnih zapisov** shranjujejo v fizičnih datotekah. Fizični zapisi so po strukturi lahko enaki logičnim zapisom, lahko pa vsebujejo tudi dodatne (meta)podatkovne elemente. Fizični zapisi se **zapisujejo v naslovljiva polja**, na katera so razdeljeni fizični bloki. Posamezna polja lahko vsebujejo zapise ali pa so prosta (nezasedena). Če je pomemben naslov polja, v katerem je shranjen posamezen zapis, potem je zapis **vezan** na polje, v nasprotnem primeru pa je **nevezan**.

**Osnovne operacije nad zapisi** v fizičnih datotekah:
- Iskanje zapisov po datoteki
- Dodajanje zapisov v datoteko
- Brisanje in spreminjanje v datoteki obstoječih zapisov

Zapise lahko iščemo po vrednosti poljubne kombinacije podatkovnih elementov, pri čemer je **iskanje po vrednosti ključa najhitreje** izvedljivo, ker je podprto z datotečno organizacijo. Iščemo jih lahko tudi s pomočjo naslovov polj ali skupin polj, v katerih naj bi se nahajali.

Najpreprostejša datotečna organizacija je **neurejena datoteka**, pri kateri ni predpisa, ki bi urejal lego zapisov v njej. **V zaporedni datoteki so zapisi urejeni po vrednosti ključa** – ključ predhodnega zapisa mora biti manjši ali enak ključu naslednika. Zaporedje je lahko urejeno s **fizičnim zaporedjem** zapisov v datoteki (samo za nevezane datoteke) ali pa s **kazalci** (če so zapisi vezani na polje).

Za **iskanje zapisov** po datoteki je v odvisnosti od njene urejenosti uporabno:
- Zaporedno (iskanje po vrednosti kateregakoli elementa)
- Dvojiško (bisekcija)
- Statistično (če poznamo razporeditev vrednosti ključev, po formuli)
- Neposredno (lega zapisa odvisna od vrednosti ključa)

V **razpršenih datotekah** urejajo lego zapisov v njih razpršilne funkcije (hash funkcije). Zapisom se lahko prirejajo polja ali pa skupine polj. Če se 2 ali več zapisom priredi isti domači naslov, govorimo o **prelivu** (overflow). Pri **statičnem** razprševanju se število skupin ne spreminja, pri **dinamičnem** pa se prilagaja zasedenosti datoteke z zapisi (odpravlja problem preliva). Pri razpršenih datotekah je **možen hiter dostop** do posameznih zapisov.

**Indeks** je datoteka, ki s pomočjo **sistema kazalcev** omogoča **hiter dostop** do zapisov z njo indeksirane osnovne datoteke in tudi zaporedni dostop do zapisov nezaporedne datoteke. Z ozirom na to ali so indeksirani vsi zapisi osnovne datoteke ali ne, je indeks lahko **gost** ali **redek** – s kazalci na skupine polj (le če je osnovna datoteka zaporedna in indeks primarni indeks). Če je indeksiranje izvedeno **po ključu** je indeks **primaren**, sicer pa **sekundaren** (popolni ali selektivni – indeksirajo se le izjeme). V primeru, ko je tudi indeks indeksiran, govorimo **o večnivojskem indeksiranju**. Indeksiramo ga z redkim indeksom. V primeru, ko se indeks prilagaja vsebini indeksirane datoteke, je indeksiranje **dinamično**, sicer pa **statično** (različni iskalni časi za različne vrednosti ključev).

**Indeksiranje B** je primer redkega večnivojskega dinamičnega indeksiranja zaporedne datoteke, indeksiranje **B+** pa je gosto večnivojsko dinamično indeksiranje, ki se lahko uporabi pri poljubno organizirani osnovni datoteki (med indeksom B in osnovno datoteko se nahaja **še gosti indeks** na osnovno datoteko).

**Lastnosti indeksa B:**

- Oblikovan je kot uravnoteženo drevo
- Vozlišča drevesa so fizični indeksni bloki
- Vsak indeksni zapis razen prvega je sestavljen iz para (ključ, kazalec)
- Indeksni zapisi so v blokih urejeni zaporedno po vrednostih ključev
- Na nivoju 0 kaže na fizični blok osnovne datoteke oz. gostega indeksa
- Vrednost ključa v najbolj levem delu vozlišča je določena s kazalcem na nižji nivo

## 1.4 PODATKOVNI MODELI

**Podoba sveta** je človekova predstava o svetu, v katerem živi in vključuje **tudi njega samega**. Človekovo modeliranje sveta s podatkovno bazo temelji na podobi sveta, ki jo je možno opisati s **propozicijami** - pomenskimi sestavami stavka v naravnem jeziku.

**Propozicijo sestavljajo**:

- Predikati (predstavljajo posamezni dogodek ali stanje)
- Participanti (predstavljajo udeležence v dogajanju)
    - Aktanti (neposredni udeleženci)
    - Cirkumsanti (okoliščine)

**Entiteta** je tisti najmanjši del, ki ga lahko ločimo ali ga želimo ločiti od drugih delov podobe sveta in je predstavnik posamezne stvari ali dogajanja, ki obstaja ali mislimo, da obstaja v svetu. Entitete, ki obstajajo, imenujemo **participativne** entitete, in so udeleženke v dogodkih, entitete pa, ki se dogajajo, imenujemo **predikatne** entitete.

**Entitete pripadajo entitetnim tipom** (ustrezajo določeni predstavi), ki se dele na **naravne tipe** (neka lastnost jim pripada po naravi) in **tipe z ozirom na vlogo** (pripišemo neko lastnost). Entitetnemu tipu pripada **entitetna množica**, ki je časovno spremenljiva. Entitete imenujemo s pomočjo besed, ki jih imenujemo entitetna imena. **Entitetna imena** so podatki.

**Dogodki** so dogajanje, v katerem entitete sodelujejo ali stanje, v katerem se entitete nahajajo. Za vse dogodke, ki pripadajo določenemu **dogodkovnemu tipu**, veljajo določene zakonitosti.

**Pravilo** predstavlja veljavno in nespremenljivo **zakonitost o razmerjih** med entitetami v dogodkih določenega tipa. Pravila se dele na preslikovalna (razmerja med entitetami) in vrednostna (ekstenzije imenskih tipov).

**Podatkovni model** je strukturni mehanizem, s pomočjo katerega specificiramo zunanje, konceptualno in notranjo shemo podatkovne baze. Obsegajo podatkovno strukturo, operacije nad podatkovno strukturo in integritetne omejitve.

**Podatkovni modeli se delijo na**:

- **Fizične modele** (na notranjem nivoju, opis po datotečni plati)
- **Logične modele** (na konceptualnem in zunanjem nivoju, opis po pomenski plati)
    - Površinski
        - Relacijski (temelji na relacijah)
        - Mrežni (temelji na grafih)
        - Hierarhični (temelji na drevesih)

Imajo **določene slabosti**, predvsem pri semantiki, ki ostaja stvar konvencij, pri uveljavljanju integritetnih omejitev (uporabnik sam skrbi) in pri zagotavljanju podatkovne neodvisnosti. Primerni za računalnike in njihove programerje.

- Globinski
  - "entiteta-razmerje"
  - binarni podatkovni model
  - podatkovni modeli na osnovi semantičnih mrež
  - infološki podatkovni modeli

  Osnovne ideje so predvsem razlikovanje med E. tipi, E. imeni in entitetami, zmanjšanje semantične razdalje, hierarhična razvrstitev E. tipov in opredelitev njihovih vlog v relacijah. **Bolj so prilagojeni** končnim uporabnikom – ljudem.

**Problem sožitja** med površinskimi in globinskimi podatkovnimi modeli rešuje **konsistenčni princip**, po katerem je konceptualni nivo podatkovne baze razdeljen na **infološki** (globinski PM) in **dataloški** del (površinski PM).

## 1.5 RELACIJSKI PODATKOVNI MODEL

Relacijski podatkovni model se uporablja za predstavitev PB na **konceptualnem** in **zunanjem nivoju**, hkrati pa je tudi sredstvo za **specifikacijo** njene konceptualne in zunanjih **shem**.

Lastnosti:
- Je formalno definiran
- Ne vsebuje elementov fizičnega shranjevanja podatkov
- Relacije, na katerih temelji, so predstavljive s tabelami

Osnovna koncepta relacijskega podatkovnega modela sta domena in relacija.

**Domene** so množice, njihovi elementi pa so vrednosti predstavljive v obliki podatkov.
(domena, ki obsega imena I = {Tine, Meta, Jure, Ana})

**Atribut** je preslikava množice objektov O v pripadajočo domeno $D_i$. Če je atribut **funkcija**, se relacija nahaja v 1. normalni formi in je predstavljiva z dvo-dimezionalno tabelo. Relacijo v tem primeru definiramo s pomočjo preslikav $A_1$, $A_2$,....
(Atribut: Ime: O → I)

**Relacija** je podmnožica kartezijskega produkta liste domen oziroma množica n-teric, katerih komponente so elementi domen. Število domen, nad katerimi je relacija definirana, se imenuje **stopnja** relacije, številno n-teric v njej pa **moč** relacije.
r= { (Ime($o_1$)), (Ime($o_2$)), (Ime($o_3$)), (Ime($o_4$)) } = { (Tine), (Meta), (Jure), (Ana) }

**Relacijska shema** (čelna vrstica) pojasnjuje pomen pripadajoče relacije. Sestavljata jo oznaka relacijske sheme in lista oznak atributov s pripadajočimi oznakami domen.
R ($A_1$:$D_1$, $A_2$:$D_2$,...$A_n$:$D_n$) – razviden **pomen relacij**
Relacijske sheme predstavljajo **semantično komponento** relacijske PB in so del konceptualne oz. zunanjih shem. Relacijska shema lahko pojasnjuje več relacij, obratno ni možno. Relaciji, ki se razlikujeta med seboj le po

vrstnem redu komponent v n-tericah, z matematičnega stališča nista enaki, po pomenu – informacijski vsebini – pa sta ekvivalentni.

Nekatere zakonitosti v svetu, ki ga relacije modelirajo, se v relacijskem podatkovnem modelu opišejo s pomočjo funkcionalnih, večvrednostnih in stičnih odvisnosti. **Funkcionalna odvisnost** množice atributov Y od množice atributov X velja v relacijski shemi, če v nobeni pripadajoči relaciji ne moreta obstajati dve n-terici, ki bi se **ujemali v vrednostih atributov X in se ne bi ujemali v vrednostih atributov Y**.

**Ključ relacije** oziroma njene sheme je tista **minimalna množica** atributov, ki **funkcionalno določa vse** ostale atribute v relacijski shemi. S pomočjo vrednosti ključa lahko ločimo med seboj poljubni 2 n-terici v relaciji. Relacijska shema ima lahko tudi več ključev, eden izmed njih se izbere kot **primarni ključ**. **Nadključ** je vsaka množica atributov, v kateri je vsebovan ključ.
**Operacije nad relacijami** delimo v:

- Operacije, ki omogočajo povpraševanje oziroma iskanje podatkov v PB
- Operacije, ki omogočajo ažuriranje podatkov

Omenjene operacije so sestavni del **povpraševalnega jezika**. Z ažurirnimi operacijami se n-terice dodajajo relacijam, brišejo iz njih in se spreminjajo vrednosti njihovim komponentam. Povpraševalni del povpraševalnega jezika lahko temelji na relacijski algebri ali relacijskem računu.

Povpraševanje s pomočjo operacij **relacijske algebre** temelji na **zaporedju** operacij, ki se izvedejo nad relacijami PB (postopkovni jeziki). Rezultat povpraševanja je spet relacija.

**Osnovne** operacije relacijske algebre:

- Unija (n-terice iz obeh relacij, brez podvajanja)
- Razlika (n-terice iz p, ki niso hkrati v r)
- Kartezijski produkt (vsi možni stiki n-teric obeh relacij)
- Projekcija (n-terice, sestavljene le iz komponent, ki jih izberemo pri projekciji)
- Selekcija (n-terice, ki pri zamenjavi prostih spremeljivk v F naredijo pravilno izjavo)

**Izvedene** operacije so:

- Presek (n-terice, ki nastopajo v obeh relacijah)
- $\theta$-stik (selekcija + kartezijski produkt)
- Naravni stik (selekcija + kartezijski produkt + projekcija)
- Količnik (razlika atributov prve in druge sheme)

Povpraševanje s pomočjo **relacijskega računa** temelji na specifikaciji **lastnosti** relacije, ki jo želimo kot rezultat povpraševanja (nepostopkovni jeziki). Iskana relacija se opiše z izrazom, ki zajema **formulo predikatnega računa** z **eno prosto** n-terično spremenljivko (n-terični račun) ali **več prostimi** domenskimi spremenljivkami (domenski račun). Izraz relacijskega računa je **varen**, če določa končno relacijo, pri čerem je za zagotovitev, da posamezna n-terica v njej nastopa, dovolj preiskati le relacije v podatkovni bazi. V relaciji, ki jo določa, nastopajo le n-terice s komponentami iz **Dom(F)** in da **podformule** "eksistira" in "za vsak" vrnejo končno mnogo možnosti.

Primer relacijskega povpraševalnega jezika, ki **združuje** ukaze za **rokovanje** s podatki in njihovo **deklariranje, kreiranje in zaščito**, je **SQL**. Podatki so predstavljeni v obliki tabel in pogledov, pri katerih deklaraciji se lahko specificirajo integritetne omejitve kot sta **ključ** in **obvezno podana vrednost** atributov ob dodajanju vrstic (NOT

NULL). Povpraševalni del jezika sloni **na n-teričnem relacijskem računu**, kar uvršča SQL med **nepostopkovne jezike**. SQL je primeren za **interaktivno** rabo, njegovi ukazi pa so lahko tudi vgnezdeni v gosteče jezike.

## 1.6 MREŽNI PODATKOVNI MODEL

Podatkovni strukturi, v kateri nastopajo med seboj povezani zapisi, pravimo **mreža**. Njena osnovna gradnika sta **zapis in set**. Povezave med zapisi so določene s funkcijo S, ki preslikuje zapise enega tipa v zapise drugega tipa (lahko si jih predstavljamo kot razmerja med entitetami). S sme biti totalna ali parcialna funkcija in je v mreži predstavljena s seti.

**Zapis** je sestavljen iz podatkovnih elementov, katerih vsak predstavlja vrednost posamezne lastnosti entitete. (lahko si jih predstavljamo kot entitete)

**Set** je urejena množica zapisov, ki jo sestavljajo **lastnik** (zapis enega tipa) in **člani** (zapisi drugega tipa). Posamezen zapis je lahko sočasno lastnik enega ali več setov, lahko pa, da ne nastopa v nobenem setu. Posebne vrste set je **sistemski set**, ki služi za ureditev in zaporedni dostop do zapisov določenega tipa.

**Mrežna shema** pojasnjuje obliko in pomen zapisov ter njihovih povezav v mreži. Sestavljajo jo **deklaracije tipov zapisov in tipov setov**.

Deklaracija tipa **zapisa** obsega:
- Ime tipa
- Imena in zaporedje podatkovnih elementov
- Navedbo ključa, ki je lahko razločevalen ali nerazločevalen

Deklaracija tipa **seta** obsega:
- Ime tipa seta
- Ime tipa lastnikov
- Ime tipa članov
- Navedbe urejenosti setov, vrste članstva in načina včlanjevanja v sete.

Set je lahko **urejen** z zaporedjem dodajanja članov vanj ali po vrednosti ključa članov.

**Članstvo** v setu je lahko
- Fiksno
- Obvezno
- Opcijsko

Način vključevanja članov v set je lahko **ročen ali avtomatičen**

**Vrednostne integritetne omejitve** so v mrežni shemi specificirane s pomočjo tipov podatkovnih elementov v zapisih (dopustne vrednosti podatkovnih elementov), **preslikovalne omejitve** pa s pomočjo deklaracije ključa zapisov in deklaracije vrste članstva v setih.

**Operacije nad mrežo** se dele v:

- Operacije za **iskanje** zapisov
    - Zapis se lahko poišče s pomočjo specifikacije vrednosti **ključa** (ali kakega drugega podatkovnega elementa) ali s pomočjo **setov** (povezav med zapisi). Z ozirom na tekoči zapis lahko poiščemo njegovega **predhodnika ali naslednika** v okviru specificiranega seta. Zapisu članu lahko poiščemo lastnika seta, prav tako pa tudi prvi ali zadnji zapis v tekočem setu.

- Operacije za **ažuriranje** zapisov
    - Zapis lahko dodamo mreži, na podlagi poiskanega naslova zapisa pa preberemo, ažuriramo ali izbrišemo iz mreže. Brisanje se lahko nanaša le na tekoči zapis, lahko pa prizadane kaskadno tudi člane njegovih setov.

- Operacije za **včlanjevanje** zapisov
    - Zapisi se z ozirom na vrsto članstva lahko v sete včlanijo, se iz njih izčlanijo ali pa prečlanijo iz enega v drugi set istega tipa.

Operacije nad mrežo se lahko izvedejo uspešno ali neuspešno. Kako se je operacija izvedla, sporoča SUPB preko **statusnega polja**, ki ga mora **uporabniški program** po zaključku vsake operacije nad PB **preveriti**.

## 1.7 HIERARHIČNI PODATKOVNI MODEL

Osnovni gradnik hierarhične podatkovne strukture je **drevo**. Množica dreves istega tipa tvori **gozd**, hierarhična podatkovna struktura baza pa obsega enega ali več gozdov. Uporabnikov pogled na PB je v določenem trenutku omejen na en sam gozd.
Drevesa so sestavljena iz med seboj povezanih zapisov. **Povezave zapisov** v drevesa določajo preslikave med logičnimi datotekami. Preslikava ene logične datoteke v drugo je **totalna funkcija**, pri čemer sme posamezna datoteka nastopati kot domena le v eni preslikavi. Zapisom, ki nastopajo v domeni preslikave pravimo **otroci**, zapisi, ki nastopajo v območju preslikave pa so njihovi **starši**. Listi drevesa so le otroci, zapis v korenu pa le starš, preostali zapisi pa nastopajo v obeh vlogah hkrati. Zapisi so v drevesih in gozdu urejeni v hierarhično zaporedje.

Vsa drevesa posameznega gozda so **enako strukturirana**, v vseh se na istih nivojih nahajajo zapisi iz istih logičnih datotek. Tip drevesa, ki mu pripadajo, izraža njihove skupne lastnosti in pomen.

**Deklaracija tipa drevesa** obsega:

- Ime drevesa
- Deklaracije tipov zapisov
    - Ime tipa zapisa
    - Ime starša
    - Deklaracijo ključa
    - Listo imen podatkovnih elementov vključno z njihovimi osnovnimi tipi

Hierarhično shemo hierarhične PB sestavljajo deklaracije tipov dreves, ki nastopajo v posameznih gozdovih. **Transformacija** hierarhične sheme v mrežno je preprosta in enolična. Poglavitna **slabost** hierarhičnih struktur je redundantno predstavljanje entitet z zapisi, saj so hierarhične strukture namenjene predstavitvi hierarhij med entitetami. Predstavitev nehierarhičnih odnosov pa vodi do redundančnih problemov.

**Operacije nad gozdom** omogočajo dostop do zapisov in njihovo ažuriranje v drevesih gozda, ki predstavlja trenutni uporabniški pogled. Povpraševanje se izvaja z uporabo **iskalnih ukazov**, ki omogočajo poiskati in prebrati **prvi zapis** navedenega tipa, ki ustreza iskalnemu pogoju, poiskati in prebrati **naslednji zapis ob enakem iskalnem pogoju** ter poiskati in prebrati **naslednji zapis v okviru istega starša** (istega poddrevesa). Iskanje zapisov poteka vedno v skladu s hierarhičnim zaporedjem.

**Ažurirne operacije** omogočajo:

- Brisanje zapisov v gozdu (se nanaša na tekoči zapis, briše se poddrevo)
- Dodajanje zapisov v gozdu (vključimo enega izmed dreves ali poddreves)
- Ažuriranje zapisov v gozdu (se nanaša na tekoči zapis, le na elementih, ki niso vsebovani v ključu, saj je s ključem določena lega zapisa v drevesu)

Za predstavitev **nehierarhičnih struktur** se uporablja koncept **povezanih dreves**, kjer se povezujejo med seboj drevesa **različnih fizičnih gozdov**. Zapisi so povezani v fizično drevo s pomočjo fizičnih kazalcev. Kazalci, ki povezujejo med seboj zapise **različnih tipov**, pripadajo tipu **otrok**, kazalci, ki povezujejo med seboj zapise **istih tipov**, pa pripadajo tipu **dvojček**.

**Drevesa različnih tipov** se povezujejo med seboj s pomočjo kazalcev tipa:

- Logični otrok
- Logični dvojček
- Logični starš

S povezanimi drevesi **rešujemo** problem zapisov, ki bi se naj kot otroci nahajali v več gozdovih in s tem povzročali **redundančne težave**. Nad povezanimi fizičnimi gozdovi je možno definirati vrsto **uporabniških pogledov – logičnih gozdov**.
Logični gozd lahko definiramo tudi s pomočjo **inverzije dreves**. Inverzija dreves temelji na **direktnem dostopu** (sekundarni indeks) do zapisov določenega tipa, ki se v drevesih fizičnega gozda nahajajo hierarhično pod korenom. Zapis z direktnim dostopom **postane** v logičnem drevesu **korenski** zapis, ki vključuje tudi nadrejene zapise, hierarhija podrejenih zapisov pa ostane nespremenjena.

## 1.8 OBNAVLJANJE PODATKOVNE BAZE

Postopke, s katerimi se ohranja konsistentnost podatkovne baze v primerih podatkovnih nesreč, imenujemo **obnavljanje**. Izvaja jih **SUPB pod nadzorom UPB**.
**Obnavljanje** obsega:

- Pripravo podatkov za obnovitev
- Detekcijo podatkovne nesreče
- Obnovitev PB v veljavno stanje pred nesrečo

**Podatkovne nesreče** se dele v:

- Transakcijske nesreče (lokalnega značaja, vplivajo le na posamezno transakcijo)
  - Odkrijejo uporabniški programi sami (vhodni podatki protislovni)
  - Odkrijeta SUPB ali OS (deljenje z 0, neobstoječi naslov)
- Sistemske nesreče (izguba podatkov v RAMu, prizadanejo vse trenutno aktivne transakcije)
- Diskovne nesreče (prizadanejo vse trenutno aktivne transakcije)

Zaporedje ažuriranj, ki povzroče prehod PB iz enega veljavnega stanja v drugo, imenujemo **transakcija**. Ažuriranja PB, ki sestavljajo transakcijo, se **morajo izvesti vsa ali pa niti eno**. Transakcija se prične z operacijo "**Začetek transakcije**", zaključi pa se z ukazom "**Pomni**", ali pa neuspešno z ukazom "**Pozabi**". Transakcija v času svojega obstoja prehaja **iz ene faze v drugo** in je v začetku aktivna transakcija, nato postane uspešna ali pa ponesrešena transakcija, konča pa kot uspešno ali neuspešno zaključena transakcija.

Podatki, ki jih obdeluje transakcijski program, se kot fizična PB nahajajo na disku in kot delovni podatki v notranjem pomnilniku.
**Pomnilniški ukazi**, ki jih v okviru transakcij izvaja transakcijski program, so:

- PoiščiPreberi (delovni spremenljivki priredi vrednost podatka na disku)
- Spremeni (podatku na disku priredi vrednost delovne spremenljivke)
- Dodaj (na disk doda podatek z vrednostjo delovne spremenljivke)
- Zbriši (z diska zbriše podatek)

Omenjene ukaze podpirata operacija "**PreberiBlok**", ki prenese fizični blok z diska v datotečni vmesnik in "**IzpišiBlok**", ki prenese fizični blok iz vmesnika na disk. To stori OS po navodilih SUPB-ja. Vmesniki se ne izpisujejo pogosto, ker obstaja **verjetnost**, da se bo kateri izmed naslednjih ukazov transakcijskega programa **nanašal** na podatke že prebranega bloka.

**Obnavljanje podatkovne baze** temelji na zapisovanju redundantnih podatkov:

- Dvojna PB (preprostejša varianta je kopija PB, ki omogoča obnovitev celotne PB)
- Vhodni transakcijski podatki (omogočajo ponovno izvajanje transakcij, se beležijo v dnevnik)
- Stare vrednosti podatkov pred ažuriranjem (omogočajo razveljavitev ažuriranj, se beležijo v dnevnik)
- Nove vrednosti podatkov po ažuriranju (omogočajo uveljavitev ažuriranj, se beležijo v dnevnik)

**Dvojna podatkovna baza** temelji na podvojenem zapisovanju FPB v dva **ločena** diskovna pomnilnika. Zagotavlja **dobro zaščito** pred diskovnimi nesrečami ter **neprekinjeno dostopnost** PB, saj je v času obnavljanja ene kopijo na voljo druga. Taka vrsta zaščite je draga.

Pri **kopiji podatkovne baze** se vsebina FPB periodično kopira na ločen pomnilniški medij. V času kopiranja naj se transakcije ne izvajajo.

Obnavljanje s **senčnimi stranmi** služi za obnovitev PB po transakcijskih nesrečah. Temelji na **zapisovanju ažuriranih fizičnih** blokov na dotlej neuporabljen prostor **na disku**, ne direktno v PB, pri čemer se ohranjajo tudi prvotni fizični bloki. V primeru transakcijske nesreče obsega PB prvotne bloke, če se transakcija uspešno zaključi, pa se vanjo vključijo namesto prvotnih ažurirani bloki. Uporablja **tekoči in senčni indeks** (stare vrednosti).

**Obnavljanje z dnevnikom** in kopijo temelji na **občasni izdelavi kopije PB**, s katero lahko PB v primeru diskovne nesreče obnovimo v takratno veljavno stanje, ter pisanju dnevnika, v katerega se zapisujejo podatki, s

katerimi je možno **s kopijo obnovljeno PB obnoviti** v zadnje veljavno stanje tik pred podatkovno nesrečo. Dnevnik pa lahko vsebuje tudi podatke, s katerimi je možno vse v trenutku nesreče prekinjene transakcije ponovno izvesti. Pri **odloženem ažuriranju** se nove vrednosti podatkov shranijo **najprej v dnevnik**, po uspešnem zaključku transakcije pa se uveljavijo v podatkovni bazi. Pri **neposrednem ažuriranju** se ažuriranje PB izvaja sproti, **v dnevnik** pa se zapisujejo **stare vrednosti podatkov**. V primeru transakcijske nesreče se ažuriranja v PB z njihovo pomočjo razveljavijo.

Če pri **sistemski ali diskovni nesreči** nismo prepričani, ali se je pri **uspešno** izvedeni transakciji vsebina pomnilniških vmesnikov izpisala na disk, je potrebno njihova ažuriranja ponoviti. Pri **neuspešnih** transakcijah (pri neposrednem ažuriranju) je potrebno transakcijo ponovno razveljaviti. Pri **prekinjenih** transakcijah izvedemo najprej njihovo razveljavitev, nato pa jih vrnemo transakcijskim programom v ponovno izvajanje. Pri uspešni transakciji najdemo v dnevniku zapis z oznako "Pomni", pri neuspešni pa oznako "Pozabi".

V okviru izvajanja **kontrolne točke** (pri uporabi neposrednega ažuriranja) se izvede **izsiljeni izpis datotečnih vmesnikov na disk in registriranje vseh trenutno aktivnih transakcij**, s čimer je v tem trenutku popolnoma definirano stanje FPB. V primeru **sistemske nesreče** se obnovitev PB omeji le na uveljavljanje oz. razveljavljanje transakcij, ki so bile **aktivne ob času** izvajanja kontrolne točke oz. so se pričele **izvajati kasneje**. V primeru diskovne nesreče nam kontrolna točka ne pomaga.

**Dnevnik** se zaradi varnosti pogosto shranjuje na **2 ločena medija**. Ko se zaključijo vse transakcije, katerih zapisi se vpisujejo v dnevnik A, se izvede kontrolna točka - **filtriranje**.

## 1.9 NADZOR NAD SOČASNO UPORABO PODATKOVNE BAZE

Izmenično izvajanje transakcijskih programov pod nadzorom večuporabniškega OS povzroča posledično tudi **izmenično izvajanje transakcij**, kar potencialno ogroža konsistentnost PB. **Naloga nadzora nad sočasno uporabo PB** je ohraniti PB v konsistentnem stanju in zagotoviti konsistentnost povpraševanj, pri tem pa dopustiti kar največjo sočasnost izvajanja transakcij. To se rešuje z zaseganjem podatkov in časovnim označevanjem.

Zaporedje izvajanja ukazov v okviru transakcij imenujemo **razpored**, ki je **zaporeden**, če se pri sočasnem izvajanju transakcij izvedejo najprej vsi ukazi ene transakcije, nato vsi ukazi druge transakcije in tako naprej, in je **izmeničen**, če se časovno med ukazi ene transakcije izvedejo tudi ukazi drugih transakcij. Za zaporedni razpored velja, če vsaka izmed transakcij ohranja konsistentnost PB, potem jo **ohranja tudi vsak njihov zaporedni razpored**. Izmenični razpored transakcij, ki učinkuje na PB enako kot kakšen izmed možnih zaporednih razporedov, imenujemo **zaporedniški** razpored. Če vsaka izmed transakcij ohranja konsistentnost PB, potem jo **ohranja tudi vsak njihov zaporedniški razpored**.

Zaporedniško izvajanje transakcij se lahko doseže z izvajanjem po protokolu, ki temelji na **zaseganju podatkov**, ali protokolu, ki temelji na **časovnih oznakah**. Podatki, ki se v okviru transakcije nameravajo uporabljati, se lahko zasežejo **ekskluzivno**, **deljeno** ali pa se **sploh ne** zasežejo.

**Ekskluzivno** zaseženega podatka ne more dodatno zaseči še kakšna druga transakcija, **deljeno** zaseženi podatek pa se lahko še dodatno **deljeno zaseže**, **ne** more pa se zaseči **ekskluzivno**. Protokol, ki temelji na **ekskluzivnem in deljenem** zaseganju s sprostitvijo zaseženj ob zaključku transakcije, je protokol **PSC**, protokol, ki temelji **samo na ekskluzivnem** zaseganju pa se imenuje **PXC.**

**PXC in PSC** obsegata naslednja pravila:

- Transakcija, ki želi ažurirati podatek, ga mora najprej eks. zaseči
- Če zahteva ne more biti takoj odobrena, preide v stanje čakanja
- Vsa zaseženja se sprostijo šele po zaključku transakcije
- Transakcija, ki želi podatek prebrati, ga lahko prebere neglede na zaseganje

**PSC** pa obsega še:

- Transakcija, ki želi ekskluzivno zaseči podatek, mora imeti prej odobreno njegovo deljeno zaseganje

Ukazi za zaseganje podatkov lahko nastopajo **samostojno** ali pa se **povezujejo** z ukazi za rokovanje s podatki. Protokol PSC dopušča sočasno izvajanje dveh zgolj povpraševalnih T, ki zahtevata zaščito pred ažuriranjem, protokol PXC pa ne. Vendar **PSC** hkrati veliko bolj dopušča nastop **mrtve zanke**. Sprostitev vseh zaseženj je vključena v ukaza "Pomni" in "Pozabi", ki ju transakcija izda ob svojem zaključku.
**Objekti zaseženja** so lahko logični ali fizični objekti.
**Logični objekti** so v odvisnosti od vrste podatkovnega modela:

- Relacije, n-terice v relacijah
- Seti določenega tipa, posamezni seti, logične datoteke, posamezni logični zapisi
- Drevesa določenega tipa, posamezna drevesa, poddrevesa, posamezna vozlišča v drevesih

**Fizični objekti** pa so lahko:

- Celotna FPB
- Fizične datoteke
- Fizični bloki oz. strani
- Fizični zapisi

**Ukazi** pri protokolu **PXC**:

- PoiščiPreberi(X,x)
- E-PoiščiPreberi(X,x)
- Ažuriraj(X,x)

**Ukazi** pri protokolu **PSC**:

- PoiščiPreberi(X,x)
- D-PoiščiPreberi(X,x)
- E-Ažuriraj(X,x)

Če se transakciji **zaseženje ne more pri priči** odobriti, preide v stanje čakanja na odobritev, iz česar izvirata 2 nevarnosti. Posamezno transakcijo lahko pri odobritvi zaseženja določenega podatka prehitevajo vse ostale transakcije in se s tem njeno **čakanje raztegne na nedoločen čas**, kar se rešuje z ustreznim **algoritmom dodeljevanja zaseženj**, ki mora upoštevati čas čakanja transakcij na odobritev zaseženj. Druga nevarnost je **mrtva zanka**, ki nastopi v primeru, ko dve ali več transakcij **čaka druga drugo** na sprostitev podatkov.

**Mrtvi zanki** se lahko **izognemo** z:

- Urnikom izvajanja transakcij (ne dopustimo konfliktnih situacij, omejenost)
- Vnaprejšnjimi zahtevami po zaseženju podatkov
- Ureditvijo objektov zaseženja (zaseženje le v pravilnem zaporedju)
- Odloča transakcijski program
- Prekinitev in ponovno izvajanje transakcij (Čakaj ali izdihni, Rani ali čakaj)

**Čakaj ali izdihni**: TA čaka, če je starejša, drugače se prekine
**Rani ali čakaj**: Ponovno se začne izvajati TB če je TA starejša, če je TA mlajša, čaka TA

Pri obeh protokolih (Čakaj ali izdihni, Rani ali čakaj), se **prekine izvajanje mlajših** transakcij.
Protokol, ki tudi preprečuje nastop mrtve zanke in ne uporablja zaseganja podatkov, je **časovno označevanje** aktivnih transakcij in prebranih ali ažuriranih podatkov. Po tem protokolu se transakcije izvedejo tako, da je njihov **zaporedniški razpored ekvivalenten zaporednemu razporedu**, po katerem se **starejša transakcija izvede pred mlajšo**. **Konfliktne** oz. potencialno konfliktne zahteve po branju ali ažuriranju podatkov se **razrešujejo** s **prekinitvijo** in ponovnim izvajanjem **starejše** transakcije – pri ponovnem izvajanju se ji priredi **nova časovna oznaka**, postane mlajša.

## 2.1 NAČRTOVANJE INFORMACIJSKEGA SISTEMA

**Razvojni cikel** informacijskega sistema obsega:

- **Študijo izvedljivosti**
  - Možne alternativne izvedbe – koristi in stroški
  - Časovno zaporedje faz – najtežji problemi na začetku
- **Zbiranje zahtev in analiza**
  - Določimo namen in cilje
  - Zbiramo zahteve in želje uporabnikov
- **Načrtovanje**
  - Podatkovno (konceptualno, logično in fizično)
  - Postopkovno (specifikacija lastnosti, prog. moduli, prog. specifikacije)
- **Izdelava prototipa** (uporabnike seznanimo z lastnostmi, uporabniški vmesnik)
- **Implementacija**
  - Izberejo se optimalne rešitve
  - Programi se kodirajo, dokumentirajo, integrirajo v module
  - Testiranje
- **Vrednotenje** (pravilno delovanje, učinkovitost, zmogljivosti)
- **Obratovanje in vzdrževanje**
  - Traja vse nadaljne obdobje
  - Sistem potrebno prilagajati ali zamenjati z novim

Pri relacijskih podatkovnih bazah s podatkovnim jezikom SQL med konceptualno (logično) in zunanjo (uporabniško) plastjo ni več prave razmejitve, zato jo obravnavamo **kot eno samo – logično plast**. Na logično shemo je vezan poglaviten del načrtovanja, do katerega lahko pridemo po 2 poteh:

- **Direktno** načrtovanje relacijskih shem z normalizacijo
- **Konceptualno** načrtovanje z enim izmed semantičnih podatkovnih modelov

## 2.2 NAČRTOVANJE RELACIJSKE PODATKOVNE BAZE

Za izhodišče načrtovanja služi **množica atributov**, ki jih načrtovalec spozna kot pomembne lastnosti objektov, nastopajočih v svetu, ki ga namerava predstaviti s podatkovim modelom. Glavni problem predstavlja **grupiranje** atributov v ustrezne **relacijske sheme**. Grupirati jih moramo tako, da se pri ažuriranju relacij in dopolnjevanju PB z novimi atributi ne bodo pojavljale težave.

**Ažurirne anomalije** se kažejo pri spreminjanju vsebine (dodajanju, spreminjanju, brisanju n-teric) relacij v primeru, če le-te niso normalizirane in so posledica odvisnosti med atributi v relacijskih shemah. Če so trditve v relacijah zapisane **redundantno**, se poveča možnost nekonsistence relacij pri **dodajanju in spreminjanju**, pri **brisanju** pa se pojavi problem izgubljanja vseh podatkov, ki so v relaciji.

**Relacija** je preslikava, katere domena je **kartezijski produkt** vrednostnih množic, območje pa dvojiška množica {res, ni_res} – predpostavka zaprtega sveta. Po dogovoru sestavljajo relacijo vse in samo **resnične** trditve o svetu, vrednostne množice pa množice predstavljivih vrednosti.
Vsak objekt lahko okarakteriziramo z lastnostmi, ki jih imenujemo atributi. **Atribut** je preslikava dane množice objektov O v pripadajočo domeno Di.

$A_i$: O → $D_i$

Relacijo lahko definiramo tudi s pomočjo atributov. V tem primeru relacija vsebuje **opise vseh objektov iz** množice **O**, vsak objekt je predstavljen s po eno n-terico. **N-terice** so urejene liste vrednosti atributov, tako da istoležne komponente n-teric predstavljajo vrednosti enega in istega atributa.

**Relacijska shema** predstavlja semantično komponento PB in je del konceptualne oziroma zunanjih shem. Iz nje je razviden **pomen relacije**, ki pripada tej shemi. Sestavljena je iz oznake R, oznak atributov Ai s pripadajočimi oznakami domen Di:

$R(A_1:D_1:,A_2:D_2,A_3:D_3,...A_n:D_n)$

Vsaki relaciji pripada natanko ena relacijska shema, medtem ko lahko posamezna relacijska shema pripada več relacijam.

**Omejitve**, ki veljajo v svetu, predstavimo v relacijah na 2 načina:
- S pomočjo vrednostnih množic (starost ne more biti negativna,...)
- S pomočjo odvisnosti med atributi (funkcionalne, večvrednostne, stične)

Postopku preoblikovanja relacij v obliko, pri kateri anomalije ne morejo nastopiti, pravimo **normalizacija**, izvedemo pa jo s pomočjo **dekompozicije** ali razstavljanja posamezne relacije v več novih – nadomestnih relacij. Posamezna normalna oblika relacije samodejno omejuje njeno dopustno vsebino in s tem preprečuje pojav nekonsistentnosti v njej.

Normalne oblike:
- Prva normalna oblika
- Druga normalna oblika
- Tretja normalna oblika
- Boyce-Coddova normalna oblika
- Četrta normalna oblika
- Peta normalna oblika

Višje normalne oblike postavljajo vse strožje in strožje omejitve na dopustno vrednost relacij.

Relacija se nahaja v **prvi normalni obliki**, če so komponente n-teric, ki jo sestavljajo, zgolj atomarne (nedeljive) vrednosti.
- vsak atribut v relacijski shemi mora biti (enovrednostna) funkcija
- elementi vrednostnih množic (območja atributov) smejo biti le atomarne vrednosti, nikakor pa množice ali relacije

Odvisnost X → Y, kjer sta X in Y dve podmnožici sheme R, je **logično implicirana** z množico odvisnosti $F_R$, če vsaka relacija s shemo R, ki zadošča odvisnostim $F_R$, zadošča tudi odvisnosti X → Y ($F_R$ X → Y).

Množici vseh funkcionalnih odvisnosti v relacijski shemi R, ki so logično implicirane z množico odvisnosti $F_R$, pravimo **zaprtje** dane množice odvisnosti $F_R$ in jo označimo s $F_R^+$.

**Armstrongovi aksiomi**:

- **A1** (refleksivnost funkcionalnih odvisnosti, trivialne odvisnosti)
  - Če $Y \subseteq X$ in $X \subseteq R$, potem $X \rightarrow Y$
- **A2** (razširitev obstoječe funkcionalne odvisnosti)
  - Če $X \rightarrow Y$ in $Z \subseteq R$, potem $XZ \rightarrow YZ$
- **A3** (tranzitivnost funkcionalnih odvisnosti)
  - $\{X \rightarrow Y, Y \rightarrow Z\} \quad X \rightarrow Z$

**Inferenčna pravila**, izpeljana iz Armstrongovih aksiomov:

- **I1** (dekompozicija funkcionalnih odvisnosti v dve novi odvisnosti)
  - Če $X \rightarrow Y$ in $Z \subseteq Y$, potem $X \rightarrow Z$
- **I2** (združitev dveh funkcionalnih odvisnosti)
  - $\{X \rightarrow Y, X \rightarrow Z\} \quad X \rightarrow YZ$
- **I3** (psevdotranzitivnost dveh funkcionalnih odvisnosti)
  - $\{X \rightarrow Y, YW \rightarrow Z\} \quad XW \rightarrow Z$

Množico atributov $X^+$, ki jo imenujemo **zaprtje množice atributov** X z ozirom na odvisnosti F, sestavljajo vsi atributi, ki so funkcionalno odvisni od množice X z ozirom na podano množico funkcionalnih odvisnosti F:
$X^+ \equiv \{A: X \rightarrow A \in F^+\}$ in velja $X \rightarrow X^+$

Množica odvisnosti **F pokriva** množico odvisnosti **E**, če je zaprtje odvisnosti E podmnožica zaprtja odvisnosti F:
F pokriva $E \Leftrightarrow E^+ \subseteq F^+$
Če dve množici funkcionalnih odvisnosti druga drugo pokrivata, sta **ekvivalentni**. V tem primeru velja, da sta njuni zaprtji enaki:
$(F \Leftrightarrow E) \Leftrightarrow (F^+ = E^+)$

**Minimalno pokritje množice odvisnosti** F pravimo tisti minimalni množici, ki pokriva množico odvisnosti F in je hkrati ekvivalentna množici F, označimo jo s $F_{min}$.

- Vsaka odvisnost v $F_{min}$ sme imeti le en atribut na desni strani (kanonična oblika)
- Iz $F_{min}$ ne moremo odstraniti nobene odvisnosti, pri čemer bi ostali množici $F_{min}$ in F še vedno ekvivalentni
- V $F_{min}$ ne moremo zamenjati nobene odvisnosti $X \rightarrow A$ z odvisnostjo $Y \rightarrow A$, kjer bi bil $Y \subset X$, pri čemer bi ostali množici $F_{min}$ in F še vedno ekvivalentni

Če relacijsko shemo dekomponiramo v dve ali več relacij, pri tem pa se informacijska vsebina relacij ohranja, pravimo, da ima dekompozicija **neizgubni stik**. S pomočjo **naravnega stika** relacij $r_1$, $r_2$,... $r_n$ naj bo možno restavrirati relacijo r. Dekompozicija relacijske sheme R v shemi $R_1$ in $R_2$, za kateri velja $R_1 \cup R_2 = R$, ima neizgubni stik, če v relacijski shemi R obstaja vsaj ena od naslednjih odvisnosti:

- $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$
- $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$

Shemi v dekompoziciji $\rho \equiv$ {R1, R2} morata imeti **neprazen presek**, le-ta pa mora biti **ključ** v vsaj eni od obeh shem.

Dekompozicija **naj po možnosti ohranja funkcionalne odvisnosti**. Pri dekompoziciji sheme R z odvisnostmi F v $\rho \equiv \{R_1, R_2,... R_n\}$, je potrebno za vsak $R_i$ izračunati množico odvisnosti $F_i$, ki se projecirajo v posamezno množico atributov $R_i$. Če je unija množic funkcionalnih odvisnosti $F_1, F_2,... F_n$ ekvivalentna množici funkcionalnih odvisnosti F izhodiščne sheme, pomeni, da se funkcionalne odvisnosti pri dekompoziciji ohranjajo.

**Druga normalna oblika** je relikt zgodovine in je svoje čase služila kot vmesni člen pri definiranju 3. normalne oblike. Relacijske sheme, ki so v 1. NO in niso v 2. NO, imajo na levi strani odvisnosti pravo podmnožico ključa, na desni pa neosnovni atribut (atribut, ki ni v ključu).

Relacijska shema se nahaja v **tretji normalni obliki**, če je za vsako v njej veljavno funkcionalno odvisnost $X \rightarrow A$ (upoštevamo le kanonično obliko) izpolnjen vsaj eden od naslednjih pogojev:
- $X \rightarrow A$ je trivialna odvisnost
- X je nadključ sheme R
- A je osnovni atribut

**Postopek dekompozicije** je naslednji:
- izračunamo minimalno pokritje funkcionalnih odvisnosti
- vsaki (kanonični) odvisnosti iz minimalnega pokritja $X \rightarrow A \in F_{min}$ priredimo relacijsko shemo XA, ki jo dodamo v dekompozicijo $\rho$ razen v primeru, če v $\rho$ že obstaja taka shema ali shema, ki XA vključuje kot podmnožico
- ključu relacijske sheme priredimo shemo, ki obsega vse atribute ključa, če le-ta ne nastopa že kot podmnožica v kakšni od shem, ki se nahajajo v $\rho$.

Omenjena dekompozicija **ohranja funkcionalne odvisnosti**, kar je razvidno iz tega, da vsaki odvisnosti priredimo svojo relacijsko shemo, v kateri ta odvisnost velja.

Relacijska shema se nahaja v **Boyce-Coddovi normalni obliki**, če za vsako v njej veljavno fukcionalno odvisnost $X \rightarrow A$ velja vsaj eden od naslednjih pogojev:
- $X \rightarrow A$ je trivialna odvisnost
- X je naključ sheme R

Razlog za BCNO se kaže v spoznanju, da nekatere relacije v 3. NO še vedno kažejo določene **ažurirne anomalije**.

**Dekompozicija v BCNO**:
V $\rho$ poiščemo tako shemo, ki se ne nahaja v BCNO. Posamezno shemo razdelimo v shemi S1 in S2 na osnovi funkcionalne odvisnosti, ki krši pogoje BCNO, pri čemer poiščemo tako odvisnost $X \rightarrow A$ med vsemi odvistnostmi, ki se nahajajo v zaprtju $F_S^+$ omenjene sheme. Shema **S1** je v BCNO, ker obsega atribute XA s ključem X. Shema **S2** obsega vse atribute razen A, ki kot funkcionalno odvisen nastopa le v shemi S1. Delitev shem ponavljamo toliko časa, dokler se vse sheme ne nahajajo v BCNO.

Dekompozicija v BCNO je neizgubna, vendar zahteve po ohranjanju funkcionalnih odvisnosti ne izpolnjuje vedno. V tem primeru imamo 2 možnosti:
- Zadovoljimo se z relacijo v 3. NO in ohranimo morebitne **pomnilniške anomalije**
- Izberemo BCNO, **integritetne omejitve** pa **zakodiramo** v uporabniške programe

Pravimo, da množica atributov **X večvrednostno določa** množico atributov **Y** (X  Y), če v katerikoli relaciji s shemo R za poljubni vrstici *s* in *t* iz te relacije, ki se ujemata v atributih X, obstajata nadaljni dve vrstici *u* in *v* tako, da veljajo naslednje enakosti:

- s[X] = t[X] = u[X] = v[X]
- u[Y] = s[Y] in u[R-X-Y] = t[R-X-Y]
- v[Y] = t[Y] in v[R-X-Y] = s[R-X-Y]

Značilno za te odvisnosti je, da vedno **nastopajo v parih**.

Funkcionalne odvisnosti so poseben primer večvrednostnih, D = F ∪ V je enotna množica odvisnosti, nad katero lahko deifiniramo še 5 Armstrongovih aksiomov:

- **A4** (komplementarnost večvrednostnih odvisnosti)
  - {X  Y}  X  (R-X-Y)
- **A5** (razširitev večvrednostne odvisnosti)
  - Če X  Y in V ⊆ W ⊆ R, potem XW  VY
- **A6** (tranzitivnost večvrednostnih odvisnosti)
  - {X  Y, Y  Z }  X  (Z-Y)
- **A7** (funkcionalna odvisnost poseben primer večvrednostne odvisnosti)
  - X → Y  X  Y
- **A8** (združevanje večvrednostne in funkcionalne odvisnosti)
  - Če X  Y in Z ⊆ Y ⊆ R,  W ⊆ R ter W∩Y={} in obstaja W → Z, potem obstaja tudi X → Z

In še 2 inferenčni pravili, ki sta posledica teh aksiomov:

- **I4** (pravilo o uniji)
  - {X  Y, X  Z }  X  YZ
- **I5** (pravilo o razstavljanju)
  - {X  Y, X  Z }  {X  (Y∩Z), X  (Y-Z), X  (Z-Y)}

Relacijska shema se nahaja v **četrti normalni obliki** z ozirom na množico v njej veljavnih funkcionalnih in večvrednostnih odvisnosti D, če je za vsako v njej veljavno odvisnost X  Y izpolnjena vsaj ena od naslednjih trditev:

- X  Y je trivialna odvisnost
- X je nadključ sheme R

Za neizgubni stik mora veljati:

- $(R_1 \cap R_2)$  $R_1 - R_2$
- $(R_1 \cap R_2)$  $R_2 - R_1$

Dekompozicija v 4. NO je podobna dekompoziciji v BCNO, tako, da tudi ta ne ohranja vedno odvisnosti, rezultat dekompozicije pa je **prav tako odvisen od zaporedja razcepljanja**.

**Stična odvisnost** obstaja, če je možno določeno relacijo dekomponirati v več relacij, ne da bi se pri tem informacijska vsebina spremenila.

Relacijska shema R se nahaja v **peti normalni obliki** z ozirom na množico odvisnosti D, če je za vsako v njej veljavno stično odvisnost $(R_1, R_2,... R_n)$ izpolnjen vsaj eden od naslednjih dveh pogojev:

- $(R_1, R_2,... R_n)$ je trivialna odvisnost
- vsak $R_i$, naveden v stični odvisnosti, je nadključ sheme R

Relacijsko shemo, ki se ne nahaja v 5. NO, dekomponiramo v peto normalno obliko **na osnovi posamezne stične odvisnosti**. Vsaki stični odvisnosti $(R_1, R_2,... R_n)$, ki obstaja v množici odvisnosti D, se prirede relacijske sheme $R_1, R_2,... R_n$. Dekompozicija je potem $\rho=\{R_1, R_2,... R_n\}$.
V principu obstaja v množici odvisnosti D več stičnih odvisnosti, obstaja pa tudi **ena odvisnost, ki vključuje vse ostale** netrivialne odvisnosti. Dekompozicija se izvede po tej slednji odvisnosti, ki da za rezultat podsheme, ki so vse po vrsti v 5. NO, tako da nadaljna dekompozicija ni več potrebna.

## 2.3 KONCEPTUALNO NAČRTOVANJE

Pod konceptualnim načrtovanjem razumemo predstavitev uporabnikovih podatkovnih zahtev s pomočjo konceptualnega oz. semantičnega podatkovnega modela, ki predstavlja osnovo za kreiranje PB in je **univerzalen**. Za semantiko konceptualnega modela je **odgovoren človek** – načrtovalec, ki pa lahko dober model zgradi zgolj ob **tesnem sodelovanju** in interakciji z bodočimi uporabniki PB. Ena izmed pomembnih vlog konceptualnega modela je **opis pomena podatkov** shranjenih v PB (služi kot referenčni dokument). Tudi vse potrebne **spremembe** in dopolnitve se najprej izvedejo na konceptualnem modelu, šele nato se uveljavijo v PB.

**Načrtovalski koraki**:

- Percepcija sveta, ki ga modeliramo s PB
  - najpomembnejša - abstrakcija
- Predstavitev intencionalnega dela percepcije
  - pravila opišemo z uporabo konceptualne sheme
- Preslikava konceptualnega modela v logični model
- Opis logičnega modela z jezikom za opis podatkov
  - kreiranje tabel, indeksov, ključev
- Kreiranje fizične PB in zajem podatkov
  - datoteke na disku, inicialna polnitev s podatki

Do konceptov in objektov se dokopljemo z opazovanjem okolja, ki ga želimo modelirati, s pomočjo procesa, imenovanega abstrakcija. **Abstrakcija** je **mentalni proces**, s pomočjo katerega se pri opazovanju in razmišljanju o svetu osredotočimo le na določene, za nas **pomembne karakteristike**, vse nebistvene pa zanemarimo.

**Vrste abstrakcij**:

- **Klasifikacija**
  - definiranje koncepta kot tipa objektov na osnovi njihove podobnosti oz. skupnih lastnosti

    - te lastnosti služijo za klasificiranje objektov v pripadajoče tipe
- **Ureditev tipov**
    - tip B je podtip tipa A, če so vsi primerki tipa B tudi primerki tipa A
    - urejeno s Hassejevim diagramom
    - Generalizacija
        - prirejanje skupnega nadtipa dvema ali večim obstoječim tipom
    - Specializacija
        - prirejanje podtipov posameznemu tipu
        - običajno temelji na vrednosti neke lastnosti primerkov (spol)
- **Agregacija**
    - posamezni deli, sestavljeni v celoto, predstavljajo novo kvaliteto (računalnik)

Množica primerkov agregiranega tipa je sestavljena iz n-teric, katerih komponente so primerki obstoječih tipov. Agregacija je lahko dvojiška, trojiška ali pa poljubne, n-te stopnje. **Dvojiško agregacijo** si lahko predstavljamo kot **preslikavo**, za opis preslikave se pogosto uporablja kardinalnost.

**Kardinalnost je par (minC, maxC)**. Minimalna kardinalnost minC predstavlja najmanjše število povezav, maksimalna kardinalnost maxC pa največje število povezav, ki jih ima kakšen primerek tipa X ali Y s primerkom tipa A.

Z ozirom na pripadajočo kardinalnost **poimenujemo preslikavo** kot:
- parcialno, če je minC = 0
- totalno, če je minC > 0
- enovrednostno, če je maxC = 1
- večvrednostno, če je maxC > 1

**Lastnosti konceptualnega modela** (jezika za opis okolja):
- **Izraznost**
    - z vrsto različnih konceptov za predstavitev tipov objektov, njihovih lastnosti in povezav
- **Preprostost**
    - enostaven in lahko razumljiv
    - služi tudi za komunikacijo med uporabnikom in načrtovalcem
- **Minimalnost**
    - naj ne obsega sestavljenih konceptov
- **Formalnost**
    - vsak koncept mora imeti precizno, enoumno in dobro definirano interpretacijo oz. pomen
- **Grafična popolnost**
    - grafična predstavitev
- **Berljivost**
    - grafični simboli se morajo jasno razlikovati med seboj

**Osnovni načrtovalski elementi** modela ER (entiteta-razmerje):
- Entitetni tip
- Atribut
- Razmerje med entitetnimi tipi
- Entitetni identifikator
- Ureditev tipov z generalizacijo in specializacijo

**Entitete** so posamezne stvari, ki obstajajo ali mislimo, da obstajajo v svetu, ki ga modeliramo in ki jih lahko ločimo drugo od druge. Za entitete, ki ustrezajo določeni predstavi, pravimo, da pripadajo določenemu **entitetnemu tipu**. Vsak trenutek mu pripada množica entitet tega tipa (**entitetna množica**), ki je časovno spremenljiva.

**Atribut** (a) je preslikava entitetne množice ($E_T$), pripadajoče določenemu tipu (T), v vrednostno množico ali zalogo vrednosti ($V_T$). Atribut je lastnost, ki je tako pomembna, da jo vključimo v model, predstavljen je s krožcem.

**Kardinalnost atributa**:

- totalni atribut (1,n), kjer je $n \geq 1$
- parcialni atribut (0,n), kjer je $n \geq 1$
- enovrednostni atribut (m,1), kjer je $m \in \{0,1\}$
- večvrednostni atribut (m,n), kjer je $m \in \{0,1\}$ ter $n > 1$

Vrste povezav med entitetami se v modelu ER obravnavajo kot **razmerja med entitetnimi tipi**, lahko jih obravnavamo kot agregirani tip. Predstavljamo jih z rombom. Vsaki izmed preslikav je pripisana **kardinalnost**.

Množico atributov, ki so potrebni in zadostni za identifikacijo posamezne entitete, imenujemo **entitetni identifikator**.

Posamezno entiteto, ki pripada **močnemu entitetnemu tipu**, je možno identificirati zgolj z vrednostmi njenih lastnih atributov.

**Entitetni identifikator** močnega entitetnega tipa sestavlja podmnožica njegovih atributov $I \equiv \{a_1,...a_k\}$, ki mora izpolnjevati naslednje tri pogoje:

- $a_1,...a_k$ so vsi po vrsti totalni enovrednostni atributi
- preslikava kartezijskega produkta pripadajočih vrednostnih množic v entitetno množico tipa T: $V_1 \times ... \times V_k \rightarrow E_T$ je totalna ali parcialna enovrednostna funkcija
- ne obstaja prava podmnožica $I' \subset I$, ki bi tudi izpolnjevala drugi pogoj

**Identifikator šibkega entitetnega tipa** je sestavljen ne le iz lastnih atributov, pač pa tudi iz identifikatorjev enega ali več močnih entitetnih tipov, s katerimi je v razmerju in izpolnjuje naslednje pogoje:

- $a_1,...a_k$ so totalni enovrednostni atributi šibkega entitetnega tipa
- preslikava kartezijskega produkta identifikacijskim atributom pripadajočih vrednostnih množic in močnim entitetnim tipom pripadajočih entitetnih množic v entitetno množico tipa T: $V_1 x...x V_k x E_{T1} x...x E_{Tn} \rightarrow E_T$ je totalna ali parcialna enovrednostna funkcija
- ne obstaja prava podmnožica $I' \subset I$, ki bi tudi izpolnjevala drugi pogoj

**Vrste pokrivanja** z generalizacijo oz. specializacijo

- **totalno in ekskluzivno pokrivanje** (t,e)
    - $E_B \cup E_C = E_A$ in $E_B \cap E_C = \{\}$
- **totalno in prekrivno pokrivanje** (t,p)
    - $E_B \cup E_C = E_A$ in $E_B \cap E_C \neq \{\}$
- **delno in ekskluzivno pokrivanje** (d,e)
    - $E_B \cup E_C \subset E_A$ in $E_B \cap E_C = \{\}$
- **delno in prekrivno pokrivanje** (d,p)
    - $E_B \cup E_C \subset E_A$ in $E_B \cap E_C \neq \{\}$

Posamezna **transformacija preslika vhodno shemo v izhodno**, ki se nato uporabi kot vhodna shema pri naslednji transformaciji s temi lastnostmi:

- skladno s koncepti se smiselno preslikajo tudi imena konceptov
- podedujejo se vse logične povezave (razmerja, atribute, hierarhijo)
- semantična vsebina se ohranja
- izhodna shema ima enak pomen, a opisuje modelirani svet podrobneje

**Elementarne transformacije z vrha navzdol**:

- **V0**: kreira v začetni shemi ent. tip, ki služi kot izhodišče za modeliranje
- **V1**: preslika entitetni tip v eden, dva ali več med seboj z razmerjem povezanih entitetnih tipov
- **V2**: preslika entitetni tip v specializacijo tipa
- **V3**: preslika razmerje med entitetnima tipoma v dve novi razmerji
- **V4**: preslika razmerje v dve razmerji, med njima pa nastopa novo kreirani entitetni tip
- **V5**: izvede eksplicitno predstavitev atributov ent. tipa ali razmerja in pripiše kardinalnosti
- **V6**: preslika posamezen atribut v dva ali več atributov

**Elementarne transformacije od spodaj navzgor**:

- **S0**: izvede kreiranje atributov
- **S1**: pripiše množici atributov pripadajoči entitetni tip ali pa razmerje in pripiše kardinalnosti
- **S2**: poveže dva entitetna tipa z medsebojnim razmerjem in pripiše kardinalnosti
- **S3**: izvede generalizacijo že obstoječih entitetnih tipov

Strategija načrtovanja ***z vrha navzdol*** temelji na postopno vse bolj natančnem modeliranju sveta. Uporabljajo se transformacije z vrha navzdol, izhodiščna shema je entitetni tip, ki predstavlja shemo celotnega sveta.

Izhodiščno shemo strategije ***od spodaj navzgor*** tvori množica atributov, ki jih nato smiselno grupiramo, kreiramo entitetne tipe, ki jim skupine atributov pripadajo in povežemo z razmerji. Pri tem uporabljamo transformacije od spodaj navzgor.

Pri strategiji ***od znotraj navzven*** tvori izhodiščno shemo eden izmed najpomembnejših entitetnih tipov, ki mu nato po korakih dodajamo nove tipe in razmerja med njimi. Pri tem se uporabljata obe skupini elementarnih transformacij.

Strategija ***po delih*** je med najbolj uporabnimi (obsežnejše načrtovanje), saj kreiramo najprej okvirno shemo z najpomembnejšimi ent. tipi in razmerji, nato pa se lotimo načrtovanja posameznih delov sheme, katerih jedra so prav ti tipi. Pri tem je možno izvajati načrtovanje zaporedno.

# 2.4 NAČRTOVANJE ZUNANJIH SHEM

**Zunanja shema** ali uporabniški pogled je pogled na podatkovni del, kot ga vidijo posamezni uporabniki oz. skupine uporabnikov.

**Načini specifikacije uporabniških zahtev**:
- opis v naravnem jeziku
  - najobičajnejši
- obrazci
  - načrtovanje temelji na okencih, njihovih naslovih in komentarjih
- datotečni formati
  - načrtovanje glede na opise datotečnih zapisov

**Načrtovanje na osnovi zapisanih zahtev**:
- **Analiza zahtev**
  - odkrivanje morebitnih neskladji
  - izogibamo se sinonimom in homonimom
  - dejstva zapišemo na nedvoumen način
  - standardna oblika stavkov
  - pojmovnik izrazov
- **Kreiranje začetne sheme**
  - stavki, ki govore o sorodni tematiki, se grupirajo v odstavke
  - ključne besede – začetni model
- **Načrtovanje**
  - strategiji *od znotraj navzven* in *po delih*

**Sestava obrazcev**:
- **Vrednostni del**
  - vpisujemo podatke – konkretne vrednosti atributov
- **Pomenski del**
  - ime polja – pričakovana vsebina polja
- **Opisni del**
  - če obstajajo določena pravila za vpisovanje v polja
- **Certifikacijski del**
  - datiranje in avtoriziranje – podpisnik zagotavlja resnične vrednosti

**Postopek načrtovanja obrazcev**:
- določi področja in podpodročja na obrazcu
- za vsako področje / podpodročje identificiraj pomenski, vrednostni in opisni del ter ga predstavi z enim ali več ent. tipi
- zgradi okvirno shemo
- za vsako področje izdelaj področne sheme
- s pomočjo okvirne sheme poveži vse področne sheme med seboj

Pri **načrtovanju na osnovi datotečnih zapisov** je potrebno parcialne podatkovne modele, ki so implicitno vključeni v strukture datotek, integrirati v integralen podatkovni model družbe. Uvedemo **centralizirano upravljanje** podatkov, ki zmanjšujejo možnost napačnih odločitev zaradi različnih vrednosti enakih podatkov.

## 2.5 ZDRUŽEVANJE ZUNANJIH SHEM

Pri združevanju zunanjih shem lahko naletimo na težave, saj se posamezne sheme verjetno **ne bodo dale** povsem **mehanično sestaviti** v globalno shemo, saj se v presečnih delih shem lahko pojavijo neskladja med podatkovnimi modeli, ki so posledica načrtovanja **različnih načrtovalcev** (vsak ima svoj pogled na svet).

**Neskladja med shemami**:

- **Različne perspektive**
  - pogledi načrtovalcev so različni
- **Nezdružljive integritetne omejitve**
  - različne kardinalnosti atributov in razmerji
  - vrsta pokrivanja pri generalizacijah
- **Imenska neskladja**
  - homonimi (enako poimenovanje, različen pomen)
  - sinonimi (različno poimenovanje, enak pomen)
  - entitetni tipi morajo biti različno poimenovani
  - razmerja so lahko enako poimenovana, če ne povezujejo istih entitetnih tipov
  - atributi, ki pripadajo različnim ent. tipom, imajo lahko ista imena

Združevanja shem se je potrebno lotiti **sistematično**, v vsakem koraku združevanja **združimo** praviloma le po **dve shemi na način**:

- **preko skupnega entitetnega tipa**
  - atributi, ki jih ta tip prevzame, so unija atributov obeh tipov
- **z vpeljavo hierarhične povezave med shemama**
- **z vpeljavo razmerja med shemama**
  - ki mora v modeliranem svetu zares obstajati

## 2.6 KVALITETA KONCEPTUALNE SHEME

**Karakteristike za oceno kvalitete konceptualne sheme:**

- **Popolnost**
  - predstavlja vse pomembne lastnosti modeliranega sveta
  - preverimo za vsako uporabniško zahtevo
- **Pravilnost**
  - pravilni sintaktična in semantična komponenta
- **Minimalnost**
  - vsak aspekt uporabniških zahtev se pojavi le enkrat
  - izvedeni (redundantni) atributi le, ko so stroški za njegov izračun preveliki
- **Izraznost**
  - shemo je možno doumeti samo po sebi
  - namesto atributa "tip" razmerja uvedemo raje 2 novi razmerji
- **Berljivost**
  - ent. tipi in razmerja v približno enaki velikosti
  - povezave potekajo horizontalno in vertikalno, brez križanja
  - povdarjene simetrične lastnosti
  - hierarhije narisane vertikalno (zgoraj nadtip, spodaj podtip)

# 2.7 MODEL ER IN RELACIJSKI MODEL

**Elementi** entitetnega modela, ki jih je **potrebno preslikati** v relacijski podatkovni model, so naslednji:

- Močni entitetni tipi z atributi
- Šibki entitetni tipi z atributi
- Razmerja z atributi
- Hierarhija entitetnih tipov

Preslikave naj bojo izvedene tako, da **rezultirajo v normaliziranih relacijskih shemah**. Pogoj za to je, da so vsakemu entitetnemu tipu pripisani le njegovi **lastni atributi**.

**Močni entitetni tipi z atributi**:

- shema, ki vsebuje vse enovrednostne atribute + identifikator
- po ena shema za vsak večvrednostni atribut + identifikator, ključ sheme tvorijo vsi atributi skupaj

**Razmerje med močnimi entitetnimi tipi**:

- relacijska shema, ki vsebuje vse atribute identifikatorjev ent. tipov in vse enovrednostne atribute razmerja
    - o če so vse kardinalnosti razmerja vrste (0/1,n), tvorijo ključ sheme vsi atributi vseh identifikatorjev ent. tipov
    - o če je ena izmed kardinalnosti razmerja (0/1,1), tvorijo ključ sheme le atributi pripadajočega ent. tipa
    - o če je v razmerju več kardinalnosti tipa (0/1,1), potem ima shema več ključev
- po ena relacijska shema za vsak večvrednostni atribut razmerja, ki vsebuje ta večvrednostni atribut in atribute ključa rel. sheme
    - o ključ rel. sheme tvorijo vsi atributi v shemi skupaj

**Šibki entitetni tip**:

- shema, ki vsebuje vse enovrednostne atribute šibkega entitetnega tipa in entitetni identifikator šibkega ent. tipa, katerega sestavni del je identifikator enega ali več močnih ent. tipov
    - o ključ sheme tvorijo atributi identifikatorja šibkega ent. tipa
- po ena shema za vsak večvrednostni atribut šibkega ent. tipa, ki vsebuje ta atribut in atribute identifikatorja
    - o ključ sheme tvorijo vsi atributi sheme

**Hierarhije tipov** se ne da direktno preslikati v relacijske sheme, pač pa jo je potrebno zamenjati z drugimi koncepti entitetnega modela.

- Tipa lahko združimo v enega
    - o atribut podtipa postane parcialni atribut (0,n) – lahko NULL
- Shemo naredimo za vsakega posebej
    - o ohranja totalne atribute
    - o shema podtipa podeduje atribute identifikatorja nadtipa, ima manjšo moč (manj vrstic)