Loeng13. Hajussüsteemides kasutatavad mõisted, mõningad tegevusjuhised hajussüsteemide projekteerimisel, probleemid projekteerimisel. Alateema - hajussüsteemi staatilise struktuuri (andmete jaotuse) projekteerimine.

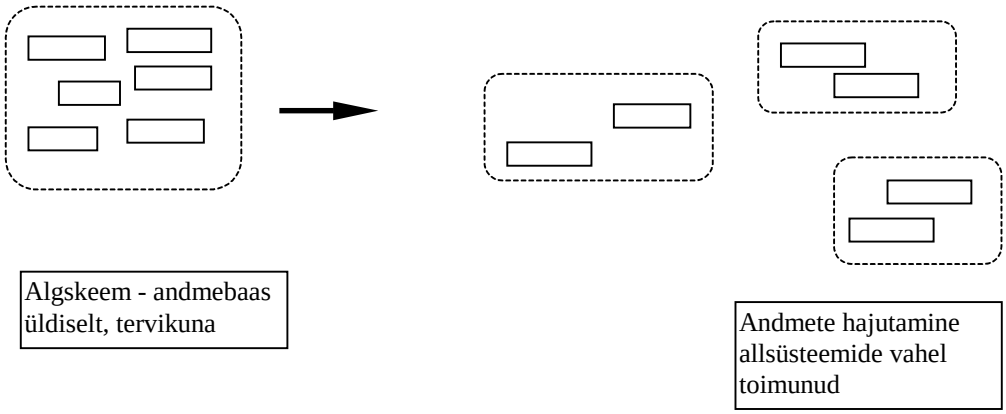
Klient/server süsteem on hajussüsteem. Arvutivõrgule ülesehitatud hajutatud andmebaasidega süsteem on hajussüsteem. Andmebaas, millele kasutajad saavad ligi, kasutades Internetti, on samuti mingis mõttes hajussüsteem (kuigi mitte andmete paiknemise mõttes) hajussüsteem.

Täiemahulise hajussüsteemi tunnuseks on see, et nii rakendused (kasurajaliidesed, kasutajaprogrammid) kui ka andmed paiknevad hajutatult.

Hajussüsteemidel vajab lahendamist kaks põhiprobleemi:

- Andmete staatililine struktuur - kuidas on andmed paigutatud süsteemis laiali , millised andmed kus asuvad.
- Andmete vahetamise, andmeülekandeid realiseerivate protseduuride (inglise keeles võib kohata ka väljendeid *services, transactions, protocols*)ülesehitus. Kui süsteem on ülesehitatud, kasutades SQL andmebaase, realiseeritakse sellised protseduurid serveri SQL protseduuridena, mis on vastavate transaktsioonide koosseisus. SQL baaside puhul tähendab see transaktsioonide modellerimist ja realiseerimist SQL-is.

Andmete staatiline struktuur - selle projekteerimisel oleks mõttekas alustada süsteemi andmemudeli ehitamist nii, et alguses ei arvestata hajussüsteemilisi omadusi, st. süsteemi andmemudel projekteeritakse nii, nagu paikneksid kõik andmed ühel serveril. (inglise keele nn. *monolithic schema*)

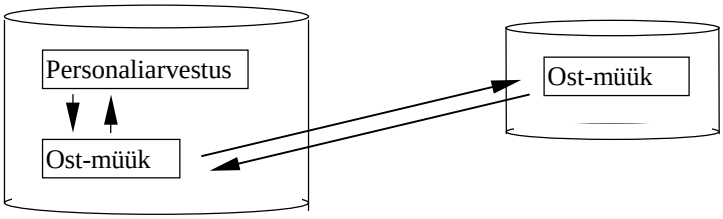


Joonis 1.

Pärast üldise, terviliku andmebaasi skeemi väljatöötamist võib ette võtte andmete hajutamise analüüsi ja paiknemise projekteerimise. Sellist tegevusjuhist saab järgida muidugi kõige enam uute süsteemide projekteerimisel.

Hajussüsteemide projekteerimisel tuleb eristada ka sisulisi andmete allsüsteeme (n. “tööarvestus”, “personaliarvestus”, “ost-müük”) ja neid andmete allsüsteeme , mis paiknevad ruumis hajusalt.

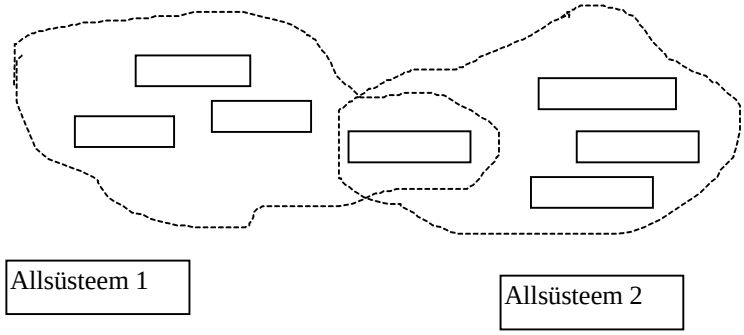
- Mitme sisulises (semantika) mõttes allsüsteemi (“ost-müük”, “personaliarvestus”) andmed võivad paikneda samal serveril(samas andmebaasis), kus sellest hoolimata võib nende allsüsteemide vahel käia andmete vahetus.
- Ühe sisulise allsüsteemi andmed (“ost-müük”) võivad süsteemis paikneda hajusalt (erinevates andmebaasides, erinevatel serveritel) ja vahetada omavahel andmeid.



Joonis 2

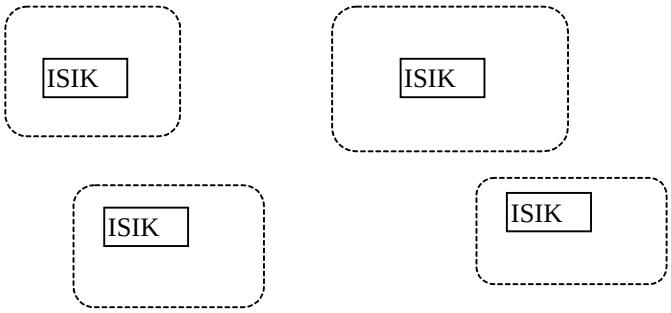
Andmete jagamisel tuleks lähtuda samadest põhimõtetest , millest on olnud juba juttu eelnevates loengutes seoses klient/server süsteemidega.

- Arvestada seda, kus asuvad andtud andmete (allsüsteemide) kasutajad - milliste serverite “külge” on nad ühendatud.
- Allsüsteemides võivad olla andmed (andmetabelid), mida kasutavad ka teised allsüsteemid (mitme allsüsteemi jaoks ühised andmetabelid). Projekteerimisel tuleks ka seda arvestada, ühiste tabelitega tihedamalt seotud allsüsteemid võiks paigutada ühele serverile.



Joonis 3

- Andmete hajutamisel võib mingitel põhjustel olla vajalik hoida samadest andmetest koopiaid erinevates allsüsteemides.



Joonis 4

Siin tuleb lahendada järgmised probleemid -

- millistes allsüsteemides toimub dubleeritud tabelite uuendamine/muutmine
- kuidas tehtud muudatused jõuavad teistesse allsüsteemidesse

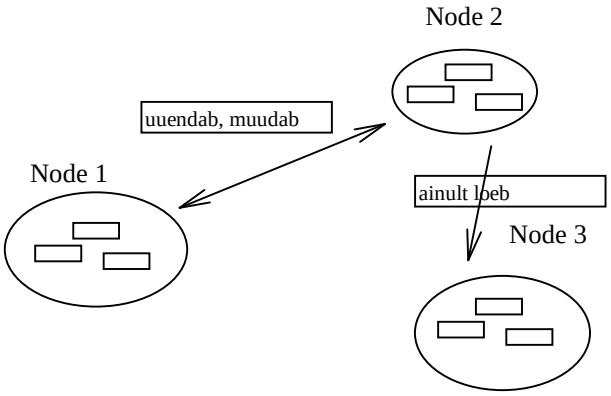
Kuidas kindlustada andmete samasust ja õigsust olikorras, kus samad andmed paiknevad erinevates allsüsteemides ja nende uuendamise ja muutmise taktid ning uuendamise allikad on erinevad.

Erinevates allsüsteemides võidakse samanimelistes tabelites hoida ka erinevaid andmeid, või siis andmeid nn. ülekattega - osa on lokaalsed andmed, osa andmeid on pärit teistest süsteemidest. *Replication*.

Alateema: Hajussüsteemides kasutatavad mõisted, mõningad tegevusjuhised hajussüsteemide projekteerimisel, probleemid projekteerimisel.
Alateema - hajussüsteemi dünaamika projekteerimine - andmevahetus, sündmused mis käivitavad andmevahetuse, andmevahetusprotseduurid, transaktsioonid. Põhiteemaks **sündmused** ja **protsessid** ning **transaktsioonid**.

Transaktsioon on loogiline töö ühik, mis sisaldab endas teatavat hulka SQL-lauseid.Transaktsioon on *atomaarne* - täidetakse kas kõik selles sisalduvad SQL-lause (*committ*) või ei täideta ühtegi (*rollback*). Transaktsioon algab esimese täidetava SQL-lausega ja lõbeb kas COMMITT-I või ROLLBACK-iga.
Transaktsiooni näide: pangas raha ülekandmine ühelt pangaarvelt teisele sisaldab endas tegelikult kolme tegevust 1. ühelt arvelt maha arvamine 2. teisele arvele juurdepanemine 2. ülekande registreerimine. Kõik need protseduurid on vaja viia ühe transaktsiooni koosseisu, nii et nad kas täidetakse kõik koos või ei täideta ühtegi.

Allsüsteem (*node*) on hajussüsteemi mingi punkt, kus asub kas **andmebaas** või mingi andmetöötlusnõudeid tekitav **tarkvarasüsteem** (näiteks kliendiprogramm) . Enamikus allsüsteemides on olemas **mõlemad**. *Node* = allsüsteem



Joonis 1. Hajussüsteemi punktid (*nodes*)

Esiteks üks võimalik hajusandmebaasi ülesehituse skeem, millel mõnevõrra näha ka süsteemi funktsionaalsust ja toimemehhanisme.

Süsteemi komponendid (vt. joonis 2):
- Kasutajaprogramm(liides) (User Request Interface)
- Jaotatud transaktsioonide koordinaator (Distributed Transaction Manager , **DTM**) - võtab vastu teenindusnõudeid kasutajaliideselt **transaktsioonide** teostamiseks, "tõlgib" nad andmebaasissüsteemidele vajalikku vormi ja edastab andmebaasi mänedzheridele (konkreetsete andmebaasidega ühendatud programmidele), mida enamasti on mitu ja mis paiknevad arvutivõrgul hajusalt.
- Andmebaasi mänedzher (Database Manager , **DM**) on programm, mis vastutab talle DTM poolt saadetud käskude (SQL-lausete) täitmise ees selles konkreetses andmebaasis, mille külge ta on ühendatud.

DTM ja DM moodustavad kokku hajusandmebaasisüsteemi ehk **DDBMS-i** (Distributed Database Management System)

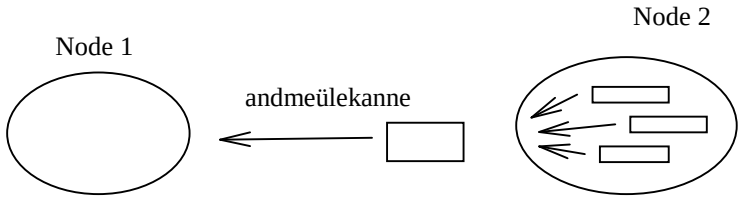
Sellise hajussüsteemi üheks iseseisvaks osaks (süsteemi punktiks, ingl.keeles *node*) võib olla allsüsteem oma riistvaraga, mis suudab realiseerida kas DTM-i, DM-i või mõlemad koos. Sellist süsteemi võib põhimõtteliselt lõpmatult laiendada DTM-ide ja DM-ide lisamise teel.

Põhiprobleemid ja ülesanded protseduuride ja transaktsioonide projekteerimisel hajussüsteemis

* kasutajatele (süsteemi teistele osadele , *nodes*) tuleb kindlaks määrata protseduursed õigused mingi andmebaasi suhtes - millised allsüsteemidest võib selle andmebaasi andmeid lugeda, millised võivad uuendada ja muuta. Süsteemi realiseerimisel võivad tekkida probleemid sellega, et mitmest allsüsteemist soovitakse juurdepääsu samadele andmetele, andmebaas ei jõua nõudeid (*requeste*) teenindada, tekivad probleemid samade andmete muutmisel mitmest küljest (nn. *concurrency control*)

* andmetel andmebaasis on mingid kindlad tekkimise ja uuendamise protseduurid, kusjuures mõned andmed võidakse üle kirjutada uuete andmetega. Samas võivad olla allsüsteemid, mis vajaksid neid vanemaid andmeid ka peale seda , kui ülekirjutamine on juba toimunud st. soovitakse näha nn. aegridasid, st. kuidas on aja jooksul andmed muutunud (tähtis eriti) finanantssüsteemides). Siin saab olla lahenduseks see, et "asjast huvitatud" allsüsteemid kannavad vajalikud andme endale üle enne, kui need antud andmebaasis üle kirjutatakse. Teiseks võimaluseks (ja soovitatavaks) on siiski see, et andmebaasis endas säilitatakse andmete muutumise ajalugu ja andmeid mitte ei kirjutata üle vaid lisatakse (sellisel juhul peab olema muidugi ka vastav andmebaasi struktuur, et selline säilitamine oleks võimalik).

* erinevate allsüsteemide vahel toimuvad **andmeülekanded** - andmeid kantakse ühest andmebaasist teise allsüsteemi teise andmebaasi. Erinevates allsüsteemides võivad andmestruktuurid olla erinevad - seega peavad andmeülekandeprotseduurid suutma andmed ühtedest struktuuridest kanda üle teiste andmestruktuuridesse nii, et ei tekiks vigu andmetes. Struktuurimuutused seisnevad enamsti selles , et mingi kaugemal asetsev allsüsteem vajab konkreetsest andmebaasist mingeid **väljavõtteid , koondeid, päringu tulemusi**, mis siis andmeülekande käigus muudetakse eraldi andmetabeliks ja kantakse üle andmeid vajavasse allsüsteemi. St. andmete struktuur muutub ülekandmisel enamasti lihtsamaks.



Joonis 2.

Süsteemi realiseerimisel on need koondid , päringud vormistatud andmebaasis SQL-i VIEW-dena.

* Erinevates allsüsteemides võivad ülekantavatele andmetele kehtida erinevad piirangud (CONSTRAINTS) ja õigsuse kontrollid (millised andmevälja ei tohi olla nullid vms). Andmete ülekandmisel võivad tekkida konfliktid juhul, kui selles allsüsteemis, **kuhu** andmed kantakse , on piirangud nendele **rangemad**, kui selles allsüsteemis, **kust** ülekanne tehakse. (*Node* 1 piirangud on rangemad kui *node* 2 piirangud ja andmeid kantakse *nodest* 2 *node* 1)

* Allsüsteemides võivab andmebaasi struktuur ja sellest tulenevalt andmekoosseis muutuda (peame meeles seda , et hajussüsteemis võib allsüsteemide väljatöötamine ja arendus toimuda suhteliselt sõltumatult). Andmebaasi struktuuri muutumisel tuleb muuta ka andmete ülekandmise protsewduure.

* Need andmeuuendused, mis peavad olema "kokku kogutud" transaktsioonidesse, peaksid olema projekteeritud nii, et nad ei haara väga suurt hajussüsteemi osa, et nad ei haara väga laiali paiknevaid allsüsteeme, sest siis on transaktsiooni täitmise tõenäosus väiksem (võrguühenduste ebakindlus ja väike jõudlus vahepeal).
NB!Et transaktsioone saaks nii projekteerida , peab olema vastavalt projekteeritud ka hajussüsteemi staatiline struktuur(st. see kuidas on admed jagatud allsüsteemide vahel) Siin on kokkupuutepunkt hajussüsteemi staatilise struktuuri ja protseduuride projekteerimise vahel. Andmed,

mille puhul on näha, et "üle nende" peaksid käima transatktsioonid on selles mõttes omavahel seotud ning on soovitav sellised andmed paigutada ühte allsüsteemi (ühele andmebaasi serverile).

* Hajussüsteemi projekteerimisel tuleb realiseerida mitut erinevat tüüpi "läbipaistvust" (*transparency*), neist tähtsamad on:

- **koha läbipaistvus** (*location transparency*) , st. kasutajale ei loe see , millises allsüsteemis talle vajalikud andmed paikneva, tema jaoks paistavad nad ühtse andmeruumina. Saavutatakse erinevate andmebaaside võrku ühendamise ja andmebaasi serveritel vajalike SQL view-de formeerimise teel.
- **andmetöötluse kiiruse läbipaistvus** (*perfomance transparency*) st. kasuta seisukohast toimub töö kaugemal paiknevate ja/või raskesti kättesaadavate ja töödeldavate andmetega sama kiiresti, kui lokaalsete andmetega.
- **andmebaasi skeemi muutuse läbipaistvus** (schema change transparency) st. kui süsteemi lisatakse või eemaldatekse objekt, jõuab see muutus kõigisse vajalikkesse hajussüsteemi punktidesse, muudatuste laialikanduvus ja läbipaistvus. Kui süsteemi laiendatakse , peab saama lisada uusi andmevälju,tabeleid või neid eemaldada ja modifitseerida, eesmärgiks muu hulgas ka kogu süsteemi jõudluse tõstmine.Vajalik paindlik süsteemi muutmine.Raske realiseerida.

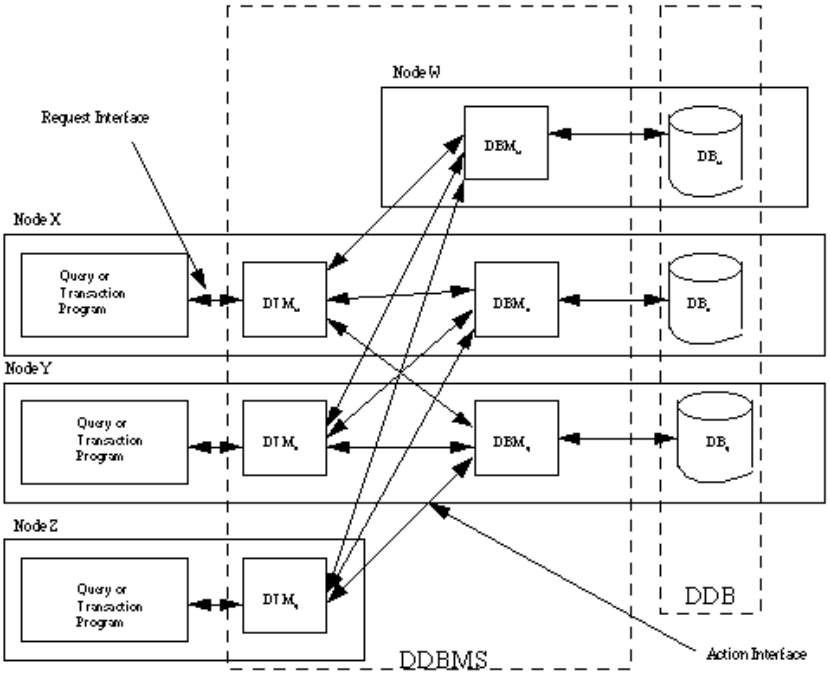
* Hajussüsteemides tekib sageli konflikte objektide (tabelite, andmeväljade) nimedega - erinevates allsüsteemides on samad andmed erinevate nimede all ja vastupidi - erinevates allsüsteemides on samanimelised tabelid. Probleeme tekib mõlemal juhul, hajuspäringute korral tekib probleeme samanimeliste tabelitega erinevates andmebaasides.

* Hajussüsteemides on üheks oluliseks ülesandeks ka andmekaitse organiseerimine ja kasutajate (allsüsteemide) õigused andmetele juurdepääsuks. Securtity vähendab samal ajal muidugi süsteemi jõudlust. Andmeülekannete andmekaitse, andmete krüpteerimine ülekandel.

Mõningaid eesmärke, mille poole tuleks püüda hajussüsteemi realiseerimisel.

- üksik allsüsteem peaks olema nii andmete kui protsesside (transaktsioonide) poolest võimalikud **autonoomsed**.Nii on üksikud allsüsteemid ka paremini omaette arendadavad ja muudetavad.
- süsteemi **jõudluse** parandamine, optimiseerimine . Saavutatakse andmete liikumise ja päringute optimiseerimisega .

Distributed Database Architecture



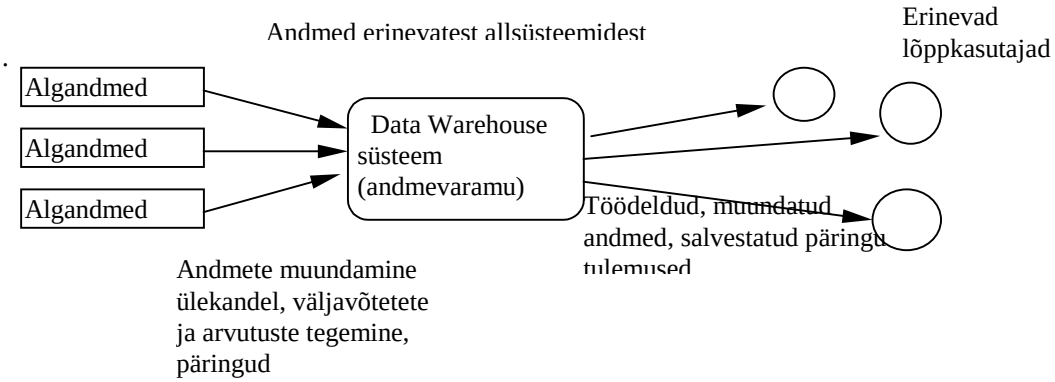
Teema24. **Data Warehouse**

Data Warehouse (andmevaramu):

- Vajadus: On tarvis andmed andmete tekkimise kohtadest (lokaalsed andmebaasid, allsüsteemid) teha kättesaadavaks teiste allsüsteemide kasutajatele, või kogu terviksüsteemi kasutajatele. Kusjuures tarvis on teistele kättesaadavaks teha ainult teatud andmed, teatud kriteeriumite järgi kogutud andmed. Sellist süsteemi, mis kogu infosüsteemis olevaid andmeid organiseerib ja serveerib erinevat tüüpi ja erinevates kohtades asuvale kasutajale, võib nimetada andmevaramuks.

- Põhimõte: Andmevaramu põhineb andmete kogumise, salvestamise ja kasutajale serveerimise süsteemil - on teada süsteemi kasutajate andmevajadused
  - on teada nendele kasutajatele vajalikud andmed (asukoht, andmestruktuur, formaadid jne.) . Algandmete asukoht on allsüsteem oma andmebaasiga, kuhu andmed oma esialsel kujul tekivad - andmete sisestamine, tootmine.

Andmevaramus on olemas andmed oma esialgsel kujul (algallikas) ja andmed oma muutunud kujul - sellisel kujul, nagu nad esitatakse mingile teatud tüüpi kasutajale



Joonis: Andmevaramu - andmete kogumise ja esitamise süsteem

Andmevaramu ülesandeks on nüüd nö. "hankida" lõppkasutajate jaoks vajalikud andmed , muundada need kasutaja jaoks vajalikku vormi ja esitada.

NB! Andmevaramu põhiomaduseks ongi andmete muundamine, mingite kriteeriumite alusel andmete väljavalimine päringute abil nendest allsüsteemidest , kus andmed tekivad. Reeglina vajavad kasutajad väljastpoolt konkreetset allsüsteemi (mille jaoks Data Warehouse tegelikult andmeid serveerib) mitte algandmeid, vaid nende algandmete põhjal koostatud andmestikku, koondtabelid, üldistatud infot. Selliste tarbijale vajalikke koondite jaoks andmete kogumisega (pärigute tegemine vastavatesse allsüsteemidesse, andmeülekanne) ja saadud andmete (s.o. päringutulemuste) salvestamise ja kasutajale esitamisege tegelebki Data Warehouse.

- Komponendid, omadused:

- Data warehouses peavad olema vajalikud andmete kogumise mehhanismid ja reeglid

- Peab olema päringutulemuste salvestamise koht reegline (kuigi mõnel juhul võidakse allsüsteemidest kogutud andmed edastada kasutajale ka otse, ilma vahepealse salvestamiseta.

Mõlema eeltoodud komponendi realiseerimiseks on kasutatav SQL andmebaasi serverid, kuhu on siis projekteeritud vastavad SQL skriptid päringutega ja samuti nende päringutulemuste salvestamiseks vajalikud andmestruktuurid

omadus: Data warehouse võib andmeid samadest allsüsteemidest hankida ja töödelda paralleelselt ja täiesti erinevalt erinevat tüüpi kasutajate jaoks - sõltuvalt sellest, millised on kasutaja infonõuded (milliste näitajate ja kriteeriumite järgi kasutajale tuleb infot valida ja millised arvutisi sellele andmestikule rakendada, kehtib eriti majandusega seotud süsteemide puhul - summad, keskmised, miinimumid jne.)

omadus: Data warehouse puhul ei suhtle (ei saa andmeid sealt) kasutajad otse andmeallikatega (nende allsüsteemidega, kus asuvad algandmed), vaid Data Warehouse süsteemiga, mis hoiab enda serveril kasutale vajalikke ja juba töödeldud andmeid neist allsüsteemidest või siis vahendab kasutaja päringut nendesse allsüsteemidesse.

Omadus: Tavalielt peavad Data Warehouse mehhanismid oma andmeid värskendama, kuna andmed allsüsteemides muutuvad ja uuenevad - perioodilised päringud, mille tulemused salvestatakse Data warehouse serverile. Andmete värskendamise sagedus sõltub kasutaja nõudmistest ja algandmete muutumise kiirusest ja iseloomust.