# TALLINNA PEDAGOOGIKAÜLIKOOL

**Matemaatika-loodusteaduskond**
**Informaatika osakond**

**Erik Iter**


# ANDMEHOIDLAD TEOORIAS JA PRAKTIKAS

**Diplomitöö**


**Juhendaja: Mihkel Märtin**


Autor: …… "…" ……. 2004
Juhendaja: ……...……. "…" …..…… 2004
Osakonna juhataja: …….. "…" …… 2004

**TALLINN 2004**

# Sisukord

# Sissejuhatus

Tänapäeva äriolustikus tunnevad ettevõtted järjest kasvavat survet vähendada kulusid ja suurendada kasumit ning eelisolukorras on need ettevõtted, mis suudavad kohaneda kiiresti muutuvate nõudmistega. Töötajad, kes peavad ärilisi otsuseid tegema, ei oma tihti selleks piisavalt informatsiooni. Suured andmehulgad, mida ettevõte oma töö käigus kogub, ei jõua tagasi kasutajani kasutatava informatsiooni kujul. Enamus sellisest informatsioonist, mida on vaja teadlike ja põhjendatud otsuste tegemiseks, on ettevõtte igapäevaselt kasutatavates, niinimetatud operatiivsetes süsteemides luku taga. Ettevõtted on teinud suuri investeeringuid nendesse süsteemidesse, et oma äritegevuse andmeid jälgida ja säilitada, kuid sellised süsteemid tüüpiliselt ei võimalda lõppkasutajal andmetele kergelt ligi pääseda ja neid analüüsida.

Andmehoidla (ingl k *data warehouse*) on võti, mis võib avada ukse sellesse informatsiooni ja seeläbi tõsta ettevõtte konkurentsivõimet. Andmehoidla koondab ettevõtte äriandmed tsentraalsesse, integreeritud andmebaasi, mis võimaldab analüüsida ka mineviku ehk ajaloolisi andmeid. Kasutajatel on seeläbi kiire ja otsene moodus uurida äriandmeid.

Andmehoidla kõrval võib kohata mitmeid teisi eestikeelseid vasteid *data warehouse* mõistele – andmeait, andmeladu, andmevaramu. Kuna infotehnoloogia terministandard seda mõistet ei sisalda, kasutatakse käesolevas töös üht levinumat, sõna andmehoidla [5] [8].

Esmapilgul võib andmehoidla paista üsna lihtsa rakendusprogrammina. Tegelikkuses on ta kõike muud kui lihtne. Seda nii suuruse (andmehoidla andmebaasid on ühed suurimaist maailmas, juba 1997 aastal oli keskmine suurus 272 gigabaiti) ja sellest tulenevate jõudluseprobleemide kui ka andmete struktuuri tõttu [2]. Käesolev töö peakski selgitama andmehoidlate olemust, kuidas neid kasutatakse ning praktilise näite varal nende arendamise juures tekkida võivaid probleeme.

Käesolev töö oletab, et lugejal on vähemalt mingid baasteadmised relatsiooniteooriast ning SQL päringukeelest ja seetõttu nende tehnilisi üksikasju ja süntaksit lähemalt ei seletata. Rohked näited töö esimeses, seletavas osas, peaksid siiski tunduvalt kergendama andmehoidlate kontseptsiooni mõistmist.

# 1. Ülevaade andmehoidlatest

Käesolevas peatükis antakse ülevaade andmehoidlatest. Vastused leitakse järgmistele küsimustele:

- Mis on nõustussüsteemid ja kuidas nad on seotud andmehoidlatega?
- Mis erinevus on operatiivse ja strateegilise informatsiooni vahel?
- Mis on andmehoidla definitsioon ja omadused?
- Millised on andmehoidla kasutajatüübid?
- Missugune on andmehoidla üldine arhitektuur?

## 1.1. Nõustussüsteemid

Andmehoidlad on tihedalt seotud üldise äriterminiga, mis on tuntud kui otsuste toetamine (ingl k *decision support*). Otsuste toetamine omakorda võib tugineda nõustussüsteemidele. Selleks, et mõista andmehoidlate põhimõtet, peab enne mõistma nõustussüsteemide (ingl k *decision support system* – DSS) üldist otstarvet [5].

Nõustussüsteemi otstarve on varustada ettevõtte või organisatsiooni otsusetegijaid neile vajaliku informatsiooniga. See informatsioon omakorda edendab ja suurendab otsusetegijate teadmisi mingil viisil, aidates neil organisatsiooni strateegia ja tegutsemisviiside kohta otsuseid vastu võtta.

Nõustussüsteemid on erineval kujul eksisteerinud palju aastaid. Ammu enne esimeste andmebaasihaldurite (ingl k *database management system* - DBMS) leiutamist eraldati programmidest infot, et aidata juhtkonnal efektiivsemalt oma ettevõtet juhtida. Andmehoidlate seos nõustussüsteemidega seisneb selles, et tänapäevased nõustussüsteemid baseeruvad suures osas andmehoidlatel.

### 1.1.1. Eesmärgid, omadused, näited

Nõustussüsteem omab tavaliselt järgmisi karakteristikuid:

- On peamiselt suunitletud lahendama probleeme, mida ei saa hästi määratleda ega sõnastada ja millega tüüpiliselt tegeleb kõrgem juhtkond.
- Omadused, mis võimaldavad süsteemi interaktiivselt kasutada ka infotehnoloogiaga mittetegelevatel inimestel.
- Nad on küllalt paindlikud ja kohandatavad, et lubada teha muudatusi kasutuskeskkonnas ja lähenemisviisil otsuste tegemisele.

[7]

Nõustussüsteemi ülesanne on tavaliselt pakkuda faktilisi vastuseid kasutaja poolt formuleeritud küsimustele. Näiteks müügijuhti paneks kindlasti muretsema fakt, kui toodete müük jääb ülemuse poolt määratud eesmärkidest väiksemaks. Küsimus, mida ta kindlasti küsida tahaks, võiks olla:

*Miks on mu toodete müük seatud eesmärkidest väiksem?*

Tänapäeval ei eksisteeri veel ühtki arvutisüsteemi, mis võimaldaks sellisele küsimusele vastata. Raske on ka ette kujutada vastavat SQL (*Structured Query Language*) keele päringut, mis nõutuga hakkama saaks. Esitatud küsimused peavad olema süstemaatilisemad, et nõustussüsteemil oleks võimalik anda faktilisi vastuseid. Esimene küsimus võiks näiteks olla:

*Mis on iga toote kogumüük ja seatud eesmärk sellel aastal?*

Nõustussüsteem tagastaks vastuseks toodete ja nende müügisummade nimekirja. 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 ettevõtte turuosa nende toodete osas ja kas see turuosa väheneb. Kui see on nii, siis võib põhjuseks olla näiteks hiljutine hinnatõus. Nõustussüsteemi otstarve on vastata sellisetele niiöelda *ad hoc* (lad k, selle asja jaoks) küsimustele, et kasutaja jõuaks lõpuks mingile järeldusele ja oleks võimeline otsustama.

Suur piirang nõustussüsteemide arendamises on andmete kättesaadavus ehk juurdepääs õigetele andmetele õigel ajal. Kuigi andmebaasissüsteemide kiire levik ja arenemine on seda probleemi küll leevendanud, valmistab andmete kättesaadavus isegi tänapäeval enamuses organisatsioonides probleeme. Peamine põhjus on see, et organisatsioonid arenevad aja jooksul ning enamasti tekib vajadus muuta ka arvutisüsteeme. Mingil hetkel tekib peaaegu kõigi rakendusprogrammide elutsüklis olukord, 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 kasutavad arendajad ära kõikvõimalikud tehnoloogilised uuendused, mis selle programmi eluajal on leiutatud. Näiteks võis algne programm kasutada andmete säilitamiseks tekstifaile ja olla kirjutatud programmeerimiskeeles COBOL või RPG, sest see oli tollal sobiv tehnoloogia. Tänapäeval on üldlevinud viis andmete säilitamiseks aga relatsioonbaasihaldurid (ingl k *Relational Database Management System* – RDBMS) ja eelmainitud programmeerimiskeeled on samuti uuemate varju jäänud. 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 tarkvaratehnoloogiaid ja seetõttu on andmete kättesaadavus raskendatud.

Lisaks paiknevad suurte organisatsioonide süsteemid ka mitmesugustel erinevatel riistvaraplatvormidel. Üsna üldine on olukord, et ühe ettevõtte rakendusprogrammid laotuvad üle järgmiste platvormide:

- Suurarvuti (ingl k *mainframe*)
- Mitu väiksemat multiprotsessoriga masinat
- Välised teenusepakkujad
- Võrku ühendatud ja autonoomsed (ingl k *stand-alone*) PC tüüpi arvutid

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

### 1.1.2. Operatiivne ja strateegiline informatsioon

Tähtis on ka mõista strateegilise informatsiooni ja operatiivse informatsiooni erinevusi. Andmehoidla saab aidata strateegiliste küsimuste juures, mis käsitlevad üldiselt planeerimist ja tegevussuundade kujundamist.

Paar näidet strateegilistest otsustest:

- Kui telekommunikatsioonifirma otsustab lisaseadmete paigaldamise asemel suurema koormusega toime tulemiseks tarvitusele võtta väga odavad tipptunnivälised telefonitariifid, et meelitada inimesi sel ajal rääkima.
- Suur kaubamajade kett otsustab avada oma poed ka pühapäeval.
- Üldine 20 % hinnalangus ühe kuu jooksul, et suurendada turuosa.
- Kindlustusfirma otsustab lisaks muule varakindlustusele hakata pakkuma ka autokindlustust.

Kui strateegilised probleemid on seotud planeerimise ja poliitikaga, siis operatiivsed küsimused tegelevad rohkem organisatsiooni või äri igapäevaste protsessidega. Operatiivseid tegevusi võib pidada organisatsiooni strateegia rakendamiseks.
Igapäevane tarvikute tellimine, klientide tellimuste täitmine ja uute töötajate palkamine on kõik näited operatiivsetest protseduuridest. Neid protseduure toetab tavaliselt tarkvara ja seetõttu peab see tarkvara oskama vastata ka operatiivsetele küsimustele nagu:

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

Tüüpiliselt saab operatiivsete süsteemide abil sellistele küsimustele kergelt vastused leida, sest need on küsimused praegusel hetkel kehtiva situatsiooni kohta. Kõigile esitatud

küsimustele on võimalik lisada lõppu sõnad "praegusel hetkel" ja ikka oleksid need küsimused loogilised. Sellised küsimused tekivad organisatsiooni normaalse, igapäevase tegevuse käigus. Aga sellist liiki küsimused, mida firma direktorid ja juhtkond tahaks küsida, on:

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

Need ei ole selgelt "praegusel hetkel" tüüpi küsimused ja harilikult operatiivsed süsteemid ei oska sellistele küsimustele vastata. Põhjus seisneb nende süsteemide olemuses. Nad on arendatud toetamaks organisatsiooni igapäevaseid tegevusi. Võib öelda, et operatiivsed 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 lausetüübid). Kui süsteemid korraks paigale tarduksid, näitaksid nad organisatsiooni seisundi peegeldust täpselt sellel momendil. Üks sekund varem või hiljem võib olukord olla juba muutunud.

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

*Milliste tooterühmade populaarsus kasvab ja milliste oma langeb?*

See on ilmselt mõistlik strateegiline küsimus paljudes firmades. Olenevalt vastusest, võib juhtkond vastu võtta ühe järgnevaist otsustest:

- Suurendada mõne toote levikut ja vähendada teiste toodete oma
- Pakkuda tooteid vähendatud hindadega või hinnaalandust hulgi ostes
- Investeerida nende toodete reklaami, mille populaarsus on langemas

Ainus viis, kuidas on võimalik hinnata, kas tooterühma populaarsus kasvab või kahaneb, on jälgida selle nõudlust aja jooksul. Juhul, kui tellimuste töötlemise informatsiooni

hoitakse relatsioonandmebaasis, saaks koostada näiteks SQL päringu, mis kogub kokku kõik käesoleval kuupäeval müüdud tooted ja nende koguväärtused. Oskus leida kõigi tänaste tellimuste väärtus on hea algus. See on kasulik informatsioon, aga tegelikult tahetakse uurida näiteks viimase kuue kuu trendi või võrrelda selle kuu tulemusi eelmise aasta sama kuuga.
Lahendus oleks - käivitada sama päringut iga päev ja lisada iga päeva kohta saadud tulemused andmebaasitabelisse. Nii saaks aja jooksul üles ehitada vajaliku ajaloolise informatsiooni. See on andmehoidla algus.

## 1.2. Andmehoidla definitsioon ja karakteristikud

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

1. Teemale orienteeritud (ingl k *subject oriented*)
2. Püsiv (ingl k *nonvolatile*)
3. Integreeritud (ingl k *integrated*)
4. Ajateisendlik (ingl k *time variant*)

[7]

Teemale orienteeritus tähendab, et andmed on koondatud ümber teemade (nagu müük), mitte ümber operatiivsete tegevuste (nagu tellimuste töötlemine). Operatiivsed andmebaasid on mõeldud äriotstarbeliste programmide toetuseks ehk on programmidele orienteeritud (ingl k *application oriented*). Võimalikud andmehoidla teemad oleks näiteks telefonikõnede maksumus telekommunikatsioonifirma puhul või kogutud preemiad kindlustusfirmas.

Püsivuse all mõeldakse, et kord andmehoidlasse paigutatud andmed tavaliselt enam ei muutu. Andmehoidla kasutaja võib kindel olla, et päring annab alati sama vastuse olenemata sellest, kui tihti seda käivitatakse. Operatiivsetes andmebaasides olevad andmed ei ole püsivad ja muutuvad pidevalt. Ebatõenäoline on, et sama päring annab kaks korda sama vastuse, kui ta kasutab tabeleid, mida tihti uuendatakse.

Integreeritus tähendab, et andmed on ühilduvad. 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 märgistik (ingl k *character* set). Enamik süsteeme kasutavad ASCII (*American Standard Code for Information Interchange*) märgistikku, aga mõned mitte. Ühe suurima arvutitootja maailmas, IBM-i, kõik suuremad serverisüsteemid ja ka paljud väiksemad baseeruvad täiesti erineval märgistikul, mida nimetatakse EBCDIC (*Extended Binary Coded Decimal Interchange Code*). Nii on tähel "K" ASCII märgistikus numbriline väärtus 75, aga EBCDIC-s hoopis 210 (väärtusega 75 olev märk on EBCDIC-s "."). ASCII sõna "Kool" tõlgendub EBCDIC-s kujule ".??%" ning on raskem midagi vähem integreeritumat ette kujutada.

Teised erinevused on väiksemad, näiteks kuupäevade talletusformaadid. Veel raskemini märgatavad erinevused leiduvad sama tehnoloogiat kasutatavates programmides. See juhtub, kui näiteks üks programmeerija otsustab hoida kliendi aadressi viies 25 märgi pikkuses veerus, aga teine kasutab ühtainsat veergu ja 100 märgi pikkust formaati. Enne kui andmed võib andmehoidlasse sisestada, peavad nad olema integreeritud. Integratsioon on järelikult protsess, mida andmed läbivad pärast operatiivsest süsteemist lahkumist ja enne andmehoidlasse sisenemist.

Ajateisendlikkus tähendab, et salvestatakse ka ajaloolisi andmeid ehk andmeid mineviku kohta. Peaaegu kõik päringud, mis andmehoidlast andmeid loevad, on seotud kuidagi ajaga. Enam-vähem võimatu on minevikku vaatlemata ennustada, mis tulevikus juhtub. Andmehoidla aitab sellist fundamentaalset probleemi lahendada, lisades operatiivsetest andmebaasidest võetud andmetele ajaloo dimensiooni juhul, kui see puudub. Näiteks allolevas tabelis on ajaloolisteks andmeteks esimene rida, mis kehtis ajavahemikul 01.01.2000 kuni 01.12.2000. Teine, praegu kehtiv rida on niiöelda jooksev informatsioon.

| ***Nimi*** | ***Aadress*** | ***Alguskuupäev*** | ***Lõpukuupäev*** |
|---|---|---|---|
| Mihkel | Vesiroosi 20 | 01.01.2000 | 01.12.2000 |
| Mihkel | Nurmenuku 14 | 02.12.2000 | 31.12.9999 |

### 1.2.1. Andmehoidla kasutajad

Üks andmehoidlate kohta levinud väärarvamusi on, et kui see valmis ehitada, küll siis ka

kasutajaid leidub. Kuid lihtsalt andmehoidlasse laetud andmed ei aita kasutajal teadmisi omandada. Vastupidiselt ei ole kasu ka päringutarkvarast ilma andmeteta. Samuti on andmetele ligipääsuks mõeldud päringutarkvara tihti äriinimestele kasutamiseks liiga keeruline [9].

Järgnevalt ülevaade erinevatest andmehoidla kasutajatüüpidest ja nende käitumismallidest:

- Juhtivtöötaja: tahab hõlpsalt ligipääsetavat informatsiooni ettevõtte olukorra kohta. Vajab eelnevalt defineeritud raportite hulka, mida saab kergesti menüüde kaudu leida ja kaldub eelistama kasutajaliideses suuri nuppe, mis ärifunktsioonidele viitavad. Tulemused peaksid olema kuvatud graafiliselt. Juhul kui soovib lisaanalüüsi, siis on tihti järgmine samm telefonikõne või e-kiri sobivale inimesele või osakonnale.
- Algaja / juhuslik kasutaja: keegi, kes vajab andmetele ligipääsu suhteliselt harva. Ei kasuta süsteemi iga päev või isegi iga nädal. Tänu suurtele ajavahemikele kasutamiskordade vahel vajab samuti kergestikasutatavat liidest. See kasutajatüüp võib olla huvitatud ka raportite parameetrite muutmisest, aga vajab selleks abistavaid dialoogiaknaid.
- Ärianalüütik: üks paljudest lõppkasutajatest, kes kasutab informatsiooni iga päev, kuid ei oma (ega tingimata ka soovi omada) vajalikke tehnilisi teadmisi, et raporteid täiesti algusest peale luua. Muudab regulaarselt parameetreid, et tulemusi erinevast aspektist vaadelda.
- Lauskasutaja (ingl k *power user*): kasutaja, kellele meeldib tehnoloogia. Võib kasutada arvutit aktiivselt ka väljaspool tööaega. Seda tüüpi kasutaja tahab kindlasti muuta kõikvõimalikke parameetreid, manipuleerida tulemustega ja võib luua nii endale kui teistele raporteid/analüüse täiesti algusest peale.

## 1.3. Andmehoidla üldine arhitektuur

Võtmeküsimused andmehoidla loomisel on järgmised:

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

Tavaliselt luuakse andmehoidla kõikidest teistest rakendustest eraldi, iseseisva rakendusena. Üks ilmne põhjus on põhjalikud erinevused lähtesüsteemide olemustes (erinevad platvormid, operatsioonisüsteemid, tarkvaraarhitektuurid). Kuid on ka teisi põhjusi:

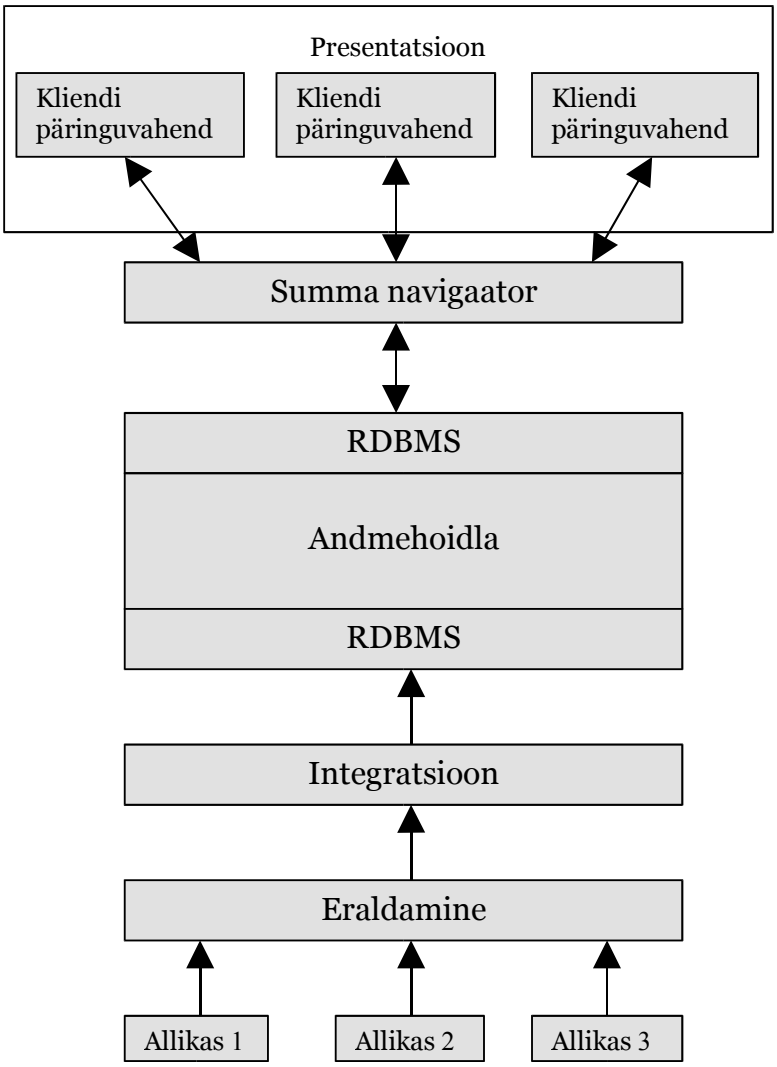
1. Operatiivsete ja andmehoidla süsteemide vahel on konflikt – esimesed toetavad operatiivseid nõudmisi ja ei ole teemale orienteeritud, teised aga toetavad analüütilisi nõudmisi ning on teemale orienteeritud.
2. Operatiivsetes süsteemides olevad andmed muutuvad pidevalt, aga nõustussüsteemide andmed on püsivad.
3. Operatiivsete süsteemide andmebaasiskeemid on tihti väga keerulised, suure arvu erinevate tabelitega. SQL keele päring peab sellisest süsteemist andmete lugemiseks kokku ühendama (ingl k *join*) mitmeid tabeleid, mis teeb päringud aeglaseks ja samuti päringute kirjutamise keerulisemaks. Andmehoidla andmebaasiskeem on tunduvalt lihtsam ja selgem.
4. Andmehoidla mõju teistele samas masinas töötavate rakenduste jõudlusele ei oleks arvatavasti kasutajatele aktsepteeritav ega meelepärane. Strateegilistele küsimustele vastuse saamiseks kuluvad mitu minutit või isegi tundi on tavaliselt vastuvõetavad. Operatiivsetes süsteemides tehtavad toimingud aga ei tohi harilikult rohkem kui paar sekundit aega võtta.

Veel üks põhjus süsteemide eraldi hoidmiseks on see, et operatsioonilised süsteemid tavapäraselt ei säilita ajaloolisi andmeid. Soovides ajaloolist informatsiooni juurutada, peab olemasolevat programmi mingil viisil muutma, sageli on muudatused mastaapsed. Enamasti ei ole organisatsioonid valmis taluma häireid, mis sellega kaasnevad.

*De facto* standardiks andmehoidlate arendamise tehnoloogiate vallas on saanud relatsioonbaasihaldurid. Osaliselt on selle põhjuseks see, et andmehoidlate arendamisel kasutatakse tavaliselt dimensionaalset modelleerimist ja relatsiooniline andmemudel toetab dimensionaalset mudelit väga hästi. Dimensionaalsest mudelist on juttu edaspidi, peatükis 2.1.1.

Andmehoidla koosneb mitmetest komponentidest. Osad on seotud informatsiooni eraldamisega lähtesüsteemidest ja selle andmehoidlasse sisestamisega, teised aga on seotud

informatsiooni kättesaamisega andmehoidlast ja selle kasutajale kuvamisega. Joonisel 1 on ära toodud põhikomponendid [7].




**Joonis 1.** Andmehoidla süsteemi põhikomponendid.

# 2. Andmehoidla arendusprotsess

Töö selles osas on pikemalt juttu andmehoidla süsteemi komponentidest ja nende arendamisega kaasnevatest küsimustest ning probleemidest. Samuti tutvustatakse enamlevinud andmemudeleid ja modelleerimise tehnikaid. Ülevaade antakse ka andmehoidlatega seotud terminitest - metaandmed, andmevakad, andmete kaevandamine, OLAP (*Online Analytical Processing*).

## 2.1. Andmehoidla andmebaas

Kuidas ja mille põhjal siis andmehoidla andmebaas kujundatakse?
Üks edukamaid andmehoidla kavandamise tehnikaid on dimensionaalne analüüs ja dimensionaalne modelleerimine (ingl k *dimensional analysis and dimensional modelling*).

### 2.1.1. Modelleerimine

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 kui otsusetegijatel paluti kirjeldada oma organisatsiooni puudutavaid küsimusi, millele nad vastuseid sooviksid, siis peaaegu alati soovisid nad järgnevat:

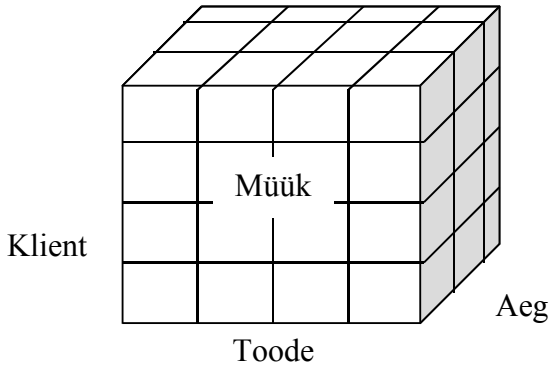
- Summeeritud informatsiooni koos võimalusega summade sisu detailsemalt vaadelda.
- Summeeritud informatsiooni analüüsi oma organisatsiooni komponentide nagu osakonnad või regioonid kohta.
- Võimalust "lõigata ja tükeldada" informatsiooni igal neile sobival ja mõeldaval viisil.

- Informatsiooni kuvamist nii graafiliselt kui tabelitena.
- Võimalust vaadelda informatsiooni mingi ajaperioodi kohta.

[7]

Näitena võib tuua postimüügifirma, kes soovib aruandeid oma toodete müügi kohta nii toote kaupa kui ka kliendi kaupa, või isegi müüki mõlema - toote ja kliendi kaupa.

Dimensionaalse analüüsi lähenemine on organisatsiooni sobivaid otsusetegijaid intervjueerides kindlaks teha, mis teemast nad kõige rohkem huvitatud on ja mis on tähtsamad 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 2 oleva kolmedimensionaalse kuubina:




**Joonis 2**. Kolmedimensionaalne andmekuup.

Joonis 2 näitab müüki koos järgmiste telgedega:

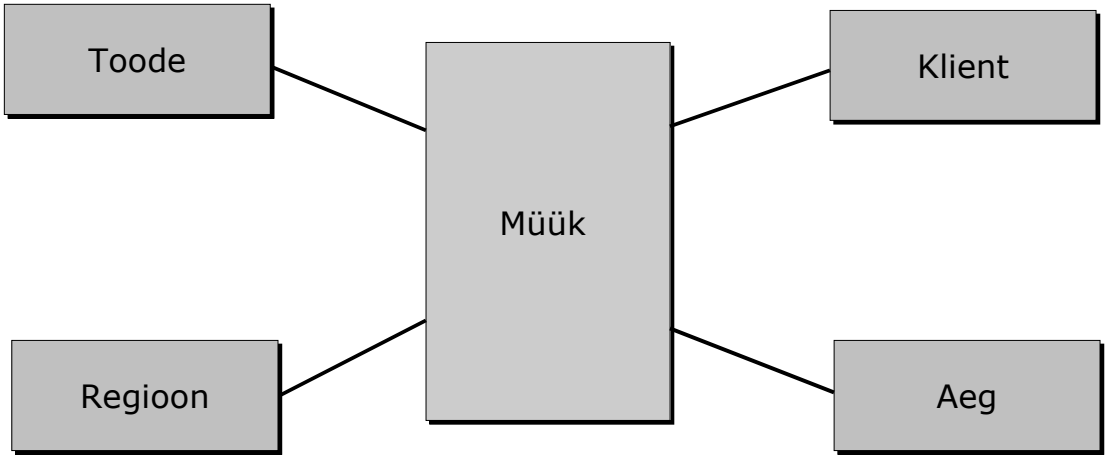
1. Klient
2. Toode
3. Aeg

Aega suhtutakse kui kohustuslikku 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. Iga kuubi element (iga minikuup) sisaldab seega konkreetse toote müügi väärtust konkreetsele kliendile mingil kindlal ajahetkel.

Multidimensionaalne kuup joonisel 2 näitab teemana müüki koos kolme analüüsi dimensiooniga. Tegelikku limiiti mudelis kasutatavate dimensioonide arvul ei ole, küll aga on piir dimensioonide arvul, mida on võimalik joonistada.

#### 2.1.1.1. Täht-skeem

Oletame, et postimüügifirma direktorid vajavad lisaks ka raporteid müügi kohta regiooni kaupa. Analüüsi dimensioonid on seega toode, klient, regioon ja aeg. Kuna neljadimensionaalset mudelit joonistada ei saa, võib esitada kontseptuaalse dimensioonilise mudeli nagu on näidatud joonisel 3.




**Joonis 3**. Näide dimensionaalsest mudelist müügi jaoks.

Joonisel 3 näidatud diagrammitüüpi nimetatakse täht-skeemiks (ingl k *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, kuid väga suure hulga ridu. Täht-skeem on kõige sagedamini kasutatav diagramm dimensionaalsete mudelite juures [7].

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.

Joonisel kuvatud olemitele vastavad loogilised andmebaasiskeemid võivad näiteks olla järgmised (primaarvõtmesse kuuluvad atribuudid on allajoonitud):

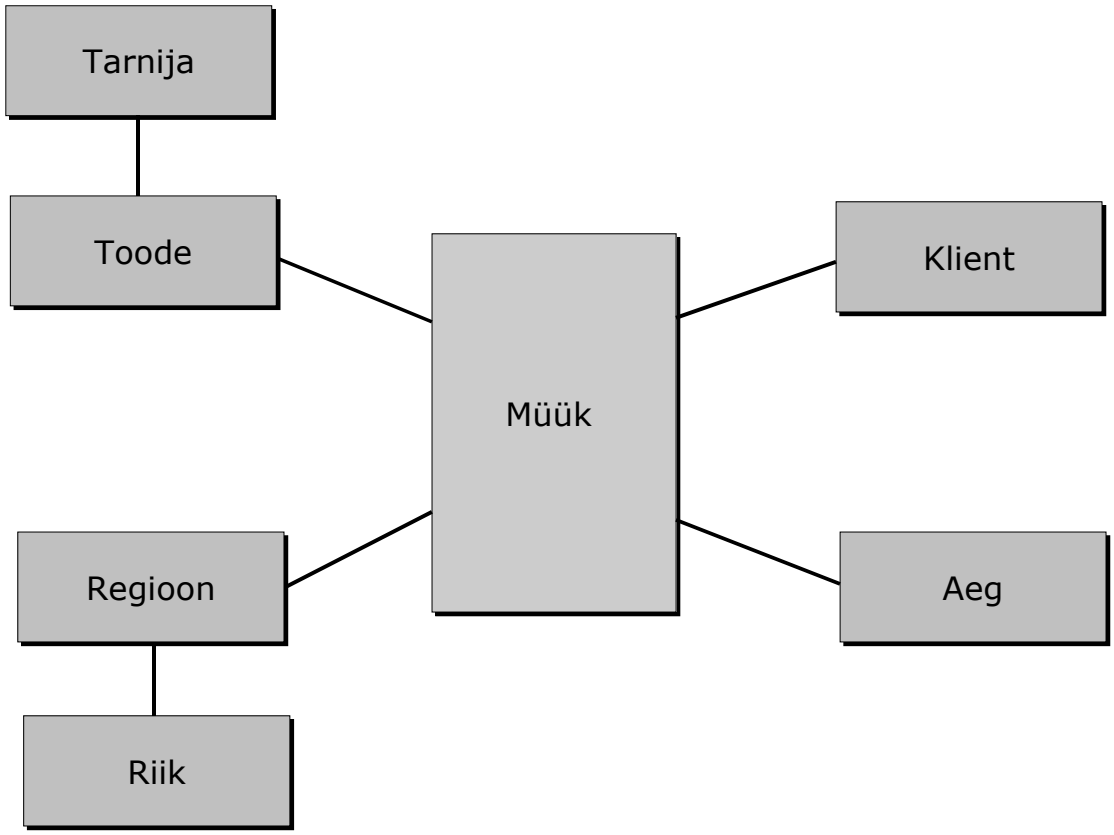
**Müük-skeem** ={ Kliendikood, Tootekood, Regioonikood, Ajatempel, Kogus, Maksumus }
**Klient-skeem** = { Kliendikood, KliendiNimi, KliendiAadress }
**Toode-skeem** = { Tootekood, Nimi, Tükihind, Kaal, Materjal, Värv }
**Regioon-skeem** = { Regioonikood, RegiooniKirjeldus }
**Aeg-skeem** = { Ajatempel, Kuupäev, KuuNumber, KvartaliNumber, Aasta }

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

Sellises süsteemis, mille eesmärgiks on vastata strateegilistele küsimustele, ei oma individuaalsed read kuigi suurt tähendust. Andmehoidla peab oskama vastata küsimustele aja jooksul toimunud müügi kohta, et saada tähenduslikku informatsiooni klientide ostuharjumuste kohta. Näiteks aruande saamiseks, mis näitab müüki toodete kaupa, peab konkreetse toote üksikud müügid kokku liitma. See tähendab, et päring peab andmeid lugema sadadest, tuhandetest või isegi miljonitest ridadest. Sellest omakorda tuleneb, et faktitabelis olevatele faktiveergudele (Kogus ja Maksumus) rakendatakse peaaegu alati mingit agregaatfunktsiooni. Kõige sagedamini andmehoidlas kasutatavateks SQL päringukeele agregaatfunktsioonideks on SUM() ja AVG() ehk summa ja keskmine.
Sellest tulenevalt ka üks reegel faktitabeli kohta: primaarvõtmesse mittekuuluvad veerud peavad olema summeeritavad või allutatavad mingile muule agregaatfunktsioonile.

#### 2.1.1.2. Lumehelves-skeem

Lumehelves-skeem on sisuliselt täht-skeemi keerulisem versioon. Nime on ta saanud selle järgi, et skeemi esitav diagramm meenutab kujult lumehelvest. Lihtne seda tüüpi diagramm, joonisel 3 olnud täht-skeemi laiendus, on kuvatud joonisel 4.



**Joonis 4**. Lumehelves-skeemi näide.

Lumehelves-skeemi saab luua, normaliseerides täht-skeemi dimensioonid eraldi mitmesse tabelisse ühe suure tabeli asemel. Normaliseerimine on protsess, mille üks eesmärkidest on elimineerida andmete liiasus, aga selle sügavam käsitlemine ei mahu käesolevasse töösse.

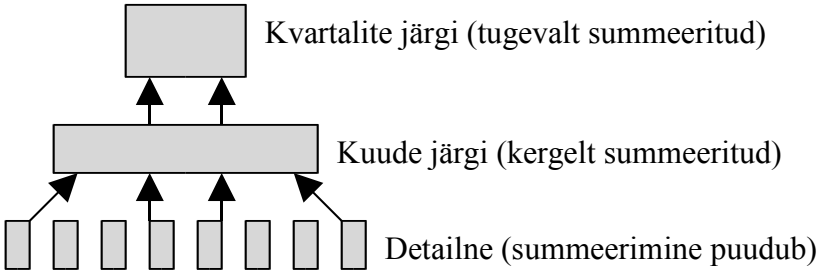
Lumehelves-skeemitüüp peegeldab täpsemini aluseks olevaid äriprotsesse ning tänu normaliseerimisele võtab füüsiliselt vähem ruumi. Suurimaks miinuseks on see, et päringutes tuleb rohkem tabeleid võõrvõtme abil ühendada ja see võib mõjuda halvavalt jõudlusele.

#### 2.1.1.3. Summeerimine

Kui faktitabelis on mitukümmend miljonit rida (mis ei ole sugugi haruldane) ja päring peab lugema andmeid kõikidest ridadest, siis vastuse saamiseks ja selle kasutajale esitamiseks võib kuluda üsna palju arvutusressurssi. Mitme samaaegse kasutaja ja keerulisemate

päringute puhul, kus mitu tabelit kokku ühendatakse, võib tulemuse tagastamiseks kuluda ressurssi veelgi enam. Enamasti ei ole strateegilistele küsimustele välkkiireid vastuseid vaja, vastupidiselt päringutele, kus näiteks keegi teiselpool telefoni vastust ootab. See ei tähenda aga, et andmehoidla arendaja võiks süsteemi jõudluse arendamise käigus tähelepanuta jätta.

Operatiivsetes andmebaasides kasutatakse päringute jõudluse tõstmiseks tavapäraselt indekseid. Andmehoidlas ei ole faktitabelis otstarbekas indekseid kasutada, sest suurem osa päringuid kasutab rohkem kui poolt faktitabeli andmetest. Mõistlikumaks osutub kogu tabeli järjestikune skaneerimine (ingl k *full sequential scan*). Üks võimalik lahendus on kokkuvõtetes ehk summatabelites. Kui on mingil määral võimalik ennustada, mis tüüpi päringuid kasutajad rohkem esitavad, siis võib ette valmistada sellised summatabelid, mis seda infot sisaldavad. Kui need agregaattabelid nõutud infot pakkuda ei suuda, võib kasutaja ikka detailseid andmeid kasutada. Joonis 5 näitab summeerimise tasemeid, mida tihti kasutatakse.




**Joonis 5**. Näide summeerimise tasemetest andmehoidlas.

Paar näidet võimalikest summatabelitest ja nendele vastavatest tabeliskeemidest, kui andmehoidla teemaks oleks müük:

- Kliendid toote järgi iga kuu jaoks.
  Tabeli skeem = { KliendiNimi, TooteNimi, KuuNr, MüügiSumma }
- Kliendid toote järgi iga kvartali jaoks
  Tabeli skeem = { KliendiNimi, TooteNimi, KvartaliNr, MüügiSumma }
- Tooted regiooni järgi iga kuu jaoks
  Tabeli skeem = { TooteNimi, Regioon, KuuNumber, MüügiSumma }
- Tooted regiooni järgi iga kvartali jaoks
  Tabeli skeem = { TooteNimi, Regioon, KvartaliNr, MüügiSumma }

Summeerimise tasemete olemasolu annab kasutajatele võimaluse ühest summeerimise tasemest "puurida sügavamale" (ingl k *drill down*) järgmisele detailsemale tasemele. Näiteks avastatakse, et just üks tootegrupp on eriti edukas või vilets. Uurides täpsemalt selle grupi üksikuid tooteid, on näha, kas kogu tootegrupp on mõjutatud või äkki ainult üks isoleeritud toode. Ümberpöördult, võimalusega "puurida kõrgemale" (ingl k *drill up*) saab kindlaks teha, kas üks kehv toode ei mõjuta kogu tootegruppi.

Summatabelite tarvitusele võtmine püstitab paar küsimust:

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

Üks lahendus on kasutada summa navigaatorit (ingl k *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 analüüsib seda, et tuvastada, missuguseid veerge ja summeerimise taset vajatakse. Tänapäeval on mõnesse relatsioonbaasihaldurisse juba summatabelite toetus koos automaatse uuendamisega juba sisse ehitatud.

#### 2.1.1.4. Aja käsitlemine ja aeglaselt muutuvad dimensioonid

Aja dimensiooni lisamine võimaldab hoida ja kasutada ajaloolist informatsiooni. See tähendab, et andmehoidla kasutajad saavad vaadelda oma ettevõtte olukorda mingil kindlal ajahetkel või mingi perioodi vältel. Ajalooliste andmete väärtus ja tähtsus on üldiselt tunnustatud ning selliste andmete puudumine operatiivsetes süsteemides on üks mõjuvaimaid faktoreid andmehoidla loomiseks [7].

Kui operatiivsetes süsteemides olevad olemid muutuvad, siis tavaliselt asendatakse vanad väärtused uutega. See loob mulje, nagu ei oleks vanad väärtused kunagi eksisteerinudki. Näiteks, kui klient kolib teise kohta ja ta aadress muutub, siis pole tellimuste töötlemise süsteemis mingit põhjust vana aadressi alles hoida. Informatsiooni alleshoidmine selle kohta võib isegi segadust tekitada ja luua riski, et tellimus saadetakse kogemata valele

aadressile.

Ajalises süsteemis, nagu seda on andmehoidla, mis peab tõest ajaloolist infot hoidma, on väga tähtis eristada vanal aadressil esitatud tellimusi nendest, mida klient esitas uuel aadressil elades. Näiteks on sellist informatsiooni vaja, kui ettevõte soovib uurida müüki regioonide järgi. Informatsioon tellimuste kohta, mis esitati eelmisel aadressil elades, peab säilima, et see tolle regiooni müügis kajastuks. Kui vana aadress uuega lihtsalt üle kirjutada, siis jääb mulje, nagu oleks kõik tellimused esitatud uuelt aadressilt, mis muidugi ei ole tõsi. Samal põhjusel ei saa klientide andmeid ära kustutada lihtsalt sellepärast, et nad ei ole enam kliendid. Kui nad on kasvõi ühe asja ostnud või tellinud, peavad nad süsteemis püsima. Kõikide atribuutide eelnevaid väärtusi loomulikult ei ole vaja säilitada. Näiteks inimese nime puhul ei saa eelneva väärtuse säilitamisest mingit praktilist kasu.

Ralph Kimball, tuntud innovaator andmehoidlate vallas, nimetab selliseid muutusi dimensioonide atribuutides aeglaselt muutuvateks dimensioonideks (ingl k *slowly changing dimensions*) [7]. Ta kirjeldab ka kolme meetodit dimensionaalsete atribuutide muutuste jälgimiseks ja kohtlemiseks. Neid meetodeid nimetab ta lihtsalt tüüpideks üks, kaks ja kolm.

Esimene meetod ehk Tüüp 1, tähendab lihtsalt vanade väärtuste asendamist uute väärtustega. See tähendab, et ei ole vajadust säilitada eelnevat väärtust. Sellise lähenemise positiivne külg on selle lihtsus ja kerge teostatavus, aga ilmselgelt ei paku see lahendus mingit ajalist toetust. Mõnel juhul on see meetod aga kõige sobivam. Iga andmebaasis oleva elemendi ajaloolisi väärtusi ei ole vaja jälgida ja üle kirjutamine on sel juhul õige tegevus. Näiteks võib tuua eelnevaltki mainitud kliendi nime.

Teine lahendus aeglaselt muutuvate dimensioonide jaoks on Tüüp 2, mis on keerulisem lahendus, kui Tüüp 1 ja üritab ka ajaloolisi väärtusi salvestada, kasutades selleks erilaadset versioonikontrolli. Seda on kõige kergem taas selgitada näite varal, kus kliendi aadressi eelnevaid väärtusi peab alles hoidma, et need regiooni müügis korrektselt kajastuksid. Tüüp 2 meetod üritab probleemi lahendada uute andmebaasikirjete ehk ridade loomisega. Iga kord, kui mingi atribuudi väärtus muutub, luuakse uus rida andmebaasi, kus muutunud atribuudil on uus väärtus, kõik teised aga jäävad samaks. Selline lähenemine loob olukorra, kus rikutakse primaarvõtme unikaalsuse reeglit, sest uue rea võti oleks sama, mis

eelmise rea oma. Selle probleemi vältimiseks võib kasutada surrogaatvõtmeid (ingl k *surrogate keys*). Surrogaatvõti on süsteemi poolt genereeritud identifikaator ning selle väärtuse määramiseks kasutatakse kahte peamist viisi:

1. Identifikaatorit pikendatakse versiooninumbri võrra. Näiteks klient identifikaatori väärtusega "1234" saab väärtuseks "1234001". Peale esimest muutust, uue rea loomisel oleks sel uuel real identifikaatoriks "1234002".
2. Identifikaator on täiesti üldistatud ja ei oma mingit seost kliendi eelmise identifikaatoriga. Iga kord, kui uus rida lisatakse, genereeritakse täiesti uus identifikaator, näiteks väärtusega "8A99BBFA5A00D211B17308005AA5B22".

Tüüp 2 lähenemist võib lahendada ka algus- ja lõpukuupäeva sisaldavate veergude lisamisega igale dimensioonile. Seda nimetatakse ka ridade ajatembeldamiseks (ingl k *row timestamping*) [7].

Kolmas meetod (Tüüp 3) hõlmab lahendust, kus salvestatakse nii atribuudi praegune kui ka esialgne väärtus. See tähendab, et andmebaasitabelisse peab looma ühe täiendava veeru, mis esialgset väärtust hoiab. Praegust, parasjagu kehtiva väärtuse veergu uuendatakse iga muutuse puhul, esialgset väärtust sisaldav veerg aga ei muutu. Vahepealsed väärtused lähevad seega kaotsi. Selle meetodi toetus ajalooliste andmete säilitamisele on üsna veider ja ei paku erilist praktilist väärtust.

### 2.1.2. Metaandmed

Metaandmed on lühidalt andmed andmete kohta ehk teisisõnu andmehoidla olemite ja atribuutide kirjeldused. Metaandmed võib klassifitseerida tehnilisteks ja ärilisteks [1].

Tehnilised metaandmed sisaldavad andmehoidla arendamise ja haldamise käigus kasutatavaid andmeid. Need hõlmavad muuhulgas:

- Informatsiooni lähtesüsteemide kohta, kust andmed pärit on
- Transformatsioonide kirjeldusi, mille andmed enne andmehoidlasse jõudmist läbisid
- Reegleid, mille alusel andmeid transformeeritakse
- Andmete sõltuvusi teistest andmeelementidest (kui sõltuvusi eksisteerib)
- Ligipääsuõigusi, varundamise ajalugu, andmete eraldamise ajalugu, informatsiooni erinevate tabelite kasutamise kohta, jne

Ärilised metaandmed kirjeldavad täpsetes äriterminites, mida andmeelemendid tegelikult tähendavad. Iga süsteemis oleva olemi, seose ja atribuudi kohta peab saama anda selge ja üheselt mõistetava definitsiooni.

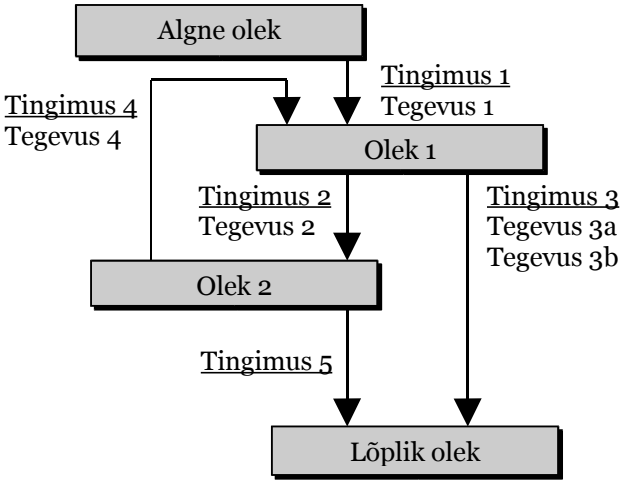
Metaandmete haldamine toimub tavaliselt spetsiaalse tarkvara abil ning samuti asetsevad metaandmed ise enamasti andmehoidlast eraldi.

## 2.2. Andmete eraldamine

Andmete eraldamise protsessi illustreerimiseks võib taas näiteks tuua postimüügifirma, kes soovib aruandeid oma toodete müügi kohta nii toote kui ka kliendi kaupa. 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 kuhugi, et see sobival ajal andmehoidlasse salvestada.

Käesoleval juhul on müük seotud kindlasti olemiga tellimus. Tellimusel on olemas oma kindel elutsükkel. Igas elutsükli etapis on tal mingi olek ja tüüpiline tellimus liigub ühest olekust teise kuni ta elutsükkel on lõppenud ja ta süsteemist eemaldatakse. Selle probleemi – millal info eraldada – saab lahendada, koostades olekute ülemineku diagrammi, mis jälgib olemi elutsüklit ja määratleb need sündmused, mis võimaldavad olemil ühest loogilisest olekust järgmisesse minna. Joonis 6 esitab olekute ülemineku diagrammi üldist kuju [7]. Kuigi sel diagrammil on neli olekut ja viis üleminekut, ei ole tegelikkuses olekute ja üleminekute arvul piiri.




**Joonis 6**. Üldine olekute ülemineku diagramm.

Postimüügifirma võib näiteks otsustada, et müük on toimunud, kui tellimus on kliendile ära saadetud. See on hetk, kus tellimuse kohta käiva informatsiooni peab salvestama ja tallele panema, andmehoidlasse sisestamist ootama. Ainus viis seda saavutada on teha mingi muudatus programmi nii, et see muutus olekus ära tuntakse ja salvestatakse.

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

Relatsioonbaasihaldurid (edaspidi RDBMS) pakuvad tihti võimalust kasutada andmebaasis niinimetatud "päästikprotsesse" (ingl k *trigger*). Need päästikprotsessid on tabeli-põhised SQL laused, mis käivitatakse iga kord, kui mingi rida lisatakse, uuendatakse või kustutatakse. Võimalik on luua päästikprotsess, mis jälgib tellimuse olekut iga kord kui seda uuendatakse või muudetakse. Niipea, kui olek näitab, et müük on toimunud, saab päästikprotsess tellimuse mingisse ajutisse tabelisse sisestada, et hiljem sinna kogunenud andmed andmehoidlasse viia.

Sellega ei ole eraldamise protsess veel läbi. Veel on vaja eraldada informatsioon toodete ja klientide kohta ehk dimensioonid, mida soovitakse analüüsida. Kui klientide ja toodete andmeid hoitakse samuti relatsioonandmebaasis, siis on üsna lihtne nende tabelite sisu kätte saada. Vajalikud tabelid saab faili salvestada kasutades RDBMS-ga kaasas olevat või mõne kolmanda osapoole eksporditarkvara. Uute klientide/toodete detaile ja

olemasolevatesse andmetesse tehtud muudatusi on veidi raskem hankida, aga tihti on parim viis samad ülalkirjeldatud päästikprotsessid.

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

## 2.3. Andmete integratsioon

Integratsiooni võib veel omakorda jagada kaheks: formaadi integratsioon ning semantiline integratsioon [7].

### 2.3.1. Formaadi integratsioon

Enamikes organisatsioonides on tavaliselt mitmeid juhtumeid, kus ühe süsteemi atribuutide formaat erineb teiste süsteemide samade või sarnaste atribuutide omast. Näiteks:

- Pangaarve või telefoninumbrid võivad olla tekstina ühes ja numbrina 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”. Enamikel andmebaasihalduritel on kuupäeva hoidmiseks DATE andmetüüp, aga talletamisformaadid võivad lahku minna üksteisest. Näiteks Oracle andmebaasihalduris sisaldab andmetüüp DATE kuupäevale lisaks ka kellaaega ning spetsiaalselt kellaaja säilitamiseks mõeldud andmetüüp TIME puudub üldse. IBM-i andmebaasihaldur DB2 kasutab omakorda ajatempli andmetüübis TIMESTAMP tundide, minutite ja sekundite eraldajana punkti, teised süsteemid aga enamasti koolonit.
- Rahalised atribuudid põhjustavad samuti raskusi. On süsteeme, mis salvestavad raha täisarvuna ja ootavad, et teine programm sisestaks ise komakoha. Muudel

võivad olla komakohad sisse ehitatud.

- Erinevates süsteemides kasutatakse erinevaid stringi pikkusi nimede, aadresside, tootekirjelduste jms jaoks.

Formaadi ebatäpsused on väga tavalised, eriti tõenäoline on see 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 loodud kindlustamaks andmehoidlasse laetavate andmete standarditele vastavust. Et kõik kuupäevad oleksid samas formaadis, rahalised väärtused oleksid kujutatud identselt, jne.
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.
Integratsioonireeglite komplekti kasutatakse nagu kaardistikku, mis defineerib kuidas lähtesüsteemidest, allikatest niiöelda "välja tõmmatud" infot peab konverteerima, transformeerima enne, kui sellel lubatakse siseneda andmehoidlasse.

### 2.3.2. Semantiline integratsioon

Andmehoidlad koondavad infot paljudest erinevatest süsteemidest. Tüüpiline on, et firmas kasutavad erinevaid süsteeme ka erinevad inimesed - finantssüsteeme kasutavad raamatupidajad, müügisüsteeme – müüjad, jne. Kõigil neil inimestel võib olla oma ettekujutus näiteks mõiste müük tähendusest. 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 tihti otsuste toetamiseks organisatsiooni kõige

kõrgemal tasandil. Seepärast on tähtis, et igal objektil, mis andmehoidlasse lisatakse, oleks kindel, kõigi poolt üheselt mõistetav tähendus. Selle teostamiseks peab andmehoidla koosseisus olema informatsiooni kataloog, milles kirjeldatakse ära iga andmehoidlas oleva atribuudi kindel tähendus. Selliseid "andmeid andmete kohta" nimetatakse tavaliselt metaandmeteks (ingl k *metadata*) [5].

Kuigi integratsiooni osa nimetatakse tihti ka transformatsiooniks (ingl k *transformation*), sobib see termin eelkõige formaadist rääkides, sest semantika puhul on tegemist mõistete ühtlustamise, kõigile üheselt arusaadavaks tegemisega, mitte muundamisega.

## 2.4. Andmete laadimine

Kui vajalik informatsioon on lähtesüsteemidest eksporditud ja integreeritud, on ta valmis andmehoidlasse sisestamiseks. Üldiselt kulgeb laadimise protsess selliselt, et iga päev lisatakse antud päeva andmed eelnevatele, pikendades seeläbi andmete "ajalugu" veel ühe päeva võrra. Lisamine võib toimuda ka pikema intervalli tagant, näiteks kord kuus või nädalavahetusel, et organisatsiooni elurütmi vähem häirida.

Osa tarkvaratooteid, mis kõiki kolme protsessi - eraldamist, transformatsiooni ja laadimist (ingl k *extraction, transformation and loading*) - teostavad, töötavad niiöelda konveiermeetodil. Nad loevad ühe kirje lähtesüsteemist, transformeerivad selle, kirjutavad andmehoidlasse, võtavad järgmise ja kordavad protsessi, jne [7]. Suuremate andmekoguste puhul võib selline lähenemine aeglaseks jääda ja oleks otstarbekam RDBMS-i endaga kaasas olevat laadimistarkvara kasutada.

## 2.5. Juurdepääsu, päringute, raportite tarkvara

Enamus andmehoidlaid teostatakse klient-server konfiguratsiooni kasutades. Klient-serveri mõiste siin kontekstis tähendab kasutaja eraldamist andmehoidlast, ehk kasutaja tarvitab oma personaalarvutit ja andmehoidla asetseb eemalasuval serveril [7].

Relatsioonandmebaasidel baseeruvatele andmehoidlatele ligipääsuks on arvukalt erinevat tarkvara. 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 ühilduvad veebiserveriga, mistõttu kasutajal on vaja ainult veebibrauserit info kuvamiseks.

Vähemlevinud (kuid mitte vähem kasulikud) omadused hõlmavad näiteks pakkraporteerimist, ärisemantika sisaldust ja raportite jaotamist. Pakkraporteerimine hõlmab näiteks võimalust käivitada päringuid automaatselt öösiti. Ärisemantika tähendab andmete esitamist kasutajale kujul, mis neile kergesti mõistetav on. Näiteks võib andmebaasis olla veerg nimega KL_TEL_V, aga päringutarkvara esitab selle kasutajale kujul “Kliendi tellimuse väärtus”. See lähenemine eeldab enamasti, et andmehoidlal on olemas metaandmete kiht ja päringutarkvara oskab seda kasutada. Raportite jaotus võib tähendada, et erinevatele inimestele on võimalik eriilmelisi või ka teistsuguse sisuga raporteid saata, näiteks kliendihaldurile ainult tema klientide kohta käivat informatsiooni.

## 2.6. Andmete kaevandamine

Andmehoidlate leiutamisest alates on tekkinud aga ka mõned spetsialiseeritud analüüsimeetodid. Üks nendest on andmete kaevandamine (ingl k *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 osa tööst ise ära. Kasutajad kirjeldavad andmeid andmete kaevandamise programmile, määrates ära vajalikud andmetüübid ja valiidsete väärtuste vahemiku. Programm käivitatakse seejärel andmehoidla peal ja rakendades standardseid seoste äratundmise algoritme, otsitakse ja kuvatakse andmetes olevate seoste detaile. Kasutaja ei pruugi nende seoste kohta eelnevalt midagi teada. Esineb kaks peamist andmete kaevandamise tüüpi: hüpoteesi kinnitus (ingl k *hypothesis verification*) ja teadmuse avastamine (ingl k *knowledge discovery*) [7].

Hüpoteesi kinnitus kujutab olukorda, kus kasutajal on idee või aimdus, et andmete vahel on mingi tähendusrikas seos. Näiteks pangakontode haldur võib arvata, et abielus, kõrge sissetulekuga ja kindlas vanusevahemikus kliendid paigutavad raha avatud investeerimisfondi. Pärast vastava päringu esitamist ja tulemuste tõlgendamist võib ta leida, et selliste omadustega klientide puhul on see tõenäosus piisavalt suur ja tulevikus võib sinna gruppi kuuluvatele klientidele vastavaid pakkumisi esitada.

Teadmuse avastamine on situatsioon, kus andmete vahel võib eksisteerida seniajani teadmata statistiliselt tähendusrikas seos, aga see seos on selline, mille märkamine võib inimesest analüütikul võimatu olla. Seda põhjusel, et inimene üritab andmeid loogiliselt ühendada, programm aga otsib andmetes lihtsalt reeglipärasust ja tõenäosuslikke seoseid.

Andmete kaevandamise tarkvara ei suuda siiski midagi maagilist korda saata. Ta vajab töötamiseks vigadeta ja kvaliteetseid andmeid ning õnnestumiseks analüüsi tulemusi tõlgendavate inimeste kogemust ja asjatundlikkust seotud teemaga. Pealtnäha kasutu korrapära või muster (ingl k *pattern*) andmetes võib õige ärilise tõlgenduse korral muutuda kasulikuks informatsiooniks ning vastupidi.

## 2.7. Andmevakad

Andmevaka (ingl k *data mart*) termini jäik definitsioon on - andmete kogum, mis on integreeritud andmehoidla alamhulk [1]. Andmevakk on kindlalt piiritletud teemaga ning on suunatud mingile kindlale kasutajategrupile. Mõnikord võib andmevakk paikneda andmehoidlaga samas andmebaasis. Enamusel juhtudel asub ta siiski füüsiliselt eraldiseisval andmebaasiserveril, mis asub näiteks andmevakka kasutava osakonna või kasutajategrupi lokaalvõrgus.

Andmevakad, mille sisu pärineb tsentraalsest andmehoidlast, on sõltuvad andmevakad (ingl k *dependent data marts*). Kõik seda tüüpi andmevakad omavad kõrget väärtust, sest nende kasutajad näevad ühtset, integreeritud informatsiooni, olenemata sellest, kui mitu iseseisvat andmevakka paigaldatud on ja mis tehnoloogiaid nad kasutavad.

Kahjuks on levinud ka väärarusaamad, et andmevakk on odav ja lihtne alternatiiv

andmehoidlale. Sellise lähenemise tulemuseks on sõltumatu andmevakk (ingl k *independent data mart*), mis koondab kokku mingi fragmendi organisatsiooni ärilisest informatsioonist. See iseenesest on kasulik, aga võib põhjustada probleeme. Näiteks enam kui ühe sellise andmevaka tekkimisel ei pruugi andmed olla enam kooskõlas ning võivad duplitseeruda, sest andmed ei ole pärit ühtsest, integreeritud andmehoidlast. Samuti võib andmevaka mahu kasvamisel või kasutajate arvu suurenemisel tekkida probleeme skaleeritavusega (ingl k *scalability*), sest algselt "väikese ja odava" andmevaka planeerimisel seda arvesse ei võetud.

## 2.8. OLAP

Üks rakenduste klass, mis samuti andmehoidlatest kasu lõikab, on OLAP. Lahti kirjutatuna on OLAP inglise keeles *OnLine Analytical Processing* ning seda võib tõlkida kui onlain-analüüs [8]. See aga ei selgita mingil määral, mida OLAP-tarkvara teeb või miks teda vaja on.

"The OLAP Report" nimeline organisatsioon on loonud oma definitsiooni, mis nüüdseks on laialt tunnustatud. See definitsioon koosneb viiest märksõnast, kriteeriumist, millele üks OLAP-toode vastama peaks ning kannab nime FASMI – *Fast Analysis of Shared Multidimensional Information* [4]. Eesti keelde tõlgituna oleks see "Jagatud Multidimensionaalse Informatsiooni Kiire Analüüs".

- Jagatus tähendab, et süsteem peab täitma turvalisuse ja konfidentsiaalsuse nõudeid mitme kasutajaga keskkonnas.
- Multidimensionaalsus on kriteeriumitest kõige tähtsam. Süsteem peab võimaldama mitmedimensionaalset, kontseptuaalset vaadet andmetest, kaasa arvatud toetust dimensioonide hierarhiatele.
- Informatsioon hõlmab kogu vajaminevat andmestikku, kus see ka füüsiliselt ei paikneks. OLAP-toodete mahutavus on väga erinev – suurimad suudavad hoida vähemalt tuhat korda rohkem andmeid, kui väikseimad.
- Kiirus tähendab, et süsteem peab tagastama kasutajale tulemuse keskmiselt viie sekundiga, kusjuures lihtsad analüüsid ei tohiks aega võtta rohkem kui ühe sekundi ja väga vähesed rohkem kui kakskümmend sekundit. OLAP-tarkvara tootjad

kasutavad selle eesmärgi saavutamiseks erinevaid viise, muuhulgas spetsiifilisi andmete salvestamise vorminguid, päringute tulemuste eel-kalkuleerimist ja kõrgendatud nõudmisi riistvarale.

- Analüüs tähendab, et süsteem peab hakkama saama kõigi ärilise ja statistilise analüüsi aspektidega, mis kasutajale olulised on. Samas peab analüüsi tulemus olema piisavalt lihtsal ja kasutajale mõistetaval kujul. Kasutajal peab olema võimalus defineerida uusi päringuid ja kasutada andmeid igal soovitaval kujul ilma, et ta peaks ise programmeerima.

Relatsioonilise andmemudeli leiutaja Dr Edgar Codd on loonud oma reeglid (algul 12, hiljem lisaks veel 6) OLAP-toodete jaoks, kuid eelpoolnimetatud FASMI on hõlpsamini mõistetav ja meeldejäävam [1] [4].

Tüüpiline OLAP arhitektuur koosneb OLAP-i serverist, mis paikneb andmehoidla ja kasutaja vahel. Iga OLAP-i toode kuulub ühte kolmest kategooriast [7]:

MOLAP: Multidimensionaalne OLAP (ingl k *Multidimensional OLAP*). Terminit MOLAP kasutatakse toodete puhul, mis kasutavad tootjapõhist andmebaasisüsteemi. See tähendab, et nende aluseks olev andmebaas ei ole relatsiooniline, ega ka mingile muule standardile vastav. Tihti on selleks aluseks arvutustabel, mis sarananeb hiiglasuure Microsoft Exceli töölehega. Sellise lähenemise tulemuseks on olukord, kus mingi suvalise individuaalse lahtri positsiooni on võimalik välja arvutada vaid dimensioonide väärtusi kasutades. Selle omaduse tõttu hiilgavad taolised süsteemid oma andmete lugemise kiiruse poolest. MOLAP-lahendusi vaevab aga paar suuremat probleemi. Esimene on nende skaleeritavus. MOLAP-süsteemile tuleb kirjeldada dimensioonide umbkaudne suurus. See võib toimuda näiteks sisestades lihtsalt klientide, toodete, jne arvu, mida süsteem peab suutma hoida. Seejärel, teada saanud kõikide dimensioonide suurused, korrutab süsteem lihtsalt kõik numbrid omavahel, et teada saada, mitu lahtrit ta peab oma töölehe maatriksis looma. Näiteks kui ettevõttel on kaks miljonit klienti, 1000 produkti ja tahab säilitada iga päeva müüki 10 aasta vältel, siis selline maatriks sisaldaks:

$(2000000 * 1000 * 10 * 365) = 7.3 * 10^{12}$ lahtrit

Selline arvutamisviis eeldab küll vaikimisi, et ettevõte müüb iga toodet iga päev igale kliendile. Ettevõttele see kindlasti meeldiks, aga see on siiski ebarealistlik ja tulemuseks on suurel hulgal tühje lahtreid, mida nimetatakse andmete hõreduseks (ingl k *sparsity*).

Hõredus on multidimensionaalse andmebaasi tühjade lahtrite mõõdik. MOLAP-süsteemide puhul võib hõredus tihti olla suurem kui 90% [7].
Teine probleem seisneb päringukeelte standardite puudumises. Relatsioonandmebaasides on juba pikka aega olnud standardiks SQL. Sellel võib olla küll puudusi, aga enamasti võib olla kindel, et ühes RDBMS-is töötavad SQL laused võib vähese vaevaga tööle panna ka teistes süsteemides.

ROLAP: Relatsiooniline OLAP (ingl k *Relational OLAP*). ROLAP-i puhul baseerub andmemudel relatsioonandmebaasil. Sellise süsteemi puhul tuleb kasutajal sisuliselt defineerida dimensioonid ja faktid ning ROLAP-i "mootor" loob ise andmemudeli ning võimaldab sellele mudelile päringuid esitada. Seda lahendust ei vaeva tavalised MOLAP-ga seotud probleemid. Kuna andmed asuvad tavapärases relatsioonandmebaasis, siis skaleeritavuse ja hõreduse küsimused enamasti ei kehti. Päringukeeleks on standardne SQL ja ROLAP-toode suudab tavaliselt seda ise genereerida, et kasutaja neid käsitsi kirjutama ei peaks. Nagu eelnevalt mainitud, üks MOLAP-i suurimaid plusse on jõudlus, sest kirjeid on võimalik leida lihtsa arvutustehte abil ning puuduvad aeganõudvad tabelite ühendamised. ROLAP-i puhul on olukord teistsugune ning taas tuleb tegeleda jõudluse küsimustega, mis traditsiooniliselt relatsioonilise mudeliga kaasnevad.

HOLAP: Hübriid OLAP (ingl k *Hybrid OLAP*). See on kompromiss kahe eelneva lahenduse vahel. HOLAP-i taga peituv idee seisneb selles, et summeeritud informatsiooni jaoks saab kasutada MOLAP-arhitektuuri kasulikke omadusi, aga kui on vaja detailseid andmeid, siis süsteem pöördub nende lugemiseks tagasi relatsioonandmebaasi poole. Seega on tagatud jõudluse kasv (kuigi ainult summade tasemel) ja skaleeritavuse küsimused saab kõrvaldada, sest kõik detailsed andmed asuvad tavalises relatsioonandmebaasis.

## 2.9. Arendusprotsessi kokkuvõte

Andmehoidla arendamist alustatakse tavaliselt andmebaasi komponendist. Üks levinumaid modelleerimise tehnikaid andmehoidla loomisel on dimensionaalne modelleerimine. Selle käigus määratakse ära andmehoidla teema ja analüüsi dimensioonid. Modelleerimise tulemuseks oleva andmemudeli enamlevinud variatsioonid on täht-skeem ja lumehelves-skeem.

Teised tähtsamad komponendid on andmete eraldamine lähtesüsteemidest, eraldatud andmete integratsioon ja nende laadimine andmehoidlasse. Küsimused, millele tuleks suuremat tähelepanu pöörata nende protsesside juures, on järgmised:

- Kas eraldatud saavad ikka kõik vajalikud andmed?
- Kas eraldatud andmed on vigadeta, ühtses vormingus ja üheselt mõistetavad?
- Mis intervalli tagant andmehoidla andmeid uuendada?

Tehnoloogiline platvorm, millel andmehoidlad põhiliselt baseeruvad, on üldjuhul relatsioonbaasihaldurid.

# 3. Andmehoidla realisatsioon kindlustusfirma näitel

## 3.1. Eesmärk ja nõudmised

Praktilise osa eesmärk on luua täisfunktsionaalne andmehoidla kindlustusfirma näitel. Täisfunktsionaalne tähendab, et ei piirduta vaid andmebaasi komponendi loomisega, vaid automatiseeritud eraldamise, transformatsiooni ja laadimise protsess peab ka korrektselt töötama. Päringutarkvara ei ole otstarbekas ega jõukohane ise luua, seetõttu tuleb kasutada mingi kolmanda osapoole toodet.

Kindlustusfirma oli loomulik valik, sest autor töötab juba pikemat aega projektis, mille eesmärk on ühele Soome firmale tarkvara luua (operatiivset, mitte andmehoidla tarkvara) ning seetõttu on käepärast ka vastav andmestik, mille alusel andmehoidla üles ehitada. Samuti on olemas raportite näiteid, mida kõne all olev või mõni teine sarnase suunitlusega firma on soovinud oma operatiivsete süsteemide abiga omandada. Reaalsete nõudmiste puudumisel on neid raporteid ka mugav kasutada nõudmiste kirjeldamisel ja andmete modelleerimisel. Enne nõudmiste kirjeldamist ja nende realiseerimist aga tuleks lühidalt tutvustada andmete lähtesüsteemi ehk kindlustussüsteemi Once&Done andmebaasi ülesehitust.

### 3.1.1. Kindlustussüsteemi Once&Done andmebaasistruktuuri tutvustus

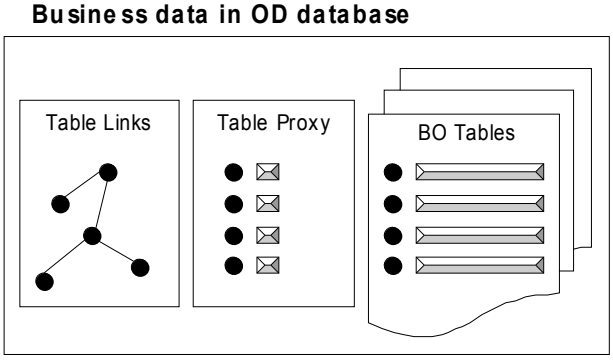
Once&Done kindlustussüsteem (edaspidi O&D) kasutab relatsioonandmebaasi oma operatiivsete andmete hoidmiseks ehk kõik parameetrilised andmed ja äriandmed asuvad tabelites. Parameetrilisi andmeid sisaldavad tabelid hõlmavad näiteks informatsiooni tariifide, reeglite, kasutajaõiguste ja muu sellise kohta.

Äriandmete hoidmiseks olevad tabelid võib jagada kolme kategooriasse:
Mitu tabelit äriobjektide andmete hoidmiseks (aadressid, isikud, poliisid, autod, jne). Iga äriobjekti tüübi jaoks on eraldi tabel. Tabelite nimed on sarnased äriobjektide tüüpidele,

näiteks Aadressi objektid on salvestatud tabelisse nimega Aadress, jne.

Üks tabel prokside hoidmiseks. Iga äriobjekt, olenemata oma tüübist, omab talle vastavat proksi objekti. Proksi sisaldab selle äriobjekti kohta üldist informatsiooni, mis on ühine kõigile äriobjektidele, näiteks objekti nimi, objekti tüüp, jne.

Üks tabel linkide hoidmiseks. Linke kasutatakse kahe objekti omavahelise seose kirjeldamiseks. Kõik seosed äriobjektide vahel on kirjeldatud linke kasutades. Teisisõnu on kõikide kaustade struktuur kirjeldatud linkide abil. Kuna igal äriobjektil on vastav proksi objekt, siis lingid kirjeldavad ka seoseid prokside vahel. Joonis 7 üritab illustreerida seda olukorda.




**Joonis 7**. O&D andmebaasis äriobjektide andmeid hoidvad tabelid.

Äriobjekti loomisel genereeritakse talle OID ehk globaalne unikaalne identifikaator (ingl k GUID - Global Unique Identificator). Sellise OID andmetüüp andmebaasi salvestamisel on BINARY(16) (DB2 andmebaasisüsteemi puhul on vastav andmetüüp CHAR(16) FOR BIT DATA). Kõik äriobjektid kasutavad identifikaatorina seda sama andmetüüpi.

**Miks lingid ?**

Klassikalises relatsioonandmebaasis kirjeldatakse kõik seosed äriobjektide vahel kasutades võõrvõtmeid (üks-mitmele seoste puhul) või seosetabeleid (mitu-mitmele seoste puhul). Sellisel juhul on seoste teostus laiali mitmes tabelis üle andmebaasi. Muutused seostes põhjustavad muutusi andmebaasiskeemis.

O&D süsteemis on kõikide äriobjektide vahelised seosed, olenemata objektide tüübist, salvestatud ühte tabelisse – Lingid. See on võimalik, sest kõikidel äriobjektidel on

standardne identifikaator - OID ning neid identifikaatoreid kasutatakse objektide vaheliste seoste salvestamiseks. Kuna linke kasutatakse kõikide äriobjektide vaheliste seoste esitamiseks, on võimalik kõik seosed mällu laadida ilma äriobjektide tabeleid kasutamata. Samuti on võimalik seoseid lisada, muuta ja kustutada ilma andmebaasiskeemi muutmata.

**Miks proksid?**

Proksi, mis sisaldab üldist informatsiooni mingi äriobjekti kohta, omab ka sama OID väärtust nagu talle vastav äriobjekt. Sama OID-d kasutatakse ka linkide tabelis seoste kirjeldamisel. Seega on võimalik kausta struktuur ja üldine informatsioon objektide kohta laadida mällu, kasutades ainult kahte tabelit - proksi ja lingid. Äriobjektide tabeleid ei kasutata. Samuti saab proksisid kasutada kiireks objekti valiidsuse perioodi määramiseks.

O&D toetab äriobjektide ajaloolisi versioone. See tähendab, et andmebaasist on võimalik pärida, mis ajaperioodil mingi konkreetne äriobjekt valiidne on ja mis väärtused tal sel perioodil on. Sellise funktsionaalsuse toetamiseks on igale äriobjektile lisatud üks lisaatribuut, jõustumise kuupäev (ingl k *effectivedate*). Sel juhul võib iga äriobjekti kohta olla mitu rida talle vastavas tabelis. Alloleval joonisel on näide tabeli Isik sisust.

| *oid* | *effectivedate* | *firstname* | *lastname* | *other data* |
|---|---|---|---|---|
| 00131028F262D311ACC000A024A8E52D | 2000-01-19 | Silvia | Lockhead | xxxxx |
| 00131028F262D311ACC000A024A8E52D | 2000-05-20 | Silvia | Haroldy | xxxxx |
| 00131028F262D311ACC000A024A8E52D | 2000-09-29 | Silvia | Haroldi | xxxxx |

Kui tabeli Isik sisu on selline nagu antud näites, siis selle isik-tüüpi äriobjekti, mille OID=00131028F262D311ACC000A024A8E52D, alguskuupäevaks on 2000-01-19. Esimese versiooni jõustumise kuupäev on samuti 2000-01-19 ning lõppemise kuupäevaks on 2000-05-19. Teise versiooni jõustumise kuupäevaks on 2000-05-20 ja lõppemise kuupäevaks 2000-09-28. Kolmanda versiooni jõustumise kuupäevaks on 2000-09-29 ja lõppemise kuupäev puudub (kehtib lõpmatuseni). Kirjeldatud viisil on võimalik samast äriobjektist salvestada andmebaasi mitu erinevat versiooni erinevate andmetega.

### 3.1.2. Nõudmised andmetele

Reaalsete nõudmiste puudumisel tuleb need ise tuletada, analüüsides saadavalolevaid andmeid ning kasutades näidetena raporteid, mida üks või teine firma on erinevatel aegadel

soovinud.

Lähtesüsteemiks olev Once&Done süsteem on loodud toetamaks kindlustuslepingute ehk poliiside sõlmimist, mis kindlustusfirma seisukohalt tähendab tegelikult kindlustuse müüki. Seega võib julgelt paika panna loodava andmehoidla teema, milleks on müük, või nagu kindlustusfirmad ise seda nimetavad – kogutud preemiad. Teine valdkond, mis kahtlemata kindlustusfirmasid huvitab, on väljamakstud kahjud. Seda valdkonda haldav süsteem eksisteerib, kuid asub Once&Done süsteemist eraldi ning talle puudub ligipääs, seetõttu ei saa ka käesolevas töös lähtesüsteemina kasutada. Kõigest ühe lähtesüsteemi olemasolu hõlbustab ka andmehoidla loomist.

Andmehoidla teema on selgunud ning peatüki ülejäänud osas üritatakse selgelt välja kirjutada loodava andmehoidla täpsed nõudmised – milliseid analüüse/raporteid on vaja, mis peab olema tulemus, kuidas kuvatud olema, mis on riistvaraline ja tarkvaraline platvorm. Seejärel need nõudmised ka modelleeritakse.

### 3.1.3. Eeldatav tulemus

Oletame, et kindlustusfirma soovib raporteid, mis kajastavad järgmiseid teemasid:

- Kogutud preemiad toodete ehk kindlustusliikide järgi
- Kogutud preemiad agentide ehk müüjate järgi
- Kogutud preemiad kindlustatud objektide tüüpide järgi
- Kõige tasuvamate klientide ehk kindlustusvõtjate nimekiri
- Kindlustusvõtjate arv müüjate järgi

Kõiki raporteid peab saama käivitada erinevate ajaperioodide kohta.
Kogutud preemiad peavad olema talletatud hüvitisliigi täpsusega. Hüvitisliik on kindlustuskaitse hierarhias kõige madalamal tasemel. Näiteks hoone kindlustuskaitse – veeavarii – taastamiskulud hierarhias on hüvitisliigiks taastamiskulud (veeavarii on risk). Vajalikud atribuudid ja omadused kindlustusliikide, agentide, kindlustusvõtjate ja kindlustatavate objektide kohta kirjeldatakse ära modelleerimise peatükis.

Raportite tulemusi peab olema võimalik kuvada nii tabulaarsel kujul kui ka graafiliselt, diagrammidena.

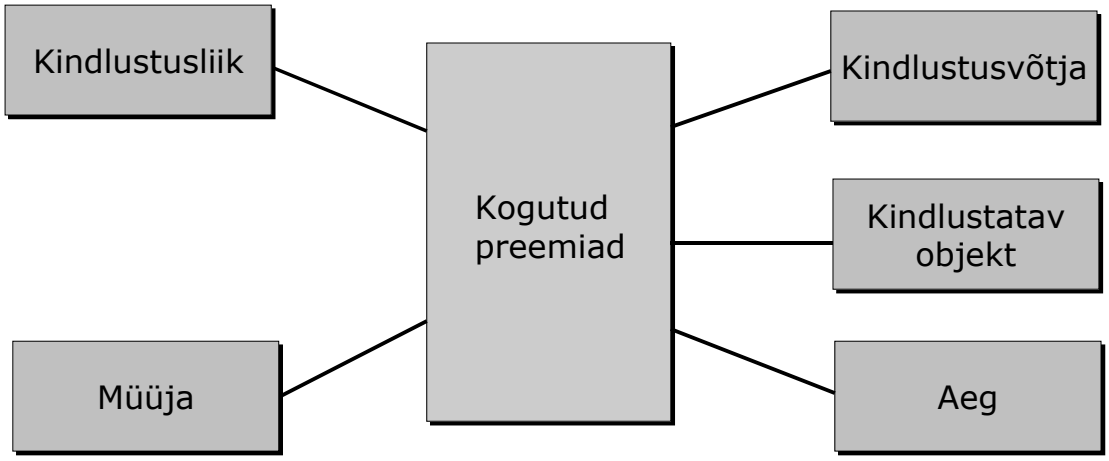
### 3.1.4. Kasutatav riistvara ja tarkvara

Parema platvormi puudumisel on arendus- ja testkeskkonnana kasutusel autori enda tööjaam. Olulisemad parameetrid: protsessor AMD XP2000+, 1 GB operatiivmälu ja operatsioonisüsteemiks Windows XP. Arenduseks kasutatav tarkvara on IBM-i andmebaasisüsteem DB2 8.1 Personal Edition. DB2 sai valitud, sest ta sisaldab kõiki andmehoidla loomiseks vajalikke komponente ning lisaks on Personal Edition kasutamiseks tasuta. Tasuta on mainitud versioon seetõttu, et teda ei saa kasutada serverina, küll sobib ta aga arenduseks. IBM pakub DB2-st ka spetsiaalselt andmehoidlate tarbeks kohandatud varianti "Warehouse Edition", mis sisaldab muuhulgas OLAP-i, andmete kaevandamise tarkvara ning klasterite toetust. Kaalutud sai ka Oracle 9i kasutamist, aga tuli kõrvale jätta, sest ta ei sisalda eraldamise, transformatsiooni ja laadimise tarkvara.

Raportite väljastamiseks on kasutusel Business Objects nimelise firma toode Crystal Reports. Antud tarkvara suudab ühenduda enamike tänapäeval levinud andmebaasisüsteemidega, toetab interaktiivsete raportite loomist, võimaldab raportite tulemusi kuvada laia valiku diagrammidena ning eksportida neid näiteks HTML, XML, PDF ja RTF formaati.

### 3.1.5. Andmete modelleerimine

Esmalt tuleb luua kontseptuaalne skeem andmehoidla jaoks. Andmehoidla teema ehk see osa skeemist, mis fakte esitab, on kogutud preemiad. Eelnevalt defineeritud raportite nõudmisi uurides selguvad ka analüüsidimensioonid, milleks on kindlustusliik, müüja, kindlustusvõtja, kindlustatav objekt ning alati vajalik aeg. Loodava andmehoidla kontseptuaalset skeemi võib seega kujutada järgmiselt:




**Joonis 8.** Kindlustusfirma andmehoidla kontseptuaalne skeem.

Tegemist on täht-skeemiga, sest üheski dimensioonis ei esine hierarhiaid. Ühtne regiooni dimensioon puudub, sest informatsioon aadressi, postiindeksi ja maakonna kohta on salvestatud kindlustusvõtja dimensioonis. Kuigi kindlustatava objekti aadress võib olla kindlustusvõtja omast erinev (linnas elav kindlustusvõtja võib kindlustada oma maamaja), ei säilitata seda andmehoidlas eraldi.

Järgnevalt tuleb kirjeldada dimensioonidele vastavad olemid ja nende atribuudid. Et ajaloo käsitlemisele suuremat rõhku asetada, peab kõik komponendid (olemid, atribuudid, seosed) klassifitseerima, olenevalt sellest, kuidas nende eelmisi (ajaloolisi) väärtusi kohelda tuleb. Seda klassifikatsiooni nimetatakse retrospektsiooniks ning tal on kolm võimalikku väärtust [7]:

1. Tõene
2. Väär
3. Püsiv

Tõene retrospektsioon tähendab, et objekt peegeldab minevikku tõeselt ehk omab tõeseid ajaloolisi väärtusi. Objektil võib olla katkendlik elutsükkel ehk mitmeid aktiivseid perioode vaheldumisi mitteaktiivsete perioodidega. Tõene retrospektsioon on andmehoidla objekti elu kõige täpsem kujutamine. Väär retrospektsioon tähendab, et vaade ajaloole muutub, kui objekti väärtus muutub. Teisisõnu, kui muutus toimub, siis läheb vana väärtus kaduma, sest ta kirjutatakse uuega lihtsalt üle. Jääb mulje, nagu poleks vanu väärtusi üldse eksisteerinudki. Püsiv retrospektsioon tähendab, et objekti väärtus ei muutu aja jooksul.

Andmete kirjeldamiseks kasutatakse meetodit nimega punktmodelleerimine (ingl k *dot modelling*) [7]. Selle põhjalikum kirjeldamine ei mahu kahjuks selle töö raamidesse, kuid antud meetodi notatsioon on üsna kergesti mõistetav ka ilma pikema seletuseta.

All olev tabel kujutab üht osa punktmodelleerimise töölehest, mis kirjeldab kindlustusvõtja olemit. Lisaks olemile endale on esitatud nii atribuudid, nende retrospektsiooni tüüp kui ka metaandmed. Samuti on töölehel olemas lahtrid tehnilise informatsiooni jaoks, nagu transformatsioon, andmetüüp, uuendamise sagedus ja primaarvõti (PV), aga nende täitmine peaks toimuma alles järgnevate peatükkide käigus. Täiskomplekti punktmodelleerimise töölehti kõikide olemite kohta võib leida lisast 1.

| **Olemi nimi** | | **Retrospektsioon** | **Sagedus** |
|---|---|---|---|
| Kindlustusvõtja | | Tõene | |
| **Metaandmed**<br>Kindlustusvõtja olem sisaldab kõiki kindlustusvõtja detaile. Olemi eksistents võib olla katkendlik ehk lõppeda ühel hetkel ja alata mingi ajaperioodi pärast uuesti. | | | |
| **Atribuudi nimi** | Isikukood | | **PV?** |
| **Retrospektsioon** | **Sagedus** | | **Sõltuvus** |
| Püsiv | | | Puudub |
| Metaandmed:<br>Kindlustusvõtja unikaalne isikukood. | | | |
| **Allikas** | **Transformatsioon** | | **Andmetüüp** |
| O&D | | | |

**Tabel 1**. Osa punktmodelleerimise töölehest.

Iga dimensiooni olemi atribuutide valik sõltus sellest, missugused atribuudid olid lähtesüsteemist kättesaadavad ning mis tundus mõistlik tabeleid ja nende sisu uurides. Näiteks tabelil, mis lähtesüsteemis ehk O&D andmebaasis sisaldab isikute andmeid, on olemas veerud autojuhilubade olemasolu ning kodakondsuse kohta. Kumbagi aga pole mõistlik kindlustusvõtja atribuutide hulka lisada andmehoidlas, sest autojuhilubade veergu ei täideta O&D süsteemis ja kodakondsus on peaaegu kõigil soome. Valitud said lühidalt järgnevad atribuudid:

Kindlustusvõtjat kirjeldavad atribuudid: isikukood, perekonnanimi, eesnimed, sünnikuupäev, sugu, kliendinumber, aadress, postiindeks, maakond.

Müüjat kirjeldavad atribuudid: id, perekonnanimi, eesnimed, müüja tüüp, otsene ülemus, kontori number, töötelefon, mobiiltelefon.

Kindlustusliiki kirjeldavad atribuudid: kindlustuse tüüp, hüvitisliigi nimi, riski nimi, kindlustuskaitse nimi.
Kindlustatavat objekti kirjeldavad atribuudid: oid, tüüp, alamtüüp, kindlustussumma.
Aega kirjeldavad atribuudid: kuupäev, päeva number, päeva nimi, nädala number, kuu number, kuu nimi, kvartal, aasta.

## 3.2. Andmehoidla arendamine

Käesolev osa tööst tegeleb andmehoidla reaalse implementatsiooniga. Järgnevate peatükkide käigus luuakse andmehoidla andmebaas, tegeletakse andmete eraldamise, transformatsiooni, laadimisega ning valmistatakse ette raportid. Käsitlemist leiavad arendamise käigus tekkinud probleemsed olukorrad ning neile leitud võimalikud lahendused.

### 3.2.1. Andmehoidla andmebaas

Andmebaasi loomine on enamasti ühekordne tegevus ning selleks on kõige mugavam kasutada vastavat viisardit, mida saab käivitada DB2 Control Centerist, mis on loodud kogu andmebaasisüsteemi keskseks majandamiseks. Pärast andmebaasi loomist võib seadistamiseks käivitada DB2 Configuration Advisori ehk siis "konfiguratsiooni nõuniku". Tegemist on samuti viisardiga, kus on võimalik muuhulgas määrata, et loodud andmebaas on optimeeritud andmehoidla tüüpi päringutele. Tabelite loomiseks võib kasutada vastavat interaktiivset viisardit või juhul, kui soovitakse käsitsi loomise laused käivitada, siis DB2 Command Centerit.

Kõigi tekstiliste veergude puhul on kasutatud VARCHAR andmetüüpi ruumi säästmiseks ning põhjusel, et tavaline CHAR andmetüüp ei oma mingit eelist. Rahaliste väärtuste ja teiste ujuvkomaga arvude salvestamiseks kasutatakse andmetüüpi DECIMAL(9,2), mis tähendab, et sellise veeru väärtus võib olla maksimum üheksakohaline arv koos kahe komakohaga. Kuigi DB2 toetab Unicode märgistikku ja seeläbi ka täpitähti, on nende kasutamist siiski välditud. Osaliselt on vältimise põhjuseks DB2 laadimistarkvara, mis kasutamisel andis mõnikord veateateid täpitähtedega veergude kohta. Kas tegelik probleem

oli tarkvaras või selle kasutajas, jäi selgusetuks.

Järgnevalt on ära toodud andmehoidla fakti- ja dimensioonitabelite loogilised skeemid. Kõik täpsed andmehoidla tabelite loomise laused (CREATE TABLE) on kirjas lisas 2 ja duplitseerida neid käesolevasse peatükki ei ole mõtet.

**Kindlustusvotja-skeem** = {isikukood, perekonnanimi, eesnimed, synnikp, sugu, kliendinumber, aadress, postiindeks, maakond, alguskp, lopukp }
Atribuut alguskp kuulumine primaarvõtmesse on vajalik ajalooliste versioonide säilitamiseks. Kuna see on tehniline veerg ja andmehoidla lõppkasutaja ei pea tema olemasolust teadlik olema, puudub ta punktmodelleerimise töölehelt (nagu ka lopukp). Määramata väärtus ehk NULL on lubatud veergudel kliendinumber, aadress, postiindeks, maakond ja lopukp.

**Myyja-skeem** = { id, perekonnanimi, eesnimed, myyja_tyyp, ylemus, kontorinr, telefon, mobiiltelefon, alguskp, lopukp }
Alguskp on vajalik sarnaselt kindlustusvõtja tabeliga ajalooliste versioonide säilitamiseks. Määramata väärtus võib olla veergudel: eesnimed, ylemus, kontorinr, telefon, mobiiltelefon, lopukp.

**Kindlustusliik-skeem** = { kindlustustyyp, hyvitisliiginimi, riskinimi, kaitsenimi }
Primaarvõtme ehk atribuudi kindlustustyyp puhul on tegemist komposiit identifikaatoriga, milles on ühendatud hüvitisliigi tüüp, riski tüüp ja kindlustuskaitse tüüp. Andmed kõigi kolme mõiste kohta on ühendatud kokku, et vältida tabelite hierarhia tekkimist ja seeläbi täht-skeemi muutumist lumehelves-skeemiks. Määramata väärtused ei ole selles tabelis lubatud ühegi veeru puhul.

**Kindlustatav-skeem** = { oid, tyyp, alamtyyp, ksumma }
Kuigi lähtesüsteemis kasutati oid veerge andmetüübiga CHAR(16) FOR BIT DATA paljudes tabelites identifikaatoritena, on siin tabelis olev veerg andmehoidla andmebaasis ainuke. Kindlustatavaid objekte võib olla väga erinevat tüüpi, seetõttu on tabelis ainult üldised atribuudid, mis on ühised kõigile kindlustatud objektidele. Lisaks oid-le peavad kindlasti olema täidetud ka tüüp ja alamtüüp, ksumma võib olla määramata.

**Aeg-skeem** = { kp, paevanr, paevanimi, nadalanr, kuunr, kuunimi, kvartal, aasta }
Primaarvõtmeks olev kp ehk kuupäev määrab ära ühtlasi andmehoidla granulaarsuse aja suhtes, mis tähendab, et preemia laekumise võib määrata kõige rohkem päeva täpsusega. Erinevalt toimunud müügist või telefonikõnedest, kus võib vaja olla sekunditäpsusega informatsiooni, ei ole laekunud preemiate puhul veel täpsema teabega midagi peale hakata. See tabel on ka ainuke, mille sisu on võimalik lihtsalt genereerida ning ei vaja andmeid lähtesüsteemist. Määramata väärtused ei ole lubatud ühegi veeru puhul.

**Preemia-skeem** = { kindlvotja_id, myyja_id, kindlustus_id, kindlustatav_id, aeg_id, netopreemia }
Täht-skeemi puhul on tavaliselt reegliks, et faktitabeli primaarvõti koosneb kõikide dimensioonide primaarvõtmete ühendist ning andmebaasi terviklikkuse tagamiseks on kõik võtmeks olevad veerud faktitabelis defineeritud ka võõrvõtmetena. Käesoleval juhul ei ole aga see täies ulatuses võimalik, sest kahe dimensioonitabeli – Kindlustusvotja ja Myyja – primaarvõti koosneb kahest atribuudist. Mõlema tabeli korral on põhjuseks alguskp atribuut, mida on vaja ajalooliste versioonide salvestamiseks. Seetõttu on loobutud ka teiste faktitabeli primaarvõtme veergude võõrvõtmena defineerimisest. Tulemuseks võib olla situatsioon, kus faktitabelis on andmeid, millele dimensioonitabelites vastavad andmed puuduvad. Seega peab käesoleval juhul püüdma andmebaasi viitetervikliikkust tagada juba andmete eraldamise käigus, enne andmehoidlasse sisestamist.

#### 3.2.1.1. Indeksid

Dimensioonitabelitele indekseid luua ei ole vaja, sest primaarvõtmeks olevatele veergudele tehakse indeksid automaatselt andmebaasihalduri poolt juba tabeli loomise käigus. Põhjusel, et enamus faktitabelit kasutavatest päringutest loevad sealt enam kui poolt andmetest, ei ole mõistlik ka faktitabelile ühtki indeksit luua.

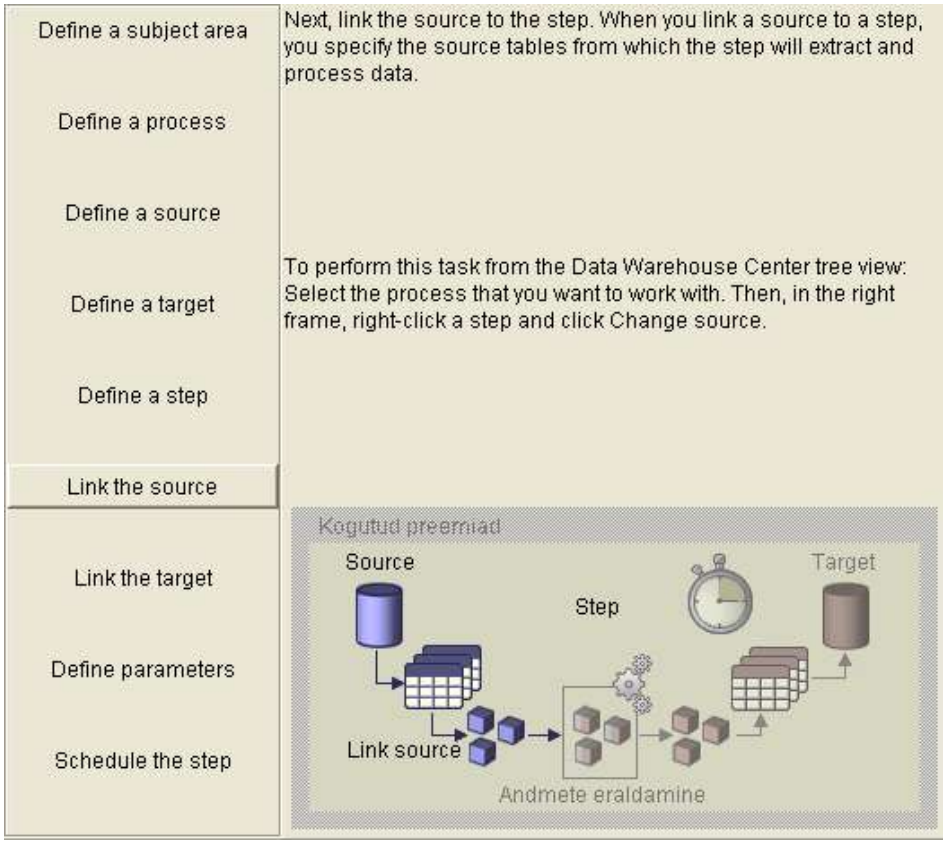
#### 3.2.1.2. Summatabelid

DB2 relatsioonbaasihaldurisse on summatabelite toetus sisse ehitatud (vähemalt alates versioonist 7.1, varasemate versioonidega kogemus puudub). DB2 dokumentatsioonis ja

juhendites ei nimetata neid küll summatabeliteks, vaid materialiseerunud päringutabeliteks (ingl k *materialized query table* – MQT). Viis sellist tabelit sai loodud ka andmehoidlasse – üks summatabel iga dimensiooni kohta). Nende täpsed loomise laused on samuti kirjas lisas 2 (laused 7 – 11). Pärast loomist tuleb need tabelid täita kasutades käsku REFRESH TABLE, millele järgneb vastava tabeli nimi. See on õnneks ühekordne tegevus, sest tänu võtmesõnadele REFRESH IMMEDIATE tabelite loomisel uuendatakse neid alati automaatselt.

### 3.2.2. Andmete eraldamine

Edasine töö toimub DB2 Data Warehouse Center nimelise tarkvaraga, mis on mõeldud kogu eraldamise, integratsiooni/transformatsiooni ja laadimise protsessi defineerimiseks ja haldamiseks. Suureks abiks on sealjuures viisardilaadne "stardiplatvorm" (Launchpad), sest programmi põhiaknas olev skeemide, transformatsioonikomponentide, lähte- ja sihtsüsteemide "rägastik" kipub alguses segadust tekitama. Nimetatud stardiplatvorm on kuvatud ka joonisel 9.




**Joonis 9.** DB2 Data Warehouse Center Launchpad.

Warehouse Center eeldab järgmist töövoogu: esmalt tuleb defineerida teema (ingl k *subject area*), siis selle teema alla üks või enam protsessi (ingl k *process*). Protsesside alla tuleb omakorda kirjeldada allikad (ingl k *source*) ning sihtmärgid (ingl k *target*), mis võivad olla näiteks failid või andmebaasitabelid. Seejärel tuleb defineerida mingi loogiline samm (ingl k *step*), milleks võib olla andmete eksport, SQL-keele lause või salvestatud protseduuri käivitamine, statistilised funktsioonid ja palju muud. Peale seda tuleb kokku ühendada selleks sammuks vajalik sisend (allikas) ja väljund (sihtmärk) ning olenevalt sammu tüübist, määrata parameetrid. Kogu töövoogu tuleb korrata, kuni kõik vajalik on defineeritud, misjärel tuleb paika panna ajakava (ingl k *schedule*), kus on kirjas, mis ajal, mis järjekorras ja mitu korda neid samme käivitama peab. Kõige selle tulemuseks peab olema valiidse informatsiooniga täidetud andmehoidla andmebaas.

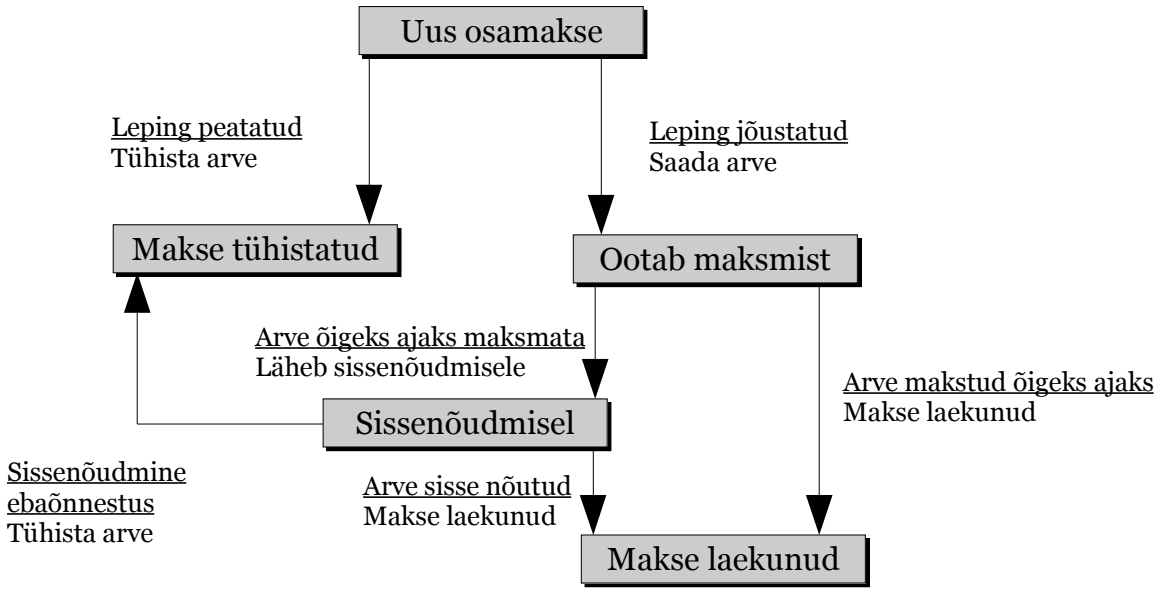
Suurimad probleemid andmehoidla arendamisel tekkisid andmete eraldamine või täpsemalt valiidsete andmete eraldamise juures. Põhjus on lähtesüsteemiks oleva O&D andmebaasi struktuur ja andmed, mida see andmebaas sisaldab. O&D ajalooliste andmete toetus on keerukas ning võimalusterohke, kuid andmete kättesaamine on nende võimaluste tõttu samuti raskem. Ükski süsteem ei ole täiuslik ning O&D ei ole selles suhtes mingi erand. Aastate jooksul on tekkinud andmebaasi vigaseid andmeid, mida on ka parandatud, kuid osa on kahjuks märkamata jäänud. Andmehoidla tarbeks andmete eraldamisel tulid mõned sellised veaolukorrad ka ilmsiks.

Tekkis situatsioone, kus päringu tulemusena oleks valiidsete andmete hulgas tagastatud ka mittevaliidseid ehk vigaseid andmeid. Sama päringu korrigeerimisel kujule, mis välistaks vigaste andmete tagastamise, tekkis omakorda oht, et välja jääb ka osa valiidsetest andmetest. Teine päring võis jällegi tagastada õiged andmed, kuid olla samas talumatult aeglane. See sundis otsima alternatiive, kuidas päringut teisiti formuleerida, aga saada sama tulemus.

Kõigi nende faktorite tõttu ei saa 100%lise kindlusega väita, et loodud andmehoidlasse sai operatiivsest süsteemist kokku koondatud absoluutselt kõik valiidsed andmed. Kuna tegemist on testkeskkonnaga, mitte reaalse produktsioonikeskkonnaga, võib saavutatud täpsust siiski piisavaks lugeda. Järgnevalt eraldamise protsessist lähemalt.

#### 3.2.2.1. Preemia laekumise tuvastamine

Esmalt tuleb tuvastada, millal on andmehoidla teema jaoks oluline sündmus toimunud ehk millal võib preemia üldse kogutuks või laekunuks lugeda. O&D kindlustussüsteemis on see võimalik, uurides osamakseid kirjeldava tabeli (Installment) staatuse veergu (inststatus). Uue, äsja süsteemi sisestatud osamakse puhul sisaldab see veerg väärtust 10, makstud on 20, sisse nõutud 50 ja tühistatud tähistab 80. Selle informatsiooni põhjal saab koostada vastava olekute ülemineku diagrammi, mis on kuvatud joonisel 10. Seda tüüpi diagrammidest on juttu ka peatükis 2.2.




**Joonis 10**. Olekute ülemineku diagramm osamakse elutsükli kohta.

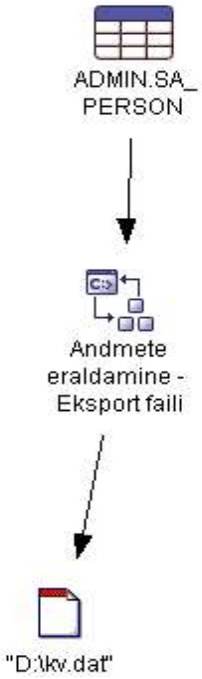
Esimest korda andmehoidlat täites tuleb niiöelda "pumbata" O&D andmebaasist välja andmed kõikide selliste osamaksete kohta, kus staatus on "Makstud" või "Sisse nõutud". Edaspidi aga on vaja andmehoidlat täiendada mingi ajavahemiku tagant nende maksete andmetega, mis sel perioodil on laekunud. Uuesti ja pidevalt kõiki osamaksete andmeid ümber tõsta ei ole mõistlik, sest protsess on üsna aeganõudev. Selle dilemma lahendamiseks sai loodud päästikprotsess, mis jälgib muutusi osamakse tabelis ning reageerib neile. Tabeli staatuse veeru väärtuse muutumisel makstuks või sissenõutuks käivitatakse SQL-lause, mis võtab muutunud osamakse unikaalse identifikaatori (Oid), lepingunumbri, kliendinumbri, laekumise kuupäeva ja salvestab need andmed spetsiaalselt

loodud vahetabelisse. Teoreetiliselt oleks võimalik käivitada SQL-lause, mis ühendades mitmeid tabeleid salvestaks korraga rohkem andmeid, mitte ainult antud neli atribuuti. Tabelite ühendamise puhul aga kulub SQL-lausel oma töö lõpetamiseks mitmeid kordi rohkem arvutusressurssi ning see võib operatiivse süsteemi tööd häirida. Eelmainitud päästikprotsessi loomise süntaks DB2 andmebaasi jaoks asub lisas 2 (lause 17).

Täiendamise nõue kehtib ka andmehoidla dimensioonitabelite kohta. Kuid O&D andmebaasi tabelid, mille andmetega täidetakse dimensioonitabelid, on käesoleval juhul suhteliselt väikese mahuga, mistõttu võib kogu nende sisu uuendamise käigus andmehoidlasse ümber tõsta.

#### 3.2.2.2. Vahetabelid

Lähtesüsteemi ehk O&D andmebaasi luuakse spetsiaalselt vahetulemuste talletamiseks eraldi tabelid ehk vahetabelid. Neid tabeleid on vaja, et eraldada ainult vajalikud andmed nii eelmainitud osamaksete kui ka isiku, kindlustuspoliisi ja hüvitisliigi kohta. Vahetabelite loomise laused lisas 2 on numbritega 12 - 15 ning nende täitmise laused 18 - 21. Erinevalt andmehoidlast on nende tabelite nimed ja atribuudid inglise keelsed, sest kõik teised O&D tabelid on samuti inglise keelsete nimedega. DB2 Warehouse Manager tarkvara hoolitseb olenevalt sihtmärgist reaalse andmete sisestamise eest ise ja seetõttu on vaja defineerida ainult SELECT-tüüpi laused, kuid lisasse on jäetud ka lause INSERT osa, et oleks selge, mis tabeli täitmine toimub. Andmete allikat ja sihtmärki on mainitud tarkvaras võimalik määrata ka graafiliselt komponente ühendades, nagu kuvatud joonisel 11.




**Joonis 11**. Komponentide graafiline ühendamine.

Nagu ülalolevalt jooniseltki näha, sai testimise eesmärgil proovitud ka andmete eraldamist failidesse, kus veergude väärtused on komaga eraldatud. Protsess töötas laitmatult ning vajadusel võib seda kasutada alternatiivina.

### 3.2.3. Andmete integratsioon

Integratsiooni protsess kulges ilma märkmisväärsete viperusteta, sest vajalikud muutused olid suhteliselt lihtsad. Nii formaadi, kui sisulisteks muudatusteks kasutati SQL-keele funktsioone. Lisaks juba olemasolevatele vahetabelitele tuli lisaks luua ainult üks - kindlustusvõtja andmete jaoks. Mainitud tabeli loomise ja täitmise laused on lisas 2, numbritega 16 ja 22. Ülejäänud muutused on nii lihtsad, et ei olnud otstarbekas eraldi tabelit luua, vaid konverteerimine toimus otse sama SQL lause käigus, mis ühtlasi andmed andmehoidlasse laadis.

Lihtne näide formaadi konverteerimisest: CHAR andmetüübiga atribuudist VARCHAR saamiseks rakendati veerule funktsiooni VARCHAR() ja seejärel funktsiooni RTRIM(), mis eemaldas teksti lõpust ülemäärased tühikud. Sisulise muutuse näitena võib tuua

inimese sugu tähistava atribuudi transformatsiooni. O&D süsteemis on sugu tähistav atribuut numbriline, kus mehele vastab väärtus 10 ja naisele 20. Andmehoidlas on vastav atribuut tekstiline, pikkusega üks tähemärk ja võimalike väärtusega “M” ja “N”. Konverteerimisel kasutati SELECT-lause sees CASE lauset, mis on sarnane paljudes programmeerimiskeeltes kasutatava switch/case lausega. Selle süntaks on lihtne:

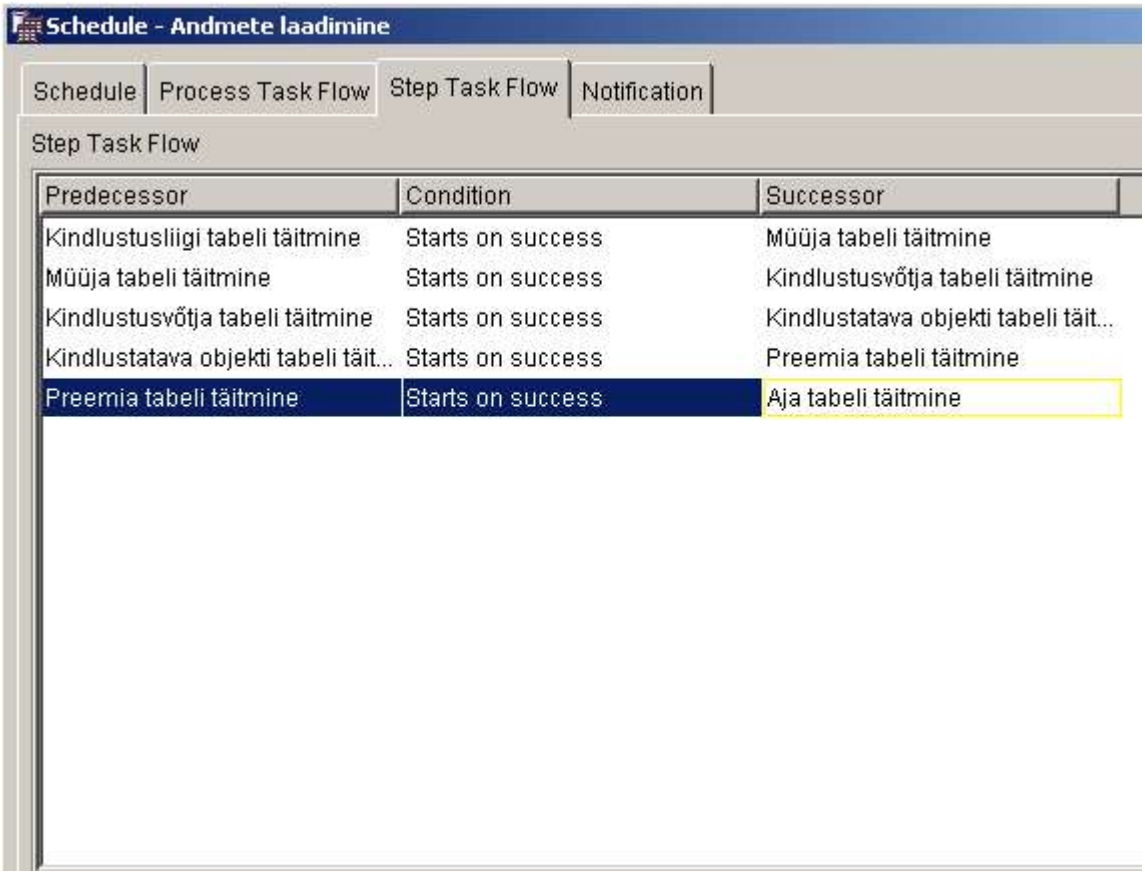
```
CASE
  WHEN P.sex = 10 THEN 'M'
  WHEN P.sex = 20 THEN 'N'
  ELSE 'L'
END
```

Transformatsioonide tüübid on kõik kirjas punktmodelleerimise töölehtedel vastavate atribuudite kirjelduste juures lisas 1 ning nende tüüpide seletused asuvad lisas 3.

### 3.2.4. Andmete laadimine

Andmete laadimine toimus identselt teiste protsessidega - SQL-i abil ühe andmbaasi tabelitest teise andmebaasi tabelitesse. Teine võimalik lahendus oleks olnud eelnevalt eraldada vajalikud andmed failidesse ja siis sealt andmehoidlasse laadida. Kuid sellise variandi testimisel tekkis paar probleemi. Üks nendest oli seotud täpitähtedega - andmefailides olid need korrektsel kujul olemas, kuid pärast laadimist andmehoidla vastavat tabelit uurides olid need asendunud arusaamatute märkidega. Ei aidanud ka laadimise erinevate parameetrite, näiteks märgistiku, muutmine. Teine probleem seisnes oid veerus, mille andmetüüp (CHAR(16) FOR BIT DATA) vajab erikohtlemist. Failis on ta kuueteistkümnendsüsteemis olev 32 märgi pikkune sõne, näiteks väärtusega “9C31F5D6FCE3D511AF8700807CF18A3A”. Failist laadimise parameetrites aga ei ole võimalik määrata, et väärtus on vaja binaarseks konverteerida ja tarkvara arvab, et tegemist on tavalise 32 märgi pikkuse sõnega.

Esmakordne laadimine erineb andmehoidla järgnevatest uuendamistest ainult ridade arvu poolest osamaksete vahetabelis. See tähendab, et osamaksete vahetabeli täitmist (lause 18) tehakse ainult üks kord ning edaspidi tekivad sinna andmed päästikprotsessi (lause 17) tegevuse tulemusena. Nii laadimise kui ka teiste protsesside automatiseerimiseks, käivitamise järjekorra ja aja määramiseks kasutati DB2 Warehouse Manageri sisse ehitatud plaanurit. Plaanur on kuvatud ka alloleval joonisel 12.




**Joonis 12**. Protsesside ja sammude järjekorra määramine plaanuris.

Lõpuks ka natuke statistikat ehk kui kaua andmebaasi tabelite täitmine aega võttis ja kui mitu rida esimese laadimise tulemusena seal oli. Lõviosa ajast kulus tõenäoliselt võrguliikluse peale, sest andmeid tuli liigutada Soomes asuvast serverist kohalikku tööjaama.

Preemia tabel – laadimine võttis aega ligikaudu 8 tundi, tabelis ridu – 7546047

Kindlustusvõtja tabel – laadimine 2 minutit, tabelis ridu - 38415

Müüja tabel – laadimine 20 sekundit, tabelis ridu – 3993

Kindlustatava objekti tabel – laadimine 20 minutit, tabelis ridu - 230371

Kindlustusliigi tabel – laadimine 3 minutit, tabelis ridu – 67

Aja tabel – laadimine 30 minutit, tabelis ridu – 1694

DB2 Warehouse Manager tarkvara juures häiris veel fakt, et ei ole võimalik defineerida loogilist sammu, mis kasutaks SQL-i DELETE-tüüpi lauset. Seda oleks vaja, et pärast laadimist näiteks vahetabeleid tühjendada.

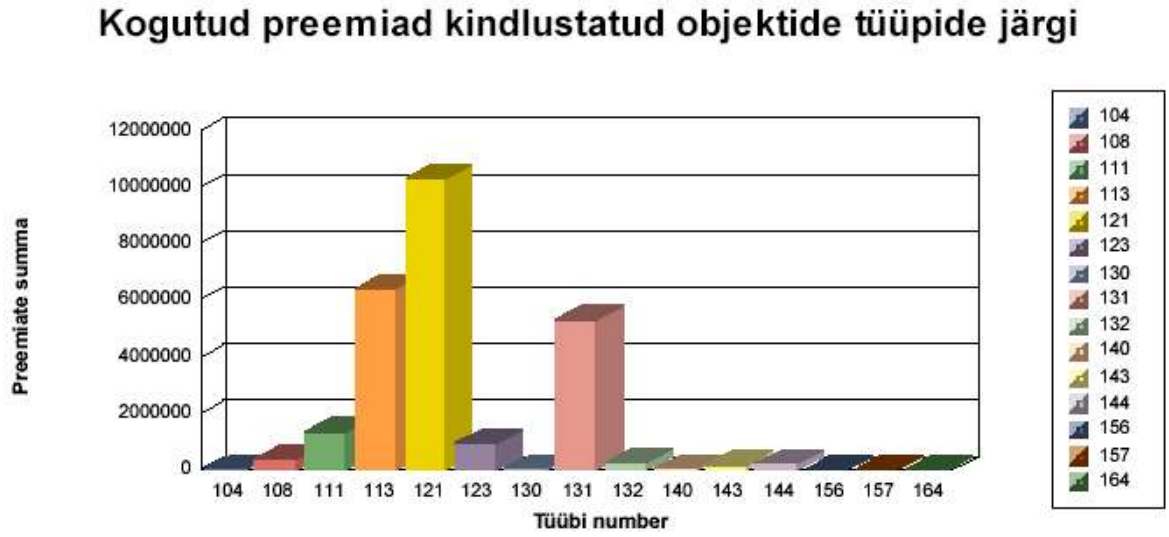
### 3.2.5. Juurdepääs andmetele

Selles peatükis jõutakse lõpptulemuseni – raportiteni, mille nõudmised said esitatud peatükis 3.1.3.

Tarkvara, mille abil raportid on koostatud - Crystal Reports – ei võimalda otse SQL päringut sisestada, vaid vajalikud tabelid ja atribuudid tuleb määrata erinevate parameetriakende ja viisardite abil. Seetõttu ei ole eraldi ära toodud ka päringute SQL-lauseid. Nimetatud tarkvara suutis ka korrektselt ühenduda DB2 serveriga, kuid ei võimaldanud päringutes kasutada summatabeleid või ei suutnud nende olemasolu tuvastada.

Käesolevas töös olevad raportite näited on püütud hoida suhteliselt lihtsad, et nad ühele leheküljele ära mahuksid ja samas ka loetavad oleksid. Siia peatükki on ära toodud kaks raportit, ülejäänud asuvad lisas 4. Kuna isikuandmeid ei ole lubatud avaldada, on müüjate ja kindlustusvõtjate kohta kirjas ainult sisemise koodina kasutuses olevad id ja kliendinumber.

All olev joonis 13 kuvab, nagu pealkirigi ütleb, kogutud preemiaid kindlustatud objektide tüüpide järgi. Siin on kohane mainida veel üht Crystal Reportsi puudust. Nimelt ei leidnud võimalust määrata diagrammi juures tüübi numbritele tekstilisi väärtusi.




**Joonis 13**. Raport kogutud preemiate kohta kindlustatud objekite tüüpide järgi.

Järgnev joonis, numbriga 14, kuvab kogutud preemiaid kindlustusliikide järgi tabulaarsel kujul. 2004. aasta andmed hõlmavad nelja esimest kuud.

| | 2 000 | 2 001 | 2 002 | 2 003 | 2 004 | Total |
|---|---|---|---|---|---|---|
| Total | 1 422 010,33 | 3 686 321,96 | 6 747 192,52 | 9 399 643,16 | 3 728 421,17 | 24 983 589,14 |
| Eläinturva | 21 360,43 | 55 085,93 | 108 893,86 | 153 614,99 | 56 209,49 | 395 164,70 |
| Henkilön toiminnan | 3 103,09 | 4 381,40 | 5 852,83 | 5 915,31 | 2 124,32 | 21 376,95 |
| Huoneistoturva | 2 802,54 | 6 762,46 | 12 202,80 | 16 848,57 | 6 239,27 | 44 855,64 |
| Irtaimistoturva | 356 436,51 | 944 024,53 | 1 735 103,64 | 2 361 253,34 | 906 508,57 | 6 303 326,59 |
| Kiinteistöturva | 246 881,89 | 705 982,68 | 1 363 598,32 | 1 981 080,34 | 814 438,86 | 5 111 982,09 |
| LVISA-turva | 43 851,14 | 130 201,68 | 258 933,73 | 366 745,08 | 146 324,80 | 946 056,43 |
| Metsäturva | 1 109,93 | 4 956,27 | 11 367,65 | 20 326,96 | 8 545,54 | 46 306,35 |
| Ryhmän toiminnan | 82 070,57 | 205 087,97 | 360 564,85 | 489 647,81 | 190 956,92 | 1 328 328,12 |
| Tonttiturva | 11 163,26 | 31 913,22 | 62 115,12 | 85 323,04 | 33 352,90 | 223 867,54 |
| Täysajan henkilö | 634 699,40 | 1 552 190,54 | 2 732 308,56 | 3 763 309,05 | 1 513 603,32 | 10 196 110,87 |
| Veneturva | 18 531,57 | 45 735,28 | 96 251,16 | 155 578,67 | 50 117,18 | 366 213,86 |

Kogutud preemiad kindlustusliigi ja aasta järgi

**Joonis 14**. Raport kogutud preemiate kohta kindlustuliikide järgi.

Seega on kõigile esiletoodud probleemidele lahendus leitud ja rahuldava lõpptulemuseni jõutud.

## Kokkuvõte

Andmehoidla tähendab eelkõige strateegilise informatsiooni kättesaadavaks tegemist. Andmehoidla koondab sel eesmärgil andmed ühtsesse tsentraalsesse andmebaasi, võimaldades kasutajatel kergesti strateegilisele informatsioonile ligi pääseda ja seda analüüsida. Kuna rõhk on seatud analüütilise ja strateegilise iseloomuga küsimustele vastamises, on andmehoidlad tihti nõustussüsteemide aluseks. Nõustussüsteemi otstarve on varustada ettevõtte või organisatsiooni otsusetegijaid neile vajaliku informatsiooniga.

Käesolevas töös on antud ülevaade andmehoidlatest ja nende arendusprotsessist. Täpsemalt vaadeldakse andmehoidlate tähtsamaid komponente – tsentraalset andmebaasi, andmete eraldamist lähtesüsteemidest, andmete integratsiooni, nende laadimist andmehoidlasse. Käsitletakse ka järgmisi andmehoidla arendamist puudutavaid probleeme:

- Andmete modelleerimine
- Operatiivsetest süsteemidest andmete eraldamine
- Ajalooliste andmete käsitlemine
- Andmehoidla andmetele ligipääsu ja raportite tarkvara

Põgusalt tutvustatakse ka andmehoidlatega seotud tehnoloogiaid – andmete kaevandamist, andmevakkasid, OLAP-i.

Arendusprotsessi ja selle käigus tekkivaid probleeme uuritakse praktilise näite varal – luues reaalse andmehoidla süsteemi. Loodud andmehoidla baseerub reaalselt eksisteeriva Soome kindlustusfirma andmetel ning selle teemaks on kindlustusfirma poolt kogutud preemiad. Arendamisel tuli lahendada kõiki eelnevalt esiletoodud küsimusi, kuid suurimaks probleemiks oli kahtlemata andmete eraldamise protsess. Erinevaid lahendusvariante proovides sai lõpuks ka see tõke ületatud ning eduka lõpptulemuseni jõutud.

## Summary

In today's business climate companies are feeling the pressure to cut expenses and increase profits. The advantage goes to those companies that can respond to rapidly changing demands. Employees who have to make business decisions have limited knowledge. The large volumes of data that companies collect during their day-to-day operations are not being delivered back to business users in the form of usable information. Much of the information needed for making accurate, insightful decisions is locked up in data sources that are difficult to access. Companies have made large investments in operational systems to support their business activities, but those systems commonly don't allow users to easily access and analyze the information contained within them.

A data warehouse offers the solution to that problem by storing a company's business data in an integrated, subject-oriented database. It provides a historical perspective of information for decision-support and enables end users to quickly and directly access and analyze business data.

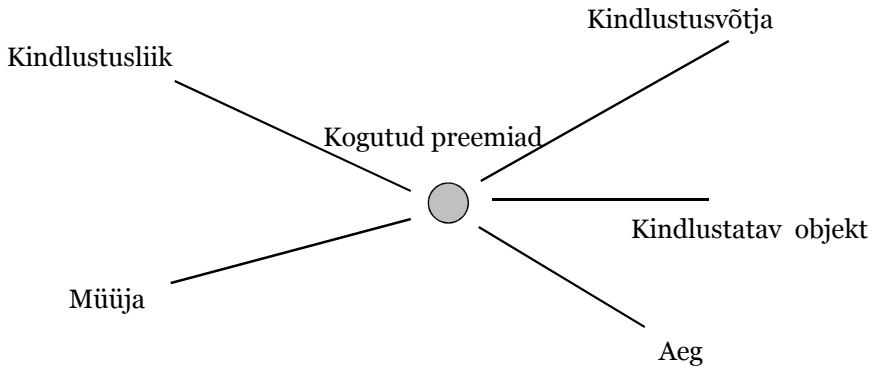
The aim of this diploma work is to explain the essence of data warehouses, why and where they are used and to describe the problems that one may encounter in the data warehouse development process. The development issues are illustrated through the implementation of a real data warehouse system. The data for the warehouse is based on the operative system of a Finnish mutual insurance company. The subject of the created data warehouse is the premiums collected by the insurance company.

## Kasutatud kirjandus


1. **Berson, A., Smith, Stephen J.** Data warehousing, data mining & OLAP, McGraw-Hill, 1997, 612 pg.
2. **DSstar**. Data warehousing market poised for growth, http://www.tgc.com/dsstar/98/0915/100301.html, [14.03.2004].
3. **Inmon, W. H.** Building the Operational Data Store, John Wiley & Sons, Inc, 1996, 276 pg.
4. **Pendse, Nigel**. What is OLAP?, http://www.olapreport.com/fasmi.htm, [14.03.2004].
5. **KeeleWeb**. Infotehnoloogia terministandard 2382, http://ee.www.ee/ITterminid/, [14.03.2004].
6. **Orr, Ken.** Data Warehousing Technology, White Paper, www.kenorrinst.com/dwpaper.html, [14.03.2004].
7. **Todman, Chris.** Designing a data warehouse: supporting customer relationship management, Prentice Hall PTR, 2001, 323 pg.
8. Vallaste, H. E-teatmik, http://www.vallaste.ee, [14.03.2004].
9. **Vidette, Poe.** Building a data warehouse for decision support, Prentice Hall PTR, 1996, 210 pg.

**Lisa 1. Punktmodelleerimise töölehed kindlustusfirma andmehoidla jaoks.**

| **Punktmodelleerimise andmemudel** | |
|---|---|
| **Mudeli nimi:** Kindlustusfirma kogutud preemiad | |
| Mõõdetavad faktid | |
| **Fakti nimi** | **Metaandmed** |
| Netopreemia | Netopreemia ehk lõplik preemia, mis kindlustusfirmale laekub. Kõik maksud ja lisatasud on netopreemiasse juba sisse arvestatud. Tagasimakse korral võib väärtus olla ka negatiivne. |
| Diagramm | |




| **Punktmodelleerimine - Aeg** | | |
|---|---|---|
| **Atribuudi nimi** | **Kirjeldus** | **Andmetüüp** |
| Kp | Kuupäev (aaaa-kk-pp) | DATE |
| Päeva number | Päeva number 1 – 7 (esmaspäev = 1) | SMALLINT |
| Päeva nimi | Tavalised päevade nimed | VARCHAR(11) |
| Nädala number | Nädala number 01 – 52 | SMALLINT |
| Kuu number | Kuu number 01 -12 (jaanuar = 1) | SMALLINT |
| Kuu nimi | Tavalised kuude nimed | VARCHAR(9) |
| Kvartal | Kvartalite numbrid (1, 2, 3, 4) | SMALLINT |
| Aasta | Aasta number | SMALLINT |

| **Punktmodelleerimise olemid ja atribuudid** | | |
|---|---|---|
| **Olemi nimi** | **Retrospektsioon** | **Sagedus** |
| Müüja | Tõene | Igakuine |
| **Metaandmed**<br>Müüja olem sisaldab kõiki vajalikke müüja ehk kindlustusagendiga seotud detaile. Olemi eksistents võib olla katkendlik ehk lõppeda ühel hetkel ja alata mingi ajaperioodi pärast uuesti. | | |
| **Atribuudi nimi** | Id | **PV?** Jah |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Müüja identifikaator. See kood pärineb operatiivsest süsteemist. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |
| **Atribuudi nimi** | Perekonnanimi | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja perekonnanimi. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Tekst2 | VARCHAR(40) |
| **Atribuudi nimi** | Eesnimed | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja eesnimi või eesnimed. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Tekst2 | VARCHAR(40) |
| **Atribuudi nimi** | Müüja tüüp | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja tüüp. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |

| **Atribuudi nimi** | Ülemus | **PV?** Ei |
|---|---|---|
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja otsene ülemus. Väärtusena peab olema mingi teine sama tabeli id. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |
| **Atribuudi nimi** | Kontori number | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Kontori number, kus müüja töötab. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |
| **Atribuudi nimi** | Töötelefon | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja töötelefoni number. Väärtus võib ka puududa. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(20) |
| **Atribuudi nimi** | Mobiiltelefon | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Müüja mobiiltelefoni number. Väärtus võib ka puududa. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(20) |

| **Punktmodelleerimise olemid ja atribuudid** | | |
|---|---|---|
| **Olemi nimi** | **Retrospektsioon** | **Sagedus** |
| Kindlustusvõtja | Tõene | Igakuine |
| **Metaandmed**<br>Kindlustusvõtja olem sisaldab kõiki kindlustusvõtja detaile. Olemi eksistents võib olla katkendlik ehk lõppeda ühel hetkel ja alata mingi ajaperioodi pärast uuesti. | | |
| **Atribuudi nimi** | Isikukood | **PV?** Jah |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtja unikaalne isikukood. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(11) |
| **Atribuudi nimi** | Perekonnanimi | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtja perekonnanimi. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Tekst1 | VARCHAR(40) |
| **Atribuudi nimi** | Eesnimed | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtja eesnimi või eesnimed. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Ühinemine1 + Tekst1 | VARCHAR(40) |
| **Atribuudi nimi** | Sünnikuupäev | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtja sünnikuupäev. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | DATE |

| **Atribuudi nimi** | Sugu | **PV?** Ei |
|---|---|---|
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtja sugu. Võimalikud väärtused: M või N. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Tekst3 | VARCHAR(1) |
| **Atribuudi nimi** | Kliendinumber | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Kindlustusvõtjale määratud kliendinumber. Pärineb operatiivsest süsteemist. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | INTEGER |
| **Atribuudi nimi** | Aadress | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Tõene | Igakuine | Puudub |
| Metaandmed:<br>Aadress, kus kindlustusvõtja elab. Atribuudi eelnevaid väärtusi peab säilitama. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(64) |
| **Atribuudi nimi** | Postiindeks | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Tõene | Igakuine | Puudub |
| Metaandmed:<br>Kindlustuvõtja elukoha postiindeks. Atribuudi eelnevaid väärtusi peab säilitama. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Tekst2 | VARCHAR(10) |
| **Atribuudi nimi** | Maakond | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Tõene | Igakuine | Puudub |
| Metaandmed:<br>Kindlustuvõtja elukoha maakonna nimi. Atribuudi eelnevaid väärtusi peab säilitama. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(20) |

| **Punktmodelleerimise olemid ja atribuudid** | | |
|---|---|---|
| **Olemi nimi** | **Retrospektsioon** | **Sagedus** |
| Kindlustatav objekt | Väär | Igakuine |
| **Metaandmed**<br>Olem sisaldab kõiki kindlustatava objektiga seotud detaile. Kindlustatav objekt võib olla elutu (krunt, maja, kodune vara, elektroonika, väärisasjad, jne) kui ka elus (inimene, lemmikloom). | | |
| **Atribuudi nimi** | Oid | **PV?** Jah |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustatava objekti unikaalne identifikaator. See kood pärineb operatiivsest süsteemist. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | CHAR(16) FOR BIT DATA |
| **Atribuudi nimi** | Tüüp | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustatava objekti tüüp. Tüüpide definitsioonid pärinevad operatiivsest süsteemist. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |
| **Atribuudi nimi** | Alamtüüp | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustatava objekti alamtüüp. Tüüpide definitsioonid pärinevad operatiivsest süsteemist. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | SMALLINT |
| **Atribuudi nimi** | Kindlustussumma | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Väär | Igakuine | Puudub |
| Metaandmed:<br>Summa, mille peale on objekt kindlustatud. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Number1 | DECIMAL(9,2) |

| **Punktmodelleerimise olemid ja atribuudid** | | |
|---|---|---|
| **Olemi nimi** | **Retrospektsioon** | **Sagedus** |
| Kindlustusliik | Püsiv | Igakuine |
| **Metaandmed**<br>Olem sisaldab informatsiooni kindlustusliikide kohta. | | |
| **Atribuudi nimi** | Kindlustuse tüüp | **PV?** Jah |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Komposiit identifikaator, milles on ühendatud hüvitisliigi tüüp, riski tüüp ja kindlustuskaitse tüüp. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Ühinemine2 | INTEGER |
| **Atribuudi nimi** | Hüvitisliigi nimi | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Hüvitisliigi nimi. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(40) |
| **Atribuudi nimi** | Riski nimi | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Riski nimi. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(40) |
| **Atribuudi nimi** | Kindlustuskaitse nimi | **PV?** Ei |
| **Retrospektsioon** | **Sagedus** | **Sõltuvus** |
| Püsiv | Igakuine | Puudub |
| Metaandmed:<br>Kindlustuskaitse nimi. | | |
| **Allikas** | **Transformatsioon** | **Andmetüüp** |
| O&D | Puudub | VARCHAR(40) |

**Lisa 2. Andmehoidla tabelite, päästikprotsesside loomise ja päringute SQL-laused.**

Tabelite ja atribuutide nimedes on täpitähti ühilduvuse huvides meelega välditud.

**Lause 1.** Kindlustusvõtja dimensioonitabeli loomine.

```
CREATE TABLE Kindlustusvotja (
isikukood           VARCHAR(11) NOT NULL,
perekonnanimi       VARCHAR(40) NOT NULL,
eesnimed            VARCHAR(40) ,
synnikp              DATE NOT NULL,
sugu                VARCHAR(1) NOT NULL,
kliendinumber       INTEGER,
aadress             VARCHAR(64),
postiindeks         VARCHAR(10),
maakond             VARCHAR(20),
alguskp             DATE NOT NULL,
lopukp              DATE,

PRIMARY KEY (isikukood, alguskp)
);
```

**Lause 2.** Müüja dimensioonitabeli loomine.

```
CREATE TABLE Myyja (
id                  SMALLINT NOT NULL,
perekonnanimi       VARCHAR(40) NOT NULL,
eesnimed            VARCHAR(40),
myyja_tyyp          SMALLINT NOT NULL,
ylemus              SMALLINT,
kontorinr           SMALLINT,
telefon             VARCHAR(20),
mobiiltelefon       VARCHAR(20),
alguskp             DATE NOT NULL,
lopukp              DATE,

PRIMARY KEY (id,  alguskp)
);
```

**Lause 3.** Kindlustusliigi dimensioonitabeli loomine.

```
CREATE TABLE Kindlustusliik (
kindlustustyyp      INTEGER NOT NULL,
hyvitisliiginimi    VARCHAR(40) NOT NULL,
riskinimi           VARCHAR(40) NOT NULL,
kaitsenimi          VARCHAR(40) NOT NULL,

PRIMARY KEY (kindlustustyyp)
);
```

**Lause 4.** Kindlustatava objekti dimensioonitabel loomine.

```
CREATE TABLE Kindlustatav (
oid                 CHAR(16) FOR BIT DATA NOT NULL,
tyyp                SMALLINT NOT NULL,
alamtyyp            SMALLINT NOT NULL,
ksumma              DECIMAL(9,2),

PRIMARY KEY (oid)
);
```

**Lause 5.** Aja dimensioonitabeli loomine.

```
CREATE TABLE Aeg (
kp                  DATE NOT NULL,
paevanr             SMALLINT NOT NULL,
paevanimi           VARCHAR(11) NOT NULL,
nadalanr            SMALLINT NOT NULL,
kuunr               SMALLINT NOT NULL,
kuunimi             VARCHAR(9) NOT NULL,
kvartal             SMALLINT NOT NULL,
aasta               SMALLINT NOT NULL,

PRIMARY KEY (kp)
);
```

**Lause 6.** Faktitabeli loomine.

```
CREATE TABLE Preemia (
kindlvotja_id       VARCHAR(11) NOT NULL,
myyja_id            SMALLINT NOT NULL,
kindlustus_id       INTEGER NOT NULL,
kindlustatav_id     CHAR(16) FOR BIT DATA NOT NULL,
aeg_id              DATE NOT NULL,
netopreemia         DECIMAL(9,2) NOT NULL,

PRIMARY KEY (kindlvotja_id, myyja_id, kindlustus_id, kindlustatav_id, aeg_id)
);
```

**Lause 7.** Summatabel - kogutud preemiad kindlusvõtja ja aasta järgi.

```
CREATE TABLE SUMMA_KV_AASTA AS
(SELECT KV.isikukood,
     A.aasta,
     SUM(P.netopreemia) AS "Preemiate summa",
     COUNT(*) AS "Number"
FROM Preemia P, Kindlustusvotja KV, Aeg A
WHERE P.kindlvotja_id = KV.isikukood
AND P.aeg_id = A.kp
GROUP BY KV.isikukood, A.aasta)
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION;
```

**Lause 8.** Summatabel - kogutud preemiad kindlustatud objekti tüübi ja aasta järgi.

```
CREATE TABLE SUMMA_KOBJ_AASTA AS
(SELECT KO.tyyp,
      A.aasta,
      SUM(P.netopreemia) AS "Preemiate summa",
      COUNT(*) AS "Number"
FROM Preemia P, Kindlustatav KO, Aeg A
WHERE P.kindlustatav_id = KO.oid
AND P.aeg_id = A.kp
GROUP BY KO.tyyp, A.aasta)
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION;
```

**Lause 9.** Summatabel - kogutud preemiad kindlustuskaitse, aasta ja kuu järgi.

```
CREATE TABLE SUMMA_KLIIK_KUUNIMI AS
(SELECT KL.kaitsenimi,
      A.aasta,
      A.kuunimi,
      SUM(P.netopreemia) AS "Preemiate summa",
      COUNT(*) AS "Number"
FROM Preemia P, Kindlustusliik KL, Aeg A
WHERE P.kindlustus_id = KL.kindlustustyyp
AND P.aeg_id = A.kp
GROUP BY KL.kaitsenimi, A.aasta, A.kuunimi)
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION;
```

**Lause 10.** Summatabel - kogutud preemiad müüja, aasta ja nädala järgi.

```
CREATE TABLE SUMMA_MYYJA_NADAL AS
(SELECT M.perekonnanimi,
      M.eesnimed,
      A.aasta,
      A.nadalanr,
      SUM(P.netopreemia) AS "Preemiate summa",
      COUNT(*) AS "Number"
FROM Preemia P, Myyja M, Aeg A
WHERE P.myyja_id = M.id
AND P.aeg_id = A.kp
GROUP BY M.perekonnanimi, M.eesnimed, A.aasta, A.nadalanr)
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION;
```

**Lause 11**. Summatabel - kogutud preemiad aasta ja kuu järgi.

```
CREATE TABLE SUMMA_AEG_KUU AS
(SELECT A.aasta,
      A.kuunr,
      SUM(P.netopreemia) AS "Preemiate summa",
      COUNT(*) AS "Number"
FROM Preemia P, Aeg A
WHERE P.aeg_id = A.kp
GROUP BY A.aasta, A.kuunr)
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION;
```

**Lause 12**. Osamaksete jaoks vahetabeli loomine.

```
CREATE TABLE SA_installment (
oid                 CHAR(16) FOR BIT DATA NOT NULL,
debactdate          DATE,
clientnumber        INTEGER,
agreementnumber     INTEGER,

PRIMARY KEY(oid)
);
```

**Lause 13**. Isikuandmete jaoks vahetabeli loomine.

```
CREATE TABLE SA_person (
oid                 CHAR(16) FOR BIT DATA NOT NULL,
effectivedate       DATE,
name                VARCHAR(40),
firstnames          VARCHAR(20),
secname             VARCHAR(20),
socialsecuritynum   VARCHAR(11),
birthdate           DATE,
sex                 SMALLINT,
clientnumber        INTEGER,

PRIMARY KEY(oid)
);
```

**Lause 14.** Poliisi andmete jaoks vahetabeli loomine.

```
CREATE TABLE SA_policy (
oid                 CHAR(16) FOR BIT DATA NOT NULL,
agreementnumber     INTEGER,
agreemntsupvisor    SMALLINT,

PRIMARY KEY(oid)
);
```

**Lause 15.** Hüvitisliigi andmete jaoks vahetabeli loomine.

```
CREATE TABLE SA_ecov (
oid                 CHAR(16) FOR BIT DATA NOT NULL,
insobjoid           CHAR(16) FOR BIT DATA,
suminsured          DOUBLE PRECISION,

PRIMARY KEY(oid)
);
```

**Lause 16.** Kindlustuvõtja andmete jaoks vahetabeli loomine.

```
CREATE TABLE SA_polholder (
oid                 CHAR(16) FOR BIT DATA,
ssn                 VARCHAR(11),
name                VARCHAR(40),
firstnames          VARCHAR(40),
birthdate           DATE,
sex                 VARCHAR(1),
clientnumber        INTEGER,
streetaddress       VARCHAR(64),
zipcode             VARCHAR(10),
postalarea          VARCHAR(20),
startdate           DATE,
enddate             DATE

);
```

**Lause 17.** Osamaksete salvestamise päästikprotsessi loomine.

```
CREATE TRIGGER Trigger_inst
NO CASCADE AFTER UPDATE OF (inststatus) ON Installment
REFERENCING NEW AS newrow
FOR EACH ROW MODE DB2SQL
WHEN (newrow.inststatus = 20 OR newrow.inststatus = 50)
BEGIN ATOMIC
      INSERT INTO SA_installment VALUES (
      newrow.oid, newrow.effectivedate,
      newrow.clientnumber, newrow.agreementnumber);
END
```

**Lause 18**. Osamaksete eraldamine vahetabelisse.

```
INSERT INTO SA_installment
SELECT I.oid, I.debactdate, I.clientnumber, I.agreementnumber
FROM Installment I
WHERE (inststatus = 20 OR inststatus = 50)
AND invalnumber IS NULL
;
```

**Lause 19**. Kindlustusvõtja andmete eraldamine vahetabelisse.

```
INSERT INTO SA_person
SELECT P1.oid, P1.effectivedate, P1.name, P1.firstnames, P1.secname,
P1.socialsecuritynum, P1.birthdate, P1.sex, P1.clientnumber
FROM Person P1
WHERE P1.invalnumber IS NULL
AND P1.socialsecuritynum IS NOT NULL
AND P1.effectivedate = (
      SELECT MAX(P2.effectivedate)
      FROM Person P2
      WHERE P2.oid = P1.oid
      AND P2.invalnumber IS NULL)
AND P1.oid IN (
            SELECT L.parent_oid
            FROM Links L
            WHERE L.link_type = 18)
;
```

**Lause 20**. Poliisi andmete eraldamine vahetabelisse.

```
INSERT INTO SA_policy
SELECT P1.oid, P1.agreementnumber, P1.agreemntsupvisor
FROM Policy P1
WHERE P1.invalnumber IS NULL
AND P1.effectivedate = (
      SELECT MAX(P2.effectivedate)
      FROM Policy P2
      WHERE P2.oid = P1.oid
      AND P2.invalnumber IS NULL)
;
```

**Lause 21**. Hüvitisliigi andmete eraldamine vahetabelisse.

```
INSERT INTO SA_ecov
SELECT E1.oid, E1.insobjoid, ROUND(E1.suminsured, 2)
FROM Elementarycoverage E1
WHERE E1.invalnumber IS NULL
AND E1.status = 60
AND E1.effectivedate = (
      SELECT MAX(E2.effectivedate)
      FROM Elementarycoverage E2
      WHERE E2.oid = E1.oid
      AND E2.invalnumber IS NULL
      AND E2.status = 60)
;
```

**Lause 22**. Kindlustusvõtja andmete teise vahetabelisse lisamine.

```
INSERT INTO SA_polholder
SELECT P.oid, P.socialsecuritynum, P.name,
CASE
  WHEN P.secname IS NULL THEN P.firstnames
  ELSE CONCAT(CONCAT(P.firstnames, ' '), P.secname)
END,
P.birthdate,
CASE
  WHEN P.sex = 10 THEN 'M'
  WHEN P.sex = 20 THEN 'N'
  ELSE 'L'
END, P.clientnumber, A.streetaddress, RTRIM(VARCHAR(A.zipcode)),
A.postalarea, A.effectivedate, A.invalnumber
FROM SA_person P, Address A, Links L
WHERE A.invalnumber IS NULL
AND A.type = 10
AND L.invalnumber IS NULL
AND L.parent_oid = P.oid
AND L.child_oid = A.oid
AND L.link_type = 10
;
```

**Lause 23.** Kindlustusvõtja andmete laadimine andmehoidlasse.

```
INSERT INTO Kindlustusvotja
SELECT PH.ssn, PH.name, PH.firstnames, PH.birthdate, PH.sex, PH.clientnumber,
PH.streetaddress, PH.zipcode, PH.postalarea,
PH.startdate, PH.enddate
FROM SA_polholder PH
;
```

**Lause 24**. Kindlustatava objekti andmete laadimine andmehoidlasse.

```
INSERT INTO Kindlustatav
SELECT P.oid, P.type, P.subtype, SUM(ROUND(SA_ecov.suminsured, 2))
FROM Proxy P, SA_ecov
WHERE P.oid = SA_ecov.insobjoid
AND P.invalnumber IS NULL
GROUP BY P.oid, P.type, P.subtype
;
```

**Lause 25**. Müüja andmete laadimine andmehoidlasse.

```
INSERT INTO Myyja
SELECT U.user_nr, RTRIM(VARCHAR(U.surname)), RTRIM(VARCHAR
(U.first_names)),
U.salesperson_type, U.boss, U.office_nr, RTRIM(U.workphone), RTRIM(U.cellphone),
U.begin_date, U.end_date
FROM User_data U
WHERE U.user_nr IS NOT NULL
;
```

**Lause 26**. Kindlustusliigi andmete laadimine andmehoidlasse.

```
INSERT INTO Kindlustusliik
SELECT DISTINCT INTEGER(CONCAT(CONCAT(VARCHAR(E.coveragetype), VARCHAR
(E.risktype)), VARCHAR(E.type))),
RTRIM(P1.name), RTRIM(P2.name), RTRIM(P3.name)
FROM Elementarycoverage E, Proxy P1, Proxy P2, Proxy P3
WHERE E.effectivedate = '2000-01-01'
AND E.coveragetype = P1.subtype
AND E.risktype = P2.subtype
AND E.type = P3.subtype
AND E.invalnumber IS NULL
AND P1.type = 138 AND P1.startdate = '2000-01-01'
AND P2.type = 112 AND P1.startdate = '2000-01-01'
AND P3.type = 118 AND P1.startdate = '2000-01-01'
AND E.activitynumber = P1.activitynumber
AND E.activitynumber = P2.activitynumber
AND E.activitynumber = P3.activitynumber
AND P1.name NOT LIKE 'Irtaimistoturva/%'
AND P1.name NOT LIKE '%suppea%'
AND P1.name NOT LIKE 'Suppea%'
AND P1.name NOT LIKE 'Seuraeläinturva%'
AND P1.name NOT LIKE 'Matkustajaturva%'
AND P1.name NOT LIKE 'Lapsen henkilöturva%'
AND P1.name NOT LIKE 'Vapaa-ajan henkilöturva%'
;
```

**Lause 27**. Preemia andmete laadimine andmehoidla faktitabelisse.

```
INSERT INTO Preemia
SELECT PH.socialsecuritynum, P.agreemntsupvisor,
INTEGER(CONCAT(CONCAT(VARCHAR(EI.coveragetype), VARCHAR(EI.risktype)),
VARCHAR(EI.ecotype))),
EC.insobjoid, I.debactdate, ROUND(EI.netpremium, 2)
FROM SA_installment I, SA_person PH, SA_policy P,
Eleminstallment EI, SA_ecov EC
WHERE I.agreementnumber = P.agreementnumber
AND I.clientnumber = PH.clientnumber
AND I.oid = EI.installmentoid
AND EI.ecoverageoid = EC.oid
;
```

**Lause 28**. Aja andmete laadimine andmehoidlasse.

```
INSERT INTO Aeg
SELECT DISTINCT aeg_id,
                DAYOFWEEK_ISO(aeg_id),
                DAYNAME(aeg_id),
                WEEK_ISO(aeg_id),
                MONTH(aeg_id),
                MONTHNAME(aeg_id),
                QUARTER(aeg_id),
                YEAR(aeg_id)
FROM Preemia
;
```

**Lisa 3. Transformatsioonide kirjeldused.**

Punktmodelleerimse töölehtedel esinenud transformatsioonitüüpide seletused:

**Tekst1** – Tekstiline muutus, mis tähistab transformatsioone, kus sihtveeru andmetüüp andmehoidlas on VARCHAR, lähtesüsteemi atribuut on samuti VARCHAR tüüpi ning transformatsiooni käigus muutub ainult atribuuti hoidva tabeli veeru pikkus.

**Tekst2** – Tekstiline muutus, mis tähistab transformatsioone, kus lähtesüsteemi atribuut andmetüübiga CHAR muutub VARCHAR-ks ning seejures võib muutuda ka veeru pikkus.

**Tekst3** – Lähtesüsteemis tähistatab inimese sugu numbriline atribuut, kus mees = 10 ja naine = 20. Andmehoidlas võib vastav atribuut omada vaid tekstilist väärtust, pikkusega üks tähemärk. Konverteerimisel muutub väärtus 10 täheks "M" ja 20 täheks "N".

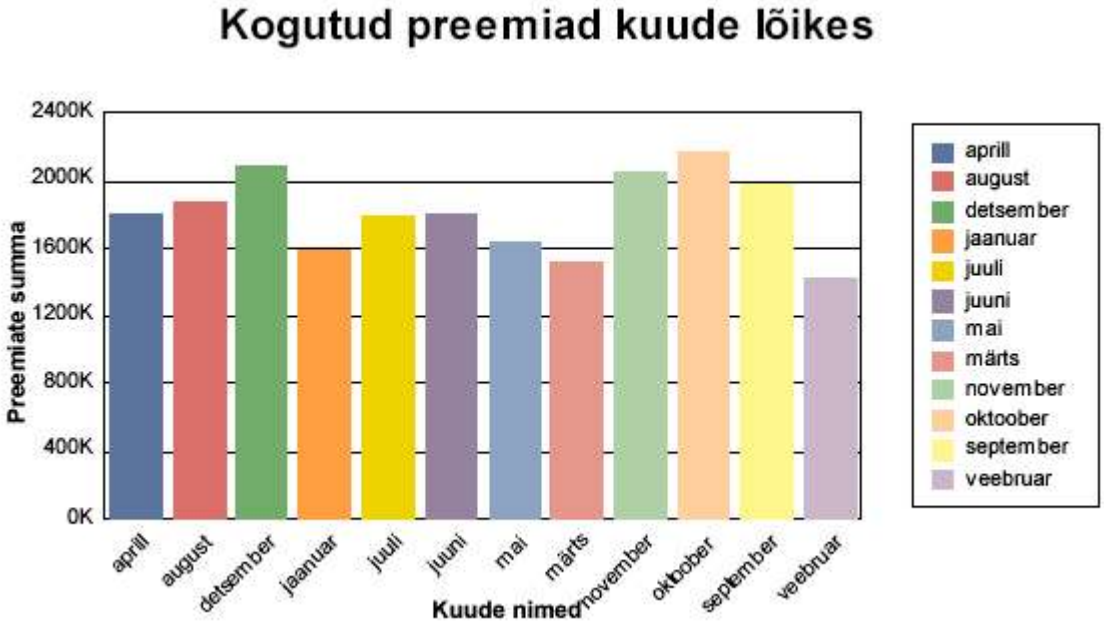
**Number1 –** Numbriline muutus, kus lähtesüsteemi atribuut andmetüübiga DOUBLE ümardatakse kahe komakohani ja konverteeritakse andmetüübiks DECIMAL(9,2).

**Ühinemine1 –** Kindlustusvõtja atribuudi „eesnimed" andmetüübiga VARCHAR(40) saamiseks ühendatakse omavahel kokku lähtesüsteemi atribuudid „eesnimi" andmetüübiga VARCHAR(20) ja „teised eesnimed" andmetüüpidega VARCHAR(20).

**Ühinemine2** – Kindlustusliigi tabeli primaarvõtme ehk atribuudi „kindlustustyyp" saamiseks ühendatakse omavahel kolm lähtesüsteemi atribuuti, milleks on „hüvitisliigi tüüp", „riski tüüp" ja "kindlustuskaitse tüüp". Kõigi kolme atribuudi numbrilised väärtused konverteeritakse tekstiliseks, ühendatakse omavahel ning konverteeritakse tagasi numbriks. Näiteks lähteandmetest 10, 70, 130 saab üks number väärtusega 1070130.

**Lisa 4. Raportite näited.**

**Raport 1**. Kogutud preemiate kuude järgi. Kuud on aastate kaupa summeeritud.




**Raport 2**. Klientide arv müüjate järgi (lõik tabelist).

## Klientide arv müüjate järgi

| | Klientide arv |
|---|---|
| Kokku | 30 342 |
| 100 | 95 |
| 120 | 312 |
| 125 | 212 |
| 128 | 165 |
| 130 | 209 |
| 131 | 335 |
| 132 | 28 |
| 133 | 812 |

**Raport 3**. Kogutud preemiad agentide ehk müüjate järgi.

**Kogutud preemiad müüjate järgi**

| | Summa |
|---|---|
| 100 | 58 768,10 |
| 120 | 685 406,01 |
| 125 | 981 471,16 |
| 128 | 510 052,44 |
| 130 | 277 660,02 |
| 131 | 820 526,61 |
| 132 | 28 837,72 |
| 133 | 1 746 475,08 |
| 137 | 162 840,22 |
| 138 | 1 014 725,88 |
| 140 | 2 840 747,00 |
| 141 | 180 307,36 |
| 142 | 626 442,45 |
| 143 | 718 973,68 |
| 145 | 18 858,06 |
| 146 | 115 462,44 |
| 147 | 88 418,18 |
| 148 | 24 075,90 |
| 150 | 27 792,96 |
| 156 | 4 512 651,30 |
| 157 | 446 499,60 |
| 159 | 11 319 759,25 |

**Raport 4**. Kõige tasuvamad kliendid ehk 10 kõige enam preemiat maksnud kindlustusvõtjat.

| Kliendinumber | Preemiate summa |
|---|---|
| 277740 | 28964.79 |
| 9198981 | 17303.68 |
| 9013858 | 17023.04 |
| 9003171 | 15220.36 |
| 86970010 | 15210.44 |
| 311922 | 14954.52 |
| 461131 | 13444.08 |
| 11662415 | 13368.24 |
| 317617 | 13039.48 |
| 9133604 | 12534.52 |