TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

CASE vahendite poolt pakutavad andmemudelite kontrollimise võimalused

magistritöö

Üliõpilane: Triin Merivald
Üliõpilaskood: 132550IABM
Juhendaja: dotsent Erki Eessaar

Tallinn 2014

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Triin Merivald .....

(*allkiri*)

[kuupäev]

Annotatsioon

Töö eesmärgiks on uurida andmemudelite automaatse kontrollimise võimalusi CASE vahenditega. Selleks tuli kaardistada ja kategoriseerida andmemudelitele rakendatavad reeglid, võrrelda erinevate CASE vahendite poolt pakutavaid võimalusi andmemudelite automaatseks kontrollimiseks, viia ühe CASE vahendiga läbi eksperiment ja analüüsida tulemusi. Eksperiment kujutab endast meelega lisatud probleemseid kohti sisaldava andmemudeli automaatset kontrollimist väljavalitud CASE vahendiga.

Andmemudelite kvaliteet on otseselt seotud selle poolt kirjeldatud andmebaasile toetuva tarkvara kvaliteediga. Igasuguste vigade ja probleemide lahendamine on lihtsam ja odavam (tarkvaraarenduse) protsessi alguses. Mudelipõhises arenduses oleks kasulik tuvastada vead ja need parandada juba mudelites, enne, kui neist midagi (uus mudel, kood, test, aruanne) genereeritakse. Mudelitest probleemseid kohti otsida ja neid parandada saab ka käsitsi, kuid selle protsessi automatiseerimine aitaks tõsta efektiivsust ja vähendada otsimisel tehtavate vigade hulka.

Töös kaardistasin andmemudeli kontrollimise reeglid, võttes aluseks Oracle SQL Developer Data Modeler CASE vahendi, kuna töö algusest saati oli mulle teada, et see vahend võimaldab andmemudeleid automaatselt kontrollida. Lisasin ka ise reegleid, mille kontrollimist pidasin oluliseks. Kaardistatud reeglid jaotasin kolme kategooriasse: kontseptuaalse, loogilise ja füüsilise andmemudeli kontrollreeglid. Iga reegel võib kuuluda ühte või mitmesse kategooriasse. Võrdlesin kaheksat CASE vahendit andmemudelite kontrollimise funktsionaalsuse osas ja leidsin, et neist kõige võimekam on PowerDesigner. Lisasin PowerDesigneris eeldefineeritud reeglitele ise paar kontrollreeglit ja viisin läbi eksperimendi, mille käigus lisasin andmemudelitesse (loogilisse ja füüsilisse) meelega probleemseid kohti ning lasin PowerDesigneril need tuvastada. Viimaks analüüsisin eksperimendi tulemusi, mille käigus möönsin, et PowerDesigner on andmemudelites probleemsete kohtade tuvastamisel tõhus ja et andmemudelite automaatne kontrollimine võiks leida laialdasemat kasutust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 74 leheküljel, 3 peatükki, 30 joonist, 5 tabelit.

Abstract

Title of the thesis is Possibilities Provided by CASE Tools to Check Data Models. The aim of my work is to investigate automated data model checking with CASE tools. For that I had to gather and categorize rules applicable to data models, compare different CASE tools regarding possibilities for automatic data model checking, carry out an experiment of data model checking with one of the tools and analyse the results. The experiment consists of creating problematic places in a data model and letting the chosen CASE tool to detect them.

Data model quality is directly linked to the quality of the software that uses the database that has been created according the model. It is easier and cheaper to correct mistakes in earlier phases of (software) development process. In case of model driven development, one should solve problems in the models before generating a new artefact (a new model, code, test, report) from them. One can check and correct models manually, but automated process would increase efficiency and reduce possible mistakes.

To gather rules to be checked automatically with CASE tools, I took Oracle SQL Developer Data Modeler CASE tool as the reference, since I knew from the beginning that this tool provides automatic checks for data models. I added some other rules that I thought could and should be checked. I divided all the rules into three categories: rules for conceptual data model, rules for logical data model and rules for physical data model. Each rule can belong to one or more of those categories. I compared eight different CASE tools in the aspect of model checking. PowerDesigner turned out to be the most advanced. Therefore, I selected this tool to carry out my experiment. In addition to the predefined rules, I added some custom checks to PowerDesigner myself. After that I carried out the experiment by adding mistakes to logical and physical data model and let PowerDesigner to detect them. I analysed the results of the experiment and concluded that PowerDesigner is efficient for checking models and the automaded data model checking practice should be more widely used.

The thesis is in Estonian and contains 74 pages of text, 3 chapters, 30 figures, 5 tables.

Lühendite ja mõistete sõnastik

MDD	***Model-Driven Development*** Mudelikeskne arendus
AMDD	***Agile Model-Driven Development*** Agiilne Mudelikeskne arendus
CASE	***Computer Aided Software Engineering*** Arvutipõhine tarkvaratehnika
UML	***Unified Modelling Language*** Unifitseeritud modelleerimiskeel
ERD	***Entity-Relationship Diagram*** Olemi-suhte diagramm

Jooniste nimekiri

Tabelite nimekiri

Sisukord

Sissejuhatus

Halvasti üles ehitatud, puudulikult kommenteeritud andmebaas on üsna sage nähtus. Halb andmebaasi disain mõjub halvasti infosüsteemi jõudlusele, hooldatavusele ning edasiarendatavusele. Juba kasutusele võetud ja rakendatud halvasti disainitud andmebaasi võib olla raske muuta. See tähendab täiendavat süsteemile kuluvat arendustööd, mis kujutab endast ajalist ja rahalist kulu. Sõltuvalt andmebaasi muudatuste ulatusest võib selline arendus üpris kulukaks osutuda. Andmebaasi disaini vigade vältimine andmete modelleerimise käigus on tunduvalt soodsam ja valutum kui valmistehtud ja kasutuses oleva andmebaasi parandamine. Et andmete modelleerimisel õigeid valikuid teha, tulevad appi andmemudeli kontrollimise reeglid.

Andmemudelil on tarkvaraarenduse maailmas kaks tähendust. Nendest üks tähendab andmebaaside üldiste ehitusplokkide, kitsenduste ja operatsioonide kirjeldust (nagu näiteks relatsiooniline andmemudel). Antud töö kontekstis pean andmemudeli all silmas konkreetse andmebaasi moodustavate andmeobjektide kirjeldust koos nende atribuutide ja suhetega (kontseptuaalne andmemudel) või kontseptuaalsest kirjeldustest tuletatud andmestruktuuride kirjeldust (loogiline ja füüsiline andmemudel). Andmete modelleerimisel võib kasutada erinevaid keeli/notatsioone (UML, Information Engineering, Barkeri notatsioon, Object-Role Modeling, IDEF1X jne). Käesolev töö käsitleb olukorda, kus:

- andmebaas luuakse SQL-andmebaasina,
- andmete modelleerimisel kasutatakse UML või Information Engineering notatsiooni,
- andmebaasi projekteerimisel luuakse kontseptuaalne-, loogiline- ja füüsiline andmemudel.

Teadaolevalt pakub Oracle SQL Data Modeler CASE vahend võimalust SQL-andmebaaside kavandamise käigus loodud andmemudeleid automaatselt kontrollida. Vahendis on juba eeldefineeritud hulk reegleid, kuid neid on võimalik ka ise juurde kirjutada. Töö üheks eesmärgiks on kaardistada ja kategoriseerida reeglid, mille abil saaks andmemudeleid automaatselt kontrollida. Samuti selgitan välja, millised on mõningate teiste CASE vahendite andmemudelite automaatse kontrollimise võimalused. Võrdlen väljavalitud CASE vahendite

võimalusi andmemudeli kontrollimiseks ja valin eksperimendi läbi viimiseks välja ühe vahendi. Eksperimendi eesmärgiks on hinnata kontrollreeglite tõhusust andmemudelite kontrollimisel. Antud töös uurin vaid andmemudelite kui eraldiseisvate tulemite kontrollimist ja ei käsitle mudelite omavahelise kooskõla kontrollimist.

Töö tulemused on kasulikud andmete modelleerimise CASE vahendite valijatele, kes võiksid eelistada CASE vahendeid, milles on paremad võimalused andmemudelite kontrolliks. Töö tulemused on ka kasulikud CASE vahendite arendajatele ja andmete modelleerijatele, kes saavad tööst ideid, milliseid kontrolle andmemudelitega seoses läbi viia. Andmete modelleerijad võivad muidugi neid kontrolle viia läbi ka käsitsi kuid kui kasutusel on CASE süsteem, siis oleks efektiivne selliseid kontrolle nii palju kui võimalik automatiseerida. Töö tulemused pakuvad huvi ka ettevõttele, kus töötan, kuna siiani on mudeleid käsitsi kontrollitud ja automaatse kontrollimise võimalusest ei olda teadlikud. Kui oma töö teemast kolleegidele rääkisin, pakuti välja, et võiksin tulemusi ka töö juures tutvustada. Selles mõttes on kasulikud ka kasutusjuhendile sarnased kirjeldused, mis leiduvad töö eksperimendi osas. Need võivad aidata veenda kolleege, et mudelite kontrolli funktsionaalsus ei ole keeruline, kuid on kasulik CASE vahendi funktsionaalsus.

Töö on jaotatud kolmeks suuremaks osaks. Esmalt tutvustan mudelipõhise arenduse ja andmete modelleerimise teoreetilist tausta, kirjutan andmemudelite kvaliteedist ja kaardistan ning kategoriseerin reeglid, mida andmemudelite juures automaatselt kontrollida. Pärast kontrollreeglite kaardistamist tutvustan CASE vahendeid, mille oma töö jaoks välja valisin. Viimases osas võrdlen erinevate CASE vahendite võimalusi andmemudelite kontrollimiseks ning valin välja ühe CASE vahendi, millega läbi viia andmemudeli kontrollimise eksperiment. Viimaks analüüsin eksperimendi tulemusi.

1. Mudelikeskne arendus

Infosüsteemide disaini nõuded muutuvad üha keerukamaks. Nõutakse suuremat jõudlust ja vastupidavust, kõrgemat efektiivsust ja turvalisust, rohkem funktsionaalsust, väiksemat hinda, madalamat energiakulu jne. Samas on kasutusel väga palju erinevaid tehnoloogiaid ja kõik need tuleb omavahel kuidagi suhtlema ja koos töötama panna. Mudelikeskne arendus (*Model Driven Development*, MDD) aitab kogu seda keerukust kergemini hallata.

MDD on tänapäeval tänu tehnoloogia arengule laialdaselt kasutusele võetud lähenemine tarkvara arendusele. MDD tähendab, et tarkvara tootmist alustatakse mudelite tegemisega, millest hiljem genereeritakse reaalne tarkvara. Mudel on reaalsuse lihtsustatud kirjeldus. Antud kontekstis räägime mudelite alamosast ehk infosüsteemi või selle osa kirjeldavatest mudelitest. MDD-le on iseloomulik, et mudel ja tarkvara peavad kogu aeg sünkroonis olema. Selleks piisab muudatuste tegemisest ühes kohas: mudelis või koodis. CASE vahendi abil saab mudelid ja koodi omavahel uuesti sünkroniseerida. Hea praktika kohaselt tehakse muudatused enne mudelis ja need kantakse automaatselt üle koodi.

MDD-d on otstarbekas rakendada kuna see võimaldab süsteemi arhitektuuri mudelite abil detailselt analüüsida, samas ei ole see tarbetu ajakulu, kuna kõik, mis on mudelina kirja pandud, on ühtlasi ka osa reaalsest lahendusest. Asjaolu, et mudelid kajastavad lihtsustatult seda, milline süsteem reaalselt on või olema peab ja kuidas see toimib, muudab mudelid erinevatele projektiga seotud osapooltele väärtuslikuks infoallikaks. Samuti toetab MDD süsteemi dokumenteerimist, mida tänases agiilses tarkvaraarenduse maailmas kiputakse unarusse jätma. Mudelite muutmisel tuleks alles hoida ka varasem versioon, sest nii säilib info mudelites aja jooksul tehtud muudatuste kohta. [1]

MDD vähendab riski, et arendajad saavad ülesandest valesti aru ja teevad töö oodatust teisiti. Kui tarkvara mudelid on täpselt ära kirjeldanud koos kõikvõimalike reeglite, piirangute ja kitsendustega ning lisatud veel ka näiteks ühiktestid, mille töötav tarkvara peab läbima, on arendajal väga selged raamid ja täpne arusaam sellest, mida ja kuidas tuleb teha. Sel moel vähendab MDD riske, tõstab kvaliteeti ja suurendab tarkvara arendamise tõhusust. [1]

Tarkvaraarendus järgib üha enam agiilseid printsiipe. MDD käib ajaga kaasas ja nii on välja kujunenud agiilne mudelikeskne arendamine (*Agile Model-Driven Development*, AMDD). Traditsioonilise MDD korral tehakse põhjalikud kõikehõlmavad mudelid kohe alguses ja siis genereeritakse nende pealt tarkvara. Nii nagu agiilne tarkvaraarendus kasutab iteratiivset

lähenemist, saab ka MDD-le läheneda iteratiivselt. Vaata Tabelist 1, kuidas on MMD ja agiilne arendus omavahel seotud. AMDD korral luuakse igas iteratsioonis agiilseid mudeleid, mis katavad vaid hetkevajadused (*just barely good enough*). Igas iteratsioonis modelleeritakse täpselt nii palju, kui on vajalik, et käesoleva iteratsiooni eesmärgid saaksid täidetud. [2, 3, 4, 5]

Tabel 1. MDD praktikad ja vastavad agiilsed praktikad. Maatriksis tähistab s seotust ja v vastavust. [2]

		Agiilsed praktikad							**Agiilse juhtimise praktikad**			
		Paarisprogrammeerimine	Testipõhine arendus	Iteratiivne ja inkrementaalne arendus	Töötav tarkvara on olulisem kui ulatuslik dokumentatsioon	Automatiseeritud regressioonitestimine	Pidev integratsioon	Tihe tagasiside	Igapäevane kiirkoosolek	Igapäevane arenduse staatuse jälgimine	Lühikesed sprinditsüklid	Sprindi järgne koosolekud (Retrospectives)
MDD praktikad	Modelleerimine kui koodi kirjutamine	s										
	Paarismodelleerimine	v										
	Testipõhine modelleerimine		v									
	Iteratiivne ja inkrementaalne modelleerimine			v								
	Modelleerimine kui disaini dokumentatsioon				s							
	Automatiseeritud simuleerimine					v						
	Pidev modelleerimine						v					
	MDD vahendite ahel							s				
AMDD juhtimise praktikad	Igapäevane MDD plaani uuendamine								s			
	Igapäevane MDD ülesannete staatuste uuendamine									s		
	MDD plaan käesoleva sprindi jaoks										s	

		Agiilsed praktikad							Agiilse juhtimise praktikad			
		Paarisprogrammeerimine	Testipõhine arendus	Iteratiivne ja inkrementaalne arendus	Töötav tarkvara on olulisem kui ulatuslik dokumentatsioon	Automatiseeritud regressioonitestimine	Pidev integratsioon	Tihe tagasiside	Igapäevane kiirkoosolek	Igapäevane arenduse staatuse jälgimine	Lühikesed sprinditsüklid	Sprindi järgne koosolekud (Retrospectives)
	MDD viimase sprindi õppetunnid (mida jätkata, muuta, proovida)											s

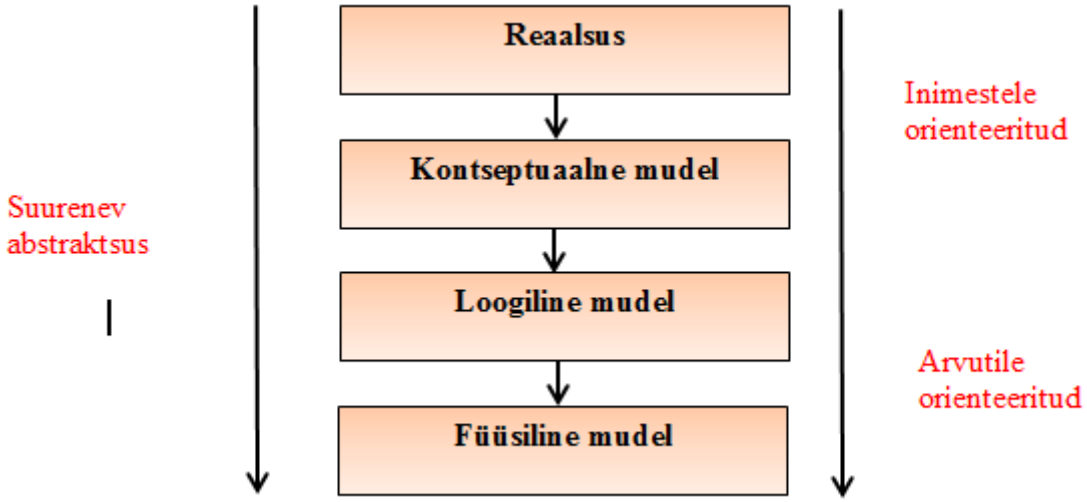
Mudelikeskset arendust muudab raskemaks see, et kasutajad peavad süsteemi mitmevaateliseks kirjeldamiseks looma samaaegselt erinevat tüüpi mudeleid ning tagama nende mudelite sisemise ja ka omavahelise kooskõla. Sageli kasutatakseerinevate mudelite loomiseks erinevaid keskkondi (erinevad CASE vahendid, elektroonilised joonistusvahendid, tahvlile joonistamine), mis ei muuda olukorda sugugi lihtsamaks. Dori [6] märgib, et isegi suurepärase CASE vahendi toe olemasolu korral muutub see ülesanne arendajate jaoks väga keeruliseks ja raskeks. Töö autor on seisukohal, et kui soovitakse mudelipõhist arendust kasutada on igasugune süsteemi poolne abi teretulnud ja kui süsteem suudab ise mudelit kontrollida, siis see aitab modelleerimisele kaasa kuid ei lahenda loomulikult võluvitsaga kõiki probleeme.

MDD ning AMDD põhimõtted on rakendatavad ka andmebaaside loomisel. Väga paljud CASE vahendid toetavad mudelite põhjal andmebaasi skeemi genereerimist ja ka vastupidi, andmebaasi skeemi põhjal seda skeemi kirjedava mudeli tekitamist.

1.1 Andmete modelleerimine

Andmebaas on enamikele rakendustele nurgakiviks. Seetõttu on oluline andmebaas hästi disainida. Traditsioonilises arenduses koosneb andmete modelleerimine kolmest sammust: kontseptuaalse andmemudeli koostamine (kontseptuaalne disain), loogilise andmemudeli

koostamine (loogiline disain), füüsilise andmemudeli koostamine (füüsiline disain). Iga sammuga muutuvad mudelid detailsemaks ja tehnilisemaks (vaata Joonis 1).



Joonis 1 Andmemudelite abstraktsuse tasemed [10]

Kontseptuaalne andmemudel on kõrgtaseme andmeobjektide kirjeldus: esitatakse sellised olemitüübid ja nendevahelised seosetüübid, millele vastavaid andmeid soovitakse hakata andmebaasis talletama. Kontseptuaalne andmemudel ei ole kuidagi seotud ega kajasta kasutatavat andmebaasisüsteemi ega selle aluseks olevat ja andmebaasi üldiseid ehitusplokke kirjeldavat andmemudelit. Seda kasutatakse ärinõuete esialgseks paika panekuks koos äripoole esindajatega. [7, 8] Väga levinud meetodiks on kasutada kontseptuaalse andmebaasi disaini tulemuste esitamiseks olemi-suhte diagramme (*Entity Relationship Diagram*, ERD)[11].

Loogiline andmemudel on juba andmebaasi tehniline kirjeldus, mis arvestab andmebaasi aluseks oleva andmemudeliga (näiteks SQL aluseks olev mudel), kuid mitte veel konkreetse andmebaasisüsteemiga, milles see andmebaas realiseeritakse. Loogilisi andmemudeleid kasutatakse enamasti traditsioonilistes tarkvaraarendusprojektides, kuid agiilsetes tarkvara projektides jäetakse see samm sageli vahele ja pärast kontseptuaalse andmemudeli koostamist asutakse kohe kujundama füüsilist andmemudelit. [7]

Füüsiline andmemudel on aluseks reaalse andmebaasi loomisel. Füüsilise andmemudeli loomisel on juba määratletud ka andmebaasihalduse süsteem (andmebaasisüsteem). Füüsiline andmemudel kirjeldab SQL-andmebaasi korral detailselt ära kõik tabelid, veerud, kitsendused (sh välisvõtme kitsendused), andmetüübid lähtuvalt andmebaasisüsteemist ja muud tehnilised

parameetrid. Enamus kaasaegseid CASE vahendeid võimaldavad füüsilise andmemudeli põhjal genereerida andmebaasi skeemi loomiseks mõeldud andmebaasikeele laused.

Andmete modelleerimisele saab samuti läheneda agiilselt, lähtudes AMDD põhimõtetest. Sellisel juhul täiendatakse andmemudeleid iteratiivselt projekti iteratsioonide käigus. Agiilses projektis, milles kasutatakse näiteks *Agile Unified Process* või *Scrum* metoodikat, visandatakse tavaliselt kontseptuaalne andmemudel näiteks tahvlile, kuhu see jääb projekti lõpuni. See peaks meeskonnaliikmetele alati lihtsalt kättesaadav olema. Sellest lähtudes on meeskonnaliikmetel hiljem ühine keel klasside ja tabelite kavandamiseks. Agiilsetes tarkvaraprojektides areneb füüsiline andmemudel koos ülejäänud tarkvaraga samm-sammult. Nii nagu kõigi mudelitega AMDD korral, lähtutakse ka andmemudelite koostamisel põhimõttest, et igas iteratsioonis tehtaks just nii palju, kui on hetkel vaja, kuid mitte rohkem. „*Agile data models are just barely good enough for the task at hand.* „*Agile developers solve today's problem today and trust that they can solve tomorrow's problem tomorrow*[9].“ Kõik andmebaasi muudatused (uute tabelite ja veergude lisamine, tabeli ja veergude nimede muutmine, veergude andmetüüpide muutmine, tabelite ja veergude kustutamine) tehakse paralleelselt koodimuudatustega. See eeldab tihedat koostööd projekti meeskonnaliikmete vahel. Seega on agiilne andmete modelleerimine nii evolutsiooniline kui ka koostööle suunatud. Evolutsioonilise andmebaasi arendamise miinuseks on pidev andmebaasi refaktoreerimise vajadus. See lisab projekti keerukust, kuid on möödapääsmatu, kui tahetakse lähtuda agiilse arenduse põhimõtetest. [9, 4, 5]

IT arendusele pühendunud ettevõttes kus töötan ei ole kehtestatud ühtset arendusmetoodikat. Mitte üheski projektis, kus olen osalenud, ei ole olnud metoodika määratletud. Tehakse pigem nii, nagu on varem tehtud (kui on tegu mingi pikaaegse kliendiga) või nii nagu projektijuht õigeks peab. Suuremates projektides, kus olen osalenud, ei olnud aru saada, kas tegemist on kosemudelil põhineval arendusel või on tegemist agiilse arendusega – võis leida nii ühele kui teisele omaseid tunnuseid. Minu kõige esimene projekt, kus osalesin nooremanalüütikuna ja kus andmete modelleerimisega kokku puutusin, oli ühe äriprotsessi automatiseerimine ettevõtte pikaaegsele koostööpartnerile. Seal tuli teha andmemudel ja andmebaas nullist. Vaatamata sellele, ei kasutanud ma CASE vahendit kontseptuaalse ega loogilise andmemudeli tegemiseks. Hakkasime kohe tegema füüsilist andmemudelit. Mingi visiooni saamiseks kritseldasin küll tabelid esmalt paberile, kuid seda ei saa nimetada kontseptuaalseks mudeliks. Pigem oli tegu füüsilise andmemudeli mustandiga. Selline lähenemine oli võimalik tänu sellele, et arendatav süsteem ei olnud väga suur (andmebaasi tuli tabeleid umbes 15) ja vanemanalüütikul, kellega

koos töötasin, oli selle kliendiga juba pikaaegne koostöökogemus ning seetõttu tundis nende süsteeme ja äriolemeid. Järgmine projekt, milles osalesin, oli juba suur projekt, mille käigus tuli liidestada kõik Eesti Politsei- ja Piirivalve Ameti süsteemid Schengeni Infosüsteemiga. Selles projektis ma andmemudeleid ise ei teinud. Pidin küll olemasolevatesse tabelitesse uusi veerge lisama, kuid vastavad SQL laused kirjutasin käsitsi. Sama projekti käigus pidi andmeid modelleerima üks minu kolleegidest ja ta kasutas selleks CASE vahendit (Enterprise Architect), millega tegi vaid füüsilise andmemudeli (lisaks ka oleku- ja tegevusdiagrammid). Ka ülejäänud projektides, kus olen osalenud, olen kokku puutunud vaid füüsilise andmemudeliga.

Arvan, et kontseptuaalne ja loogiline andmemudel on vajalikud eelkõige siis, kui hakatakse arendama täiesti uut süsteemi ja projekti käigus tuleb teostada ka ärianalüüs Loogilise andmemudeli võib küll agiilsetes projektides välja jätta (kui kasutatav andmebaasisüsteem on algusest peale teada), kuid kontseptuaalse mudeli kasutamine on hädavajalik. Kui aga tegu on mingi süsteemi edasiarendamisega ja andmemudel palju ei muutu, piisab füüsilise andmemudeli täiendamisest. Samuti ei ole kontseptuaalne ja loogiline andmemudel hädavajalikud kui arendatav süsteem on väga pisike ja äriobjektid, reeglid- ja –protsessid on analüütikutele/arendajatele hästi teada.

1.2 Andmemudeli kvaliteet

Kuigi andmete modelleerimine moodustab vaid ühe osa infosüsteemi arendamise jõupingutustest, on selle mõju süsteemi kvaliteedile sageli suurem kui ühegi teise infosüsteemi arendustegevuse oma. Andmemudeli kvaliteet mõjutab süsteemi arenduskulusid, süsteemi paindlikkust, integratsiooni teiste süsteemidega ja süsteemi võimet rahuldada kasutaja vajadusi. [13]

Andmemudeli kvaliteet on osa tarkvara kvaliteedist, mida käsitleb standard ISO/IEC 9126. Standard koosneb neljast osast: kvaliteedi mudel (*quality model*), tarkvara käitumist hindavad (tarkvara töötamise ajal) mõõdikud (*external metrics*), tarkvara sisemisi staatilisi omadusi hindavad mõõdikud (*internal metrics*) ja mõõdikud, mis hindavad tarkvara kasutaja soovidele ja vajadustele vastavust (*quality in use metrics*). Standardi esimeses osas on välja toodud tarkvara kvaliteedi karakteristikud (näitajad). Kõige kõrgema taseme karakteristikud, mis jagunevad omakorda alamkarakteristikuteks, on järgmised:

- Funktsionaalsus (*Functionality*) – näitab seda kui hästi tarkvara täidab funktsionaalseid nõudeid.

- Töökindlus (*Reliability*) – näitab seda kui hästi tarkvara säilitab jõudluse mingites konkreetsetes tingimustes.

- Kasutatavus (*Usability*) – näitab seda kui lihtne on tarkvara mõista, kasutama õppida ja kui meeldiv selle kasutamine on.

- Efektiivsus (*Efficiency*) – näitab seda kui hästi on tarkvara võimeline tagama nõutud jõudlust suhtes kasutatavate ressurssidega.

- Hooldatavus (*Maintainability*) – näitab seda kui lihtne on tarkvara hooldada.

- Teisaldatavus (*Portability*) – näitab seda kui lihtne on tarkvara viia ühest keskkonnast teise.

ISO standard annab ettevõtetele raamistiku kvaliteedi mudeli loomiseks. [15]

Selleks, et määratleda, milline on kvaliteetne andmemudel, peab esmalt küsima: „Mis on andmemudeli eesmärk?“[16]. Andmemudelitega on seotud paljud süsteemi osapooled ja erinevate osapoolte jaoks on kvaliteedi faktorid erinevad. Süsteemi disain sõltub kõigi vajalike osapoolte osalusest disainimise protsessis ja nende rahulolust sellega. Võtmerollid andmete modelleerimise protsessis on järgmised.

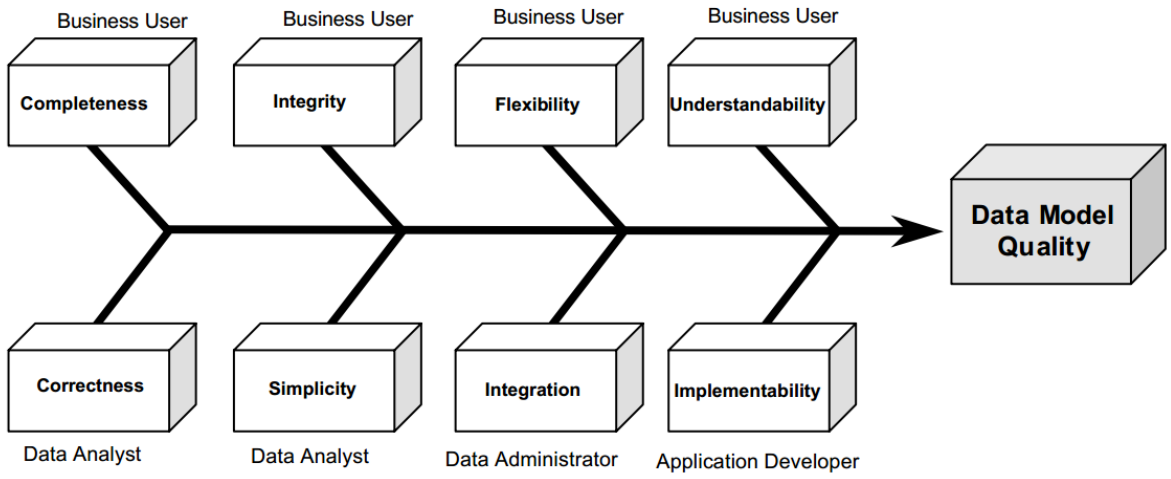
- Ärikasutaja (*Business user)* – nende ärinõuded peavad olema andmemudelis kajastatud.

- Andmete modelleerija/analüütik (*Data analyst*) – koostab erineva taseme andmemudelid. Roll võib sõltuvalt projektist või ettevõttest jaguneda veel alamrollideks, mis on vastutavad erineva taseme andmemudelite eest (kontseptuaalne, loogiline ja füüsiline).

- Andmete administraator (*Data administrator*) – vastutab selle eest, et oleks tagatud andmemudeli kooskõla ettevõtte ülejäänud andmetega.

- Tarkvaraarendaja (*Application developer*) – vastutab andmemudeli realiseerimise eest.

Igat rolli võib täita mitu inimest ja üks indiviid võib omada mitut rolli. Joonisel 3 on näha andmemudeli kvaliteedi näitajad ning iga näitaja juurde on lisatud ka osapool, keda antud näitaja enim mõjutab. [13]



Joonis 2 Andmemudeli kvaliteedi näitajad [13]

Andmemudeli kvaliteedi näitajad võivad olla aluseks andmemudelite hindamisele ja alternatiivide võrdlemisele. Kõik andmemudeli kvaliteedi näitajad on seotud tarkvara kvaliteedi karakteristikutega. Kvaliteedi näitajate definitsioonid Daniel L. Moody töös „*Measuring the Quality of Data Models*“[14] on järgmised:

- Korrektsus (*Correctness*) näitab andmemudeli vastavust modelleerimistehnika reeglitele. See sisaldab modelleerimise konventsioone ja reegleid.
- Täielikkus (*Completeness*) näitab andmemudeli vastavust kasutaja nõuetele (ärinõuetele).
- Terviklikkus (*Integrity*) näitab andmemudeli vastavust ärireeglitele ja äriprotsessidele.
- Arusaadavus/loetavus (*Understandability*) näitab seda, kui lihtsalt on andmemudel mõistetav.
- Lihtsus (*Simplicity*) iseloomustab mudeli keerukust (komponentide arv).
- Paindlikkus (*Flexibility*) iseloomustab seda kui kergelt tuleb andmemudel toime äri või regulatiivsete muudatustega.

- Integratsioon (Integration) on defineeritud kui andmemudeli kooskõlalisus organisatsiooni ülejäänud andmestikuga.

- Teostatavus (*Implementabiliy*) näitab seda kui lihtne on andmemudelit teostada lähtudes projekti eelarve-, aja- ja tehnoloogiapiirangutest.

Järgnevalt toon tabeli kujul välja (Tabel 2) enda nägemuse, kuidas Moody ja Shanks-i andmemudeli kvaliteedi näitajad suhestuvad ISO standardi kvaliteedi karakteristikutega. Maatriksi veerud tähistavad tarkvara kvaliteedi karakteristikuid ning read andmemudelite kvaliteedi näitajaid. Märge tabeli lahtris näitab, et vastav andmemudeli omadus mõjutab vastavat tarkvara omadust. Seejärel selgitan, kuidas andmemudeli kvaliteedi näitajad vastavaid tarkvara kvaliteedi karakteristikuid mõjutavad.

Tabel 2. Tarkvara kvaliteedi ja andmemudeli kvaliteedi karakteristikute maatriks

Tarkvara kvaliteedi karakteristik --- Andmemudelite kvaliteedi näitaja	**Funktsionaalsus**	**Töökindlus**	**Kasutatavus**	**Efektiivsus**	**Hooldatavus**	**Teisaldatavus**
Korrektsus		✓		✓		
Täielikkus	✓				✓	✓
Terviklikkus	✓		✓			
Arusaadavus			✓		✓	✓
Lihtsus					✓	
Paindlikkus					✓	✓
Integratsioon		✓	✓		✓	
Teostatavus	✓					

Andmemudeli korrektsus on otseselt seotud süsteemi töökindluse ja efektiivsusega. Kui andmemudel ei ole korrektne, võivad tekkida tõrked andmebaasi kasutamisel ja andmebaasi

tarbiva rakenduse kasutamisel. Andmemudeli täielikkus on seotud tarkvara funktsionaalsuse parameetriga, kuna täielikkus näitab vastavust ärinõuetele ja ärinõuded mõjutavad tarkvara funktsionaalsust. Samuti mõjutab täielikkus tarkvara hooldatavust ja teisaldatavust. Kui andmemudel ei ole täielik, on sellest raskem aru saada, mistõttu on süsteemi raskem hooldada ja teisaldada. Andmemudeli terviklikkus kajastab mudeli kooskõla ärireeglite ja –protsessidega, mis on aluseks tarkvara funktsionaalsuse arendamisel. Kui andmemudel ei toeta kõiki äriprotsesse, siis ei saa seda teha ka kõiki protsesse toetavat tarkvara. Arusaadavus, lihtsus ja paindlikkus mõjutavad kõik süsteemi hooldatavust. Kui andmemudel ja andmebaas on mitte-arusaadavad ja keerukad, siis võib olla ka sellele toetuvat süsteemi raske mõista ja kasutada. Süsteemi mõistmata on väga raske seda hooldada või edasi arendada ning teisaldada. Süsteemi hooldamise alla käib ka süsteemi muutmine. Andmemudeli ja andmebaasi paindlikkus on otseselt seotud sellega, kui lihtne on süsteemi muuta. Teise keskkonda viimisel on mõnikord vaja teha süsteemis muudatusi, seega mõjutab andmemudeli paindlikkus tarkvara teisaldatavust. Andmemudeli integreeritus ettevõtte teiste süsteemidega mõjutab tarkvara töökindlust, kuna kui uus andmemudel ei ole korrektselt integreeritud ettevõtte teiste süsteemidega, võivad tekkida andmete konfliktid ja sellest tulenevalt tõrked ka tarkvara töös. Samuti mõjutab integreeritus tarkvarast arusaamist, mis omakorda mõjutab tarkvara kasutatavust ja hooldatavust. Andmemudeli teostatavus mõjutab tarkvara funktsionaalsust seeläbi, et kui andmemudel ei ole etteantud raha, aja ja muude piirangute tõttu teostatav või on teostatav ainult osaliselt või mingite piirangutega, siis see omab mõju ka tarkvara funktsionaalsusele.

1.3 Andmemudelite kontrollimine

Andmemudelite kontrollimine on oluline osa andmebaasi loomise protsessist. Kontrollida tuleks nii kontseptuaalset, loogilist kui ka füüsilist andmemudelit. CASE vahendite arenguga on tekkinud võimalus mudeleid *automaatselt* kontrollida. Sellist võimalust pakub näiteks Oracle SQL Developer Data Modeler. Kuna agiilsetes tarkvaraprojektides tihti ei kasutata kontseptuaalse andmemudeli tegemiseks CASE vahendit ja loogilise andmemudeli tegemine jäetakse üldse vahele, siis sellistes projektides on võimalik automaatselt kontrollida vaid füüsilist andmemudelit (eelduse aluseks on asjaolu, et tegemist on vähemalt andmete modelleerimise vaates mudelipõhise arendusega).

Andmemudelit on soovitatav kontrollida võimalikult tihti, kuid eriti otstarbekas on seda teha enne kui mudeli põhjal midagi genereeritakse. Näiteks enne loogilisest andmemudelist füüsilise mudeli genereerimist ja enne füüsilisest andmemudelist andmebaasi loomist või füüsilise andmemudeli ja andmebaasi sünkroniseerimist. MDD puhul genereeritakse mudelitest koodi. Testimine ja paranduste tegemine on kiirem ja odavam tarkvaraarenduse protsessi võimalikult varases faasis. Seega tuleks testida ja teha parandused juba otse mudelites, enne, kui neist uus tulem genereeritakse.

Väga paljud andmemudelitele seatavad reeglid sõltuvad ettevõttes kehtivatest modelleerimise standarditest ja konkreetsest projektist. Mõned reeglid sõltuvad andmebaasisüsteemi või selle aluseks oleva andmemudeli piirangutest. Lisaks on olemas disainivigu, mida ei ole võimalik automaatse andmemudeli kontrollimisega tuvastada või on vea tuvastamiseks vajalik ka modelleeritavates andmestruktuurides olevate andmete olemasolu. Samas on olemas hulk reegleid, mida saaks rakendada andmemudelite automaatseks kontrollimiseks. Töö üheks eesmärgiks ongi selliste reeglite kaardistamine ja kategoriseerimine.

1.3.1 Andmemudelite kontrollreeglite kaardistamine

Andmemudelite automaatse kontrollimise reeglite kaardistamisel võtan aluseks Oracle SQL Developer Data Modeler-is olemasolevad kontrollreeglid ja vajadusel lisan reegleid, mida minu arvates, lähtuvalt isiklikust kogemusest, veel kontrollida tuleks. Järgnevalt loetlen reeglid, mida võiks automaatselt kontrollida. Iga reegli kohta lisan kontrollimise vajaduse põhjenduse ja seose andmemudeli kvaliteedi faktoriga. Andmemudeli kvaliteedi faktoritest kirjutasin peatükis 1.2 Andmemudeli kvaliteet. Käesolevas töös ei vaadelda andmemudelite kooskõla kontrolli teiste mudelitega, sest mitmed vaadeldavatest CASE vahenditest on mõeldud spetsiifiliselt andmete modelleerimiseks ning süsteemi teistes vaadetes tuleb kasutada mõnda muud modelleerimisvahendit. Erinevate mudelite kooskõla kontrollimiseks tuleks kõik mudelid viia ühte ja samasse keskkonda. Selliste võimaluste uurimine oleks mahult juba eraldi lõputöö teema.

Oracle SQL Developer Data Modeler disaini reeglid:

- **Atribuutide/veergude olemasolu olemitüübil/tabelil/vaatel** – SQL ei luba ilma veergudeta tabeleid ning olemitüübid kirjeldatakse selleks, et nende alusel hiljem

tabelid luua. Seega on ka olemitüüpide atribuutide olemasolu vajalik. Reegliga seotud kvaliteedifakor on **korrektsus**.

- **Iga olemitüüp/tabel on seotud mõne teise olemitüübiga/tabeliga** – Andmemudelis peaksid kõik olemitüübid või tabelid olema *üldjuhul* omavahel seotud. Tegemist ei kriitilise veaga, mis takistaks andmebaasi loomist, kuid selliste olukordade puhul tuleks olla tähelepanelik ja mõelda, kuidas see eraldiseisev olemitüüp või tabel mõne teise olemitüübi või tabeliga seotud on. Seose puudumine võib viidata sellele, et mingid andmed on mudelist puudu [17]. Reegliga seotud kvaliteedifaktorid on **korrektsus**, **täielikkus** ja **arusaadavus**.

- **(Primaar)võtme olemasolu igal baastabelil** – (Primaar)võti on veergude hulk kuhu kuulub üks või rohkem veergu. Primaarvõtme väärtus on tabeli rea unikaalne tunnus. See on vajalik selleks, et SQL päringu kaudu ühte konkreetset rida kätte saada ning ridu üksteisest eristada. Andmebaasi skeemi moodustavates baastabelites primaarvõtme deklareerimine tagab, et selles tabelis ei saa olla korduvaid ridu. Tabelid, millele viidatakse teistes tabelites, peavad omama võtit (deklareeritud PRIMARY KEY või UNIQUE kitsenduse abil), et teised tabelid saaksid sellesse kuuluvatele veergudele oma välisvõtmega viidata. Primaarvõtmeks valitakse üks tabeli kandidaatvõtmetest. Ülejäänud kandidaatvõtmed on alternatiivvõtmed ja jõustatakse UNIQUE ja NOT NULL kitsendusega. Mõnikord võib olla primaarvõtme valikut raske teha (alternatiivid on samaväärsed) kuid selle tegemine on levinud praktika. Võtmete deklareerimine aitab ka lisaks jõustada erinevaid ärireegleid. Reegliga seotud kvaliteedifaktorid on **korrektsus, terviklikkus** ja **arusaadavus**.

- **Kõik tabelite ja veergude nimed järgivad nimetamisstandardeid** – On levinud mõned andmebaasiobjektide nimetamise tavad, kuid nimetamisstandardid on tavaliselt defineeritud ettevõtte siseselt. Näiteks kas ja milliseid prefikseid kasutada tabelinimedes, suur- ja väiketeähted kasutamine jne. Reegliga seotud kvaliteedifaktorid on **korrektsus**, **arusaadavus** ja **integratsioon**.

- **Nimed ei ületa maksimaalset lubatud pikkust** – Kui füüsilise andmemudeli objektide (tabelite, vaadete, veergude, indeksite jne) nimed on liiga pikad, siis füüsilisest mudelist andmebaasi loomine ebaõnnestub. Erinevate andmebaasisüsteemide puhul on see

erinev. Näiteks Oracle puhul ei tohi enamike objektide nimede pikkus ületada 30 baiti. Reegliga seotud kvaliteedifaktor on **korrektsus**.

- **Tabelite, vaadete ja veergude ja muude andmebaasiobjektide nimed ei sisalda ebakorrektseid sümboleid** – Ebakorrektsed märgid sõltuvad kasutatavast andmebaasisüsteemist ja ka ettevõttes kehtestatud nimetamisstandardist. Kui nimetamisel kasutada märke, mida ei toeta andmebaasisüsteem, on tegemist kriitilise veaga. Kui kasutada märke, mida andmebaasisüsteem lubab, kuid ettevõttesisesed konventsioonid ja standardid mitte, on tegemist lihtsalt halva praktikaga. Reegliga seotud kvaliteedifaktorid on **korrektsus** ja **integratsioon**.

- **Kommentaaride olemasolu igal tabelil, vaatel ja veerul** – Tabelite ja veergude kommentaarid on süsteemi hooldamisel ja edasiarendamisel asendamatud infoallikad. Tegemist ei ole kriitilise veaga ja ka mõne tabeli või veeru kommentaar ei pruugi olla vajalik, kuid kommentaaride olemasolu tuleks siiski kontrollida, sest enamike tabelite ja veergude puhul on nende olemasolu kasulik ja tegemist on hea tavaga. Kommentaar ei tohiks koosneda ainult tühikutest ega korrata kommenteeritava objekti nime. Reegliga seotud kvaliteedifaktor on **arusaadavus**.

- **Kõigil olemitüüpide/baastabelite atribuutidel/veergudel on määratud andmetüüp** – Kontseptuaalses andmemudelis esitatud olemitüüpide atribuutidel peab olema andmetüüp määratletud kui sellest genereeritakse loogiline või füüsiline andmemudel. Veeru andmetüübi määratletus on andmebaasi loomisel kohustuslik. Paljud CASE vahendid ei lasegi füüsilisele andmemudelile lisada veergu ilma andmetüübi määramiseta. Reegliga seotud kvaliteedifaktorid on **korrektsus** ja **arusaadavus**.

Mida veel kontrollida tuleks.

- **Olemitüüpide/tabelite nimed on ainsuses** – See reegel ei ole absoluutne, kuid üldjuhul peaksid olemitüüpide/tabelite nimed olema ainsuses. See lihtsustab ja aitab vältida vigu võimsustike määramisel. Tegemist on üldlevinud konventsiooniga. Automaatselt seda 100%-lise kindlusega testida ei saa, kuna mitmuse lõpud on eri keeltes erinevad ja alati leidub erandeid. Näiteks inglise keeles on üldjuhul mitmuse tunnuseks s, kuid sõna *men* on samuti mitmuses kuigi tal s-i lõpus pole. Analoogseid näiteid võib tuua ka eesti keeles. Näteks sõnad keeld ja kood lõppevad d-ga, kuid sõnad pole mitmuses. Reegliga seotud kvaliteedifaktorid on **korrektsus** ja **arusaadavus**.

- **Atribuutide/veergude loogiline järjekord** – Hea tava on, et olemitüübi atribuudid või tabeli veerud on loogilises järjekorras, et olemitüüpide ja tabelite kirjeldused mudelist lihtsamini mõistetavad oleks. Unikaalsetesse identifikaatoritesse kuuluvad atribuudid / primaarvõtmesse ja alternatiivvõtmetesse kuuluvad veerud peaks olema kõige ees, seejärel välisvõtmetesse kuuluvad veerud (loogilise ja füüsilise andmemudeli puhul) ning siis ülejäänud atribuudid/veerud sisu järgi loogilises järjekorras. Seda, kas veerud on sisuliselt korrektses järjekorras, automaatselt kontrollida ei saa. Reegliga seotud kvaliteedifaktor on **arusaadavus**.

- **Mitu-mitmele seosetüüpide puudumine** – Kontseptuaalse andmemudeli puhul võib mitu-mitu seosetüüp viidata puuduvale infole. Võimalik, et mitu-mitu seosetüübi taga on eraldi olemitüüp koos oma oluliste atribuutidega [17] SQL-andmebaasis tuleb sellise seosetüübi realiseerimiseks tekitada kolmas tabel, mis „lõhub“ ühe mitu-mitmele seosetüübi kaheks üks-mitu seosetüübiks. Reegliga seotud kvaliteedifaktorid on **korrektsus** ja **täielikkus**.

- **Üks-ühele seosetüüpide puudumine** – Kui kaks olemitüüpi või tabelit on üks-ühele seoses, siis tegelikkuses võib olla nii, et kaks äriobjekti tuleks kirjeldada ühe objektina. Võib olla ka nii, et tegemist on tegelikult üks-mitu seosega, kuid modelleerija on unustanud või ei oska võimsustikku määrata.[17] Reegliga seotud kvaliteedifaktor on **korrektsus**.

- **Seosetüüpide võimsustike olemasolu** – Seoste võimsustikud on väga tähtsad olemitüüpide või tabelite omavaheliste suhete määratlemisel. Reegliga seotud kvaliteedifaktorid on **korrektsus**, **täielikkus** ja **arusaadavus**.

- **Puuduvad tsüklid seosetüüpide vahel** – Tsükkel seostes tähendab seda, et olemitüübid/tabelid on üksteisest järjestiku sõltuvad ja moodustavad silmuse. Näiteks tabel A viitab tabelile B, tabel B viitab tabelile C ja tabel C omakorda viitab tabelile A. Olemitüüpide/tabelite arv tsüklis ei ole oluline. Tsüklid võivad põhjustada vigaseid ja vastuolulisi andmeid. Reegliga seotud kvaliteedifaktor on **korrektsus**.

Autor ei väida, et see nimekiri on ammendav, kuid siia välja toodud probleemid moodustavad kindlasti olulise osa andmemudelite probleemidest ja autor on selliste probleemidega oma õppe- ja tööelu jooksul kokku puutunud.

Kui mudelist leitakse mõni nimekirjas esitatud reeglitele mittevastav koht, siis see ei tähenda *tingimata*, et tegemist on veaga. On kindlasti reegleid, mille vastu eksimine on suure tõenäosusega viga (näiteks lubatust pikemad nimed) ning on reegleid, mille vastu võib põhjenduste olemasolu korral eksida (näiteks üks-ühele seostüübi reegel). Kuid kindlasti on mistahes siin nimetatud reegli vastu eksimise korral tegemist viitega sellele, et mudel tuleks sellest kohast veelkord üle vaadata ja otsustada kas see vajab parandamist või mitte.

1.3.2 Kontrollreeglite kategoriseerimine

Kontrollreeglid jaotuvad antud töös kolme kategooriasse: kontseptuaalse andmemudeli kontrollreeglid, loogilise andmemudeli kontrollreeglid ja füüsilise andmemudeli kontrollreeglid. Kõik reeglid kuuluvad ühte kuni kolme kategooriasse. Vaata allolevat tabelit.

Tabel 3. Kontrollreeglite kategoriseerimine

	Kontseptuaalne andmemudel	Loogiline andmemudel	Füüsiline andmemudel
Atribtuutide/veergude olemasolu olemil/tabelil	✓	✓	✓
Iga olemitüüp/tabel on seotud mõne teise olemitüübiga/tabeliga	✓	✓	✓
(Primaar)võtme olemasolu igal tabelil		✓	✓
Kõik tabelite nimed järgivad nimetamisstandardeid		✓	✓
Kõik veergude nimed järgivad nimetamisstandardeid		✓	✓
Nimed ei ületa maksimaalset lubatud pikkust			✓
Tabelinimed ei sisalda ebakorrektseid sümboleid			✓

	Kontseptuaalne andmemudel	Loogiline andmemudel	Füüsiline andmemudel
Veerunimed ei sisalda ebakorrektseid sümboleid			✓
Kommentaaride olemasolu igal tabelil			✓
Kommentaaride olemasolu veergudel			✓
Kõigi atribuutide/veergude andmetüüpide määratletus		✓	✓
Olemitüüpide/tabelite nimed on ainsuses	✓	✓	✓
Atribuutide/veergude loogiline järjekord	✓	✓	✓
Mitu-mitmele seosetüüpide puudumine	✓		
Üks-ühele seosetüüpide puudumine	✓	✓	✓
Seosetüüpide võimsustike olemasolu	✓	✓	✓
Puuduvad tsüklid seosetüüpide vahel	✓	✓	✓

2. Andmete modelleerimiseks mõeldud CASE vahendid

CASE vahendid pakuvad tarkvara arendusele automatiseeritud tuge. Termini lõi 1982. aastal tarkvarafirma Nastec Corporation of Southfield, tuues turule integreeritud graafika ja tekstiredaktori GraphiText. CASE vahendite eesmärk on vähendada tarkvaraarendusega seotud kulusid ja tõsta arendatavate süsteemide kvaliteeti. [18]

2.1 Väljavalitud CASE vahendid

Valisin välja hulga CASE vahendid, millega saab andmeid modelleerida. Uurin väljavalitud vahendite võimalusi andmemudelite automaatseks kontrollimiseks. Järgnevalt on loetletud uuritavad CASE vahendid ja igaühe kohta lühitutvustus. Osade vahenditega olen ise kokku puutunud tööl või koolis, ülejäänute kohta tunnen lihtsalt huvi, kuna olen neist kuulnud. Valiku tegemisel oli abiks ka: http://www.databaseanswers.org/modelling_tools.htm, kus on loetletud suur hulk andmete modelleerimise CASE vahendeid. Iga loetelus toodud vahendi kohta toon põhjenduse, miks antud vahendi oma töösse valisin.

- **SQL Developer Data Modeler** on suhteliselt uus graafiline andmete modelleerimise vahend. Sellega saab luua uusi ja muuta olemasolevaid andmebaase. Vahend toetab päri- ja pöördprojekteerimist (*forward engineering* ja *reverse engineering*) ning loogilise ja füüsilise andmemudeli loomist. Vahendi suurimaks miinuseks on see, et see toetab väga väheseid andmebaasisüsteeme. Lisaks Oracle andmebaasisüsteemidele veel vaid Microsoft SQL Server 2000 ja 2005 ning IBM DB2/390 ja DB2 LUW. Firmadele, mis arendavad süsteeme paljudele erinevatele ettevõtetele (nagu ka ettevõte, kus ma ise töötan), ei ole SQL Developer Data Modeler parim valik.

  SQL Developer Data Modeler on Oracle toode ja seda pakutakse tasuta. Oracle on suurkorporatsioon, mille peakontor asub Redwood City-s, Californias. Ettevõtte lõid 1977. aastal Ed Oates, Larry Ellison ja Bob Miner.

  SQL Developer Data Modeler puhul on töö algusest saadik teada, et see vahend võimaldab automaatselt andmemudeleid kontrollida ja ka ise reegleid juurde kirjutada. Kui peaks selguma, et ükski teine väljavalitud vahenditest ei toeta andmemudelite automaatset kontrollimist või see osutub liiga keeruliseks ja töömahukaks, siis on alati võimalus andmemudelite kontrollimise eksperiment läbi viia selle vahendiga ja analüüsida tulemusi.

- **Enterprise Architect (EA)** on universaalne modelleerimisvahend, mis toetab laialdasi modelleerimisvõimalusi alates äriprotsessidest ja lõpetades tarkvara disainiga. Muuhulgas on toetatud ka andmete modelleerimine kontseptuaalsest füüsilise mudelini, võimalik on nii päri- kui ka pöördprojekteerimine. Lisaks on võimalik loogilisest andmemudelist (andmebaasisüsteemist sõltumatu) automaatselt genereerida füüsiline andmemudel (andmebaasisüsteemi põhine). EA toetab paljusid erinevaid andmebaasisüsteeme, sealhulgas DB2, Firebird/InterBase, MS Access erinevad versioonid, MS SQL Server erinevad versioonid, MySQL, SQLite, Oracle, PostgreSQL ja veel mõned

  Enterprise Architect on Austraalia ettevõtte Sparx Systems toode, mille hind sõltuvalt versioonist on 99,25–624.17 eurot. Ettevõte loodi 1996. aastal Geoffrey Sparksi poolt.

  Valisin selle vahendi uurimiseks, kuna kasutan seda igapäevaselt oma töös. Samuti on juhendaja sõnul kaalutud selle vahendi kasutuselevõttu üliõpilaste õpetamiseks TTÜ Informaatikainstituudis.

- **Toad Data Modeler** on andmebaasi disaini vahend uute andmebaaside loomiseks või olemasolevate haldamiseks ja dokumenteerimiseks. Nagu kõik kaasaegsed CASE vahendid, toetab ka Toad Data Modeler nii päri- kui ka pöördprojekteerimist. Vahendiga saab teha loogilisi ja füüsilisi andmemudeleid. Toad Data Modeler toetab paljusid andmebaasisüsteeme, sealhulgas Oracle, DB2, SQL Server, Sybase, MySQL, PostreSQL, Ingres, MS Access.

  2004. aastal turule tulnud vahendi esialgne nimetus oli "CASE Studio 2", kuid nimi muutus kui vahendi uueks omanikuks sai 2006. aastal Quest Software. Alates 2012. aasta septembrist kuulub Toad Data Modeler firmale Dell. Toad Data Modeler-i hind on 479.65 eurot. Ettevõte Dell loodi 1984. aastal ja selle peakontor asub Round Rockis, Texases.

  Valisin selle vahendi uurimiseks, kuna töö juures kasutavad osad kolleegid rakendust TOAD (Tool for Oracle Application Developers). TOAD Data Modeler-i heaks omaduseks on paljude andmebaasisüsteemide tugi.

- **Rational Rose** on UML-l baseeruv modelleerimise tarkvara, mis on juba üsna eakas (kasutatav versioon 7.0 pärineb aastast 2010). Sarnaselt Enterprise Architectile pakub

ka Rational Rose võimalusi modelleerimiseks, sealhulgas andmete modelleerimiseks. Ka Rational Rose-s on toetatud nii päri- kui ka pöördprojekteerimine ja kontseptuaalsest andmemudelist füüsilise andmemudeli genereerimine erinevate andmebaasisüsteemide jaoks.

Rational Rose kuulub USA ettevõttele IBM, mis loodi juba 1911 aastal Charles Ranlett Flint-i ja Thomas J. Watson-i poolt. Esialgselt kandis firma nime Computing Tabulating Recording Company (CTR), kuid aastal 1924 sai firma nimeks International Business Machines (IBM). IBM-i peakontor asub Armonk-s, New York-s. Rational Rose Enterprise-i hind on 4343.46 eurot.

Rational Rose-i valisin uuritavate CASE vahendite hulka, kuna seda kasutati minu õpingute perioodil (2009 kuni 2014) Tallinna Tehnikaülikoolis üliõpilaste õpetamiseks ja seal olen ka ise selle vahendiga töötanud.

- **PowerDesigner** on äriprotsesside ja andmete modelleerimise vahend, millega saab hallata nii ettevõtte kui ka selle infosüsteemi arhitektuuri. Vahendi poolt pakutav andmete modelleerimise funktsionaalsus võimaldab nii kontseptuaalse, loogilise kui ka füüsilise andmemudeli loomist ning päri- ja pöördprojekteerimist. Vahend toetab kõiki suuremaid andmebaasisüsteeme.

  PoweDesigner kuulub firmale Sybase, mis loodi aastal 1984 ja mille peakontor asub Californias. Vahendi esimene versioon tuli turule 1992. aastal kandes nime „S-Designor“. 1995. aastal nimetati vahend ümber PowerDesigner-ks. Vahendi hind sõltuvalt versioonist on 2440,00 kuni 10388,40 eurot.

  Valisin selle vahendi oma töösse, kuna tegemist paistab olevat ühe populaarseima CASE vahendiga maailmas. LinkedIn foorumi arutelu „Which one is your favorite Data Modeling tool?“ põhjal on see soosituim CASE vahend andmete modelleerimiseks.

- **CA ERwin** on vahend andmete modelleerimiseks. See võimaldab luua kontseptuaalset, loogilist ja füüsilist andmemudelit ning loogilisest mudelist füüsilise genereerida. Samuti on toetatud nii päri- kui ka pöördprojekteerimine. CA Erwin toetab järgmisi andmebaasisüsteeme: DB2, IDS (Informix), MySQL, Oracle, Progress, SAS, SQL Server, Sybase, Sybase IQ, Teradata ning lisaks ODBC andmeühenduse tehnoloogia.

CA ERwin kuulub USA firmale CA Technologies, mille lõid 1976. aastal Russell Artzt ja Charles Wang. Ettevõtte peakontor asub Islandia külas New Yorki osariigis. ERWin-i lõi 1998. aastal ettevõte Logic Works ning müüs selle ettevõttele Platinum Technology. 1999. aastal liitus Platinum Technology ettevõttega Computer Associates (CA Technologies) ning toode nimetati hiljem ümber CA ERWin-ks. CA Erwin-l on tasuta versioon (Community Edition). Standard versiooni hind on 3495 eurot ja Workgroup versiooni hind on 4895 eurot.

CA Erwin on veebilartikli http://www.databaseanswers.org/modelling_tools.htm andmeil üks turuliidritest ja see on ka põhjuseks, miks ma seda vahendit oma töös uurin.

- **ER/Studio** on andmete modelleerimise ja andmebaasi disainimise vahend. Vahend toetab väga paljusid andmebaasisüsteeme ja sellega saab luua nii kontseptuaalset, loogilist kui ka füüsilist andmemudelit, samuti loogilisest mudelist füüsilist genereerida. Selles on toetatud nii päri- kui ka pöördprojekteerimine. ER/Studio on saadaval kahes versioonis. Data Architect on suunatud kitsalt andmete modelleerimisele ja andmebaasi disainile. Enterprise versioon toetab lisaks ka äriprotsesside modelleerimist ja organisatsioonisisest tihedat koostööd.

  ER/Studio tõi 1998. aastal turule USA ettevõte Embarcadero Technologies. Ettevõte loodi 1993. aastal ja selle peakontor asub San Franciscos. ER/Studio maksab sõltuvalt versioonist ja valitud lisadest 1,263.78 kuni 8,903.58 eurot.

  Mina valisin selle vahendi oma töösse, kuna Wikipedia andmeil on see peamine konkurent PowerDesigner-le ja CA Erwin-le [19, 20].

- **DB Main** on vabavaraline andmete modelleerimise vahend. Vahend võimaldab teha nii kontseptuaalset, loogilist kui ka füüsilist andmemudelit ja koodi genereerimist. Toetatud on nii päri- kui pöördprojekteerimine.

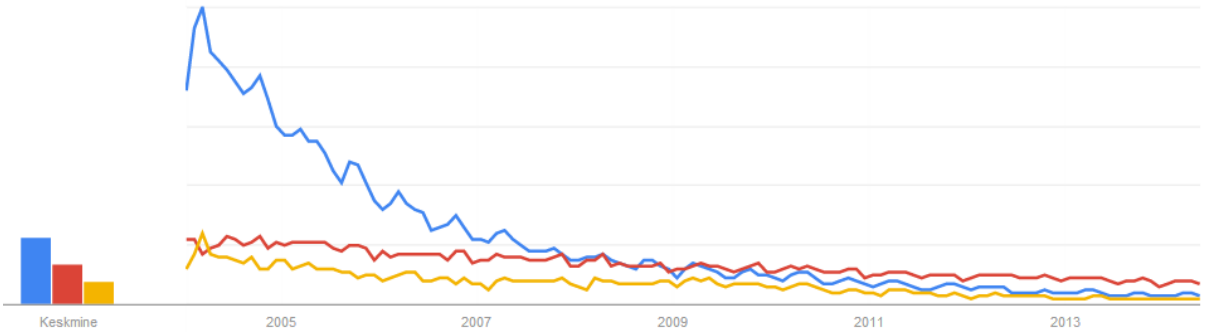
  DB Main loodi 1991. aastal alustatud projekti käigus LIBD (*Laboratoire d'ingénierie des applications de bases de données*) laboratooriumi poolt Namuri Ülikoolis Belgias. Alates 2004. aasta jaanuarist arendab ja turustab toodet Belgia ettevõte REVER S.A. Kuna toode on vabavaraline, saab igaüks selle tasuta alla laadida.

  DB Main vahendi valisin oma töösse, kuna puutusin sellega kokku kui õppisin vahetusüliõpilasena Šveitsi ülikoolis EPFL (École Polytechnique Fédérale De

Lausanne). Toona sai vahendit kasutatud vaid lihtsa olemi-suhte diagrammi tegemiseks, kuid mulle pakub huvi, mis funktsionaalsust see vahend veel pakub.

Uurisin *Google Trends* abil ka väljavalitud CASE vahendite populaarsust Google otsingus aastatel 2004-2014. Graafiku vertikaaltelg näitab otsinguhuvi ja graafiku kõrgeima tipu suhet. Google Trends on praegu *beta* versioonina ja võrdlusesse saab panna maksimaalselt viis otsinguterminit. Kindlasti ei tohiks neid graafikuid võtta puhta tõena iga konkreetse CASE vahendi reaalse populaarsuse kohta. Erinevad CASE vahendid on populaarsed eri piirkondades ja ka otsingumootorite kasutamine võib olla ajas ja piirkonniti väga erinev.

Google Trends-i kohaselt on praegu kõige otsitumad CASE vahendid Enterprise Architect ja PowerDesigner, kuid kuni 2008. aastani oli Rational Rose kõige sagedasem otsingutermin (vaata Joonis 3).

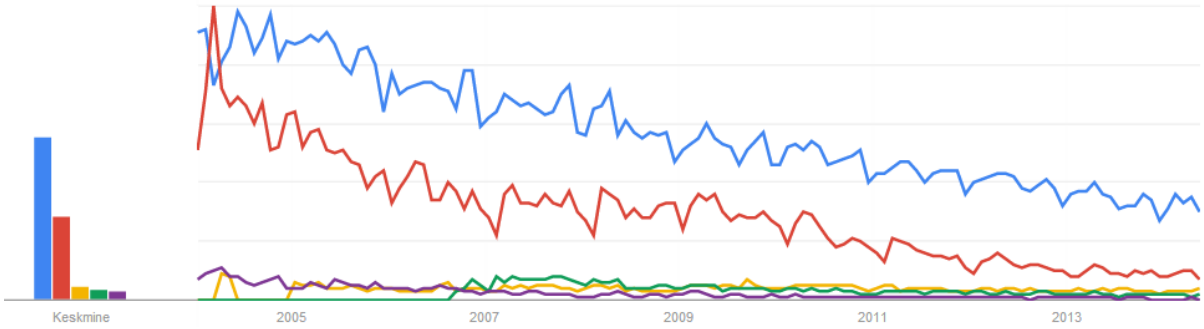


Joonis 3 Rational Rose-i (sinine), PowerDesigner-i (punane) ja Enterprise Architect-i (kollane) otsingu trendid.

Kuna Rational Rose-i otsinguhuvi kõrgeim tipp oli teistest nii palju kõrgemal, eemaldasin selle graafikult, et ka vähempopulaarsete CASE vahendite otsingupopulaarsuse trendid pisut jälgitavamad oleks. Lisasin graafikule CA Erwin-i, Toad Data Modeler-i ja ER/studio. PowerDesigner-i ja Enterprise Architect-i jätsin võrdluse huvides graafikule alles (vaata Joonis 4). Graafikult on selgelt näha, et väljavalitud vahenditest on Enterprise Architect ja PowerDesigner, vähemalt väljavalitud vahendite hulgast, selged huvi osas liidrid, kuid nende otsingusagedus on aasta-aastalt aina vähenenud, mis võib viidata CASE vahendite populaarsuse vähenemisele.

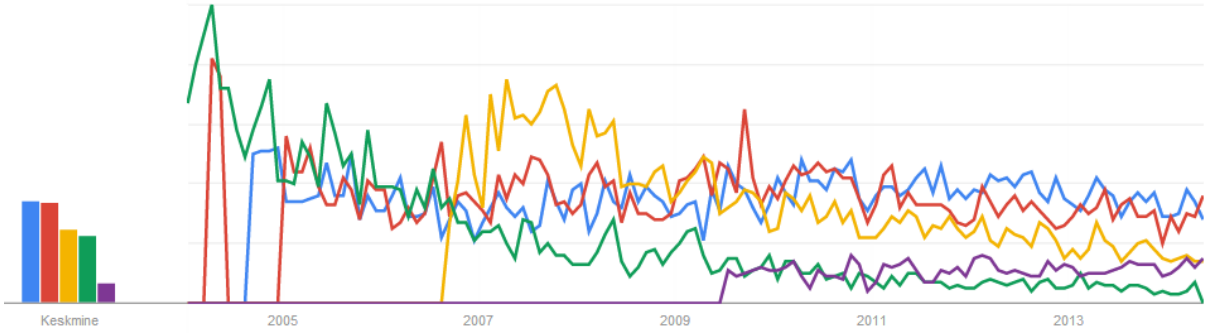
Võib spekuleerida, et kui CASE vahend on kõigest kallihinnaline joonistusvahend, siis on loomulik, et võetakse kasutusele lihtsamad ja odavamad joonistusvahendid. Mudelite kontrolli

võimalus on üks sellistest funktsionaalsustest, mis CASE vahendeid joonistusvahenditest eristavad ja CASE vahendid tõeliselt kasulikuks muudavad.



Joonis 4 PowerDesigner-i (sinine), Enterprise Architect-i (punane), CA Erwin (kollane), Toad Data Modeler (roheline), ER/Studio (lilla) otsingu trendid.

Koostasin ka graafiku, kust kõige populaarsemad CASE vahendid on välja jäetud (Rational Rose, Enterprise Architect ja PowerDesigner) ning lisatud on kõik ülejäänud töös kasutatavad vahendid. Väljavalitud vahenditest kõige vähem on otsitud Oracle SQL Developer Data Modeler-i, mis on suures osas põhjendatav sellega, et tegemist on alles hiljuti turule tulnud vahendiga. Üllatavalt populaarne seevastu on vabavaraline projekt-vahend DB Main (vaata Joonis 5).



Joonis 5 DB Main-i (sinine), CA Erwin-i (punane), Toad Data Modeler-i (kollane), ER/Studio (roheline) ja Oracle SQL Developer Data Modeler-i (lilla) otsingu trendid.

3. Eksperiment

Selles peatükis tutvustatakse CASE vahendite andmemudelite kontrolli võimekuse hindamiseks tehtud eksperimenti ning selle tulemusi.

3.1 Eksperimendi kirjeldus

Eksperimendi eesmärgiks on uurida, kui tõhus on andmemudeli automaatne kontrollimine potentsiaalsete vigade tuvastamiseks. Selleks annan CASE vahendile ette meelega lisatud probleemsied kohti sisaldava andmemudeli ja analüüsin tulemusi. Eksperimendi viin läbi ühe väljavalitud CASE vahendiga. Võimalusel kasutaksin selleks Enterptise Architecti, sest vahend on töö juures praegu kasutusel ja andmemudelite automaatse kontrollimise funktsionaalsuse kasutuselevõtt tuleks kindlasti kasuks. Esmalt uurin kas ja millised CASE vahendid võimaldavad andmemudelite automaatset kontrollimist. Minule teadaolevalt võimaldab seda kindlasti Orale SQL Developer Data Modeler. Pärast väljavalitud vahendite võrdlemist andmemudelite automaatse kontrollimise osas otsustan, millise vahendiga viin läbi eksperimendi. CASE vahendite uurimisel võrdlen neid järgmistes aspektides:

- Kas vahend toetab andmemudelite automaatset kontrollimist?
- Kui vahend toetab andmemudelite automaatset kontrollimist, siis kas kontrollreegleid on võimalik ise juurde kirjutada?
- Kui vahendil ei ole andmemudelite automaatse kontrollimise funktsionaalsust, siis kas seda on võimalik ise lisada (programmeerida CASE vahendile uus funktsionaalsus)?

Selleks, et ma kaaluksin CASE vahendi valimist eksperimendi läbi viimiseks, peab see toetama andmemudelite automaatset kontrollimist ja reeglite lisamist. Kui leidub mitu sellist vahendit, siis järgnevaks kriteeriumiks on toetatavad andmebaasisüsteemid. Kui ka selle kriteeriumi alusel jääb mitu valikut, muutuvad määravaks vahendi muud funktsionaalsused ja hind. Infot CASE vahendi andmemudeli automaatse kontrollimise võimaluste kohta otsin Internetist. Samuti paigaldan kõik vahendid endale arvutisse ja uurin võimalusi läbi katsetamise. Katsetamise all pean silmas seda, et avan CASE vahendi ja otsin rakendusest üles mudelite kontrollimise liidese. Vajadusel loon pisikese andmemudeli paari olemitüübi/tabeliga või avan mõne olemasoleva näidisprojekti. Kui vahend toetab andmemudelite automaatset kontrollimist ja ka võimalust ise reegleid juurde tekitada, siis andmemudelite automaatse kontrollimise funktsionaalsuse ise programmeerimise võimalusi ma ei uuri. Tulemused on esitatud Tabelis 4.

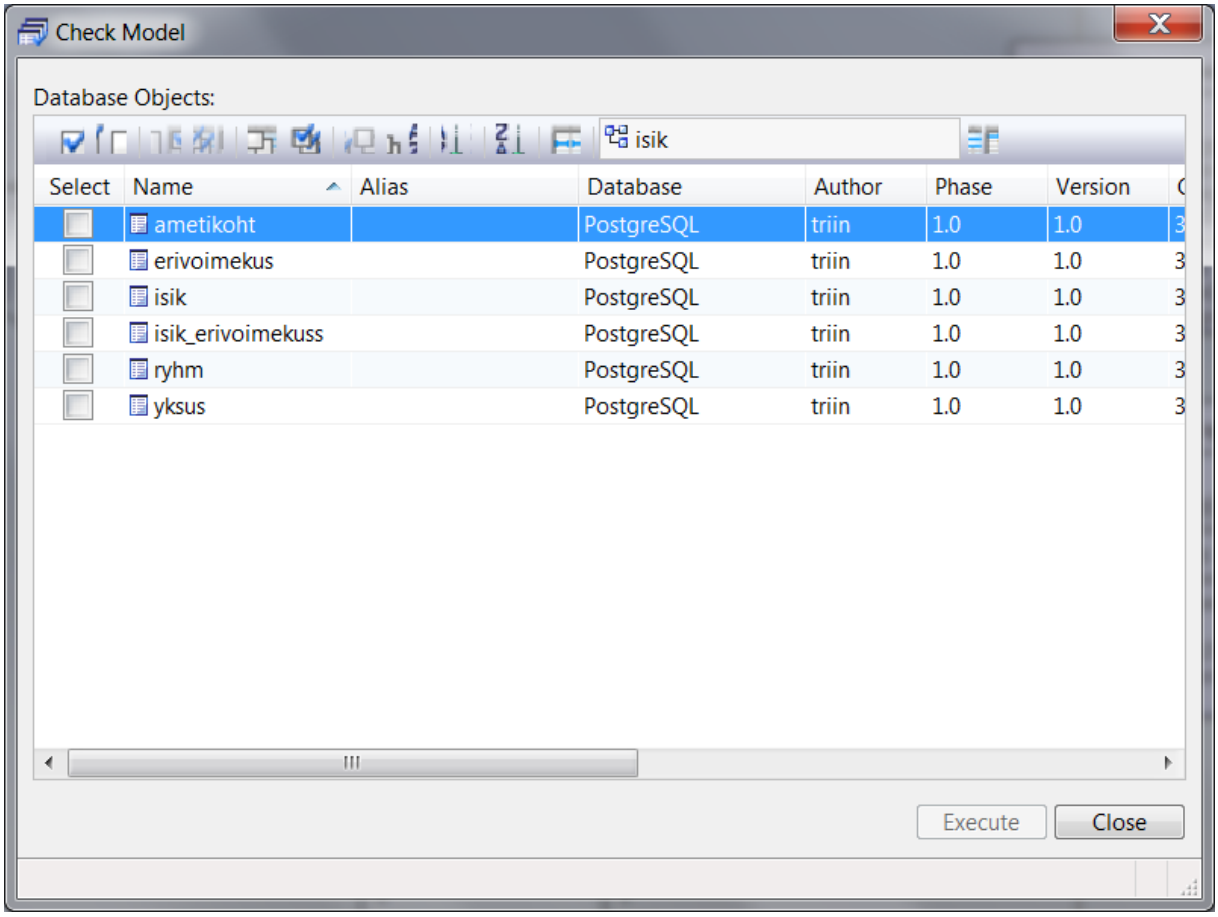
3.1.1 CASE vahendite võrdlus

Tabel 4. Andmete modelleerimiseks mõeldud CASE vahendite võrdlus andmemudelite automaatse kontrollimise seisukohalt

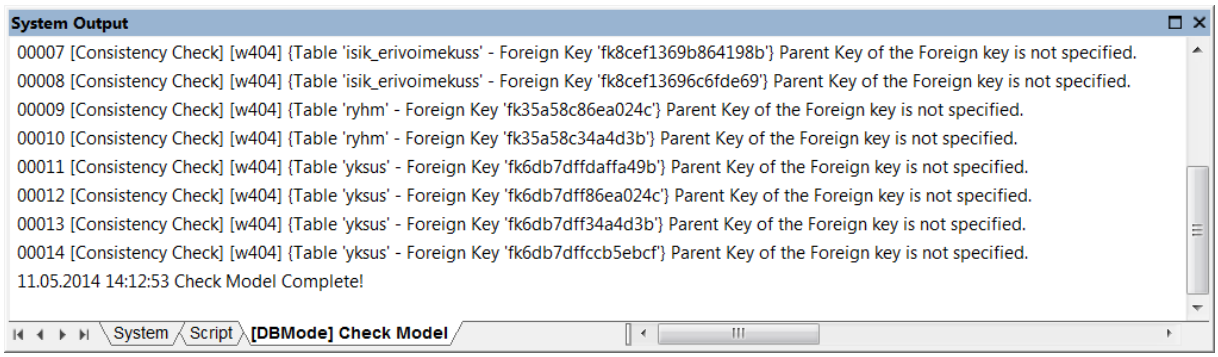
Vahendi nimi ja versioon	**Kas vahend toetab andmemudelite automaatset kontrollimist?**	**Kas kontrollreegleid on võimalik ise juurde kirjutada?**	**Kas andmemudelite automaatse kontrollimise funktsionaalsust on võimalik ise lisada?**
SQL Developer Data Modeler (versioon 4.0.1)	Jah	Jah	Eelnevast tulenevalt ei uurinud
Enterprise Architect (versioon 11, Professional)	Jah	Ei	Ei
Toad Data Modeler (versioon 5.2)	Ei (saab kontrollida nimede vastavust standarditele)	Ei (saab ise luua nimetamise standardeid, mida kontrollida)	Ei
Rational Rose (versioon 7.0.0)	Jah	Ei	Jah
PowerDesigner (versioon 16.5)	Jah	Jah	Eelnevast tulenevalt ei uurinud
CA Erwin (versioon 9.5.00.4043)	Ei	Ei	Ei
ER/Studio	Jah	Ei	Ei
DB Main (versioon 9.2.b)	Ei	Ei	Jah

Enterprise Architect-ga (EA) saab andmemudeleid kontrollida siis, kui paigaldada vahendile plugin (pistikprogramm) **DBMode.** DBMode plugina viimane versioon 3.1 nõuab EA versiooni 9 või uuem ja vähemalt Professional väljaannet. Kuna töö juures on kasutusel EA versioon 8, siis paigaldasin antud töö jaoks oma arvutisse uuema versiooni, mida saan kasutada 30 päeva

jooksul. Andmemudeli kontrollimiseks tuleb valida ülevalt menüüribalt Extensions -> DBMode -> Check Model. Avaneb dialoogiaken, kust saab valida kontrollitavad objektid (Joonis 6). DBMode on andmemudeli kontrollimise seisukohast üsna algeline. Kontrolli on küll väga lihtne läbi viia, kuid puudub kontrollitavate reeglite valiku võimalus ning mitte kusagil pole kirjas, milliste reeglite täidetust kontrollitakse. Kui mingi viga leitakse, siis lihtsalt selle kohta tuleb teade. Lisaks ei toiminud korrektselt välisvõtmete kontroll vanemast EA versioonist imporditud mudeli kontrollimisel – väidetavalt olid kõik välisvõtmed vigased (Joonis 7), kuigi nad seda ei olnud. Tegemist oli andmebaasist imporditud mudeliga ja kui välivõtmete detaile vaadata, siis seal oli kõik korras. Kui genereerisin EA vanemas versioonis mudelist SQL-i, siis ka seal olid välisvõtmed defineeritud. Kui tegin käsitsi uue mudeli, siis seal tunnistas ka mudeli kontrollimine välisvõtmed korrektseks.

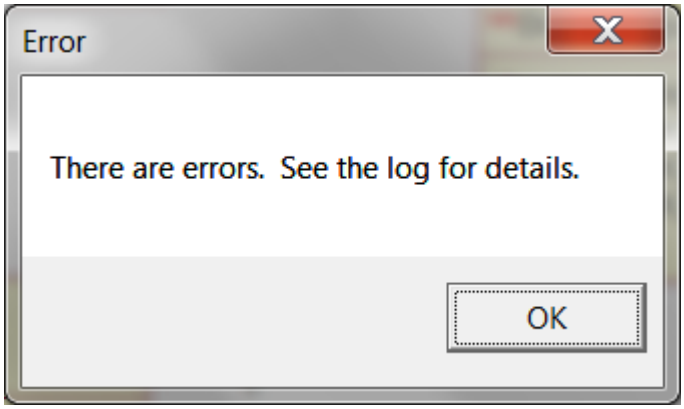


Joonis 6 Enterprise Architect Check Model

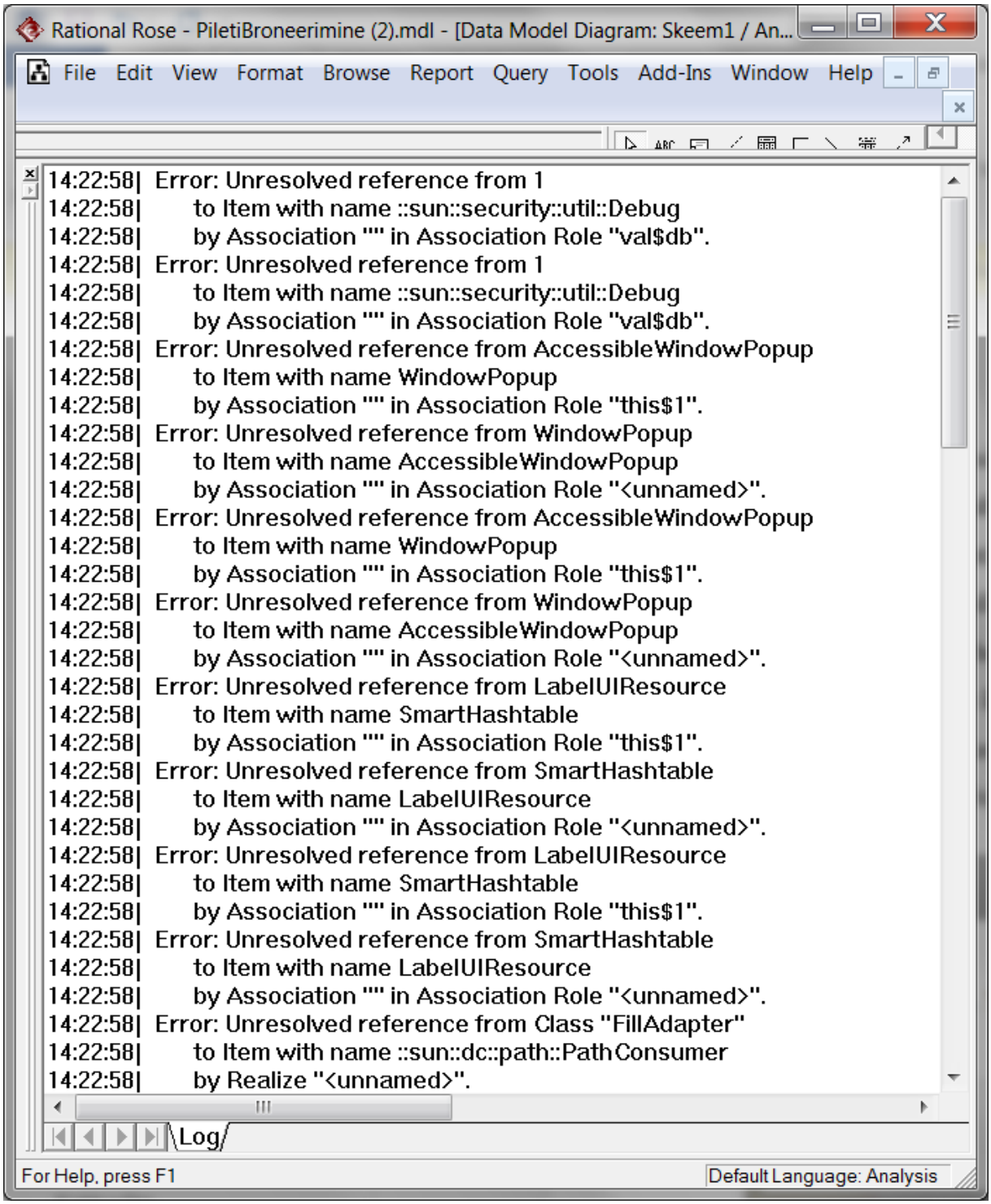


Joonis 7 Enterprise Architect andmemudeli kontrolli väljund

Rational Rose-s on sisse ehitatud mudeli kontrollimine (menüüst Tools -> Check Model), kuid ka selle vahendi puhul pole võimalik kontrollreegleid valida ega näha, milliseid reegleid kontrollitakse. Samuti ei õnnestunud mul selle kohta internetist dokumentatsiooni leida. Ainult katse-eksitus meetodil on võimalik välja selgitada, mida kontrollitakse. Katsetasin mudeli kontrollimist TTÜ „Andmebaasid II“ õppeaine raames tehtud projektis. Sain teate, et eksisteerib vigu (Joonis 8). Kui logist detaile loen, on väga raske aru saada, kus on viga ja milles see seisneb (Joonis 9). Samas on võimalik Rational Rose-i ise oma vajadustele kohandada ja edasi arendada kasutades vastavat liidest *Rose Extensibility Interface* [21].



Joonis 8 Rational Rose Check Model Errors

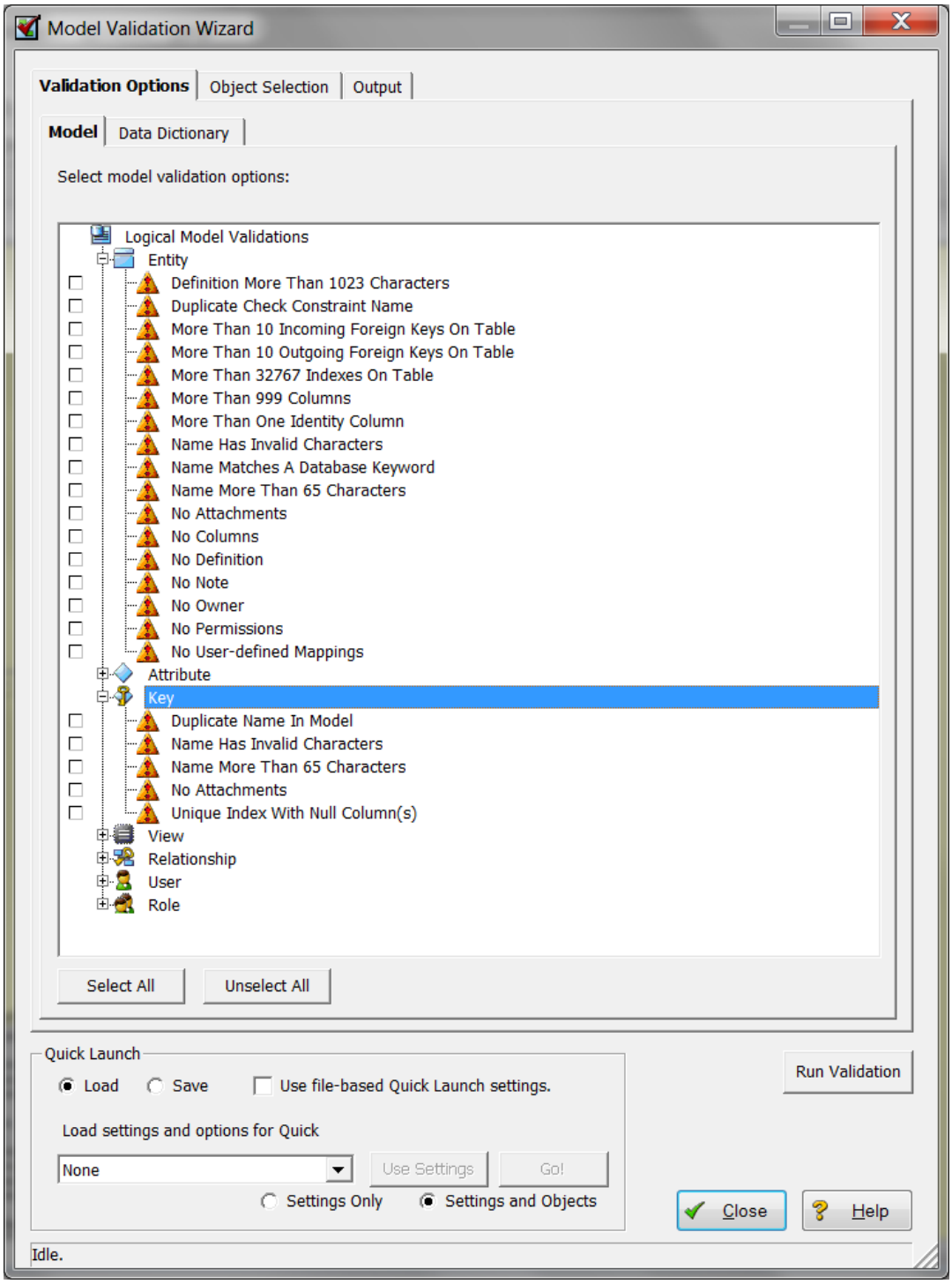


Joonis 9 Rational Rose-i mudeli kontrollimise logi

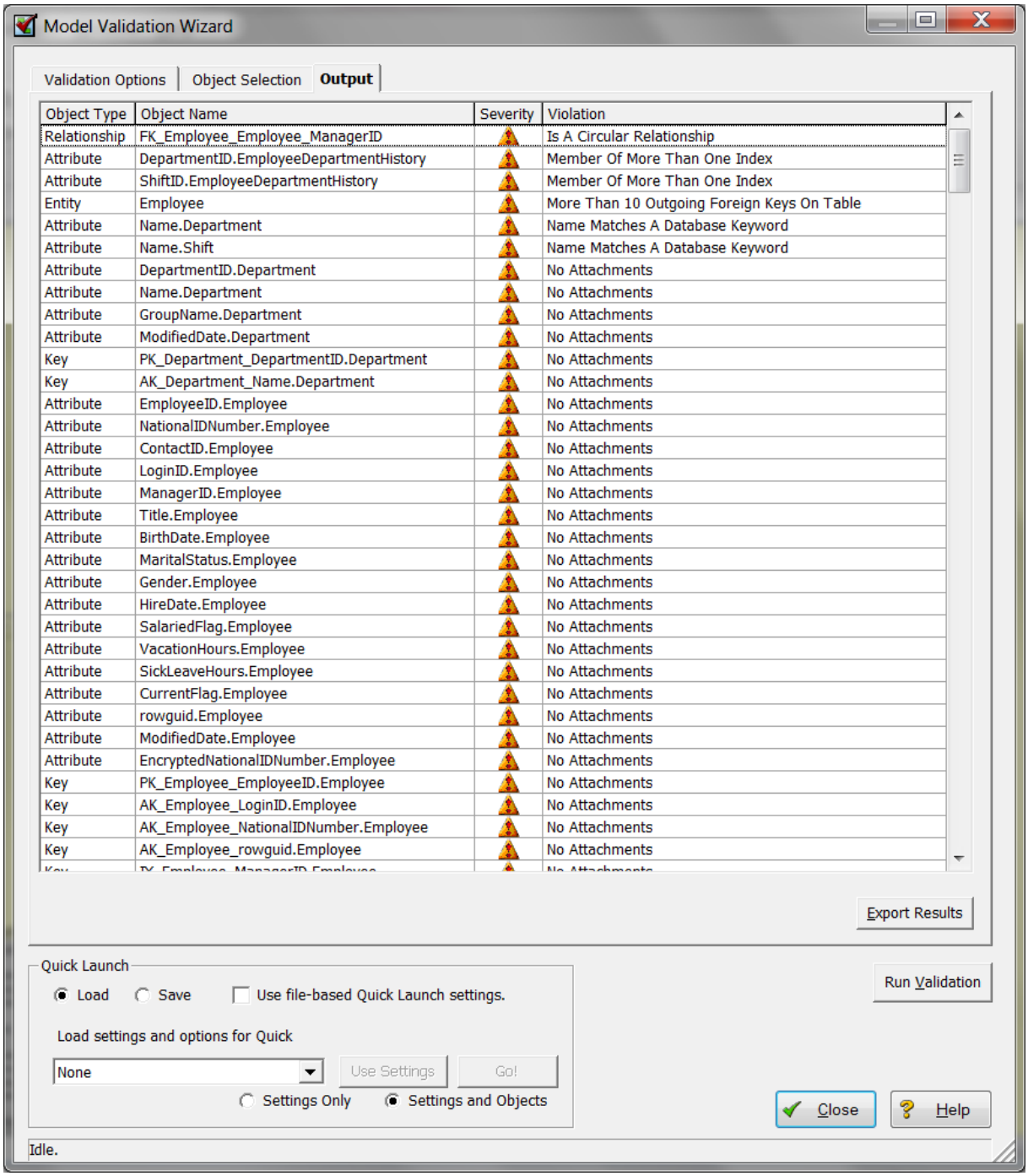
PowerDesigner-s on mudelite kontrollimise võimalus sisse ehitatud. Kui menüüst avada Tools -> Check Model, siis avaneb dialoogiaken, kust saab valida reeglid, mida kontrollida. Reeglite valik sõltub mudeli tüübist ja elementidest, mis on mudelisse lisatud. Lisaks on PowerDesigner-s olemas ka uute kontrollreeglite skriptimise võimalus. PowerDesigneri eeliseks SQL

Developer Data Modeler-i ees on see, et see toetab rohkemaid andmebaasisüsteeme ja eeldefineeritud reeglite hulk on tunduvalt suurem. Lisaks uute kontrollreeglite skriptimise võimalusele on võimalus kirjutada ka skriptid leitud vigade automaatseks parandamiseks.

ER/Studio-sse on samuti mudelite kontrollimise võimalus sisse ehitatud (Model -> Validate Model...). Sarnaselt PowerDesigner-le on eeldefineeritud reeglite hulk suur ja reeglite valik mudeli valideerimisel sõltub kontrollitavast mudelist. Vaata ekraanipilti mudeli kontrollimise dialoogiaknast (Joonis 10). Käivitasin mudeli kontrolli ER/Studio näidismudelil. Valisin välja kõik objektid ja kõik reeglid. Raport vigade kohta tuli sama dialoogiakna teisele lehele (Joonis 11). Väljundis on vead selgelt välja toodud, kuid sealt ei saa otse navigeerida probleemi allika juurde, et seda parandada. Vea asukoha peab siiski ise projektist üles otsima. Kahjuks puudub ER/Studio-s võimalus ise kontrollreegleid juurde kirjutada.



Joonis 10 ER/Studio mudeli valideerimise dialoogiaken



Joonis 11 ER/Studio mudeli kontrolli väljund

DB Main ei paku vaikimisi andmemudelite automaatse kontrollimise võimalust, kuid pakub võimaluse ise vajalikke mooduleid kirjutada või tellida. Selleks, et ise DB Main-i edasi arendada, on võimalik läbida spetsiaalne koolitus arendajatele, mis annab juhiseid, kuidas seda teha. Eeldusteks on baasteadmised andmebaasidest, modelleerimisest ning Java programmeerimiskeele tundmine. Leian, et DB Main-i kasutuslevõttu tuleks kaaluda Tallinna Tehnikaülikoolis, kuna vahend sisaldab enamikke olulisi funktsionaalsusi ja modelleerimisvõimalusi, mida pakuvad ka kommertsvahendid (kasutusjuhu diagrammid,

tegevusdiagrammid, kontseptuaalne-, loogiline- ja füüsiline andmemudel, päri- ja pöördprojekteerimine). Paraku ei võimalda vahend veel seisundidiagrammide tegemist, kuid sellise funktsionaalsuse võiks mõni üliõpilane näiteks oma lõputöö raames lisada.

Võrreldavatest vahenditest pakuvad automaatset andmemudeli kontrollimist mitmed vahendid, kuid kontrollreeglite lisamis toetavad neist vaid kaks: Oracle SQL Developer Data Modeler ja Sybase PowerDesigner. Kuna PowerDesigner toetab rohkemaid andmebaasisüsteeme ning eeldefineeritud reeglite hulk on suurem, kasutan eksperimendi läbiviimiseks seda vahendit.

3.1.2 Kontrollitav andmemudel

Andmemudeli kontrollimise eksperimendis võtan aluseks „Andmebaasid II“ ainetöös tehtud füüsilise Postgre SQL-i andmemudeli. Kuna mudel on tehtud Rational Rose-s, kasutan ära selle vahendi füüsilisest andmemudelist SQL skripti genereerimise funktsionaalsust ja teisalt PowerDesigner-i võimet genereerida füüsiline andmemudel SQL-ist (viia läbi pöördprojekteerimist). Andmemudeli importimine Rational Rose-st PowerDesigner-sse oli väga lihtne. Üldisel kõik, mis oli SQL-s kirjas, tuli probleemideta PowerDesigner-sse üle ilma, et ma midagi muutma oleks pidanud.

Siiski tekkis üks viga. Tabelite paaride *Isik-Klient* ja *Isik-Check_in_tootaja* vahele esitas PowerDesigner diagrammil 1:M seosetüübi kuigi võtmetest tulenevalt peaks seal olema 1:1 seosetüüp.

Rational Rose-ga genereeritud SQL laused on töö lisas 1 ja mudeli pilt PowerDesigner-s on lisas 2.

3.2 Eksperimendi läbiviimine

Tabelis 4 on toodud töö käigus kaardistatud reeglite vastavus PowerDesigner-s eeldefineeritd reeglitega. Vastava reegli puudumisel panen lahtrisse kriipsu. PowerDesigner-i reeglitele lisan tunnuse CDM, LDM ja/või PDM, mis näitab, mis tüüpi andmemudelile reegel on rakendatav. CDM – kontseptuaalne andmemudel; LDM – loogiline andmemudel; PDM – füüsiline andmemudel; Paraku kasutavad erinevad autorid/allikad andmemudelitest rääkides sageli

erinevat terminoloogiat ja viitavad kasutatavate terminitega samadele või erinevatele mõistetele. PowerDesigner-s kasutatakse loogilise andmemudeli puhul termineid olemitüüp (*entity*), atribuut (*attribute*) ja seosetüüp (*relationship*). Segadus erinevate tasemete andmemudelite ja neis kasutatavate objektide terminite osas on levinud probleem. Mina lähtun siiski töös eelnevalt kajastatud seisukohast, et loogilises andmemudelis ei kirjeldata mitte olemitüüpe koos nende atribuutide ja suhetega, vaid kontseptuaalsest kirjeldusest tuletatud andmestruktuure.

Tabel 5. Kontrollreeglite vastavus

Kaardistatud kontrollreeglid	**Vastav kontrollreegel PowerDesigner-s**
Atribuutide/veergude olemasolu olemitüübil/tabelil	Existence of attributes (CDM/LDM) Existence of column (PDM)
Iga olemitüüp/tabel on seotud mõne teise olemitüübiga/tabeliga	Existence of relationship (CDM/LDM) Existence of reference (PDM)
(Primaar)võtme olemasolu igal tabelil	Existence of key (PDM)
Nimed ei ületa maksimaalset lubatud pikkust	Table code maximum length (PDM) Column code maximum length (PDM)
Kõik tabelite nimed järgivad nimetamisstandardeid	Vastavaid reegleid PowerDesigner-s pole, kuid nimetamise standardeid, lubatud ja keelatud märke on võimalik määratleda (Tools -> Model Options...).
Kõik veergude nimed järgivad nimetamisstandardeid	
Tabelinimed ei sisalda ebakorrektseid sümboleid	
Veerunimed ei sisalda ebakorrektseid sümboleid	
Kommentaaride olemasolu igal tabelil	-
Kommentaaride olemasolu veergudel	-

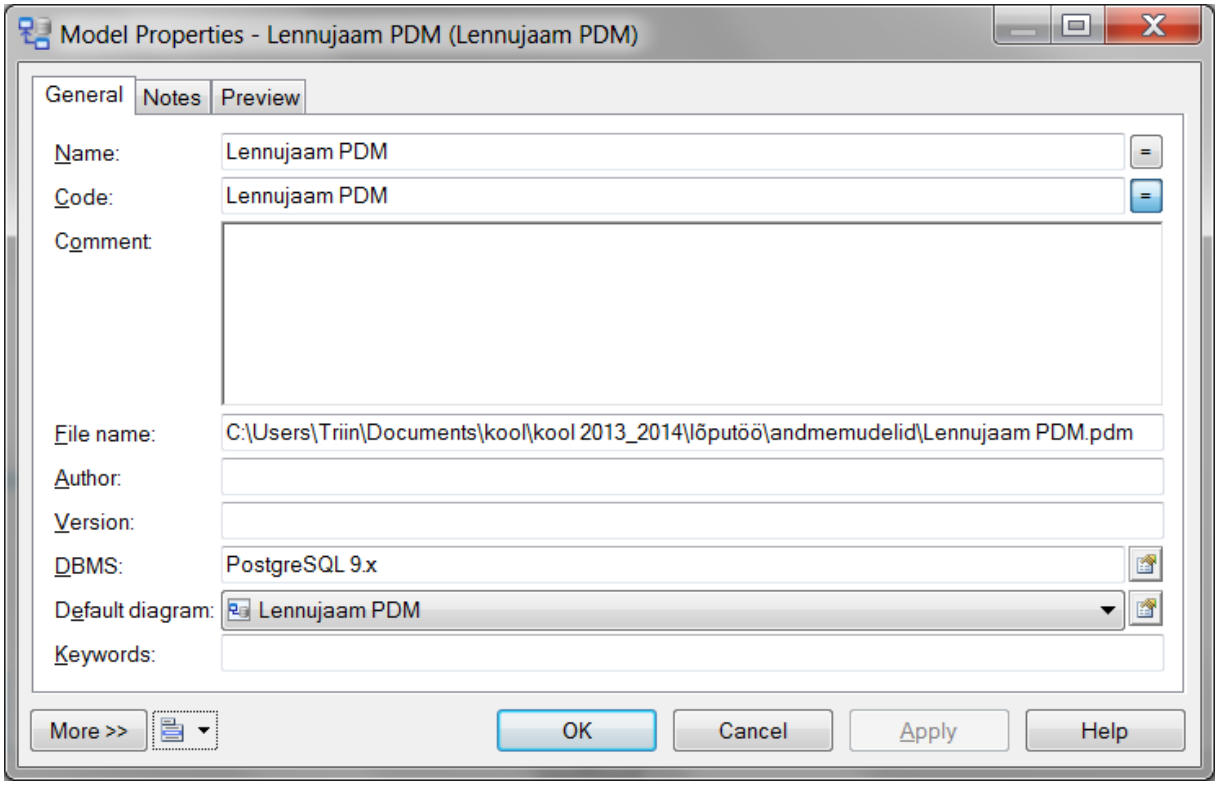
Kaardistatud kontrollreeglid	**Vastav kontrollreegel PowerDesigner-s**
Kõigi atribuutide/veergude andmetüüpide määratletus	Undefined data type (LDM/PDM)
Olemitüüpide/tabelite nimed on ainsuses	-
Atribuutide/veergude loogiline järjekord	-
Mitu-mitmele seosetüüpide puudumine	'Many-many' relationships (LDM). PowerDesigner-s pole füüsilise andmemudeli kahe tabeli vahele võimalik tekitada mitu-mitu seost, kuna lähtetabelile ei saa võimsustikku määrata (saab määrata ainult välisvõtme välja kohustuslikkuse).
Üks-ühele seosetüüpide puudumine	-
Seosetüüpide võimsustike olemasolu	-
Puuduvad tsüklid seosetüüpide vahel	Circular references (PDM) Circularity with mandatory links (CDM/LDM)

Demonstreerimaks PowerDesigner-i võimalust kontrollreegleid juurde programmeerida, programmeerin ise juurde paar kontrollreeglit, mida seal vaikimisi ei ole. Kasulik oleks lisada füüsilise andmemudeli tabeli ja veergude kommentaaride olemasolu kontroll, kuna süsteemselt on seda lihtne kontrollida, kuid manuaalselt seda teha on väga töömahukas – eriti, kui tabeleid on palju. Loogilisele andmemudelile lisan üks-ühele seosetüüpide puudumise kontrolli.

3.2.1 Reeglite lisamine PowerDesigner-s

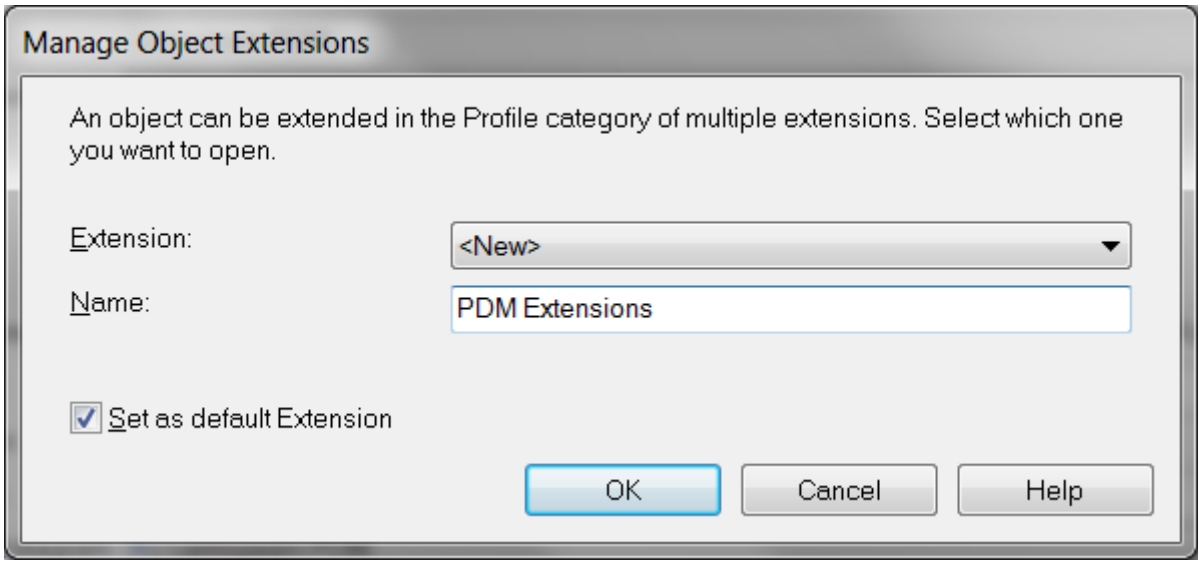
Mudeli elemente, nende süntaksit ja semantikat kirjeldab metamudel. PowerDesigner-s on metamudel jaotatud mitmesse paketti (*PdBPM - Business Process Model; PdCDM -*

Conceptual Data Model; PdEAM - Enterprise Architecture Model; PdLDM - Logical Data Model; PdOOM - Object Oriented Model; PdPDM - Physical Data Model; PdPRJ – Project jne). Metamudeli ühed peamised komponendid on metaklassid, mis esitavad mõisteid nagu olemitüüp, klass, tabel, veerg, seosetüüp jne. Metamudel on aluseks PowerDesigner-i kohandamise skriptide loomiseks. Reeglite lisamisel on suureks abiks metamudeli kirjeldus, mida saab vaadata valides menüüst *Help -> Metamodel Objects Help*. Avanevas dokumendis on kirjeldatud ära kõikide metaklasside atribuudid ja meetodid. Selleks, et PowerDesigner-s uusi kontrollreegleid salvestada, peab rakenduse avama administraatorina. Füüsilisele andmemudelile ise kontrollreeglite lisamiseks tuleb teha paremklõps metaklassil ja valida seejärel New -> Custom Check. Selleks tuleb avada andmemudel ning valida menüüribalt *Model -> Model Properties.* Avaneb dialoogiaken Model Properties. Vaata Joonis 12.

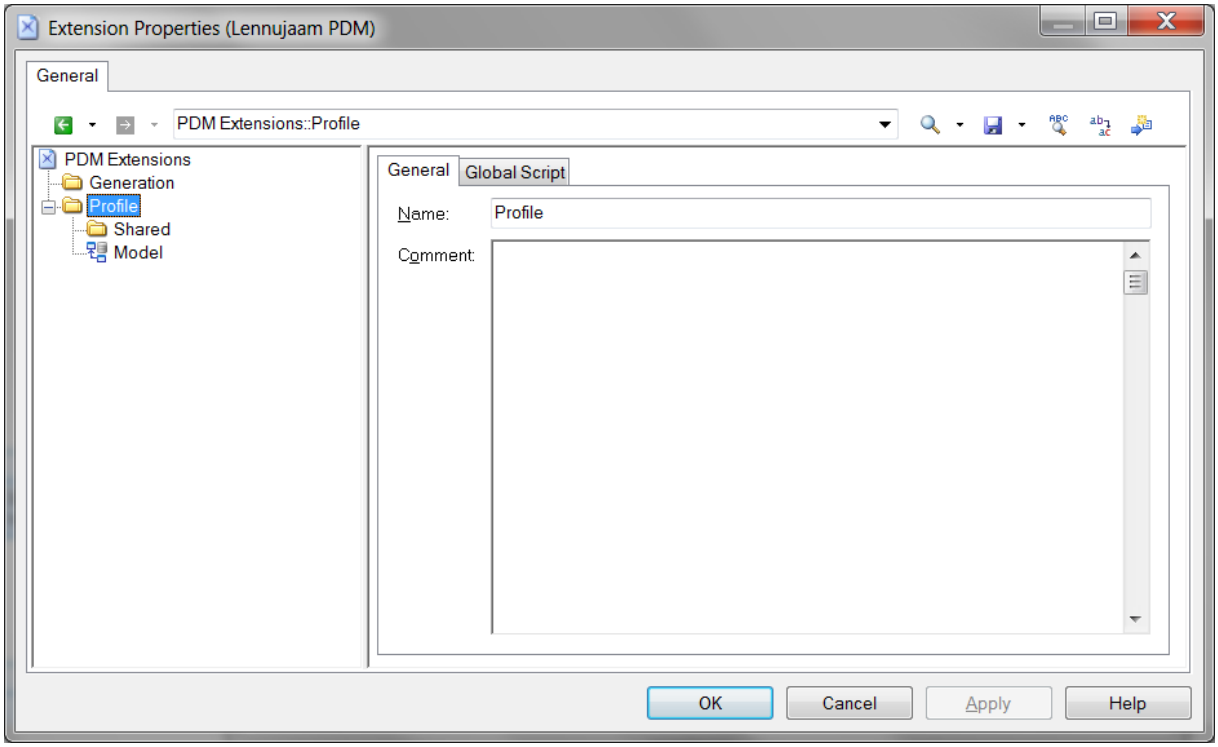


Joonis 12 Dialoogiaken DBMS Properties

Dialoogiakna all vasakul nurgas, nupu *More>>* kõrval saab avada rippmenüü ning sealt tuleb valida *Manage Object Extensions...*. Avaneb dialoogiaken Manage Object Extensions, kus saab valida laienduse ja panna sellele nime (vaata Joonis 13). Kui vajutada OK, avaneb uus dialoogiaken Extension Properties (Joonis 14) ja mudeli navigatsioonipuusse tekib uus kaust *Extensions*.

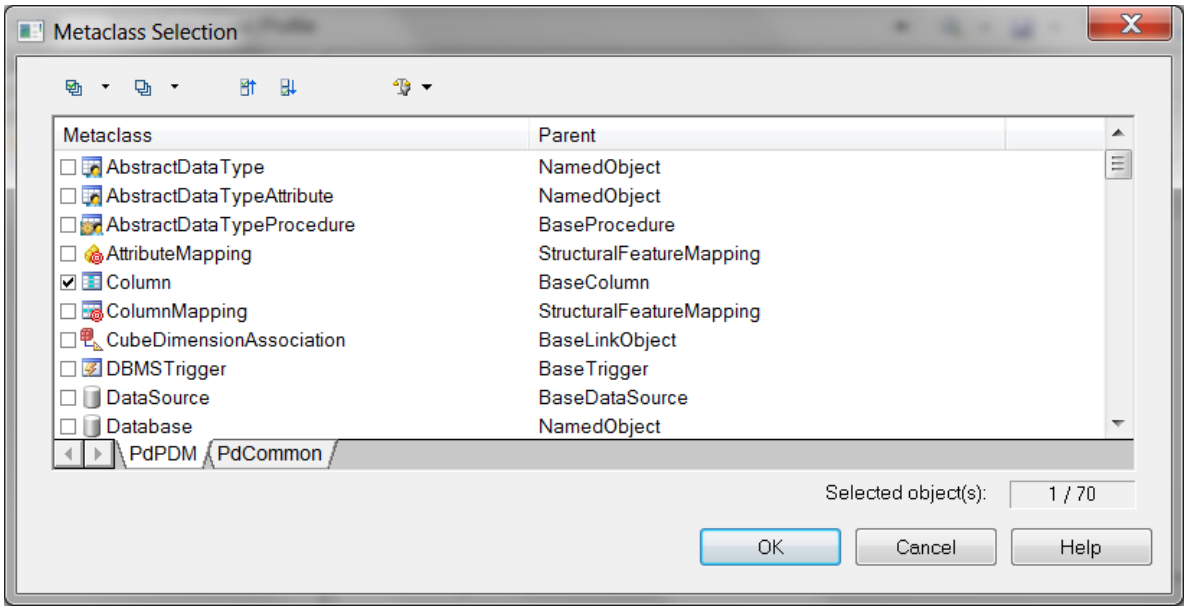


Joonis 13 Manage Object Extensions dialoogiaken

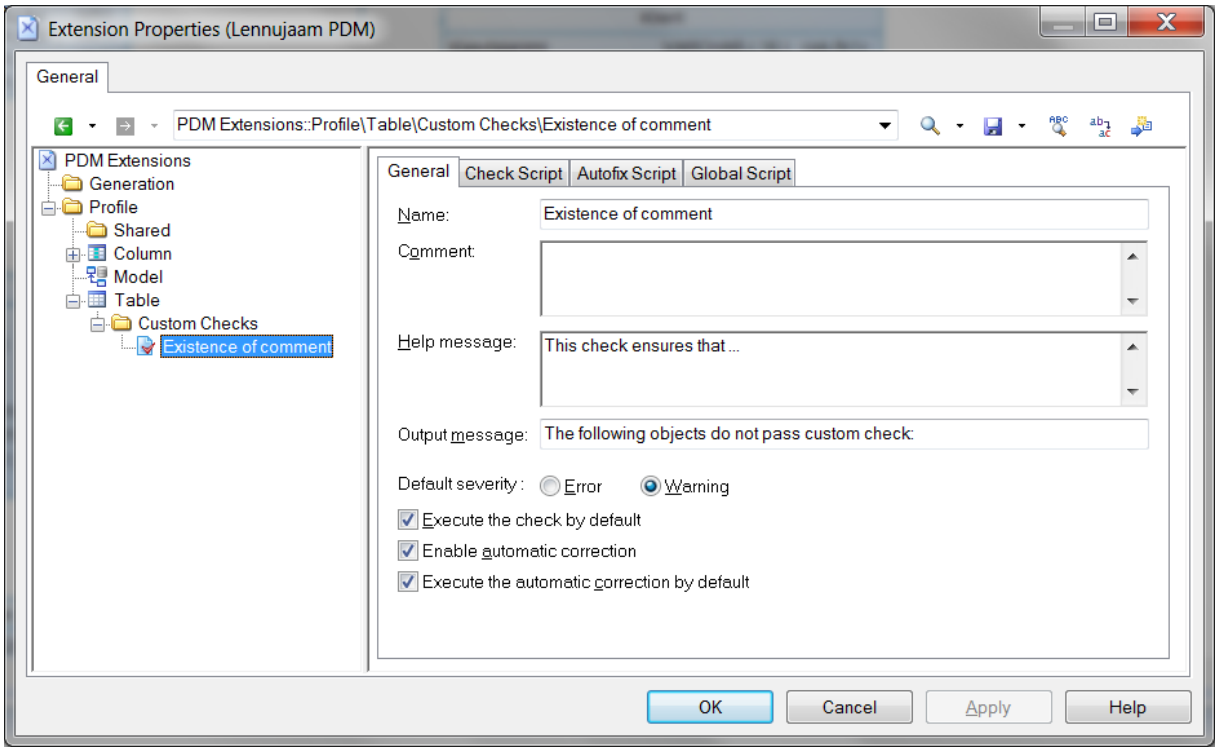


Joonis 14 Extension Properties dialoogiaken

Extension Properties dialoogiaknas teen paremklõpsu menüüpunktil *Profile*. Avaneb kohtmenüü, kust valin *Add Metaclasses...*. Avaneb dialoogiaken metaklassi valimiseks (Joonis 15). Valin metaklassid *Column* ja *Table* ning vajutan OK. Pärast seda ilmuvad uued metaklassid *Extension Propertis* aknasse *Profile* kausta alla. Teen paremklõpsu metaklassil *Table*. Avaneb kohtmenüü, kust valin *New -> Custom Check*, mille peale avaneb uue kontrollreegli loomise vorm (Joonis 16).



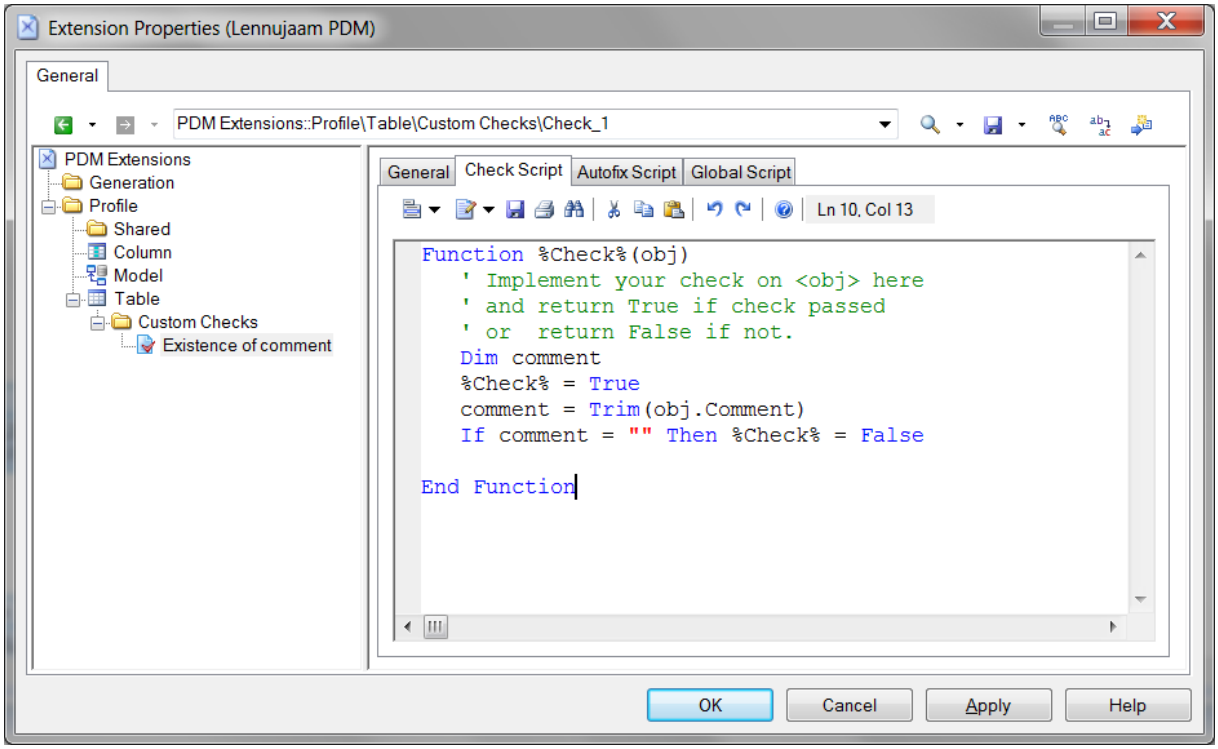
Joonis 15 Metaclass Selection dialoogiaken



Joonis 16 Custom Check

Kontrollreegli loomise vormil on 4 lehekülge. Esmalt on avatud lehekülg reegli üldandmete sisestamiseks. Joonisel 16 on näha üldandmete lehel tabeli kommentaari kontrollimise reegli üldparameetrid. Järgmine lehekülg on Check Script, kuhu saab kirjutada kontrollreegli koodi skriptimiskeeles VBScript. Sellele järgneb lehekülg Autofix Script, kuhu saab soovi korral kirjutada skripti tuvastatud probleemide automaatseks parandamiseks. Viimasel lehel *Global Script* saab defineerida staatilised muutujad ja korduvkasutatavad funktsioonid. Joonis 17

näitab reegli lisamise vormi teist lehekülge, kus on tabeli kommentaari olemasolu kontrollreegli skript. Funktsiooni sisend (obj) on tabel. Funktsioon väljastab *False*, kui sisendiks oleval tabelil pole kommentaari. Vastasel korral väljastab funktsioon *True*, mis tähendab, et kontroll on edukalt läbitud. Vältimaks seda, et kontroll läbitakse ka siis, kui tabeli kommentaariks on lisatud ainult tühikud, kasutasin funktsiooni *Trim()*, mis eemaldab teksti algusest ja lõpust kõik tühikud.

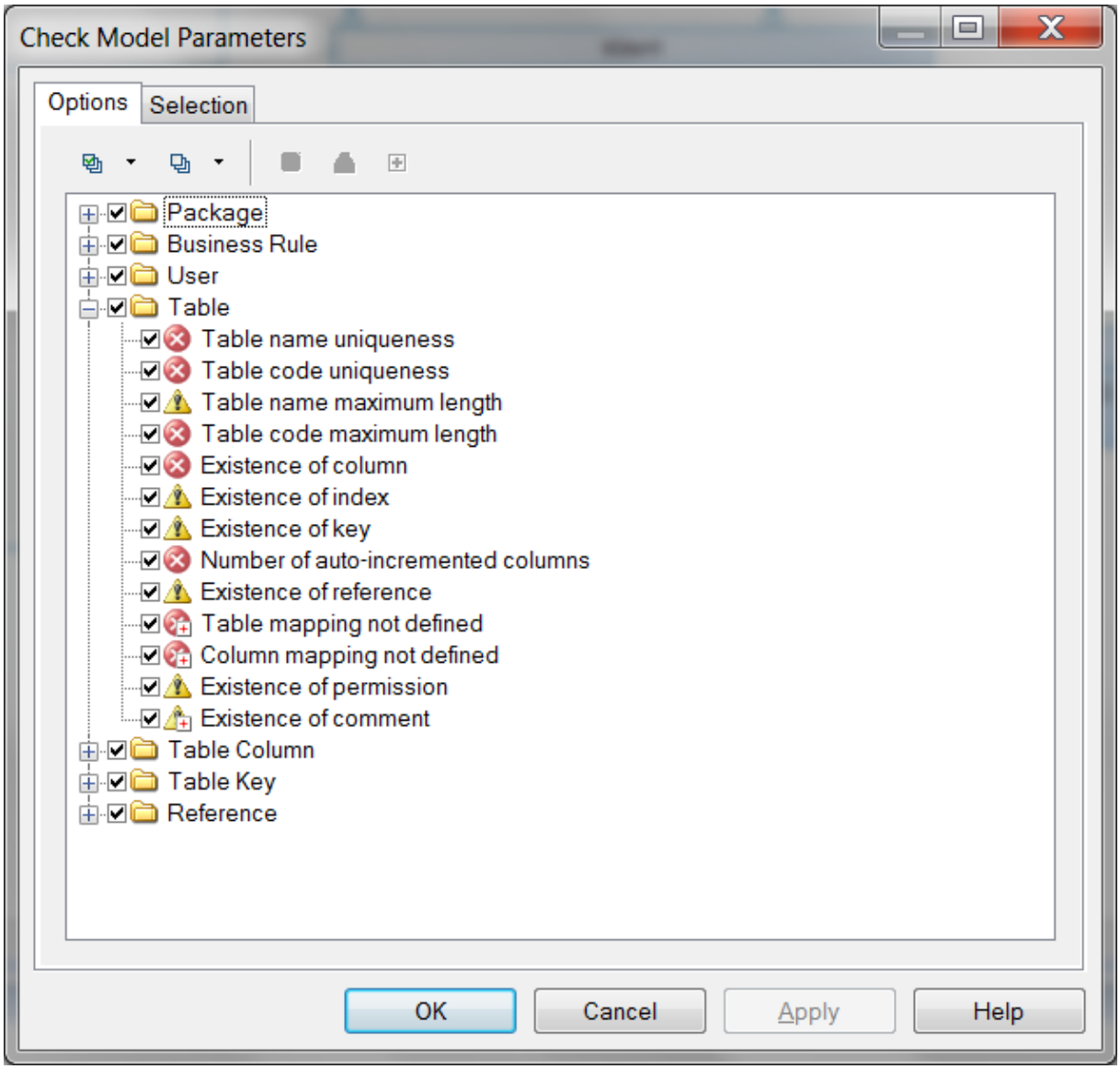


Joonis 17 Lisatud reegli (kommentaari olemasolu tabelil) skript

Pärast salvestamist (nupp „Apply“) lisandub uus kontrollreegel mudeli kontrollimise dialoogiaknasse (Joonis 18). Veerule kontrollreegli lisamine käib analoogselt tabelile kontrollreegli lisamisega (kontrolli kood on täpselt sama, lihtsalt funktsiooni sisend on erinev). Kuna minu töös kasutatavas andmemudelis on kõikide tabelite kõik veerud ilma kommentaarita, lisasin nii tabeli kui veeru kommentaari olemasolu kontrollile ka automaatse paranduse, et ei peaks kõiki kommentaare käsitsi lisama. Automaatse paranduse käigus pannakse tabeli kommentaariks tabeli nimi ja veeru kommentaariks veeru nimi. Paranduse eesmärgiks oli konkreetsel juhul vähendada aega, mis mul oleks kulunud käsitsi kõikide kommentaaride lisamiseks.

Antud eksperimendis ei oma kommentaaride sisu tähtusust, seetõttu sain automaatset parandust näite otstarbel rakendada. Reaalses projektis pole sellised vormilised kommentaarid

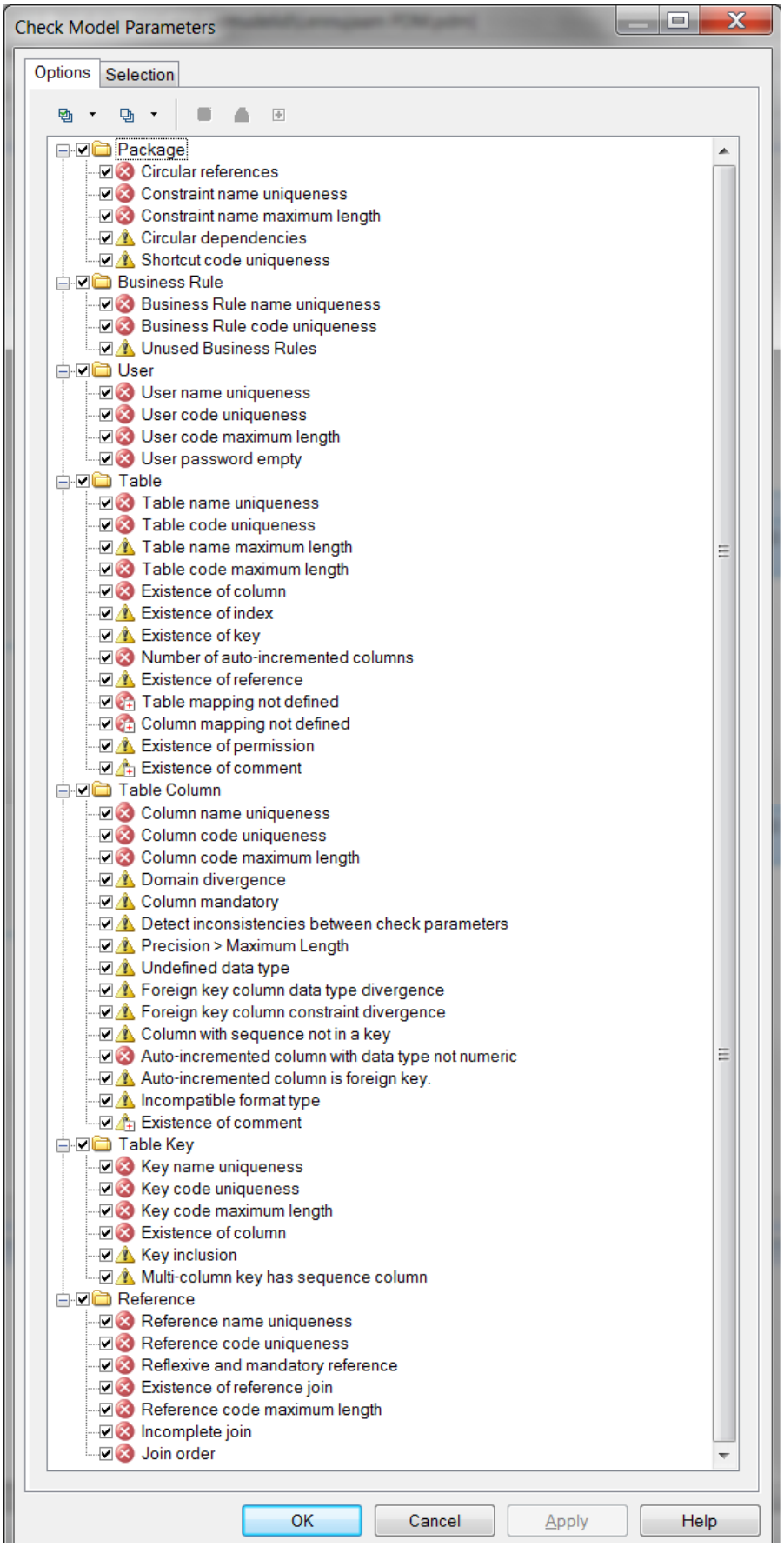
loomulikult mõttekad ja tuleks kirjutada sisulised kommentaarid. See viib omakorda mõtted kontrollreegli edasiarendusele, mis kontrollib lisaks kommentaari olemasolule ka seda, et peale algusest ja lõpust tühikute eemaldamist on kommentaar vähemalt mingi minimaalse pikkusega. Kommentaari sisukust kahjuks sellisel viisil kontrollida ei saa. Saaks ainult kontrollida seda, et kommentaar tõepoolest ei kordaks vastava tabeli/veeru nime.



Joonis 18 Tabeli kontrollreeglid. Viimane (Existence of comment) on ise lisatud.

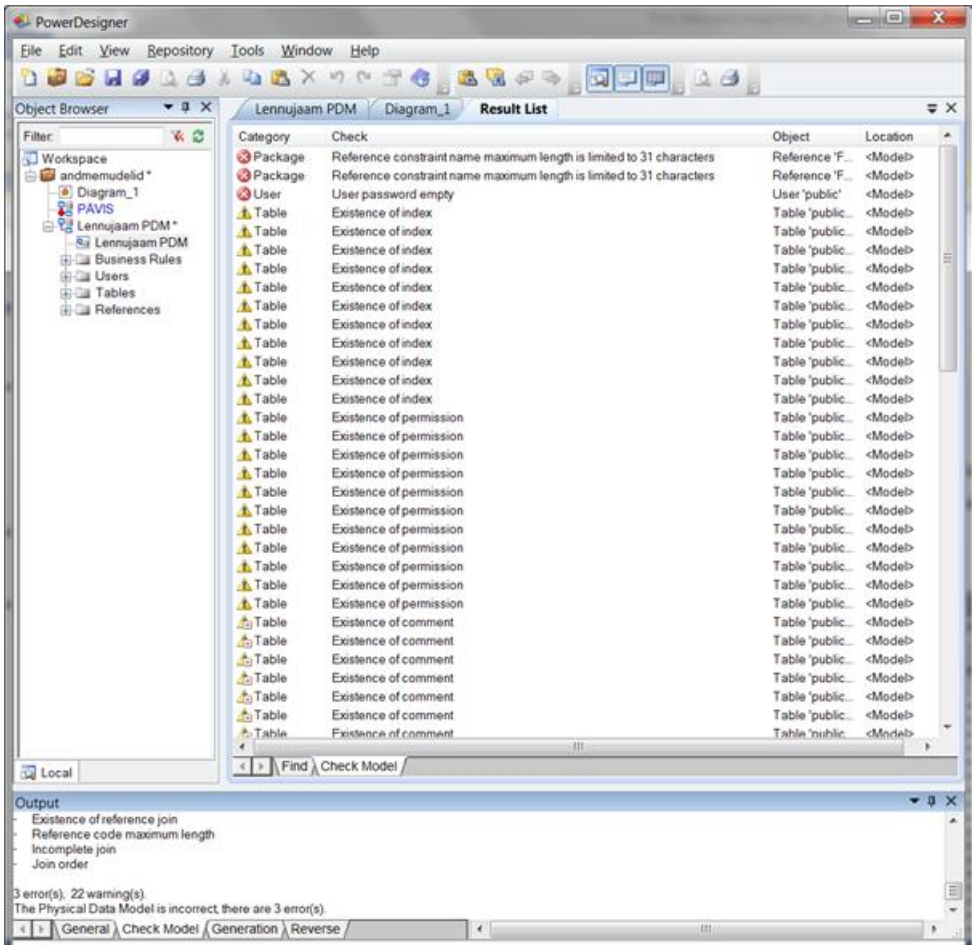
3.2.2 Andmemudeli automaatne kontrollimine

Esmalt kontrollin füüsilist andmemudelit sellisena, nagu ta esialgselt oli. Andmemudeli SQL ja diagrammid on vastavalt töö lisades 1 ja 2. Mudeli kontrollimiseks avan mudeli kontrollimise dialoogiakna ja valin kõik kontrollid. Joonis 19 näitab nimekirja kõigist antud mudelile rakendatavatest kontrollidest. Kontrollreegli ees olev sümbol näitab vea kriitilisust. Punane valge ristiga ringike tähendab, et tegemist on kriitilise veaga. Kollane hüüumärgiga kolmnurk tähistab hoiatust. Kui vea või hoiatusmärgi all paremas nurgas on pisike punane ristike, siis vea tuvastamisel parandatakse see süsteemi poolt automaatselt (*autofix*). Automaatse parandamise valikut ei ole kõigil kontrollidel ja isegi kui automaatse parandamise võimalus on olemas, on selle rakendamine valikuline. Kõigi eeldefineeritud kontrollide kirjeldused on dokumenteeritud PowerDesigner-i kasutusjuhendis, mis on veebist vabalt kättesaadav. Dokumentatsioonile saab ligi ka läbi rakenduse *Help* menüü.



Joonis 19 Füüsilise andmemudeli kontrollreeglid

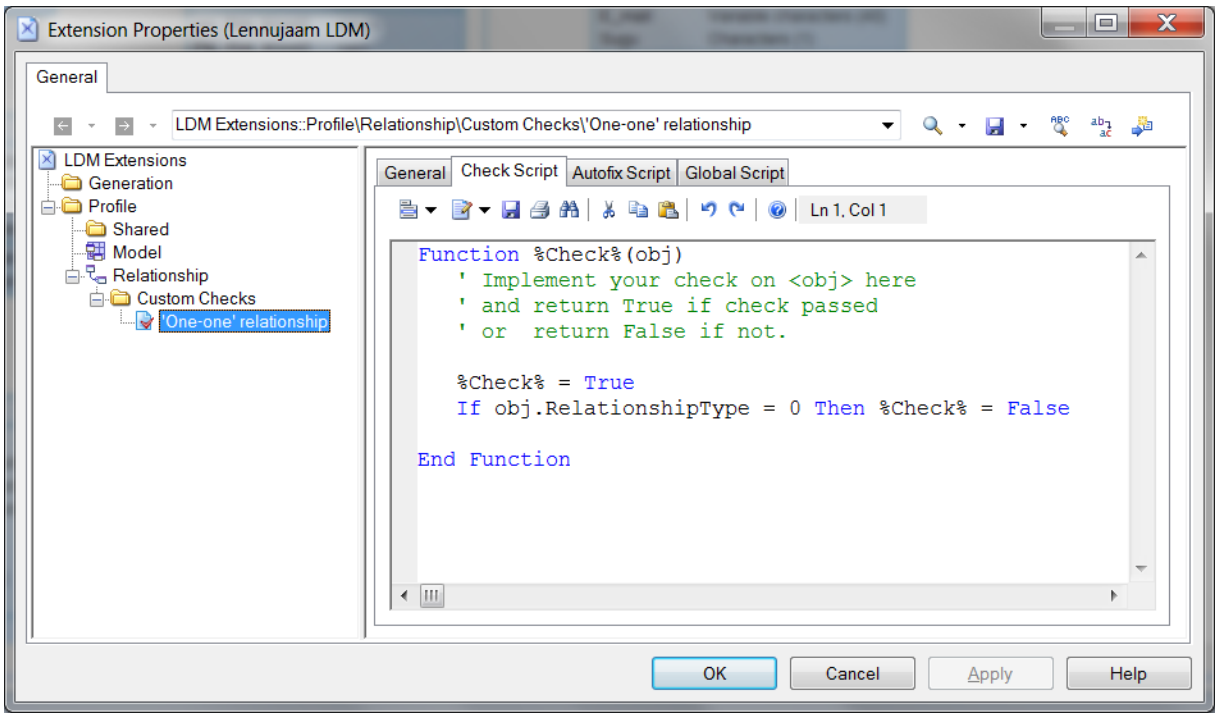
Pärast kontrolli käivitamist kuvatakse kontrolli tulemused põhiaknas eraldi lehel. Kontrolli tulemused on kuvatud Joonisel 20. Kõik hoiatused ei mahu ekraanile ära, kuna kõigil tabelitel ja veergudel olid kommentaarid puudu. Kontrolli teistkordsel käivitamisel on hoiatuste nimekiri tunduvalt lühem. Esialgsest mudelist leiti kolm viga: kahe välisvõtme kitsenduse nimetus on liiga pikk ja andmebaasi *public* kasutaja parool on määratlemata. Tehes topeltklõpsu leitud veal, avab rakendus vigase objekti dialoogiakna, kus saab viga parandada. Enne loogilise andmemudeli genereerimist parandan kolm leitud viga. Indekseid ja õigusi ma lisama ei hakka, kuna see on antud töö raames otstarbetu. Kontroll ei tuvastanud viga, et kuigi SQL-i põhjal oli tabelite *Isik-Klient* ja *Isik-Check_in_tootaja* vahel 1:1 seosetüüp, esitas PowerDesigner diagrammil nende asemel 1:M seosetüübid.



Joonis 20 Esialgse füüsilise mudeli automaatse kontrolli tulemused

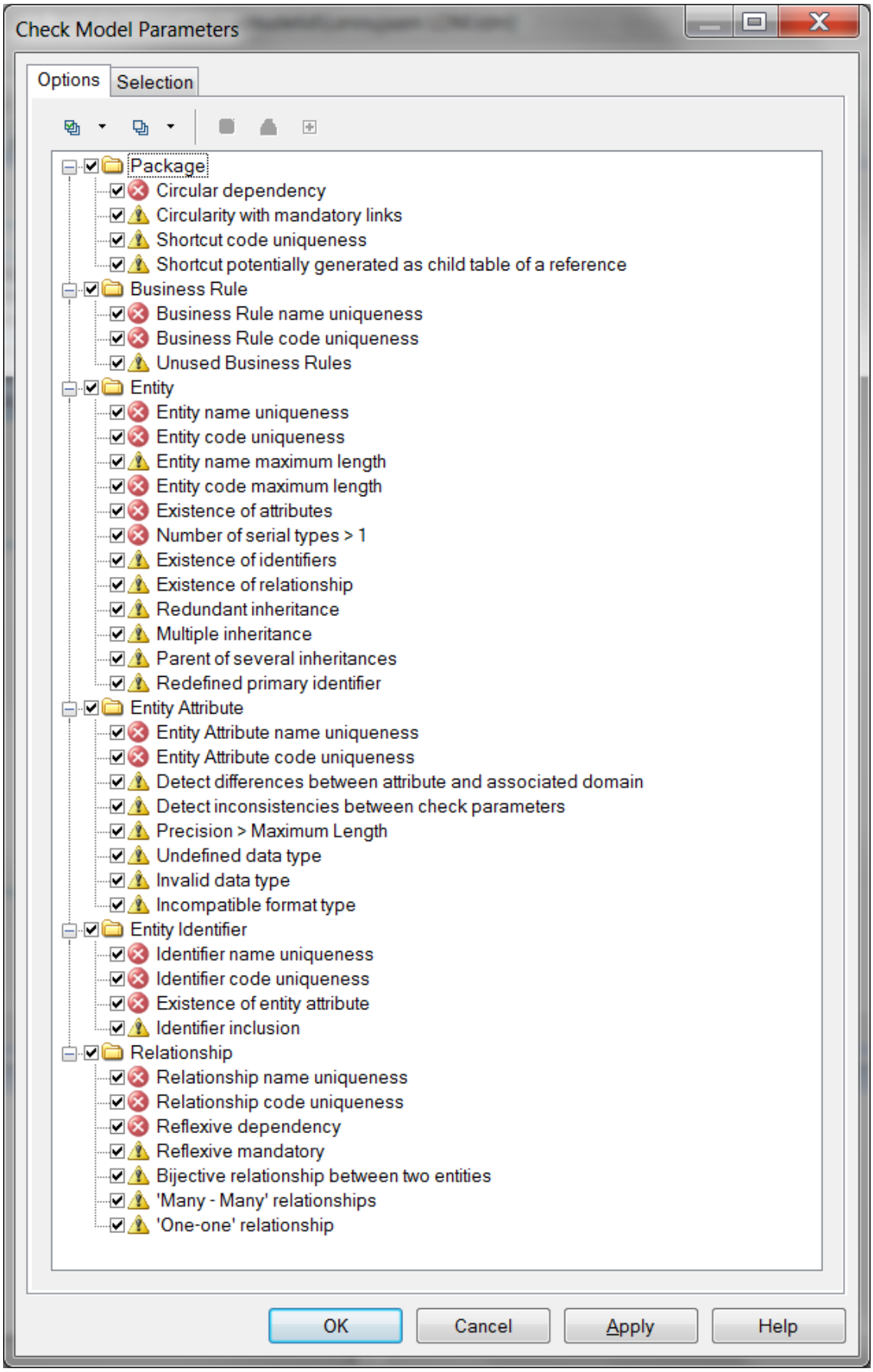
Parandasin PowerDesigner-i poolt tuvastatud andmemudeli vead ära, käivitasin kontrolli uuesti ja enam PowerDesigner mudelis vigu ei tuvasta. Nüüd saan füüsilise andmemudeli põhjal genereerida loogilise andmemudeli ja lisada sellesse meelega probleemseid kohti, et uurida ja analüüsida süsteemi võimet neid tuvastada. Loogilise andmemudeli genereerimiseks valin

menüüribalt Tools -> Generate Logical Data Model. Genereeritud loogilise andmemudeli diagramm on töö lisas number 3. Analoogselt füüsilisele andmemudelile kontrollreeglite lisamisele, saab nüüd kontrollreegleid lisada ka loogilisele andmemudelile. Lisasin kontrollreegli üks-ühele seoste tuvastamiseks loogilisest andmemudelist. Seda oli väga lihtne teha, kuna metaklassile *Relationship* on eeldefineeritud funktsioon *RelationshipType()*, mis tagastab täisarvu vahemikust 0–3 vastavalt sellele, mis tüüpi seosega on tegu (üks-ühele, üks-mitmele, mitu-ühele, mitu-mitmele). Vaata Joonis 21.



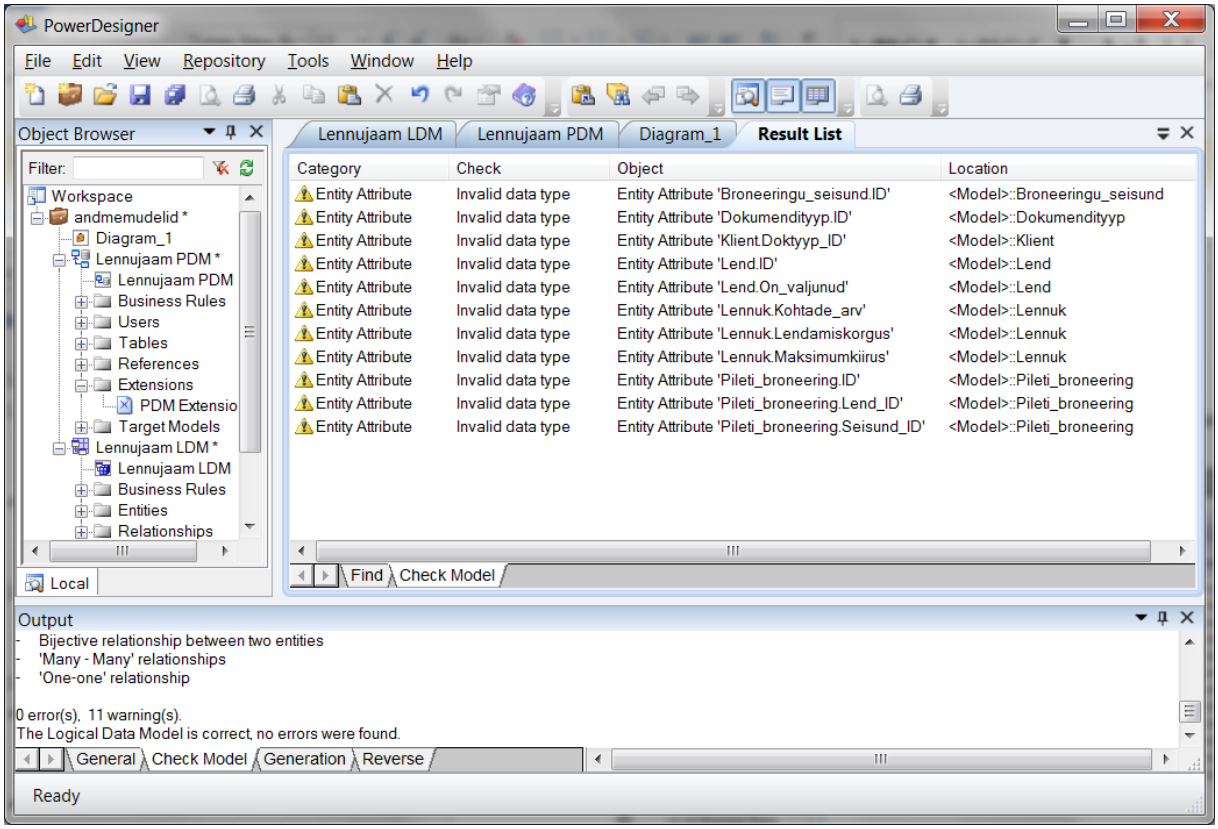
Joonis 21 Kontrollreegel loogilises andmemudelis üks-ühele seosetüüpide tuvastamiseks.

Enne kui lisan loogilisse andmemudelisse probleemseid kohti, kontrollin esialgset loogilist mudelit. Loogilise andmemudeli kontrollimine käib täpselt sama moodi kui füüsilise andmemudeli kontrollimine. Joonisel 22 on näha nimekiri kõigist loogilisele andmemudelile rakendatavatest reeglitest (kaasa arvatud ’One-one’ relationship, mille ise lisasin).



Joonis 22 Loogilise andmemudeli kontrollreeglid.

Loogilise andmemudeli kontrollimise tulemusena antakse 11 hoiatust kehtetute andmetüüpide kohta. Ühtegi võimalikku viga loogilises mudelis pole (Joonis 23). Kõik hoiatust põhjustavad atribuudid on kas INTEGER või SMALLINT tüüpi. Kui muuta kõik hoiatust andvad andmetüübid käsitsi Integer tüüpi ja kontrollida mudelit uuesti, siis ühtegi hoiatust ei saa. Ilmselt ei osanud süsteem füüsilisest andmemudelist loogilise mudeli genereerimisel INTEGER ja SMALLINT tüüpi teisendada ja jättis alles füüsilise andmemudeli andmetüübid.



Joonis 23 Esialgse loogilise mudeli automaatse kontrolli tulemused

Nüüd on nii loogiline kui ka füüsiline mudel valmis selleks, et sinna meelega probleemseid kohti lisada. Kontseptuaalset andmemudelit ma genereerima ega kontrollima ei hakka, kuna kontseptuaalsele andmemudelile rakendatavad kontrollid on rakendatavad ka loogilisele andmemudelile. Mudelisse probleemsete kohtade tekitamisel võtan aluseks kaardistatud kontrollreeglid, mis on PowerDesigner-s olemas (ka need, mis ise lisasin).

3.2.2.1 Meelega lisatud probleemsete kohtadega loogilise andmemudeli kontrollimine

Järgnevalt loetlen reeglid, millede toimimist ma loogilise andmemudeli puhul kontrollin. Sulgudes on kirjas kontrollreegli nimi PowerDesigner-s. Probleemsete kohtadega andmemudel on kujutatud loetelu all oleval joonisel 24.

- Veergude olemasolu tabelis (*Existence of attributes*) – Reegli kontrollimiseks eemaldan veerud tabelist Dokumendityyp.

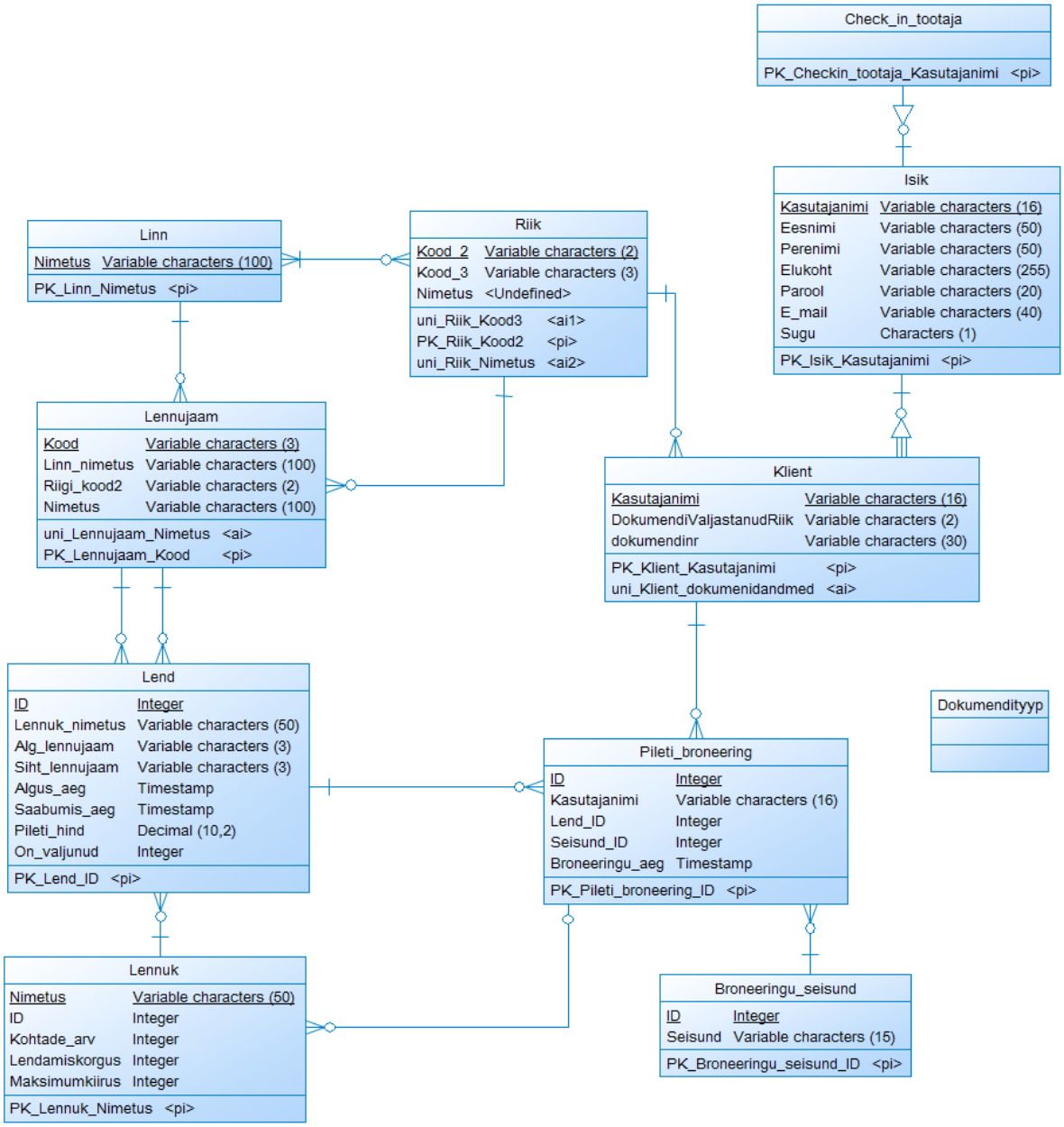
- Iga tabel on seotud mõne teise tabeliga (*Existence of relationship*) – Reegli kontrollimiseks eemaldan tabeleid Klient ja Dokumendityyp siduva välisvõtme.

- Kõigi veergude andmetüüpide määratletus (*Undefined datatype*) – Reegli kontrollimiseks eemaldan andmetüübi tabeli Riik veerult nimetus.

- Mitu-mitmele seosetüüpide puudumine (*'Many-many' relationships*) – Reegli kontrollimiseks tekitan mitu-mitmele seosetüübi tabelite Linn ja Riik vahele. Vaikimisi ei ole võimalik PowerDesigner-s tekitada mitu-mitmele seosetüüpi kahe tabeli vahele. Selleks, et see oleks võimalik, tuleb teha paremklõps andmemudelil, valida kohtmenüüst *Model Options…* ja teha avanevas dialoogiaknas linnuke *Allow n-n relationships* ette.

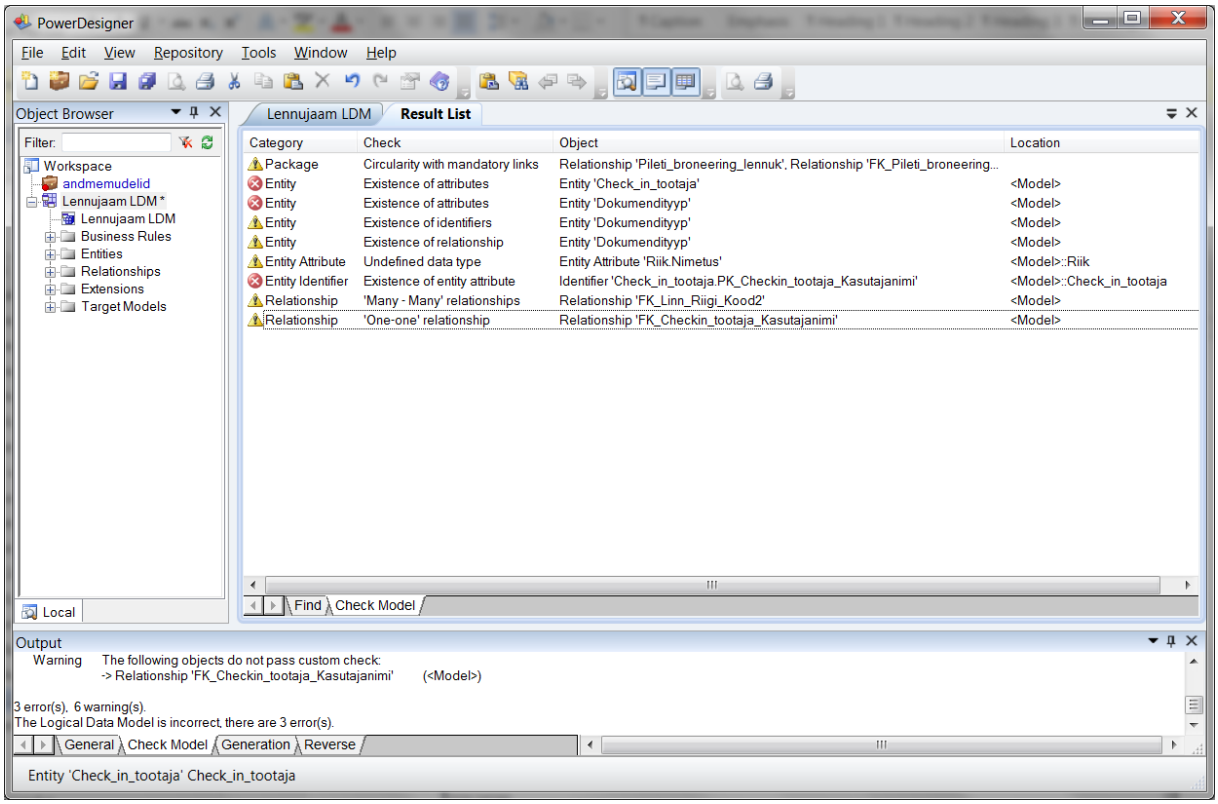
- Üks-ühele seosetüüpide puudumine (*'One-one' relationship*) – Reegli kontrollimiseks tekitan üks-ühele seosetüübi tabelite Isik ja Check_in_tootaja vahele. Esialgses mudelis Rational Rose-s oli üldistus, et töötaja on isik ja seega andmebaasi tekiks 1:1 seos tabelite Isik ja Töötaja vahele. Tegemist ei ole veaga. Sisuliselt on tegemist korrektse seosetüübiga, kuid kontrollimisel tuleb olukord siiski tuvastada.

- Puuduvad tsüklid (*Circularity with mandatory links*) – Reegli kontrollimiseks tekitan tsükli tabelite Lennuk, Lend ja Pileti_broneering vahele.



Joonis 24 Meelega tehtud probleemseid kohti sisaldav loogiline andmemudel.

Loodud mudeli automaatsel kontrollimisel leiti kolm viga ja anti 6 hoiatust. Kõik eelnevalt loetletud probleemsed kohad tuvastati. Vaata Joonis 25.



Joonis 25 Meelega lisatud probleemseid kohti sisaldava loogilise andmemudeli kontrollimise tulemus.

3.2.2.2 Meelega lisatud probleemsete kohtadega füüsilise andmemudeli kontrollimine

Järgnevalt loetlen reeglid, millede toimimist ma füüsilise andmemudeli puhul kontrollin. Sulgudes on kirjas kontrollreegli nimi PowerDesigner-s. Probleemsete kohtadega andmemudel on kujutatud Joonisel 26.

- Veergude olemasolu tabelil (*Existence of column*) – Reegli kontrollimiseks eemaldan veerud tabelist Dokumendityyp.
- Iga tabel on seotud mõne teise tabeliga (*Existence of reference*) – Reegli kontrollimiseks eemaldan välisvõtme, mis loob seose tabelite Klient ja Dokumendityyp vahel.
- (Primaar)võtme olemasolu igal tabelil (*Existence of key*) – Koos kõigi veergude eemaldamisega tabelist Dokumendityyp eemaldan ka võtmed.
- Tabeli nimi ei ületa maksimaalset pikkust (*Table code maximum length*) – Tabeli nime maksimaalset pikkust saab vaadata/muuta kui avada füüsiline andmemudel ja seejärel

avada menüüst *Database -> Edit Current BMS...*. Avanevas dialoogiaknas tuleb struktuuripuus avada *Scrip -> Objetcs -> Table -> Maxlen*. Vaikimisi on maksimaalseks tabeli nime pikkuseks 31 märki. PowerDesigner kasutab kontrollimisel seda arvu vaid juhul kui see on väiksem või võrdne andmebaasisüsteemi poolt määratud maksimaalsest pikkusest. Reegli kontrollimiseks muudan tabeli Dokumendityyp nime pikemaks kui 31 märki

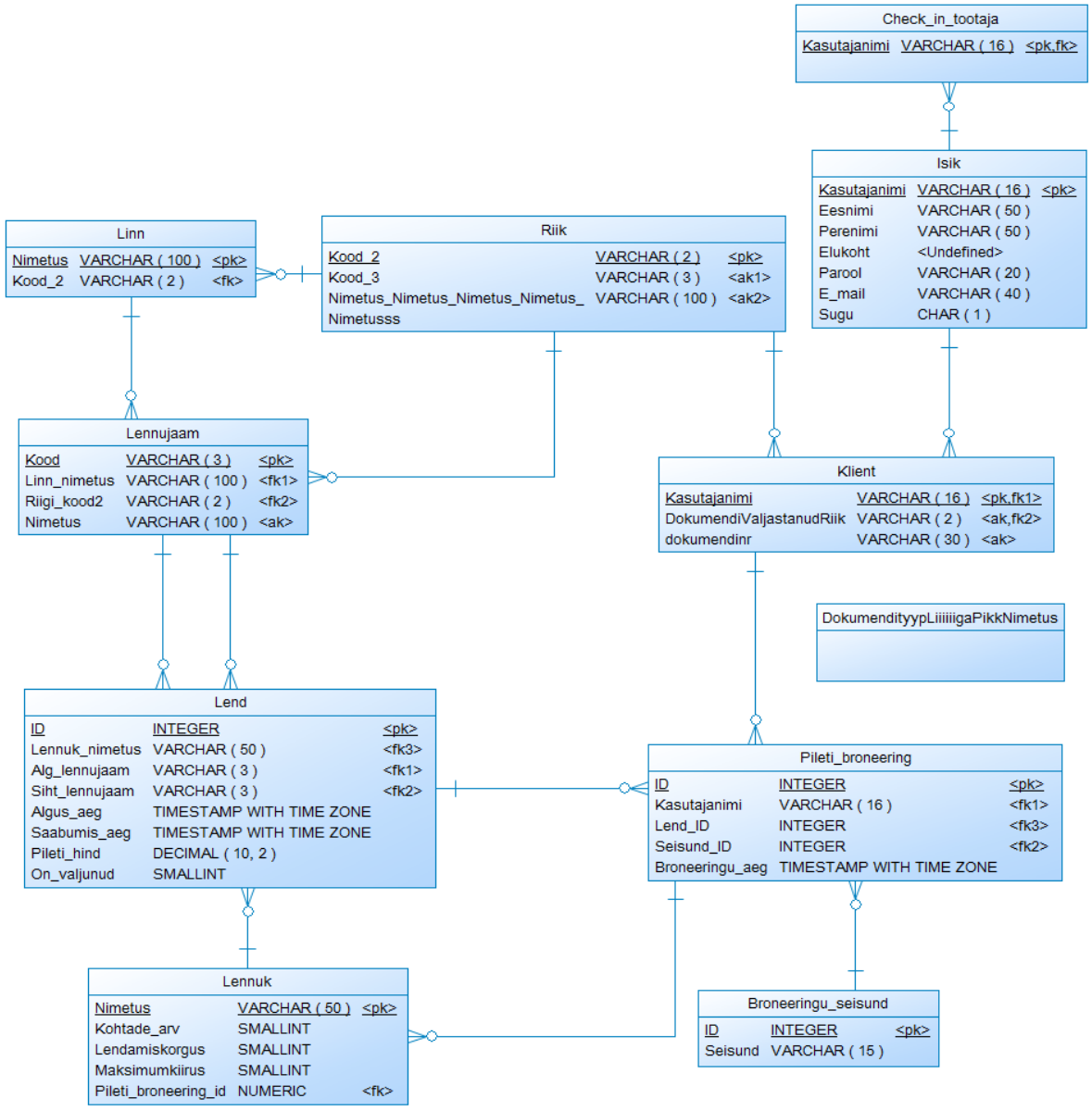
- Veeru nimi ei ületa maksimaalset pikkust (*Column code maximum length*) – Veeru nime maksimaalse pikkuse vaatamine/muutmine käib analoogselt tabeli nime maksimaalse pikkuse määramisega. Veeru nime maksimaalseks pikkuseks on samuti vaikimisi 31 märki. Reegli kontrollimiseks muudan tabeli Riik veeru nimetus nime pikemaks kui 31 märki.

- Kommentaari olemasolu tabelil (*Existence of comment*) – Reegli kontrollimiseks eemaldan kommentaari tabelilt Isik.

- Kommentaari olemasolu veerul (*Existence of comment*) – Reegli kontrollimiseks eemaldan kommentaari tabeli Klient veerult dokumendinr.

- Kõigi veergude andmetüüpide määratletus (*Undefined data type*) – Reegli kontrollimiseks eemaldan andmetüübi tabeli Isik veerult elukoht.

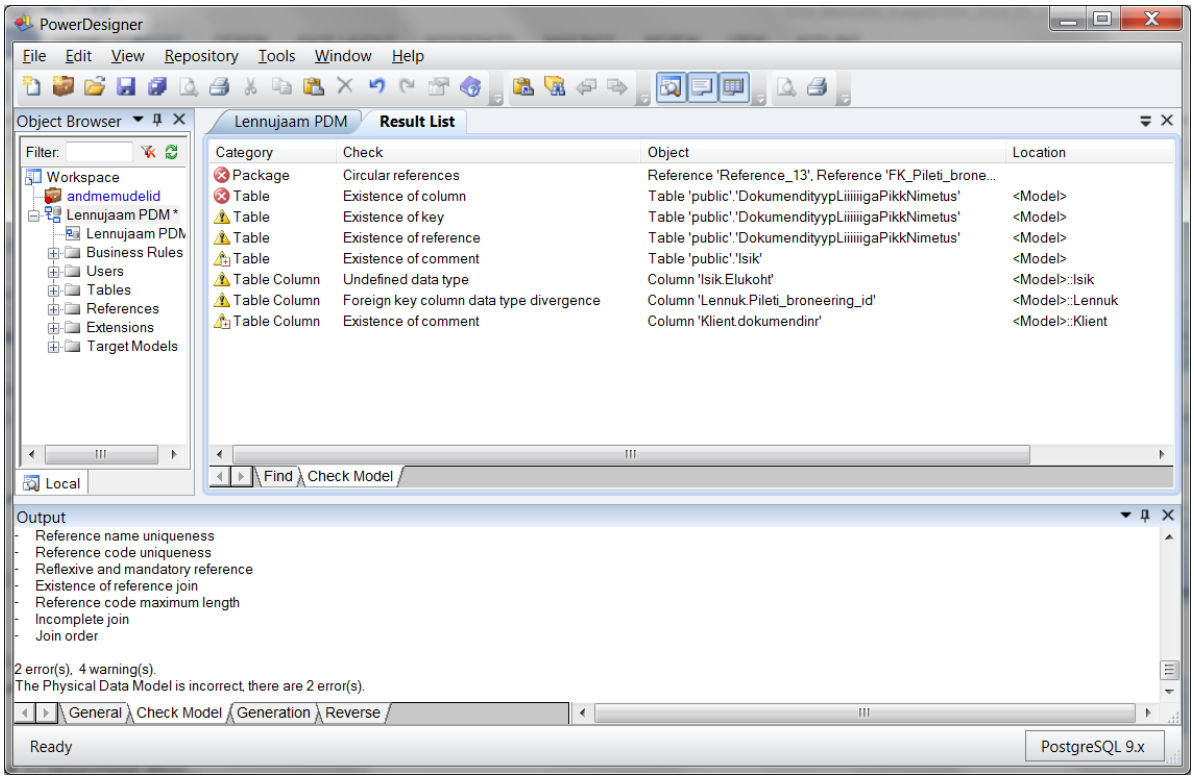
- Puuduvad tsüklid seosetüüpide vahel (*Circular references*) – Reegli kontrollimiseks loon tsükli tabelite Lennuk, Lend ja Pileti_broneering vahele.



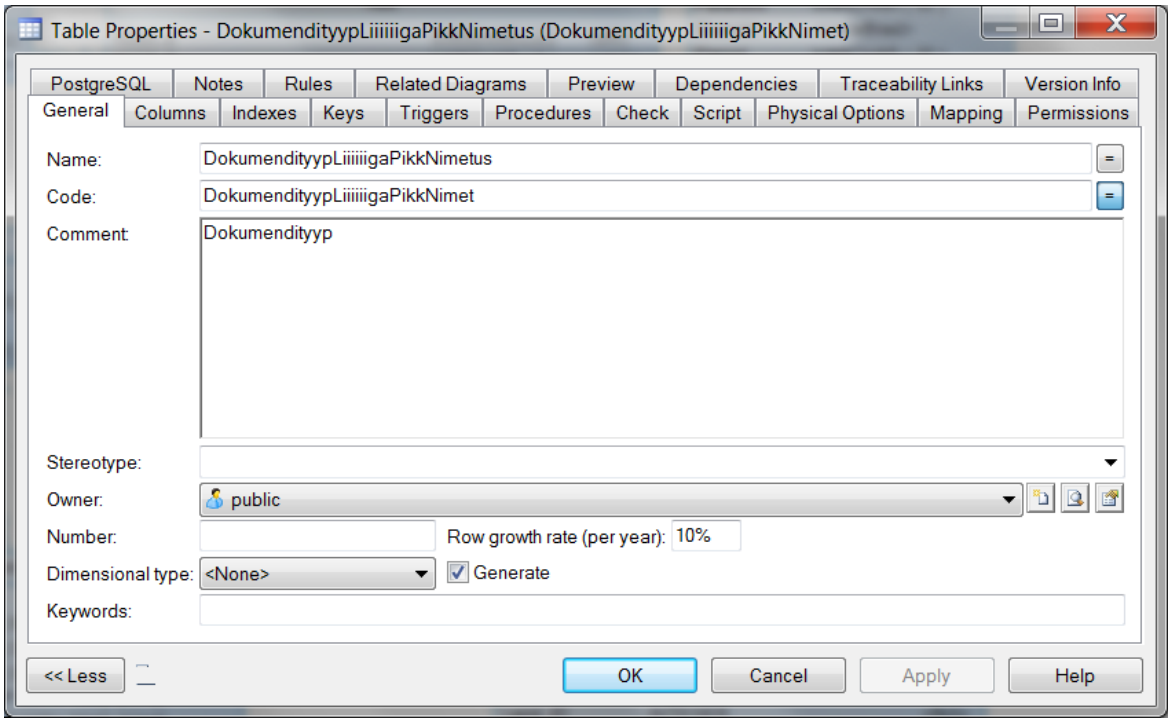
Joonis 26 Meelega lisatud probleemseid kohti sisaldav füüsiline andmemudel.

Füüsilise andmemudeli kontrollimisel ei rakenda ma seekord indeksite ja õiguste olemasolu kontrolli, kuna sain nende kohta hoiatuse juba esialgse mudeli kontrollimisel ega pole seda ära lahendanud. Probleemsete kohtadega mudeli automaatsel kontrollimisel leiti kaks viga ja anti kuus hoiatust (Joonis 27). Füüsilisest mudelist leiti kõik meelega lisatud probleemsed kohad peale selle, et ühe tabeli nimi ja ühe tabeli veeru nimi ületasid maksimaalset pikkust. See tuleneb sellest, et ma muutsin tabeli nime ja veeru nime küll pikaks, kuid tabeli kood ja veeru kood on lubatud pikkusega. Objektide nimed on PowerDesigner-i kasutajatele suunatud. Neid kuvatakse diagrammidel. Koodid seevastu on tehnilised tunnused, mida kasutatakse väliste objektide (näiteks SQL) genereerimiseks ja neile kehtivad tehnilised kitsendused (maksimaalne

lubatud pikkus). Vaikimisi on tabeli kood sama, mis tabeli nimi ja veeru kood on sama, mis veeru nimi, kuid kui nimi ületab maksimaalset pikkust, siis kood jääb lubatud piiridesse (nime lõpp võetakse vajalikus ulatuses maha). Vaata Joonis 28. Sealt on näha, et tabeli kood on paar märki lühem kui tabeli nimi, et mitte ületada maksimaalset lubatud pikkust.



Joonis 27 Meelega lisatud probleemseid kohti sisaldava füüsilise andmemudeli kontrollimise tulemus.



Joonis 28 Tabeli nimi ja tabeli kood

3.3 Eksperimendi tulemuste analüüs

Mudelite kontrollimine PowerDesigner-ga toimib hästi. Kõik meelega lisatud või eelneval vaatlusel tuvastatud probleemsed kohad tuvastati. Mudelite kontrollimise liides on intuitiivne ja tuvastatud võimalike vigade asukoha üles leidmine on tehtud väga lihtsaks. Vea- või hoiatusteatel topeltklõpsu tegemisel avaneb vigade allikaks oleva objekti seadete dialoogiaken. Näiteks kui on kuvatud hoiatusteade, et tabelil x puudub kommentaar, siis avaneb tabeli x redigeerimise dialoogiaken. Kui viga puudutab mudelit üldiselt, mitte mingit konkreetset ühte objekti sellest mudelist (näiteks mudelis esineb tsükkel), avaneb mudeli seadete dialoogiaken. Kui mudelist avastatakse tsükkel, siis on ka veateates välja toodud, milliste objektide vahel tsükkel leiti.

PowerDesigner-sse uute kontrollide lisamine on samuti tehtud väga kasutajasõbralikuks. Objektide atribuudid ja funktsioonid on põhjalikult ja selgelt dokumenteeritud. Eeldefineeritud funktsioonide hulk on päris suur, mis teeb samuti uute kontrollide lisamise lihtsamaks. Uute kontrollide juures oli kõige keerulisemaks etapiks leida üles moodus nende lisamiseks. Dokumentatsioonis oli kirjas, et tuleb teha paremklõps profiilis oleval metaklassil ja sealt edasi valida *New Custom Check*, aga see, kuidas pääseda ligi profiilile, ei olnud selgelt ja üheselt

mitte kusagil kirjas. Pärast pikka dokumentatsiooni lugemist ja mingi aimduse saamist sellest, mida on vaja otsida, leidsin lõpuks õige koha.

Andmemudelitele on rakendatavad kahte tüüpi reeglid: ranged reeglid, millede vastu eksimisel on tegemist veaga, ja mitte-ranged reeglid, millede vastu eksimisel ei pruugi olla tegu veaga, ning nende tuvastamisel väljastatakse hoiatus. Ranged reeglid on enamasti sellised, et kui nende vastu on eksitud, siis neid sisaldava mudeli pealt ei saa või ei ole soovitatav midagi genereerida. Näiteks kui füüsilise andmemudeli mingi objekti identifikaator ületab andmebaasisüsteemi poolt lubatud maksimaalset pikkust, siis enne vea parandamist ei ole mõtet mudeli põhjal andmebaasi loomise lauseid genereerida. Hoiatusi väljastavad reeglid seevastu tuvastavad mudelist potentsiaalseid probleeme, kuid sõltuvalt olukorrast ei pruugi probleemi lahendamine olla vajalik. Selle näiteks on kommentaaride olemasolu tabelitel ja veergudel. Kommentaaride panemine on küll hea praktika ja võib olla äärmiselt kasulik, kuid kui näiteks tabelite primaarvõtmeks on surrogaatvõti, siis selle veeru sisu kommenteerimine ei ole tingimata vajalik. Samuti võib olla põhjendatud olemitüüpide või tabelite vaheline üks-ühele või mitu-mitmele seos, kuigi enamikel juhtudel viitab see probleemile.

PowerDesigner-i näite põhjal leian, et mudelite automaatne kontrollimine on efektiivne ja võiks tulevikus leida laiemat kasutust. Automaatse kontrollimise käigus ei ole võimalik leida kõiki probleemseid kohti ja süsteem võib tuvastada probleeme, mis tegelikult ei ole vead. Sellele vaatamata arvan, et mudelite automaatse kontrollimisega on võimalik parandada andmemudelite kvaliteeti. See võimaldab mudelit väga regulaarselt kontrollida ja probleeme varakult tuvastada.

Kokkuvõte

Töö eesmärgiks oli uurida andmemudelite automaatset kontrollimist CASE vahendiga, mille eesmärgiks on tõsta andmemudelite kvaliteeti. Selleks oli vaja kaardistada ja kategoriseerida reeglid, võrrelda erinevate CASE vahendite võimalusi andmemudelite kontrollimiseks, viia läbi andmemudelite kontrollimise eksperiment ühe väljavalitud CASE vahendiga ning analüüsida eksperimendi tulemusi.

Töö käigus kaardistati hulk reegleid, mida võiks automaatselt CASE vahendiga kontrollida. Jaotasin reeglid kolme kategooriasse: kontseptuaalse andmemudeli reeglid, loogilise andmemudeli reeglid ja füüsilise andmemudeli reeglid. Üks reegel võib kehtida ühele või mitmele andmemudeli tüübile. Valisin välja kaheksa CASE vahendit ning selgitasin välja järgneva.

- Kas antud vahendil on andmemudeli automaatse kontrollimise funktsionaalsus?
- Kui automaatse kontrollimise funktsionaalsus on olemas, siis kas reegleid saab ise juurde kirjutada?
- Kui andmemudelite automaatse kontrollimise funktsionaalsus puudub, siis kas seda on võimalik ise juurde teha?

Väljavalitud vahenditest toetasid andmemudelite automaatset kontrollimist *Oracle SQL Developer Data Modeler*, *Enterprise Architect*, *ER/Studio*, *Rational Rose* ja *PowerDesigner*. Reeglite lisamist võimaldavad *SQL Developer Data Modeler* ja *PowerDesigner*. Automaatse kontrollimise funktsionaalsuse lisamise võimalus on *Rational Rose*-l ja *DB Main*-l. Eksperimendi läbiviimseks valisin vahendi *PowerDesigner*. Töös on kirjeldatud mudelite kontrollimise funktsionaalsuse kasutamine, reeglite lisamine ja viited kasulikele allikatele seoses mudelite kontrollimise ja reeglite loomisega *PowerDesigner*-s. Töös on kirjeldatud ja ekraanipiltidega ilmestatud eksperimendi läbi viimise protsess ja tulemuste analüüs.

Andmemudelite automaatne kontrollimine CASE vahenditega on efektiivne ja võiks tulevikus leida laialdasemat kasutust. Kõiki probleemseid kohti ei ole võimalik CASE vahenditega tuvastada ja tuvastatavad probleemid ei pruugi olla vead. Sellegi poolest aitaks andmemudelite regulaarne automaatne kontrollimine tõsta nende kvaliteeti ja seeläbi ka andmebaasile toetuva rakenduse kvaliteeti. Enamike uuritud vahendite mudelite kontrollimise võimalused on praegu

väga algelised ja mitte kuigi panindlikud ja kasutajasõbralikud. PowerDesigneris on autori hinnangul andmemudelite kontrollimine lihtne ja paindlik.

Kuna mudelite automaatse kontrollimise valdkonda ei ole praegusel hetkel laialdaselt uuritud, on sellel teemal võimalik kirjutada mitmeid jätkutöid. Kuna antud töö käsitles ainult andmemudelite kontrollimist, võiks uurida ka näiteks muud tüüpi mudelite automaatse kontrollimise võimalusi. Samuti võiks uurida CASE vahenditega erinevate mudelite kooskõla kontrollimise funktsionaalsust. Programmeerijatele võib olla huvitav mõnele CASE vahendile mudelite kontrollimise funktsionaalsuse ise lisamine või DB Main-le seisundidiagrammide lisamise funktsionaalsuse arendamine.

Summary

The aim of this thesis was to investigate the possibility to improve the quality of data models using automated model checks of CASE tools. For that, I needed to gather and categorize rules that a CASE tool could automatically check. One of the purposes of the work was also to compare different CASE tools in the aspect of automated data model checking. After comparing different CASE tools, I had to choose one to carry out an experiment with and analyse the results. Experiment constitutes of adding errors to a data model and letting the CASE tool detect them.

In my thesis, I gathered rules that CASE tools could automatically check and divided them into three categories: conceptual data model rules, logical data model rules and physical data model rules. One rule can fit into one or more categories. For the thesis, I chose eight CASE tools (*Oracle SQL Developer Data Modeler*, *TOAD Data Modeler*, *Enterprise Architect*, *Rational Rose*, *PowerDesigner*, *ER/Studio*, *CA Erwin*, *DB Main*) to explore their ability of automatic data model checking. I tried to find answers to the following questions.

- Does the CASE tool support automatic data model checking?
- If it does, then is there a possibility to add custom checks?
- If the CASE tool does not have the functionality of automatic model checking, then is it possible to add it?

Out of the selected tools only *Oracle SQL Developer Data Modeler*, *Enterprise Architect*, *ER/Studio*, *Rational Rose,* and *PowerDesigner* support the automatic model checking. However, only *SQL Developer Data Modeler* and *PowerDesigner* provide the interface for creating custom checks. *Rational Rose* and *DB Main* can be extended so there is a possibility to improve or add the model checking functionality. I chose *PowerDesigner* to carry out the experiment. In my work, it is described how to use the model checking functionality, how to add custom checks, and I referenced useful sources regarding model checking and adding rules in *PowerDesigner*. I described and illustrated the process of carrying out the experiment in detail. In addition, I analysed the results.

Taking *PowerDesigner* as a reference, I find checking data models with CASE tools effective and in my opinion it should find more extensive use in the future. It is not possible to detect all

mistakes in a data model with a CASE tool. In addition, some of the detected problems might actually not be mistakes. Nevertheless, regular automated checking of data models would help to improve their quality and thereby improve the quality of the systems relying on that database. Most of the explored CASE tools provide rather primitive interface for model checking. They are not very flexible or user-friendly. In my opinion *PowerDesigner* is intuitive and easy to use in addition to be very flexible.

Kasutatud kirjandus

1. Hofstader, J. (2006). Model-Driven Development [WWW] http://msdn.microsoft.com/en-us/library/aa964145.aspx (07.02.2014)
2. Zhang, Y., Patel, S. (2011). Agile Model-Driven Development in Practice. — *IEEE Software*, 28(2), 84-91. [Online] http://www.profs.iffca.edu.br/~rosana/Pos-gradua%E7%E3o/artigos%20MDE%20MDD%20MDA%20sugest%F5es/8%20-%20Agile%20Model-Driven%20Development%20in%20Practice.pdf (13.03.2014)
3. Ambler, S. W. Agile Model Driven Development (AMDD): The Key to Scaling Agile Software Development. [WWW] http://www.agilemodeling.com/essays/amdd.htm (13.03.2014)
4. Ambler, S. W. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. New York: John Wiley & Sons, Inc.
5. Ambler, S. W. (2004). The Object Primer: Agile Model-Driven Development with UML 2.0. Third edition. Cambridge: Cambridge University Press.
6. Dori, D. (2002). Why Significant Change in UML is Unlikely. — *Communications of the ACM*, 45(11), 82-85. [Online] http://75.98.171.106/docs/article71.pdf (13.03.2014)
7. Ambler, S. W. Data Modeling 101. [WWW] http://www.agiledata.org/essays/dataModeling101.html (07.02.2014)
8. Gupta, S. (2012). Difference between Conceptual, Logical and Physical Data Models. [WWW] http://uksanjay.blogspot.com/2012/06/difference-between-conceptual-logical.html (09.03.2014)
9. Ambler, S. W. Agile/Evolutionary Data Modeling: From Domain Modeling to Physical Modeling. [WWW] http://agiledata.org/essays/agileDataModeling.html (14.03.2014)
10. Jabeur, N. (2006). Chapter 4 Spatial data modeling and generation. [WWW] http://theses.ulaval.ca/archimede/fichiers/23356/ch04.html (14.03.2014)
11. Eessaar, E. (2014). Andmebaaside projekteerimiseks kasutatavad mudelid. [Online] maurus.ttu.ee(11.04.2014)
12. Hay, D. C. (2003). Data Model Quality: Where Good Data Begins [WWW] http://www.tdan.com/view-articles/5286 (28.03.2014)

13. Moody, D. L., Shanks, G. G. (2002). Improving the quality of data models: empirical validation of a quality management framework. — *Information Systems*, 28(6), 619–650 [Online] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.4270&rep=rep1&type=pdf (28.03.2014)

14. Moody, D. L. (2003). Measuring the Quality of Data Models: An Empirical Evaluation of the Use of Quality Metrics in Practice. [Online] http://is2.lse.ac.uk/asp/aspecis/20030099.pdf (24.04.2014)

15. Wikipedia artikkel ISO/IEC 9126. [WWW] http://en.wikipedia.org/wiki/ISO/IEC_9126 (22.04.2014)

16. Claxaton, J., McDougall, P. A. (2000). Measuring the Quality of Models. [WWW] http://www.tdan.com/view-articles/4877/ (04.04.2014)

17. McDonald, K., Matts, C. (2013). The Seven Information Smells of Domain Modelling. [WWW] http://www.infoq.com/articles/seven-modelling-smells (07.05.2014)

18. Krishnamurthy, G. CASE Tools. [WWW] http://www.umsl.edu/~sauterv/analysis/F08papers/View.html (21.02.2014)

19. Wikipedia artikkel PowerDesigner. [WWW] http://en.wikipedia.org/w/index.php?title=PowerDesigner&oldid=600431724 (21.02.2014)

20. Wikipedia artikkel CA ERwin Data Modeler. [WWW] http://en.wikipedia.org/w/index.php?title=CA_ERwin_Data_Modeler&oldid=600431791 (21.02.2014)

21. Using the RoseExtensibility Interface. [Online] ftp://ftp.software.ibm.com/software/rational/docs/v2003/win_solutions/rational_rose/rose_rei_guide.pdf (05.04.2014)

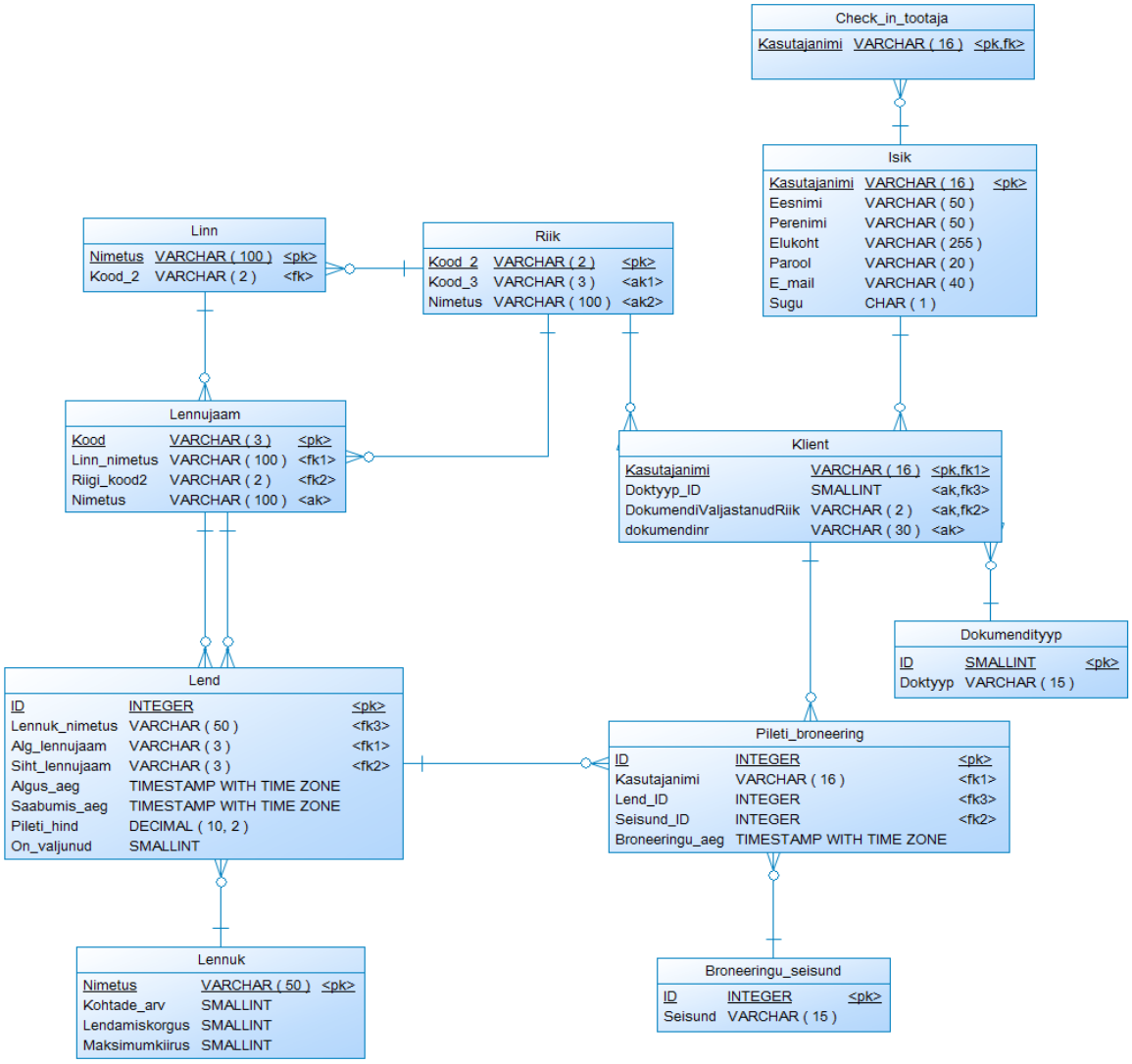
Lisa 1. Füüsilise andmemudeli SQL

```
CREATE TABLE Broneeringu_seisund (
    ID INTEGER NOT NULL,
    Seisund VARCHAR ( 15 ) NOT NULL,
    CONSTRAINT PK_Broneeringu_seisund_ID PRIMARY KEY (ID)
    );
CREATE TABLE Check_in_tootaja (
    Kasutajanimi VARCHAR ( 16 ) NOT NULL,
    CONSTRAINT PK_Checkin_tootaja_Kasutajanimi PRIMARY KEY (Kasutajanimi)
    );
CREATE TABLE Lennuk (
    Nimetus VARCHAR ( 50 ) NOT NULL,
    Kohtade_arv SMALLINT NOT NULL,
    Lendamiskorgus SMALLINT NOT NULL,
    Maksimumkiirus SMALLINT NOT NULL,
    CONSTRAINT PK_Lennuk_Nimetus PRIMARY KEY (Nimetus),
    CONSTRAINT chk_Lennuk_Kohtade_Arv CHECK (Kohtade_arv>0),
    CONSTRAINT chk_Lennuk_Maksimumkiirus CHECK (Maksimumkiirus BETWEEN 500
AND 1500),
    CONSTRAINT chk_Lennuk_Lendamiskorgus CHECK (Lendamiskorgus BETWEEN 5000
and 15000)
    );
CREATE TABLE Isik (
    Kasutajanimi VARCHAR ( 16 ) NOT NULL,
    Eesnimi VARCHAR ( 50 ) NOT NULL,
    Perenimi VARCHAR ( 50 ) NOT NULL,
    Elukoht VARCHAR ( 255 ) DEFAULT  NOT NULL,
    Parool VARCHAR ( 20 ) NOT NULL,
    E_mail VARCHAR ( 40 ) NOT NULL,
    Sugu CHAR ( 1 ) DEFAULT M NOT NULL,
    CONSTRAINT PK_Isik_Kasutajanimi PRIMARY KEY (Kasutajanimi),
    CONSTRAINT chk_Isik_Kasutajanimi CHECK
((kasutajanimi~'^[[:alnum:]]{6,15}$')),
    CONSTRAINT chk_Isik_Eesnimi CHECK ((trim(Eesnimi)!='')),
    CONSTRAINT chk_Isik_E_mail CHECK ((e-mail~'^.+@.+[.].{2,3}$')),
    CONSTRAINT chk_Isik_Elukoht CHECK ((trim(Elukoht)!='')),
    CONSTRAINT chk_Isik_Perenimi CHECK ((trim(Perenimi)!='')),
    CONSTRAINT chk_Isik_Sugu CHECK (Sugu IN ('M','N'))
    );
CREATE TABLE Klient (
    Kasutajanimi VARCHAR ( 16 ) NOT NULL,
    Doktyyp_ID SMALLINT NOT NULL,
    DokumendiValjastanudRiik VARCHAR ( 2 ) NOT NULL,
    dokumendinr VARCHAR ( 30 ) NOT NULL,
    CONSTRAINT PK_Klient_Kasutajanimi PRIMARY KEY (Kasutajanimi),
    CONSTRAINT uni_Klient_dokumenidandmed UNIQUE (dokumendinr,
DokumendiValjastanudRiik, Doktyyp_ID)
    );
CREATE TABLE Lennujaam (
    Kood VARCHAR ( 3 ) NOT NULL,
    Linn_nimetus VARCHAR ( 100 ) NOT NULL,
    Riigi_kood2 VARCHAR ( 2 ) NOT NULL,
    Nimetus VARCHAR ( 100 ) NOT NULL,
    CONSTRAINT uni_Lennujaam_Nimetus UNIQUE (Nimetus),
    CONSTRAINT PK_Lennujaam_Kood PRIMARY KEY (Kood),
    CONSTRAINT chk_Lennujaam_Kood CHECK ((Kood~'^[[:alpha:]]{3}$'))
    );
CREATE TABLE Pileti_broneering (
    ID INTEGER NOT NULL,
```

```
    Kasutajanimi VARCHAR ( 16 ) NOT NULL,
    Lend_ID INTEGER NOT NULL,
    Seisund_ID INTEGER NOT NULL,
    Broneeringu_aeg TIMESTAMP WITH TIME ZONE NOT NULL,
    CONSTRAINT PK_Pileti_broneering_ID PRIMARY KEY (ID)
    );
CREATE TABLE Linn (
    Nimetus VARCHAR ( 100 ) NOT NULL,
    Kood_2 VARCHAR ( 2 ) NOT NULL,
    CONSTRAINT PK_Linn_Nimetus PRIMARY KEY (Nimetus)
    );
CREATE TABLE Riik (
    Kood_2 VARCHAR ( 2 ) NOT NULL,
    Kood_3 VARCHAR ( 3 ) NOT NULL,
    Nimetus VARCHAR ( 100 ) NOT NULL,
    CONSTRAINT uni_Riik_Kood3 UNIQUE (Kood_3),
    CONSTRAINT PK_Riik_Kood2 PRIMARY KEY (Kood_2),
    CONSTRAINT uni_Riik_Nimetus UNIQUE (Nimetus),
    CONSTRAINT chk_Riik_Kood3 CHECK ((Kood_3~'^[[:alpha:]]{3}$')),
    CONSTRAINT chk_Riik_Kood2 CHECK ((Kood_2~'^[[:alpha:]]{2}$'))
    );
CREATE TABLE Lend (
    ID INTEGER NOT NULL,
    Lennuk_nimetus VARCHAR ( 50 ) NOT NULL,
    Alg_lennujaam VARCHAR ( 3 ) NOT NULL,
    Siht_lennujaam VARCHAR ( 3 ) NOT NULL,
    Algus_aeg TIMESTAMP WITH TIME ZONE NOT NULL,
    Saabumis_aeg TIMESTAMP WITH TIME ZONE NOT NULL,
    Pileti_hind DECIMAL ( 10, 2 ) NOT NULL,
    On_valjunud SMALLINT DEFAULT 0 NOT NULL,
    CONSTRAINT PK_Lend_ID PRIMARY KEY (ID),
    CONSTRAINT chk_Lend_On_Valjund CHECK (on_valjunud=0 OR on_valjunud=1)
    );
CREATE TABLE Dokumendityyp (
    ID SMALLINT NOT NULL,
    Doktyyp VARCHAR ( 15 ) NOT NULL,
    CONSTRAINT PK_Dokumendityyp_ID PRIMARY KEY (ID)
    );
ALTER TABLE Linn ADD CONSTRAINT FK_Linn_Riigi_Kood2 FOREIGN KEY (Kood_2)
REFERENCES Riik (Kood_2)  ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Lend ADD CONSTRAINT FK_Lend_Alg_lennujaam FOREIGN KEY
(Alg_lennujaam) REFERENCES Lennujaam (Kood)  ON DELETE NO ACTION ON UPDATE
NO ACTION;
ALTER TABLE Lend ADD CONSTRAINT FK_Lend_Siht_lennujaam FOREIGN KEY
(Siht_lennujaam) REFERENCES Lennujaam (Kood)  ON DELETE NO ACTION ON UPDATE
NO ACTION;
ALTER TABLE Lend ADD CONSTRAINT FK_Lend_Lennuk_nimetus FOREIGN KEY
(Lennuk_nimetus) REFERENCES Lennuk (Nimetus)  ON DELETE NO ACTION ON UPDATE
NO ACTION;
ALTER TABLE Pileti_broneering ADD CONSTRAINT
FK_Pileti_broneering_Kasutajanimi FOREIGN KEY (Kasutajanimi) REFERENCES
Klient (Kasutajanimi)  ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Pileti_broneering ADD CONSTRAINT
FK_Pileti_broneering_Seisund_ID FOREIGN KEY (Seisund_ID) REFERENCES
Broneeringu_seisund (ID)  ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Pileti_broneering ADD CONSTRAINT FK_Pileti_broneering_Lend_ID
FOREIGN KEY (Lend_ID) REFERENCES Lend (ID)  ON DELETE NO ACTION ON UPDATE
NO ACTION;
ALTER TABLE Check_in_tootaja ADD CONSTRAINT FK_Checkin_tootaja_Kasutajanimi
FOREIGN KEY (Kasutajanimi) REFERENCES Isik (Kasutajanimi)  ON DELETE NO
ACTION ON UPDATE NO ACTION;
```

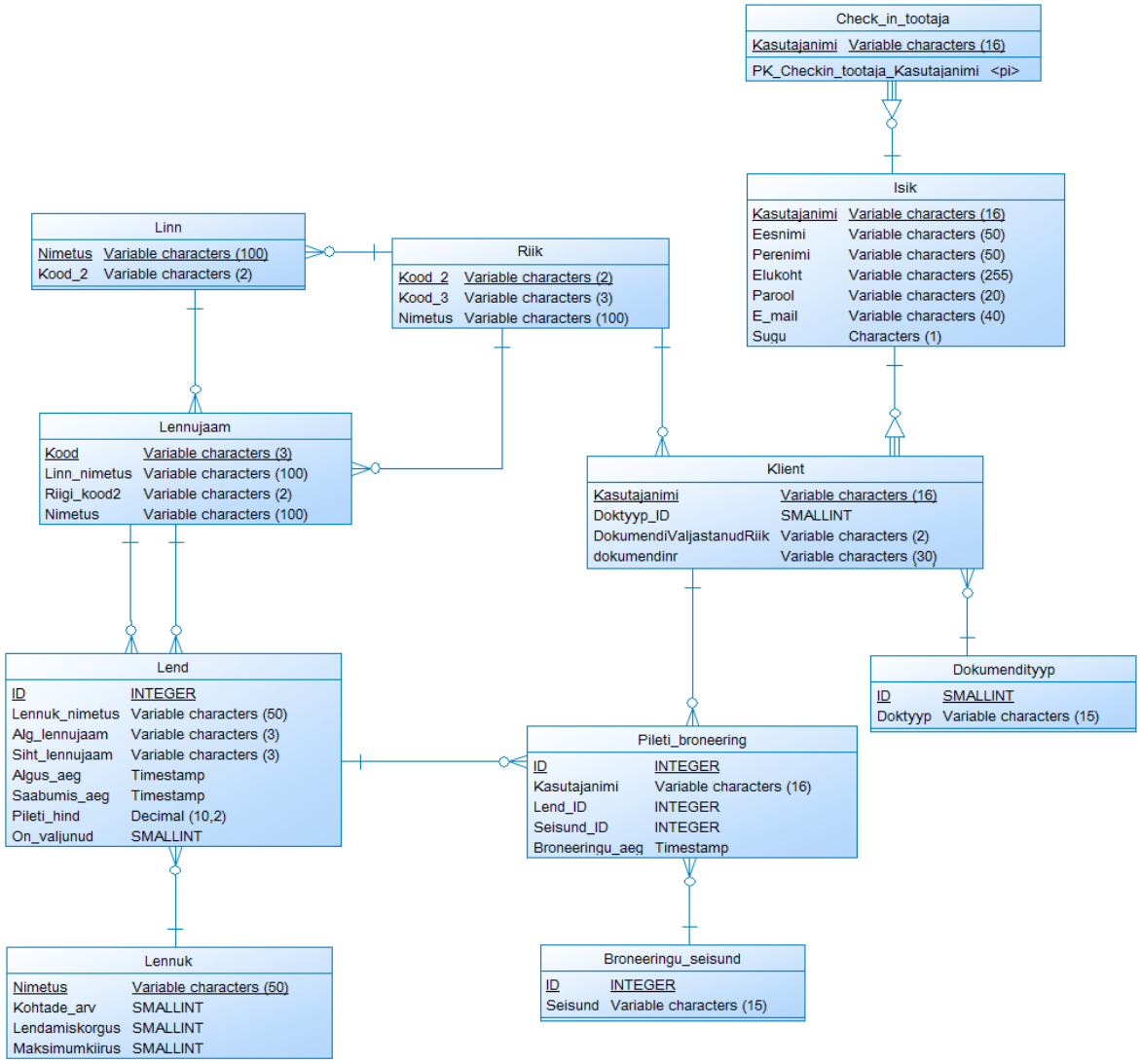
```
ALTER TABLE Lennujaam ADD CONSTRAINT FK_Lennujaam_Linn_Nimetus FOREIGN KEY
(Linn_nimetus) REFERENCES Linn (Nimetus)  ON DELETE NO ACTION ON UPDATE NO
ACTION;
ALTER TABLE Lennujaam ADD CONSTRAINT FK_Lennujaam_Riigi_kood2 FOREIGN KEY
(Riigi_kood2) REFERENCES Riik (Kood_2)  ON DELETE NO ACTION ON UPDATE NO
ACTION;
ALTER TABLE Klient ADD CONSTRAINT FK_Klient_Kasutajanimi FOREIGN KEY
(Kasutajanimi) REFERENCES Isik (Kasutajanimi)  ON DELETE NO ACTION ON
UPDATE NO ACTION;
ALTER TABLE Klient ADD CONSTRAINT FK_Klient_DokumendiValjastanudRiik
FOREIGN KEY (DokumendiValjastanudRiik) REFERENCES Riik (Kood_2)  ON DELETE
NO ACTION ON UPDATE NO ACTION;
ALTER TABLE Klient ADD CONSTRAINT FK_Klient_Doktyyp_ID FOREIGN KEY
(Doktyyp_ID) REFERENCES Dokumendityyp (ID)  ON DELETE NO ACTION ON UPDATE
NO ACTION;
```

Lisa 2. SQL-st genereeritud andmemudel



Joonis 29 SQL-st genereeritud füüsiline andmemudel PowerDesigner-s

Lisa 3. Füüsilisest andmemudelist genereeritud loogiline andmemudel



Joonis 30 Füüsilisest andmemudelist genereeritud loogiline andmemudel PowerDesigner-s