7. Andmemudelite normaliseerimine

andmete modelleerimise üldistamine

Alates sellest ajast, kui on tegeletud teadliku andmete modelleerimisega, on otsitud reegleid, millest juhindudes saaks teha võimalikult häid ja hästi töötavaid andmemudeleid. Peamiseks eesmärgiks on leida reeglite kogumeid, selleks et vältida suuri vigu. Vanimaks ja rohkem arendatud selliseks reeglistikuks on andmemudelite normaliseerimise reeglid.

normaalkuju *(normal form)*

Normaliseerimine põhineb kindlate, kuid samas suhteliselt üldise iseloomuga, piirangute kogumitel, mida rakendatakse andmemudelitele. Neid piirangute kogumeid nimetatakse normaalkujudeks *(normal form)*. Normaalkujud on andmemudeli olemite (andmebaasi tabelite) ja olemite atribuutide (tabeli veergude) organiseerimisreeglid, mis võimaldavad hallata andmemudeli transaktsioonilisi ja struktuurilisi eripärasid, mis kanalideeruvad andmete talletamise ja käsitlemise anomaaliatena.

anomaaliad:
info liiasus
mälu raiskamine
"sunnitud" lisa-tegevuste tegemine
seoste riknemine
andmete muutumine ebaadekvaatseks

Andmete talletamise anomaaliad on info liiasus ja mälu raiskamine. Andmete käsitlemise anomaaliaid iseloomustab see, kui lisaks konkreetsetele andmete haldamise tegevustele, on vaja teha veel selle tegevusega otseselt mitte seotud lisategevusi ilma milleta andmete struktuur rikneb. Andmete riknemise viise on kaks - riknevad andmete vahelised seosed või andmed muutuvad mitmeti mõistetavaks.

Normaalkujud võimaldavad hallata andmeskeemide terviklikkust (*integrity*).

7 normaalkuju ja normaliseerimata kuju

Peamisi kasutatavaid normaalkujusid on seitse. Nendele lisandub veel normaliseerimata kuju, mis tegelikkuses on samuti normaalkuju (nii nagu ka "värvitu" on värv).

Normaalkujusid kirjeldatakse järjest alates normaliseerimata kujust kuni domeenvõtme normaalkujuni kasutades iga järgneva normaalkuju kirjeldamiseks tema erinevusi eelmisest normaalkujust.

7.1. Andmemudeli normaliseerimata kuju

normaliseerimata andmemudelis on osades või kõigis tabelites korduv-grupid

Andmemudel on normaliseerimata, kui temas leidub olemeid (tabeleid), milles on korduv-gruppe:

NÄIDE

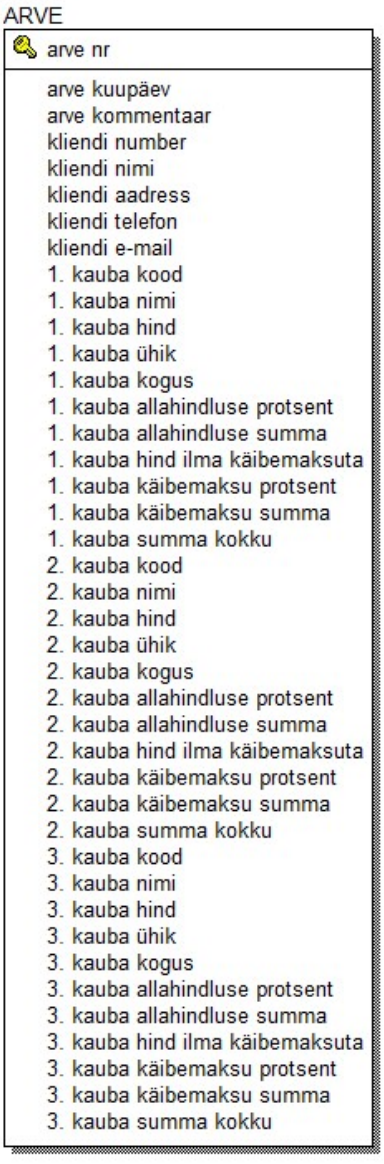
Siin on olem (tabel), milles on kirjeldatud arve esitamiseks vajalik andmestruktuur. Olemi esimeses osas on arve päise andmed. Need on "ühe kordsed" andmed - iga arve kohta on ainult üks komplekt andmeid. Arve

ridadega on teine lugu. Siin tabelis on iga arve kohta võimalik luua kolm arve rida. Seega saab kirjeldatud tabelis talletada kas ühe-, kahe - või kolme realisi arveid. Iga arve rea jaoks on üks grupp veerge. Kuna kõik grupid on samasuguseid nimetataksegi neid korduv-gruppideks.

Neid gruppe võib olla nii vähem kui ka rohkem - minimaalne korduv-gruppide arv on 2. Korduv-gruppide arvu ülemine piir on piiratud kas mudeli loogikaga (näiteks: üle 10-ne realisi arveid on harva) või konkreetse andmebaasi süsteemi poolt seatud füüsiliste piiridega. Näiteks võib olla konkreetsel andmebaasisüsteemil piiranguks see, et ühes tabelis ei tohi olla rohkem kui 256 veergu. Antud näite korral ei saaks selles olemis olla rohkem kui 22 korduv-gruppi so. int((256-8)/11).

Andmemudeli normaliseerimata kuju põhjustab järgmisi anomaaliaid:

andmemudeli normaliseerimata kuju põhjustatud anomaaliad



andmete lisamise anomaalia

Andmete lisamise anomaalia - kui on vaja lisada rohkem korduvaid objekte, kui seda võimaldab korduv-gruppide arv olemis (tabelis), siis tuleb lisada uus kirje, kus jätkatakse eelmist kirjet. Seejuures tuleb ka kirje mitte korduv osa nüüd sisestada korduvana – mitmesse järjestikusesse kirjesse. See kõik nõuab lisa programmikoodi kirjutamist, mis oskab siduda samasuguste päistega kirjed samaks loogiliseks dokumendiks ja samas jaotada ühe loogilise dokumendi mitmete füüsiliste kirjete vahel.

andmete kustutamise anomaalia

Andmete kustutamise anomaalia - väga vaevaline on korduv-gruppides olevate andmete kustutamine baasist. Kui kustutatakse andmed korduv-grupist (korduv-grupi väljad tehakse tühjaks), mis ei ole dokumendis viimane, tuleb tagumisi korduv-gruppe "üles poole nihutada", et tekkida tekkinud tühimik. Kui dokument on jaotatud mitme füüsilise kirje vahel ja kustutatakse objekti andmed esimese kirje korduv-grupist hakkavad andmed nihkuma

tagumiste kirjete korduvgruppidest ees pool olevate kirjete korduv-gruppidesse. Muidugi võib sellest väärtuste nihutamisest loobuda ja tekkinud tühimikud korduv-gruppidesse sisse jätta. See aga teeb kirjete töötluse oluliselt keerulisemaks ja korduv-gruppidees olevate objektide järjestuse säilitamiseks tuleb ühel hetkel nii või teisiti hakata objektide väärtusi korduvgruppide jadas "tihendama". Seda sellel hetkel, kui korduv-gruppidesse andmete lisamisel selgub, et viimane korduvgrupp on väärtustega täidetud aga vahepeal on korduv-gruppe, mis on tühjad. Probleem on muidugi väiksem sellisel juhul, kui meil pole vaja säilitada objektide järjestust - siis võib uute objektide väärtustega täita ka vahepealseid korduvgruppe

info liiasuse anomaalia

Info liiasuse anomaalia - kuna iga kord kui objekti andmed kirjutatakse korduvgrupi väljadesse kirjutatakse sinna kõik seda objekti iseloomustavad andmed (mujale ju neid kirjutada pole), siis tekivad baasi sama objekti kohta andmed paljudes kirjetes, mis on aga ilmne info liiasus. Piisaks kui objekti põhilised andmed kirjutataks ühte kirjesse. Normaliseerimata mudel seda aga ei võimalda.

andmete uuendamise anomaalia

Andmete uuendamise anomaalia - kuna korduv-gruppides olevate objektide andmeid korratakse väga paljused korduv-gruppides, siis ühe konkreetses kirjes oleva korduvgrupi andmete uuendamisel võib tekkida vajadus uuendada andmeid väga paljudes kirjetes - kõikides kirjete, kus on sama objekti andmed. Sisuliselt on tegemist andmebaasiserveri protsessori aja raiskamisega (ühe kirje uuendamise asemel tuleb teha palju uuendusi), mis tänapäeva infosüsteemides on üks kõige kriitilisemaid ressursse.

andmete adekvaatsuse anomaalia

Andmete ebaadekvaatseks muutumise anomaalia - kuna normaliseerimata mudel tingib info liiasuse ja samu andmeid tuleb uuendada erinevates kirjetes, siis on lihtne tekkima situatsioon, kus sama objekti kohta on andmebaasi erinevates kirjetes erinevad andmed. See olukord tekkib tavaliselt programmi vigade tõttu või siis "käsitsi" andmete andmebaasis parandamise tõttu, kus mingis kirjes andmete uuendamise vajadus võib kahe silma vahele jääda. Nende vigade tekkimise hetkel on neid peaaegu võimatu avastada. Hiljem on aga väga raske määrata, millises kirjes on objekti kohta õiged andmed.

mälu raiskamise anomaalia

Mälu raiskamise anomaalia - nendes kirjetes, kus osade korduv-gruppide väljad jäävad tühjaks, kuna kirje võimaldab salvestada rohkem korduv-gruppe, kui seda vajas konkreetne baasi talletatud dokument, reserveeritakse andmebaasis ilma asjata tühja mäluruumi - kettale salvestatakse ju ka tühjad korduv-grupid. See anomaalia oli eriti oluline minevikus kui kasutatav ketta ruum oli piiratud.

Tänapäeval, kui kõvaketaste mahud on drastiliselt suurenenud, sellele enam eriti tähelepanu ei pöörata.

normaliseerimata andmemudel põhjustab kõiki võimalikke anomaaliaid

Kokkuvõtlikult võib öelda, et normaliseerimata andmemudel põhjustab kõiki võimalikke anomaaliaid.

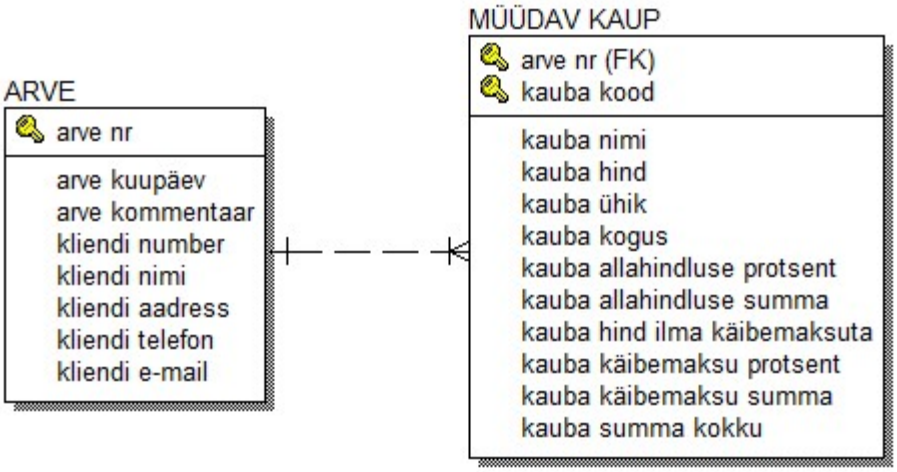
7.2. Andmemudeli esimene normaalkuju *(1NF - first normal form)*

esimene normaalkuju - 1NF

Esimene noormaalkuju erineb normaliseerimata kujust peamiselt selle poolest et tabelite sees puuduvad korduv-grupid. Samas peab seal leiduma olemeid, kus on jaotatud võtmeid *(distributed keys)* ja transitiivseid seoseid *(transitive dependencies)*. Kuna neid mõlemaid ei õnnestu tekitada ühte olemisse, siis peab olema esimese normaalkuju reeglitele vastavas andmemudelis vähemalt kaks olemit.

normaliseerimata kujust saame esimese normaalkuju korduv-gruppide kõrvaldamisel

Esimese normaalkuju saame me siis, kui kõrvaldame andmemudeli normaliseerimata kujust korduv-grupid. Selleks eemaldame korduvgrupiga tabelist kõik korduvgrupid, võtame ühe korduvgrupi, eemaldame sealt korduvgrupi loenduri ja teeme sellest veergude kogumist uue tabeli. Pärast seda seome esialgse tabeli ja uue tabeli selliselt, et üks-mitmene suhe suundub enne olemas olnud tabelist uude tabelisse. Seejuures tuleb silmas pidada seda, et kummaski üks-mitmese seose otsas ei saa olla "mulli", mis kirjeldab seose puudumise võimalust. Seda selle pärast, et korduv- grupiga tabelis on alati täidetud vähemalt ühe korduv-grupi väljad:



liit-võti

Uue olemi primaarvõti saadakse seotud olemi (suhte "üks" poolel oleva olemi) primaarvõtme ja uues olemis kirjeldatava objekti võtme liitmise teel. Oluline on, et mõlemas tabelis oleks unikaalne primaarvõti. liitvõtit vajame selle pärast, et kumbki võtme komponent eraldi võetuna ei tagaks primaarvõtme jaoks vajalikku unikaalsust.

jaotatud võti

Uues tekkinud tabelis on jaotatud võti. Kui jälgida selle tabeli välju, siis näeme, et osa nendest väljadest on seotud tugevamini ühe võtme komponendiga ja osa võtme teise komponendiga. Nii on "kauba nimi", "kauba hind" ja "kauba ühik" seotud rohkem "kauba koodiga" (kauba andmed) ja ülejäänud väljad "arve numbriga" (müügi andmed). Seega jagavad võtme erinevad komponendid kirje tinglikult kaheks.

Jaotatud võtmed tekivadki tavaliselt normaliseerimata mudeli esimesele normaalkujule ümber struktureerimise tulemusena.

transitiivne seos

Transitiivne seos asub selles tabelis, mille ümber struktureerimise tulemusena me andmemudeli normaliseerimata kujust saime esimese normaalkuju. Kui vaadata toodud näites seda tabelit, siis näeme, et seal on ühest väljast koosnev primaarvõti, samas on aga andmeväljadega kirjeldatud kaks objekti (arve ja klient), millest ainult üks (arve) on tugevalt seotud primaarvõtmega. Teise objekti andmed pole aga võtmega üldse seotud.

jaotatud võti

Nii jaotatud võti kui ka transitiivne seos tingivad mõlemad sama asja - objektide andmed saavad tekkida ainult paaridena (arve ja klient, kaup ja müük). See on aga andmemudeli toimimise seisukohalt mitte kõige parema asi. Meil puudub võimalus selliste objektide andmete ette salvestamine, mida tegelikult oleks võimalik teha. Nii saaksime me luua klientide baasi, kui klientide andmed oleks kirjeldatud eraldi olemina ja kaupade loendi, kui kaubad oleksid mudelis kirjeldatud eraldi olemina. Praegu tekivad nii kaubad kui isikud baasi alles siis, kui toimunud müük - kliendi andmed siis, kui talle esimest korda midagi müüdi ja kauba andmed siis, kui seda kaupa esimest korda müüdi.

anomaaliate muutumine võrreldes mudeli normaliseerimata kujuga

Mis on saanud nüüd anomaaliatest? Mitte midagi erilist, sest kadus ära ainult mälu raiskamise anomaalia. Kõik ülejäänud anomaaliad jäid alles. Mõned neist ainult teisenesid veidi:

andmete lisamise anomaalia

Andmete lisamise anomaalia - transitiivse seose võtmega mitte seotud objekti andmeid ei ole võimalik baasi salvestada enne kui on tekkinud vajadus võtmega seotud objekti andmete sisestamiseks. Kuna üldjuhul on nende võtmega

seostamata objektide näol selliste objektidega, mille andmed saaks eelnevalt baasi sisestada ja seda takistab ainult mudeli puudulikkus, siis on tegemist olulise anomaaliaga. Sama pädeb jaotatud võtmega olemi kohta. Seal on objektiks, mille andmed iseseisvalt baasi tekkida ei saa, vaid ainult koos põhiobjektiga, selle objekti andmed, mis on seotud võtmega, mis ei ole välisvõti (FK). Meie näites on selleks kauba andmed. Kuna üldjuhul on nende välisvõtmega seostamata objektide andmete näol selliste objektidega, mille andmed saaks eelnevalt baasi sisestada ja seda takistab ainult mudeli puudulikkus, siis on tegemist olulise anomaaliaga.

andmete kustutamise anomaalia

Andmete kustutamise anomaalia seisneb nüüd selles, et transitiivse seose selle objekti andmete kustutamisel, mis on seotud primaarvõtmega kaovad baasist ka sõltuva objekti andmed. Kui see oli ainus koopia baasis selle objekti andmetest, siis kaovad selle andmed baasist ehkki nad vahepeal olid seal juba. Sama pädeb ka jaotatud võtmega tabeli kohta. Kui baasist kustutada selle objekti andmed, mis on seotud välisvõtmega kaovad baasist ka teise objekti andmed. Kui see oli selle objekti andmete viimane eksemplar baasis, siis kaovad need andmed sealt baasist täielikult vaatamata sellele, et nad vahepeal olid baasis juba olemas.

kirje uuendamise anomaalia

Andmete uuendamise anomaalia - kuna transitiivses seoses olevate võtmega mitte seotud objektide andmeid ja jaotatud võtme selle võtmes osaga, mis ei ole välisvõti, seotud objektide andmeid korratakse väga paljused kirjetes, siis ühe konkreetses kirjes olevate andmete uuendamisel võib tekkida vajadus uuendada andmeid väga paljudes kirjetes - kõikides kirjete, kus on sama objekti andmed. Sisuliselt on tegemist andmebaasiserveri protsessori aja raiskamisega (ühe kirje uuendamise asemel tuleb teha palju uuendusi erinevates kirjetes), mis tänapäeva infosüsteemides on üks kõige kriitilisemaid ressursse.

info liiasuse anomaalia

Info liiasuse anomaalia - iga kord kui transitiivse seosega olemi või jaotatud võtmetega olemi sõltuva objekti andmed kirjutatakse andmebaasi kirjutatakse sinna kõik seda objekti iseloomustavad andmed (mujale ju neid kirjutada pole), siis tekivad baasi sama objekti kohta andmed paljudes kirjetes, mis on aga ilmne info liiasus. Piisaks kui sõltuvate objektide põhilised andmed kirjutataks ühte kirjesse. Mudeli esimene normaalkuju seda ei võimalda. Sama pädeb ka jaotatud võtmega olemite sõltuvate objektide andmete salvestamisel.

andmete adekvaatsuse anomaalia

Andmete ebaadekvaatseks muutumise anomaalia - kuna transitiivsed seosed ja jaotatud võtmed tingivad info liiasuse ja samu andmeid tuleb uuendada erinevates kirjetes, siis on lihtne tekkima situatsioon, kus sama objekti kohta on andmebaasi erinevates kirjetes erinevad andmed. See olukord tekkib tavaliselt

programmi vigade tõttu või siis "käsitsi" andmete andmebaasis parandamise tõttu, kus mingis kirjes andmete uuendamise vajadus võib kahe silma vahele jääda. Nende vigade tekkimise hetkel on neid peaaegu võimatu avastada. Hiljem on aga väga raske määrata, millises kirjes on objekti kohta õiged andmed.

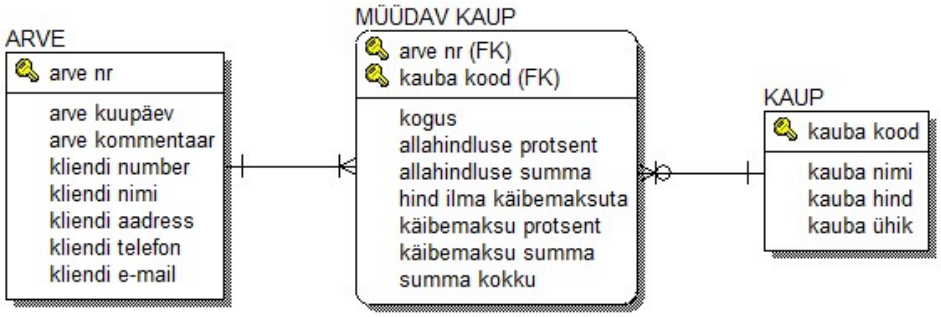
7.3. Andmemudeli teine normaalkuju *(2NF - second normal form)*

teine normaalkuju - 2NF

Teine noormaalkuju erineb esimesest normaalkujust selle poolest et sealt on kaotatud jaotatud võtmetega olemid. Samas peab seal leiduma olemeid, kus on transitiivseid seoseid *(transitive dependencies).*

esimesest normaalkujust saame teise normaalkuju jaotatud võtmete kõrvaldamisel

Teise normaalkuju saame me siis, kui kõrvaldame andmemudeli esimesest kujust jaotatud võtmetega olemid. Selleks eemaldame jaotatud võtmega olemist need veerud, mis ei ole seotud välisvõtmega (FK) ja teeme sellest veergude kogumist uue olemi. Pärast seda seome esialgse olemi ja uue olemi selliselt, et üks-mitmene suhe suundub enne uuest olemist sellesse olemisse mida me ümber struktureerisime. Seejuures tuleb silmas pidada seda, seose mitmeses otsas on "mull", mis kirjeldab seda, et "kõrvale tõstetud" (uude olemisse) objekt võib eksisteerida ka iseseisvalt. See oligi ju meie eesmärk . muuta senine sõltuv objekt iseseisvaks.



liit-võti "uuel kujul"

Uue olemi primaarvõti saadakse seal kirjeldatud objektide unikaalsest võtmest. Restruktureeritud olemi primaarvõtme struktuur jääb samaks, kuid muutub selle saamise viis. Nüüd koosneb ta kahest välisvõtmest (FK) millest üks saadakse sealt kus varemgi, teine aga uuest loodud tabelist. Nendest kokku moodustub unikaalne primaarvõti, mida kumbki välisvõti iseseisvalt pole.

Uues tabelis on väljad ainult ühe objekti andmete talletamiseks ja seega selles uusi anomaaliaid tekitavaid struktuure pole. Sama juhtus ka restruktureeritud tabelis. Kuigi primaarvõti seal koosneb endiselt kahest komponendist on kõik andmeväljad selles tabelis seotud võtme kui tervikuga. Selles olemis kirjelduva

objekti semantika on "müüdud kaup". Ka see olem ei genereeri rohkem anomaaliaid.

anomaaliate muutumine võrreldes mudeli esimese normaalkujuga

Mis on saanud nüüd anomaaliatest? Kadunud on kõik jaotatud võtmest tingitud anomaaliad. Transitiivne seos ja temast tingitud anomaaliad on jäänud samaks.

andmete lisamise anomaalia

Andmete lisamise anomaalia - transitiivse seose võtmega mitte seotud objekti andmeid ei ole võimalik baasi salvestada enne kui on tekkinud vajadus võtmega seotud objekti andmete sisestamiseks. Kuna üldjuhul on nende võtmega seostamata objektide näol selliste objektidega, mille andmed saaks eelnevalt baasi sisestada ja seda takistab ainult mudeli puudulikkus, siis on tegemist olulise anomaaliaga.

andmete kustutamise anomaalia

Andmete kustutamise anomaalia seisneb nüüd selles, et transitiivse seose selle objekti andmete kustutamisel, mis on seotud primaarvõtmega kaovad baasist ka seotud objekti andmed. Kui see oli ainus koopia baasis selle objekti andmetest, siis kaovad selle andmed baasist ehkki nad vahepeal olid seal juba.

kirje uuendamise anomaalia

Andmete uuendamise anomaalia - kuna transitiivses seoses olevate võtmega mitte seotud objektide andmeid korratakse väga paljused kirjetes, siis ühe konkreetses kirjes olevate andmete uuendamisel võib tekkida vajadus uuendada andmeid väga paljudes kirjetes - kõikides kirjete, kus on sama objekti andmed. Sisuliselt on tegemist andmebaasiserveri protsessori aja raiskamisega (ühe kirje uuendamise asemel tuleb teha palju uuendusi), mis tänapäeva infosüsteemides on üks kõige kriitilisemaid ressursse.

info liiasuse anomaalia

Info liiasuse anomaalia - iga kord kui transitiivse seosega olemi sõltuva objekti andmed kirjutatakse andmebaasi kirjutatakse sinna kõik seda objekti iseloomustavad andmed (mujale ju neid kirjutada pole), siis tekivad baasi sama objekti kohta andmed paljudes kirjetes, mis on aga ilmne info liiasus. Piisaks kui sõltuvate objektide põhilised andmed kirjutataks ühte kirjesse. Mudeli teine normaalkuju seda ei võimalda.

andmete adekvaatsuse anomaalia

Andmete ebaadekvaatseks muutumise anomaalia - kuna transitiivsed seosed tingivad info liiasuse ja samu andmeid tuleb uuendada erinevates kirjetes, siis on lihtne tekkima situatsioon, kus sama objekti kohta on andmebaasi erinevates kirjetes erinevad andmed. See olukord tekkib tavaliselt programmi vigade tõttu või siis "käsitsi" andmete andmebaasis parandamise tõttu, kus mingis kirjes andmete uuendamise vajadus võib kahe silma vahele jääda. Nende vigade tekkimise hetkel on neid peaaegu võimatu avastada.

Hiljem on aga väga raske määrata, millises kirjes on objekti kohta õiged andmed.

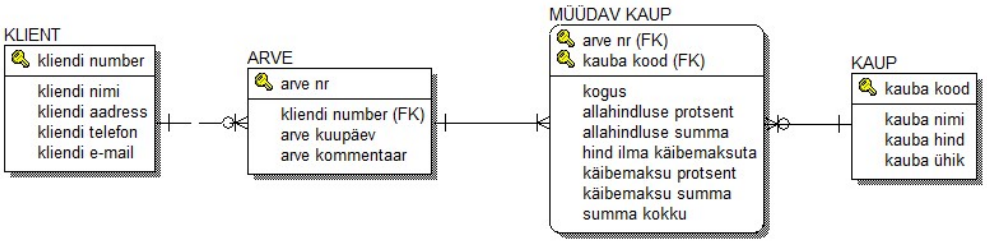
7.4. Andmemudeli kolmas normaalkuju *(3NF - third normal form)*

kolmas normaalkuju - 3NF

Kolmas noormaalkuju erineb teisest normaalkujust selle poolest, et sealt on kaotatud transitiivsete seostega olemid.

teisest normaalkujust saame kolmanda normaalkuju transitiivsete seoste kõrvaldamisel

Kolmanda normaalkuju saame me siis, kui kõrvaldame andmemudeli teisest kujust transitiivse seosega olemid. Selleks eemaldame transitiivse seosega olemist need veerud, mis ei ole seotud primaarvõtmega ja teeme sellest veergude kogumist uue olemi. Pärast seda seome esialgse olemi ja uue olemi selliselt, et üks-mitmene suhe suundub uuest olemist sellesse olemisse, mida me ümber struktureerisime. Seejuures tuleb silmas pidada seda, seose mitmeses otsas on "mull", mis kirjeldab seda, et "kõrvale tõstetud" (uude olemisse) objekt võib eksisteerida ka iseseisvalt. See oligi ju meie eesmärk . muuta senine sõltuv objekt iseseisvaks.



Uue olemi primaarvõti saadakse seal kirjeldatud objektide unikaalsest võtmest. Restruktureeritud olemi primaarvõtme struktuur jääb samaks. Seose tekitamisel ei pärita uue olemi primaarvõtit restruktureeritud olemi primaarvõtmesse vaid see jääb selliseks nagu see oli seni. Välisvõti (FK) päritakse lihtsalt tabeli koosseisu.

Uues tabelis on väljad ainult ühe objekti andmete talletamiseks ja seega selles uusi anomaaliaid tekitavaid struktuure pole. Sama juhtus ka restruktureeritud tabelis. Nüüd on mudel selline, kus igas olemis kirjeldub ainult ühte tüüpi objektid.

senised anomaaliad on kadunud

Mis on saanud nüüd anomaaliatest? Kõik seni kirjeldatud anomaaliad on kadunud.

primaarvõtme uuendamise anomaalia

Kõigist seni vaadeldud anomaaliatest saime lahti, kuid jäänud on üks, mis eksisteeris kogu aeg, kuid seda polnud mõtet kirjeldada, kuna ülejäänud anomaaliad olid oluliselt "saatuslikumad". Normaliseerimata kujul seda ei eksisteerinud, kuid nii esimesel, kui ka teisel normaalkujul oli see täiesti olemas. Tegemist on primaarvõte uuendamise anomaaliaga. Kui uuendada mõnda primaarvõtit sellises olemis, kus ei ole päritud primaarvõti (meie näites on päritud primaarvõtmega olemiks olem MÜÜDAV KAUP), siis iga primaarvõtme uuendamine tingib hulgaliselt uuendusi kõikides nendes kirjetes, kus antud primaarvõtme väärtust on kasutatud seostava paari tekitamiseks välisvõtmega. Kui seda mitte teha, siis riknevad kõik uuendatava primaarvõtme väärtusega kirje seosed. Väärtused tuleb uuendada ka seotud kirjete välisvõtmetel.

primaarvõtme uuendamise ohud

Siit genereerub kaks ohtu. Esiteks andmebaasi serveri vähese protsessori ressursi raiskamine - tuleb ju ühe uuenduse asemel teha mitu. Teiseks andmete adekvaatsuse kao oht, kuna mõned vajalikud uuendused võivad jääda kahe silma vahele. Selle tulemusena tekkivad seoseta kirjed ja hiljem on väga raske otsustada, millise kirjega nad varem seotud olid.

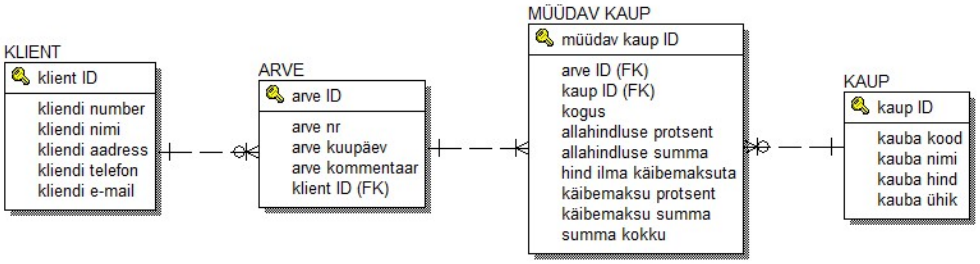
reaalne elu ja primaarvõtme uuendamine

Selle vältimiseks võib küll kehtestada reegli, et primaarvõtme väärtusi ei muudeta kunagi, aga sellest võib olla väga raske kinni pidada, kuna tegemist on nö. "eluliste andmetega" ja kui päris elus need andmed muutuvad, siis ei jää meil midagi üle - me peame neid muutma ka andmebaasis.

7.5. Domeen-võtme normaalkuju *(DKNF - Domen Key Normal Form)*

domeen-võtme normaalkuju - DKNF

Kolmandalt normaalkujult üle minnes domeen-võtme normaalkujule saame lahti viimastest anomaaliast - primaarvõtme uuendamisest tingitud anomaaliatest. Kolmandalt normaalkujult domeen-võtme normaalkujule üleminekuks asendame kõigi olemite "reaalse elu andmeid" sisaldavad primaarvõtmed tehniliste primaarvõtmetega - identiteetidega *(ID - identity).* Senised primaarvõtme väljad taandatakse "tavalise andmevälja" tasemele:



Nüüd saame mudeli, kus primaarvõtmete uuendamine ei oma enam mõtet. ID on tehniline andmeelement, mille ainuke tähendus on olla sama olemi piires unikaalne. Tavaliselt kasutatakse ID-de väärtusena järjenumbreid alates 1-st. Esimese tabelisse lisatava kirje ID on 1, teisel 2, kolmandal 3 jne. Oluline on veel jälgida seda, et isegi siis kui vastava ID-ga kirje on baasist kustutatud, ei antaks seda ID-d enam ühelegi uuele kirjele (andmekomplektile). See välistab ühe uue anomaalia, seose väära taastumise anomaalia tekkimise.

tehniline võti - ID
ID ainus tähendus on olla unikaalne

Mis on siis too seose väära taastumise anomaalia? Kui käituda ebakorrektselt ja kustutada baasist suhte "üks" poolse otsa kirje nii, et seotud kirjed (seose otsa "mitu" poolel) jäävad kustutamata ja hiljem anda mõnele uuele kirjele kustutatud kirje ID, siis "haakuvad" need kirjed, mille seosed kunagi "läbi lõigati" tolle uue kirjega. Nüüd tekib andmete adekvaatsuse kadu, kuna selle uuel kirjel pole nende vanade andmetega reaalses elus mingit seost. Sellist seoste "regenereerumist" on väga raske, kui mitte peaaegu päris võimatu süstemaatiliselt avastada. Selle vältimiseks kasutatakse kirjete kaskaadkustutamist (CASCADE) – kui kustutatakse baasist ülemuskirje, kustutatakse ka kõik alluvad kirjed (seose mitmeses otsas olevad kirjed). Teine võialus on rakendada RESTRICTi st kui kirjel on alluvaid kirjeid, siis ei ole lubatud selle kustutamine. Seega tuleb kõigepealt kustutada kõik alluvad kirjed ja alles seejärel saab kustutada ülemuskirje.

seose väära taastumise anomaalia andmete adekvaatsuse kao anomaalia.

7.6. Boyce-Codd normaalkuju (Boyce-Codd Normal Form)

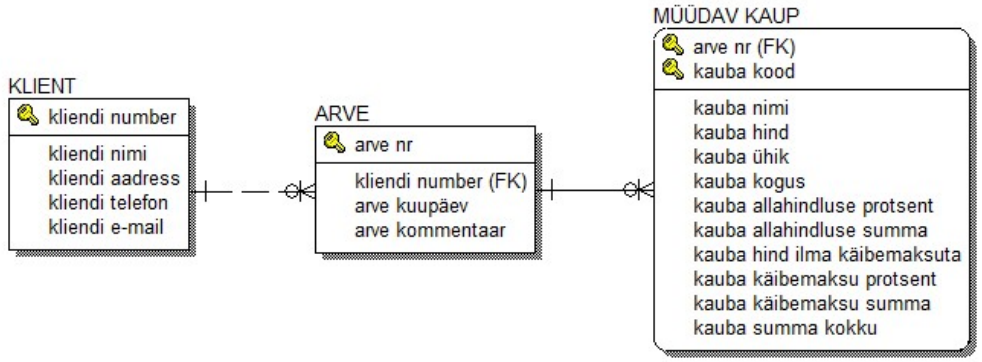
Buyce-Codd normaalkuju täidab ühe lünga normaalkujude sidususes. Eelnevalt vaatasime me normaalkujude järgnevust esimesest kolmandani. Seal saadi esimesest normaalkujust teine jaotatud võtmete kaotamisega mudelist ja teisest normaalkujust saadi kolmas transitiivse seose kaotamise teel Boyce ja Codd otsustasid liikuda esimeselt normaalkujult kolmandale normaalkujule teist teedpidi - kaotades esimesest normaalkujust kõigepealt ära transitiivsed seosed saades Boyce-Coddi normaalkuju. Sellest jaotatud võtmete kaotamisel saame jällegi kolmanda normaalkuju.

Boyce-Codd'i normaalkuju - BCNF

Esimesest normaalkujust transitiivse seose kaotamisel saame Boyce-Codd'i normaalkuju. Selleks eemaldame transitiivse seosega olemist need veerud, mis ei ole seotud primaarvõtmega ja teeme sellest veergude kogumist uue olemi. Pärast seda seome esialgse olemi ja uue olemi selliselt, et üks-mitmene suhe suundub uuest olemist sellesse olemisse, mida me ümber struktureerisime. Seejuures tuleb silmas pidada seda, seose mitmeses otsas on "mull", mis

esimesest normaalkujust saame Boyce-Coddi normaalkuju kui kõrvaldame jaotatud võtmed

kirjeldab seda, et "kõrvale tõstetud" (uude olemisse) objekt võib eksisteerida ka iseseisvalt. See oligi ju meie eesmärk . muuta senine sõltuv objekt iseseisvaks.



Uue olemi primaarvõti saadakse seal kirjeldatud objektide unikaalsest võtmest. Restruktureeritud olemi primaarvõtme struktuur jääb samaks. Seose tekitamisel ei pärita uue olemi primaarvõtit restruktureeritud olemi primaarvõtmesse vaid see jääb selliseks nagu see oli seni. Välisvõti (FK) päritakse lihtsalt tabeli koosseisu.

Selle normaalkuju anomaaliad on enam-vähem samad, mis teisel normaalkujul. Vahe on ainult selle, et kui teises normaalkujus põhjustas need anomaaliad transitiivne seos, siis siin on nende põhjustajaks jaotatud võti.

andmete lisamise anomaalia

Andmete lisamise anomaalia - jaotatud võtme välisvõtmega mitte seotud objekti andmeid ei ole võimalik baasi salvestada enne kui on tekkinud vajadus välisvõtmega seotud objekti andmete talletamiseks. Kuna üldjuhul on nende võtmega seostamata objektide näol selliste objektidega, mille andmed saaks eelnevalt baasi talletada ja seda takistab ainult mudeli puudulikkus, siis on tegemist olulise anomaaliaga.

andmete kustutamise anomaalia

Andmete kustutamise anomaalia seisneb nüüd selles, et jaotatud võtme selle objekti andmete kustutamisel, mis on seotud välisvõtmega kaovad baasist ka sõltuva objekti andmed. Kui see oli ainus koopia baasis selle objekti andmetest, siis kaovad selle andmed baasist ehkki nad vahepeal olid seal juba.

kirje uuendamise anomaalia

Andmete uuendamise anomaalia - kuna jaotatud võtme selle võtmes osaga, mis ei ole välisvõti, seotud objektide andmeid korratakse väga paljused kirjetes, siis ühe konkreetses kirjes olevate andmete uuendamisel võib tekkida vajadus uuendada andmeid väga paljudes kirjetes - kõikides kirjete, kus on sama objekti andmed. Sisuliselt on tegemist andmebaasiserveri protsessori aja raiskamisega (ühe kirje uuendamise asemel tuleb teha palju uuendusi), mis tänapäeva infosüsteemides on üks kõige kriitilisemaid ressursse.

Info liiasuse anomaalia - iga kord kui jaotatud võtmega olemi sõltuva objekti andmed kirjutatakse andmebaasi kirjutatakse sinna kõik seda objekti iseloomustavad andmed (mujale ju neid kirjutada pole), siis tekivad baasi sama objekti kohta andmed paljudes kirjetes, mis on aga ilmne info liiasus. Piisaks kui sõltuvate objektide põhilised andmed kirjutataks ühte kirjesse. Mudeli Boyce-Codd'i normaalkuju seda ei võimalda.

info liiasuse anomaalia

Andmete ebaadekvaatseks muutumise anomaalia - kuna jaotatud võtmed tingivad info liiasuse ja samu andmeid tuleb uuendada erinevates kirjetes, siis on lihtne tekkima situatsioon, kus sama objekti kohta on andmebaasi erinevates kirjetes erinevad andmed. See olukord tekkib tavaliselt programmi vigade tõttu või siis "käsitsi" andmete andmebaasis parandamise tõttu, kus mingis kirjes andmete uuendamise vajadus võib kahe silma vahele jääda. Nende vigade tekkimise hetkel on neid peaaegu võimatu avastada. Hiljem on aga väga raske määrata, millises kirjes on objekti kohta õiged andmed.

andmete adekvaatsuse anomaalia

7.7. Neljas ja viies normaalkuju (4NF - Fourth Normal Form, 5NF - Fifth Normal Form)

Neljas ja viies normaalkuju on peamiselt teoreetilist laadi normaalkujud, millel ei ole praktikas erilist tähendust.

Andmemudel on neljandal normaalkujul siis ja ainult siis, kui ta on Boyce-Codd'i normaalkujul, millel ainult ühes olemis on jaotaud võti. Selline lähenemine vähendab anomaaliate esinemist ei likvideeri neid aga täielikult.

neljas normaalkuju - 4NF

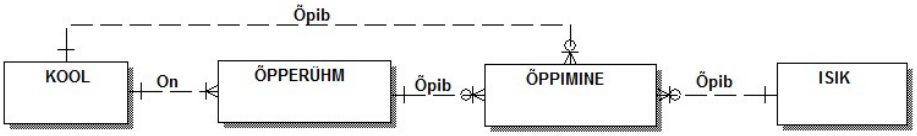
Andmemudel on viiendal normaalkujul siis ja ainult siis, kui ta on neljandal normaalkujul ja selles puuduvad suhte projitseerumised.

viies normaalkuju - 5NF

Mis on suhte projitseerumine? Suhte projitseerumine on see, kui meil on võimalik andmemudelis liikuda ühest olemist teise pikki üks-mitmese suhte päri suunda (suhte üks-otsast mitu-otsa poole) nii et liikumise semantika jääb samaks. Seejuures võidakse erinevatel teekondadel läbida erinev arv olemeid.

suhte projitseerumine

NÄIDE



Joonisel on suhte projitseerumine olemite KOOL ja ÕPPIMINE vahel - meil on võimalik öelda "kooli grupis õpib..." või "koolis õpib...". Kui vaadelda neid kahte

seost olemite KOOL ja ÕPPIMINE vahel, siis on need suhted sama tähenduslikud.

suhte projitseerumise anomaalia

Suhte projitseerumisega kaasneb ka üks anomaalia. See on andmete uuendamise anomaalia. Kui uuendada seoseid kirjeldavaid andmeid (välisvõtmeid) selles tabelis, mis asub suhte projitseerumise lõpp-punktis (antud juhul ÕPPIMINE) siis tuleb osadel juhtudel uuendada kõik välisvõtmed, kuna kui ei uuendata kõiki projitseeritud välisvõtmeid võib see viia andmete adekvaatsuse kaoni. Seda juhul, kui toimub suhte alluvuse muutus mõnes projektsiooni ahelas.

Vaatame ülal toodud näidet. Kui isik vahetab kooli, tuleb olemis ÕPPIMINE uuendada seos nii olemiga KOOL kui ka ÕPPERÜHM. Muidu võib juhtuda, et ühte seost pidi (läbi õpperühma) õpib isik ühes koolis aga seoses olemisse KOOL näidatakse seost teise kooliga. Kui muutub ainult õpperühm piisab, kui uuendada õpperühma ja õppimise vahelist seost.

suhte projitseerumise anomaalia vältimine

Kindluse mõttes on oluline iga kord, kui muutub mõni projitseeritud seostes, uuendatakse kõik projitseeritud seosed korraga.

Viiendal normaal kujul aitab suhte projitseerumise anomaaliat vältida see, kui kogu ahelas päritakse alluvate tabelite primaarvõtmetesse ülemus-tabelite primaarvõtmed, nii et nendest moodustub kogu teekonda kattev ahel. See on küll tõhus meetod, kuid praktikas väga ebamugav kasutada, kuna võtmed lähevad liiga pikaks ja muutuvad raskepäraselt käsitletavateks.

6.8. Andmemudeli denormaliseerimine

denormaliseerimine kui andmete kasutamise jõudluse tõstmise meetod

Andmemudeli denormaliseerimine on protsess, mille käigus normaliseeritud andmemudelit muudetakse andmebaasi kasutamise jõudluse vajadustest lähtudes. Selle muutmise tulemusena "antakse järele" normaliseerimise rangetele reeglitele, selleks et tagada andmemudeli alusel loodava andmebaasi kiirem kasutamine. Normaliseerimine aitab mõista andmete omavahelisi seoseid, kuid see ei ole mingi imerohi, mille kasutamisel kõik andmemudelid muutuvad "heaks ja õigeks". Tihti viib andmestruktuuri "täielik" normaliseerimine (viimine viiendale normaal kujule) andmestruktuuri sellise liigse detailsuseni, mis muudab selle andmestruktuuri kasutamise teatud kohtades väga keeruliseks.

Denormaliseerimine vastavalt kasutamise (jõudluse) vajadustele

Andmete modelleerimise teoreetikud on alati rääkinud sellest, et tavaliselt andmemudeleid üle kolmanda normaalkuju ei disainita. Siiski tehakse

kolmandale normaalkujule kasutamise jõudluse vajadustest lähtuv denormaliseerimine.

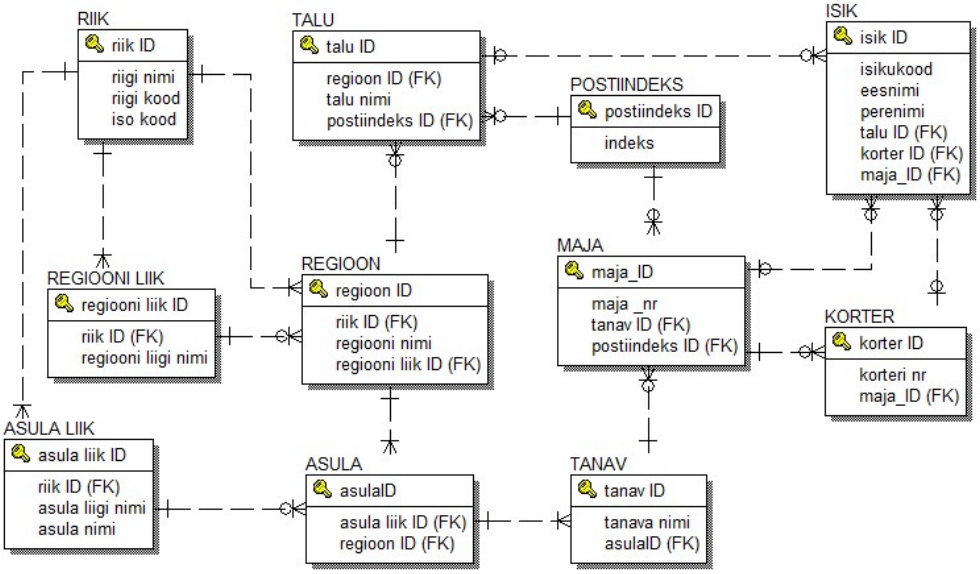
Andmemudeli denormaliseerimise käigus lisatakse andmemudeli olemitele liit-atribuute, agregeeritud andmeid, grupeerivaid atribuute ja tingimuslikult eksisteerivaid atribuutide gruppe. Mudelile kui tervikule lisatakse koond-olemeid.

7.8.1. Olemite liit-atribuudid

liit-atribuudid

Väärtused, mis normaliseerimisreeglite järgi peaks paiknema erinevates veergudes või isegi erinevates olemites on teinekord mõtet panna kokku ühe tabeli ühte veergu. Selle tulemusena saadakse liit-atribuut, mis ühendab endas erinevate semantikaga väärtusi. Parimaid sellekohaseid näiteid on aadress. Aadress koosneb tavaliselt tänava nimest, maja numbrist, ja korteri numbrist või talu nimest ja maakonna nimest, asula nimest ning postiindeksist ja riigist. Selliselt kirjeldatud aadressi normaliseeritud struktuur näeks välja järgmine (domeenvõtme normaalkujul):

NÄIDE



haldamiseks "mõttetult" keerukad andmestruktuurid

Siin on andmemudel, mille ainsaks eesmärgiks on määrata isikutele nende aadressid. Isiku andmed on olemis ISIK ja seal on kirjeldatud ka tema aadress "elamisena" kas talus, majas (era maja), või korteris. Kõik ülejäänud olemid on vajalikud selleks, et ära kirjeldada aadress-süsteem. Oletame näiteks, et sellisena on lahendatud aadressid mingis keskmise suurusega ettevõtte personali süsteemis. Kes hakkab seda aadresssüsteemi haldama? Ja kas sellel ongi üldse mõtet? On ju meil vaja vaid sisestada meie töötajate kontakt-aadressid, mitte pidada üleval tervet logistika süsteemi. Seepärast piisab, kui

me lisame olemisse ISIK ühe liit-atribuudi, kus kõik ühe inimese aadressi komponendid kirjutatakse tekstina ühte välja:

tarkvara osa andmete haldamisel

Nüüd tuleb vaid tarkvaraga, mille kaudu aadressi andmeid hallatakse, tagada selle kirje sisu korrektne struktuur. Sedasi õnnestus loobuda suurest ja antud ülesande jaoks mõttetust aadress-süsteemi ülal hoidmisest ja asendada see ühe ainsa ja lihtsalt hallatava liit-atribuudiga.

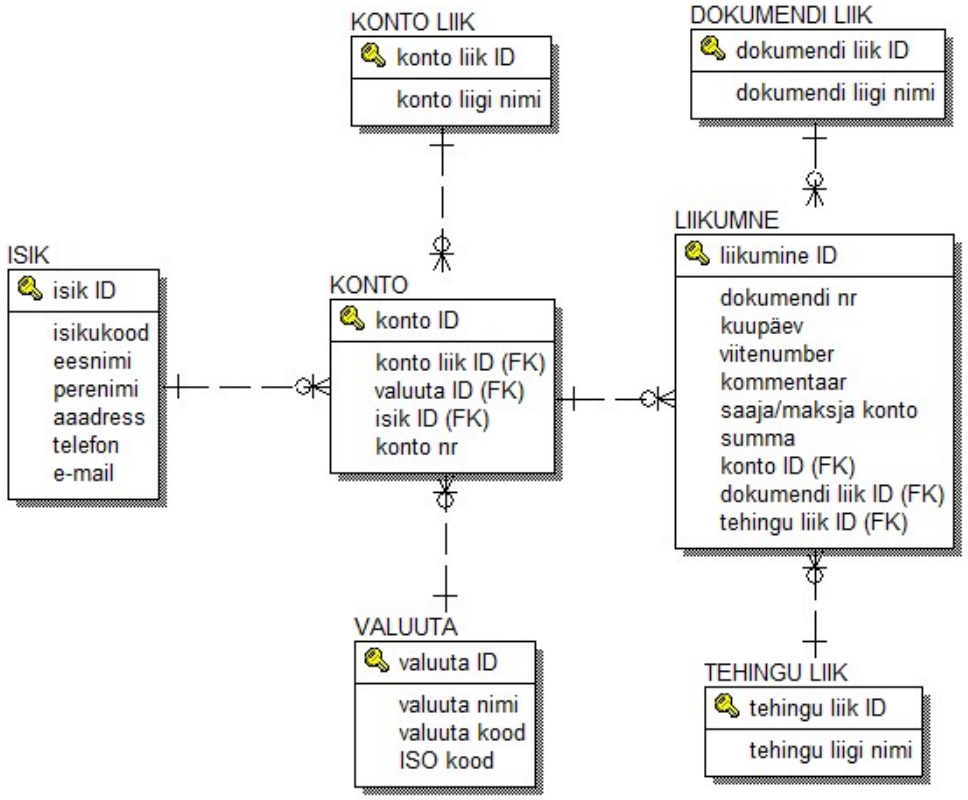
7.8.2. Andmete agregeerimine

agregeeritud andmed

Tihti on mõtet andmebaasis hoida koondsummasid, välja arvutatud keskmisi, loendusandmeid jms., mille kohta on olemas küll ka detailsed andmed, kuid millede alusel koond-summade *on-line's* ümber arvutamine võib raisata kriitiliselt palju andmebaasiserveri protsessori aja ressurssi.

NÄIDE

Toome siin näite pangandusest. Klientidel on pangas kontod, millede saldo moodustub laekumiste ja väljamaksete sumade vahest. Konto saldo on oluline andmeelement, mida kontrollitakse paljudel juhtudel ja mida võetakse arvesse paljude tehingute tegemisel. Seepärast tuleb seda arvutada üsna tihti. Mida vanem on kliendi konto, seda rohkem on seal liikumisi ja seda rohkem tuleb konto saldo leidmiseks vaadata läbi raha liikumise ridu:



agregeerimine

Siin skeemis on kõik kontol toimuvad rahalised liikumised kajastatud olemis LIIKUMINE. Raha laekumiste kirjetes on välja SUMMA väärtus positiivne, väljamaksete korral negatiivne. Konto saldo leidmiseks tuleb lihtsalt liita kokku selle kontoga tabelis LIIKUMINE seotud kirjete väljade SUMMA väärtused. Kuid nagu öeldud, mida kauem on konto eksisteerinud, seda rohkem on temaga seotud kirjeid tabelis LIIKUMINE ja seda kauem võtab aega konto saldo arvutamine. Kehtib ka reegel, et mida kauem on infosüsteem "elanud" seda suurem on saldo keskmine leidmise aeg, kuna tabeli LIIKUMINE maht tõuseb tervikuna.

uus väli agregeeritud andmete jaoks

Selleks, et antud probleem lahendada, tuleb tabelisse konto teha juurde uus väli - KONTO Saldo ja igakord, kui konto saldo muutub arvutada sinna uus konto jooksev saldo. Selleks piisab, kui võtta senine saldo ja liita sinna otsa tehingu väärtus ilma mingi liigse pärimiseta andmebaasist. Nüüd saab vajadusel pärida konto saldo ühest kohast, mis võtab oluliselt vähem aega, kui kõikide liikumise kirjete läbi vaatamine.

See näide on ainult üks paljudest võimalikest. Sedasi "ette arvutada" saab algjääke, keskmisi, standardhälbeid, loendusarve jms.

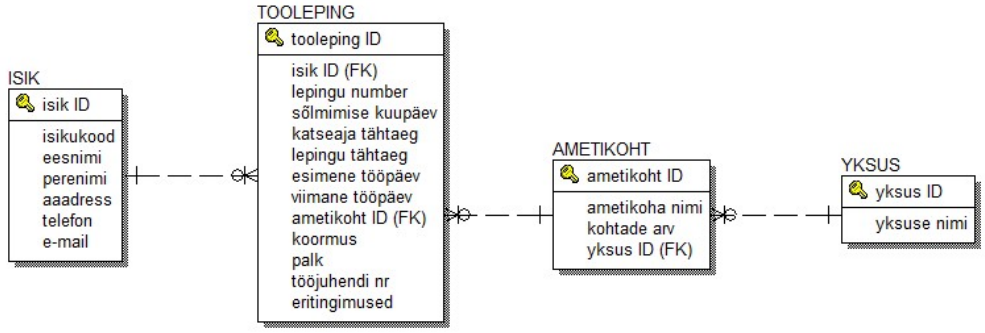
7.8.3. Grupeerivad atribuudid

grupeerivad atribuudid ja nende kasutamine primaarvõtme rollis

Vahel on vaja pidada järge objekti andmete muutumise ajaloo kohta. Sellisel juhul ei saam e uuendada kirjes andmeid vaid peame iga kord, kui andmed muutuvad, salvestama uue kirje ja sulgema vana kirje. Kirje avamine ja sulgemine toimub avamise ja lõppemise kuupäeva märkimisega. Kirje avamise (kehtima hakkamise kuupäev) on alati kirje loomise hetk. Kirje kehtivuse lõpu kuupäev jäetakse kirje loomisel tühjaks või pannakse sinna "maailma lõpu kuupäev" so. mingi väga kauge kuupäev. Praegusel hetkel on kehtivad need kirjed, mille algus-kuupäev on väiksem või võrdne kui tänane kuupäev ja lõpu kuupäev tühi või hilisem kui tänane kuupäev.

samade andmete seostamata versioonid

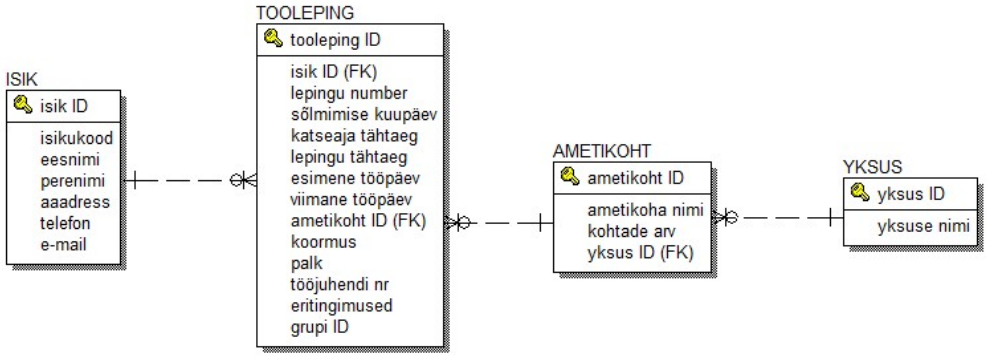
Kõik tundub kena olevat, kuid kui ainult seni, kuni me tähelepanuta jätnud ühe fakti - kõikidel sama objekti kirjeldavatel erinevatel kirjetel on erinev ID ja nad ei ole omavahel kuidagi seotud. Muutuse hetkel oli seost määratud, kuid ilma spetsiaalseid meetmeid kasutusele võtmata pole meil hiljem tagant järele võimalik seoseid taastada. Vaatame näiteks töölepingut, mille kõik muutused tuleb talletada:



Kui luuakse uus tööleping, siis kirjutatakse baasi esimene kirje. Kui töölepingus muutub amet või palk või koormus vms, siis kirjutatakse kehtivale lepingu versioonile viimase tööpäeva kuupäev, tehakse uus kirje, kopeeritakse sinna kõik vana kirje andmed, muudetakse need andmed mis muutusid ja kirjutatakse lepingule uus esimese tööpäeva kuupäev, mis nüüd märgib muutuste jõustumise kuupäeva. Kõik oleks justkui korras - meil on võimalik määrata kelle omad need lepingu versioonid on, esimese tööpäeva kuupäev määrab töölepingu variantide kirjete järjestuse ja sama töölepingu number nendes kirjetes seob isegi need kirjed ära. Siin ei tohi aga unustada ühte asja - töölepingu number võib muutuda. Samas võib töötajal olla ka mitu lepingut. Kui nüüd sellistes tingimustes muutub töölepingu number ei ole enam võimalik

öelda, millised kirjed sisaldavad sama töölepingu erinevaid variante. Seda selle pärast, et grupeeriv tingimus on lõhutud.

Sellest olukorrast aitab meid välja, kui me loome töölepingu olemisse ise grupeeriva välja grupp ID.

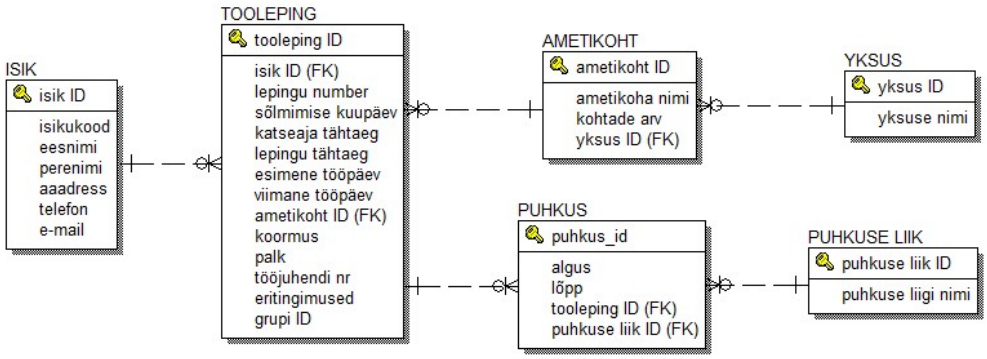


grupeeriva ID väärtustamine

Töölepingu esimese kirje baasi kirjutamisel võrdsustame selle välja töölepingu ID-ga. Hiljem, töölepingu kirje uute versioonide kirjetes jätame selle muutmata ja selliselt saamegi väärtuse, mis grupeerib 100%-lise täpsusega kõiki sama töölepingu versioone. Seejuures see kirje, milles töölepingu ID langeb kokku grupi ID-ga on töölepingu esimene versioon ja samasse gruppi kuuluv kõige suurema töölepingu ID-ga kirje on töölepingu viimane versioon.

versioneeritud kirjete teiste kirjetega sidumise raskused

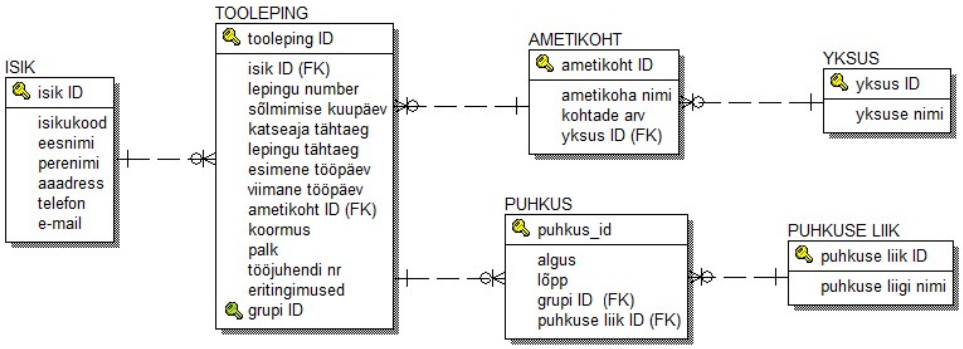
Lahendatud on veel üks probleem. Kui sedasi kirjet versioneerida, siis ei saa kirje ID-ga siduda ühtegi teist olemit, kuna objekti jooksva versiooni ID muutub kogu aeg. Nii näiteks on vaja töölepinguga siduda puhkuseid. Kui me seome puhkused töölepingu ID-ga, siis kirjutatakse puhkuse kirjesse selle töölepingu kirje ID, mis puhkuse hetkel oli aktiivne (viimane, suurima ID-ga kirje selles grupis):



grupeeriv võti andmete seostamisel

Sellisel juhul on aga väga keeruline vaadelda töölepinguga seotud puhkuste andmeid, kuna iga üks neist võib olla seotud erineva töölepingu versiooniga ja puudub neid ühtselt seostav tunnus. Kui nüüd puhkused siduda mitte

töölepingu primaarvõtmega vaid grupeeriva ID-ga, siis on probleem lahendatud, kuna kõikidel töölepingu versioonidel on see grupeeriv ID sama:



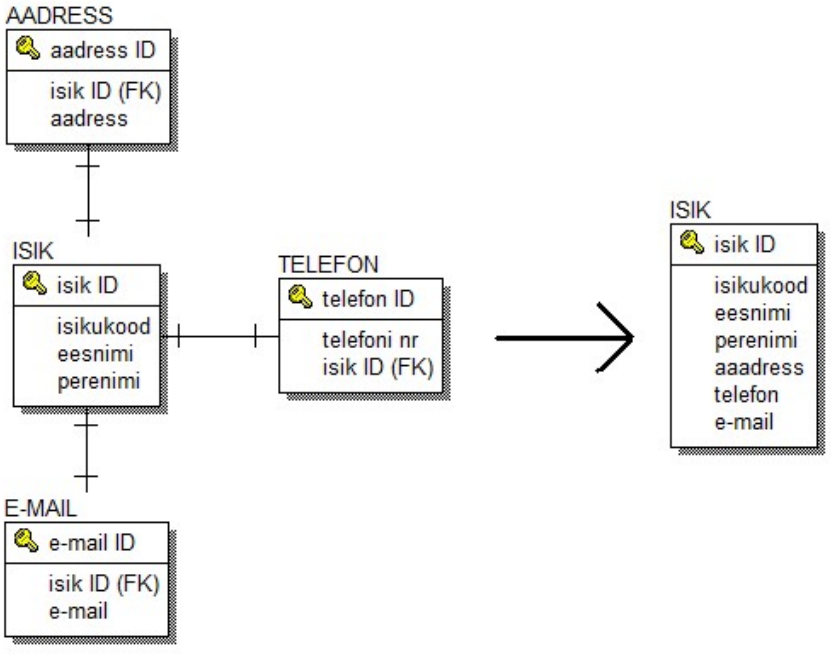
Siin on kasutatud ebaregulaarset ja normaliseerimise reeglitega vastuolus olevat konstruktsiooni - kirjeid grupeerivat võtit on olemite vahelise seose üles ehitamisel kasutatud primaarvõtme rollis

grupeeriv võti primaarvõtme rollis

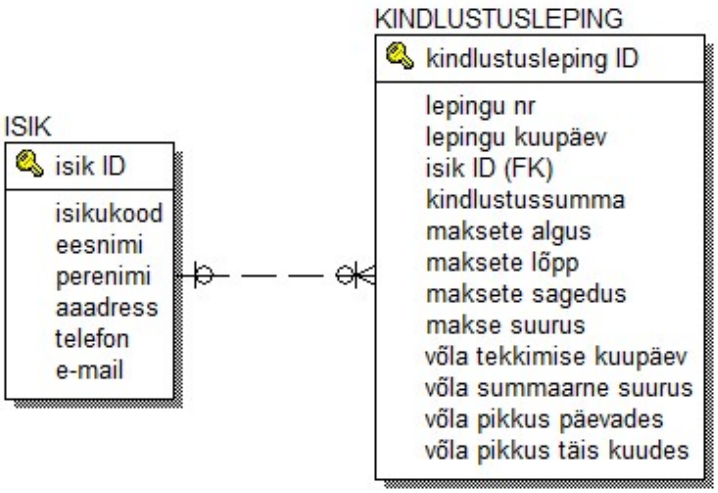
7.8.4. Tingimuslikult eksisteerivad atribuutide grupid

Tingimuslikult eksisteerivad atribuutide grupid on üks-ühese seose rakendus andmemudelites. Normaliseerimise reeglid üks-ühele seost ei tunnista, sest kui kahe olemi vahele peaks tekkima üks-ühene seos, siis tuleb need olemid ühendada üheks olemiks, mille struktuur sisaldab mõlema seni üks-üheses seoses olnud olemite atribuute (välja arvatud primaarvõti ja välisvõti, mis ühendab neid olemeid). Kui omavahel on üks-üheses seoses rohkem kui kaks olemit, siis liidetakse need kõik üheks olemiks ja kõikide liidetud olemite atribuudid kantakse üle sellesse koond-olemisse:

tingimuslikult eksisteerivad atribuutide grupid

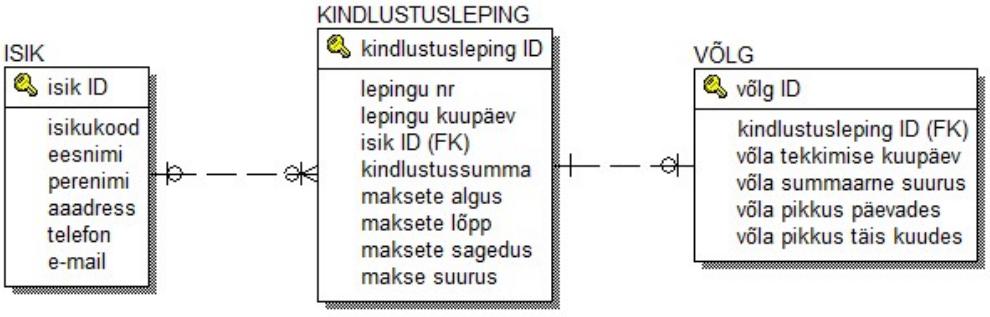


Tingimuslikult eksisteerivate atribuutide gruppide denormaliseerimise meetod eirab seda loogikat ja seetõttu on võimalik saavutada päris huvitavaid tulemusi. Olgu meil kindlustusfirma, millel on palju kindlustuslepinguid. Põhiline osa agregaat-andmeid on koondatud olemisse LEPING, kus on lepingu number, kuupäev, lepingu omanik, maksete algus, maksete sagedus, makse suurus, kindlustussumma suurus aga ka võlgnevuse tekkimise kuupäev, võla summaarne suurus, võlas oldud päevade arv ja võlas oldud täis kuude arv:



olemi jaotamine kaheks olemiks

Kogu lepingute loendist huvitavad meid kogu aeg rohkem need lepingud, mis on millegi pärast võlgnevusse sattunud. Oletame, et meil on lepinguid 100000 ja umber 6% nendest on võlas (seda teame me tehtud statistilisest analüüsist) so. umbes 6000 lepingut.. Kui meil on kõik andmed ühes tabelis siis peame me võlglaste leidmiseks igakord tegema otsingut 100 000 kirje hulgas. Tegelikult oleks aga hea kui suudaksime need 6000 võlas olevat lepingut kuidagi füüsiliselt eristada. See võimaldaks meil oluliselt hoida kokku päringuks kulutatavat protsessori aega. Siin tulevadki meile kasuks teadmised üks-ühestest suhetest. Kui me jaotame praeguse laenulepingute olemi kaheks olemiks selliselt, et ühte olemisse jäävad lepingu andmed ja teise võla andmed. Kuna üks leping saab korraga võlas olla ainult üks kord, siis tekib loodud olemite vahele üks-ühene suhe:



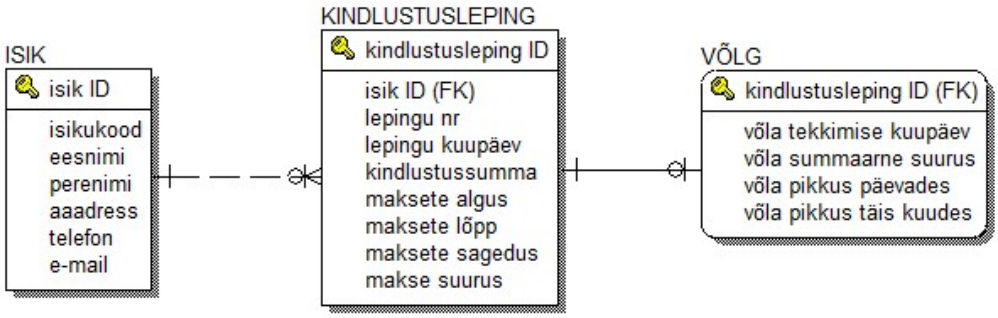
läbi vaadatavate kirjete arvu vähendamine

Olemisse VÕLG tekivad kindlustuslepingu kohta andmed ainult siis, kui leping on võlas. Seega selles tabelis ongi meil alati umbes 6000 rida ja kuna need on

ainult võlas olevate lepingute andmed, siis võlas olevate lepingutega tegelemisel me selle lepinguga töötamegi - 100 000 kirje asemel tegeleme nüüd 6000 kirjega. Kokkuhoid missugune.

primaar- ja välisvõtme ühendamine

Ühe optimisatsiooni võime teha veel. Kuna olemis VÕLG saab olemi KINDLUSTUSLEPING iga rea kohta olla ainult üks rida, siis ei vaja see rida eraldi ID-d vaid me võime kasutada selleks kindlustuslepingu ID-d. Nüüd tekkib huvitav situatsioon, kus primaarvõti on ka välisvõti (FK).



Sellist andmeskeemi üks-ühes seose esitust, kus seotud olemi primaarvõti on ühtlasi ka välisvõti, mis seob teda põhi-olemiga, nimetataksegi tingimuslikult eksisteerivaks atribuutide grupiks - seotud olemis olevad väärtused eksisteerivad ainult osadel põhi-olemis kirjeldatud objektidel. Seda siis, kui tingimused nende esinemiseks on täidetud. Meie näites oli selleks tingimuseks võla tekkimine.

7.8.5. Koond-olemid

koond-olemid, andmete valimite kollektsioneerimine

Analoogiliselt agregaat-summadega, mis on üksikud andmeelemendid, on mõni kord vaja hoida operatiivselt üleval terveid andmete loendeid. Põhjuseks on see sama, mis agregaatsummade puhulgi - iga kord kui on vaja andmeid sellest loendist, on väga kulukas seda üles ehitama hakata. Samas, kui vajalikke andmeid kollektsioneerida "vähe haaval", andmete muutumise protsessi kestel. Siis hajub selle loendi moodustamise aeg üldisesse töö protsessi ja tegelikult kulub ka aega vähem, kuna loendisse lisatakse andmeid ainult siis kui nad tekivad, uuendatakse andmeid kui nad muutuvad ja kustutatakse nad sealt, kui vajadus nende järele kaob.

spetsiaalne olem andmete kollektsioneerimiseks

Sellise metoodika rakendamiseks luuakse andmemudelisse spetsiaalne olem, mille struktuur vastab kollektsioneeritavate andmete kooslusele. Andmed kollektsioneeritakse sellese tabelisse ja kõik hilisemad päringud tehakse sellesse tabelisse.

7.8.6. Andmemudelite denormaliseerimise kokkuvõte

denormaliseerimise ainsateks põhjenduseks saab olla tarkvara jõudluse tõstmine ja mudeli põhjendamatu keerukuse lihtsustamine

Denormaliseerimiseks saab olla AINULT KAKS põhjust - loodava tarkvara jõudluse tõstmine või süsteemi põhjendamatu keerukuse vähendamine. Mitte mingid muud põhjused ei saa olla andmemudeli denormaliseerimise põhjuseks. Kuna normaliseeritud mudelile vastava andmebaas võtab alati vähem ketta ruumi kui sama mudel denormaliseerituna (denormaliseerimine lisab alati info liiasust), siis mälu ruumi kokkuhoid denormaliseerimise põhjendusena on enese pettus. Süsteemi keerukuse põhjendust saab hinnata ainult süsteemi lõppkasutajaga koostöös. See hinnang on alati subjektiivne ja selle täpsus on otseses sõltuvuses projekteerijate kogemusest.

mudeli denormaliseeriva muudatuse otsustamine

Mudeli iga denormaliseeriva muudatuse käigus tuleb hinnata, mida muudatus kaasa toob - millised on plussid (head omadused, mis mudelile juurde tulevad) ja millised on miinused (halvad omadused, mis mudelile juurde tulevad), ning kumb kumba üles kaalub. Isegi siis, kui esmasel hinnangul kaaluvad mudelile lisandunud head omadused üles lisandunud halvad omadused, ei tohi kohe otsustada muudatuste kasuks. Enne lõpliku otsuse tegemiseks tuleb läbi mõelda see, kuidas me saavutame mudelile lisandunud halbade omaduste riskide maandamise ja millised on need meetodid andmete käsitlemiseks, mis võimaldavad rahuldava mugavuse ja kiirusega andmekäsitluse mudeli denormaliseeritud osas, vaatamata tekkinud ebakõladele.

denormaliseerimine ja andmemudelite areng

On üks asi veel, mida tuleb enne andmemudelit denormaliseeriva muudatuse lõplikku läbi viimist. Mudelis seda kohta tuleb vaadata tuleviku arengute seisukohast - tuleb kaalutleda, kuidas läbi viidav muudatus võib hakata tulevikus mõjutama mudeli arengut. Andmemudelit denormaliseerivatel muudatustel on üldjuhul andmemudeli arengut suunav iseloom - nad hakkavad piirama mudeli teatavaid "loomuliku arenemise" suundi ja sunnivad peale teatavaid spetsiifilisi muudatusi, mis kaasnevad tulevikus pea kõigi muudatustega. Seepärast tuleb iga andmemudelit denormaliseerivat muudatust enne selle andmemudelisse sisse viimist kaaluda väga hoolega. Paljud infosüsteemid on oma arengus jõudnud "tupikteeni" just andmemudeli arengu varajases arengustaadiumis tehtud hästi läbi kaalumata denormaliseerimiste tõttu. Samas võib olla tegemist ka vastupidise efektiga - infosüsteemid on jõudnud oma arengus "tupikteeni" just liigselt normaliseeritud (denormaliseerimata) andmemudelite tõttu. Nii et jõudu tööle - otsused on teie teha.

normaliseerimine v. denormaliseerimine

Andmete normaliseerimise teooria ütleb, et andmemudel tuleb kõigepealt normaliseerida nii täiusliku tasemeni ja alles pärast seda, kui kõik elementaarsed andmekooslused ja seosed on selgunud, tohib hakata denormaliseerima. Tegelikkuses oskavad kogenud andmemudelite loojad, oma kogemustest lähtudes, teha "õigel tasemel" normaliseerimise, mis sisaldab juba "vajaliku koguse" denormaliseeritud andmestruktuure, juba projekteerimise üldise protsessi käigus. Algajatel tuleb muidugi läbida just seda esimesena pakutud toimimismudelit - kõigepealt detailne normaliseerimine, siis jõudluse analüüs ja denormaliseerimine.