2. Põhimõisted

Meid ümbritsev maailm on tunnetatav meie meelte ja neid abistavate sensorite abil, kuid kuna reaalne maailm on lõpmata mitmekesine ja keeruline, siis igasugune tunnetus on alati eesmärgipärane lihtsustus;

- seda lihtsustust tehes liitub 'vahetule tunnetusele' alati vähemalt kaks asja:
  1) senine kogemus ja
  2) eesmärk;
- niisugust lihtsustust nimetatakse mudeliks (reaalsest maailmast), tema loomist ja kasutamist kui keerukat protsessi aga modelleerimiseks, mistarvis teadus on oma arengu käigus välja töötanud komplitseeritud metoodikad;
- modelleerimise tulemusel saadud järeldused me kanname üle reaalsele maailmale, selleks et saavutada oma eesmärki.

Informaatikas võiks aluseks olla järgmine neljaosaline tunnetuse mudel:

Reaalne maailm ➜ **Reaalsusmudel**
↓
Infomudel (kontseptuaalmudel)
↓
Andmemudel
↓
Esitusmudel

2.1 Reaalsusmudel, nähtused ja nähtusteklassid

Reaalsusmudel on reaalse maailma lihtsustatud kujutis:

- keskseks mõisteks on **nähtus** (*phenomen*);
- reaalne maailm on ruumiliselt, ajaliselt ja temaatiliselt pidev, kuid jagatav unikaalseteks (kordumatuteks) üksiknähtusteks (fenomenideks), näiteks Tartu Toomemägi;
- tunnetuse käigus ühendatakse üksiknähtused sarnasuse alusel nähtuste kogumitesse – **nähtuseklassidesse** – ja kasutatakse neid, näiteks Tartu Toomemägi võib kuuluda klassidesse 'linnamäed', 'künkad', 'pargid' vms.

2.2 Infomudel, olemid, atribuudid ja seosed

Infomudel kui nähtuseklasside ja nendevaheliste seoste käsitlemine kontseptuaalsel tasemel, seetõttu nimetatakse seda sageli ka **kontseptuaalmudeliks**.

- keskseks mõisteks on **olem** (*entity*, mõnel puhul ka *feature*), näiteks 'küngas';
- tavaliselt saadakse ühe nähtusteklassi alusel üks olemiklass, näiteks nähtusteklassi 'künkad' jaoks võetakse infomudelis kasutusele olemiklass 'küngas'.

- siin pole olulised mitte üksiknähtuste konkreetsed omadused, vaid see, milliseid **omadusi** või **suhteid** peetakse ühe või teise nähtustekogumi jaoks olulisteks; näiteks nähtusteklassi 'küngas' üks oluline omadus on 'suhteline kõrgus', nähtusteklassi 'park' puhul 'puuliikide arv';
- kui me tahame infomudeli puhul rõhutada üht konkreetset küngast tema omaduste alusel (näiteks Tartu Toomemäge tema suhtelise kõrguse alusel), siis räägime **olemi esindajast** (*entity instance*); tavaliselt see pakub infomudeli tasandil vähe huvi;
- küngastest kui nähtuseklassist infomudelit koostades võivad meil mõnede küngaste juures olla olulised ühed omadused ja seosed, teiste puhul teised; näiteks ühel juhul suhteline kõrgus, teisel juhul genees (nähtuseklassidena voored, moreenkünkad, terrikoonid jne.). Sel juhul räägime infomudelis **olemi tüüpidest** (*entity types*);
- olemi olulisi omadusi nimetatakse **atribuutideks**, nende seas on eriline tähtsus nn. **võtmeatribuudil** e. identifikaatoril, mis on igal olemil unikaalne. Võtmeid liigitatakse omakorda (näit. liht- ja liitvõti), aga see läheks meie jaoks liiga põhjalikuks.

Mõnikord määratletakse infomudelis ära ka atribuutide väärtusvaru (*domain*), kuid meie peame õigeks vaadelda seda andmemudeli juures ja rääkida tunnuste väärtusvarudest.

- kui on tegemist inimesele kõige tavapärasema infomudeliga, siis olemiklassile 'KÜNKAD' vastaks tühi tabel, millel oleks niipalju veerge, kuipalju olemil 'küngas' on (olulisi) omadusi (nn. **atribuute**) ja niipalju ridu, kuipalju on vaatlusaluseid künkaid (**olemeid**); kas tabeli lahtrid on tühjad või täidetud (ja millega), pole infomudeli tasemel oluline;
- infomudel peab olema reaalsusmudeli eesmärgipärane lihtsustus, mis ei sõltu (edaspidi) kasutatavatest andmetöötlusvahenditest.
- nii nagu reaalsusmudelis ei eksisteeri nähtused omaette vaid vastastikustes seostes, peab ka infomudel kajastama olulisi seoseid (*relationships*), mis esitatakse nn. **olemiseosmudelina** (**ER-mudel**: *entity-relationship model*).
- seoseid veidi lihtsustatult käsitledes võime eristada
  - seose **tüübi**:
    - **1 : 1**
    - **1 : n**
    - **m : n**
  - seose **liigi**:
    - **ühendamisseos** (*aggregation*) - tavaliselt esitub verbina "koosneb" (auto koosneb kerest, mootorist, ratastest...);

- **üldistamisseos** (*generalization*) määrab 'alamklass-ülemklass' seosed: 'künkad', 'orud' ja 'tasandikud' saab üldistada olemiklassi 'pinnavormid'; orud on teatud liiki pinnavormid;
- **assotsieeriv seos** (*association*) on tavaliselt ühesuunaline füüsiline või kontseptuaalne side olemite vahel: "auto kasutab kütust", "juht juhib autot", "tee lõikub jõega".

2.3 Andmemudel, tunnused, eksemplarid ja väärtused

Andmemudel on infomudeli konkretiseerimine kasutatavate andmete tarbeks (näiteks tabeli puhul veeru laiuste määramine) ja vastavalt reeglitele, mida seab kasutatav tarkvara;

- kesksek mõisteks on **objekt**, millel esinevad **tunnused**;
- kui andmemudel esitada tabelina, siis selle reale vastaks objekt ja veerule - tunnus (analoog andmetöötluse 'objekt-tunnus' maatriksile); iga konkreetne rida kirjeldaks üht **eksemplari** ja tabeli lahter - andmeelemendi **väärtust;**

2.4 Esitusmudel, päringud ja kasutajavaated

Esitusmudel on reeglid andmete esitamiseks tarbijale sobival kujul, näiteks asukohaandmete esitamine postiaadressile seatud nõudmistele vastavalt.

- kesksetek mõisteteks on **esitusmall** (*template*), mille alusel esitatakse olemi atribuudid vastavalt **kasutajavaatele**, ja **sümbolid**, kui tähestik tunnuste kodeerimiseks kasutaja jaoks sobivale kujule.

2.5 Andmebaas

Kõige lihtsam viis on andmeid hoida mingite säilitus-ühikute kaupa: raamatukogus raamatute ja riiulite kaupa, arvutis failide ja kaustade kaupa. Failide kogumit kutsutakse ka **teegiks** (*library*).

Termin "andmebaas" (*data base, Datenbank*) ilmus käibesse 20. sajandi 60-aastate keskel, kui arvutis hakati programsete vahenditega arendama andmete integreeritud kasutust, seega hakati looma tarkvara, mis hõlbustas infomudeli alusel loodud teatud kindlat tüüpi andmemudelite haldamist. Nüüdseks on "andmebaas" muutnud üldkeele sõnaks ja teda kasutatakse sageli lihtsalt 'andmehulga', 'andmemassiivi', 'andmetabeli' jms. tähenduses.

Klassikalise definitsiooni järgi:

Andmebaas on korrastatud ja loogiliselt seostatud andmete hulk, mida säilitatakse andmekandjal minimaalse liiasusega ning mida võivad üksteisest sõltumatult kasutada mitu rakendusprogrammi

Seega andmebaasi kontseptsioon:

- realiseerib andmemudeli
- võimaldab ühiskasutust
- minimeerib andmete liiasuse
- eristab andmete füüsilise säilitamise andmekandjal nende kasutamisest
- võimaldab andmete muutmist ilma süsteemi teisi komponente muutmata.

Andmebaasi kui terviku mõistmisel on peatähtis andmebaasi kontseptuaalmudel (infomudel). Selle alusel on loodud konkreetse tarkvara võimalusi kasutav andmemudel kui **andmebaasi loogiline struktuur** (nimetatakse ka välisvaateks), mis tavaliselt erineb andmebaasi **füüsilisest struktuurist** (nimetatakse ka sisevaateks). Viimane on nüüdisaegsetes personaalarvutites üha rohkem kasutaja eest varjatud ja "arvuti siseasi".

Erinevatel kasutajatel on erinevad vaated andmebaasile [**kile**]. Tegelikkuses võib vaadete mitmekesisus olla hoopis suurem kui toodud põhimõttelisel skeemil.

Näiteks panga andmebaasi "Kasutajaks" võib olla:

- tellerile ekraanivormi kuvav programm,
- arve trükimallile andmeid edastav programm,
- meeldetuletuskirju koostav programm,
- panga andmebaasi rakendusprogrammist,
- Internetipanga klient,
- Maksuameti kontroll,
- audiitor,
- jne.

2.6 Andmebaasihaldur (*DBMS*)

Eespoolkirjeldatud nõudeid rahuldavat tarkvarakompleksi, mis võimaldab andmebaase luua ja kasutada (andmeid ohjata) nimetatakse eesti keeles (ISO standardi järgi) **andmebaasi-halduriks**. Varasemad, samuti kirjanduses kasutatavad sõnad on andmebaasisüsteem ja andmeohjesüsteem.

2.7 Andmemudelite realiseerimisviisid

Spetsialistid jaotavad andmemudelid tavapäraselt 4 klassi: hierarhiline mudel, võrkmudel, relatsiooniline mudel ja muud tüüpi mudelid (Isotamm, 1996:48). Aktuaalset praktikat arvestades võtame me alljärgnevas vaatluse alla viis realiseerimisviisi – lisaks eeltoodule veel lameda mudeli, mille kohta (*op.cit.*) 'andmemudeli mõiste ei laiene' ja muude mudelite seast peatume objektorienteeritud mudeli aluseks oleval kontseptsioonil, kuivõrd see on oluline igapäevaselt kasutatava tarkvara (*Micro-*

soft) töö mõistmisel.

2.7.1 *Lame*

Lameda andmebaasi mõistet kasutatakse mõnikord ühe-tabeli-süsteemi puhul (näiteks Exceli tööleht või selle osa). Sel juhul polegi andmemudelit vaja, sest tegemist on vaid ühe olemiga (seoseid ei saa kajastada). Näide: elektrooniline telefoniraamat.

Kõige parem on lamedat andmebaasi ette kujutada andme-**tabelina** (millele füüsiliselt võib vastata **fail**). Tabeli ridu nimetatakse **kirjeteks** (*records*) ja tabeli veerge **väljadeks** (*fields*), mis kirjeldatakse tavaliselt (tabeli **päises**) järgmiste komponentide abil:

- nimi;
- tüüp (näit. *integer, real, character, array, date, picture*);
- kitsendused (näit. minimaal- ja maksimaalväärtus, väärtusvaru, maksimaalne sümbolite arv jne.).

2.7.2 *Hierarhiline*

Maailmas on hierarhilised suhted väga levinud ja seetõttu sobib niisugune mudel nende kirjeldamiseks hästi.

Näited:

- Eesti jaguneb maakondadeks, need valdadeks, vallad küladeks, külad taludeks;
- raamatukogu temaatilise kataloogi UDK jaotus (näiteks märksõnaloend ei ole hierarhiline)

Hierarhiline andmemudel väljendab infoobjektide üks-mitu-suhteid (1:M suhteid) ja talle vastav abstraktne andmestruktuur on **puu** (mille tipud on kirjed).

Hierarhiline andmemudel ei toeta M:N suhteid.

2.7.3 *Võrkmudel*

Võrkmudel põhineb **graafil** (ka puu on teatud omadustega graaf: orienteeritud atsükliline graaf).

Võrkmudelis võivad loogiline ja füüsiline struktuur olla realiseeritud väga erinevalt. Sellised süsteemid kasutavad keerulist **viitade** ja juurdepääsutabelite süsteemi. Andmebaasid on realiseeritud suurtel arvutitel ja keerulised.

2.7.4 *Relatsiooniline andmemudel (RDB)*

Koosneb tabelitest ehk relatsioonidest; tabeli veergudest eristatakse võtit: andmevälja, mille väärtus on unikaalne iga kirje jaoks ja mis seetõttu identifitseerib kirjeid.

Lihtsustatult võime öelda, et omavahel seostatud andme-tabelid moodustavadki RDB:

- erinevalt lamedatest mudelist on mitu seostatud tabelit;

- erinevalt hierarhilisest ja võrkmudelist on seosed kaudsed (tabelid ei sisalda otseselt infot teiste tabelite kohta) ja seetõttu paindlikud.

Tabelite ühendamine (*relational join*) toimub ühiste veergude alusel ja tabelite tükeldamine (*normalization*) loob andmete liiasust [**kiled**].

RDB on intuitiivselt inimesele hästi arusaadav (seepärast lihtrakendustes väga levinud), kuid tal on rida puudusi, eriti just arvutispetsialistide poole pealt vaadatuna – ta ei ole "arvutipärane". Seetõttu mitmesugustes arvutirakendustes endis kasutatakse hoopis teistsugust, nn. **objektorienteeritud lähenemist** (näiteks meie poolt kasutatava *MS Accessi* andmemudeli loogiline struktuur järgib küll relatsioonilist andmemudelit, kuid füüsiline struktuur ja tarkvara enda tarbijaliides põhineb objektorienteeritusele.

2.7.5 *Objektorienteeritud (OO) lähenemisviis*

Relatsiooniline mudel lähtub suurel määral klassifitseerivast paradigmast, samal ajal kui teadusliku tunnetuse põhiparadigmaks on kaasajal **süsteemkäsitlus**. Relatsioonilist andmemudelit realiseeriva andmebaasihalduri programne realiseerimine läheb real juhtudel küllaltki keeruliseks. Näiteks kollektiivse masinprojekteerimise juures ei rahulda ta absoluutselt. Kui mõnd kaasaegset *Windowsi* rakendust hakkaksime püüdma rakendada näiteks DOS-i tasemel, siis näeme, et asja keerukus kipub ületama vahendite võimalusi. Nii tõstsid uued vahendid ja kasvanud võimalused esile ka uue(d) vaate(d). Näit. sündmus-orienteeritus *Mac*-is (*AppleEvents*) kui hiirele põhinev arvutikasutus.

Hetkel arvutimaailmas valitsev on **objektorienteeritud paradigma**, mis võimaldab ühendada senised eraldiolnud andmekirjelduskeele ja andmeohjekeele ning lisada andmebaasidele ka **teadmised** (teadmiste baasid- *knowledge bases*).

OO võiks lühidalt kokku võtta kui:

- objektid süsteemi elementidena
- teadete saatmine toimingutena objektidega
- klassid objektide generaliseerimismallidena
- omaduste pärimine klasside kirjeldamise mehhanismina

objektorienteeritus = objektid + klassid + pärimine

Kui vaatame lähemalt, siis näeme, et OO kui mõtlemisviis rakendus juba ka meie infomudeli koostamises:

- mingi süsteemi olemust ja omadusi võib kirjeldada tema komponentide kui **objektide** (pange tähele, siin ei kasutata mõistet 'element')olemuse ja omaduste kaudu ja seoste kaudu;
- kogu süsteemi võib vaadelda samuti ühena objektidest mingis

suuremas süsteemis;

- objekti saab määratleda lõpliku hulga **omaduste** (*properties*), tema **olekuga** (*state*) ja tegevustega (*actions*).
  näiteks: auto (omadused: mudel, vanus, nimi, uste arv, mootori kubatuur,...; seisund: ühtlane liikumine kiirusega 60 km/t; tegevused: hakata liikuma edasi, hakata liikuma tagasi, peatuma, pöörama vasakule, pöörama paremale).
- objektid moodustavad klasse(see klassidevaheline suhe on **agregatsioon** - koosnemine):
  (näiteks autode klass koosneb sõiduautodest, veoautodest ja vormelitest)
  - iga objekt kuulub mingisse objektiklassi;
  - oma klassi (üksik)objektid on "*oma liigi isendid*";
  - objektide määratlemisel kasutatakse teisi objekte:
    - emaobjekt ja tütarobjektid (mitmel tasemel) vastavalt sellele, kuidas omadused on **päritud** (*inheritance*). Pärimistehnika on suureks abiks uute klasside defineerimisel (klass - alamklass, originaalobjekti omaduste laiendamine)
    - muudatused superklassis mõjuvad automaatselt alamklassidele, kaob ära vajadus eri versioonide ja nende ühitamise järele;
    - pärimine võib olla ühene ja mitmene (viimasel juhul peavad olema vahendid võimalike konfliktide lahendamiseks);
- objektid suhtlevad omavahel **teadete** (*request*) vahendusel. Teade sisaldab kaht osa: teadet vastuvõtva objekti nime ja operatsiooni nime, mida vastuvõtja peab rakendama. Seega võib sama operatsiooni ('joonista') edastada erinevatele objektidele ('ring', 'ruut', 'romb'). Või teadet 'ehita' – autotehasele ja lennukitehasele. Vastuvõttev objekt on ise operatsiooni täitja (sisaldades meetodeid operatsiooni täitmiseks). See OO eripära tagab operatsioonide polümorfismi (sama teade eri objektidele tekitab erinevaid aktsioone) ja võimaldab objektide kihistumist (s.o. nende käitumise inkapsuleerimist *encapsulation*) - on võimalik defineerida kindlaid operatsioone objektidel; objekti olekuga ei saa manipuleerida otse, vaid ainult temaga seotud operatsioone rakendades, näiteks autoga ei saa lennata.;

Objektorienteerituse eelkäijaks loetakse 60-ndate lõpus rajatud programmeerimiskeelt *SIMULA*, kus formuleeriti mõisted **objekt** ja **klass**.

Esimieseks reaalseks OO süsteemiks oli "Xeroxi" Palo Alto Uurimiskeskuses realiseeritud *Smalltalk-80*. Nüüdsed programmeerimiskeskkonnad on tavaliselt OO (näiteks *VisualBasic*). Kuigi juba 90-aastate algul tekkisid ka esimesed OO DBMS-id, pole nad siiani laiemat kõlapinda leidnud, pigem vastupidi.

Tulevik: püüeldakse sinnapoole, et võimalduks ühitada andmete/teadmiste/objektide/informatsiooni ohjamine, ja et see oleks seotud teiste tehnoloogiatega (kasutajakeskkonnad, tehisaru, jaotatud arvutused).

2.8 Andmete järjestamine: sortimine ja indekseerimine

Andmete kiire ja efektiivne kättesaadavus muutub seda olulisemaks, mida suuremaks läheb andmebaas. GIS-ides on see üks olulisemaid küsimusi. Selleks peavad kirjed olema efektiivsel viisil järjestatud. Vahel tehakse vahet füüsilise järjestamise e. **sortimise** ja virtuaalse järjestamise e. **indekseerimise** vahel.

Tabeli korrastamise viisi järgi eristatakse järgmisi tabelitüüpe:

- korrastamata
  sellisest *n*-kirjelisest tabelist tuleb mingi *k*-nda kirje leidmiseks teha keskmiselt (n+1)/2 otsimissammu ja sellisest tabelist kõikide kirjete leidmiseks kindlas järjekorras tuleb teha n(n+1)/2 sammu,
  seega **kiirushinnang** on $O(n^2)$
- järjestatud tabel
  saame kasutada **kahendotsingut**, mille kiirushinnang on $O(\log_2 n)$, ning järjestamise kiirushinnang on $O(c \times n \times \log_2 n)$, kus c sõltub kasutatavast meetodist ja heade meetodite puhul on $c \leq 3$
  näide:
  $n=10^6$, $n^2=10^{12}$, $\log_2 n=20$, $c \times n \times \log_2 n=6 \times 10^7$
- otseadresseeritav tabel
  otseadresseeritava tabeli saame moodustada siis, kui iga *i*-nda kirje jaoks leidub mingi funktsioon *f*, mis annab võtme k_i jaoks oma väärtuseks kirje aadressi tabelis:
  $$a_i = f(k_i),$$
  kusjuures $a_i \neq a_j$ kui $i \neq j$.
  Niisuguse funktsiooni leidmine pole kahjuks sageli võimalik
- paisktabel
  Eelmist nõuet nõrgendatakse sellevõrra, et lubatakse mõnikord $a_i=a_j$, kuigi $i \neq j$. Niisugust situatsiooni nimetatakse **põrkeks**. Sellisel juhul lisatakse täiendav funktsioon.
  Hea kiire andmete salvestamisel. Põrgete vähendamiseks võetakse *n*-kirjelise tabeli salvestamiseks näiteks 1,5 korda rohkem ruumi ja paisatakse kirjed juhuslikult.

2.8.1 *Kirjete dünaamiline järjestus*

Sel juhul toimub kirjete järjestamine nende laekumise käigus.

- sortimine kasutussageduse alusel

  Panema kasutatud kirje tabeli algusesse. Kui kirjete kasutamise sagedus on küllalt erinev ja tabelit kasutatakse küllalt pikka aega, siis järjestuvad kirjed kasutussageduse alusel ja efektiivne on otsida kirjeid järjest.

Tabeli korrastamine ei tähenda tingimata tema kirjete füüsilist ümbertõstmist, vaid seisneb sageli mingit korrastatud juurdepääsu tagava pealisehituse (indeksite) konstrueerimises. Tänu sellele võib olla ühele tabelile "peale ehitatud" mitu juurdepääsuvarianti.

Indekseerimine kui andmete virtuaalne järjestamine on eriti oluline töös suurte andmebaasidega (näiteks FoxPro-s nn. *Rushmore'*i tehnoloogia, mis miljonite kirjete korral kiirendab andmete kättesaadavust tuhandeid kordi).

2.9 Metaandmed

Andmed andmete kohta, näiteks mitmesugused kataloogid, aitavad üles leida ja/või tõlgendada vajalikke andmeid (kui näiteks raamatukogus poleks katalooge, mis siis saaks)

- peavad olema avalikud
- peaksid vastama teatud standarditele (näit. bibliograafilised kirjed)
- peaksid sisaldama infot andmete täpsuse, saamisviiside ja -meetodite, loojate/koostajate, aktualiseerimisaegade jms. kohta
- umbes 1/10 andmebaasi hoolduskuludest läheb metaandmete hooldusele.