TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

PostgreSQL SQL lausete paindlikuks muutmiseks mõeldud infosüsteemi kavandamine

Magistritöö

Üliõpilane: Janina Voronova

Üliõpilaskood: 093056IAPM

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.

…… ……

(kuupäev) (lõputöö kaitsja allkiri)

PostgreSQL SQL lausete paindlikuks muutmiseks mõeldud infosüsteemi kavandamine

Annotatsioon

Käesoleva lõputöö eesmärgiks oli luua infosüsteem, mis võimaldab hallata PostgreSQL andmebaasi andmete lugemise ja muutmise SQL lausete koodi. See aitab kiirendada andmebaasi SQL lausete arendusprotsessi. Töö teine eesmärk oli luua infosüsteemi juurutamise ja kasutamise protsesside kirjeldused.

Selle töö kõige olulisem tulemus on PostgreSQL SQL lausete paindlikuks muutmiseks mõeldud infosüsteemi kavand ja prototüüp. Teine tulemus on infosüsteemi juurutamise ja kasutamise protsesside kirjeldus. Lisaks loodud infosüsteem oli rakendatud konkreetsele süsteemile. Töö tulemus on tegelikult üldlahendus, mida saab kasutada ka teiste andmebaasisüsteemide korral, mis täidavad teatud tehnilisi nõudeid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 136 leheküljel, 7 peatükki, 34 joonist, 12 tabelit.

Designing an Information System for the Flexible Modification of PostgreSQL SQL Statements

Annotation

The main aim of this degree work is to present an information system that would allow us to manage SQL statetements in a PostgreSQL database. The usage of the solution provided in this work helps us to improve the speed of development process. The second aim of this work was to describe the processes of implementation and usage of the information system.

The main result of this work is the design and prototype of the information system for the flexible modification of PostgreSQL SQL statements. Another result is the description of the processes for implementation and usage of the created information system. In addition the created information system was applied on a specific system. The result of the work is actually a general solution that can also be used in case of ohter database management systems that have certain technical characteristics.

The thesis is written in Estonian and contains 136 pages of text, 7 chapters, 34 figures, 12 tables.

Sisukord

Jooniste nimekiri

Tabelite nimekiri

Mõisted

Mõiste	**Mõiste ingl k**	**Selgitus**
Ärieskpert	Business expert	Inimene, kes omab põhjalikke teadmisi ettevõttes kehtivate reeglite kohta.
Arvutusreegel	Calculation rule	Tarkvarasüsteemi arvutusvalemit kirjeldav reegel
Otsustusreegel	Decision rule	Tarkvarasüsteemi sees tehtavate otsuste tegemise reeglid
Arendusprotsess	Development process	Tarkvarasüsteemi kavandamise ja realiseerimise protsess

Sissejuhatus

Iga tarkvarasüsteemi nõuded muutuvad aja jooksul. Need muudatused on vaja sisse viia ka tarkvara koodi. Kui muudatus tuleb ette liiga tihti, siis on oht, et tarkvara eest vastutavad inimesed ei jõua neid õigel ajal realiseerida (Mens, Guéhéneuc, Fernández-Ramil, ja D'Hondt, 2010).

Tarkvara nõuded võivad olla funktsionaalsed või mittefunktsionaalsed. Funktsionaalsed nõuded kirjeldavad soove süsteemi toimimise kohta. Mittefunktsionaalsed nõuded aga kirjeldavad süsteemi omadusi (missugune süsteem peab olema) (en.wikipedia.org, 2014). Selles töös käsitletakse just funktsionaalseid nõudeid.

Funktsionaalsed nõuded on seotud näiteks arvutuste ja andmete töötlusega (en.wikipedia.org, 2014). Nõuded on seotud reeglitega. Funktsionaalsed nõuded määravad, mis funktsioone peab tarkvarasüsteem täitma. Reeglid kirjeldavad, kuidas need funktsioonid peavad olema realiseeritud. Näiteks võib nõue olla järgmine: „tarkvarasüsteem peab arvutama kliendi tulu summa". Selle nõude realiseerimist kirjeldav reegel on „kliendi tulu summa arvutatakse summeerides kokku kõik kliendi sissetulekud" (en.wikipedia.org, 2014; Gelperin, 2009; blueprintsys.com, 2010).

Funktsionaalsed nõuded võivad koosneda keerulistest arvutusreeglitest, mis võivad muutuda näiteks ärikeskkonna muutumise tõttu. Reeglimuudatus võib olla vajalik ka arvutusvalemis leitud vea pärast. Kui arvutusreegel on keeruline või sõltub mingitest muutuvatest olukordadest, siis on suur tõenäosus, et hiljem tuleb seda arvutusreeglit muuta.

Iga reeglimuudatus on vaja sisse viia ka tarkvara koodi. Sellega kaasnev koodi muudatus võib olla väga väike, kuid koodi muutmine võib võtta palju aega, sest on vaja uuesti läbida paljud arendusprotsessi tegevused. Selles töös pakutakse lahendust, mis võimaldab teha programmi koodi paindlikumaks, et arendusprotsessi kiirendada.

Reeglid võivad olla realiseeritud erinevate programmeerimiskeeltega. Näiteks, kui on tegemist andmebaasi kasutatava (sealt andmeid lugevate ja andmeid muutva) rakendusega, siis mõned reeglid võivad olla realiseeritud rakenduse tasemel, kasutades näiteks JAVA või PHP keelt. Teised reeglid aga võivad olla realiseeritud andmebaasi tasemel SQL keele abil (Madis, 2010).

SQL on tänapäeval väga levinud andmebaasikeel. Käesolev töö on seotud SQL-andmebaase kasutavate (sealt andmeid lugevate ja andmeid muutvate) programmide paindlikumaks muutmisega. Üks osa sellisest programmist on kirjutatud näiteks PHP programmeerimiskeele abil ning kujutab endast kasutajaliidest, mis kasutab andmebaasi andmeid. Teine osa on realiseeritud andmebaasis ning koosneb SQL lausetest, mis on käivitatud rakenduse kasutajaliidese kaudu. Antud töös pakutav lahendus käsitleb just SQL keeles kirjutatud *andmete lugemise ja muutmise lausete* koodi paindlikumaks muutmist. Töö üldlahendus kujutab endast infosüsteemi, mis võimaldab hallata andmebaasis realiseeritud SQL lausete koodi. Infosüsteemi kaudu hallatavad SQL laused registreeritakse andmebaasis. Need laused võivad realiseerida reegleid. Infosüsteemi vahendusel SQL lausete muutmine tähendab ka reeglite realisatsiooni muutmist.

Tänapäeval on olemas ärireeglite haldamise süsteemid, mis annavad võimaluse hallata otsustusreegleid ning neid realiseerida. Ärireeglite haldamise süsteemi abil saab juhtida tarkvarasüsteemi otsustusreegleid, mille järgi tarkvara teeb järeldusi ning käitub vastavalt nendele järeldustele (en.wikipedia.org, 2013). Pakutava infosüsteemi vahendusel SQL lausete koodi muutmise kaudu on võimalik aga juhtida andmete haldamise reegleid, mis kirjeldavad andmete otsimise, lisamise, uuendamise ning kustutamise loogikat.

Antud töö võib pakkuda huvi andmebaasidega töötavatele inimestele (näiteks andmebaasi arendaja või analüütik), kes tegelevad sageli (näiteks kord kuus) muutuvate andmebaasi funktsioonidega. Töö üldlahendus annab võimalust kiiresti muuta sellise funktsiooni SQL lausete koodi ilma funktsiooni uue versiooni installeerimiseta. Töö võib esmajoones huvi pakkuda andmebaasi analüütikutele, kes kasutades üldlahendust saavad vastavalt muutunud reeglitele ise kiiresti muuta SQL lausete koodi. Arendajad võivad olla huvitatud lahenduse kasutamisest, sest see vähendab nende töö mahtu.

Töö teine eesmärk on esitada protsessidiagrammid, mis kirjeldavad töö üldlahenduse rakendamist.

Töö kolmas eesmärk on loodud üldlahendust rakendada ühes olemasolevas süsteemis. Süsteem realiseeritakse PostgreSQL andmebaasis ning on mõeldud PostgreSQL andmebaasi kasutavate rakenduste jaoks. Kuna PostgreSQLis ei saa luua protseduure kuid saab luua funktsioone, siis vastava protsessi kirjelduses viidatakse mõistele „andmebaasi funktsioon“.

Töö pakutav üldlahendus on kasutatav ka teistel platvormidel, kus saab andmebaasis luua funktsioone või protseduure.

Töö oodatav tulemus on infosüsteem SQL lausete koodi haldamiseks, mis koosneb järgmistest osadest.

1) PostgreSQL andmebaas SQL lausete koodi hoidmiseks

2) PHP keeles kirjutatud rakenduse prototüüp, mis võimaldab lisada ning muuta SQL lausete koodi

3) PL/pgSQL keele abil kirjutatud funktsioonid, mis on vajalikud infosüsteemis olevate SQL lausete kasutamiseks teistes süsteemides

4) Protsessimudelid, mis kirjeldavad infosüsteemi rakendamist ning integreerimist

1. Teoreetiline taust

Selles peatükis antakse ülevaade töö teoreetilisest taustast. Tutvustatakse mõisteid ja meetodeid, mille teadmine on kõnealuse töö tegemiseks ning mõistmiseks vajalik.

1.1. SQL keel

Struktuurpäringukeel (ingl k *Structured Query Language*, ehk SQL) on üks võimalik relatsioonandmebaasidega suhtlemiseks mõeldud keel (Limeback, 2008).

Ameerika Riikliku Standardite Instituudi (American National Standards Institute või ANSI) järgi on SQL standardkeel relatsiooniliste andmebaasisüsteemide (ingl k *Relational Database Management System* või RDBMS) jaoks (sqlcourse.com). SQL keele lauseid kasutatakse näiteks andmete otsimiseks või andmete uuendamiseks andmebaasis.

SQL keelt kasutavad andmebaasisüsteemid on näiteks Oracle erinevad süsteemid, PostgreSQL, Microsoft SQL Server, Microsoft Access, jne. Selle töö üldlahenduse realiseerimiseks on valitud avatud lähtekoodiga andmebaasisüsteem PostgreSQL.

1.1.1. SQL keele lausete ehitus

SQL laused pannakse kokku kasutades võtmesõnu (ingl k *keyword*), identifikaatoreid (ingl k *identifier*), spetsiaalmärke (ingl k *special characters*) ning konstante (ingl k *constant*) (Limeback, 2008) (postgresql.org).

Võtmesõnad on SQL-i standardis defineeritud sõnad, mida kasutatakse SQL lausete ehitamiseks.

Identifikaatorid on andmebaasi objektide (näiteks tabelid või veerud) nimed.

Konstandid on fikseeritud väärtusi esitavad literaalid.

Vaatame näiteks järgmist SQL lauset:

```
SELECT nimi FROM Klient WHERE klient_id = 7;
```

Selles näites „SELECT“, „FROM“ ning „WHERE“ on võtmesõnad. „Klient“, „nimi“ ja „klient_id“ on identifikaatorid. „Klient“ on tabeli nimi, ning „nimi“ ja „klient_id“ on

veergude nimed. „=“ märk on operaatori nimi (täisarvude võrdlemise operaator). „7“ on täisarvu seitse esitav konstant. „;“ on lause lõppu tähistav spetsiaalmärk.

Lisaks on SQL lause jagatud osadeks, mida nimetatakse osalauseteks (ingl k clause) (Limeback, 2008). Osalause nimi vastab võtmesõna nimele, millest osalause algas. Üleval esitatud näide koosneb siis järgmistest osalausetest:

- SELECT osalause:

```
SELECT nimi
```

- FROM osalause:

```
FROM Klient
```

- WHERE osalause:

```
WHERE klient_id = 7
```

On olemas erinevat tüüpi SQL lauseid (Limeback, 2008).

- DDL laused ehk andmekirjelduskeele (ingl k Data Definition Language) laused, mida kasutatakse andmebaasiobjektide haldamiseks
- DML laused ehk andmekäitluskeele (ingl k Data Manipulation Language) laused, mida kasutatakse andmebaasis olevate andmete haldamiseks
- TCL laused ehk transaktsioonide kontrolli laused (ingl k Transaction Control Language), mida kasutatakse andmebaasis transaktsioonide juhtimiseks.

DDL laused on: CREATE, ALTER ja DROP. On erinevaid arvamusi kas õiguste haldamiseks mõeldud GRANT ja REVOKE laused kuuluvad DDLi või eraldi alamkeelde – andmekontrolli keelde (ingl k *Data Control Language*).

DML laused on: SELECT, INSERT, UPDATE, DELETE ja MERGE (Limeback, 2008) (en.wikipedia.org, 2014).

TCL laused on: START TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT.

Selle töö raames käsitletakse ainult DML alamkeele lauseid. Vaatame neid detailsemalt.

SELECT lause otsib (pärib) andmed andmebaasist. Lause koosned kahest kohustuslikust osalausest SELECT ning FROM. Lisaks võib lause sisaldada WHERE, GROUP BY, HAVING, ORDER BY ning WITH osalauseid.

SELECT lause näide:

```
SELECT klient_id, nimi
FROM Klient
WHERE klient_id > 5
ORDER BY klient_id;
```

INSERT lause abil lisatakse üks või mitu rida andmebaasi tabelisse. Lause koosneb INSERT osalausest ning lisaks VALUES osalausetest või SELECT lause osalausetest.

Näiteks:

```
INSERT INTO Klient (klient_id, nimi)
VALUES (8, 'John');
```

või

```
INSERT INTO Klient(klient_id, nimi)
SELECT 8, nimi FROM Klient WHERE klient_id = 7;
```

UPDATE lause muudab andmeid andmebaasi tabelis olevates ridades. Lause koosneb UPDATE ning SET osalausetes. Lisaks võib ta sisaldada kõiki SELECT lause osalauseid.

Näiteks:

```
UPDATE Klient
SET nimi = 'Smith'
WHERE klient_id = 8;
```

DELETE lause eemaldab ühe või mitu rida andmebaasi tabelist. Lause sisaldab DELETE osalauset ning lisaks võib sisaldada kõiki SELECT lause osalauseid.

Näteks:

```
DELETE FROM Klient
WHERE klient_id = 7;
```

MERGE kombineerib INSERT, UPDATE ja DELETE lausete osalauseid. Kuna PostgreSQL hetkel (ver 9.3) seda lauset ei toeta, siis see töös käsitlemist ei leia.

1.1.2. Reeglite esitamine SQL koodis

Kasutades ülalnimetatud SQL lauseid on võimalik andmebaasi tasemel realiseerida vajalikud reeglid.

Näiteks on panga süsteemis vaja arvutada kliendi tulu viimase kuu jooksul. Selle jaoks on olemas arvutusreegel: „Kliendi tulu on kõikide tema sissemaksete summa viimaste 30 päeva jooksul“.

Panga andmebaasis on olemas tabel „Transaktsioon“, kus hoitakse andmeid kõikide panga klientide sisse- ja väljamaksete kohta (vt Tabel 1).

Tabel 1. „Transaktsioon“

Transaktsiooni_id	Kliendi_id	Tüüp	Kuupäev	Summa
1	1	1	12.04.2014	100
2	1	1	15.04.2014	980
3	1	2	20.04.2014	260
4	1	2	25.04.2014	47

Kui Tüüp = 1, siis on tegemist sissemaksega. Kui Tüüp = 2, siis on tegemist väljamaksega.

Konkreetse kliendi (antud juhul klient_id=1) tulu arvutusreegli realiseerib järgmine SQL lause:

```
SELECT SUM(T.Summa)
FROM Transaktsioon T
WHERE T.Kliendi_id = 1
AND T.Tüüp = 1
AND T.Kuupäev >= ADDDATE(CURRENT_TIMESTAMP(), INTERVAL -30 DAY);
```

Iga SQL keele osalause on reegli osa ehk alamreegel. Näiteks SELECT osalause ütleb, et on vaja liita kokku *summa* veerus olevad väärtused. FROM osalause ütleb, et on vaja arvutada transaktsioonide summad. WHERE osalause ütleb, et on vaja võtta arvesse ainult sissemaksed, mis tehti viimase 30 päeva jooksul.

Töö üldlahenduse eesmärk on anda võimalust hallata SQL lauseid, mis omakorda võimaldab juhtida tarkvarasüsteemi tööd suunavaid reegleid.

1.1.3. Andmebaasi funktsioonid

PostgreSQLi andmebaasisüsteemi puhul ei saa luua talletatud protseduure kuid saab luua funktsioone, mis täidavad sama ülesannet.

Talletatud protseduur (ingl k *stored procedure*) on alamprogramm, mis on rakendustele andmebaaside kasutamiseks kättesaadav (en.wikipedia.org, 2014).

PostgreSQLi puhul on talletatud protseduur funktsioon, mis ei tagasta mingeid väärtusi (wischner.blogspot.com).

PostgreSQL funktsioonide kirjutamiseks võib kasutada näiteks PL/pgSQL keelt. PL/pgSQL keel annab võimalus käivitada dünaamiliselt koostatud SQL lauseid. Dünaamilise SQLi lause kood koostatakse funktsiooni täitmise ajal ja seega see lause võib muutuda igal funktsiooni käivitamisel. Staatilises SQL lauses aga saab erinevatel käivitamistel muuta vaid lauses kasutatavaid konstante (docs.oracle.com).

Antud töös esitatud üldlahenduse realiseerimise jaoks kasutatakse PL/pgSQL keeles kirjutatud funktsioone, sest nende abil on võimalik koostada SQL lauseid süsteemi kasutamise ajal dünaamiliselt.

PostgreSQLi funktsioonid käivitavad SQL lauseid. Igas funktsioonis võib olla kasutatud rohkem kui ühte SQL lauset. Peale SQL lausete käivitamist võib funktsioon käivitada ka teisi funktsioone.

Funktsioone on võimalik käivitada käsitsi või rakenduse kaudu. Ühte funktsiooni on võimalik kasutada erinevates rakendustes (tutorialspoint.com).

Ülaltoodud SQL lause näidet on võimalik kasutada PostgreSQLi funktsiooni sees. Funktsiooni sisendiks on siis kliendi number ning väljundiks on kliendi tulu summa. All on toodud funktsiooni kood:

```
CREATE OR REPLACE FUNCTION arvuta_tulu_summa(kliendi_id integer)
RETURNS integer tulu_summa
AS $$

BEGIN
EXECUTE 'SELECT SUM(T.Summa)
            FROM Transaktsioon T
            WHERE T.Kliendi_id = $1
            AND T.Tüüp = 1
            AND T.Kuupäev >= ADDDATE(CURRENT_TIMESTAMP(), INTERVAL -30
DAY)'
INTO tulu_summa USING kliendi_id;
;
END;
$$ LANGUAGE plpgsql;
```

Dünaamilise SQLi kasutamisel on aga oht, et süsteemi võidakse rünnata SQL süstimise ründemeetodi abil. SQL süstimine (ingl k SQL injection) on SQL lausetega seoses tehtud rünnak, kui süsteemi poole pöördmise kasutatakse sisendeid (näiteks kirjutatakse väärtuseid kasutajaliidese väljadesse), mis sunnivad süsteemi sisemiselt käivitama SQL lauseid, mille käivitamine ei olnud programmi autorite poolt ette nähtud (Ganapathy, 2012). Näiteks, kui kasutajaliidesesse sisestatud väljade väärtused ei kontrollita, siis võib neid väärtuseid kasutav funktsioon pahatahtliku sisendi korral kustutada tabelist kõik andmed.

SQL süstimiste ohu vähendamiseks soovitatakse kõigepealt kontrollida kasutaja poolt sisestatud väärtuseid (Ganapathy, 2012). PL/pgSQL funktsioonides oleks vaja kasutada parameetritega lauseid EXECUTE (nagu ülaltoodud funktsioonis), kus parameetrid on esitatud $1, $2 jne. sümbolitega ning mida väärtutatakse kasutades USING osalauset (bobby-tables.com). PHP rakenduse puhul on abiks pg_query_params() funktsioon, mis võimaldab korraga käivitada ainult ühe SQL lause (PHP Documentation).

1.1.4. SQL lausete haldamise süsteemide näiteid

Eksisteerib erinevaid SQL lausete haldamise süsteeme, mis annavad võimaluse koostada ja salvestada SQL lauseid. Mõned näided on järgmised.

- *TSM Studio SQL Query and Macro Repository*, mis võimaldab luua ning salvestada SQL edasiseks kasutamiseks päringuid (spiritsoftware.biz).
- *Mifos SQL Query Repository*, mis on finantssüsteemi päringute kogum (mifosforge.jira.com).

- *Microsoft Server 2003 Active Directory* pakub võimalust koostada ja salvestada päringuid (web-foro.com).
- *SAS SQL Query Window* päringute koostamise võimalus erinevates andmebaasisüsteemides loodud andmebaaside põhjal (support.sas.com) .

SQL lausete haldamise süsteemid võimaldavad SQL päringute salvestamist ja taaskasutamist. Mõnedes süsteemideskirjutatakse päringud SQL keeles (näiteks TSM Studio SQL Query and Macro Repository ja Mifos SQL Query Repository). Teised süsteemid aga pakuvad ka kasutajaliidest, mille kasutamine ei nõua SQL keele teadmisi (näiteks Microsoft Server 2003 Active Directory konsool või SAS SQL Query Window).

Antud töös realiseeritav süsteem annab samamoodi võimaluse salvestada ning taaskasutada SQL lauseid. Kuid süsteemide erinevuse on selles, et olemasolevad SQL lausete haldamise süsteemid võimaldavad kasutada salvestatud päringud andmete lõppkasutajatele esitamiseks. Antud töö süsteem aga annab võimaluse kasutada salvestatud SQL lauseid mitte ainult andmete esitamiseks, vaid ka lisamiseks, muutmiseks või kustutamiseks. Antud töö süsteem on mõeldud selleks, et võimaldada kasutajatel muuta dünaamiliselt andmebaasi funktsioonide ja selle kaudu ka neid funktsioone kasutavate rakenduste toimimist.

1.2. „Minimeeri koodi, maksimeeri andmeid“ põhimõte

Kenneth Downs (2008) seletab tema väitel kõige rohkem saladuses hoitud programmeerimise printsiipi „minimeeri koodi, maksimeeri andmeid“ (ingl k *minimize code, maximize data*). Selle printsiibi põhimõte on arvutusreeglite salvestamine andmebaasi tabelites, mis vastab „maksimeeri andmeid“ põhimõttele. Sellisel juhul ei ole enam vaja programmeerida arvutusreeglid koodi sisse, mis vastab „minimeeri koodi“ põhimõttele.

Idee on selles, et koostatakse reeglite tabel, kus on kirjeldatud teatud parameetrite väärtused. Näiteks kui on tegemist toidupoega, kus on olemas iga müüdava toote jaoks selline reegel: „Kui pood müüs A tooteid eelmise kuu jooksul 30% rohkem kui üleeelmise kuu jooksul, siis tuleb vähendada A toode hinda 10% võrra, kuid lõpphind ei tohi olla alla 5“. Sarnane reegel on olemas ka teiste toodete kohta. Sellisel juhul võib teha järgmise reeglite tabeli (vt Tabel 2):

Tabel 2. Toidupoe reeglite näidistabel

Toode	Müüdud_koguse_protsendid	Hinna_alandamise_protsendid	Minimaalne_hind
A	30	10	5
B	25	5	7
C	15	20	30

Sellisel juhul näeb kirjutatud programm välja nagu tsükkel, mille sees vaadatakse läbi kõik tooted ja rakendatakse igaühele vastavat reeglit.

Teiselt poolt oleks seda loogikat võimalik realiseerida ka kirjutades iga toote jaoks kõik reeglid programmikoodi sisse. See võib olla küll esmapilgul lihtsam viis probleemi lahendamiseks, kuid siis on programmikood pikem. Eriti tekivad probleemid siis, kui on vaja mingit reeglit muuta. Siis on vaja koodist konkreetset reeglit otsida. Kuid reeglite tabeli kasutamisel võib valida soovitava reeglit ning muuta selle konkreetset väärtust. Sellisel juhul hakkab reegel töötama uue loogika järgi. Taolise muudatuse võib teha programmi ärikasutaja ilma arendaja abita. Seega Kenneth Downs leiab kokkuvõttes, et „minimeeri koodi, maksimeeri andmeid“ printsiip omab väga otsest mõju kasutajakogemusele.

Samuti märgib Downs, et isegi siis, kui arendajad mõtlevad programmi kavandamisel või kirjutamisel koodi minimiseerimisest, võivad nad jõuda liiga keerulise lahenduseni. See

juhtub, sest nad unustavad andmete maksimeerimise. Kui jälgida printsiibi mõlemat osa, siis tuleb võtta äriloogika reeglid koodist välja ning panna reeglite tabelisse. See lihtsustab ja minimeerib koodi ja samal ajal võimaldab kergemini sisse viia muudatusi äriloogikas reeglite tabeli sisu uuendamise kaudu.

Veel üks „minimeeri koodi, maksimeeri andmeid“ printsiibi rakendamisest saadav kasu on Kenneth Downsi järgi vigade otsimise lihtsustumine. Reeglite tabeleid on võrreldes programmi koodiga kergem lugeda, neist on kergem aru saada ning neid on lihtsam analüüsida.

Kenneth Downs teeb kokkuvõte, et „minimeeri koodi, maksimeeri andmeid“ põhimõtte rakendamisel on positiivne mõju nii koodi kirjutamisele kui ka vigade otsimisele, haldamisele ning kasutajakogemusele.

Järgmisena käsitletakse selles töös mõnd võimalust reeglite esitamiseks SQL-andmebaasi tabelites, mis võimaldab rakendada „minimeeri koodi, maksimeeri andmeid“ põhimõtet.

1.2.1. Reeglite esitamine andmetes

Giles (2012) kirjeldab oma raamatus reeglite andmetes (ingl k *rules-in-data*) esitamise lähenemist. Samamoodi nagu Kenneth Downs, pakub ka Giles, et reeglite hoidmiseks võib kasutada ülaltoodud näitega sarnaseid tabeleid.

Lisaks märgib Giles, et kui reeglid on programmeeritud koodis ning kui reeglite sisu peaks tihti muutuma, siis nende muudatuste koodi sisse viimine ei ole kerge. Kuid kui reeglid esitatakse tabelina, siis on ärikasutajatel lihtne muuta reeglite sisu, uuendades vastavat tabelit ning koodimuudatuste vajadust ei ole. Reeglite esitamine tabelites nõuab küll natuke rohkem tööd andmebaasi disainerilt ning arendajal on vaja programmeerida reegli üles otsimine, kuid ideaalis pole arendajal ega ka disaineril vaja tulevikus reegli muutmise protsessis enam osaleda.

Vaatame detailsemalt erinevaid variante, millised võiksid olla reeglite hoidmiseks mõeldud tabelid.

1.2.1.1. Otsustustabelid

Gilesi (2012) sõnul seisneb otsustustabelite (ingl k *decision table*) idee selles, et ärikasutajatel on võimalus mõista ja hallata ärireegleid, ning programm oskab teha otsustustabelite sisu põhjal otsuseid.

Näiteks toidupoe süsteemis kliendile soodustuse määramiseks kehtivad järgmised reeglid:

- kui ostusumma< 15 eurot, siis soodustus = 0%
- kui ostusumma >= 15 eurot ja ostusumma <20, siis soodustus = 5%
- kui ostusumma >= 20 eurot, siis soodustus = 10%

Tabel 3 näitab kuidas võiks välja näha ülalnimetatud reegli kontrolli realiseerimist lihtsustav tabel.

Tabel 3. Kliendi soodustuse määramise reeglid

Tingimus	**Tulemus**
Minimaalne ostusumma (eurodes)	Soodustus (protsentides)
0	0
15	5
20	10

Tavaliselt on otsustustabelil olemas „tingimus“ (ülaltoodud näite puhul ostusumma) ning „tegevus“ või „tulemus“ (soodustus). Tulemus võib olla nii tõeväärtuse tüüpi kui ka mingi diskreetne väärtus (näiteks 5, mis tähendab kliendi ostusumma alandamine 10% võrra).

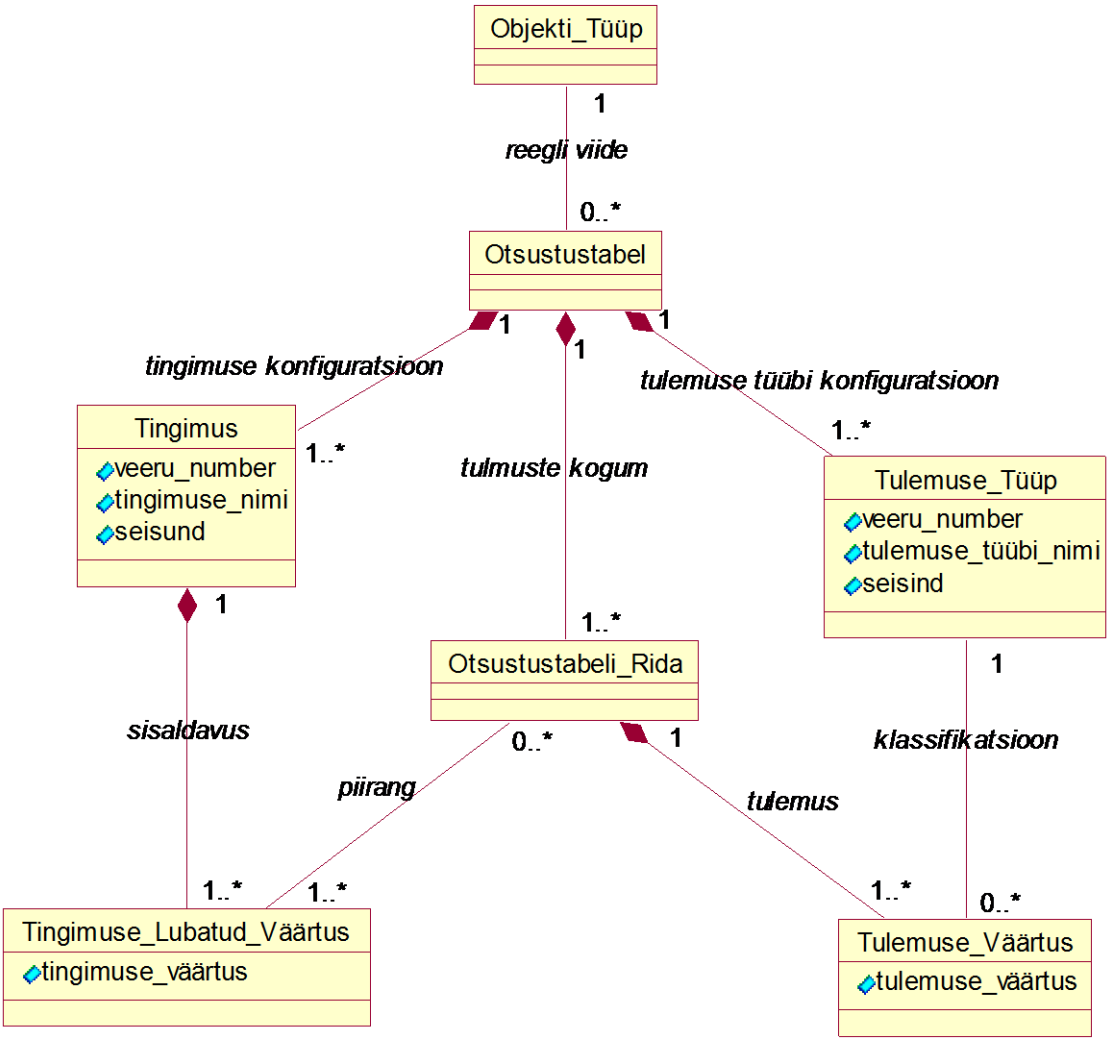
„Tingimuse“ ja „Tulemuse“ osad võivad koosneda ka rohkem kui ühest veerust. Näiteks võib ülaltoodud näidet täiendada nii, et soodustus sõltub mitte ainult ostusummast, vaid ka kliendikaardi olemasolust (kas kliendil on olemas kliendikaart või mitte). Samuti peame nüüd tulemuseks saama mitte ainult soodustuse väärtused, vaid ka info selle kohta, kas kliendil peab olema lisasoodustus 5% või mitte. Otsustustabel, mille järgi reeglid esitatakse, võiks välja näha nagu Tabel 4.

Tabel 4. Soodustuse määramiseks otsustustabeli näide

Tingimus		**Tulemus**	
Minimaalne Ostusumma (eurodes)	Kas kliendikaart on olemas?	Soodustus (protsentides)	Kas peab olema lisasoodustus?
0	FALSE	0	FALSE
0	TRUE	0	FALSE
15	FALSE	5	FALSE
15	TRUE	5	TRUE
20	FALSE	10	FALSE
20	TRUE	10	TRUE

Selliselt näeb välja fikseeritud struktuuriga otsustustabel. See on abiks, kui reeglite tingimused (väärtused) tihti muutuvad. Kuid kui muutub ka reeglite struktuur (näiteks tingimustesse uusi atribuute või ilmneb vajadus mõne senini püsivaks peetud reegli osa varieerimiseks (näiteks lisasoodustus ei ole enam ühesugune)), siis tuleb kasutada kohandatavat otsustustabelit.

Giles (2012) pakub metaandmetel põhineva otsustustabeli jaoks järgmise mudeli (Joonis 1).



Joonis 1. Metaandmetel põhineva otsustustabeli kontseptuaalmudel (Giles, 2012)

Sellise mudeli järgi on võimalik ehitada paindliku struktuuriga otsustustabeleid. See on kasulik, kui on olemas mitu erineva struktuuriga otsustustabelit, või kui mingi otsustustabeli struktuur muutub tihti.

Tulemusena saadud otsustustabelid vastavad järgmistele tingimustele.

- Igal otsustustabelil võib olla üks või rohkem tingimust („Tingimus“ mudelil), ning iga tingimus võib sisaldada erinevaid väärtuseid („Tingimuse_Lubatud_Väärtus“ mudelil).
- Igal otsustustabelil võib olla üks või rohkem tulemust („Tulemuse_Väärtus“), ning iga tulemus võib olla erinevat andmetüüpi („Tulemuse_Tüüp“).

- Iga otsustustabel võib olla seotud ainult ühe objekti tüübiga („Objekti_Tüüp“ mudelil), näiteks *ost* või *tellimus*.
- Otsustustabelis ei pea olema kõiki võimalikke kriteeriumite kombinatsioone – otsustustabel ei pea olema lõpuni täidetud.

Samas ütleb Giles (2012), et kui tingimused ning tulemuste tüübid on stabiilsed, siis on parem selle mudeli kasutamist vältida (kuna koos paindlikkusega suurendab see ka keerukust) ning kasutada fikseeritud struktuuriga otsustustabelit.

1.2.1.2. Arvutusreeglite esitamine andmetes

Otsustustabelid on kasulikud kui reeglite kriteeriumid ning nende võimalikud väärtused on konkreetselt defineeritud. Kuid need ei sobi juhul, kui reeglis on kasutuses arvutuslikke valemeid. Kui valemid on staatilised, siis on paremaks lahenduseks programmeerida need koodis. Kui aga valem ise aja jooksul muutub, siis oleks kasulik esitada sellist valemit andmetena. Teiste sõnadega, reeglite tabelis esitatakse valemi osad, kuid mitte muutujate (tingimuste) väärtused.

Näiteks poe süsteemis sõltub toote hind (TH) järgmistest muutujatest: toote ostuhind (OH), toote kogus laos (KL), toote müügi kogus eelmise kuu jooksul (MK), toote müügilt soovitud tulu (ST).

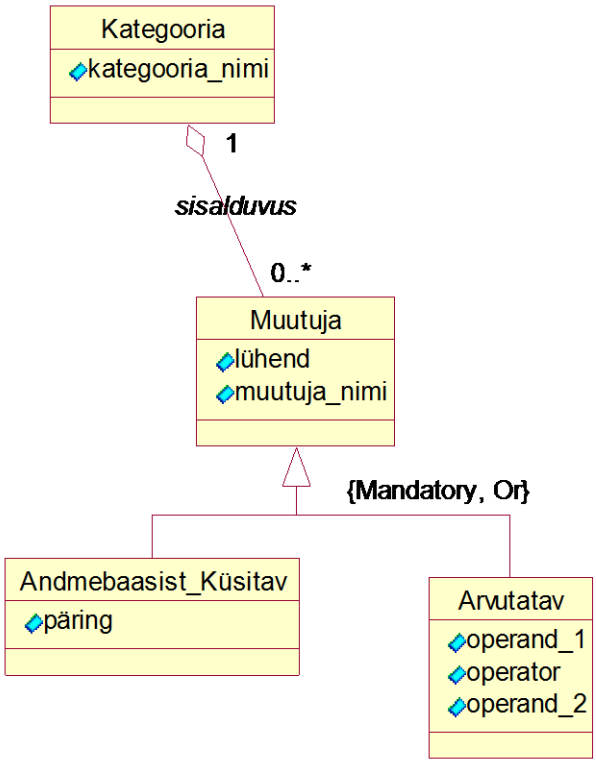
Süsteemis on iga tootekategooria jaoks oma reeglid, et toodehinda arvutada.

Kategooria A toodehindade näidisvalem on järgmine:

$$TH = OH \times (KL/MK) + ST\ (KL/MK)$$

Kui poes on mitu erinevat kategooriat ning iga kategooria arvutusloogika on erinev, siis iga kategooria valemi koodis esitamine võib olla küllaltki mahukas. See probleem ilmneb eriti teravalt siis, kui pood võtab tihti kasutusele uusi kategooriaid või ilmneb vajadus vanade kategooriate valemeid korrigeerida. Sellisel juhul on küllaltki ebamugav iga kord programmikoodi muuta.

Joonis 2 esitab mudeli, mille rakendamine aitab seda probleemi lahendada (Giles, 2012).



Joonis 2. Arvutusreeglite andmetes esitamise kontseptuaalmudel

Antud mudelis on „Andmebaasist_Küsitav“ andmebaasist küsitava väärtusega muutuja ning „Arvutatav“ on arvutatava väärtusega muutuja.

Selle mudeli järgi on võimalik ilma programmikoodi muutmiseta defineerida uusi ning muuta olemasolevaid tootekategooriate muutujate väärtuste arvutamise valemeid.

Igal kategoorial on mitu valemi muutujat, mille väärtus on vaja arvutada (Arvutatav) või küsida süsteemi andmebaasist (Andmebaasist_Küsitav). Selle disaini järgi saab koostada arvutusloogikat kirjeldava tabeli (Tabel 5).

Tabel 5. Kategooria A toodehindade arvutus

kategooria_nimi	muutuja_number	muutuja_tüüp	lühend	päring	operand_1	operaator	operand_2
A	1	Andmebaasist_ Küsitav	OH	SELECT ostuhind AS OH FROM Toode			

kategooria_nimi	muutuja_number	muutuja_tüüp	lühend	päring	operand_1	operaator	operand_2
A	2	Andmebaasist_Küsitav	KL	SELECT kogus_laos AS KL FROM Toode			
A	3	Andmebaasist_Küsitav	MK	SELECT myygi_kogus AS MK FROM Toode			
A	4	Arvutatav	KL/MK		KL	/	MK
A	5	Arvutatav	OH * (KL/MK)		OH	*	KL/MK
...	6						

Sarnaselt võib välja näha ka ärikasutajatele nähtav kasutajaliides, mille kaudu nad saaksid valemeid hallata.

Selle tabeli järgi töötab süsteem nii, et see vaatab muutuja numbri alusel järjekorras läbi kõik muutujad. Leitud muutuja väärtus võib olla sisendiks mõne järgmise muutuja väärtuse leidmisele. Kui Muutuja tüüp on „Andmebaasist_Küsitav“, siis muutuja väärtust päritakse süsteemi andmebaasist. Kui Muutuja tüüp on „Arvutatav“ ja veerud „Operand 1“, „Operaator“ ning „Operand 2“ on täidetud, siis leitakse muutuja väärtus etteantud valemi abil kasutades etteantud operande.

1.2.1.3. Vastavustabelid

Vastavuste leidmine (ingl k *mapping*) on protsess, mille käigus seotakse erinevate süsteemide mõisted (nccc.uow.edu.au, 2012). Seega vastavustabel (ingl k *mapping table*) sisaldab seoseid erinevate süsteemide mõistete vahel.

Vastavustabelite kasutamine on üks viis reeglite andmetes esitamiseks. Need on abiks, kui reegel kujutab endast mingit seost kahe või enama väärtuse vahel.

Näiteks on poe süsteemis vaja siduda tooted ning vastavad toodete kategooriad. Iga toode võib sisalduda rohkem kui ühes kategoorias. Tabel 6 esitab näiteväärtuseid toodete ja kategooriate vastavustabelis.

Tabel 6. Toodete nimetuste ja triipkoodide vastavustabeli näide

Toode_id	Kategooria_id
123	1
123	2
256	1
256	2
896	3

1.3. Ärireeglite haldamise süsteem

Äriloogika on tarkvarasüsteemi kontekstis reeglid ja algoritmid, mis kirjeldavad informatsiooni vahetust andmebaasi ja kasutajaliidese vahel (investopedia.com). Ärireeglite haldamise süsteemi kontekstis kirjeldab äriloogika, kuidas ettevõte tuletab faktidest järeldused (von Halle, Goldberg, 2009). Teisisõnu, see kirjeldab ettevõte organisatsioonilisi otsuseid.

Ettevõtte ärireeglid on aluseks ettevõtte tarkvarasüsteemide äriloogika loomisele (pic.dhe.ibm.com). Ärireeglite haldamise süsteemi kontekstis väljendab ärireegel äriloogika tingimusi ning tulenevaid järeldusi. Tingimuste järgi kontrollitakse olemasolevaid fakte ning järeldatakse uusi fakte (von Halle, Goldberg, 2009). Ärireegli näide on: „kui kliendi ostusumma on rohkem kui 20 eurot, siis teha soodustus 10%“. Kus „ostusumma“ on fakt, „rohkem kui 20 eurot“ on tingimus, ning „soodustus 10%“ on järeldusena tehtud uus fakt.

Ärireeglite haldamise süsteem (ingl k „Business Rule Management System“ või BRMS) on süsteem, mis on mõeldud äriloogika defineerimiseks, juurutamiseks, käivitamiseks, jälgimiseks ja haldamiseks (en.wikipedia.org, 2013). Ärireeglite haldamise süsteem võimaldab äriekspertidel (inimesed, kes vastutavad ärireeglite eest) defineerida ning hallata ärireegleid. Ärireeglid omakorda defineerivad otsused, mis juhtivad organisatsiooni süsteemi käitumist (ibm.com).

Igal organisatsioonil on oma loogika, mida kasutatakse tegevus- ja juhtimisülesannete täitmiseks (Chisholm, 2003). Tavaliselt on see loogika kirjeldatud erinevates dokumentides, millest mõned võivad olla kadunud või vananenud. Koodist võib olla raske välja otsida ja selgusele jõuda, mis koodi osad vastavad millisele reeglile. See võib põhjustada erinevusi reeglite realisatsiooni ja reeglitest arusaamise vahel, mis viib ebakorrektse realisatsioonini. Kui ilmneb vajadus teostada muudatusi suurte organisatsioonide infosüsteemi tarkvaras, siis

eelnevast tulenevalt võivad tekkida probleemid. Ärireeglite haldamise süsteem aitab kui mitte likvideerida, siis vähemalt vähendada kirjeldatud loogika ja tegeliku realisatsiooni erinevuse riski, kuna kõiki reegleid hallatakse ühes ja samas süsteemis.

On olemas veel selline mõiste nagu *ärireeglite mootor* (ingl k „Business Rules Engine“ või BRE). See võib olla iga süsteem, mis kasutab ükskõik millisel kujul reegleid, mis võivad olla rakendatud andmetele (docs.jboss.org). Organisatsiooni tarkvarasüsteemid saavad kasutada reeglite mootori kaudu käivitatud ärireeglite tulemusi.

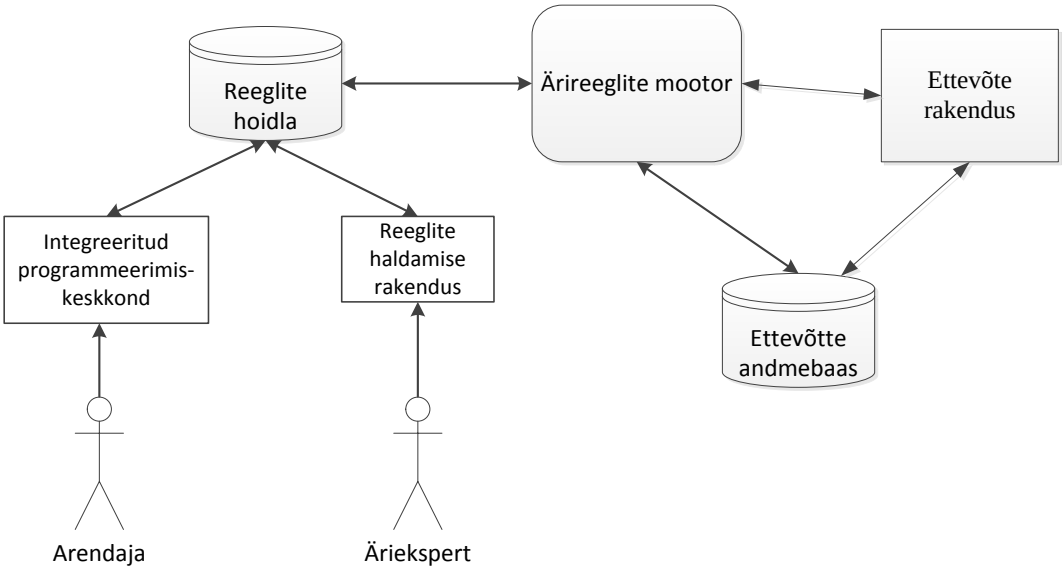
Seega on ärireeglite mootor (BRE) ärireeglite haldamise süsteemi (BRMS) tuum. Kuid lisaks sellele sisaldab BRMS ka ärikasutaja jaoks mõeldud tööriistu, mis võimaldavad reegleid defineerida ning neid definitsioone muuta ja analüüsida.

Selle töö üldlahendus on sarnane ärireeglite haldamise süsteemiga, sest need mõlemad võimaldavad kasutajatel mõjutada tarkvarasüsteemi loogikat. Põhierinevus on selles, et ärireeglite haldamise süsteemis hallatakse tarkvarasüsteemi otsustusreegleid, mis on kirjeldatud iga süsteemi jaoks spetsiifilisel viisil. Näiteks OpenRules ärireeglite haldamise süsteemis kirjeldatakse reeglid kasutades otsustustabeleid (openrules.com). Süsteemis kirjeldatud reeglite realisatsioon toimub automaatselt ärireeglite mootoris. Selles töös realiseeritavas süsteemis aga hallatakse SQL keele lauseid, mis on iseenesest reeglite realisatsioon.

1.3.1. Ärireeglite haldamise süsteemi arhitektuur

Ärireeglite haldamise süsteem koosneb vähemalt järgmistest komponentidest (Byron, 2010) (vt Joonis 3):

- Reeglite hoidla (ingl k “Rules Repository”) reeglite hoidmiseks.
- Tööriistad: integreeritud programmeerimiskeskkond (ingl k „Integrated Development Environment“ ehk IDE) arendajatele ning reeglite haldamise rakendusprogramm (ingl k „Rule Management Application“) äriekspertidele.
- Käituskeskkond, mis sisaldab ärireeglite mootorit.



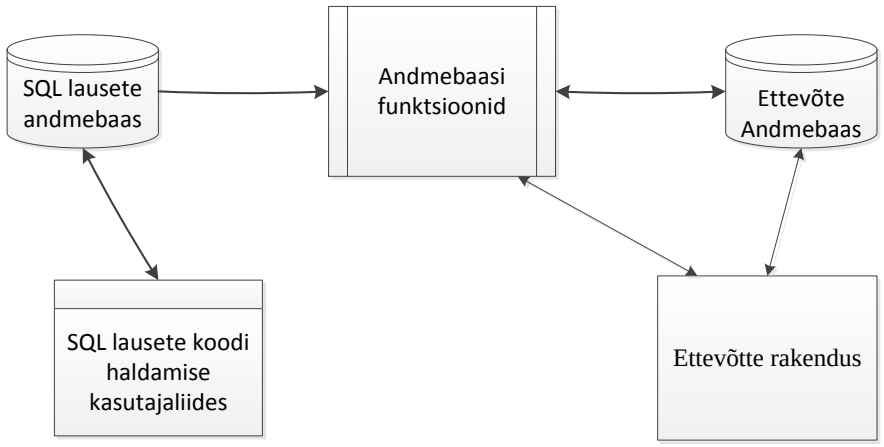
Joonis 3. Ärireeglite haldamise süsteemi arhitektuur

Kõiki ärireegleid hoitakse reeglite hoidlas, mis on tegelikult üks andmebaas. IDE tööriista abil loovad arendajad reeglite haldamiseks vajaliku raamistiku, mis sisaldab reeglite malle ja käivitamisstsenaariume.

Reeglite haldamise rakendusprogrammi abil loovad ärikasutajad ärireeglite spetsifikatsioone ning teevad nendes muudatusi. Need spetsifikatsioonid salvestatakse reeglite hoidlas.

Läbi reeglite mootori saab ettevõte infosüsteemi tarkvara langetada otsused, mis põhinevad loodud ärireeglitel (Chhatpar, 2008). Chisholm (2003) järgi on reeglite mootor tarkvara, mis on võimeline defineeritud reegleid käivitama.

Selles töös loodud üldlahenduse arhitektuur põhineb ärireeglite haldamise süsteemi arhitektuuril (Joonis 4).



Joonis 4. Töö üldlahenduse üldine arhitektuur

Reeglite hoidla asemel on andmebaas, kus hoitakse SQL laused. On olemas kasutajaliides (rakendusprogramm), mis võimaldab SQL lausete haldamist. Ärireeglite mootori asemel on andmebaasi funktsioonid, mis võimaldavad käivitada andmebaasis salvestatud SQL lauseid.

1.3.2. Ettevõte infosüsteemi arendusprotsess ärireeglite haldamise süsteemi kasutamise korral

Ärireeglite haldamise süsteemi ettevõttes kasutamine muudab selle ettevõtte infosüsteemide arendusprotsessi. Sellisel juhul lisanduvad protsessi tegevused, mille kohaselt loob arendaja reeglite mallid, mille järgi ärianalüütikud sisestavad süsteemi uued reeglid, muudavad olemasolevaid või muudavad reegleid kehtetuks (hartmannsoftware.com, 2012). Ärianalüütik saab ühe malli järgi kirjeldada rohkem kui ühte reeglit. Arendaja töö maht väheneb, kuna temalt on ära võetud ärireeglite rakendamise tarkvaralise realiseerimise kohustus.

Uue malli loomisel (või olemasoleva malli muutmisel) teeb arendaja uue malli arenduskeskkonnas valmis. Edasi installitakse see süsteemi muudatus testkeskkonda, kus testija seda testib. Ning lõpuks, kui testid on edukad olnud, siis installitakse mall toodangu keskkonda.

Uue reegli loomisel aga on protsess lühem, kuna jääb vahele arendusfaas arenduskeskkonnas. Ärianalüütik loob reegli, mida testitakse testkeskkonnas. Testid viiakse

läbi ärianalüütiku ja testija poolt. Lõpuks, kui kõik testid on edukalt läbitud, aktiveerib ärianalüütik reegli toodangu keskkonnas.

1.3.3. Ärireeglite haldamise süsteemi plussid

Ärireeglite haldamise süsteemist saadav kasu on järgmine (en.wikipedia.org, 2013).

- Arendaja töö vähendamine, sest ärikasutaja (ärianalüütik) saab ise muuta tarkvarasüsteemis rakendatavaid reegleid.
- Ärireeglite tarkvaralise rakendamise realiseerimise kiirenemine, sest väheneb arendaja töömaht.
- Suurenenud kontroll realiseeritud äriloogika üle, kuna reeglite realisatsioone haldavad ärieksperdid.
- Võimalus täpsemalt väljendada otsustusloogikat, kasutades ärisõnastikku ning graafilisi reeglite esitamise viise (otsustustabelid, puud, vooskeemid).
- Ärireeglite haldamise süsteem aitab teha tarkvarasüsteeme, mis ärireegleid paremini arvesse võtavad (ibm.com).
- Rakenduse haldamise maksumus väheneb tänu sellele, et väheneb nõuete hulk, mida tuleb arenduse käigus realiseerida (Taylor, 2006).

Kuid tõeline kasu ärireeglite haldamise süsteemidest on Byron (2010) järgi erinevate reeglite mootorite kasutamisest tulenevate probleemide lahendamine. Kui organisatsioonis on võimalik lõpuks kasutada ainult ühte ärireeglite haldamise süsteemi, mis ühendab endas reeglid erinevate rakenduste reeglite mootoritest, siis see aitab vähendada reeglite erinevust organisatsiooni erinevate osade vahel.

1.3.4. Ärireeglite haldamise süsteemide kasutamisega seotud probleemid

Ärireeglite haldamise süsteemide kasutamisega seotud probleemid on järgmised (hartmannsoftware.com, 2012).

- Süsteemi kavandamiseks ja realiseerimiseks on vaja spetsiifilisi teadmisi. Tuleb teha analüüs ärireeglite haldamise süsteemi olemasoleva süsteemiga integreerimiseks.
- IT-poole arendajad peavad olema ikkagi aeg ajalt reeglite haldamise protsessis kaasatud, näiteks selleks, et kirjeldada uusi reeglite malle.

- Üleminek ühe ärireeglite haldamise süsteemi kasutamiselt teise süsteemi kasutamisele võtab palju arendustööd, aega ning raha.

1.3.5. Olemasolevad ärireeglite haldamise süsteemid

Järgnevalt on nimetatud mõned näited olemasolevatest ärireeglite haldamise süsteemidest.

- OpenRules on avatud lähtekoodiga ärireeglite haldamise süsteem, mis pakub reeglipõhiste veebirakenduste arendamiseks erinevaid komponente (openrules.com). See võimaldab äriekspertidel muuta veebivorme ning luua nõustussüsteeme.
- JBoss Drools on ärireeglite haldamise süsteem koos otsustuspõhise reeglite mootoriga (en.wikipedia.org, 2013).
- IBM ILOG JRules on ärireeglite haldamise süsteem, mis annab kasutajatele kontrolli automatiseeritud äriotsuste üle (pic.dhe.ibm.com).

1.3.6. Antud töö raames realiseeritava infosüsteemi erinevus ärireeglite haldamise süsteemist

Selles töös realiseeritav SQL lausete haldamise infosüsteem on sarnane ärireeglite haldamise süsteemiga. Mõlemad võimaldavad kasutajatel mõjutada ärireeglite realisatsiooni. Ärireeglite haldamise süsteemis haldab kasutaja ärireeglid, mis on realiseeritud reeglite mootori abil. SQL lausete infosüsteemis aga haldab kasutaja SQL lauseid, mis realiseerivad ärireegleid.

Ärireeglid on loogilised „kui-siis“ laused, mis väljendavad ettevõte otsuste tegemise loogikat (ibm.com).

SQL laused aga võimaldavad realiseerida ka teisi tüüpi äriloogika reegleid, mitte ainult otsustusreeglid. Näiteks on SQL lausete abil võimalik küsida andmebaasist vajalikku informatsiooni. Näiteks reegli „rakenduse kasutajale tuleb tagastada andmed kõigi klientide kohta, kelle viimase kuu ostusumma on vähem kui 5 eurot“ on võimalik realiseerida SQLi SELECT lausega. Samuti on SQL lausetega võimalik uuendada, lisada või kustutada andmebaasis olevaid andmeid. Teiste sõnadega, SQL lausete abil on võimalik realiseerida andmete haldamise (otsimise, muutmise, lisamise, kustutamise) loogikat. Seega võimaldab SQL lausete infosüsteem juhtida andmete haldamise ärireegleid.

1.4. Tarkvara evolutsioon

Kuna nõuded tarkvarale muutuvad tänu muutustele ärikeskkonnas sealhulgas seadusandluses ning tehnilistes platvormides, siis on tarkvara evolutsioon väga oluline teema (Mens, Guéhéneuc, Fernández-Ramil, D'Hondt, 2010). Tarkvara evolutsioon tähendab olemasoleva tarkvarasüsteemi täiustamist, lisades uut funktsionaalsust, muutes olemasolevat funktsionaalsust või eemaldades mõne funktsionaalsuse.

Tarkvara evolutsiooni mõistega on seotud tarkvara hooldus. Tarkvara hooldus hõlmab vigade parandusi, kasutajatuge, testimist ning teisi protsesse, mis toetavad tarkvara igapäevast kasutamist (Mens, Guéhéneuc, Fernández-Ramil, D'Hondt, 2010). Tarkvara evolutsiooni protsessi alustatakse tarkvara hoolduse protsessi käigus saadud tagasiside tulemusena (Lehman, 1997).

1970-ndatel aastatel tutvustas Meir M. Lehman tarkvara evolutsiooni reegleid (Lehman, 1997). Lehmani järgi on E-tüüpi tarkvarasüsteem aktiivselt kasutatav tarkvarasüsteem, ilma milleta igapäevaseid protsesse poleks võimalik teostada (Karch, 2011). Esimese reegli järgi peab iga E-tüüpi tarkvarasüsteemi kasutajate üha suurenevate nõudmiste rahuldamiseks jätkuvalt täiendama.

Tarkvara evolutsiooni teine reegel ütleb, et tarkvara evolutsiooni käigus suureneb selle keerukus, kui ei ole tehtud tarkvara keerukuse kontrolli all hoidmiseks või vähendamiseks vajalikku tööd. Tarkvara keerukus tuleneb sellest, et uue funktsionaalsuse lisamisel tehakse iga kord uued muudatused vanade muudatuste peale. Kui tarkvara suurenevat keerukust ei hallata, siis tarkvara evolutsiooni jaoks vajalike tegevuste raskus samuti suureneb.

Tarkvara keerukust aitab vähendada tarkvara refaktoreerimine. Tarkvara refaktoreerimine on tarkvara koodi muutmise protsess, mille eesmärk on tarkvarasüsteemi struktuuri parandamine. Tulemusena ei muutu süsteemi funktsionaalsus kuid süsteemi struktuur on paremini arusaadav ning lihtsamini muudetav. Refaktoreerimise mõiste selgitatakse detailsemalt jaotises 1.4.1.

Ärireeglite haldamise süsteemi kasutamine lihtsustab tarkvara evolutsioonilist muutmist (evolutsioneerimist), sest see võimaldab lisada tarkvarasüsteemi uut funktsionaalsust ärireegleid muutes, lisades või kustutades.

Veel üheks tarkvara evolutsiooneerimist hästi võimaldanud süsteemi ajalooliseks näiteks on infosüsteemide genereerimise süsteem GENSI (Õunapuu, 1988). See süsteem võimaldas hallata ettevõtte alamsüsteemide realisatsioone. Süsteemi kasutaja muudab ettevõtte alamsüsteemi realisatsiooni GENSI süsteemis esitatud kirjelduste (spetsifikatsioonide) muutmise abil. Õunapuu järgi on üks GENSI süsteemi eelistest „realseeritava alamsüsteemi kerge adapteeruvus muutuvale olukorrale“ (Õunapuu, 1988).

Selle töö pakutav üldlahendus soodustab samuti tarkvara evolutsiooni. Üldlahendus võimaldab hallata andmebaasi funktsioonis sisalduvate SQL lausete koodi, mille tulemusena muutub funktsiooni pakutav funktsionaalsus.

1.4.1. Refaktoreerimine

Kuna selles töös realiseeritav SQL lausete infosüsteem soodustab tarkvara evolutsiooni, siis on olemas oht, et süsteem keerukus hakkab suurenema. Sellisel juhul on süsteemile vaja rakendada refaktoreerimise protsessi.

Martin Fowleri raamatu „Refaktoreerimine: olemasoleva koodi disaini parandamine“ (1999) järgi on olemas kaks refaktoreerimise definitsiooni.

- Nimisõna „refaktoreerimine“ (ingl k *refactoring*) tähendab muudatust, mis on tehtud tarkvara sisemises struktuuris, selleks, et teha tarkvara arusaadavamaks ja odavamalt muudetavaks ilma selle nähtava käitumise mõjutamiseta.
- Tegusõna „refaktoreerima“ (ingl k *refactor*) tähendab tarkvara restruktureerimist, rakendades refaktoreerimiste seeriat ilma tarkvara nähtava käitumise mõjutamiseta.

Martin Fowler rõhutab kahte aspekti.

1. Refaktoreerimise eesmärk on muuta programmi nii, et seda oleks lihtsam lugeda ning hallata. Võib teha erinevaid koodi muudatusi (näiteks optimeerimiseks), kuid refaktoreerimised peavad justnimelt muutma koodi teksti arusaadavamaks.
2. Refaktoreerimine ei muuda tarkvara nähtavat käitumist. Programm täidab samu funktsioone nagu enne. Ükski kasutaja ei näe erinevust.

Fowler (1999) märgib, et mida rohkem ta tegeleb refaktoreerimisega, seda puhtamat/paremat koodi ta oskab kirjutada, kuna tal on üha rohkem kogemust. Ta teab, missugust refaktoreerimist tuleb millistel juhtudel kasutada.

On vaja meeles pidada, et refaktoreerimise ajal ei tehta funktsionaalseid muudatusi. Samuti vastupidi – funktsionaalsuse muutmise eesmärk ei ole refaktoreerimine, kuid arendajad võivad minna üle ühelt tegevuselt teise. Näiteks arendaja võib funktsionaalsuse muutmise ajal märgata, et eelnevalt loodud koodi on vaja refaktoreerida ning alles pärast seda on mõttekas funktsionaalsust muuta. Samuti võib juhtuda, et pärast uue funktsiooni lisamist näeb arendaja, et tulemuseks saadud kood ei ole piisavalt arusaadav (kuigi töötab õieti) ning hakkab seda refaktoreerima.

Töös väljapakutava süsteemi abil ärireegleid kirjeldades ei refaktoreeri lõppkasutaja süsteemi, vaid muudab selle funktsionaalsust. Refaktoreerimist saavad ja peavad rakendama ärireeglite süsteemi arendajad, kes vajadusel teevad koodis arusaadavust ning arendatavust silmas pidavaid muudatusi. SQL keeles on keeleline liiasus, mis võimaldab enamasti ühte ja sama ülesannet mitmel erineval viisil (erineva SQL lausega) lahendada. Erinevate lausete täitmine võib võtta erineva hulga aega. Seega veel üheks refaktoreerimise viisiks oleks antud kontekstis ärireegleid realiseerivate SQL lausete muutmine üldlahenduse infosüsteemi kaudu ilma lause tulemust muutmata. Ka see jääks eeskätt arendajate ülesandeks kuna analüütikutel ei pruugi olla piisavalt tehnilisi teadmisi konkreetse SQL-andmebaasisüsteemi sisemisest toimimisest.

1.5. Lõppkasutaja arendus

Lõppkasutaja arendus (ingl k *End-User Development* või EUD) on meetodid, tehnikad ning tööriistad, mis annavad tarkvarasüsteemi lõppkasutajatele võimalust luua, muuta või täiendada tarkvarasüsteemi (Paternò, 2013). Lõppkasutajad arendavad EUD tööriistu kasutades tarkvarasüsteemi. Lõppkasutaja arenduse puhul on kasutajatel põhilised programmeerimise teadmised, kuid nad ei ole professionaalsed arendajad.

Selles töös pakutav üldlahendus soodustab lõppkasutaja arenduse kasutuselevõttu. Üldlahenduse kasutajatel on võimalus oma tegevuse kaudu hallata andmebaasi protseduuride SQL koodi, mis tähendab protseduuri arendust.

Lõppkasutaja arenduse tööriistade kasutamine eeldab üldiseid programmeerimise teadmisi (Fischer, Giaccardi, Ye, Sutcliffe, Mehandjiev, 2004). Selle töö üldlahendus tegeleb SQL

koodiga, seega kasutajatel peavad olema teadmised SQL keele kohta. Kasutajad peavad olema piisavalt motiveeritud programmeerimiskeele õppimiseks.

Kasutajate motiveerijatena võib nimetada (Fischer, Giaccardi, Ye, Sutcliffe, Mehandjiev, 2004) järgmise aspekte.

- Võimalus teha tarkvarasüsteemi muudatusi efektiivsemalt, sest lõppkasutaja saab kohe tarkvara kasutamise ajal teha ise vajalikke muudatusi.
- IT-spetsialistidele nõuete selgeks tegemise vajaduse kadumine, sest lõppkasutaja ei pea seletama oma soovi arendajatele, vaid saab võimaluse need soovid ise ellu viia.

Metadisain (ingl k „meta-design“) on lõppkasutaja arenduse lähenemisviis (Fischer, Giaccardi, Ye, Sutcliffe, Mehandjiev, 2004). Metadisaini põhieesmärk on luua keskkond, mis aitaks lõppkasutajatel tarkvarasüsteemi arenduses aktiivselt osaleda.

Metadisaini puhul ei loo arendajad lõplikku tarkvarasüsteemi, vaid nad ehitavad sellise süsteemi, mida lõppkasutaja saab kasutamise ajal muuta. Sellisel juhul on lõppkasutajatel võimalus vastavalt oma vajadustele tarkvara funktsionaalsust ise muuta. Käesolevas töös väljapakutav üldlahendus kirjeldab sarnast süsteemi. Töös väljapakutud süsteemi kasutajaks on esmajärjekorras analüütikud (IT-poole töötajad), mitte lõppkasutajad kuid sarnaselt metadisaini põhimõttega väheneb arendajate koormus süsteemi jooksvate muudatuste tegemisel.

1988. aastal esitatud infosüsteemide genereerimise süsteem GENSI võimaldab samuti ettevõtte infosüsteemi kasutajatel osaleda infosüsteemi realiseerimises (Õunapuu, 1988).

1.6. Metaandmed

Metaandmed on andmed andmete kohta. Metaandmed annavad informatsiooni objekti sisu kohta (techterms.com). Näiteks metaandmed tabeli kohta sisaldavad tabeli nime, tabeli veergude kirjeldust, sh veergude tüüpe, jne.

1.6.1. Metaandmete funktsionaalsus

Metaandmetel on kolm põhifunktsiooni (Miksa, 2000).

- Tagada informatsioon süsteemi olemite kohta, mida saavad kasutada nii tarkvarasüsteemid kui ka arendustööga tegelevad inimkasutajad. Näiteks konkreetses SQL-andmebaasis on olemiteks baastabelid, vaated, funktsioonid, indeksid jne.
- Aitab kaasa süsteemi struktureerimisele – metaandmed aitavad ehitada olemitele ligipääsu süsteemi.
- Annab aluse süsteemi kirjelduse esitamiseks.

1.6.2. Metaandmete kasutamine

Ärireeglite haldamise süsteemides kasutatakse ärireeglite kohta metaandmeid nagu reegli nimi, kirjeldus, muutujad, tulemus jne. Metaandmed aitavad kaasa süsteemis ärireeglite esitamisele ning kasutamisele.

Veel üks metaandmete kasutamise näide on päringute süsteem (Voronova, 2009), mille ma bakalaureusetöö raames ehitasin. Süsteem sisaldab päringuid, mis otsivad andmebaasi metaandmete (mis kirjeldavad näiteks andmebaasi tabelite struktuuri) alusel selle andmebaasi disaini vigu.

Oracle APEX veebirakenduste loomise keskkond on samuti suurepärane metaandemete kasutamise näide. See võimaldab ehitada veebirakendusi, kasutades metaandmeid andmebaasi ning rakenduste kohta ning Oracle SQL dialekti ja PL/SQL protseduurkeelt (oracle.com). Oracle APEX on seega metaandmetega juhitav, mis tähendab, et niipea kui tehakse muudatus metaandmetes, siis muutub veebirakenduse funktsionaalsus või rakenduse väljanägemine.

Metanadmeid kasutab ka infosüsteemide genereerimise süsteem GENSI (Õunapuu, 1988). Süsteemi kasutaja sisestab infosüsteemi genereerimise jaoks vajalikud kirjeldused (spetisifkatsioonid), mis kujutavad ennast infosüsteemi metaandmeid ning mille põhjal GENSI süsteem genereerib infosüsteemi aruanded, andmetöötlusfunktsioonid või menüüd.

1.6.3. Käesolevas töös kasutatavad metaandmed

Selles töös luuakse infosüsteem ärireegleid realiseerivate SQL lausete koodi haldamise jaoks. Selle jaoks projekteeritakse andmebaas, mis võimaldab hoida SQL lausete metaandmeid (näiteks lause alliktabelid, tingimused, valemid jne.).

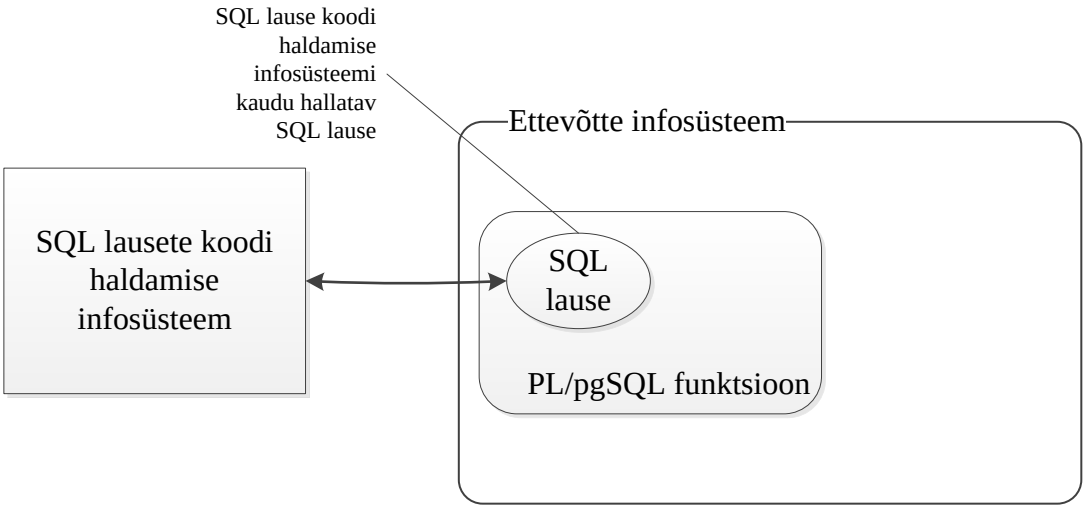
Samuti on süsteemi ehitamiseks vaja metaandmeid lause koodis kasutatavate tabelite ning veergude kohta. PostgreSQLi andmebaasi puhul on need võimalik võtta *information_schema* skeemist. *Information_schema* skeemi kirjeldab ka SQL standard ning selles sisalduvad andmebaasi süsteemikataloogi (automaatselt loodud metaandmete andmebaas) põhjal loodud standardiseeritud struktuuriga vaated. Idee on selles, et erinevate SQL-andmebaasisüsteemide korral võib metaandmete andmebaasi struktuur olla erinev, kuid kõik need süsteemid peaksid toetama ühesuguse struktuuriga vaateid selle andmebaasi põhjal.

Selle töö pakutav süsteem on sarnane Oracle APEX ja GENSI süsteemidega, sest see võimaldab samamoodi metaandmeid kasutades tarkvarasüsteemi funktsionaalsust hallata . Kuid erinevalt Oracle APEX ja GENSI süsteemidest ei võimalda antud töös pakutav süsteem muuta hallatava tarkvarasüsteemi välimust. Antud töös realiseeritav süsteem juhib ainult tarkvarasüsteemi sees kasutatavate andmebaasi funktsioonide SQL lausete loogikat, kasutades selleks SQL lausete koodi metaandmeid.

2. SQL lausete koodi haldamise infosüsteem

Antud peatükis kirjeldatakse infosüsteemi, mis võimaldab metaandmete abil hallata funktsioonides kasutatavat SQL lausete koodi.

2.1. Infosüsteemi ülevaade



Joonis 5. SQL lausete koodi haldamise infosüsteemi põhimõte.

Joonis 5 illustreerib SQL lausete koodi haldamise infosüsteemi põhimõtet. Antud töös realiseeritav infosüsteem võimaldab hallata DML tüüpi SQL lauseid, mis sisalduvad PL/pgSQL funktsioonides. Funktsioonid võivad olla kasutusel näiteks mõne ettevõte infosüsteemi osaks olevas tarkvarasüsteemis. SQL lausete koodi haldamise infosüsteem võimaldab hallata funktsiooni sees olevat SQL lauset. SQL lause muutmine tähendab ka funktsiooni sisu muutumist. Niipea kui SQL lausete koodi haldamise infosüsteemi kasutaja muudab SQL lause metaandmeid, hakkab funktsioonis kohe kehtima uuendatud SQL lause. See tähendab, et funktsiooni on muudetud.

Antud töö raames pakutava lahenduse korral peab hallatav SQL lause olema PL/pgSQL funktsiooni sees, sest funktsioon võimaldab kasutada dünaamilist SQLi, mille abil on võimalik süsteemis sisalduvate metaandemete põhjal dünaamiliselt koostada SQL lause koodi (sellest on täpsemalt kirjutatud jaotises 2.4.1). PL/pgSQL funktsiooni asemel võib olla ka mingi muu vahend, mis võimaldab kasutada dünaamilist SQLi (ehk koostada SQL

lauseid dünaamiliselt). Kuid kuna antud töös käsitletakse PostgreSQL andmebaasi, siis on dünaamilise SQL kasutamiseks on valitud PL/pgSQL funktsioonid.

Antud töö raames käsitletakse staatilisi DML laused, kuid töö üldlahendust võib laiendada, et see võimaldaks ka hallata DDL lauseid või dünaamilisi lauseid.

Töö pakutav üldlahendus saab kasutuselevõtu järel osaks ettevõtte infosüsteemist ja sellest võib mõelda kui *arendamise allsüsteemist*.

2.1.1. Infosüsteemi eesmärgid

- Funktsioonides sisalduvate SQL lausete koodi metaandemete hoidmine ja haldamise võimaldamine ning selle kaudu süsteemide muutmise paindlikumaks tegemine ja süsteemi muutmisel arendajate koormuse vähendamine

2.1.2. Põhiobjektid

- Funktsioon
- Lause (ingl k „statement“)
- Kasutaja (kasutajate haldus jääb käesolevas töös vaatluse alt välja kuid reaalses süsteemis on kahtlemata ülioluline, sest süsteemi kasutajal avaneb võimalus mõjutada teise tarkvarasüsteemi tööd ja see on suure vastutusega tegevus. Prototüübis on kasutajate tuvastamine realiseeritud nii, et süsteemi kasutamiseks tuleb logida sisse kui andmebaasiserveris CREATE USER lausega loodud kasutaja (kasutades sellele määratud kasutajanime ja parooli))
- Klassifikaator (klassifikaatorite haldus jääb käesolevas töös vaatluse alt välja)

2.1.3. Põhiprotsessid

- Funktsiooni lause sisestamine
- Funktsiooni lause muutmine
- Uue kasutaja loomine
- Uue klassifikaatori väärtuse lisamine

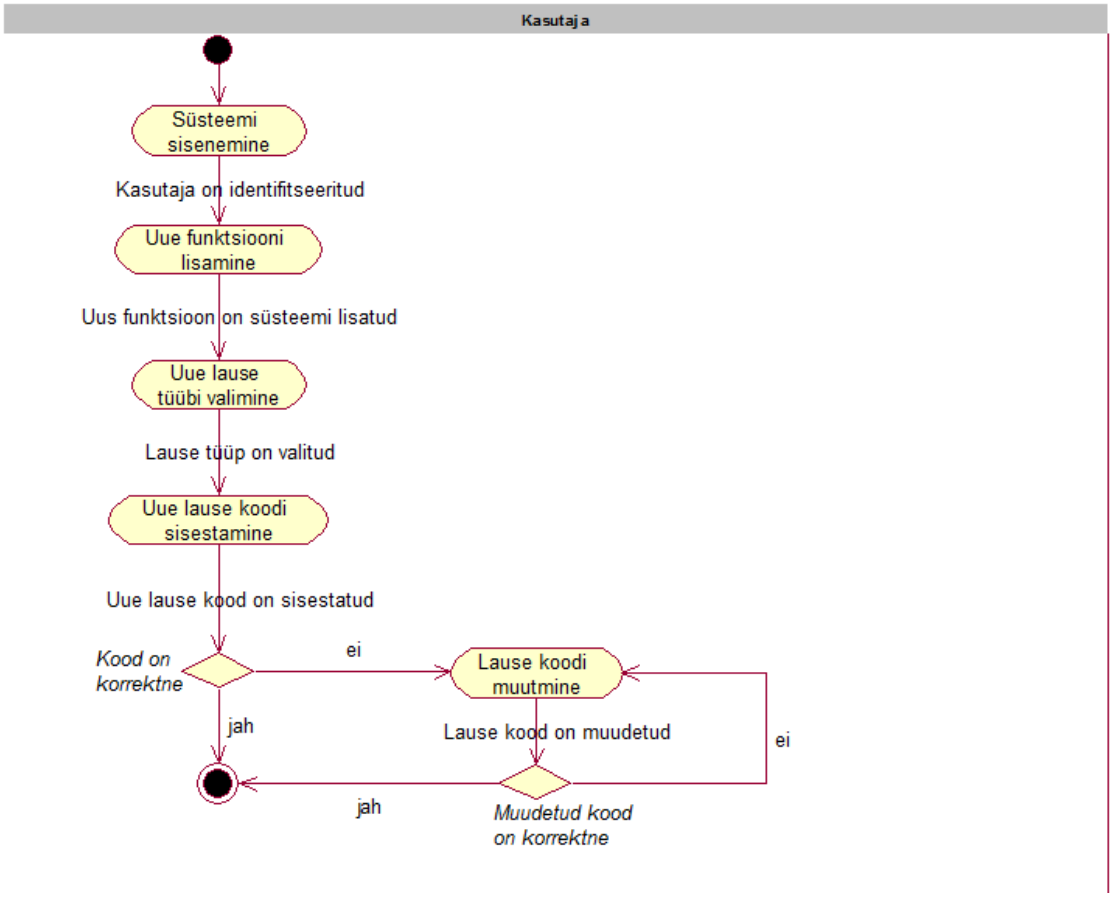
2.1.4. Põhilised sündmused

- On leitud uus funktsioon, milles sisalduva SQL lause koodi soovitakse infosüsteemi kaudu hallata.
- Muutub reegel, mis on realiseeritud infosüsteemis registreeritud funktsiooni SQL lause koodis.
- Uus analüütik tuleb tööle
- Analüütik lahkub töölt
- On vaja lisada uus klassifikaatori väärtus (näiteks hakkas andmebaasisüsteem toetama uut ühendamise tüüpi, mis tuleb klassifikaatorina registreerida või laiendati süsteemi nii, et see hakkab toetama uut tüüpi lauseid, mille eeldusena tuleb registreerida uus lause tüüp)

2.1.5. Tegutsejad

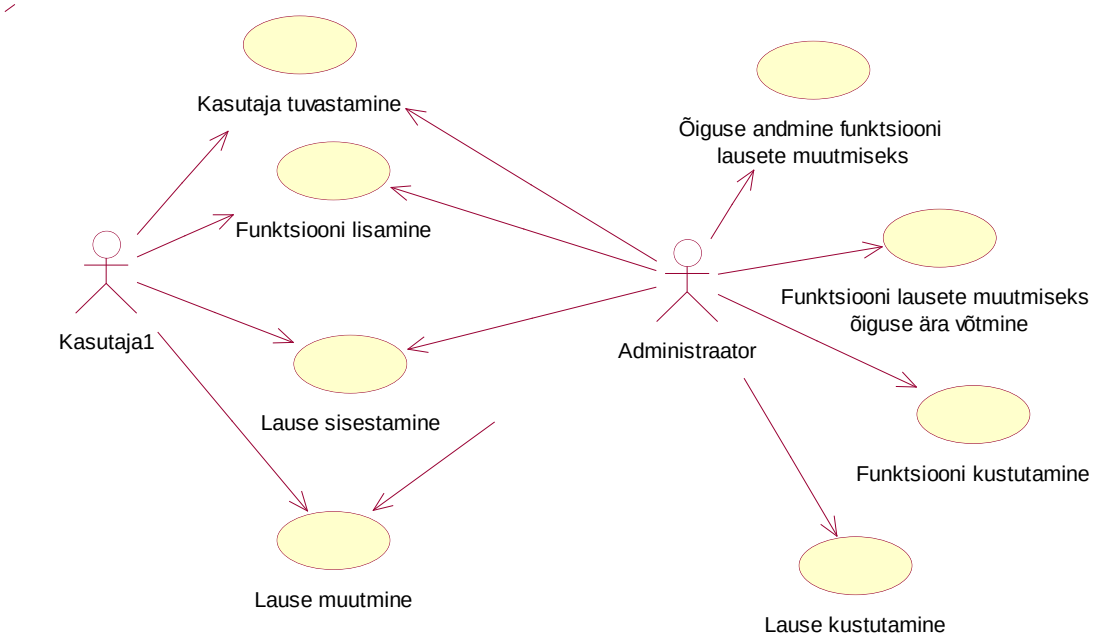
- Analüütik
- Administraator

2.1.6. Uue SQL lause koodi sisestamise protsessi tegevusdiagramm



Joonis 6. Uue SQL lause koodi sisestamise protsessi tegevusdiagramm

2.1.7. Kasutusjuhtude mudel



Joonis 7. Kasutusjuhtude diagramm

Kasutusjuht: Kasutaja tuvastamine

Tegutsejad: Analüütik, Administraator (edasi Kasutaja)

Kirjeldus: Süsteem küsib Kasutaja kasutajanime ja parooli. Kasutaja sisestab nende väärtused. Süsteem kontrollib sisestatud kasutajanime ja parooli. Kui Kasutaja on süsteemis tuvastatud, siis ta saab jätkata süsteemi kasutamist.

Kasutusjuht: Funktsiooni lisamine

Tegutsejad: Analüütik, Administraator (edasi Kasutaja)

Kirjeldus: Kasutaja sisestab uue funktsiooni nimetuse ja kirjelduse ning annab süsteemile käsu salvestada uue funktsiooni andmed.

Kasutusjuht: Lause sisestamine

Tegutsejad: Analüütik, Administraator (edasi Kasutaja)

Kirjeldus: Kasutaja valib funktsiooni, milles olevat lauset soovitakse hakata haldama. Kasutaja valib, millist tüüpi lauset ta soovib süsteemi sisestada. Lõpuks kasutaja sisestab valitud funktsiooni kohta valitud lause tüübi jaoks vajalike osalausete metaandmed (näiteks SELECT osalause või FROM osalause metaandmed) ning annab süsteemile käsu salvestada sisestatud andmed.

Kasutusjuht: Lause muutmine

Tegutsejad: Analüütik, Administraator (edasi Kasutaja)

Kirjeldus: Kasutaja valib süsteemis olemasolevate funktsioonide nimekirjast sobiva funktsiooni. Edasi muudab kasutaja funktsiooni SQL lauses sisalduvate osalausete kohta käivaid metaandmed ning annab süsteemile käsu salvestada uuendatud väärtused.

Kasutusjuht: Õiguse andmine funktsiooni lausete muutmiseks

Tegutsejad: Administraator

Kirjeldus: Administraator valib süsteemi kasutajate nimekirjast kasutaja ning annab süsteemile käsu salvestada valitud kasutaja õigus muuta valitud funktsiooni.

Kasutusjuht: Funktsiooni lausete muutmiseks õiguse äravõtmine

Tegutsejad: Administraator

Kirjeldus: Administraator valib süsteemi kasutajate nimekirjast kasutaja ning annab süsteemile käsu kustutada valitud kasutaja õigus valitud funktsiooni muuta.

Kasutusjuht: Funktsiooni kustutamine

Tegutsejad: Administraator

Kirjeldus: Administraator annab süsteemile käsu valitud funktsioon kustutada.

Kasutusjuht: Lause kustutamine

Tegutsejad: Administraator

Kirjeldus: Administraator annab süsteemile käsu kustutada valitud lause, mis sisaldub mingis funktsioonis.

2.2. Detailanalüüs

Antud jaotises on detailsemalt kirjeldatud autori hinnangul infosüsteemi kõige tähtsamad kasutusjuhud.

2.2.1. Kasutusjuhtude mudel

Kasutusjuht: Kasutaja tuvastamine

Primaarne kasutaja: Analüütik, Administraator (edasi Kasutaja).

Osapooled ja nende huvid:

- Kasutaja, Administraator: soovivad siseneda süsteemi ja teha tegevusi neile antud volituste piires.

Käivitav sündmus: Kasutaja soovib süsteemi siseneda.

Eeltingimused: Kasutaja peab olema registreeritud, et süsteemi siseneda.

Järeltingimused: On tehtud kindlaks, kas kasutajal on õigus süsteemi siseneda

või mitte. Kasutaja on autenditud ja talle on antud võimalus kasutada süsteemi talle

antud volituste piires.

Stsenaarium (tüüpiline sündmuste järjestus):

1. Kasutaja soovib siseneda süsteemi
2. **Süsteem** küsib kasutajalt tema kasutajanime ja parooli.
3. Kasutaja sisestab enda kasutajanimi ja parool.
4. **Süsteem** kontrollib, kas andmebaasis on sisestatud kasutajanime ja parooliga kasutaja.
5. **Süsteem** annab kasutajale volituse süsteemi kasutada.

Laiendused (alternatiivne sündmuste käik):

4a. Kui süsteem ei leia sisestatud kasutajanime ja parooliga kasutajat, siis kasutaja ei saa õigust süsteemi kasutada.

4a.1. **Süsteem** kuvab subjektile teate, et taolist kasutajat ei leidu ja ei anna

talle õigust süsteemi kasutada.

Kasutusjuht: Funktsiooni lisamine

Primaarne kasutaja: Analüütik, Administraator (edasi Kasutaja).

Osapooled ja nende huvid:

- Kasutaja, Administraator: soovivad lisada süsteemi andmed uue funktsiooni kohta selleks, et pärast lisada andmed funktsioonis sisalduva SQL lause kohta.

Käivitav sündmus: Kasutaja soovib lisada uue funktsiooni andmed.

Eeltingimused: Kasutaja on süsteemi poolt autenditud.

Järeltingimused: Uue funktsiooni andmed on süsteemi lisatud.

Stsenaarium (tüüpiline sündmuste järjestus):

1. Kasutaja sisestab uue funktsiooni nimetuse (koos andmebaasi skeemi nimetusega ning parameetritega) ning kirjelduse. Skeemi nimetus ja parameetrite kirjeldus on vajalik kuna erinevates skeemides saab olla sama nimega funktsioone ja ka samas skeemis saab olla sama nime kuid erinevate parameetritega funktsioone.
2. **Süsteem** lisab andmebaasi uue funktsiooni andmed.

Laiendused (alternatiivne sündmuste käik):

2a. Kui sisestatud nimetusega funktsioon juba eksisteerib andmebaasis, siis funktsiooni ei saa lisada.

2a.1. **Süsteem** kuvab teate, et sisestatud nimetusega funktsioon juba eksisteerib.

Kasutusjuht: Lause sisestamine

Primaarne kasutaja: Analüütik, Administraator (edasi Kasutaja).

Osapooled ja nende huvid:

- Kasutaja, Administraator: soovivad sisestatud või valitud funktsiooni jaoks lisada süsteemi uue SQL lause.

Käivitav sündmus: Kasutaja soovib lisada uue SQL lause.

Eeltingimused: Kasutaja on süsteemi poolt autenditud. On valitud funktsioon, mille jaoks soovitakse lisada SQL lause. Lause tüüp (näiteks SELECT või UPDATE) on valitud.

Järeltingimused: Uus SQL lause on süsteemi lisatud koos selle jaoks vajalike osalausetega.

Stenaarium (tüüpiline sündmuste järjestus):

1. Kasutaja sisestab valitud lause tüübi jaoks vajalikud osalaused.
2. **Süsteem** kontrollib, kas sisestatud SQL koodis ei ole süntaksi vigu.
3. **Süsteem** salvestab sisestatud andmed.

Laiendused (alternatiivne sündmuste käik):

2a. Kui sisestatud SQL kood sisaldab vigu, siis lauset ei saa lisada.

2a.1. **Süsteem** kuvab teate, et sisestatud SQL lause kood sisaldab vigu.

Kasutusjuht: Lause muutmine

Primaarne kasutaja: Analüütik, Administraator (edasi Kasutaja).

Osapooled ja nende huvid:

- Kasutaja, Administraator: soovivad muuta süsteemis oleva SQL lause koodi.

Käivitav sündmus: Kasutaja soovib muuta valitud funktsiooni SQL lause koodi.

Eeltingimused: Kasutaja on süsteemi poolt autenditud. On valitud funktsioon, mille SQL lauset soovitakse muuta.

Järeltingimused: SQL lause on süsteemis uuendatud.

Stsenaarium (tüüpiline sündmuste järjestus):

1. Kasutaja muudab valitud SQL lause vajalikud osalaused.

2. **Süsteem** kontrollib, kas muudetud SQL kood on nüüd korrektne.
3. **Süsteem** salvestab sisestatud andmed.

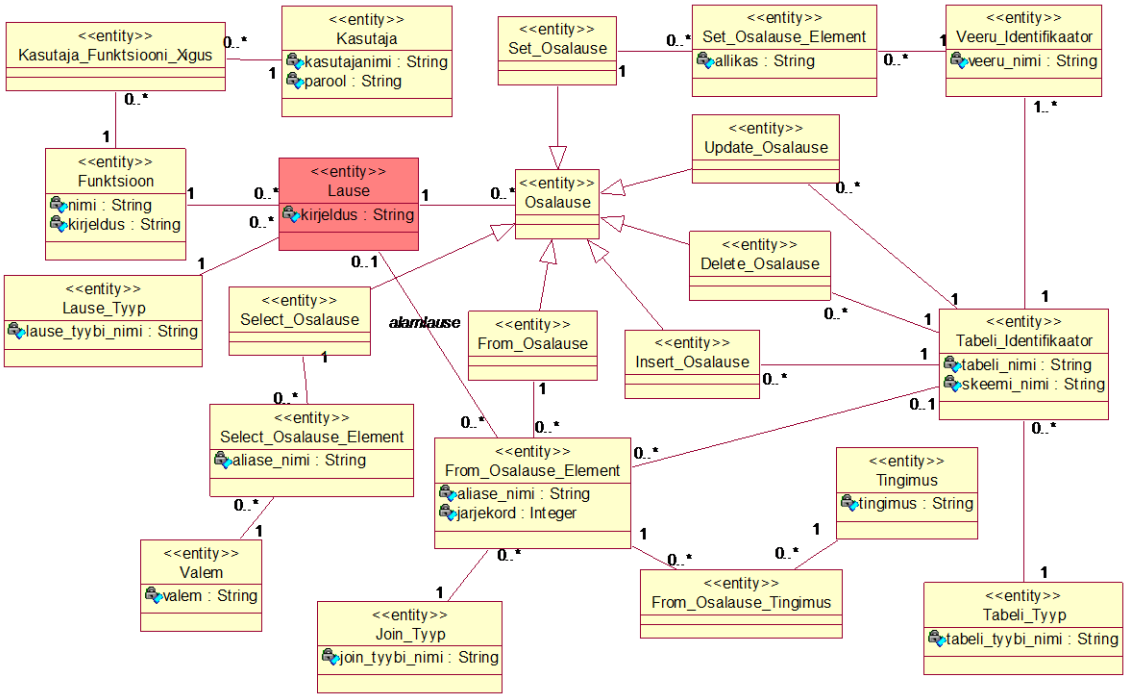
Laiendused (alternatiivne sündmuste käik):

2a. Kui pärast muudatust SQL kood sisaldab vigu, siis muudatust ei saa salvestada.

2a.1. **Süsteem** kuvab teate, et SQL lause kood sisaldab vigu.

2.2.2. Kontseptuaalne andmemudel

2.2.2.1. Olemi-suhte diagramm



Joonis 8. Olemi-suhte diagramm

Joonisel 8 kirjeldatud olemi-suhte diagrammil on esitatud ainult mõned võimalikest osalausete tüüpidest. Diagrammil on näiteks puudu INSERT lause jaoks vajalikud osalausete tüübid. Puuduvad osalausete tüübid võib lisada infosüsteemi järgmistes arenduse iteratsioonides.

Esmase prototüübi andmebaasis on realiseeritud mitte kõik antud diagrammil esitatud olemitüübid, vaid ainult need, mis on vajalikud SELECT tüüpi lause haldamiseks.

2.2.2.2. Olemitüüpide definitsioonid

Olemitüübi nimi	Definitsioon
Kasutaja	Isik, kellel on õigused süsteemi kasutada.
Funktsioon	Andmebaasi funktsioon, mida mingi rakendus saab käivitada. Funktsiooni käivitamine tingib omakorda ühe või rohkema Lause käivitamise.

Olemitüübi nimi	Definitsioon
Kasutaja_Funktsiooni_Xigus	Kasutaja õiguse olemasolu konkreetse funktsiooni muutmiseks.
Lause	SQL lause, mis sõltuvalt oma tüübist otsib, lisab, uuendab või kustutab andmeid.
Osalause	Lause osa, mis on SQL keele reeglite järgi vajalik.
From_Osalause	Osalause, mis on vajalik SELECT tüüpi Lause jaoks. Määrab lause alliktabelid ning seosed nende vahel.
From_Osalause_Element	FROM osalause allikas: tabel või alamlause.
Tingimus	Tingimus, mille järgi FROM osalauses seotakse omavahel alliktabelid.
From_Osalause_Tingimus	Seos FROM osalause alliktabeli ning sellega seotud tingimustega (JOIN tingimused).
Select_Osalause	Osalause tüüp, mis on vajalik SELECT tüüpi Lause jaoks. Määrab alliktabelite veerud, milles olevaid andmeid on vaja SQL lauses kasutada.
Select_Osalause_Element	SELECT osalause allikveerg, mida peab tagastama SELECT lause.
Valem	Arvutusvalem, mille järgi saadakse SELECT lause tulemus.
Update_Osalause	UPDATE lause jaoks vajalik osalause, mis määrab uuendatava tabeli.
Set_Osalause	UPDATE lause jaoks vajalik osalause, mis määrab uuendatavaid andmeid sisaldavad veerud.
Set_Osalause_Element	SET osalause element, mis määrab uuendatava veeru ning väärtuse, millega selle veeru uuendatakse.
Delete_Osalause	DELETE lause jaoks vajalik osalause, mis määrab tabeli, millest andmed kustutatakse.
Insert_Osalause	INSERT lause jaoks vajalik osalause, mis määrab andmete lisamise jaoks sihttabeli.
Tabeli_Identifikaator	Andmebaasis eksisteeriv baastabel või vaade.
Veeru_Identifikaator	Andmebaasis eksisteeriva baastabeli või vaade veerg.

Olemitüübi nimi	Definitsioon
Tabeli_Tyyp	Tabeli tüüp. Näiteks baastabel või vaade.
Lause_Tyyp	SQL lause tüüp. Näiteks SELECT, INSERT või UPDATE.
Join_Tyyp	Tabelite sidumise tüüp (FROM või JOIN). Näiteks FROM, INNER JOIN, LEFT JOIN või CROSS JOIN.

2.2.2.3. Atribuutide definitsioonid

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon	Näiteväärtus
Kasutaja	kasutajanimi	Kasutaja identifitseerimise jaoks vajalik nimi	t123
Kasutaja	parool	Kasutaja identifitseerimiseks vajalik salasõna	123456
Funktsioon	nimi	Andmebaasis loodud funktsiooni nimi (koos skeemi nimega ja parameetrite kirjeldusega).	kliendid.f_calc_sum (klient_id int)
Funktsioon	kirjeldus	Funktsiooni sõnaline, vabatekstiline kirjeldus	Kliendi tulusumma arvutus
Lause	kirjeldus	Lause sõnaline, vabatekstiline kirjeldus	Kliendi sissetulekute leidmine
From_Osalause_Element	aliase_nimi	Alliktabeli alias	A
Tingimus	tingimus	Alliktabelite sidumise tingimus	A.id = B.id

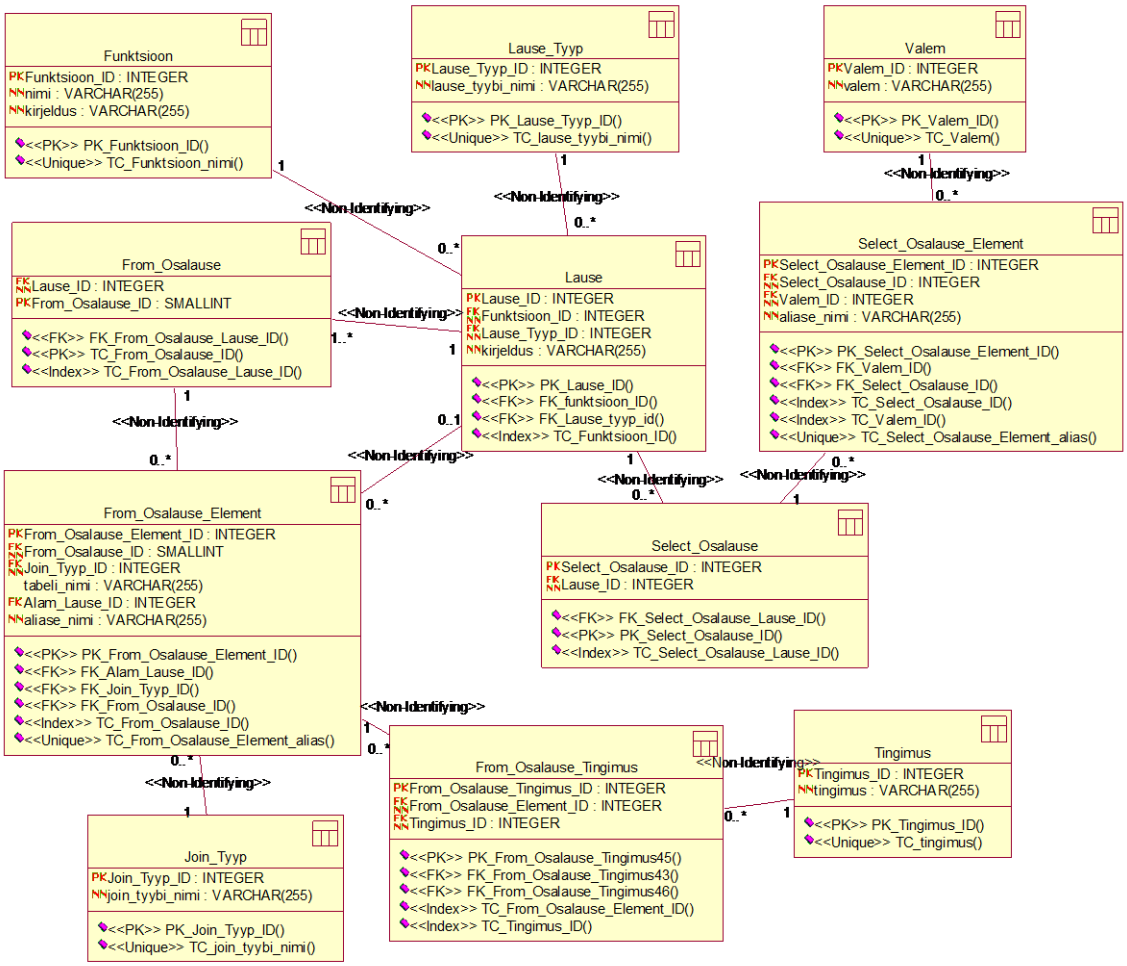
Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon	Näiteväärtus
Select_Osalause_Element	aliase_nimi	Valemi tulemuse alias	tulusumma
Valem	valem	SELECT osalauses sisalduva allikveeru (või allikveergude) kasutamist kirjeldav valem.	A.summa*B.hind
Tabeli_Identifikaator	tabeli_nimi	Andmebaasi tabeli nimi	Klient
Tabeli_Identifikaator	skeemi_nimi	Andmebaasi skeemi nimi	kliendid
Veeru_Identifikaator	veeru_nimi	Andmebaasi tabeli veeru nimi	id
Set_Osalause_Element	allikas	Väärtus, millega uuendatakse SET osalauses uuendatavat veergu.	6
Lause_Tyyp	lause_tyybi_nimi	SQL lause tüübi nimetus.	SELECT
Tabeli_Tyyp	tabeli_tyybi_nimi	Ütleb, kas tegemist on baastabeliga või vaadega	Baastabel
Join_Tyyp	join_tyybi_nimi	Alliktabelite omavahel sidumise tüübi nimetus (JOINi tüübi nimetus)	INNER JOIN

2.2.3. Infosüsteemi rollide kirjeldused

Rolli nimi	Kirjeldus
Analüütik	Analüütik on isik, kes töötab andmebaasiga (analüüsib andmebaasi disaini), omab baasteadmiseid SQL koodi kohta ning soovib muuta andmebaasi funktsiooni sees defineeritud SQL lauseid ning selle kaudu neid funktsioone kasutavate rakenduste käitumist.
Administraator	Administraator on isik, kellel on õigused lisada, muuta ning kustutada süsteemi sisestatud lauseid ning hallata funktsiooni muutmise õiguseid.

2.3. Loogiline andmebaasi disain

2.3.1. Andmebaasi diagramm



Joonis 9. Andmebaasi diagramm

Joonisel 9 kirjeldatud andmebaasi diagrammil esitatakse ainult esmase prototüübi realiseerimiseks vajalikud tabelid. Diagrammil on esitatud ainult need kontseptuaalses andmemudelis esitatud olemitüüpidele vastavad tabelid, mis on vajalikud SELECT lause koodi haldamiseks. Lisaks on prototüübi realiseerimise lihtsustamiseks andmebaasi diagrammil puudu *Tabeli_Identifikaator* ja *Kasutaja* olemitüüpidele vastavad tabelid.

2.3.2. Tabelite detailsed kirjeldused

Funktsioon

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
funktsioon_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
nimi	T	varchar	100			Jah
kirjeldus	T	varchar	500			Ei

Primary Key (funktsioon_id)
Alternate Key (nimi)

Lause_Tyyp

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
lause_tyyp_id	T	smallint				Jah
lause_tyybi_nimi	T	varchar	30			Jah

Primary Key (lause_tyyp_id)
Alternate Key (lause_tyybi_nimi)

Lause

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
lause_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
funktsioon_id	T	integer				Jah
lause_tyyp_id	T	smallint				Jah
kirjeldus	T	varchar	500			Ei

Primary Key (lause_id)
Foreign Key (funktsioon_id) REFERENCES Funktsioon(funktsioon_id)
Foreign Key (lause_tyyp_id) REFERENCES Lause_Tyyp(lause_tyyp_id)

Select_Osalause

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
select_osalause_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
lause_id	T	integer				Jah

Primary Key (select_osalause_id)
Foreign Key (lause_id) REFERENCES Lause(lause_id)

Valem

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
valem_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
valem	T	varchar	500			Jah

Primary Key (valem_id)
Alternate Key (valem)

Select_Osalause_Element

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
select_osalause_element_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
select_osalause_id	T	integer				Jah
valem_id	T	integer				Jah
aliase_nimi	T	varchar	255			Jah

Primary Key (select_osalause_element_id)
Foreign Key (select_osalause_id) REFERENCES Select_Osalause(select_osalause_id)
Foreign Key (valem_id) REFERENCES Valem(valem_id)
Alternate Key (aliase_nimi)

From_Osalause

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
from_osalause_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
lause_id	T	integer				Jah

Primary Key (from_osalause_id)
Foreign Key (lause_id) REFERENCES Lause(lause_id)

Join_Tyyp

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
join_tyyp_id	T	smallint				Jah
join_tyybi_nimi	T	varchar	40			Jah

Primary Key (join_tyyp_id)
Alternate Key (join_tyybi_nimi)

From_Osalause_Element

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
from_osalause_element_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
from_osalause_id	T	integer				Jah
join_tyyp_id	T	smallint				Jah
tabeli_nimi	T	varchar	255			Ei
alam_lause_id	T	integer				Ei
aliase_nimi	T	varchar	255			Jah

Primary Key (from_osalause_element_id)
Foreign Key (from_osalause_id) REFERENCES From_Osalause(from_osalause_id)
Foreign Key (join_tyyp_id) REFERENCES Join_Tyyp(join_tyyp_id)
Foreign Key (alam_lause_id) REFERENCES Lause(lause_id)
Alternate Key (aliase_nimi)

Tingimus

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
tingimus_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
tingimus	T	varchar	500			Jah

Primary Key (tingimus_id)
Alternate Key (tingimus)

From_Osalause_Tingimus

Veeru nimi	Tüüp/ domeen	Tüübi/ domeeni nimi	Pikkus	Võimalikud väärtused	Vaikeväärtus	Kohustuslik
from_osalause_tingimus_id	T	integer		Unikaalsed väärtused. Automaatselt genereeritud number, samm = 1		Jah
from_osalause_element_id	T	integer				Jah
tingimus_id	T	integer				Jah

Primary Key (from_osalause_tingimus_id)
Foreign Key (from_osalause_element_id)
REFERENCES From_Osalause_Element (from_osalause_element_id)
Foreign Key (tingimus_id) REFERENCES Tingimus(tingimus_id)

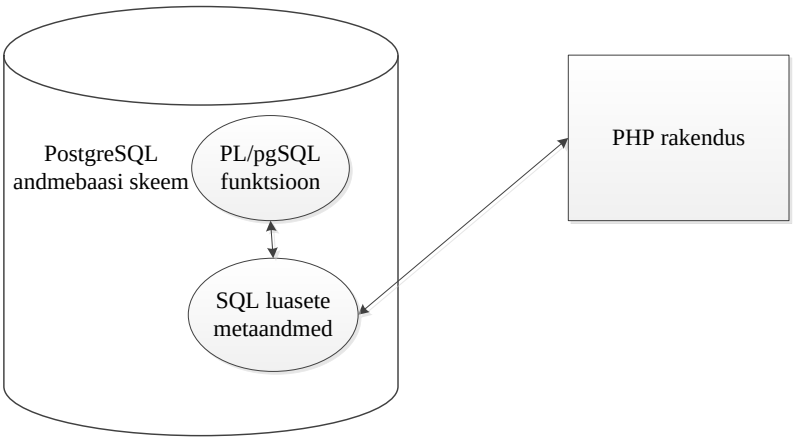
2.3.3. Klassifikaatorite väärtused

Klassifikaatori tabeli nimi	Koodi liik	Koodi väärtused	Nimed
Lause_Tyyp	Monotoonselt kasvavad täisarvud	1 2 3 4	SELECT INSERT UPDATE DELETE
Join_Tyyp	Monotoonselt kasvavad täisarvud	1 2 3 4 5 6	INNER JOIN LEFT JOIN RIGHT JOIN FULL JOIN CROSS JOIN FROM

2.4. Süsteemi tehniline arhitektuur

2.4.1. SQL lausete koodi haldamise süsteemi arhitektuur

SQL lausete koodi haldamise infosüsteemi arhitektuuri kirjeldab Joonis 10.

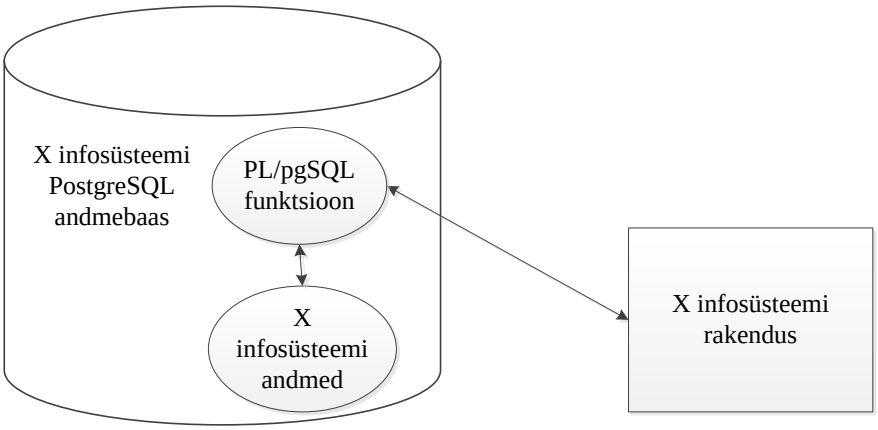


Joonis 10. SQL lausete koodi haldamise süsteemi arhitektuur

Jooniselt on näha, et süsteem koosneb andmebaasi skeemist, mis sisaldab SQL lausete metaandmeid sisalduvaid tabeleid ning PL/pgSQL funktsiooni. Funktsioon paneb kokku SQL lause koodi, kasutades SQL lausete metaandmed ning järjestades lauses kasutatud osalaused õiges järjekorras.

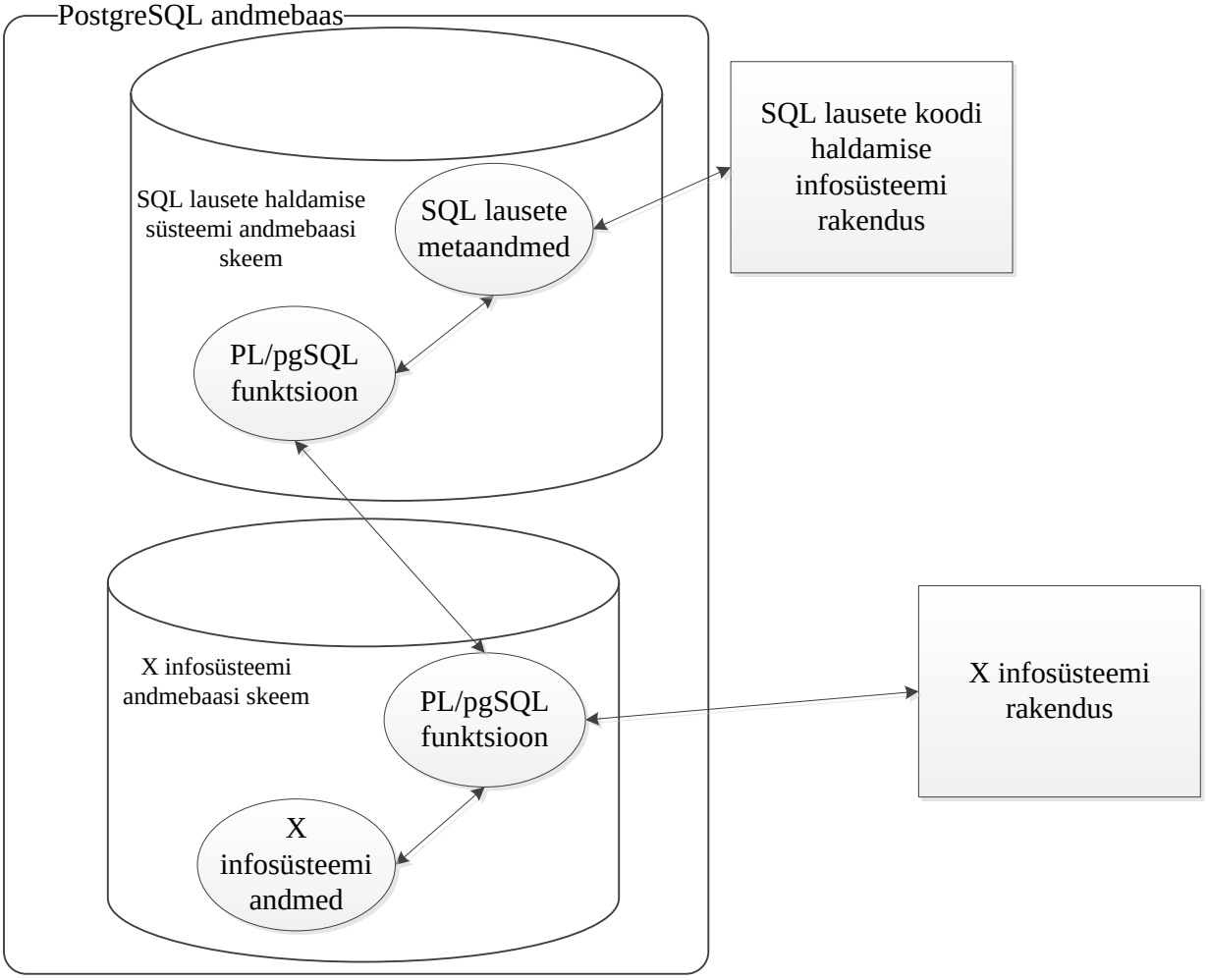
Lisaks sisaldab infosüsteem ka rakendust, mis võimaldab hallata SQL lausete metaandmed. Rakendus on projekteeritud nii, et juhul kui kasutaja ei sisestanud SQL lause jaoks vajalikku osalauset, siis rakendus annab vea ning ei salvesta SQL lause koodi.

Oletame, et on tegemist mingi ettevõtte X infosüsteemiga, mis koosneb PostgreSQL andmebaasist ning andmebaasi kasutatavast rakendusest (vt Joonis 11).



Joonis 11. Ettevõte X infosüsteemi arhitektuur enne SQL lausete koodi haldamise süsteemi integreerimist

X infosüsteemi andmebaas koosneb infosüsteemi andmetest, neid andmeid sisaldavatest andmestruktuuridest ning PL/pgSQL funktsioonidest. Infosüsteemi X rakendus kasutab neid PL/pgSQL funktsioone infosüsteemi andmete lugemiseks, sisestamiseks, muutmiseks või kustutamiseks. Funktsioonide sees on DML tüüpi SQL laused, mis otsivad, lisavad, muudavad või kustutavad infosüsteemi andmebaasist andmeid. Kui mingi SQL lause kood peaks tihti muutuma (näiteks kui igal nädalal on uued tingimused, mille järgi valitakse infosüsteemi rakenduse poolt väljastatud kliente), siis sellise koodi haldamiseks võib kasutada antud töös pakutava SQL lausete koodi haldamise süsteemi. Antud töös pakutava süsteemi kaudu saaks muuta valitud funktsiooni SQL lausetes kasutatavaid tingimusi ning need hakkavad kohe funktsioonis kehtima. Joonisel 12 on kirjeldatud ettevõtte X infosüsteemi arhitektuur pärast SQL lausete koodi haldamise süsteemi kasutuselevõtmist.



Joonis 12. Ettevõtte X infosüsteemi arhitektuur pärast SQL lausete koodi haldamise süsteemi integreerimist

SQL lausete koodi haldamise süsteemi kasutuselevõtuga seoses on infosüsteemi X andmebaasi funktsioone, mille puhul soovitakse rakendada paindlikku SQL koodi haldust, vaja muuta nii, et need kasutaksid SQL lausete haldamise süsteemi spetsiaalselt selleks otstarbeks loodud funktsiooni (kutsume seda *f*)(funktsioon võib omakorda kasutada abifunktsioone). Infosüsteemi X ning SQL lausete haldamise süsteemi andmebaasi skeemid peavad olema samas andmebaasis, selleks, et infosüsteemi X andmebaasi funktsioonid saaksid funktsiooni *f* kasutada (on olemas ka võimalus). PostgreSQLis on olemas ka võimalus kasutada *dblink* mooduli, mis annab võimalust võtta ühendust teise andmebaasiga (postgresql.com). Antud töö raames aga ei kasutata *dblink* mooduli, kuna mitte kõik andmebaasisüsteemid pakkuvad sellist funktsionaalsust.

Funktsioon *f* paneb kokku vajaliku SQL lause koodi, kasutades SQL lausete metaandmeid ning tagastab SQL lause koodi infosüsteemi X funktsioonile. Infosüsteemi X funktsioon käivitab saadud SQL lause ning tagastab tulemuse infosüsteemi X rakendusele. Sellepärast, niipea kui toimuvad mingid muudatused SQL lausete haldamise süsteemis talletatud SQL lausete metaandmetes, hakkavad need muudatused kohe kehtima ka ettevõte X andmebaasi funktsioonides.

2.4.2. Esmane prototüüp

Esimese iteratsiooni jooksul loodi PostgreSQL andmebaas funktsioonide SQL koodi hoidmiseks. Andmebaas loodi ülikooli pakutaval apex.ttu.ee serveril andmebaasis nimega *sql_code_ms* ning skeemis nimega *laused*.

Lisaks loodi skeemis *laused* PostgreSQL funktsioonid, mis kirjutati kasutades PL/pgSQL protseduurset keelt. Need funktsioonid panevad kokku SQL laused, kasutades SQL koodi haldamise süsteemi andmebaasis olevaid andmeid.

Esmases prototüübis realiseeriti ainult SELECT lausete haldamine ning SELECT tüüpi lause osalausetest realiseeriti ainult SELECT, FROM ning WHERE osalaused. Seega loodi järgmised funktsioonid:

Funktsiooni nimi: `laused.f_get_select (stmt_id int, OUT select_clause_text text)`

Kirjeldus: Funktsioon tagastab SELECT osalause koodi.

Funktsiooni nimi: `laused.f_get_join_conditions (from_element_id int, OUT join_cond_text text)`

Kirjeldus: Funktsioon tagastab JOIN või WHERE alamosa FROM osalause kohta.

Funktsiooni nimi: `laused.f_get_stmt_code(stmt_id int, OUT statement_code text)`

Kirjeldus: Funktsioon tagastab terve SQL lause koodi.

Funktsioonide koodid esitatakse töö lõpus (Lisad 1, 2 ja 3).

Nendest kolmest funktsioonist on kõige olulisem viimane funktsioon `laused.f_get_stmt_code(stmt_id int, OUT statement_code text)`, mis tagastab

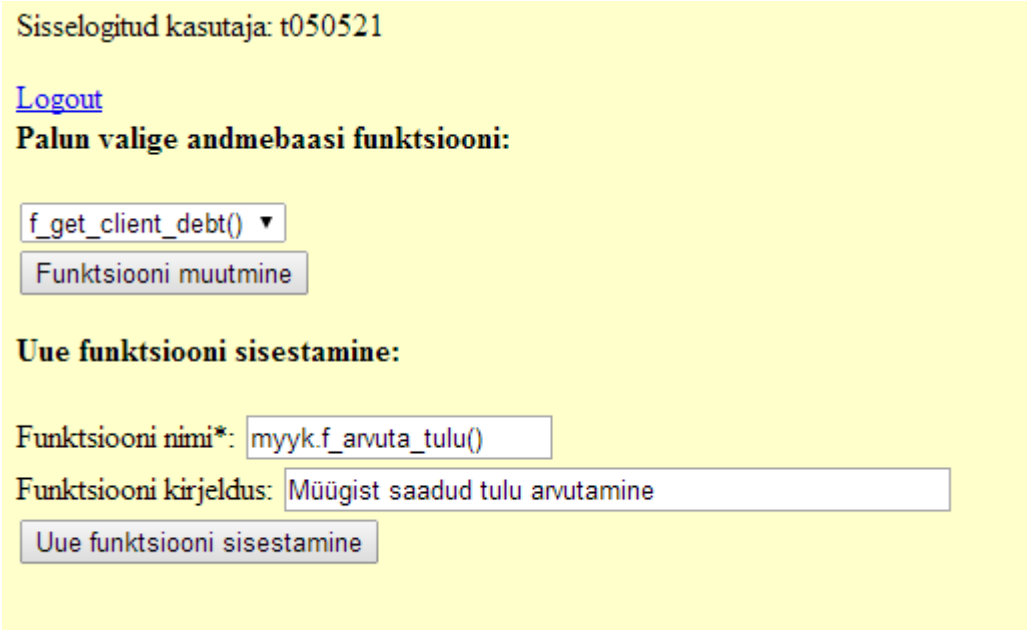
terve SQL lause koodi. Selle jaoks kasutab see kahte esimest funktsiooni, mis tagastavad SELECT osalause koodi ning JOIN või WHERE alamosa FROM osalause jaoks. Infosüsteemi kaudu hallatavat SQL lause koodi koostamiseks on vaja kasutada ainult kolmandat funktsiooni: `laused.f_get_stmt_code(stmt_id int, OUT statement_code text)`.

Viimasena loodi PHP skriptkeele abil rakenduse prototüüp, mis asub aadressil http://apex.ttu.ee/sql_code_ms.

Prototüübis realiseeriti järgmised põhilised kasutusjuhud:

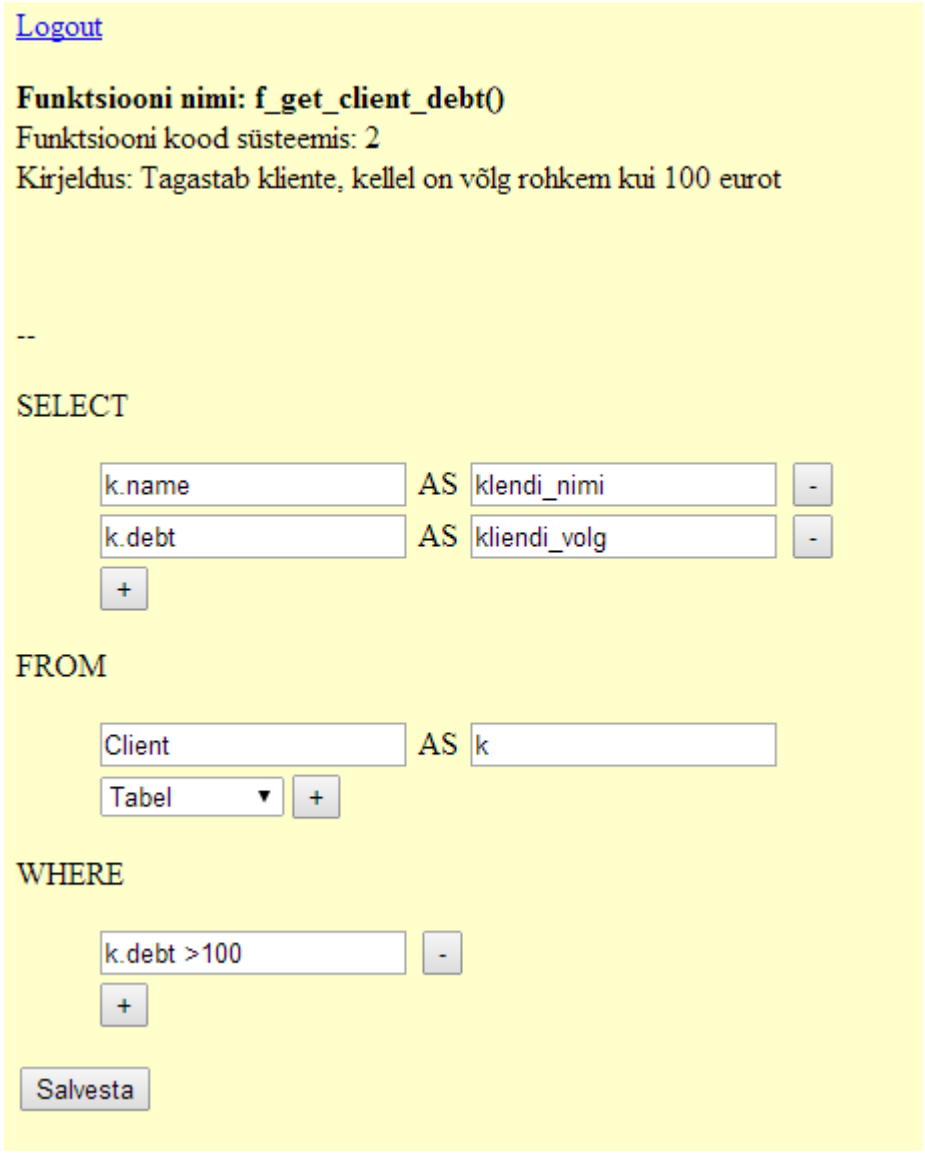
- Kasutaja identifitseerimine
- Funktsiooni lisamine
- Funktsiooni valimine
- SELECT tüüpi lause sisestamine
- SELECT tüüpi lause muutmine

Joonisel 13 on toodud, kuidas funktsiooni lisamine ja valimine on prototüübis realiseeritud.



Joonis 13. Funktsiooni lisamise ja valimise realiseerimine prototüübis

Joonisel 14 kirjeldatakse, kuidas on prototüübis realiseeritud SELECT tüüpi lause sisestamine ja muutmine.



Joonis 14. SELECT tüüpi lause sisestamise ja muutmise realiseerimine prototüübis

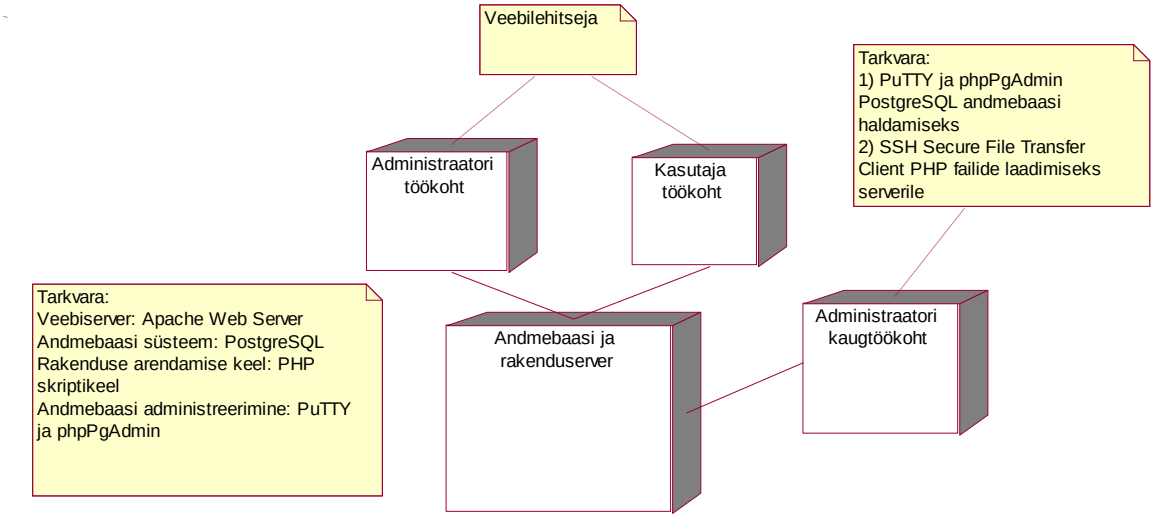
Esmase prototüübi kasutajal peavad olema vähemalt baasteadmised SQL keele kohta. Kasutaja peab teadma vähemalt muudetava lause tüübi SQL keele reegleid. Näiteks, kui kasutajal on õigus muuta ainult ühte lauset, mille tüüp on SELECT, siis kasutajal peavad olema teadmised SELECT lause ning selle osalausete süntaksi kohta, kuid ta ei pea teadma teiste lausete tüüpide (näiteks INSERT või UPDATE) süntaksi ja semantikat. Samas, kui kasutaja vajab rohkem teadmisi mingi keerulise lause ehitamiseks, siis ta saab konsulteerida andmebaasi arendajaga.

SQL lausete koodi haldamise süsteemi põhikasutaja on andmebaasi analüütik, kes igapäevaselt tegeleb andmebaasi tabelite disainiga ning tabelites sisalduvate andmete analüüsiga. Seega ta võib olla huvitatud SQL keele teadmiste saamisest, kuna see aitaks mitte ainult kasutada antud töös pakutava lahendust, vaid oleks kasulik ka teistes igapäevastes tegevustes.

Järgmistes infosüsteemi iteratsioonides võib parandada kasutajaliidest, et see oleks kergemini kasutav ning eeldaks väiksemaid SQL keele teadmisi.

Süsteemi SQL süstimise abil ründamise võimaluse vähendamiseks on kasutajaliidese prototüübi PHP koodis andmebaasiga suhtlemiseks kasutatud pg_query_params() funktsiooni, mis võimaldab korraga käivitada ainult ühe SQL lause.

2.4.3. Töötav süsteem



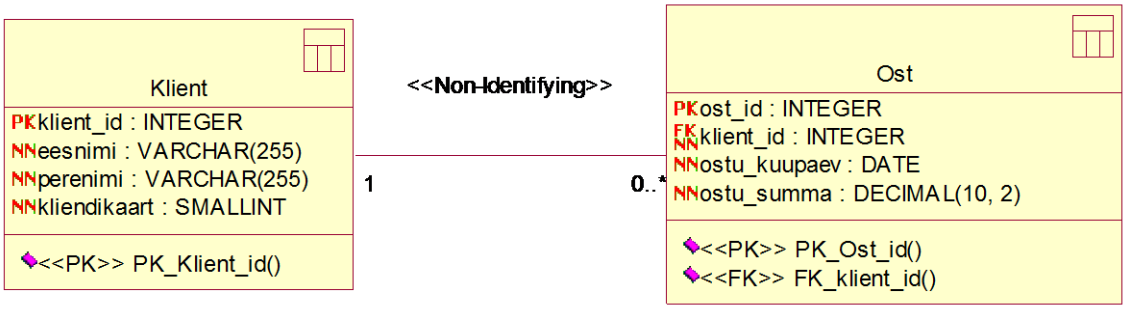
Joonis 15. Süsteemi tarkvara nõuded

2.5. Infosüsteemi kasutamise lihtne näide

Antud jaotises on esitatud lihtne SQL lausete koodi haldamise infosüsteemi kasutamise näide.

Oletame, et on tegemist toidupoe süsteemiga, kus on reegel, mille järgi süsteemi rakendus peab tagastama ilma kliendikaardita klientide ostud, kus ostu summa on suurem kui 40 eurot.

Näite illustreerimise jaoks loodi lihtsama andmebaasi ülikooli pakutaval apex.ttu.ee serveril andmebaasis nimega *sql_code_ms* ning skeemis nimega *ostud* (vt Joonis 16).



Joonis 16. *Ostud* skeemi andmebaasi diagramm.

Andmebaasi tabelitesse sisestati näidisandmed (vt Tabelid 7 ja 8).

Tabel 7. *Klient* tabeli näidisandmed

klient_id	eesnimi	perenimi	kliendikaart
1	John	Smith	TRUE
2	Kadri	Kell	FALSE
3	Tim	Rae	TRUE

Tabel 8. *Ost* tabeli näidisandmed

ost_id	klient_id	ostu_kuupaev	ostu_summa
1	1	11.03.2014	15.7
2	3	15.03.2014	12
3	2	15.03.2014	32.1
4	3	20.03.2014	97.43
5	1	15.04.2014	63.01
6	2	17.04.2014	41.86
7	2	23.04.2014	58.57

Lõpuks loodi PL/pgSQL funktsiooni `ostud.f_suured_ostud()`, mis tagastab kõik ilma kliendikaardita klientide ostud, kus ostu summa on suurem kui 40 eurot. Funktsiooni loomise jaoks on SQL lause järgmine:

```
CREATE OR REPLACE FUNCTION ostud.f_suured_ostud()
RETURNS TABLE (klient_nimi varchar, ostu_kp timestamp, ostu_summa double
precision) AS $$
BEGIN

      RETURN QUERY SELECT
                        K.eesnimi||' '||K.perenimi AS klient_nimi,
                        O.ostu_kuupaev AS ostu_kp,
                        O.ostu_summa AS ostu_summa
                  FROM ostud.Klient K
                  JOIN ostud.Ost O
                  ON O.klient_id = K.klient_id
                  WHERE K.kliendikaart = false
                  AND O.ostu_summa > 40;

END;
$$ LANGUAGE plpgsql;
```

Käesoleva näide andmebaasi skeem asub *sql_code_ms* andmebaasis, kus on juba olemas SQL lausete koodi haldamise infosüsteemi andmebaasi skeem *laused*, mis sisaldab tabeleid SQL lausete kodi metaandmete hoidmise jaoks ning SQL lausete koodi genereerimiseks mõeldud PL/pgSQL funktsiooni. Samuti on olemas infosüsteemi rakendus (aadressil http://apex.ttu.ee/sql_code_ms), mis kasutab *laused* andmebaasi skeemis olevaid andmeid.

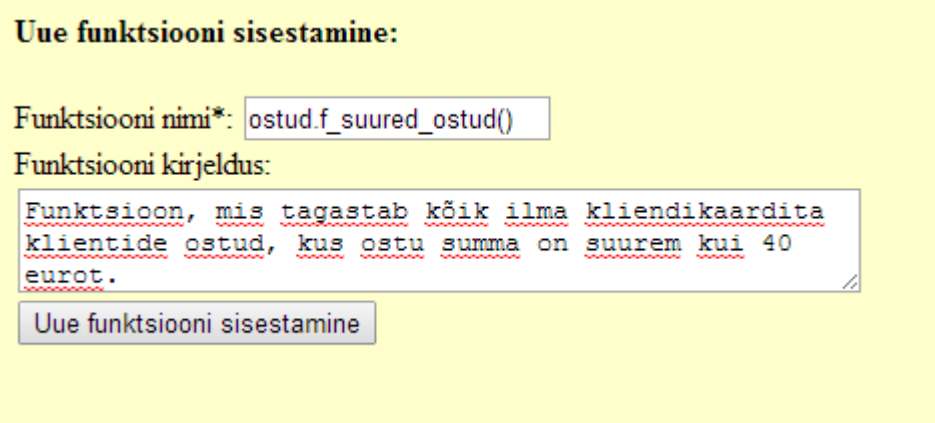
Edasi demonstreeritakse, kuidas toimub toidupoe andmebaasi funktsiooni `ostud.f_suured_ostud ()` muutmine, et selle sees olev SQL lause kood oleks hallatav SQL lausete koodi haldamise süsteemi kaudu. Selle jaoks viidi läbi järgmised sammud:

2.5.1. Funktsioonis sisalduva SQL lause koodi sisestamine SQL lausete koodi haldamise infosüsteemi.

Antud sammu juures registreeriti süsteemis `ostud.f_suured_ostud ()` funktsiooni sees olev SELECT lause:

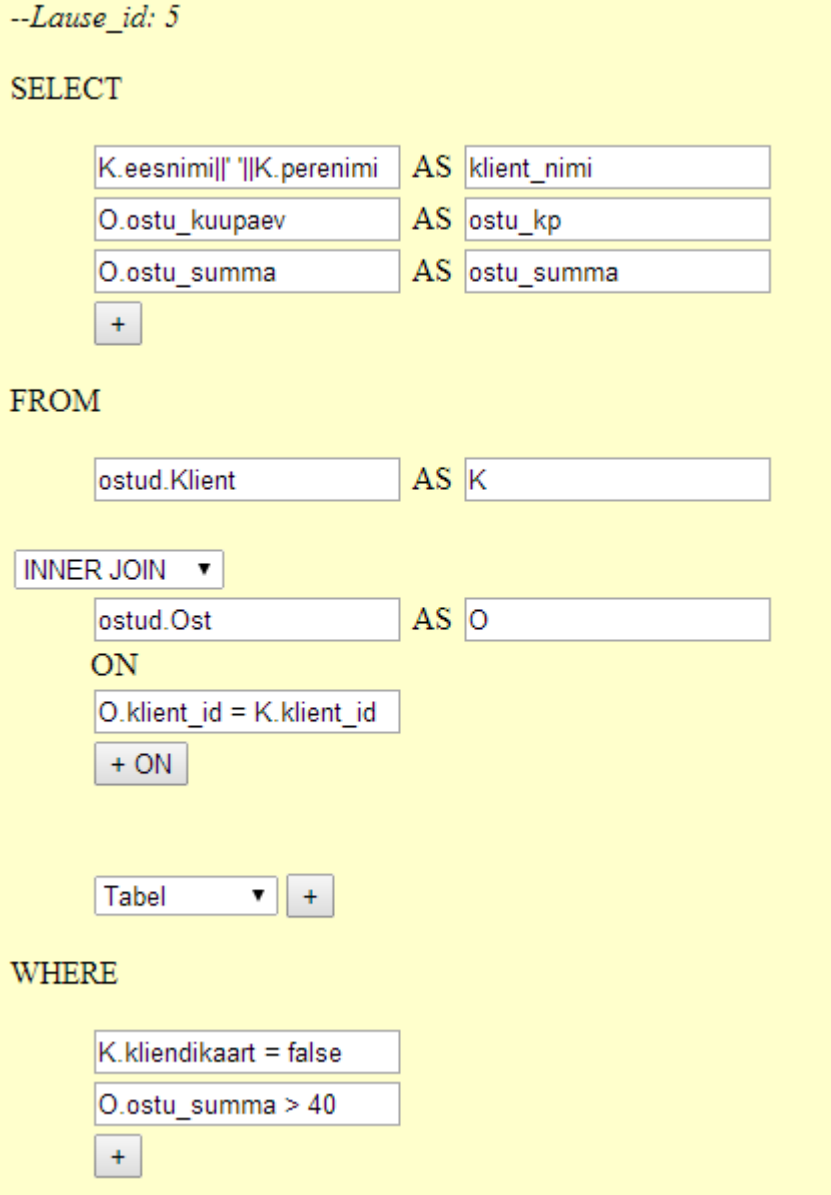
```
SELECT
      K.eesnimi||' '||K.perenimi AS klient_nimi,
      O.ostu_kuupaev AS ostu_kp,
      O.ostu_summa AS ostu_summa
FROM ostud.Klient K
JOIN ostud.Ost O
ON O.klient_id = K.klient_id
WHERE K.kliendikaart = false
AND O.ostu_summa > 40;
```

Selle jaoks lisati esimesena SQL lausete koodi haldamise süsteemi funktsiooni `ostud.f_suured_ostud()` nimi ja kirjeldus (vt Joonis 17).



Joonis 17. Funktsiooni lisamine süsteemi

Edasi registreeriti SQL lausete koodi haldamise süsteemi rakenduse kaudu SELECT lause koodi metaandmed (vt Joonis 18).



Joonis 18. Süsteemi sisestatud SELECT lause koodi metaandmed

Selle tulemusena on SQL lausete koodi haldamise süsteemi andmebaasi skeemis *laused* olemas funktsioonis `ostud.f_suured_ostud()` sisalduva SQL lause metaandmed.

2.5.2. Funktsiooni koodi muutmine SQL lausete koodi haldamise infosüsteemi abifunktsiooni kasutamiseks

Järgmise sammuna muudeti `ostud.f_suured_ostud()` funktsiooni nii, et see kasutaks SQL lausete koodi haldamise süsteemi abifunktsiooni, millega genereeritakse metaandmetest SQL kood. SQL lausete koodi haldamise infosüsteemis on olemas `laused.f_get_stmt_code (stmt_id int)` funktsioon, mis võtab sisendina lause numbri

ning väljastab lause SQL koodi. Lause numbrid on.registreeritud *Lause* tabelis ning neid näeb ka rakenduse vahendusel (vt Joonis 18, Lause_id) .

Järgmisena muudeti `ostud.f_suured_ostud()` funktsiooni koodi, et see hakkaks kasutama SQL lausete koodi haldamise süsteemi abifunktsiooni `laused.f_get_stmt_code (stmt_id int)`:

```
CREATE OR REPLACE FUNCTION ostud.f_suured_ostud()
RETURNS TABLE(klient_nimi varchar, ostu_kp timestamp, ostu_summa double
precision) AS $$
DECLARE
      stmt text;
BEGIN

      EXECUTE 'select statement_code from laused.f_get_stmt_code (5);'
      INTO stmt;

     RETURN QUERY EXECUTE stmt;

END;
$$ LANGUAGE plpgsql;
```

`laused.f_get_stmt_code (stmt_id int)` abifunktsioon tagastab sama SELECT lause, mis oli varem kirjutatud staatilisena `ostud.f_suured_ostud ()` funktsiooni sisses. Kuna abifunktsioon kasutab lause genereerimiseks SQL lause koodi metaandmeid, siis iga metaandmete muudatus mõjutab kohe `laused.f_get_stmt_code (stmt_id int)` funktsiooni poolt genereeritava SQL lause koodi.

2.5.3. Muudetud funktsiooni testimine

Viimase sammuna testitit muudetud funktsioon `ostud.f_suured_ostud ()`.

Funktsiooni nimi: `ostud.f_suured_ostud ()`
Testjuhtum: Testitakse, kas funktsioon tagastab pärast muudatust samad andmed kui enne muudatust.
Testjuhtumi käik: `SELECT klient_nimi, ostu_kp, ostu_summa FROM ostud.f_suured_ostud();`

Oodatav tulemus:

```
 klient_nimi |       ostu_kp       | ostu_summa
-------------+----------------------+------------
 Kadri Kell  | 2014-04-17 00:00:00 |      41.86
 Kadri Kell  | 2014-04-23 00:00:00 |      58.57
(2 rows)
```

Tulemus:

```
 klient_nimi |       ostu_kp       | ostu_summa
-------------+----------------------+------------
 Kadri Kell  | 2014-04-17 00:00:00 |      41.86
 Kadri Kell  | 2014-04-23 00:00:00 |      58.57
(2 rows)
```

Kokkuvõte: Testjuhtumi tulemus on positiivne.

3. SQL lausete koodi haldamise infosüsteemi integreerimise ning kasutamise protsessid

Antud peatükis kirjeldatakse SQL lausete koodi haldamise infosüsteemi rakendamise protsesse. Esimesena esitatakse näidis arendusprotsess, mida kasutatakse ettevõttes andmebaasi funktsioonides kasutatavate SQL lausete koodi muutmiseks enne käesolevas töös ehitatud üldlahenduse kasutuselevõtmist. Järgmisena pakutakse üldlahenduse olemasolevasse ettevõte süsteemi integreerimise protsess. Viimasena kirjeldatakse andmebaasi funktsioonis kasutatavate SQL lausete koodi muutmise protsessi, mis on võimalik pärast SQL lausete koodi haldamise infosüsteemi juurutamist.

Protsessid on kirjeldatud võttes arvesse, et ettevõttes on kolm keskkonda.

- Arenduskeskkond
- Testkeskkond
- Töökeskkond

Arenduskeskkond on keskkond, kus on olemas kõik tarkvarasüsteemi arendamiseks vajalikud rakendused ja süsteemid (e-uni.ee). Andmebaasi arenduskeskkond on eraldiseisev andmebaasiserver, millel on enamasti samad seadistused nagu testkeskkonna serveris ja tegelikus töökeskkonna serveris.

Testkeskkond on vahelüli arenduse ja tegelikku töökeskkonna vahel (e-uni.ee). Testkeskkond kasutab samu tarkvara versioone nagu töökeskkond. Andmebaasi testkeskkond on eraldiseisev andmebaasiserver, millel on samad seadistused nagu tegelikus töökeskkonna serveris. Testkeskkonna andmebaas sisaldab andmeid, mis on perioodiliselt töökeskkonnast kopeeritud, juhul kui ei ole tegemist konfidentsiaalsetega andmetega. Kui andmed on konfidentsiaalsed, siis testkeskkonna andmebaas sisaldab käsitsi sisestatud või automaatselt genereeritud testandmeid.

Töökeskkond on „keskkond, kus toimub tegelik töö, st mida firma tarvitab oma igapäevases äritegevuses“ (e-uni.ee).

Tabelis 9 on kirjeldatud protsessides osalejate rollid. Iga sellise rolli esindajad on seotud vähemalt ühe ülalnimetatud protsessiga kolmest.

Tabel 9. Protsesside osalejate rollid

Rolli nimi	Rolli kirjeldus
Ärianalüütik	Töötaja, kes analüüsib ja kaardistab vajadusi ning koostab analüütikule äritellimuse.
Analüütik-disainer (edasi analüütik)	Töötaja, kes analüüsib tellimuses esitatud nõudeid ning kirjeldab ärinõuete rahuldamiseks mõeldud süsteemi tehnilise lahenduse (sh koostab andmebaasi andmemudeli) ja koostab arendajale ülesandepüstituse. Tehnilise lahenduse kirjeldus on disaini dokumendis. Vajadusel nõustab testimisetapi käigus testijat.
Arendaja	Töötaja, kes kirjutab analüütiku poolt koostatud disaini dokumentatsiooni järgi SQL koodi. Vastutab koodi kvaliteedi eest ja vajadusel toetab testimise faasi.
Testija	Arenduse kvaliteedi kontrollija, kes valib testimiseks sobiva lähenemise, kirjutab ja käivitab testjuhtumeid, analüüsib ja dokumenteerib tulemusi ning raporteerib leitud vigadest.
Administraator	Töötaja, kes käivitab arendaja poolt kirjutatud SQL koodi skriptid test- või töökeskkonnas.

Tabelis 10 on kirjeldatud protsesside töötulemused, mida kasutatakse vähemalt ühes ülalnimetatud kolmest protsessist.

Tabel 10. Protsesside töötulemused

Töötulemuse nimetus	Töötulemuse kirjeldus
Äritellimus	Dokument, kus on kirjeldatud ärivaldkonna tellimus loodavale süsteemile. Sisaldab vajalike ärireeglite ja nende muudatuste sõnalist kirjeldust.
Disaini dokumentatsioon	Dokument, mis kirjeldab funktsioonide loogikat. Dokument sisaldab iga funktsiooni kohta: * funktsiooni sisend- ning väljundparameetrid, * kõik vajalikud tabelid, kust tuleb andmeid võtta, * seosed tabelite vahel, * tingimused, mille järgi tuleb andmed tabelitest võtta,

Töötulemuse nimetus	Töötulemuse kirjeldus
	* valemid, mis teisendavad tabelitest võetud andmed.
Disaini kohta saadud tagasiside	Arendaja kommentaarid, kus analüütik võiks disaini täpsustada
Funktsiooni loomise skript	PL/pgSQL koodi skript andmebaasi funktsiooni loomise lausega.
Testplaan	Dokument, kus on kirjeldatud mis objekte on plaanis testida ja kuidas seda teha
Testjuhtumid	Dokument, kus on kirjeldatud testjuhtumid. Sisaldab: * testitava objekti nimetus, * testjuhtumi eesmärk, * testjuhtumi sisu (kuidas/mida tuleb käivitada), * testjuhtumi oodatav tulemus, * testjuhtumi käivitamise tulemus.
Vea kirjeldus	Testimise käigus leitud vea tekstiline kirjeldus, mis sisaldab testitava objekti nime, käivitatud testjuhtumi identifikaatorit ning testjuhtumi tulemust.
Testaruanne	Dokument, mis sisaldab käivitatud testjuhtumid, nende tulemused ning testimise käigus leitud vead.
Muudetud funktsioon arenduskeskkonnas	Andmebaasi funktsioon, mis on käivitatav arenduskeskkonnas.
Muudetud funktsioon testkeskkonnas	Andmebaasi funktsioon, mis on käivitatav testkeskkonnas.
Muudetud funktsioon töökeskkonnas	Andmebaasi funktsioon, mis on käivitatav töökeskkonnas.
Funktsiooni tulemus	Funktsiooni käivitamisel tagastatud väärtused.
SQL lause kood infosüsteemi andmebaasis	Infosüsteemi andmebaasis salvestatud SQL lause
SQL lausete koodi haldamise infosüsteemi andmebaasi skeem arenduskeskkonnas	Arenduskeskkonna andmebaasi skeem SQL lausete koodi haldamise infosüsteemi metaandmetega, mille alusel saab SQL lauseid koostada.
SQL lausete koodi haldamise	Testkeskkonna andmebaasi skeem SQL lausete koodi

Töötulemuse nimetus	Töötulemuse kirjeldus
infosüsteemi andmebaasi skeem testkeskkonnas	haldamise infosüsteemi metaandmetega, mille alusel saab SQL lauseid koostada.
SQL lausete koodi haldamise infosüsteemi andmebaasi skeem töökeskkonnas	Töökeskkonna andmebaasi skeem SQL lausete koodi haldamise infosüsteemi metaandmetega, mille alusel saab SQL lauseid koostada.

3.1. Andmebaasi funktsioonis kasutatava SQL lause koodi muutmise näidisprotsess

Antud jaotises tuuakse näide arendusprotsessist, mis võib toimuda ettevõtte andmebaasi funktsioonis kasutatava SQL lause muudatuse tegemiseks. Oletame, et olemasolevas andmebaasi funktsiooni SQL lauses on vaja natuke muuta ühte tingimust. Lause koodis on näiteks tingimus:

```
WHERE KLIENDI_TÜÜP = 6
```

Uue reegli järgi peab tingimus olema näiteks:

```
WHERE KLIENDI_TÜÜP = 7
```

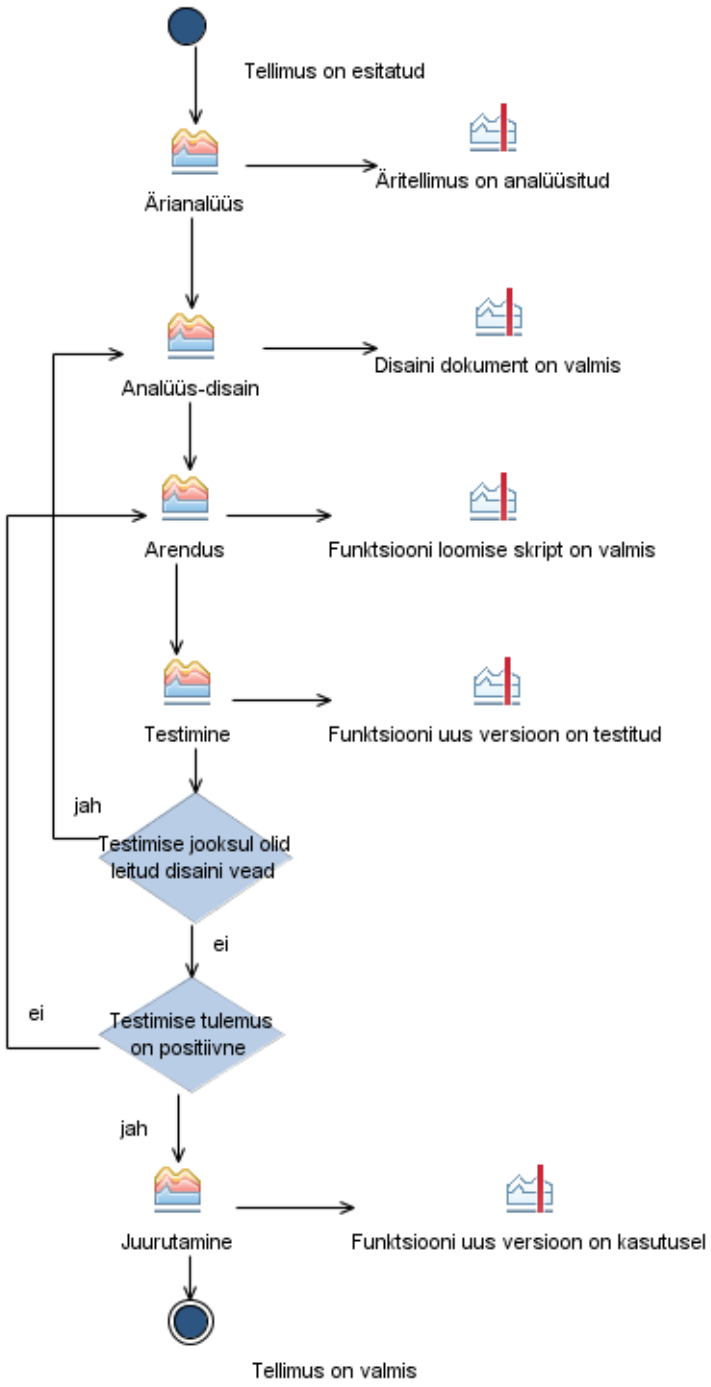
Antud protsess kirjeldab, missugused tegevused on vaja läbi viia sellise muudatuse teostamiseks ilma SQL lausete koodi haldamise infosüsteemi kasutamiseta.

See on lihtsustatud arendusprotsess, mis oli ehitatud põhinedes Rational Unified Process (RUP) raamistikul (sce.uhcl.edu, 2002).

Käesoleva töö raames huvitavad meid järgmised RUP raamistiku distsipliinid:

- Nõudmiste analüüs ehk Ärianalüüs
- Analüüs-disain
- Arendus
- Testimine
- Juurutamine

3.1.1. Protsessi üldine tegevusdiagramm



Joonis 19. Andmebaasi funktsioonis kasutatava SQL lause koodi muudatuse tegemise protsessi üldine tegevusdiagramm

Joonisel 19 esitatakse andmebaasi funktsioonis kasutatava SQL lause muutmise protsess, mis koosneb ärianalüüsi, analüüs-disaini, arenduse, testimise ning juurutamise faasidest. Antud protsess toimib tingimusel, et töökeskkonnas on juba olemas kehtiv funktsiooni

versioon, ning seda on vaja reeglite muudatuse pärast muuta. Kuna funktsioon on juba installeeritud, siis see on juba ärikasutajate poolt mingi kasutajaliidese kaudu kasutatav. Eeldame, et funktsiooni sisend- ja väljundparameetrid ei muutu. Seega ei ole vajadust muuta seda funktsiooni kasutavaid rakendusi. Sellisel juhul on antud protsessi eesmärk paigaldada funktsiooni uus versioon andmebaasi töökeskkonda ning teavitada sellest ärikasutajaid. Selline muudatus võib olla vajalik näiteks reeglis kasutatava konstandi muutumisest, mis võib olla omakorda tingitud ettevõte välistingimustest.

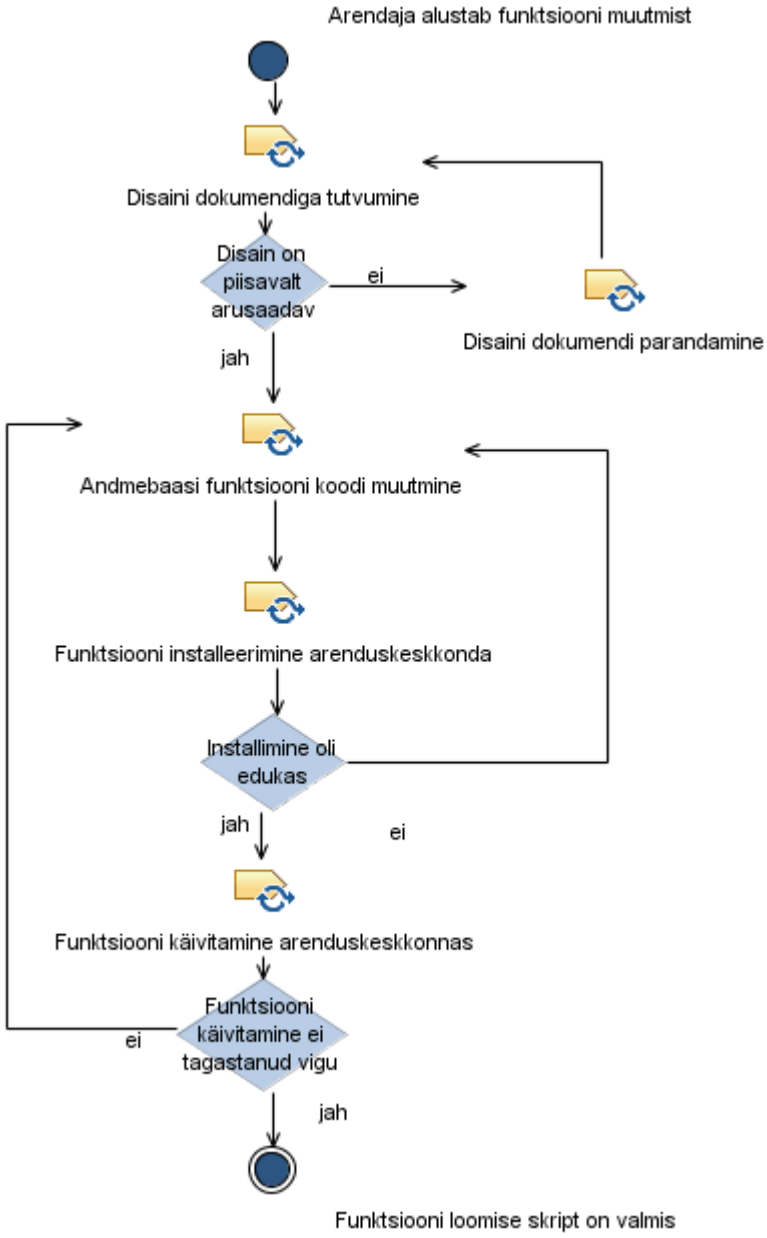
Ärianalüüsi jooksul võtab ärianalüütik vastu tellimuse reegli tingimuse muutmise kohta, kogub vajadusel lisainformatsiooni, vormistab äritellimuse ning edastab selle analüütikule.

Analüüs-disaini faasi jooksul loeb analüütik äritellimust ning muudab vajaliku funktsiooni kohta käivat disaini dokumentatsiooni osa.

Antud töös realiseeritava üldlaheduse kasutamine ei mõjuta ärianalüüsi ning analüüs-disaini faase, sellepärast on need esitatud ainult üldisel tegevusdiagrammil.

Arenduse, testimise ning juurutamise faase kirjeldatakse järgmistes jaotistes detailsemalt.

3.1.2. Arenduse faas



Joonis 20. Arenduse faasi tegevusdiagramm

Joonisel 20 esitatakse arenduse faasi tegevused. Faasi lõpptulemus on PL/pgSQL keeles kirjutatud funktsiooni loomise lause, mis loob vastavas keskkonnas andmebaasi funktsiooni vastavalt uutele reeglitele.

Järgnevalt kirjeldatakse arenduse faasi tegevusi detailsemalt.

Tegevus: Disaini dokumendiga tutvumine

Teostajad: Arendaja

Sisendid: Disaini dokumentatsioon

Väljundid: Disaini kohta tagasiside

Kirjeldus: Disaini dokumendi läbi lugemine eesmärgiga muuta andmebaasi funktsiooni koodi.

Tegevus: Disaini dokumendi parandamine

Teostajad: Analüütik

Sisendid: Disaini kohta tagasiside

Väljundid: Disaini dokumentatsioon

Kirjeldus: Disaini dokumentatsiooni täiendamine arendaja kommentaaride alusel. Seda teeb analüütik, sest tema vastutab disaini dokumentatsiooni eest.

Tegevus: Andmebaasi funktsiooni koodi muutmine

Teostajad: Arendaja

Sisendid: Disaini dokumentatsioon, Funktsiooni loomise skript

Väljundid: Parandatud funktsiooni loomise skript

Kirjeldus: Andmebaasi funktsioonis kasutatava SQL lause koodi muutmine vastavalt disaini dokumendile.

Tegevus: Funktsiooni installeerimine arenduskeskkonda

Teostajad: Arendaja

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon arenduskeskkonnas

Kirjeldus: Funktsiooni loomise lause (CREATE FUNCTION) käivitamine andmebaasi arenduskeskkonnas. Lause on kirjutatud PL/pgSQL protseduurkeeles.

Tegevus: Funktsiooni käivitamine arenduskeskkonnas

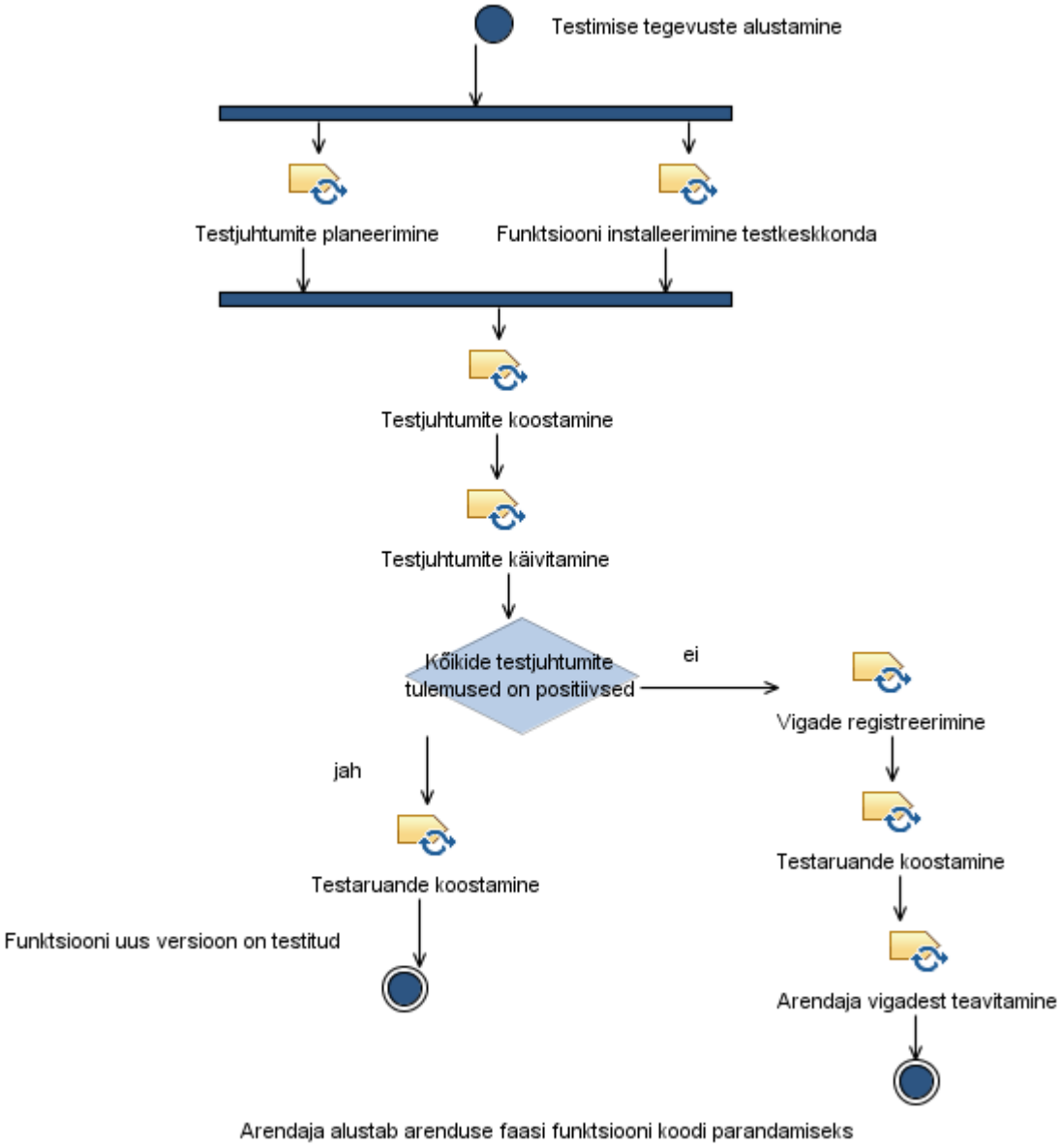
Teostajad: Arendaja

Sisendid: Muudetud funktsioon arenduskeskkonnas

Väljundid: Funktsiooni tulemus

Kirjeldus: Funktsiooni väljakutsumise lause käivitamine andmebaasi arenduskeskkonnas. Eesmärk on veenduda, et funktsiooni käivitamisel ei esine vigu.

3.1.3. Testimise faas



Joonis 21. Testimise faasi tegevusdiagramm

Joonisel 21 esitatakse testimise faasi tegevused. Kui testimise käigus leiti vähemalt üks viga, siis on sellest vaja teavitada arendajat. Sellisel juhul alustab arendaja koodi parandamiseks uuesti arenduse faasi. Kui testimise tulemusel ei leitud ühtegi viga, siis faasi tulemuseks on testitud uus funktsiooni versioon.

Järgnevalt kirjeldatakse testimise faasi tegevusi detailsemalt.

Tegevus: Testjuhtumite planeerimine

Teostajad: Testija

Sisendid: Disaini dokumentatsioon

Väljundid: Testplaan

Kirjeldus: Testimistegevuste planeerimine (mida ja mis järjekorras testida).

Tegevus: Funktsiooni installeerimine testkeskkonda

Teostajad: Administraator

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon testkeskkonnas

Kirjeldus: Arendaja poolt valmistatud skriptide käivitamine andmebaasi testkeskkonnas.

Tegevus: Testjuhtumite koostamine

Teostajad: Testija

Sisendid: Disaini dokumentatsioon, äritellimus

Väljundid: Testjuhtumid

Kirjeldus: Andmebaasi funktsiooni tulemuste testimiseks vajalikkude testjuhtumite kirjutamine.

Tegevus: Testjuhtumite käivitamine

Teostajad: Testija

Sisendid: Testjuhtumid

Väljundid: Testjuhtumite käivitamise tulemusena saadud info

Kirjeldus: Testimiseks kirjutatud testjuhtumite andmebaasis käivitamine.

Tegevus: Vigade registreerimine

Teostajad: Testija

Sisendid: Testjuhtumid

Väljundid: Vea kirjeldus

Kirjeldus: Leitud vigade kirjeldamine

Tegevus: Arendaja vigadest teavitamine

Teostajad: Testija

Sisendid: Vea kirjeldus

Väljundid: -

Kirjeldus: Arendaja teavitamine leitud vigadest. Vigade kirjelduste arendajale edastamine.

Tegevus: Testaruande koostamine

Teostajad: Testija

Sisendid: Testjuhtumid

Väljundid: Testaruanne

Kirjeldus: Käivitatud testjuhtumite tulemuste dokumenteerimine.

3.1.4. Juurutamise faas

Joonis 22. Juurutamise faasi tegevusdiagramm

Joonisel 22 on näha, et käesoleva protsessi juurutamise faas koosneb ühest tegevusest „Funktsiooni installeerimine töökeskkonda“. Kuna antud protsess on kirjeldatud eeldusel, et muudetud funktsiooni kasutamine ei muutu (ei ole vaja muuta funktsiooni kasutatavat rakendust), siis lõpptulemuse saavutamiseks piisab sellest ühest tegevusest. Kohe peale funktsiooni installeerimist töökeskkonda hakkab rakendus kasutama funktsiooni uut versiooni.

Järgnevalt kirjeldatakse juurutamise faasi tegevust detailsemalt.

Tegevus: Funktsiooni installeerimine töökeskkonda

Teostajad: Administraator

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon töökeskkonnas

Kirjeldus: Arendaja poolt valmistatud skriptide käivitamine andmebaasi töökeskkonnas.

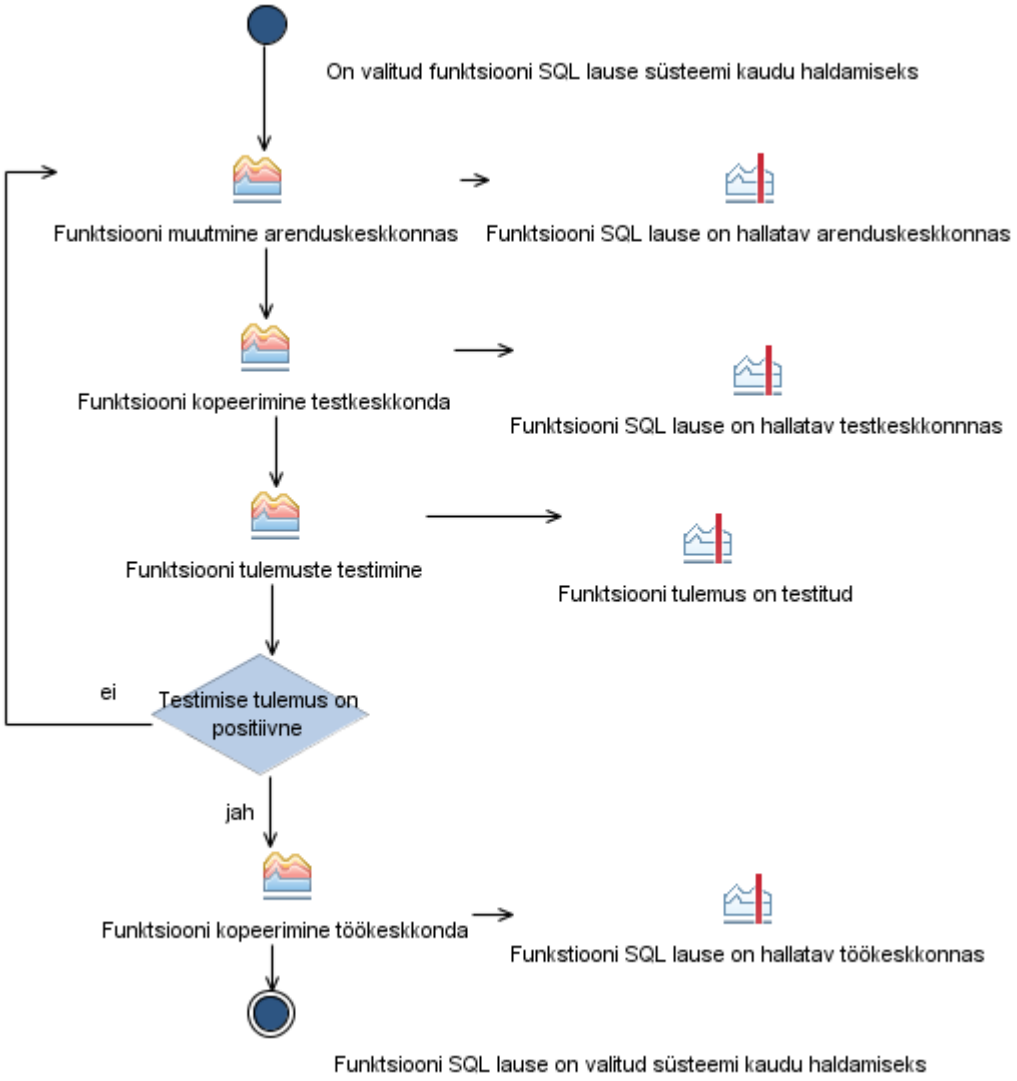
3.2. SQL lausete koodi haldamise infosüsteemi integreerimise protsess

Antud jaotises esitatakse protsess, mis aitab integreerida SQL lausete koodi haldamise infosüsteemi olemasoleva ettevõte infosüsteemi tarkvaraga. Protsess kirjeldab, missuguseid tegevusi on vaja teha, et ettevõte infosüsteemi andmebaasi funktsioonides kasutatav SQL lausete kood oleks juhitav antud töös realiseeritava infosüsteemi kaudu. Kuna eesmärk on hallata ainult andmebaasi funktsioonide SQL lausete koodi, siis infosüsteemi rakendust ei ole vaja muuta. Seega integreerimise protsess koosneb ühe või mitme andmebaasis loodud funktsiooni muutmisest nii, et nende sees olevad SQL laused oleksid hallatavad SQL lausete koodi haldamise infosüsteemi kaudu.

Protsessi eelduseks on SQL lausete koodi haldamise infosüsteemi olemasolu igas ettevõte keskkonnas. Selle jaoks peavad arendus-, test- ning töökeskkondades olema läbi viidud järgmised sammud.

1) SQL lausete koodi haldamise infosüsteemi andmebaasi skeemi lisamine olemasoleva süsteemi rakenduse andmebaasi. Tulemuseks saadud andmebaas on integratsiooni andmebaas (Fowler). Selline andmebaas sisaldab erinevate rakenduste kasutatavaid andmeid. Sellise andmebaasi põhiline kasu on selles, et ühe rakenduse kaudu tehtud muudatused on kohe kättesaadavad teiste rakenduste jaoks ning ühise andmebaasi kaudu saavutatakse rakenduste integratsioon.
2) SQL lausete koodi haldamise infosüsteemi PHP rakenduse paigaldamine olemasoleva süsteemi rakendusserverile.
3) SQL lausete koodi haldamise infosüsteemi PHP rakenduse seadistamine esimeses sammus installeeritud andmebaasi skeemi kasutamiseks.

3.2.1. Protsessi üldine tegevusdiagramm



Joonis 23. SQL lausete koodi haldamise infosüsteemi integreerimise protsessi üldine tegevusdiagramm

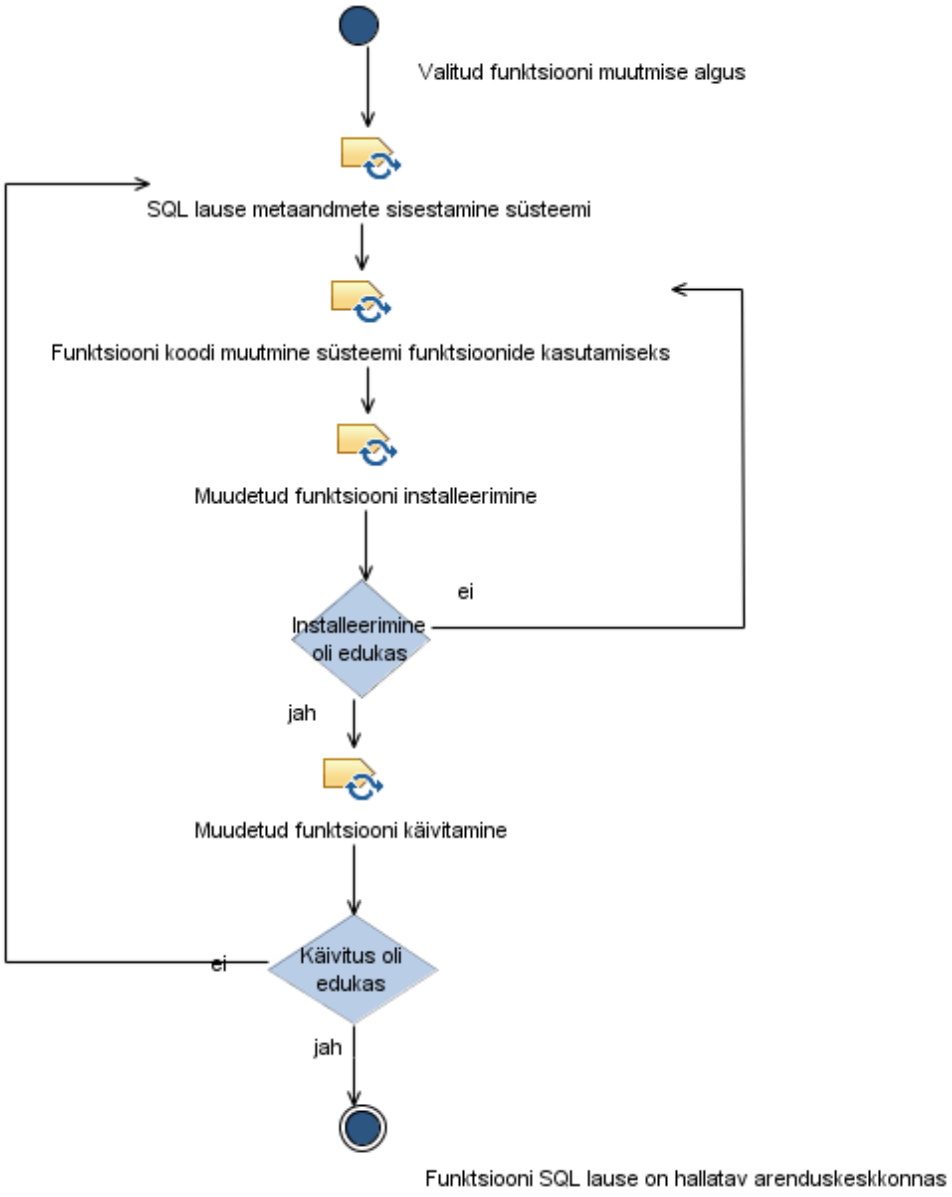
Antud töö üldlahenduse olemasoleva süsteemiga integreerimise protsess koosneb üldiselt valitud andmebaasi funktsiooni muutmisest nii, et selle SQL lausete kood oleks hallatav SQL lausete koodi haldamise infosüsteemi kaudu. Joonisel 23 esitatakse selle protsessi üldine tegevusdiagramm.

Protsessi alustamiseks peab olema valitud üks või mitu funktsiooni, mille SQL lausete koodi soovitakse hallata SQL lausete koodi haldamise infosüsteemi kaudu.

Protsessi esimeses faasis toimub valitud andmebaasi funktsiooni muutmine arenduskeskkonnas. Järgmises faasis toimub muudetud funktsiooni kopeerimine testkeskkonda ning selle testimine. Lõpuks toimub funktsiooni kopeerimine töökeskkonda.

Järgnevalt kirjeldatakse iga protsessi faasi detailsemalt.

3.2.2. Funktsiooni muutmine arenduskeskkonnas



Joonis 24. Funktsiooni arenduskeskkonnas muutmise faasi tegevusdiagramm

Joonisel 24 esitatakse tegevused, mis on vajalikud funktsiooni muutmiseks, et selle SQL lause kood oleks hallatav arenduskeskkonnas SQL lausete koodi haldamise infosüsteemi kaudu.

Järgnevalt kirjeldatakse detailsemalt antud faasi tegevused.

Tegevus: SQL lause metaandmete sisestamine süsteemi

Teostajad: Arendaja

Sisendid: Funktsiooni loomise skript

Väljundid: SQL lause kood infosüsteemi andmebaasis

Kirjeldus: Valitud funktsiooni SQL lause koodi moodustamiseks vajalike metaandmete sisestamine SQL lausete koodi haldamise infosüsteemi.

Tegevus: Funktsiooni koodi muutmine süsteemi funktsioonide kasutamiseks

Teostajad: Arendaja

Sisendid: SQL lause kood infosüsteemi andmebaasis

Väljundid: Funktsiooni loomise skript

Kirjeldus: Funktsiooni koodi muutmine nii, et ta kasutaks SQL lausete koodi haldamise infosüsteemi abifunktsioone. Infosüsteemi abifunktsioonid genereerivad korrektse SQL lause, kasutades infosüsteemi andmebaasis olevaid metaandmeid, mis hakkavad funktsiooni tööd juhtima.

Tegevus: Muudetud funktsiooni installeerimine

Teostajad: Arendaja

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon arenduskeskkonnas

Kirjeldus: Muudetud funktsiooni loomise skripti käivitamine arenduskeskkonnas.

Tegevus: Muudetud funktsiooni käivitamine

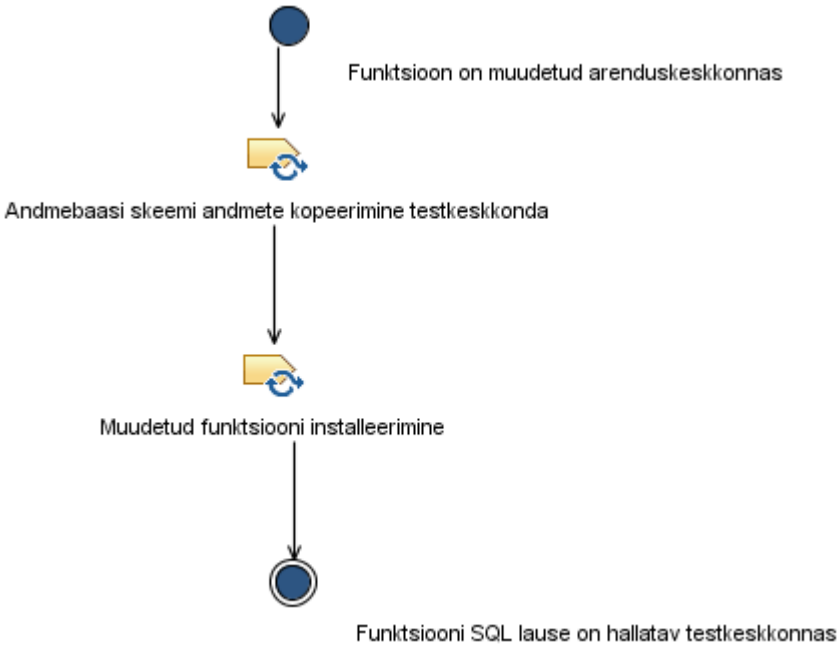
Teostajad: Arendaja

Sisendid: Muudetud funktsioon arenduskeskkonnas

Väljundid: Funktsiooni tulemus

Kirjeldus: Muudetud funktsiooni ühekordne käivitamine arenduskeskkonnas.

3.2.3. Funktsiooni kopeerimine testkeskkonda



Joonis 25. Funktsiooni testkeskkonda kopeerimise faasi tegevusdiagramm

Joonisel 25 esitatud protsessi eesmärk on teha valitud andmebaasi funktsiooni SQL lause testkeskkonnas hallatavaks.

Järgnevalt kirjeldatakse detailsemalt antud faasi tegevusi.

Tegevus: Andmebaasi skeemi andmete kopeerimine testkeskkonda

Teostajad: Administraator

Sisendid: Infosüsteemi andmebaasi skeem arenduskeskkonnas

Väljundid: SQL lausete koodi haldamise infosüsteemi andmebaasi skeem testkeskkonnas

Kirjeldus: SQL lausete koodi haldamise infosüsteemi andmebaasi skeemi andmete kopeerimine arenduskeskkonnast testkeskkonda. Kopeeritakse ainult haldamiseks valitud funktsiooni juhtimiseks vajalikud andmed.

Tegevus: Muudetud funktsiooni installeerimine

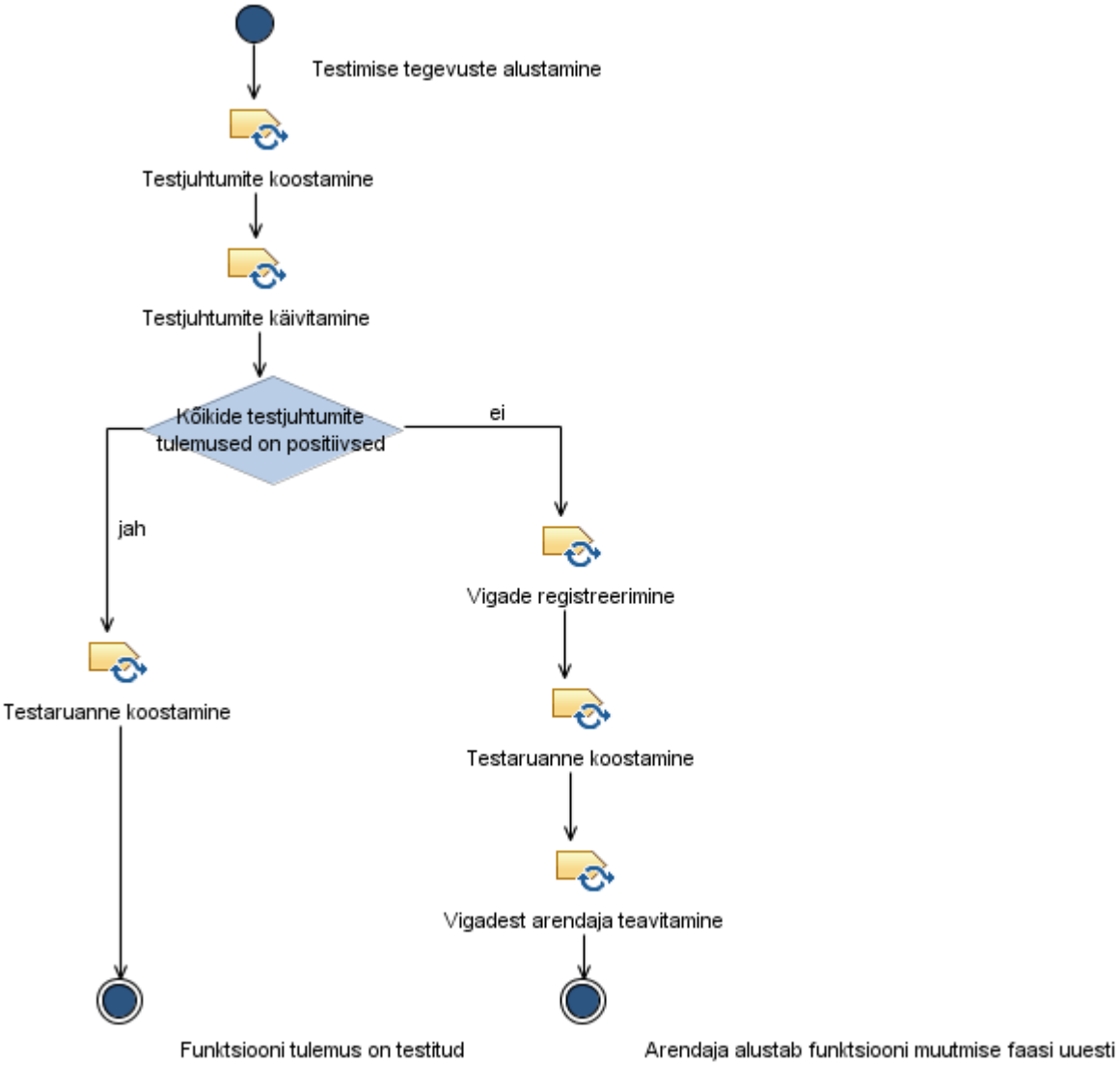
Teostajad: Administraator

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon testkeskkonnas

Kirjeldus: Muudetud funktsiooni loomise skripti käivitamine testkeskkonnas.

3.2.4. Funktsiooni tulemuste testimine



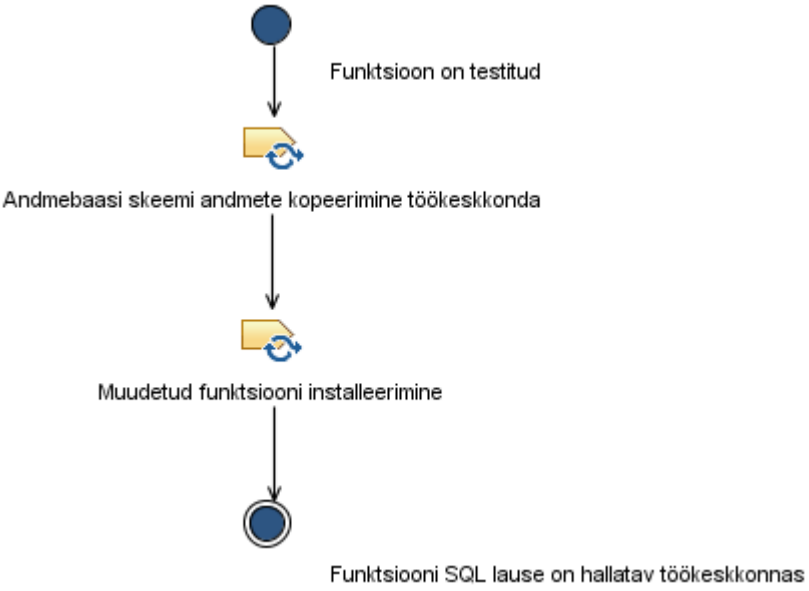
Joonis 26. Funktsiooni tulemuste testimise faasi tegevusdiagramm

Joonis 26 esitab muudetud funktsiooni testimise protsessi.

Testkeskkonnas olevad andmed on usaldusväärsemad, kui arenduskeskkonna andmed. Põhjuseks on see, et arenduskeskkonnas võib iga arendaja käivitada ükskõik millise funktsiooni, mille tulemuseks mingid andmed võivad olla vastuolus reaalsete andmetega. Sellepärast toimub integreerimise protsessi testimise osa testkeskkonnas, kus andmed vastavad rohkem töökeskkonna andmetele. Seega on testide tulemused usaldusväärsed.

Antud faasi tegevused on samad nagu eelmise protsessi testimise faasi tegevused, mida kirjeldati detailsemalt jaotises 3.1.3.

3.2.5. Funktsiooni kopeerimine töökeskkonda



Joonis 27. Funktsiooni töökeskkonda kopeerimise faasi tegevusdiagramm

Joonisel 27 esitatud protsessi eesmärk on teha valitud andmebaasi funktsiooni SQL lause töökeskkonnas hallatavaks.

Järgnevalt kirjeldatakse detailsemalt antud faasi tegevused.

Tegevus: Andmebaasi skeemi andmete kopeerimine töökeskkonda

Teostajad: Administraator

Sisendid: SQL lausete koodi haldamise infosüsteemi andmebaasi skeem testkeskkonnas

Väljundid: SQL lausete koodi haldamise infosüsteemi andmebaasi skeem töökeskkonnas

Kirjeldus: SQL lausete koodi haldamise infosüsteemi andmebaasi skeemi andmete kopeerimine testkeskkonnast töökeskkonda. Kopeeritakse ainult haldamiseks valitud funktsiooni juhtimiseks vajalikud andmed.

Tegevus: Muudetud funktsiooni installeerimine

Teostajad: Administraator

Sisendid: Funktsiooni loomise skript

Väljundid: Muudetud funktsioon töökeskkonnas

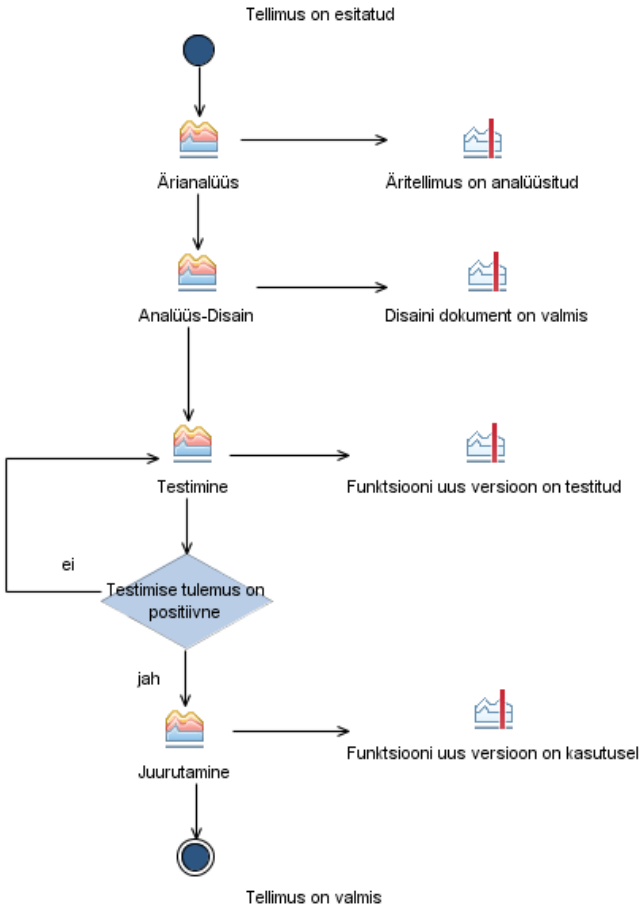
Kirjeldus: Uue funktsiooni loomise skripti käivitamine töökeskkonnas.

3.3. Andmebaasi funktsioonis kasutatava SQL lause muutmise protsess, kasutades SQL lausete koodi haldamise infosüsteemi

Antud jaotises esitatakse andmebaasi funktsioonis kasutatava SQL lause koodi muutmise protsess juhul kui kasutatakse SQL lausete koodi haldamise infosüsteemi. Protsess kirjeldab, kuidas saab muuta funktsiooni realisatsiooni, kasutades antud töö pakutavat üldlahendust.

Käesoleva protsessi eelduseks on see, et eelmises jaotises 3.2 kirjeldatud SQL lausete koodi haldamise infosüsteemi integreerimise protsess on läbiviidud.

3.3.1. Protsessi üldine tegevusdiagramm



Joonis 28. Andmebaasi funktsiooni poolt kasutatava SQL lause muutmise protsessi üldine tegevusdiagramm

Joonisel 28 esitatakse üldine tegevusdiagramm kirjeldab andmebaasi funktsiooni poolt kasutatava SQL lause muutmist antud töös realiseeritava üldlahenduse kasutamisel.

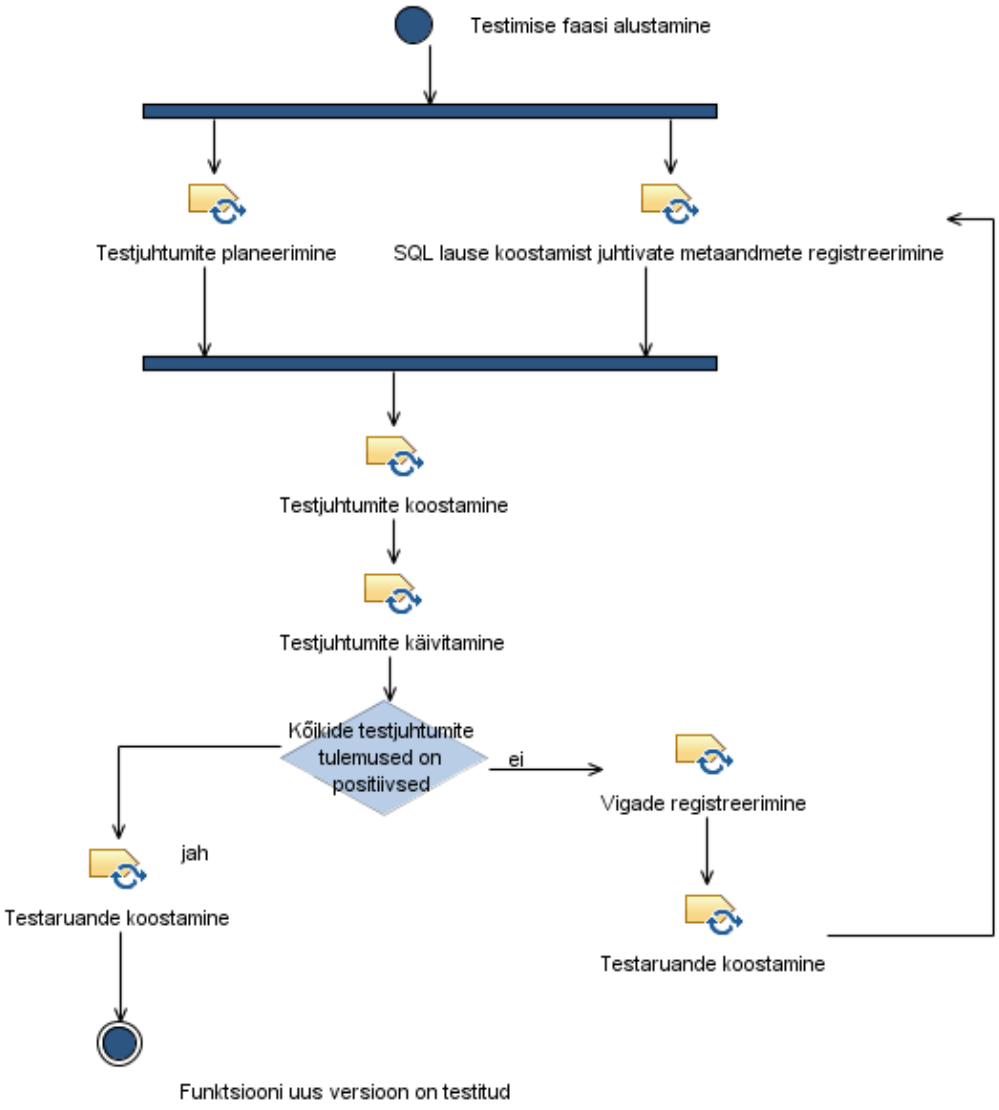
Protsess koosneb samadest faasidest nagu funktsiooni SQL lausete muutmise protsess enne antud töös realiseeritava üldlahenduse rakendamist (vt Joonis 19). Mõlemate protsesside analüüsi ja analüüs-disaini faasid on samad. Kuid erinevus on selles, et uues protsessis on puudu arenduse faas.

Antud protsessi korral on analüütik SQL lausete koodi haldamise infosüsteemi kasutaja. Analüütik ise muudab vastavalt uutele nõuetele funktsiooni tööd juhtivaid metaandmeid. Seega võib öelda, et analüütik on arendaja rollis, kuid ta teeb arendustegevusi testkeskkonnas testimise faasi alguses. Peale analüütiku poolt tehtud muudatust on funktsioonil testkeskkonnas uus funktsionaalsus. Edasi testib testija uut funktsiooni.

Kui testimise tulemused on positiivsed, siis analüütik teeb sama muudatuse töökeskkonnas juurutamise faasi raames. Selle tulemusena on töökeskkonnas uus funktsiooni versioon.

Järgnevalt on detailsemalt kirjeldatud testimise ning juurutamise faase.

3.3.2. Testimise faas



Joonis 29. Testimise faasi tegevusdiagramm

Joonisel 29 esitatakse antud protsessi testimise faasi kirjeldus. Selle faasi eesmärk on testida funktsiooni SQL lauses tehtud muudatust, veendumaks, et see vastab disaini dokumentatsioonile ning äritellimusele.

Järgnevalt kirjeldatakse antud faasi tegevusi detailsemalt.

Tegevus: SQL lause koostamist juhtivate metaandmete registreerimine

Teostajad: Analüütik

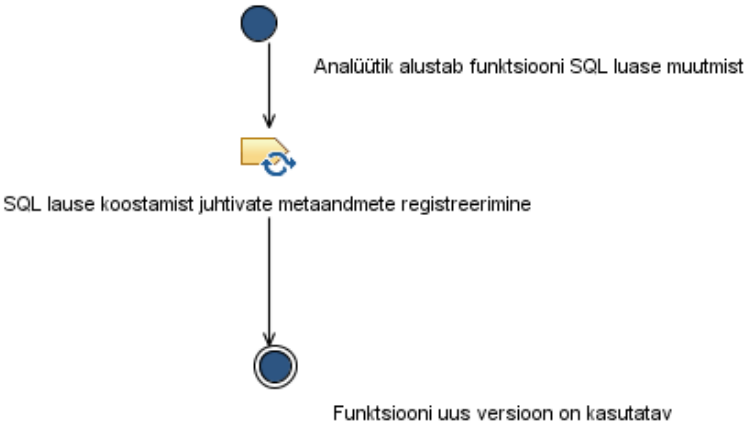
Sisendid: Disaini dokumentatsioon

Väljundid: Muudetud funktsioon testkeskkonnas

Kirjeldus: SQL lausete koodi haldamise infosüsteemi metaandmete regitreerimine ning selle kaudu funktsiooni SQL lause koodi testkeskkonnas muutmine.

Ülejäänud tegevusdiagrammil esitatud tegevused on samad, mida kirjeldati andmebaasi funktsioonis kasutatava SQL lause koodi muudatuse arendamise näidisprotsessi testimise faasi jaoks (vt jaotis 3.1.3).

3.3.3. Juurutamise faas



Joonis 30. Juurutamise faasi tegevusdiagramm

Juurutamise faasi jooksul muudab analüütik metaandmete registreerimise kaudu funktsiooni SQL lause koodi, kasutades SQL lausete koodi haldamise infosüsteemi (Joonis 30). Analüütik teeb samad muudatused, mida tehti testimise faasis.

Järgnevalt kirjeldatakse antud faasi tegevust detailsemalt.

Tegevus: SQL lause koostamist juhtivate metaandmete registreerimine

Teostajad: Analüütik

Sisendid: Disaini dokumentatsioon, Testaruanne

Väljundid: Muudetud funktsioon töökeskkonnas

Kirjeldus: SQL lausete koodi haldamise infosüsteemi metaandmete registreerimine ning selle kaudu funktsiooni SQL lause koodi töökeskkonnas muutmine.

3.4. Protsesside võrdlus

Antud jaotises võrreldakse andmebaasi funktsioonis kasutatava SQL lause koodi muutmise protsessi enne ja pärast SQL lausete koodi haldamise infosüsteemi kasutuselevõttu.

Kui võrrelda kahe protsessi üldiseid tegevusdiagramme (Joonised 19 ja 28), siis uues protsessis (pärast SQL lausete koodi haldamise infosüsteemi kasutuselevõttu) on puudu arenduse faas. Seega võrreldes vana protsessiga (enne SQL lausete koodi haldamise infosüsteemi kasutuselevõttu) ei osale arendaja otseselt uues protsessi (võib osaleda kaudselt – analüütiku konsultandina).

Kui võrrelda uue ja vana protsessi testimise faase (Joonised 21 ja 29), siis uue protsessi testimise faasis puudub funktsiooni installeerimise tegevus, mille vanas protsessis viib läbi administraator. Selle tegevuse asemel on uues protsessis funktsiooni SQL lause koodi muutmine (metaandmete haldamise abil), mille täidab analüütik.

Uue protsessi juurutamise faasi jooksul ei käivita administraator muudetud funktsiooni loomise skripti. Funktsiooni SQL lausete kood muutub kohe pärast seda, kui analüütik muudab infosüsteemi rakenduse kaudu funktsiooni SQL lausete koodi.

Võttes arvesse ülalnimetatud erinevused võib teha järgmised järeldused.

- Uus protsess võiks olla kiirem, sest sellest ei ole arenduse faasi. Uue protsessi kiiruslik eelis vana protsessi ees sõltub sellest, kui kergesti analüütik selle omaks võtab ja kasutama õpib.
- Arendaja ja administraator ei osale otseselt uues protsessis, seega nende töö maht väheneb. Kuid arendaja töömaht väheneb mitte nii palju, nagu administraatori töömaht, kuna arendaja peab vajadusel analüütiku konsulteerima.

Uue protsessi puhul on aga olemas ka ohud:

- Analüütik võib teha vigu (näiteks kanda töökeskkonda muudatusi, mis erinevad testkeskkonnas läbiproovitutest).
- Analüütiku töö maht suureneb.
- Kui analüütikul ei ole nii head SQL teadmised nagu arendajatel, siis ta võib juhuslikult mõjutada SQL koodi optimaalsust.

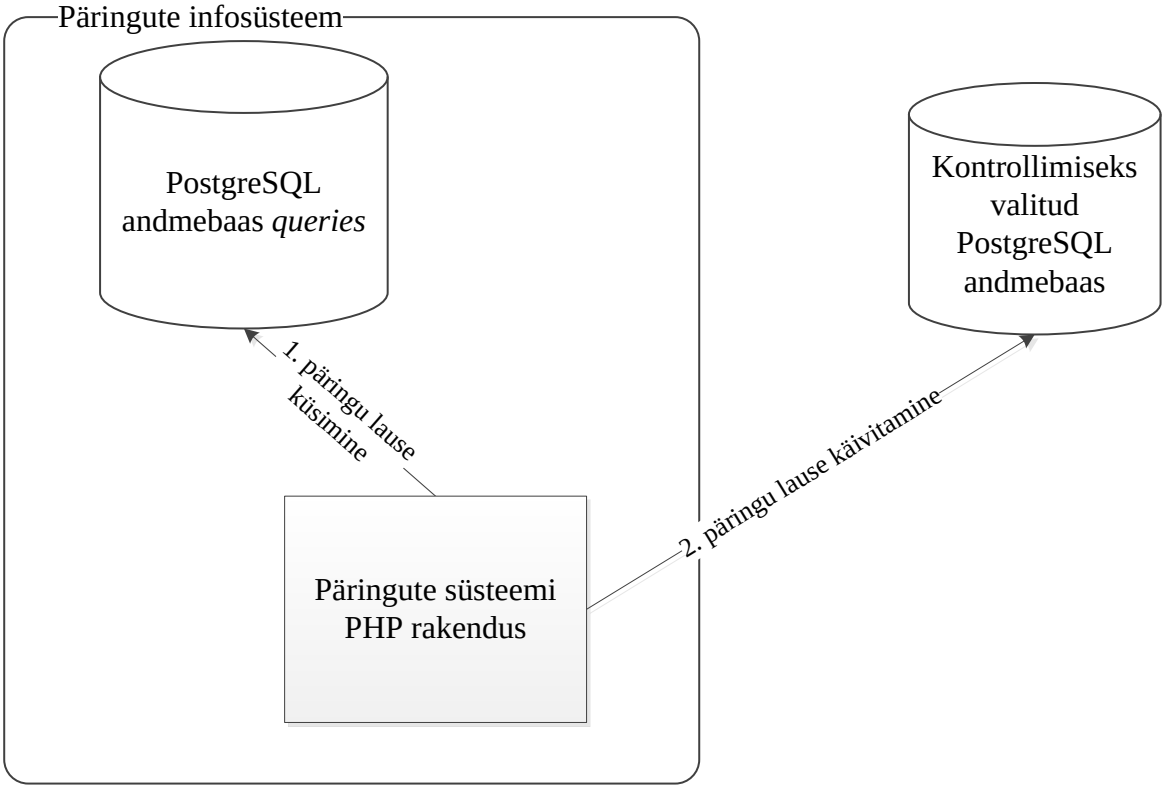
Võib tekkida küsimus, kas analüütiku asemel võiks neid muudatusi teha lõppkasutaja (lõppkasutaja võtaks üle analüütiku rolli). Põhimõtteliselt oleks see võimalik, kuid eeldab lõppkasutajalt head SQLi tundmist ning intuitiivset kasutajaliidest SQLi keerukuse varjamiseks.

4. SQL lausete koodi haldamise infosüsteemi rakendamine konkreetsele süsteemile

SQL lausete koodi haldamise infosüsteem rakendati autori poolt bakalaureusetööna loodud *päringute* süsteemile (Voronova, 2009) selleks, et demonstreerida antud töö lahenduse kasutamist konkreetse näite põhjal.

4.1. Päringute süsteem

2009. aastal kaitstud bakalaureuse lõputöö (Voronova, 2009) raames realiseerisin ma päringute süsteemi, mis oli mõeldud põhiliselt andmebaasi süsteemikataloogi põhjal tehtavate päringute abil PostgreSQL andmebaasist disainivigade otsimiseks, päringute abil andmebaasi kohta üldise informatsiooni saamiseks ning andmebaasi iseloomustavate mõõdikute väärtuste leidmiseks. Süsteemi rakenduse prototüüp asub aadressil http://apex.ttu.ee/queries3 (katsetamiseks loodud kasutajanimi = test_sql ja parool = 123456). Süsteemi andmebaas *queries* asub samamoodi apex.ttu.ee serveril ning sisaldab erinevad päringud, mis tagastavad informatsiooni valitud andmebaasi disaini kohta. Päringud kasutavad metaandmeid, millele tarkvaral on ligipääs *information_schema* andmebaasi skeemis olevate vaadete kaudu ning mõned päringud pöörduvad ka otse PostgreSQL andmebaasi süsteemikataloogi moodustavate tabelite/vaadete poole, mis on skeemis *pg_catalog*. Süsteemi rakendus võimaldab valida suvalise apex.ttu.ee serveril oleva andmebaasi ning käivitada süsteemi päringud valitud andmebaasi põhjal.



Joonis 31. Päringute infosüsteemi esialgne arhitektuur

Joonisel 31 on esitatud päringute süsteemi arhitektuur. Süsteem koosneb PHP rakendusest ning PostgreSQL andmebaasist (*queries*). PHP rakenduse kasutaja valib samal serveril asuva andmebaasi, mille kohta soovib infot saada ning valib ka päringu, mis tagastab soovitud info. Kõik päringud, mida saab süsteemis valida, asuvad *queries* andmebaasi *public* skeemi tabelis *Query*. Tabel 11 esitab mõned andmed *Query* tabelist.

Tabel 11. *Query* tabeli näiteandmed

query_id	name	description	sqlquery
13	Indexes that are not used by the database system	It is possible that a database system doesn't use some indexes. On the other hand, such indexes slow down the processes of updating data.	SELECT A.schemaname AS Schema, A.relname AS Table, A.indexrelname AS Index FROM pg_stat_user_indexes A JOIN pg_statio_user_indexes B ON A.schemaname = B.schemaname AND A.relname=B.relname AND A.indexrelname=B.indexrelname LEFT JOIN information_schema.schemata C ON A.schemaname=C.schema_name WHERE A.idx_scan=0 AND A.idx_tup_read=0 AND A.idx_tup_fetch=0 AND B.idx_blks_read=0 AND B.idx_blks_hit=0

query_id	name	description	sqlquery
			AND (A.schemaname = 'public' OR C.schema_owner<>'postgres');
14	Number of triggers in database	Triggers are used to maintain data integrity in a database by causing rejection of incorrect insertions and updates. Therefore, the number of triggers in a database gives some information about general data integrity.	SELECT COUNT(DISTINCT trigger_name) AS "Number of triggers" FROM information_schema.triggers A LEFT JOIN information_schema.schemata B ON A.trigger_schema=B.schema_name WHERE (A.trigger_schema = 'public' OR B.schema_owner<>'postgres');
21	Referential degree of a schema	Referential degree is defined as the number of foreign keys in the database schema.	SELECT table_catalog AS "Database name", table_schema AS "Schema", COUNT(*) AS "Number of foreign keys" FROM information_schema.table_constraints WHERE constraint_type = 'FOREIGN KEY' GROUP BY 1, 2;

PHP rakendus võtab valitud päringu *Query* tabeli *sqlquery* veerust ning käivitab selle kontrollimiseks valitud andmebaasi põhjal. Saadud tulemus esitatakse kasutajale rakenduse kaudu.

Käesoleva näite jaoks muudeti natukene päringute süsteemi esialgset versiooni, et demonstreerida antud töö lahenduse kasutamist. Esialgses versioonis olid kõik päringud salvestatud *Query* andmebaasi tabelisse. Nüüd aga on mõnede päringute korral lisaks salvestatud PL/pgSQL funktsiooni väljakutse, mis tagastab päringu SQL koodi. Saadud päringu lause saab käivitada valitud andmebaasi vastu ning selle päringu tulemus esitatakse päringute süsteemi kasutajale. Teiste sõnadega – kui enne pidi päringute süsteemi administraator muutma SQL lauseid käsitsi, siis nüüd on võimalik päringute süsteemi SQL lausete haldamiseks kasutada töös pakutud üldlahendust.

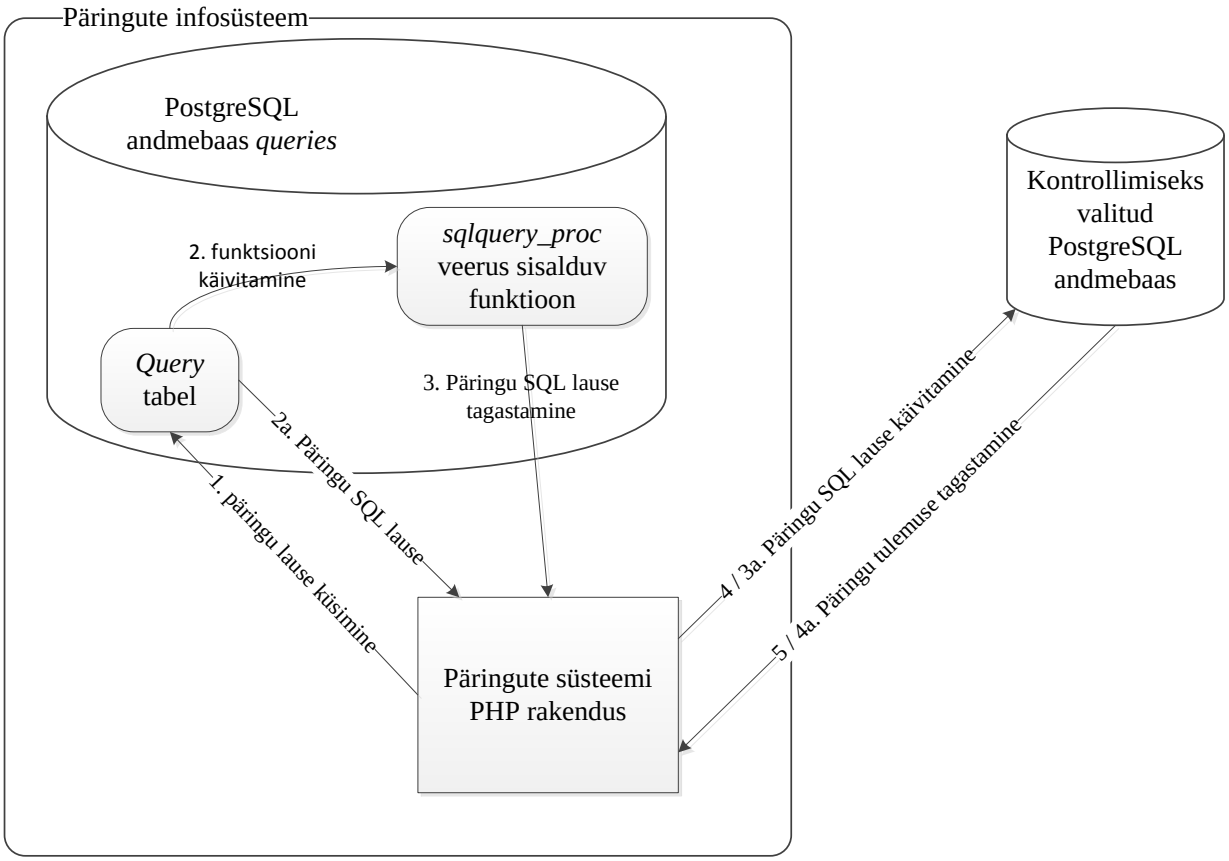
Query tabelisse on lisatud uus veerg *sqlquery_proc*, mis sisaldab päringu koodi tagastava funktsiooni väljakutse. Need funktsioonid on päringute andmebaasis *queries*. Iga funktsioon tagastab ühe konkreetse päringu SQL koodi, ning funktsiooni väljakutse jaoks loodud lause on salvestatud *sqlquery_proc* veergu. Kui päringu SQL koodi tagastamise jaoks ei ole funktsiooni loodud, siis *sqlquery_proc* veerg on tühi ning sellisel juhul kasutab süsteem

endiselt *sqlqueries* veerus olevat SQL lauset. Tabelis 12 on esitatud muudetud *Query* tabeli näiteandmed.

Tabel 12. Muudetud *Query* tabeli näiteandmed

query_id	name	description	sqlquery	sqlquery_proc
13	Indexes that are not used by the database system	It is possible that a database system doesn't use some indexes. On the other hand, such indexes slow down the processes of updating data.	SELECT A.schemaname AS Schema, A.relname AS Table, A.indexrelname AS Index FROM pg_stat_user_indexes A JOIN pg_statio_user_indexes B ON A.schemaname = B.schemaname AND A.relname=B.relname AND A.indexrelname=B.indexrelname LEFT JOIN information_schema.schemata C ON A.schemaname=C.schema_name WHERE A.idx_scan=0 AND A.idx_tup_read=0 AND A.idx_tup_fetch=0 AND B.idx_blks_read=0 AND B.idx_blks_hit=0 AND (A.schemaname = 'public' OR C.schema_owner<>'postgres');	SELECT sql_text FROM f_get_sql_query_indexes_not_used ();
14	Number of triggers in database	Triggers are used to maintain data integrity in a database by causing rejection of incorrect insertions and updates. Therefore, the number of triggers in a database gives some information about general data integrity.	SELECT COUNT(DISTINCT trigger_name) AS "Number of triggers" FROM information_schema.triggers A LEFT JOIN information_schema.schemata B ON A.trigger_schema=B.schema_name WHERE (A.trigger_schema = 'public' OR B.schema_owner<>'postgres');	NULL
21	Referential degree of a schema	Referential degree is defined as the number of foreign keys in the database schema.	SELECT table_catalog AS "Database name", table_schema AS "Schema", COUNT(*) AS "Number of foreign keys" FROM information_schema.table_constraints WHERE constraint_type = 'FOREIGN KEY' GROUP BY 1, 2;	NULL

Joonisel 32 on kirjeldatud muudetud päringute süsteemi arhitektuur.



Joonis 32. Päringute infosüsteemi muudetud arhitektuur

Joonise 32 järgi pöördub päringute süsteemi PHP rakendus alguses *Query* tabeli poole ning otsib sealt kasutaja poolt valitud päringu (1. nool joonisel 32). Edasi, kui *Query* tabelis *sqlquery_proc* veerus on leitud päringu korral väärtus, siis käivitatakse veerus viidatav funktsioon (2. nool joonisel). Funktsioon tagastab PHP rakendusele päringu SQL lause (3. nool joonisel 32). Seejärel käivitab PHP rakendus saadud SQL lause valitud andmebaasis (4. nool joonisel 32). Lõpuks saab rakendus kätte lause tulemuse ning väljastab selle kasutajale (5. nool joonisel 32).

Juhul kui *Query* tabelis *sqlquery_proc* veerus ei ole funktsiooni päringu koodi tagastamiseks, siis PHP rakendus käitub samamoodi nagu esialgses versioonis. Rakendus võtab *sqlquery* veerus oleva päringu kood (2a. nool joonisel 32), käivitab selle kontrollimiseks valitud andmebaasi peal (3a. nool joonisel 32) ning väljastab päringu tulemuse kasutajale (4a. nool joonisel 32).

Kuna SQL lausete koodi haldamise infosüsteem tegeleb PL/pgSQL funktsioonides sisalduvate SQL lausetega, siis SQL lausete haldamise süsteemi kasutamise demonstreerimiseks oli vaja päringute süsteemi kohandada (st antud juhul oli vaja teha muudatus ka päringuid käivitavas PHP rakenduses).

4.2. SQL lausete koodi haldamise infosüsteemi rakendamine päringute süsteemile

Antud jaotises kirjeldatakse SQL lausete koodi haldamise infosüsteemi integreerimise päringute süsteemi. Selle jaoks kasutati integreerimise protsessi, mida kirjeldati jaotises 3.2.

4.2.1. SQL lausete koodi haldamise infosüsteemi installeerimine

Enne integreerimise protsessi algust tuleb eelnevalt installeerida SQL lausete koodi haldamise infosüsteem päringute süsteemi keskkonda. Selle jaoks viidi läbi järgmised sammud.

1) SQL lausete andmebaasi skeemi installeerimine päringute süsteemi andmebaasi. Tulemusena on päringute süsteemi andmebaasis *queries* olemas andmebaasi skeem *laused.*
2) PHP rakenduse installeerimine. Tulemusena on kättesaadav PHP rakendus, mis asub aadressil http://apex.ttu.ee/sql_code_ms_queries (katsetamise jaoks kasutajanimi = test_sql ja parool = 123456).
3) SQL lausete haldamise PHP rakenduse seadistamine. Selle jaoks muudeti PHP rakenduse faili login.php. Selles muudeti andmebaasiga ühenduse võtmise jaoks käsku:

```
$conn_string = "host=apex.ttu.ee port=5432 user=".$kasutajanimi."
password=".$parool." dbname=queries ";
```

4.2.2. Valitud päring

SQL koodi haldamise süsteemi kaudu haldamiseks valiti SQL päring nimetusega „Indexes that are not used by the database system“. See päring tagastab info indeksite kohta, mida andmebaasisüsteem ei kasuta. Järgnevalt loodi päringule vastav funktsioon, milles sisalduvat SQL lauset soovitakse haldama hakata. Funktsioon loodi päringute andmebaasis. Funktsiooni nimi on f_`get_sql_query_indexes_not_used(OUT sql_text text)` ning funktsiooni kood on järgmine:

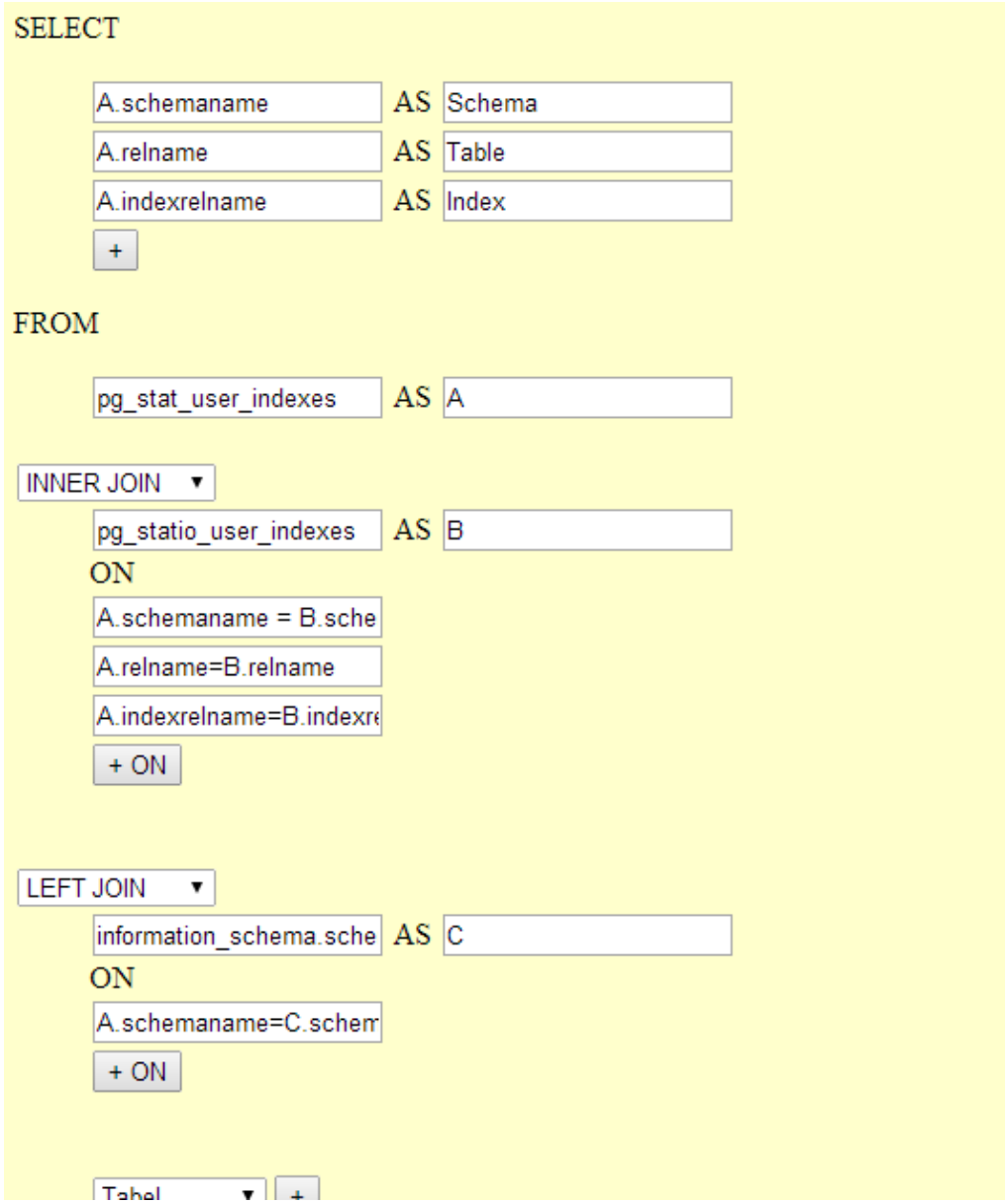
```
CREATE OR REPLACE FUNCTION f_get_sql_query_indexes_not_used (OUT sql_text
text) AS $$

DECLARE

BEGIN
      sql_text := 'SELECT A.schemaname AS Schema, A.relname AS Table,
A.indexrelname AS Index
FROM pg_stat_user_indexes A

JOIN pg_statio_user_indexes B
ON A.schemaname = B.schemaname
AND A.relname=B.relname
AND A.indexrelname=B.indexrelname

LEFT JOIN information_schema.schemata C
ON A.schemaname=C.schema_name

WHERE A.idx_scan=0
AND A.idx_tup_read=0
AND A.idx_tup_fetch=0
AND B.idx_blks_read=0
AND B.idx_blks_hit=0
AND (A.schemaname = ''public''
OR C.schema_owner<>''postgres'');'
;
END;
$$ LANGUAGE plpgsql;
```

4.2.3. Valitud päringu muutmine

Edasi sisestati loodud funktsiooni andmed SQL lausete koodi haldamise infosüsteemi ning lisaks muudeti funktsiooni nii, et see kasutaks SQL lausete haldamise süsteemi abifunktsiooni.

Selle jaoks viidi läbi sammud, mis vastavad funktsiooni muutmise faasi sammudele integreerimise protsessis (vt jaotis 3.2.2.):

1. Funktsioonis sisalduva SQL lause koodi sisestamine SQL lausete koodi haldamise infosüsteemi (Joonis 33).



Joonis 33. Süsteemi sisestatud funktsiooni kood

2. Funktsiooni koodi muutmine süsteemi abifunktsiooni kasutamiseks. Uus funktsiooni kood on järgmine:

```
CREATE OR REPLACE FUNCTION f_get_sql_query_indexes_not_used (OUT sql_text
text) AS $$
DECLARE

BEGIN

      EXECUTE 'SELECT statement_code FROM laused.f_get_stmt_code (1)'
      INTO sql_text;

END;
$$ LANGUAGE plpgsql;
```

3. Funktsiooni installeerimine. Selle jaoks käivitati eelmises punktis esitatud kood.

4. Funktsiooni käivitamine. Selle jaoks käivitati järgmine lause:

```
SELECT sql_text FROM f_get_sql_query_indexes_not_used();
```

4.2.4. Funktsiooni tulemuste testimine

Funktsiooni nimi: f_get_sql_query_indexes_not_used()

Testjuhtum: Uue funktsiooni versiooni tulemus võrdub eelmise funktsiooni versiooni tulemusega.

Testjuhtumi käik: SELECT sql_text FROM f_get_sql_query_indexes_not_used();

Oodatav tulemus: On tagastatud järgmine päringu kood:

```
SELECT A.schemaname AS Schema, A.relname AS Table, A.indexrelname AS
Index
FROM pg_stat_user_indexes A

JOIN pg_statio_user_indexes B
ON A.schemaname = B.schemaname
AND A.relname=B.relname
AND A.indexrelname=B.indexrelname

LEFT JOIN information_schema.schemata C
ON A.schemaname=C.schema_name

WHERE A.idx_scan=0
AND A.idx_tup_read=0
AND A.idx_tup_fetch=0
AND B.idx_blks_read=0
```

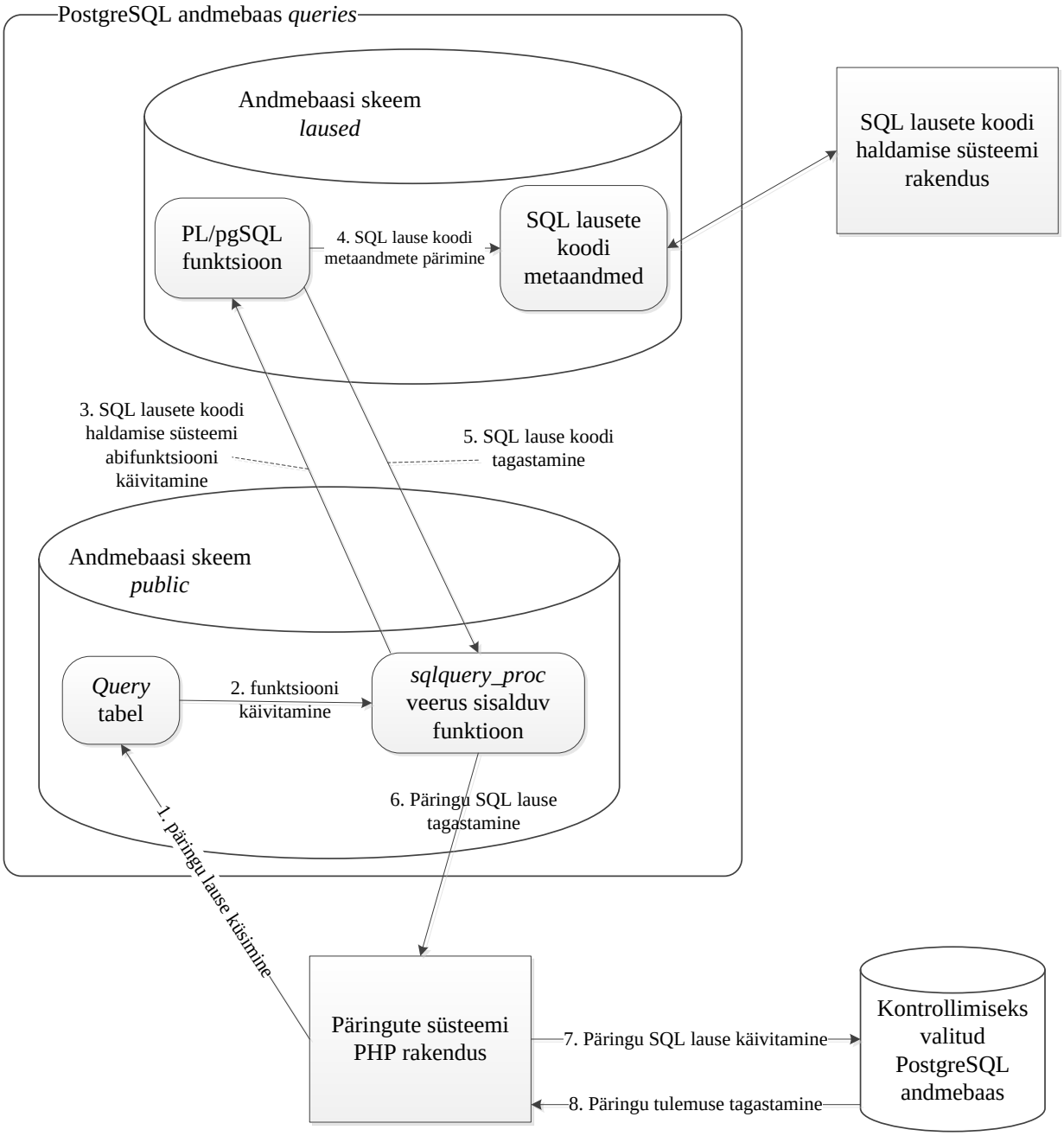
```
AND B.idx_blks_hit=0
AND (A.schemaname = 'public'
OR C.schema_owner<>'postgres');
```

Tulemus: On tagastatud oodatav päringu kood.

Kokkuvõte: Testjuhtumi tulemus on positiivne.

4.3. Päringute süsteem peale SQL lausete koodi haldamise süsteemi rakendamist

Nüüd on päringute süsteemi funktsiooni `f_get_sql_query_indexes_not_used()` poolt tagstava päringu SQL lauset võimalik muuta SQL lausete koodi haldamise süsteemi kaudu. Joonis 34 esitab päringute süsteemi arhitektuuri pärast SQL lausete koodi haldamise süsteemi rakendamist.



Joonis 34. SQL lausete koodi haldamise süsteemi kasutamine päringute süsteemis

Päringute süsteemi PHP rakendus pöördub nagu varemgi *Query* tabeli poole ning otsib sealt kasutaja poolt valitud päringut (1. nool joonisel 34). Edasi käivitatakse *Query* tabeli *sqlquery_proc* veerus viidatav `f_get_sql_query_indexes_not_used()` funktsioon (2. nool joonisel 34). Siis käivitab `f_get_sql_query_indexes_not_used()` funktsioon SQL lausete koodi haldamise süsteemi abifunktsiooni `laused.f_get_stmt_code()`(3. nool

joonisel 34). Abifunktsioon kasutab SQL lausete koodi metaandmeid (4. nool joonisel 34) SQL lause koodi genereerimiseks ning tagastab saadud SQL lause koodi päringute süsteemi funktsioonile (5. nool joonisel 34). Päringute süsteemi funktsioon (`f_get_sql_query_indexes_not_used()`) tagastab saadud SQL lause koodi päringute süsteemi PHP rakendusele (6. nool joonisel 34). Seejärel käivitab PHP rakendus saadud SQL lause kontrollimiseks valitud andmebaasis (7. nool joonisel 34). Lõpuks saab rakendus kätte lause tulemuse ning väljastab selle kasutajale (8. nool joonisel 34).

Sellise arhitektuuri puhul, niipea kui muutuvad `f_get_sql_query_indexes_not_used()` funktsioonis sisalduva SQL lause metaandmed SQL lausete haldamise süsteemis, siis hakkab päringute süsteemis `f_get_sql_query_indexes_not_used()` funktsioon kohe tagastama uuendatud SQL lauset.

SQL lausete koodi haldamise süsteemist on päringute süsteemi jaoks kasu selles, et kui peaks muutuma näiteks *information_schema* vaadete struktuur või sisu, siis päringute tagastavate funktsioonide muutmiseks oleks vaja lihtsalt muuta lausete (päringute) metaandmeid SQL lausete haldamise süsteemis, mitte muuta käsitsi SQL koodi. Samuti on võimalik, et mingi päringu sees on viga või on tekkinud täiendavad nõuded päringu poolt tagastava info kohta. Siis on samamoodi võimalik kiiresti muuta päringu tagastava funktsiooni SQL lausete haldamise süsteemi kaudu. Samuti võimaldab uus lahendus lihtsamalt läbi viia mõjuanalüüsi, sest SQL lausete metaandmete põhjal saab teha päringu leidmaks üles kõik laused, milles seda tabelit/vaadet kasutatakse.

4.3.1. Rakenduse jõudlus pärast SQL lausete koodi haldamise süsteemi rakendamist

Päringute süsteemi rakenduse jõudlus ei muutunud pärast SQL lausete koodi haldamise süsteemi rakendamist, sest päringute süsteemi andmete maht on liiga väike. Suuremate andmemahtudega süsteemide puhul tuleb aga uurida, kas SQL lausete koodi haldamise süsteemi kasutamine mõjutab nende jõudlust või mitte. Uuringu selle kohta võib teostada töö järgmistes iteratsioonides.

5. Kokkuvõte

5.1. Saavutatud tulemused

Selle töö põhitulemus on infosüsteem, mis võimaldab hallata PostgreSQL andmete lugemise ja muutmise SQL lausete koodi ja muuta selle kaudu andmebaasi funktsioonis sisalduvaid SQL lauseid. Seega SQL lause koodi muutmine tähendab andmebaasi funktsiooni muutmist. Töö teine tulemus on loodud infosüsteemi juurutamise ja kasutamise protsesside kirjeldused.

Töös pakutud lahendus realiseeriti prototüübina PostgreSQL andmebaaside jaoks. Prototüübina loodud SQL lausete koodi haldamise infosüsteemi tarkvara osa koosneb PostgreSQLi andmebaasist, andmebaasi kasutava rakenduse prototüübist ning PostgreSQL funktsioonidest. Andmebaasis hoitakse SQL lausete koodi. SQL kood on jagatud osalauseteks, ning iga osalause on jagatud osalause jaoks vajalikeks elementideks. Selline SQL lausete hoidmise vorm tähendab, et andmebaasis hoitakse metaandmeid SQL koodi kohta, mille abil saab programm jooksvalt koodi kokku panna.

Infosüsteemi rakenduse prototüüp on kirjutatud PHP skriptimiskeele abil ning asub aadressil http://apex.ttu.ee/sql_code_ms (prototüübi katsetamiseks loodud kasutajanimi = test_sql ning parool = 123456). Rakendus võimaldab kasutajatel lisada ning muuta andmeid valitud funktsioonis sisalduva SQL lause kohta. Rakenduse prototüübis on aga võimalik hallata ainult andmete otsimiseks mõeldud (SELECT) lauseid.

Andmebaasi abifunktsioonid on kirjutatud PL/pgSQL protseduurkeele abil. Abifunktsioonid tagastavad kokkupandud SQL lause, kasutades infosüsteemi andmebaasis olevaid andmeid. Abifunktsioone võib kasutada rakenduses kasutatava funktsiooni sees, kui selles funktsioonis sisalduvaid lauseid soovitakse hallata antud töö üldlahenduse kaudu. Niipea kui SQL lausete haldamise infosüsteemi kasutaja muudab valitud funktsiooni SQL lauset, salvestatakse see muudatus infosüsteemi andmebaasi. Selle tulemusena hakkab funktsioonis kehtima uus SQL lause kood, sest funktsiooni sees on kasutatud infosüsteemi abifunktsioon, mis tagastab uuendatud SQL lause koodi.

Töö teine tulemus on infosüsteemi juurutamise ja kasutamise protsesside kirjeldus. Töös on kirjeldatud loodud infosüsteemi ettevõte olemasolevasse süsteemi integreerimise protsess. Teiseks on kirjeldatud funktsiooni sees kasutava SQL lause koodi muutmise protsessi kasutades töö pakutavat üldlahendust.

Lõpuks rakendati töö üldlahendust näitena autori poolt bakalaureusetöö raames loodud „päringute“ süsteemi paindlikkuse suurendamiseks.

Töö üldlahenduse kasutajad on eelkõige andmebaasi analüütikud. Infosüsteemi kasutamiseks peavad neil olema teadmised SQL lausete koostamise reeglite kohta. Samas kui analüütik muudab ise koodi, siis on vea tekitamise oht suurem kui arendaja poolt muutmise puhul. Sellepärast sobib töö üldlahendus kõige paremini lihtsamate muudatuste puhul. Keerulisemad muudatused nõuavad rohkem SQL keele teadmisi ja seega peavad olema teostatud arendajate poolt.

Selle töö pakutava üldlahenduse kasutamine aitab kiirendada arendusprotsessi, sest koodi muudatused jõuavad töökeskkonda kohe peale vastavate koodi muudatuste tegemist üldlahenduse kaudu. Samuti väheneb arendaja töömaht, sest analüütik saab ise teha vajalikke muudatusi.

Veel üks SQL lausete koodi haldamise infosüsteemi kasutamisest saadav kasu on mõjuanalüüsi lihtsustamine. Kui on vaja leida kõik funktsioonid, milles kasutatakse konkreetset tabelit, siis selle jaoks on vaja lihtsalt teha päring infosüsteemi andmebaasi, kuna see sisaldab metaandmeid SQL koodi kohta. Selline mõjuanalüüs annab ammendavad tulemused ainult juhul kui kõikide funktsioonide SQL laused on andmebaasi sisestatud.

Peale PostgreSQl andmebaasisüsteemi on selle töö üldlahendust võimalik realiseerida ka mingis muus andmebaasisüsteemis. Selle jaoks on esiteks vaja valitud andmebaasisüsteemis luua antud töös projekteeritud andmebaasi skeemi SQL lause koodi metaandmete hoidmiseks. Teiseks on vaja realiseerida SQL lausete dünaamiline genereerimine. Kui valitud andmebaasisüsteemis on võimalik kasutada dünaamilist SQLi, siis tuleb kohandada antud töö üldlahenduse abifunktsioonide kood valitud andmebaasisüsteemi SQL dialekti kasutamiseks. Kui andmebaasisüsteem ei toeta dünaamilist SQLi, siis SQL lausete koodi dünaamiliseks genereerimiseks võib kasutada näiteks mingi ETL tööriista (ingl k *Extract, Transform, Load* ehk *ETL*), mis annab võimalust genereerida vajalikke SQL lauseid.

Antud töö üldlahendust on võimalik kasutada ka andmeaidas. Näiteks võib olla tegemist andmeaidaga, kus igapäevaselt käivitatakse erinevaid andmetöötluse SQL lauseid. Mõned SQL laused sisaldavad loogikat, mis võib muutuda mitu korda kuus. Antud töö lahendus aitaks esiteks kiirendad sellise SQL lause muutmise protsessi. Teiseks aitaks töö üldlahendus vähendada arendaja töömahtu seoses sellise SQL lause muutmisega, sest lause muudatustega tegeleks siis andmebaasi analüütik.

5.1. Edasiarendamine

Seda tööd on võimalik mitmes erinevas suunas edasi arendada. Kõigepealt saab infosüsteemi rakendust mitmel moel täiendada. Esiteks, on funktsionaalsuse laiendamiseks vaja lisada võimalus hallata lisaks SELECT lausetele ka ülejäänud andmekäitluskeele lauseid: INSERT, UPDATE ja DELETE. Samuti saab vajadusel lisada võimaluse hallata ka andmekirjelduskeele lauseid.

Lisaks võib infosüsteemi lisada võimaluse hoida koodimuudatuste ajalugu. See aitab teha analüüsi vea tekkimise puhul – leida kes vea põhjustas, millises lause osas tehtud muudatus vea põhjustas ja leida ning taastada töötav lause.

Praeguse disaini järgi võimaldab infosüsteem hallata kasutajaõiguseid ainult funktsioonide tasemel. Kuid on võimalik lisada võimalus hallata õiguseid ka detailsemalt, ehk osalausete või nende sees sisalduvate elementide tasemel või ka näiteks lause tüüpide tasemel. Siis näeks rakenduse kasutaja tervet koodi, kuid saaks muuta ainult konkreetset tingimust või teha lauseid kindlate tabelite põhjal, mille suhtes on talle õigused antud või muuta ainult kindlat tüüpi lauseid. See vähendaks vigade tekitamise riski.

Rakenduse mugavamaks muutmiseks võib lisada funktsionaalsuse, mis võtaks sisendina SQL lause koodi ja sisestaks selle süsteemi andmebaasi automaatselt. Siis ei pea arendaja käsitsi koodi sisestama või käsitsi seda muutma kui on vaja teha keerulisi muudatusi, mida analüütik ei suuda teostada.

Kindlasti on vaja teha tööd SQL lausete haldamise keskkonna kasutatavuse parandamisega, et selle võimalikel kasutajatel oleks seda lihtne õppida ja nad teeksid seal võimalikult vähe vigu.

Kasutades töö üldlahenduse andmebaasis sisalduvaid metaandmeid SQL koodi kohta saab kirjutada päringuid, mis leiaksid need koodi kohad, mida tasuks proovida refaktoreerida.

6. Summary

The main aim of this work was to create an information system that would allow us to manage PostgreSQL SQL statetements in a database. The second aim of this work was to describe the processes of implementation and usage of the information system.

The proposed solution was implemented based on PostgreSQL as a prototype. The information system that was created in this work consists of a PostgreSQL database, an application prototype that uses the database, and PostgreSQL functions. The database allows us to store metadata about SQL statements that the system can use to produce SQL statements. Each SQL statement consists of clauses and each clause is divided into specific elements that are needed for the clause. Such information represents SQL code metadada that is used by the information system's functions for dynamic SQL code generation.

The information system's application prototype is located at http://apex.ttu.ee/sql_code_ms (for testing use username = test_sql and password = 123456). The application allows its users to add and change the data about SQL statements. In the current prototype, it is possible to manage only data querying statements (SELECT).

The second result achieved in this work is the description for the processes for implementation and usage of the created information system. Fistly, we described the process of integrating the created information system into an existing organization system. Secondly, we described the process of code modification of a SQL statement using the created information system.

Finally, we used the created information system to improve the flexibility of the "queries" management system that author created as the result of her bachelor thesis.

The usage of the solution provided in this work helps its users to improve the speed of development process, because the changes made in the metadata reach the production environment immediately after the respective changes are done in using the general solution of this work. In addition, the amount of work needed to be done by a database developer decreases, because SQL stement code changes can be done by a database analyst who has the SQL knowledge needed to perform the change. The analysts have to manage the metadata that the system uses as the basis for generating SQL statements. Another possible

advantage of the solution is possibility to make queries based on the metadata of SQL statements to search statements that need refactoring or to find the impact of changing the structure of some table (what statements have to be changed).

The prototype of the system we created is based on PostgreSQL. However, the general solution proposed in this thesis can be applied also in case of other database managemet systems that support the creation of database functions/procedures and permit the usage of dynamic SQL in these.

One can develop the work further in many different ways. Firstly, one can add more functionality to the information system's application. For example, one should add opportunity to manage other data modification language (DML) statements in addition to the SELECT statement: INSERT, UPDATE, and DELETE. Definitely, one has to improve the usability of the software to make the learning of the system easier for the users.

It is also possible to add the opportunity to store the history of SQL statement changes made in the information system. It would help us to perform the analysis in case of an error – to find who caused the error, in which part of the statement it occurred, and to restore the working statetement.

7. Kasutatud materjalid

1. Mens, T., Guéhéneuc, Y., Fernández-Ramil, J., D'Hondt, M. Software Evolution, 2010.
2. en.wikipedia.org. Non-functional requirement, 2014 [WWW] http://en.wikipedia.org/wiki/Non-functional_requirement (25.04.2014)
3. en.wikipedia.org. Functional requirement, 2014 [WWW] http://en.wikipedia.org/wiki/Non-functional_requirement (25.04.2014)
4. Gelperin, D., Requirements and Business Rules: How Do They Relate?, 2009 [WWW] http://clearspecs.com/joomla15/downloads/ClearSpecs22V01_Requirements%20and%20Business%20Rules.pdf (20.04.2014)
5. blueprintsys.com. The Foundation of Good Requirements: Business Rules and Business Requirements, 2010 [WWW] http://www.blueprintsys.com/the_foundation_of_good_requirements_business_rules_and_business_requirement/ (20.04.2014)
6. en.wikipedia.org. Business rule management system, 2013 [WWW] http://en.wikipedia.org/wiki/Business_rule_management_system (10.04.2014)
7. Limeback, R., Simply SQL. SitePoint, 2008
8. sqlcourse.com. What is SQL? [WWW] http://www.sqlcourse.com/intro.html (20.04.2014)
9. en.wikipedia.org. Stored procedure, 2014 [WWW] http://en.wikipedia.org/wiki/Stored_procedure (20.04.2014)
10. wischner.blogspot.com. Creating a stored procedure / function layer in Postgres database [WWW] http://wischner.blogspot.com/2009/03/creating-stored-procedure-function.html (01.05.2014)
11. tutorialspoint.com. PostgreSQL – Functions [WWW] http://www.tutorialspoint.com/postgresql/postgresql_functions.htm (20.04.2014)
12. Downs, K., Minimize Code, Maximize Data, 2008 [WWW] http://database-programmer.blogspot.com/2008/05/minimize-code-maximize-data.html (21.04.2013)

13. Giles, J., The Nimble Elephant: Agile Delivery of Data Models using a Pattern-based Approach. Westfield: Technics Publications, 2012

14. nccc.uow.edu.au. ICD-10-AM/ACHI Mapping Tables, 2012 [WWW] http://nccc.uow.edu.au/icd10am-achi-acs/icd10ammappingtables/index.html (20.03.2013)

15. investopedia.com. Definition of 'Business Logic' [WWW] http://www.investopedia.com/terms/b/businesslogic.asp (01.05.2014)

16. von Halle, B., Goldberg, L., The Decision Model, Knowledge Partners International, LLC, 2009 [WWW] http://openrules.com/docs/DecisionModelPrimer.htm (20.04.2014)

17. pic.dhe.ibm.com. What are business rules [WWW] http://pic.dhe.ibm.com/infocenter/brdotnet/v7r1/index.jsp?topic=%2Fcom.ibm.websphere.ilog.brdotnet.doc%2FContent%2FBusiness_Rules%2FDocumentation%2F_pubskel%2FRules_for_DotNET%2Fps_RFDN_Global762.html (20.03.2014)

18. ibm.com. What is Business Rules Management? [WWW] http://www-01.ibm.com/software/websphere/products/business-rule-management/whatis/ (21.03.2013)

19. Chisholm, M., How to Build a Business Rules Engine. San Francisco: Morgan Kaufmann Publishers, 2003

20. docs.jboss.org. Drools Expert User Guide [WWW] http://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf (01.03.2014)

21. Byron, D., Understanding the benefits of Business Rules Management Software in an open Source ecosystem, 2010 [WWW] http://www.information-management.com/media/pdfs/jboss2.pdf (05.03.2013)

22. Chhatpar, A., Introduction to business rules. Taking advantage of Business Rules Management Systems, 2008 [WWW] http://www.ibm.com/developerworks/library/ar-busrules1/ (01.05.2013)

23. hartmannsoftware.com. Business Rule Management System, 2012 [WWW] http://www.hartmannsoftware.com/pub/Enterprise-Rule-Applications/brms (01.05.2013)

24. Taylor, J., Why would I use a Business Rules Management System?, RealRules Blogzine, 2006 [WWW]

http://oxygen.informatik.tu-cottbus.de/RealRules/?q=node/14 (01.03.2014)

25. openrules.com. OpenRules Overview [WWW] http://openrules.com/docs/man_2.html (01.03.2014)

26. en.wikipedia.org. Drools, 2013 [WWW] http://en.wikipedia.org/wiki/Drools (01.03.2014)

27. pic.dhe.ibm.com. Introducing ILOG JRules Business Rule Management System (BRMS) [WWW] http://pic.dhe.ibm.com/infocenter/brjrules/v7r1/index.jsp?topic=%2Fcom.ibm.websphere.ilog.jrules.doc%2FContent%2FBusiness_Rules%2FDocumentation%2F_pubskel%2FJRules%2Fps_JRules_Global7.html (01.03.2014)

28. ibm.com. Business Rules for Better Operational Decisions [WWW] http://www-01.ibm.com/software/info/business-rules/ (03.04.2014)

29. Lehman, M., Laws of Software Evolution Revisited, London: Department of Computing Imperial College, 1997 http://www.rose-hulman.edu/Users/faculty/young/CS-Classes/csse575/Resources/Lehman-LawsRevisited-96.pdf (01.05.2014)

30. Fowler, M.; Beck, K.; Brant, J.; Opdyke, W.; Roberts, D., Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999

31. Paternò, F., End User Development: Survey of an Emerging Field for Empowering People, 2013 [WWW] http://www.hindawi.com/journals/isrn/2013/532659/ (01.05.2014)

32. G. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A., Mehandjiev, N., Meta-Design: A Maniesto for End-User Development, 2004

33. techterms.com. Metadata [WWW] http://www.techterms.com/definition/metadata (01.05.2014)

34. Miksa, F., Metadata, The University of Texas in Austin, Graduate School of Library and Information Science, 2000 https://www.ischool.utexas.edu/~l38613dw/readings/Miksa-Metadata-000918.PDF (23.04.2014)

35. Voronova, J, PostgreSQL andmebaasi kohta süsteemikataloogi abil info leidmine. Bakalaureusetöö, Tallinna Tehnikaülikool, 2009

36. e-uni.ee. Süsteemi loomise keskkonnad [WWW]

http://www.e-uni.ee/e-kursused/eucip/arendus/151_ssteemi_loomise_keskkonnad.html (01.05.2014)

37. sce.uhcl.edu, Rational Unified Process: Disciplines, 2002 [WWW] http://sce.uhcl.edu/helm/rationalunifiedprocess/process/workflow/ovu_core.htm (05.05.2014)

38. postgresql.org, PostgreSQL 9.3.4 Documentation [WWW] http://www.postgresql.org/docs/9.3/interactive/sql-syntax-lexical.html#SQL-SYNTAX-SPECIAL-CHARS (05.05.2014)

39. en.wikipedia.org, Merge (SQL), 2014 [WWW] http://en.wikipedia.org/wiki/Merge_%28SQL%29 (15.05.2014)

40. Madis, A., Ärireeglite realiseerimine PHP ja PostgreSQLi abil. Bakalaureusetöö, Tallinna Tehnikaülikool, 2010

41. Õunapuu, E., Infosüsteemide genereerimise süsteem "GENSI". Tallinn: Tallinna Polütehniline Instituut, Infotöötluse kateeder, 1988

42. oracle.com, Oracle Application Express 4.2, Overview [WWW] http://www.oracle.com/technetwork/developer-tools/apex/overview/index.html (14.05.2014)

43. support.sas.com, SAS 9.2 Documentation [WWW] http://support.sas.com/documentation/cdl/en/sqlug/59570/HTML/default/viewer.htm#a001102455.htm (15.05.2014)

44. web-foro.com, Demonstration: Use Saved Queries [WWW] http://web-foro.com/wl/CompanionContent/course/crse6425b_00_02_04_07.htm (15.05.2014)

45. mifosforge.jira.com, SQL Query Repository [WWW] https://mifosforge.jira.com/wiki/display/MIFOS/SQL+Query+Repository (15.05.2014)

46. spiritsoftware.biz, SQL Query and Macro Repository [WWW] http://www.spiritsoftware.biz/sql-query-and-macro-repository/ (15.05.2014)

47. PHP Documentation, pg_query_params [WWW] http://php.net/manual/en/function.pg-query-params.php (01.03.2014)

48. bobby-tables.com, A guide to preventing SQL injection [WWW] http://bobby-tables.com/postgresql.html (05.05.2014)

49. Ganapathy, L., How to Prevent SQL Injection Attack (Explained with an Example), 2012 [WWW] http://www.thegeekstuff.com/2012/02/sql-injection-attacks/ (05.05.2014)

50. Fowler, M., Integration Database [WWW] http://martinfowler.com/bliki/IntegrationDatabase.html (15.05.2014)

51. openrules.com, Overview [WWW] http://openrules.com/overview.htm (14.05.2014)

52. docs.oracle.com, Oracle8i Application Developer's Guide - Fundamentals, Dynamic SQL [WWW] http://docs.oracle.com/cd/A87860_01/doc/appdev.817/a76939/adg09dyn.htm (15.05.2014)

53. Karch, E., Lehman's Laws of Software Evolution and the Staged-Model, 2011 [WWW] http://blogs.msdn.com/b/karchworld_identity/archive/2011/04/01/lehman-s-laws-of-software-evolution-and-the-staged-model.aspx (10.05.2014)

54. postgresql.com, dblink [WWW] http://www.postgresql.org/docs/9.3/static/dblink.html (19.05.2014)

Lisad

Lisa 1. laused.f_get_select (stmt_id int, OUT select_clause_text text) funktsiooni loomise PL/pgSQL kood

```
CREATE OR REPLACE FUNCTION laused.f_get_select(stmt_id int, OUT
select_clause_text text) AS $$
DECLARE
      counter int;
      element RECORD;
BEGIN
      select_clause_text := 'SELECT ';
      counter := 0;

      FOR element IN SELECT C.valem, B.aliase_nimi
            from laused.Select_Osalause A
            join laused.Select_Osalause_Element B
            on A.Select_Osalause_ID = B.Select_Osalause_ID
            join laused.Valem C
            ON B.Valem_ID = C.Valem_ID
            WHERE Lause_ID = stmt_id
            ORDER BY B.Select_Osalause_Element_ID
      LOOP
            counter := counter+1;
            IF counter = 1 THEN
                  select_clause_text:= select_clause_text || element.valem
|| ' AS ' || element.aliase nimi;
            ELSE
                  select_clause_text:= select_clause_text || ', ' ||
element.valem || ' AS ' || element.aliase_nimi;
            END IF;
      END LOOP;

END;
$$ LANGUAGE plpgsql;
```

Lisa 2. laused.f_get_join_conditions (from_element_id int, OUT join_cond_text text) funktsiooni loomise PL/pgSQL kood

```
CREATE OR REPLACE FUNCTION laused.f_get_join_conditions(from_element_id
int, OUT join_cond_text text) AS $$
DECLARE
      counter int;
      element RECORD;
      join_type text;
BEGIN

      EXECUTE 'select b.Join_Tyybi_Nimi
                  from laused.From_Osalause_Element a
                  join laused.Join_Tyyp b
                  on a.Join_Tyyp_ID = b.Join_Tyyp_ID
                  where a.From_Osalause_Element_ID = $1'
      INTO join_type USING from_element_id;

      IF join_type = 'FROM' THEN
            join_cond_text := 'WHERE ';
      ELSE
            join_cond_text := 'ON ';
```

```
      END IF;

      counter := 0;

      FOR element IN    select A.From_Osalause_Element_ID, A.Tingimus_ID,
B.tingimus
                  from laused.From_Osalause_Tingimus A
                  join laused.Tingimus B
                  on A.Tingimus_ID = B.Tingimus_ID
                  where A.From_Osalause_Element_ID = from_element_id
                  ORDER BY A.Tingimus_ID

      LOOP
            counter := counter+1;
            IF counter = 1 THEN
                  join_cond_text:= join_cond_text || element.tingimus ;
            ELSE
                  join_cond_text:= join_cond_text || ' AND ' ||
element.tingimus ;
            END IF;
      END LOOP;

END;
$$ LANGUAGE plpgsql;
```

Lisa 3. laused. f_get_stmt_code (stmt_id int, OUT statement_code text) funktsiooni loomise PL/pgSQL kood

```
CREATE OR REPLACE FUNCTION laused.f_get_stmt_code(stmt_id int, OUT
statement_code text) AS $$
DECLARE
      select_part text;
      element RECORD;
      conditions text;
      sub_query text;
      from_element int;

BEGIN
      EXECUTE 'select select_clause_text from laused.f_get_select($1)'
      INTO select_part USING stmt_id;
      IF select_part is not null THEN
      statement_code := select_part || ' FROM ';
      ELSE statement_code := ' FROM ';
      END IF;

      --get all the "from" elements
      FOR element IN select B.From Osalause Element ID,
                  B.Join_Tyyp_ID,
                  J.Join_Tyybi_Nimi,
                  B.Tabeli_nimi,
                  B.aliase_nimi,
                  B.Alam_Lause_ID
                  from laused.From_Osalause A
                  join laused.From_Osalause_Element B
                  on A.From_Osalause_ID = B.From_Osalause_ID
                  join laused.Join_Tyyp J
                  ON J.Join_Tyyp_ID = B.Join_Tyyp_ID
                  where A.Lause_ID = stmt_id
```

```
                  ORDER BY CASE WHEN J.Join_Tyybi_Nimi = 'FROM' THEN 0
ELSE 1 END, B.From_Osalause_Element_ID

      LOOP
            IF element.Join_Tyybi_Nimi = 'FROM' THEN --FROM

                  from_element := element.From_Osalause_Element_ID;

                  IF element.Alam_Lause_ID is null THEN
                        statement_code:= statement_code ||
element.Tabeli_nimi || ' AS ' || element.aliase_nimi;

                  ELSE --if it's a sub-query
                        statement_code:= statement_code ' ( ';
                        --add sub-query text

                        EXECUTE 'select statement_code from
laused.f_get_stmt_code($1)'
                        INTO sub_query USING element.Alam_Lause_ID;

                        statement_code:= statement_code || ' ) AS ' ||
element.aliase_nimi;

                  END IF;
            ELSE --JOIN
                  IF element.Alam_Lause_ID is null THEN
                        statement_code:= statement_code || ' ' ||
element.Join_Tyybi_Nimi || ' ' || element.Tabeli_nimi || ' AS ' ||
element.aliase_nimi;
                        -- add join conditions:

                        EXECUTE 'select join_cond_text from
laused.f_get_join_conditions($1)'
                        INTO conditions USING
element.From_Osalause_Element_ID;
                        IF conditions is not null THEN
                        statement_code:= statement_code || ' ' ||
conditions;
                        END IF;
                  ELSE

                        statement_code:= statement_code || ' ' ||
element.Join_Tyybi_Nimi || ' ( ';
                        --add sub-query text
                        EXECUTE 'select statement_code from
laused.f_get_stmt_code($1)'
                        INTO sub_query USING element.Alam_Lause_ID;

                        statement_code:= statement_code || sub_query;


                        statement_code:= statement_code || ' ) AS ' ||
element.aliase_nimi;
                        -- add join conditions:

                        EXECUTE 'select join_cond_text from
laused.f_get_join_conditions($1)'
                        INTO conditions USING
element.From_Osalause_Element_ID;
                        IF conditions is not null THEN
```

```
                        statement_code:= statement_code || ' ' ||
conditions;
                        END IF;
                  END IF;
            END IF;

      END LOOP;

      --WHERE
      EXECUTE 'select join_cond_text from
laused.f_get_join_conditions($1)'
      INTO conditions USING from_element;
      IF conditions is not null THEN
      statement_code:= statement_code || ' ' || conditions;
      END IF;

END;
$$ LANGUAGE plpgsql;
```