TALLINNA PEDAGOOGIKAÜLIKOOL

Matemaatika – Loodusteaduskond
Informaatika õppetool

Erik Iter

ANDMEHOIDLAD

Proseminaritöö

Juhendaja: Mihkel Märtin

Tallinn 2002

SISUKORD

SISSEJUHATUS

Andmehoidla (*data warehouse*) on eri liiki andmebaas, mis on viimastel aastatel infotehnoloogiamaailmas endale suurt tähelepanu tõmmanud. Andmehoidla tähendab eelkõige informatsiooni kättesaadavaks tegemist. Keegi ei kahtle info väärtuses ja kõik nõustuvad, et enamusel organisatsioonidel on potensiaalne “kullakaevandus” oma igapäevaselt kasutatavate, niinimetatud operatsioonilistes süsteemides luku taga. Andmehoidla võib olla võti, mis avab ukse sellesse informatsiooni.

Andmehoidla kõrval võib kohata ka teistsugust eestikeelset vastet *data warehouse*’i mõistele – andmeait. Siin töös on kasutatud enamsoovitatud esimest varianti. Olgu kasutatud kumba terminit tahes, eestikeelset põhjalikumat infot on selle kohta äärmiselt raske leida. Üks selle töö eesmärk ongi selle probleemi leevendamine.
Esmapilgul võib andmehoidla paista üsna lihtsa rakendusprogrammina. Tegelikkuses on ta kõike muud kui lihtne. Seda nii suuruse (andmehoidla andmebaasid on ühed suurimaist maailmas, 2001 aasta uuringu põhjal oli keskmine suurus 113 GB) ja sellest tulenevate jõudluseprobleemide kui ka andmete struktuuri tõttu. Käesolev töö peakski selgitama andmehoidlate olemust, kuidas neid kasutatakse on ning nende ehitamise juures tekkida võivaid probleeme.
Tegevuspõhimõtete tutvustamiseks on alustuseks kirjeldatud otsuste toetamise vajalikkust ja kuidas andmehoidlad seal valdkonnas aidata saavad. Edasi on räägitud erinevustest operatsiooniliste süsteemide ja andmehoidlate vahel ning dimensionaalsetest mudelitest. Ülejäänud materjal tegeleb andmehoidla põhikomponentide kirjeldamisega. Enamust mõisteid on ka üritatud illustreerida näidetega.

Käesolev töö oletab, et lugejal on vähemalt mingid baasteadmised relatsiooniteooriast ning SQL päringukeelest ja seetõttu süntaksit lähemalt ei seleta. Rohked näited aga peaksid võimaldama töö lugemist ka mittearvutiinimestel ja töö ongi põhimõtteliselt mõeldud kõigile, kes andmehoidlate põhialuseid teada tahavad.

OTSUSTE TOETAMISE SÜSTEEMID

Andmehoidlad on tihedalt seotud üldise äriharuga, mis on tuntud kui otsuste toetamine (*decision support*). Selleks, et mõista andmehoidlate põhimõtet, peab enne mõistma otsuste toetamise süsteemide (OTS, *decision support system* – DSS) üldist otstarvet.

Otsuste toetamise süsteemid on erineval kujul eksisteerinud palju aastaid. Ammu enne esimeste andmebaasi haldussüsteemide (DBMS – *database management system*) leiutamist eraldati programmidest infot, et aidata juhtkonnal efektiivsemalt oma organisatsiooni juhtida.

Sellise süsteemi otstarve on varustada firma või organisatsiooni otsusetegijaid informatsiooniga. See informatsioon omakorda edendab ja suurendab otsusetegijate teadmisi mingil viisil, aidates neil organisatsiooni strateegia ja poliitika kohta otsuseid vastu võtta.

OTS omab tavaliselt järgmisi karakteristikuid:

- Nad kalduvad olema suunatud selliste probleemide lahendamisele, mida ei saa hästi määratleda ega sõnastada ja millega tüüpiliselt tegeleb kõrgem juhtkond.
- Nende omadused võimaldavad neid interaktiivselt kasutada ka mittearvutiinimestel.
- Nad on küllalt paindlikud ja kohandatavad, et lubada teha muudatusi kasutuskeskkonnas ja lähenemisviisil otsuste tegemisele.

OTS-i ülesanne on tavaliselt pakkuda faktilisi vastuseid kasutaja poolt formuleeritud küsimustele. Näiteks müügijuhti paneks kindlasti muretsema see, kui toodete müük jääb ülemuse poolt määratud eesmärkidest väiksemaks. Küsimus, mida ta küsida tahaks, võibki olla selline:

Miks on müük seatud eesmärkidest väiksem?

Tänapäeval ei eksisteeri veel ühtki arvutisüsteemi, mis võimaldaks sellisele küsimusele vastata. Raske on ka ette kujutada sellist SQL (*Structured Query Language*) keele päringut, mis sellega hakkama saaks. Esitatud küsimused peavad olema rohkem süstemaatilised, et OTS-il oleks võimalik anda faktilisi vastuseid. Esimene küsimus võiks niisiis olla:

Mis on iga toote kogumüük ja seatud eesmärk selle aastal?

OTS vastaks toodete ja nende müügisummade nimekirjaga. Tõenäoline on, et osa tooteid on eesmärgist ees ja osa maas. Hästi koostatud aruanne võib problemaatilised tooted esile tõsta. Näiteks kuvada need punaselt või vilkuvana, et need kergemini silma paistaks. Esitatud küsimus oleks võinud ka olla:

Mis on nende toodete kogumüük ja eesmärgid sellel aastal, mille tegelik müük on väiksem kui seatud eesmärk?

Avastanud need tooted, mis pole eesmärki saavutanud, võib müügijuht küsida, mis on firma turuosa nende toodete osas ja kas see turuosa väheneb. Kui see on nii, siis võib põhjuseks olla näiteks hiljutine hinnatõus. OTS-i otstarve on vastata sellisetele ad hoc küsimustele, et kasutaja jõuaks lõpuks mingile järeldusele ja oleks võimeline otsuse tegema.

Suurem piirang OTS-de arendamises on andmete kättesaadavus – see tähendab, juurdepääs õigetele andmetele õigel ajal. Kuigi andmebaasisüsteemide kiire levik ja arenemine on aidanud andmeid programmidest kergemini kätte saada, valmistab andmete hankimine ikka raskusi. Kõrgetasemeliste andmebaasi haldussüsteemide kasutuselevõtt on seda probleemi küll leevendanud, kuid andmete kättesaadavus valmistab isegi tänapäeval enamuses organisatsioonides probleeme. Peamine põhjus on see, et enamus organisatsioone arenevad aja jooksul. Arenemise käigus ei suuda arvutisüsteemid enam organisatsiooni funktsionaalseid nõudmisi täita. Tulemusena modifitseeritakse rakendusprogramme pidevalt, et need pidevalt muutuva äriga sammu peaksid. Mingi hetk tuleb peaaegu kõigi programmide elus aeg, kus neid on modifitseeritud sel määral, et neid on võimatu või ebapraktiline rohkem muuta. Sel hetkel võetakse tavaliselt vastu otsus rakendusprogramm uuesti arendada. Kui selline olukord tekib, siis harilikult arendajad kasutavad ära kõiksugu tehnoloogilised uuendused, mis selle programmi eluajal on leiutatud. Näiteks võis originaalne programm olla kirjutatud programmeerimiskeeles COBOL, sest see oli tollal sobiv tehnoloogia. Ent suurtel organisatsioonidel võib olla kümneid või isegi sadu erilaadseid programme. Nende kasulik eluiga lõpeb eri aegadel ja neid arendatakse uuesti tükkhaaval. See tähendab, et mingil suvalisel ajahetkel töötavad organisatsioonis programmid, mis kasutavad hulgaliselt erinevaid tarkvaratehnoloogiad.

Lisaks on suurte organisatsioonide süsteemid ka mitmesuguste eri riistvaraplatvormide peal. On üldine, et ühe firma rakendusprogrammid laotuvad üle järgmiste platvormide:

- Suur server
- Mitu väiksemat multiprotsessoriga masinat
- Välised teenusepakkujad
- Võrku ühendatud ja autonoomsed (*stand-alone*) PC-d

OTS võib vajada ligipääsu mistahes programmi nende hulgast, et vastata kasutaja poolt temale seatud küsimustele.

STRATEEGILINE JA OPERATSIOONILINE INFORMATSIOON

On väga tähtis mõista erinevusi terminite strateegiline ja operatsiooniline vahel. Üldiselt, strateegilised küsimused käsitlevad planeerimist ja poliitika tegemist ja see on koht, kus andmehoidla aidata saab. Paar näidet strateegilistest otsustest:

- Kui telekommunikatsioonifirma otsustab tarvitusele võtta väga odavad tipptunnivälised tariifid inimeste meelitamiseks mitte rääkima tipptundidel selle asemel, et paigaldada lisaseadmeid ja nii toime tulla suurema nõudlusega.

- Suur kaubamajade kett otsustab avada oma poed ka pühapäeval.
- Üldine 20 % hinnalangus üheks kuuks, et suurendada turuosa.

Kui strateegilised asjad on seotud planeerimise ja poliitikaga, siis operatsioonilised küsimused tegelevad rohkem organisatsiooni või äri igapäevaste protsessidega. Operatsioonilisi tegevusi võib pidada organisatsiooni strateegia täideviimiseks, rakendamiseks.

Igapäevane tarvikute tellimine, klientide tellimuste täitmine ja uute töötajate palkamine on kõik näited operatsioonilistest protseduuridest. Neid protseduure toetavad tavaliselt arvutiprogrammid ja seetõttu nad peavad oskama vastata ka operatsioonilistele küsimustele nagu:

- Kui palju on täitmata tellimusi?
- Mis tooted on laost otsas?
- Mis seisundis on mingi kindel tellimus?

Tüüpiliselt saab operatsiooniliste süsteemide abil sellistele küsimustele kergelt vastused leida, sest need on küsimused hetkel kehtiva situatsiooni kohta. Kõigile esitatud küsimustele on võimalik lisada lõppu sõna “hetkel” ja ikka oleksid need küsimused loogilised. Sellised küsimused tekivad organisatsiooni normaalse tegevuse käigus. Aga sellist liiki küsimused, mida firma direktorid ja juhtkond tahaks küsida, on:

- Milliste tooterühmade populaarsus kasvab ja milliste langeb?
- Mis tooterühmad on hooajalised?
- Missugused kliendid esitavad regulaarselt ühesuguseid tellimusi?
- Kas mõni toode on populaarsem ühes riigi osas kui teises?

Need ei ole selgelt “hetkel” tüüpi küsimused ja harilikult operatsioonilised süsteemid ei oska sellistele küsimustele vastata. Põhjus seisneb operatsiooniliste süsteemide olemuses. Nad on arendatud toetamaks organisatsiooni igapäevaseid tegevusi. Võib öelda, et operatsioonilised süsteemid kujutavad organisatsiooni selle hetke ülesvõtet. Rakendusprogrammides hoitud väärtused muutuvad pidevalt. Suvalisel ajahetkel võidakse kas kõigis või mistahes üksikus süsteemis täide viia kümneid või sadu *insert*, *update* või *delete* lauseid korraga (*insert*, *update* ja *delete* on SQL päringukeele laused). Kui süsteemid korraks paigale tarduksid, näitaksid nad organisatsiooni seisundi peegeldust täpselt sellel momendil. Üks sekund varem või hiljem oleks olukord juba muutunud.

Kui ülalolevaid nelja strateegilist küsimust uurida, on näha et iga küsimus on seotud toodete müügiga aja jooksul. Vaadates esimest küsimust:

Milliste tooterühmade populaarsus kasvab ja milliste langeb?

See on ilmselt mõistlik strateegiline küsimus. Olenevalt vastusest, võivad direktorid näiteks teha ühte järgmistest:

- Suurendada mõnede toodete levikut ja vähendada teiste toodete oma.
- Pakkuda tooteid vähendatud hindadega või hinnaalandust hulgi ostes.
- Parandada nende toodete reklaami, mille populaarsus on langemas.

Ainus viis, kuidas me saame hinnata, kas tooterühma populaarsus kasvab või kahaneb, on jälgida selle nõudlust aja jooksul. Juhul kui tellimuste töötlemise informatsiooni hoitakse relatsioonilises andmebaasis, saab koostada näiteks järgmise SQL päringu:

```
SELECT Nimi, SUM(Kogus), SUM(Tootemaksumus)
        FROM Klienditellimus A, Tellimusekomponent B, Toode C
        WHERE A.Tellimusekood = B.Tellimusekood
        AND A.Tootekood = C.Tootekood
        AND Tellimuseaeg = <tänane kuupäev>
        GROUP BY Nimi
```

Kui seda päringut käivitada iga päeva lõpus, siis saaks vastuseks konkreetse toote kõigi tellimuste koguväärtuse selle päeva kohta. Oskus leida kõigi tänaste tellimuste väärtus on hea algus. See on kasulik informatsioon, aga tegelikult tahetakse uurida näiteks viimase kuu kuu trendi või võrrelda selle kuu tulemusi eelmise aasta sama kuuga. Lahendus oleks käivitada seda päringut iga päev ja lisada selle päeva tulemused tabelisse. Sel viisil saaks aja jooksul üles ehitada vajaliku ajaloolise informatsiooni. See on andmehoidla algus.

ANDMEHOIDLA

See peatükk defineerib andmehoidla mõiste ja analüüsib seda definitsiooni, võrreldes ja seades andmehoidlale kontrastiks operatsiooniliste süsteemide omadusi.

Andmehoidla üldtunnustatud definitsioon (omistatud Bill Inmon'ile, 1992) on - andmebaas, millel on järgmised neli omadust:

1. Teemale orienteeritud (*subject oriented*)
2. Muutumatu, püsiv (*nonvolatile*)
3. Integreeritud (*integrated*)
4. Ajateisendlik (*time variant*)

Teemale orienteeritud tähendab, et andmed on organiseeritud ümber teemade (nagu müük), mitte ümber operatsiooniliste programmide (nagu tellimuste töötlemine). Operatsioonilised andmebaasid on organiseeritud ümber äriliste programmide; nad on programmidele orienteeritud (*application oriented*).

Muutumatu tähendab, et kord andmehoidlasse paigutatud andmed tavaliselt enam ei muutu. Andmehoidla kasutaja võib kindel olla, et päring annab alati sama vastuse hoolimata sellest, kui tihti seda käivitatakse. Operatsioonilised andmebaasid ei ole püsivad ja muutuvad pidevalt. On ebatõenäoline, et päring annab kaks korda sama vastuse, kui ta kasutab tabeleid, mida tihti uuendatakse.

Integreeritud tähendab, et andmed kokkusobivad. Näiteks kuupäevad on alati salvestatud samas formaadis. Integratsioon on probleemiks paljudes organisatsioonides, iseäranis seal, kus on kasutusel mitmeid erinevaid tehnoloogiaid. Mõned erinevused on täiesti fundamentaalsed, näiteks kooditabel. Enamik süsteeme kasutavad ASCII (*American Standard Code for Information Interchange*) kooditabelit, aga mõned mitte. IBM-i, ühe suurima arvutitootja maailmas, kõik suuremad serverisüsteemid ja ka paljud väiksemad baseeruvad täiesti erineva kooditabelil, mida nimetatakse EBCDIC (*Extended Binary Coded Decimal Interchange Code*). Nii on tähel "P" väärtus 80 ASCII-s, aga 215 EBCDIC-s (väärtusega 80 olev märk on EBCDIC-s "&"). Sõna "Pool" ASCII-s tõlgendub kujule "&??%" EBCDIC-s ning on raskem midagi vähem integreeritumat ette kujutada. Teised erinevused on peenemad, näiteks kuupäevad. Enamikul andmebaasi haldussüsteemidel on "*Date*" andmetüüp (talletamisformaadid võivad küll lahku minna üksteisest), aga indekseeritud järjestikuse juurdepääsuga meetoditel selline vahend puudub. Veel peenemad erinevused leiduvad sama tehnoloogiat kasutatavates programmides. See juhtub, kui näiteks üks programmeerija otsustab hoida kliendi aadressi viies 25 märgi pikkuses veerus, aga teine kasutab VARCHAR(100) formaati. Enne kui andmed võib andmehoidlasse sisestada, peavad nad integreerima. Integratsioon on järelikult protsess, mille andmed läbivad pärast rakendusprogrammist lahkumist ja enne andmehoidlasse sisenemist.

Ajateisendlik tähendab, et salvestatakse ajaloolisi andmeid. Peaaegu kõik andmehoidla peal käivitatavad päringud on seotud kuidagi ajaga. On enam-vähem võimatu minevikku vaatlemata ennustada, mis tulevikus juhtub. Andmehoidla aitab sellist

fundamentaalset probleemi lahendada, lisades operatsioonilistes andmebaasidest võetud andmetele ajaloo dimensiooni.

Üks edukamaid andmehoidla kavandamise tehnikaid on dimensionaalne analüüs ja dimensionaalne modelleerimine (*dimensional analysis and dimensional modelling*).

DIMENSIONAALNE ANALÜÜS

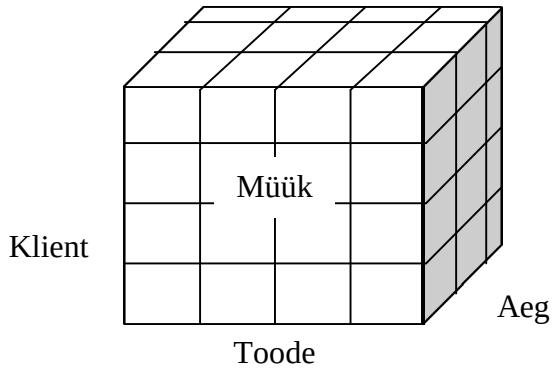
Üks lähenemisviis andmehoidla kujundamisele on arendada ja teostada "dimensionaalne mudel". See on pannud aluse dimensionaalsele analüüsile (mõnikord üldistatud kui multidimensionaalne analüüs).

Juba esimeste andmehoidlate arendamise juures märgati, et millal iganes otsusetegijatel paluti kirjeldada oma organisatsiooni puudutavaid küsimusi, millele nad vastuseid sooviksid, siis peaaegu alati soovisid nad järgnevat:

- Summeeritud informatsiooni koos võimalusega summade sisu detailsemalt vaadelda.
- Summeeritud informatsiooni analüüsi üle oma organisatsiooni komponentide nagu osakonnad või regioonid.
- Võimalust "lõigata ja tükeldada" informatsiooni igal neile sobival ja mõeldaval viisil.
- Informatsiooni kuvamist nii graafiliselt kui tabelitena
- Võimalust vaadelda oma informatsiooni mingi ajaperioodi vältel.

Näitena võib tuua kindlustusfirma, mis soovib näha aruannet, mis kuvab kindlustuspoliiside müüki toote kaupa (kinnisvara-, elu-, autokindlustus jne) või aruannet müügi kohta kliendi kaupa, või isegi müüki mõlema - toote ja kliendi kaupa.

Lähenemine on organisatsiooni sobivaid otsusetegijaid intervjueerides kindlaks teha, mis teemast nad kõige rohkem huvitatud on ja mis on enamtähtsamad analüüsi dimensioonid. Ülaltoodud näite puhul oleks teema müük. Analüüsi dimensioonid oleksid kliendid ja tooted. Nõue oleks analüüsida müüki kliendi kaupa ning müüki toote kaupa. See nõue on kujutatud joonisel 1 oleva kolmedimensionaalse kuubina:



Joonis 1. Kolmedimensionaalne andmekuup.

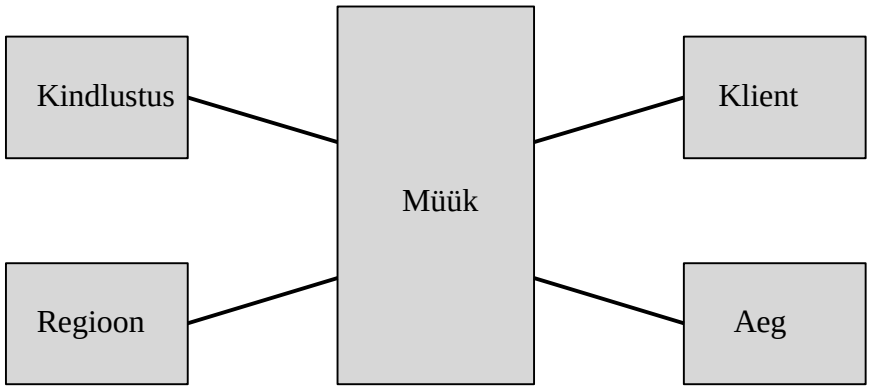
Joonis 1 näitab müüki koos järgmiste telgedega:

1. Klient
2. Toode
3. Aeg

Aega suhtutakse kui vajalikku analüüsi dimensiooni (ajateisendlikkus on üks andmehoidla omadus) ja seetõttu on ta alati analüüsi dimensioonina lisatud. See tähendab et müüki on võimalik analüüsida kliendi ja toote kaupa aja jooksul. Nii, et iga kuubi element (iga minikuup) sisaldab konkreetse toote müügi väärtust konkreetsele kliendile mingil kindlal ajahetkel.

Multidimensionaalne kuup joonisel 1 näitab teemana müüki koos kolme analüüsi dimensioonina. Tegelikku limiiti dimensionaalses mudelis kasutatavate dimensioonide arvul ei ole, küll aga on piir dimensioonide arvul, mida on võimalik joonistada.

Ütleme, et kindlustusfirma direktorid vajavad vastuseid müügi kohta – toote kaupa, kliendi kaupa, regiooni kaupa. Nende andmehoidla teema on selgelt müük. Analüüsi dimensioonid on toode, klient, regioon ja sunduslik “aeg”. Kuna neljadimensionaalset mudelit joonistada ei saa, võib esitada kontseptuaalse dimensioonilise mudeli nagu on näidatud joonisel 2.



Joonis 2. Kindlustuse müügi dimensionaalne mudel kindlustusfirma jaoks.

Joonisel 2 näidatud diagrammile viidatakse sageli kui Täht-skeemile (*Star Schema*), kuna diagramm sarnaneb kergelt tähe kujule. Teema piirkond on tähe keskpunkt ja analüüsi dimensioonid moodustavad tähe tipud. Teema piirkond joonistatakse harilikult pika ja peenikesena, sest tabel ise on ka tüüpiliselt pikk ja peenike ehk ta sisaldab vähe veerge aga väga suure hulga ridu. Täht-skeem on kõige sagedamini kasutatav diagramm dimensionaalsete mudelite juures.

ANDMEHOIDLA EHITAMINE

Kui firma on otsustanud andmehoidla ehitada, peab ta edasi otsustama:

1. Kuhu see paigutada?
2. Mis tehnoloogiat tuleks kasutada?
3. Kuidas see täita?

Neid probleeme kohtavad enamus organisatsioone, kes andmehoidlat ehitada üritavad. Nagu öeldud, rakendusprogrammid on projekteeritud kasutades tehnoloogiaid ja meetodeid, mis olid arendamise ajal sobilikud. Otsuste toetamisega harilikult ei arvestatud. Programmid projekteeriti äri operatsiooniliste nõudmiste täitmiseks.

Tavaliselt arendatakse andmehoidla kõikidest teistest rakendustest eraldi, oma andmebaasis. Üks ilmne põhjus on põhjalikud erinevused lähtesüsteemide olemustes (erinevad platvormid, operatsioonisüsteemid, tarkvara arhitektuurid). Kuid on ka teisi põhjusi:

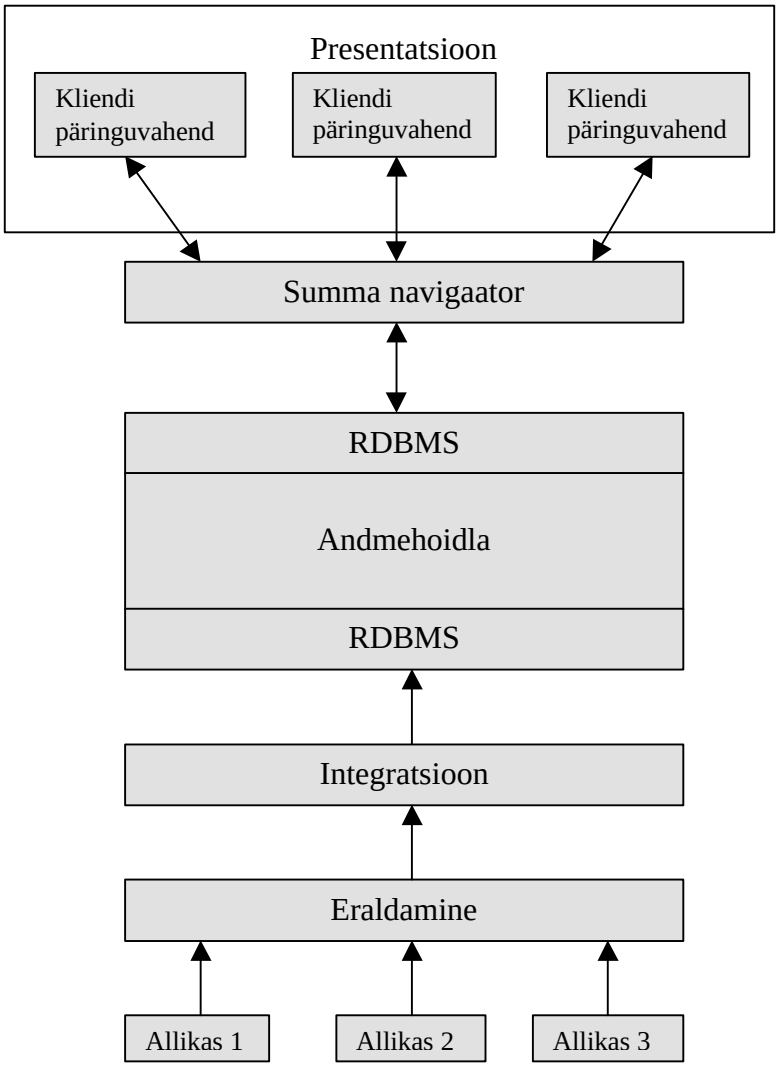
1. Operatsiooniliste ja andmehoidla süsteemide vahel on konflikt – esimesed toetavad operatsioonilisi nõudmisi ja ei ole teemale orienteeritud.
2. Operatsioonilised süsteemid muutuvad pidevalt, aga otsuste toetamise süsteemid on “vaiksed” (muutumatud, püsivad).
3. Operatsiooniliste süsteemide andmebaasiskeemid on tihti väga suured ja keerulised, kus mistahes kahel tabelil võib olla mitu ühendusteed. Sellised keerulised seosed eksisteerivad, sest ärilised protsessid vajavad neid. Sealt järeldub, et päringut kirjutaval inimesel on rohkem kui üks variant kahte tabelit kokku ühendada. Iga võimalik ühendustee toodab erineva tulemuse. Otsuste toetamise skeem lubab ainult ühte ühendusteed.

Veel üks põhjus süsteemide eraldi hoidmiseks on see, et operatsioonilised süsteemid tavapäraselt ei säilita ajalooliseid andmeid. Kuigi see on disaini küsimus ja need süsteemid võiksid kindlasti selliseid andmeid säilitada, ei oleks selle mõju jõudlusele arvatavasti kasutajatele aktsepteeritav ega meelepärane. Strateegilistele küsimustele vastuse saamiseks kuluvad mitu minutit või isegi tundi on tavaliselt vastuvõetavad. Operatsioonilistes süsteemides tehtavad toimingud aga ei tohi harilikult rohkem kui paar sekundit aega võtta.

Igal juhul ei arendata valdavat enamust andmebaase kasutavaid rakendusprogramme ajaloolist infot toetavaks. See tähendab, et tahtes ajateisendlikkust juurutada, peab programmi mingil viisil muutma, teinekord päris oluliselt ja suurel määral. Harilikult ei ole organisatsioonid valmis taluma neid häireid, mis sellega kaasnevad.

Vastuseks andmehoidla ehitamiseks parima tehnoloogia küsimusele on de facto standardiks saanud relatsiooniline andmebaasihaldussüsteem (RDBMS – *Relational Database Management System*). Osaliselt on selle põhjuseks see, et andmehoidlaid arendatakse tavaliselt kasutades dimensionaalset modelleerimist ja relatsiooniline mudel toetab dimensionaalset mudelit väga hästi.

Andmehoidlal on mitu komponenti. Osa on seotud informatsiooni eraldamisega lähtesüsteemidest ja selle andmehoidlasse sisestamisega, teised aga on seotud informatsiooni andmehoidlast kättesaamisega ja selle kasutajale kuvamisega. Joonisel 3 on ära näidatud põhikomponendid.



Joonis 3. Andmehoidla süsteemi põhikomponendid.

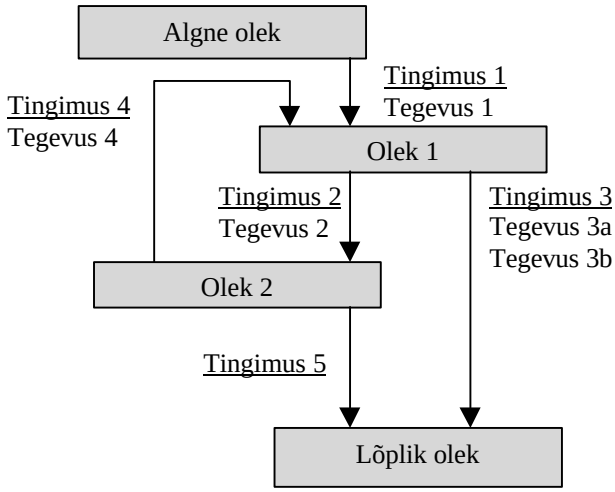
Järgnevalt kõigist põhikomponentidest lähemalt, alustades altpoolt.

ERALDAMISE KOMPONENT

Kasutades eelnevat näidet, ütleme, et tegemist on andmehoidlaga kindlustusfirma jaoks. Esimene probleem on eraldada müügi kohta käiv informatsioon lähtesüsteemidest, et oleks võimalik see andmehoidlasse sisestada.

Et sisestada detailset müügiinfot andmehoidlasse, peab eksisteerima müüki identifitseeriv mehhanism. Kui müük on identifitseeritud, peab selle "kinni püüdma" ja salvestama kusagile, et selle sobival ajal andmehoidlasse saaks panna.

Kindlustuspoliisi müügil, nagu kõigil olemitel, on elutsükkel. Igas elutsükli etapis on poliisil mingi olek ja tüüpiline poliis liigub ühest olekust teise kuni ta elutsükkel on lõppenud ja ta süsteemist eemaldatakse. Selle probleemi – millal info eraldada – saab lahendada, koostades oleku ülemineku diagrammi, mis jälgib olemi elutsüklit ja määratleb need sündmused, mis võimaldavad olemi ühest loogilisest olekust järgmisse minna. Joonis 4 esitab oleku ülemineku diagrammi üldist kuju. Kuigi diagrammil on neli olekut ja viis üleminekut, ei ole tegelikkuses olekute ja üleminekute arvul piiri.



Joonis 4. Üldine olekute ülemineku diagramm.

Kindlustuspoliisi korral võib öelda, et müük on toimunud, kui poliisi olek muutub esimest korda pakkumisest väljaantuks (jõus olevaks). See on koht, kus poliisi kohta käiva info peab salvestama ja tallele panema, et see järgnevalt andmehoidlasse sisestada. Ainus viis seda saavutada on teha mingi muudatus programmi nii, et see muutus olekus ära tuntakse ja salvestatakse.

Enamik organisatsioone ei ole valmis oma operatsioonilisi süsteeme ainult selleks uuesti arendama, et andmehoidlat tarvitusele võtta. Selle asemel peab tarkvara muutma, et nõutud andmed saaksid tabatud ja ajutiselt kuhgi talletatud, et need järgnevalt andmehoidlasse paigutada.

Relatsioonilised andmebaasihaldussüsteemid (edaspidi RDBMS) pakuvad tihti võimalust kasutada andmebaasis niinimetatud “päästikuid” (*trigger*). Need päästikud on tabeli-põhised SQL laused, mis käivitatakse iga kord, kui mingi rida lisatakse, uuendatakse või kustutatakse. On võimalik luua päästik, mis jälgib kindlustuspoliisi olekut iga kord kui seda uuendatakse või muudetakse. Niipea, kui olek näitab, et müük on toimunud, saab päästiku protsess poliisi mingisse ajutisse tabelisse sisestada, et hiljem sinna kogunenud andmed andmehoidlasse viia. Karbis ostetud rakendusprogrammide puhul võib juhtuda, et on vaja programmi algseid programmeerijaid, et andmete kättesaamiseks vajalikud muutused tehtud saaksid.

Sellega ei ole eraldamise protsess veel läbi. Veel on vaja eraldada analüüsi dimensioonide kohta käiv informatsioon – täht-skeemil oleva tähe tipud. Kui klientide, toodete, regioonide ja muu sellise andmeid hoitakse samuti relatsioonilises andmebaasis, siis on üsna lihtne nende tabelite sisuga faile saada. Vajalikud tabelid saab faili salvestada kasutades kõigi RDBMS-dega kaasas olevat ekspordi programmi. Uute klientide/toodete detaile ja olemasolevate muudatusi on veidike raskem hankida, aga tõenäoliselt on parim viis samad ülalkirjeldatud päästikud.

Kui informatsioon juhtub asuma mingi kolmanda osapoole tootes, siis selle eraldamine võib varieeruda triviaalsest võimatuni olenedes programmi kvaliteedist ja seotud tarkvara firma suhtumisest.

INTEGRATSIOONI KOMPONENT

Integratsiooni puhul eksisteerib kaks põhiaspekti: formaadi integratsioon ning semantiline integratsioon.

Formaadi integratsioon

Formaadi integratsiooni probleem on seotud põhiliselt sellega, kuidas kindlustada terviklikkuse taastamist kohtades, kus see on kaotatud. Enamuses organisatsioonides on tavaliselt mitmeid juhtumeid, kus ühe süsteemi atribuudite formaat erineb teiste süsteemide samade või sarnaste atribuutide omast. Näiteks:

- Pangaarve või telefoninumbrid võivad olla "*string*" tüüpi ühes ja "*numeric*" tüüpi teises süsteemis.
- Sugu võibolla talletatud kas viisil "mees"/"naine", "m"/"n", "M"/"N" või isegi 1,0.
- Kuupäevadel võib olla mitmeid erinevaid esitlusviise kaasa arvatud "ppkkaa", "ppkkaaaa", "aakkpp", "aaaakkpp", "pp kuu aa", "pp kuu aaaa". Need on ainult paar näidet. Mõned süsteemid talletavad kuupäevi ajatemplina, mis näitab aega tuhandiksekundilise täpsusega. Teised kasutavad jällegi täisarvu, milleks on sekundite arv alates mingist alates kindlast ajast, näiteks 1. jaanuarist 1900.
- Rahalised atribuudid põhjustavad samuti raskusi. On süsteeme, mis salvestavad raha täisarvnua ja ootavad, et teine programm sisestaks ise komakoha. Muudel võib olla komakohad sisse ehitatud.
- Erinevates süsteemides kasutatakse erinevaid stringi pikkusi nimede, aadresside, tootekirjelduste jms. jaoks.

Formaadi ebatäpsused on väga tavalised, eriti on see tõsi nendes süsteemides, mille puhul kehtib mõni järgmistest väidetest:

1. Aluseks olev riistvara on erinev.
2. Operatsioonisüsteem on erinev.
3. Tarkvara, millel rakendusprogrammid põhinevad, on erinev.

Integreerimise protseduur koosneb seeriast reeglitest, mis on kujundatud selliselt, et kindlustada andmehoidlasse laetavate andmete standarditele vastavus. Et kõik kuupäevad oleksid samas formaadis, rahalised väärtused oleksid kujutatud identselt, jne.
Miks see nii tähtis on?
Võib ette kujutada katset kasutada andmehoidlat, kus tahetakse nimekirja kõigist töötajatest grupeerituna soo, vanuse ja keskmise palga järgi, kui ükski nendest atribuutidest ei ole korralikult standardiseeritud.
See ülesanne oleks küllalt raske ettevõtmine ka kogenud programmeerijale ja kindlasti üsna võimatu mittearvutiinimesele.

Andmehoidla võtab vastu informatsiooni tervest hulgast allikatest ja koondab selle ühte oma tabelitest. See on teostatav ainult siis, kui andmete tüübid ja formaadid on ühilduvad.
Integratsiooni reeglite komplekti kasutatakse nagu kaardistikku, mis defineerib kuidas lähtesüsteemidest, allikatest välja tõmmatud infot peab konverteerima, transformeerima enne kui sellel lubatakse siseneda andmehoidlasse.

Semantiline integratsioon

Semantika on teatavasti soetud asjade tähendusega. Andmehoidlad koondavad infot kokku paljudest erinevatest süsteemidest. Tüüpiliselt kasutavad firmas erinevaid süsteeme ka erinevad inimesed - finantssüsteeme kasutavad raamatupidajad, müügisüsteeme müüjad jne. Kõigil neil inimestel võib olla oma ettekujutus näiteks mõiste müük olemusest. Müüja arvates on müük toimunud, kui kliendilt on tellimus saabunud, laohoidja meelest siis, kui ta kauba välja saadab ja raamatupidaja arvates siis, kui tellija on arve ära maksnud (tavaliselt on kaup selleks ajaks välja saadetud). Tihti pole nende süsteemide kasutajad ise probleemist teadlikud, aga andmebaasi analüüsival inimesel on raske mõista, mis infoga parajasti tegu on.

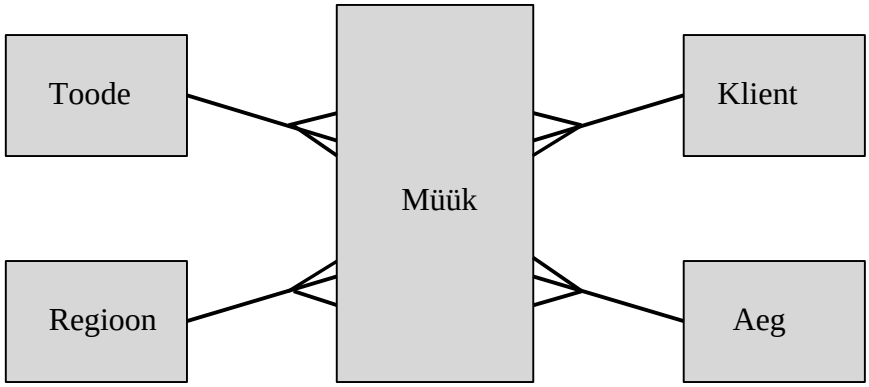
Andmehoidlat ehitades ei saa selliseid kergeid erinevusi ignoreerida, kuna päringute kaudu andmehoidlast saadud infot kasutatakse otsuste toetamiseks organisatsiooni kõige kõrgemal tasandil.
Seepärast on tähtis, et igal objektil, mis andmehoidlasse lisatakse, oleks kindel tähendus, mis om kõigi poolt üheselt mõistetav. Selle teostamiseks peab andmehoidla koosseisus olema informatsiooni kataloog, milles kirjeldatakse ära iga andmehoidlas oleva atribuudi kindel tähendus. Selliseid “andmeid andmete kohta” nimetatakse tavaliselt metaandmestikuks.

ANDMEHOIDLA ANDMEBAAS

Kui vajalik informatsioon on lähtesüsteemidest eksporditud ja integreeritud, on ta valmis andmehoidlasse sisestamiseks. Idee on selline, et iga päev lisatakse selle päeva andmed eelnevatele, pikendades seeläbi andmete "ajalugu" veel ühe päeva võrra.

Kuidas ja mille põhjal siis andmehoidla andmebaas kujundatakse?
Vaja minevate andmete kirjeldamiseks kasutatakse dimensionaalse analüüsi peatükis teemaks olnud dimensionaalset andmemudelit, mida nimetatakse täht-skeemiks. Kuna relatsiooniline mudel toetab hästi dimensionaalset mudelit, siis võib luua relatsioonilise andmebaasiskeemi, mis peegeldab vahetult täht-skeemi.

Täht-skeemi keskpunktist saab relatsiooniline tabel, nagu ka igast analüüsi dimensioonist. Keskmist tabelit nimetatakse *faktitabeliks*, sest ta sisaldab endas kõiki fakte (näiteks müük), mida enamus päringuid hakkavad kasutama. Dimensioonidest saavad lihtsalt *dimensioonitabelid.*
Täht-skeemi võib tõlgendada kui mitut tabelit (dimensioonid), mis kõik osalevad üks-mitmele seosehulgas faktitabeliga. Seda võib näidata, võttes algupärase täht-skeemi ja kinnitades selle külge seosehulgad - "varesejalad" Joonisel 5. Sel joonisel ning järgneval objektide kirjeldusel on kindlustus asendatud üldistava mõistega toode. Kindlustuse puhul oleks objektide kirjeldus läinud liialt keeruliseks, mõisted kogus ja maksumus on lihtsamalt mõistetavad.



Joonis 5. Seosehulki faktide ja dimensioonide vahel näitav täht-skeem.

Objektide kirjeldused võivad näiteks olla järgmised (primaarvõtmesse kuuluvad atribuudid on allajoonitud):

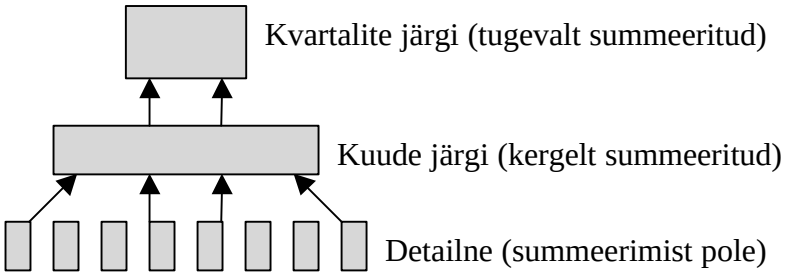
Müük (Kliendikood, Tootekood, Regioonikood, Ajatempel, Kogus, Maksumus)
Klient (Kliendikood, KliendiNimi, KliendiAadress)
Toode (Tootekood, Nimi, Tükihind, Kaal, Materjal , Värv…jne)
Regioon (Regioonikood, RegiooniKirjeldus)
Aeg (Ajatempel, Kuupäev, KuuNumber, KvartaliNumber, Aasta)

Joonise kohta selgituseks:
Seosehulkade kirjeldust ei ole vaja kirja panna. Täht-skeemis eksisteerib endastmõistetav seos faktide ja dimensioonide vahel.
Faktide primaarvõti koosneb kõikide dimensioonide primaarvõtmete ühendist.
Primaarvõtmesse mittekuuluvad atribuudid – Kogus ja Maksumus on faktitabeli tõelised faktid.

Individuaalsed read ei ole eriti tähenduslikud süsteemis, mis on loodud vastamaks strateegilistele küsimustele. Et tähenduslikku informatsiooni trendide ja muu sellise kohta saada, peab andmebaas oskama vastata küsimustele aja jooksul toimunud müügi kohta. Näiteks aruande saamiseks, mis näitab müüki toodete kaupa, peab konkreetse toote üksikud müügid kokku liitma, et kogusummat saada. See tähendab, et päring peab üle käima sadadest, tuhandetest või isegi miljonitest ridadest. See omakorda tähendab, et faktitabelis olevad faktidele (Kogus ja Maksumus) rakendatakse peaaegu alati mingit funktsiooni. Kõige sagedamini andmehoidlas kasutatavateks SQL päringukeele funktsioonideks on SUM() ja AVG() ehk summa ja keskmine.
Sellest ka üks reegel faktitabeli kohta: primaarvõtmesse mittekuuluvad veerud peavad olema summeeritavad.

Kui faktitabelis on mitukümmend miljonit rida (mis ei ole sugugi haruldane) ja päring peab igast viimasest kui reast üle käima, siis vastuse saamiseks ja selle kasutajale esitamiseks võib kuluda üsna kaua aega. Mitme samaaegse kasutaja ja keerulisemate päringute, mis mitut tabelit kokku ühendavad, puhul suureneb see aeg veelgi, kuni mitme tunnini. Põhimõtteliselt ei ole strateegilistele küsimustele küll kiireid vastuseid vaja, vastupidiselt päringutele, kus näiteks keegi teiselpool telefoni vastust ootab. See ei tähenda muidugi, et andmehoidla ehitajad ei peaks tegema kõike, mis võimalik, asjade kiirendamiseks.

Indeksite kasutamine võib aidata, aga suurem osa päringuid kasutavad rohkem kui poolt andmetest ja indeksite kasutamine ei ole nendel juhtudel kiirem kui kogu tabeli järjestikune skaneerimine. Lahendus on kokkuvõtetes ehk summatabelites. Kui on mingil määral võimalik ennustada, mis tüüpi päringuid kasutajad rohkem esitavad, siis võib ette valmistada mõned summatabelid. Kui sellised agregaattabelid nõutud infot ei suuda pakkuda, võib kasutaja ikka detailseid andmeid kasutada. Joonis 6 näitab summeerimise tasemeid, mida tavaliselt kasutatakse.



Joonis 6. Näide summeerimise tasemetest andmehoidlas.

Paar näidet summatabelitest:

- Kliendid toote järgi iga kuu jaoks
- Kliendid toote järgi iga kvartali jaoks
- Tooted regiooni järgi iga kuu jaoks
- Tooted regiooni järgi iga kvartali jaoks

Üks tehnika, mis on andmehoidla kasutajatele väga kasulik, on võimalus ühest summeerimise tasemest “puurida sügavamale” (*drill down*) järgmisele detailsemale tasemele. Näiteks on näha, et just üks tootegrupp on eriti edukas või vilets. Uurides sügavamalt üksikuid tooteid, on näha, kas kogu tootegrupp on mõjutatud või äkki ainult üks isoleeritud toode. Ümberpöördult, võimalusega “puurida madalamalt”(*drill up*) saab kindlaks teha, kas üks kehv toode ei mõjuta kogu tootegruppi.

Andmehoidla kasutamist peab jälgima, veendumaks, et andmehoidlat tarvitavad päringud ikka kasutavad summatabeleid. Kui tuleb välja, et ei kasutata, siis peaks neist loobuma ja asendama nad mõne teisega, millest rohkem kasu on.

SUMMA NAVIGAATOR

Summatabelite tarvitusele võtmine püstitab paar küsimust:

1. Kuidas kasutajad (eriti need, kes ei ole arvutispetsialistid) teavad, missugused summatabelid on saadaval ja kuidas neist kasu saada?
2. Kuidas jälgida, mis summatabeleid tegelikult üldse kasutatakse?

Üks lahendus on kasutada summa navigaatorit (*summary navigation tool*). Summa navigaator on lisakiht tarkvara, tavaliselt mingi kolmanda osapoole toode, mis paikneb kasutajaliidese (presentatsioonikihi) ja andmebaasi vahel. Ta võtab vastu kasutajalt SQL päringu ja uurib seda, et tuvastada, missuguseid veerge ja summeerimise taset on vaja.

Nende tööpõhimõte on hea näide metaandmestiku (andmed andmete kohta) kasutamisest. Summa navigaatoritel on oma metaandmestik, mida hoitakse kas andmehoidlas või ka kusagil eraldi asetsevas andmebaasis. Seda metaandmestikku kasutatakse kasutajate päringute ja andmehoidla vahelise “kaardistuse” saamiseks.

INFORMATSIOONI PRESENTATSIOON

Viimane andmehoidla komponent on presentatsiooni meetod. See on viis, kuidas andmehoidla on kasutajatele esitatud.

Enamus andmehoidlaid teostatakse klient-server konfiguratsiooni kasutades. Klient-serveri mõiste all on siin kontekstis mõeldud kasutaja eraldamist andmehoidlast, ehk kasutaja tarvitab PC-d ja andmehoidla asetseb kaughostil.

Relatsiooniliste andmebaasidele ligipääsuks on palju tooteid. Enamus neist aitavad kasutajal SQL lauseid genereerida, kasutades selleks RDBMS-i andmebaasiskeeme sisaldavaid tabeleid. Samuti on paljudel võimalus päringute tulemusi esitada mitmel eri kujul, nagu tekstilised aruanded, sektordiagrammid, histogrammid, kahe- ja kolmedimensionaalsed tulpdiagrammid jne. Arvukalt on ka neid, mis on ühilduvad veebiserveriga, mistõttu kasutajal on ainult vaja veebibrauserit info kuvamiseks.

Andmehoidlate leiutamisest alates on tekkinud aga ka mõned spetsialiseeritud analüüsimeetodid. Üks nendest on “andmete kaevandamine” (*data mining*). Otsuste toetamise küsimustes on koorem alati andmehoidla kasutajal olnud. Kasutaja on see, kes peab päringud formuleerima ja tulemuste iseloomu uurima.

Andmete kaevandamine on meetod, kus tehnoloogia teeb enamuse tööst ära. Kasutajad kirjeldavad andmeid andmete kaevandamise programmile määrates ära vajalikud andmetüübid ja valiidsed väärtused. Programm käivitatakse seejärel andmehoidla peal ja rakendades standartseid seoste äratundmise algoritme, otsib ja kuvab see andmetes olevate seoste detaile. Kasutajale ei pruugi need seosed teada olla.

Seda tehnikat on edukalt kasutatud kindlustuses, kus üks kindlustusfirma tahtis vähendada selliste elukindlustuse pakkumiste arvu, kus nõusoleku saamiseks pidi firma poole pöörduma. Kasutatud andmete kaevandamise programm teatas, et 30 – 40 aasta vanustel meestel, kelle pikkuse ja kaalu suhe mahtus kindlaksmääratud piiridesse, oli riski tõenäosus kõigest 0,015 võrra suurem. Firma lisas selle omaduse koheselt oma automaatse allakirjutamise süsteemi, tõstes seeläbi automaatse allakirjutamise taset 50 protsendilt 70 protsendini. Inimesest “andmete kaevandajal” oleks tõenäoliselt sellise seose märkamine väga kaua aega võtnud, sest inimene üritab andmeid loogilisi teid pidi ühendada, programm aga otsib lihtsalt tõenäosuslikke seoseid.

RELATSIOONILISTE ANDMEBAASIDE KASUTAMISE PROBLEEMID

Kuigi relatsiooniline mudel toetab andmehoidlate ehitamiseks vajalikke tingimusi, on mitmeid valdkondi, kus relatsioonilise mudeliga on raske hakkama saada. Järgnevalt nendest lühidalt.

Ajaga seotud probleemid

Ajateisendlikkus on andmehoidla üks tähtsamaid omadusi. Esmaspilgul paistab, et mõnes kohas dubleeritakse andmeid asjatult, näiteks kui kliendite andmed on kõik juba niigi operatsioonilistes süsteemides olemas. Sellist dubleerimist tehakse, sest on vajadus salvestada andmeid aja jooksul. Näiteks, kui kliendi aadress muutub, viiakse see muutus tõenäoliselt sisse ka operatsioonilisse andmebaasi. Seda tehes on eelmine aadress aga kaotatud. Kui pärast seda käivitada päring, kus selle kliendi detaile kasutatakse, omistatakse mistahes sellele kliendile müüdud tooted automaatselt uuele aadressile. Kui uurida müüki regiooni järgi, siis on tulemused ebaõiged (oletades et klient kolis erinevasse regiooni), sest paljud ostud sooritati siis, kui klient elas erinevas regioonis. Samal põhjusel ei saa ära kustutada klientide andmeid lihtsalt sellepärast, et nad ei ole enam kliendid. Kui nad on kasvõi ühe asja ostnud või tellinud, peavad nad süsteemis püsima.

Korraliku aja toetusega süsteemid on väga keerukad ja neil ei ole väga laia toetust. Täpsete tulemuste kindlustamiseks tuleb kasutada näiteks “alguskuupäev” ja “lõpukuupäev” mõisteid. Nendega muidugi ajaga seotud probleemid veel ei lõpe.

Probleemid SQL-ga

SQL baseerub hulgateoorial (*set theory*): ta käsitleb tabeleid kui hulki ja tagastab ka vastused hulkadena. Protseduurse loogika kasutamine aitaks mõnes kohas funktsionaalsust ja jõudlust parandada.

Järjestamine / Esimesed (*n*)

Kuigi SQL-i kasutades on võimalik saada järjestatud tulemusi, on see raskendatud. See nõuab korrelatiivset alampäringut, mis on väljaspool enamuse SQL-I kasutajate võimeid. Samuti on see väga aeganõudev. Mõned üksikud RDBMS-i müüjad pakuvad lisaomadusi, mis võimaldab selliseid päringuid kasutada, aga need ei ole standardiseeritud. See, mis ühe RDBMS-i peal töötab, tõenäoliselt teise peal enam ei tööta.

Esimesed *n* protsenti

Praktiliselt võimatu on saada näiteks nimekirja 10% klientidest, kes kõige enam oste sooritavad.

Jooksevsaldo

Samuti on peaaegu võimatu standartset SQL-i kasutades saada jooksevsaldot sisaldavat aruannet. Jooksevsaldo on nagu kontoteatis, kus esimeses veerus on maksed, teises tulud ja kolmandas bilanss, mille määravad kaks esimest veergu.

Keerulisem aritmeetika

Standartne SQL pakub küll põhilisi aritmeetilisi funktsioone, aga mitte keerulisemaid. Erinevad RDBMS-de müüjad varustavad oma süsteeme küll laiendustega selles vallas, aga need varieeruvad. Kui on näiteks vaja arvu astendada, siis osades süsteemides peab astendaja olema täisarv, teistes aga võib olla reaalarv. Kuigi andmehoidlaid kasutatakse statistiliste andmete saamiseks, puuduvad SQL-s võimalused tavaliste statistiliste mõistete nagu hälbed ja kvartiilid kui ka integraali ja diferentsiaali arvutamiseks.

Muutujad

Tavalises SQL-i päringus ei saa kasutada muutujaid.

Peaaegu neid kõiki ja ka muid puudusi saab likvideerida kirjutades programmid näiteks Java või C programmeerimiskeeles koos sinna sisse viidud SQL-ga. Samuti pakuvad paljud RDBMS-de müüjad probleemi leevendamiseks oma toodetele laiendusi protseduursete keelte toetuse näol. Standardliides presentatsioonikihi toodete ja RDBMS-de vahel – ODBC (*open database connectivity*) ja uuem JDBC (*Java database connectivity*) on kasulikud, sest sunnivad IT-tööstusele peale standartset lähenemist. Kirjutamise ajal ei toeta ta küll veel RDBMS-de müüjate poolt pakutud protseduursete keelte laiendusi.

KOKKUVÕTE

Andmehoidlad on eri liiki andmebaasid, mis on ehitatud spetsiaalselt informatsiooni kättesaamiseks, mitte sissepanemiseks, mis on enamuse rakendusprogrammide andmebaaside ülesandeks.

Rõhk on seatud strateegilise iseloomuga küsimusetele vastamises, et aidata mingi organisatsiooni juhtkonnal tuleviku jaoks plaane teha.
Andmehoidla on:

- Teemale orienteeritud
- Muutumatu, püsiv
- Integreeritud
- Ajateisendlik

Dimensionaalne analüüs on meetod, mida kasutatakse andmehoidla nõudmiste identifitseerimiseks ja mida sageli kujutatakse täht-skeemi abil. Täht-skeem näitab ära faktid ja analüüsi dimensioonid. Fakt on atribuut, nagu näiteks müügi väärtus või kõne pikkus, mida analüüsitakse üle dimensioonide. Dimensioonid on sellised objektid nagu kliendid ja tooted, mille kaudu fakte analüüsitakse. Tüüpiline päring oleks:

Näita selle kuu ja eelmise kuu toodete müügi väärtust klientide kaupa.

Aeg on alati üks analüüsi dimensioonidest.
Andmehoidla hoitakse peaaegu alati tavaliste rakendusprogrammide andmebaasidest eraldi, sest:

1. Rakendusprogrammide andmebaasid on optimeeritud käivitamaks *insert* ja *update* tüüpi päringuid, kui andmehoidlad on optimeeritud *select* tüüpi päringute käivitamiseks.
2. Rakendusprogrammide andmebaasid on pidevalt muutuvad, kui andmehoidlad on “vaiksed” (muutumatud, püsivad).
3. Rakendusprogrammide andmebaasidel on suured ja keerulised skeemid, kui andmehoidlatel on lihtsustatud, tihti denormaliseeritud, struktuurid.
4. Andmehoidlad vajavad ajaloolist informatsiooni ja see on tavaliselt rakendusprogrammide andmebaasidest puudu.

Tavalisel andmehoidlal on viis põhikomponenti:

1. Lähteandmete eraldamine hulgast rakendusprogrammide andmebaasidest. Need lähtesüsteemid kasutavad tihti väga erinevaid tehnoloogiaid.
2. Andmete integratsioon. Eksisteerib kahte tüüpi integratsiooni. Esimene on formaadi integratsioon, kus loogiliselt sarnased andmetüübid (näiteks kuupäevad) konverteeritakse nii, et neil oleks sama füüsiline andmetüüp. Teine, semantiline integratsioon, kindlustab andmete tähenduse kokkusobivuse.

3. Andmebaas ise. Andmehoidla andmebaas võib kasvada tohutuks, kui uus kiht faktilisi andmeid lisatakse iga päev. Täht-skeem teostatakse rea tabelitena. Faktitabel (tähe keskpunkt) on pikk ja peenike selles mõttes, et sisaldab suure arvu ridu ja väikse arvu veergusid. Faktiveerud peavad olema summeeritavad. Dimensioonitabelid (tähe tipud) ühendatakse faktitabelitega võõrvõtmete abil.
4. Summa navigeerimine on meetod, mis võimaldab kasutajate päringud automaatselt suunata summatabelitele, ilma et nad teaksid selle toimumisest. See on päringu kiiruse koha pealt väga tähtis.
5. Informatsiooni presentatsioon. See on viis, kuidas informatsioon kuvatakse andmehoidla kasutajatele. Enamus teostusi on valinud klient-serveri lähenemise, mis annab kasutajatele võimaluse vaadelda oma informatsiooni mitmekesise valiku tabelite või graafiliste formaatidena.

Andmehoidlad on samuti kasulikud allikad “andmete kaevandamise” (*data mining*) programmidele, mis on suuri andmebaase skaneerivad ja seoseid otsivad tarkvaratooted. Esineb ka probleeme, mida ületama peab, näiteks aja kasutamine. Samuti ei saa paljusid päringuid, mida kasutajad andmehoidlalt tüüpiliselt küsida tahaksid, kergesti standartsetesse SQL päringutesse tõlkida ja peab kasutama muid meetodeid, näiteks protseduurseid programme koos sinna sisse viidud SQL-ga.

KASUTATUD KIRJANDUS

A dictionary of computing, New York: Oxford University Press, 1996, 550 pg.

Hanson, V., Tavast, A. Arvutikasutaja sõnastik, Tallinn: Kirjastus Ilo, 1996, 220 lk.

Orr, Ken. Data Warehousing Technology, White Paper, *www.kenorrinst.com/dwpaper.html*, 22/02/2002.

Todman, Chris. Designing a data warehouse: supporting customer relationship management, Upper Saddle River: Prentice Hall PTR, 2001, 323 pg.

Berson, A., Smith, Stephen J. Data warehousing, data mining & OLAP, McGraw-Hill, 1997, 612 pg.