TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Erik Dzotsenidze 164429IAPB

POSTGRESQL VEEBIPÕHISE VISUAALSE PÄRINGUTE KOOSTAMISE TARKVARA ARENDAMINE

bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2019

Autorideklaratsioon

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

Autor: Erik Dzotsenidze

20.05.2019

Annotatsioon

Käesoleva töö eesmärgiks on luua avatud lähtekoodiga veebipõhine rakendus, mis võimaldab kasutajal koostada PostgreSQL andmebaasi põhjal visuaalselt SELECT lauseid e päringuid. Selleks loodud veebirakendus võimaldab kasutajal piisavate õiguste olemasolul ühenduda enda valitud andmebaasiga, koostada päringuid ning neid käivitada. Rakenduse eesmärgiks on pakkuda võimalust luua päringuid ilma SQL lähtekoodi kirjutamata. Töös kirjeldatakse rakenduse kavandamist, realisatsiooni, võrreldakse loodud programmi kahe sarnase programmidega ning saadakse kasutajatelt tagasisidet loodud programmi kohta.

Tarkvara tagarakenduse osa kirjutamiseks kasutati Node.js ning kasutajaliidese loomiseks React raamistikku. Rakenduse lähtekood on kaitstud MIT litsentsiga. Rakenduse lähtekood on kättessaadav GitHubi hoidlas: https://github.com/Erikdzo/postgres-visual-query-app ning loodud veebirakendus asub leheküljel http://apex.ttu.ee/postgres-query.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 46 leheküljel, 10 peatükki, 36 joonist, 1 tabelit.

Abstract

Developing PostgreSQL Visual Query Design Software

The purpose of this thesis is to create an open source and web-based application that allows users to create PostgreSQL SELECT statements, i.e., queries, by using a visual user interface. The application can be used to create SQL statements without writing SQL code. It could be used to save time while creating statements and could be used by users who are uncomfortable with writing source code, i.e., it could extend the range of users who can make themselves queries based on a database.

In todays world the Internet is used in many facets of our lives. Because of that we could expect that there is a web-based tool for building database queries. Since more and more data is stored in databases, there is a growing amount of tasks that can be solved by using that data.

This application allows users to connect to a PostgreSQL database and build queries by using a graphical user interface. After constructing a query, the user can execute this and the application displays the results. User also sees the corresponding SQL select statement. A precondition is that the user has neccessary privileges in the database. The application supports multiple languages of the user interface.

The method of research is design science. The result of the work is a technical artifact – a ready to use web-based application that uses React and Node.js to achieve the previously mentioned goals. The thesis describes requirements to the application as well as technical details of its implementation. One has to validate the produced artifact. Thus, the thesis also contains feedback from users (in this case fellow students) and a comparison with two similar applications. The feedback was used to make changes in the user interface of the application.

The source code is published with the MIT license. The source code is available in GitHub at https://github.com/Erikdzo/postgres-visual-query-app and the web application itself is located at http://apex.ttu.ee/postgres-query.

The thesis is in Estonian and contains 46 pages of text, 10 chapters, 36 figures, 1 tables.

Lühendite ja mõistete sõnastik

AJAX	*Asynchronous JavaScript And XML, asünkroone JavaScript ja XML*
API	*Application Programming Interface*, rakendusliides
CSV	*Comma Separated Values*, komaeraldusega väärtused
DOM	*DOM (Document Object Model),* dokumendiobjektide mudel
HTTP	*HyperText Transfer Protocol*, hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation*
MIT	*Massachusetts Institute of Technology,* Massachusettsi Tehnoloogiainstituut
päring	*Query* Andmebaasikeele lause andmebaasist andmete otsimiseks. SQLis on selleks SELECT lause.
SQL	*Structured Query Language*, struktuurpäringukeel
URL	*Uniform Resource Locator,* internetiaadress

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval, kui Internet on kogu meie eluga tihedalt läbipõimunud, ootaks muuhulgas, et andmebaasipäringuid saaks teha veebipõhiselt. Kuna andmeid kogutakse andmebaasisüsteemide abil loodud andmebaasidesse aina rohkem, siis tekib aina enam ka ülesandeid, mida nende andmete kasutamine aitaks lahendada. Selleks, et andmete kasutamine käiks võimalikult mugavalt ning oleks jõukohane võimalikult laiale kasutajate ringile, peaks tarkvara võimaldama koostada päringuid ilma SQLi (või mõne muu andmebaasikeele) lähtekoodi kirjutamiseta. Paraku näiteks PostgreSQL andmebaasisüsteemi populaarne veebipõhine administreerimiskeskkond phpPgAdmin (5.6) [1] seda võimalust ei paku. Antud töö kontekstis pakub huvi PostgreSQL andmebaasisüsteem [2], sest see on populaarne, tasuta, väga võimekas ning seega ka perspektiivne, avatud lähtekoodiga ja kõigele lisaks on mul sellega olnud praktiline kokkupuude. Soovin, et PostgreSQLil oleks visuaalne ja veebipõhine andmebaasikeele lausete koostamise keskkond. Laiema kasutatavuse huvides ei peaks see olema phpPgAdmin alammoodul, vaid eraldiseisev programm. Kuna PostgreSQL on avatud lähtekoodiga, siis peaks juba põhimõtte pärast olema ka see vahend avatud lähtekoodiga. Pealegi suurendab avatud lähtekood võimalust, et tulemust arendatakse edasi ja hakatakse laiemalt kasutama.

Lõputöö eesmärgiks on luua avatud lähtekoodiga veebipõhine rakendus, mis võimaldab kasutajal koostada visuaalselt PostgreSQL andmebaasi päringuid (SELECT lauseid). Kasutaja saab veebirakenduses ühendada ennast PostgreSQL andmebaasiga. Seejärel kuvatakse kõik tabelid (baastabelid, vaated, välised tabelid, hetktõmmised, päritud tabelid), millele kasutajal on õigused ning seejärel saab kasutaja koostada läbi kasutajaliidese päringu, käivitada selle ja näha tulemust. Täpsemalt peab rakendus lõppkujul võimaldama kasutajal:

- otsida tabeleid nime ja liigi järgi,
- näha (välisvõtmetest tulenevaid) tabelite vahelisi seoseid,
- päringusse veerge ja tabeleid lisada ning neid sealt eemaldada,

- päringu kontekstis veerge ja tabeleid ümbernimetada,
- päringu tulemuses veergude järjekorda muuta,
- päringu kontekstis tabeleid ühendada,
- lisada piiravaid tingimusi päringu tulemusse kuuluvatele ridadele,
- koostada koondandmete leidmise päringuid (st ridu grupeerida ja arvutada iga grupi kohta kokkuvõttefunktsiooni kasutades väärtus),
- kirjeldada päringu tulemuste sorteerimiseeskiri,
- näha koostatud päringut SQL kujul,
- käivitada koostatud päringut,
- laadida alla päringu SQL tekst ja ka selle tulemus.

Rakendus annab kasutajale kiire ja loodetavasti mugava viisi kuidas SQL päringuid koostada.

Lõputöö tegemise metoodikaks on disainiteadus [3], mille tulemuseks on tehniline artefakt ehk PostgreSQL päringute koostamise veebirakendus. Tulemuse sobivuses veendumiseks lasen kaasüliõpilastel selle abil päringute koostamise ülesandeid lahendada, kogun kokku nende tagasiside ja teen sellele tuginedes vahendisse parandusi.

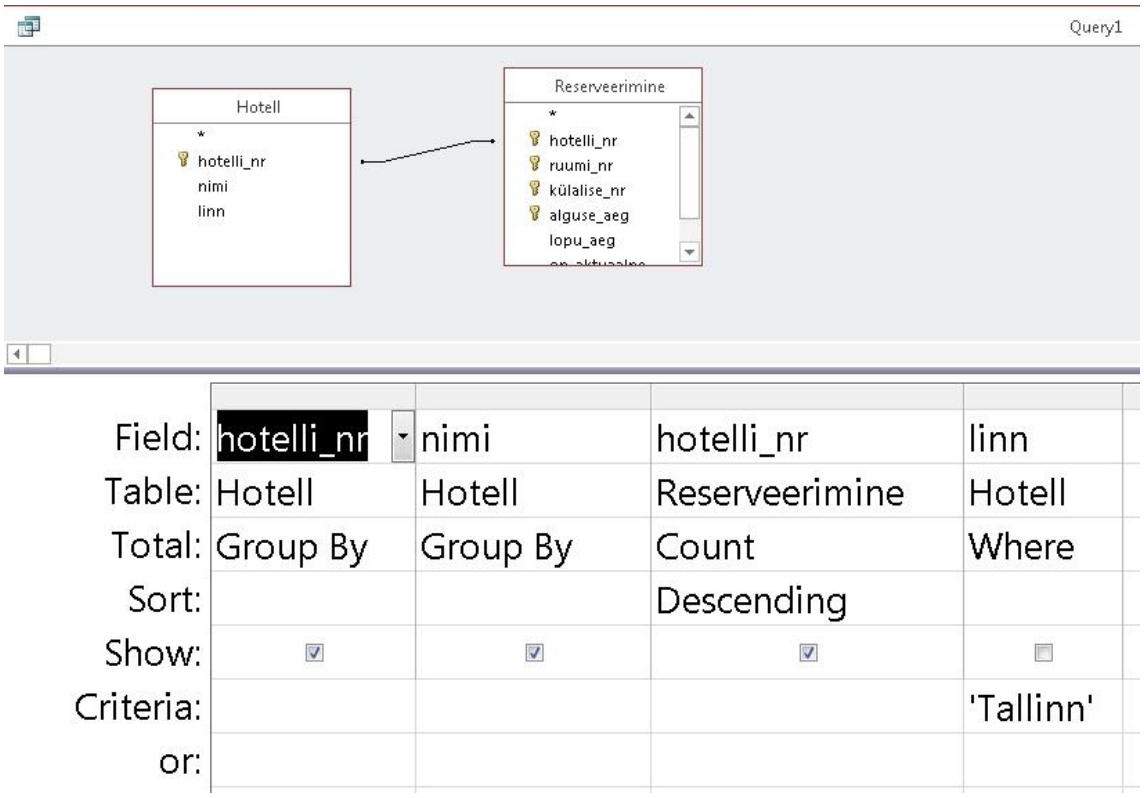
Projekti lähtekood on kättesaadav GitHubi hoidlas: https://github.com/Erikdzo/postgres-visual-query-app.

Käesolev töö koosneb kümnest osast. Teise peatükis tutvustatakse projekti tausta ja juba olemasolevat tarkvara. Kolmandas peatükis kirjeldatakse projekti arendusprotsessi. Neljandas peatükis esitatakse nõuded loodavale tarkvarale. Viiendas peatükis kirjeldatakse tagarakendust. Kuuendas peatükis kirjeldatakse kasutajaliidest. Seitsmendas peatükis esitletakse rakenduse arhitektuuri. Kaheksandas peatükis saadakse tagasisidet kasutajatelt. Üheksandas peatükis võrreldakse loodud programmi teiste programmidega. Kümnendas peatükis esitatakse käesoleva töö kokkuvõte.

2 Taust ja olemasolev tarkvara

„Lähtekood on programmeerimiskeeles kirjutatud tekst, mis kirjeldab arvutile antavaid käske.“ [4]. Lähtekood võib kasutada tekstilist märgisüsteemi või graafilist märgisüsteemi. Graafilise ehk visuaalse esituse saab alati tõlkida tekstiks ning vastupidi. Üheks graafilist märgisüsteemi pakkuva programmeerimiskeele näiteks on *„Query by Example“*. Tegemist on andmebaasikeelega e valdkonnapõhise programmeerimiskeelega, mis on mõeldud relatsiooniliste/SQL andmebaaside kasutamiseks.

Seda keelt kirjeldati esmakordselt M.M. Zloofi 1975. aasta artiklis *„Query by Example“* [5] ning kirjeldust täpsustati ja arendati edasi 1977. aasta artiklis *„Query by example: a database language“* [6]. Selles keeles saab koostada nii andmete otsimise, muutmise kui ka andmestruktuuride haldamise lauseid. Tänapäeval on selle keele põhimõtetest lähtuvalt lisatud paljudesse SQL-andmebaaside haldamise/administreerimise vahenditesse võimalus koostada andmete otsimise (ja mõni kord ka manipuleerimise) lauseid graafilise kasutajaliidese abil. Selle näiteks on MS Access töölaua andmebaasisüsteemi osaks olev Query Designer (Joonis 1, Joonis 2). Enamasti võimaldavad need vahendid soovi korral tõlkida koostatud visuaalse keele lause SQL keelde. Osad süsteemid (näiteks MS Access töölaua andmebaasisüsteemi osaks olev Query Designer) võimaldavad ka vastupidist tõlget tekstilisest SQL lausest visuaalses keeles lauseks.

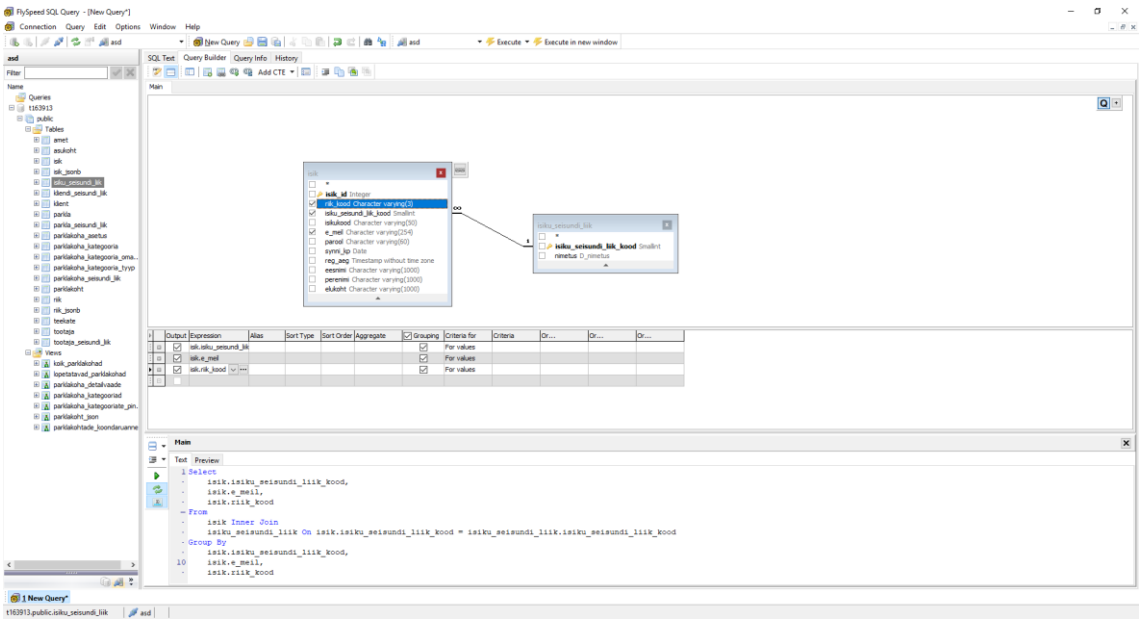


Joonis 1. MS Access andmebaasisüsteemis Query Designer vahendi abil koostatud päring.

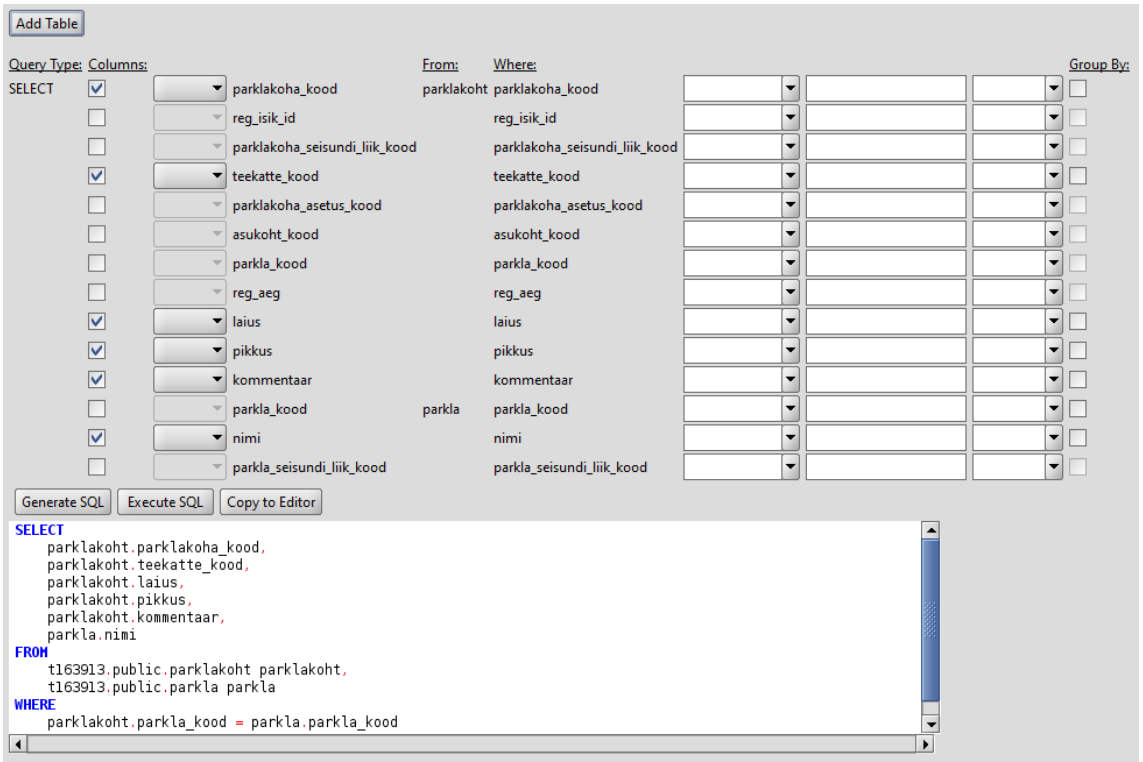
Joonis 2. Query Designeri abil koostatud päringu SQLi tõlkimise tulemus.

2.1 Olemasolev tarkvara päringute visuaalseks koostamiseks

Visuaalselt päringute koostamise rakendusi leidub palju kuid enamus neist on tasulised ning eraldiseisvad kasutaja arvutisse installeeritavad rakendused nagu näiteks *DbSchema* (8.1) [7], *FlySpeed* (3.7.5.3) (Joonis 3. Rakenduse *FlySpeed* pärigute koostamise vaade) [8], *RazorSQL* (8.2.4) (Joonis 4) [9] ja *Aqua Data Studio* (19.5) [10].



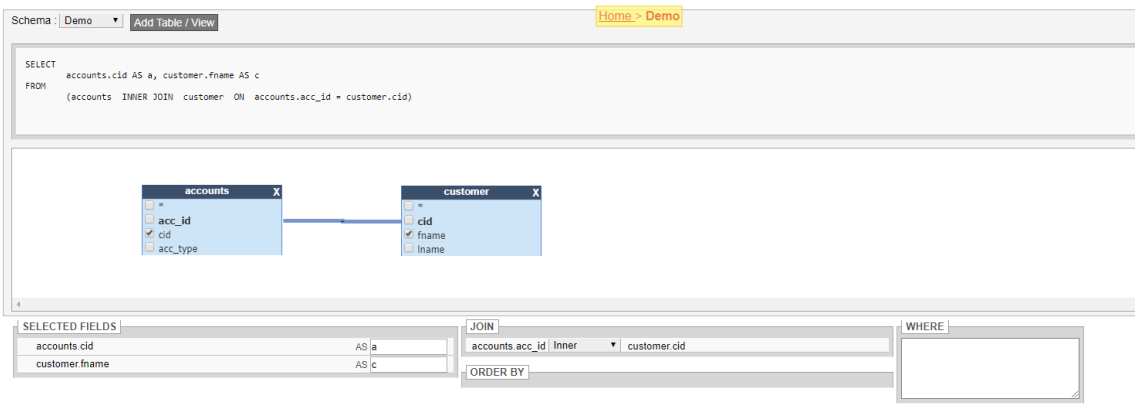
Joonis 3. Rakenduse *FlySpeed* päringute koostamise vaade



Joonis 4. Rakendus *RazorSQL* päringute loomise vaade

Veebipõhiseid rakendusi ei ole selles valdkonnas väga palju kuid ka need on tasulised või väga rangete piirangutega nagu näiteks *Skyvia Query Builder* [11] ja *DBHawk* [12]. Näiteks ei ole Skyvia Query Builderis võimalik kasutajal ise kirjeldada ühendamisoperatsioone.

Rakendused, mis on tasuta ja veebipõhised ei toeta PostgreSQL andmebaase. Näiteks *Web-based Visual Query Designer* (Joonis 5) [13] toetab ainult MySQL andmebaase ja seda pole uuendatud juba mitu aastat. PostgreSQLi toetavad lahendused on kõvasti piiratud funktsionaalsusega. Näiteks *Active Query Builder Web API* [14] vahendis on võimalik päringuid teha ainult ühe tabeli põhjal.



Joonis 5. Veebirakenduse *Web-based Visual Query Designer* päringute koostamise vaade

3 Arendusprotsess

Rakenduse loomiseks kasutatakse valikut agiilsete tarkvaraarenduse metoodikate parematest praktikatest/soovitustest. Projekti tegemise käigus väärtustati agiilse tarkvaraarenduse manifestis kirjeldatud põhimõtteid [15], mille kohaselt on *tähtsamad* (st esimene pool on tähtsam, kuid teine pool pole tähtsusetu):

- inimesed ja suhtlemine kui protsesse ja arendusvahendeid,
- töötav tarkvara kui põhjalik dokumentatsioon,
- koostöö kliendiga kui läbirääkimised lepingute üle,
- reageerimine muutunud olukorrale kui algse plaani järgimine.

Tarkvara funktsionaalsused (funktsionaalsed nõuded) pannakse kirja kasutuslugudena ning jagatakse iteratsioonide vahele ära [16]. Funktsionaalsete nõuete allikaks on lõputöö juhendaja ja ka autor, kes just eelnevalt oli läbinud kaks andmebaaside teemalist õppeainet.

Enne iteratsioonide vahele ära jagamist antakse igale funktsionaalsusele prioriteet ja raskustase. Peale prioriteetide ja raskusastmete jagamist sorteeritakse funktsionaalsused prioriteedi järgi ära ja jaotatakse iteratsioonide vahele ära. Funktsionaalsuste võrdseks jagamiseks peaks olema iga iteratsiooni funktsionaalsuste raskustasemete summa enam vähem võrdne.

Iga iteratsiooni lõppedes kohtutakse tellija rollis oleva juhendajaga, et talle iteratsiooni käigus loodud tarkvara ette näidata. Saadud tagasiside annab alust uuteks nõueteks, mida käsitletakse järgnevates iteratsioonides.

3.1 Veebirakenduse kontrollimine

Koodi kvaliteedis määramiseks kasutatakse ühikteste, et olla kindel rakenduse funktsionaalsuse korrektses toimimises. Eesmärgiks on saavutada 80% ühiktestide reakattuvus. Reakattuvus määrab ära kui suur osa programmi koodiridadest läbiti ühiktestide läbimise käigus.

Kasutajaliidese testimiseks luuakse tagasisidevorm ja stsenaariumid, mis antakse inimestele kasutajaliidese testimiseks. Peale tagasiside saamist tehakse kokkuvõte sellest ja viiakse rakendusse sisse muudatused tagasiside põhjal.

4 Analüüs

Peatükk kirjeldab veebirakenduse funktsionaalseid ja mittefunktsionaalseid nõudeid.

4.1 Funktsionaalsed nõuded

Rakenduse funktsionaalsed nõuded on koostatud kasutuslugude (*user story*) formaadis. Kasutuslugu võimaldab suhteliselt lihtsa lausena kirja panna rakenduse funktsionaalsuse lähtuvalt rakenduse kasutaja vaatenurgast. Kasutuslood järgivad tavaliselt kuju „<kasutajatüüp> tahan teha <tegevus>, et <põhjus>“ [17]. Leidsin, et antud juhul on funktsionaalseid nõudeid sellisel viisil mugavam esitada kui kasutades kasutusjuhtude mudelit (kasutusjuhtude diagrammid + kasutusjuhtude tekstilised kirjeldused).

Järgnevalt toon välja kasutuslood. Kasutuslood ei ole olulisuse järgi sorteeritud. Kasutuslood on grupeeritud nii, et seotud funktsionaalsusele viitavad kasutuslood oleksid nimekirjas lähestikku. Grupeerimise eesmärgiks on vältida sama nõude korduvat kirjapanekut ja leida puuduvaid nõudeid.

Ühendus andmebaasiga

- Kasutajana tahan luua andmebaasiga ühenduse, et saaksin hakata päringut koostama.
- Kasutajana tahan katkestada andmebaasiga ühenduse, et päringute koostamine ning käivitamine lõpetada.
- Kasutajana tahan võimalust ise määrata serverit, kus asub andmebaas ja päringute tegemiseks kasutatavat andmebaasi, et koostada just endale kõige olulisemaid päringuid.

Tabelite ja veergude vaatamine

- Kasutajana tahan näha andmebaasis olevaid tabeleid ja nende tabelite veerge, et saaksin hakata neid lisama oma päringusse.
- Kasutajana tahan otsida tabeleid andmebaasi tabelite seast nime ja skeemi järgi, et tabeleid kiiremini üles leida.
- Kasutajana tahan näha, mis tüüpi tabeliga on tegemist, et saaksin tabelite valimisel teha põhjendatud valiku.

- Kasutajana tahan näha andmebaasi tabelite vahelisi (välisvõtmetest tulenevaid) seoseid, et saada paremini aru, üle milliste veergude oleks vaja läbi viia tabelite ühendamist.
- Kasutajana soovin näha tabelite veergude andmetüüpe, et oskaksin koostada päringu tingimust ning omaksin paremat ettekujutust päringu tulemusel tagastatavatest andmetest.

Tabelite valimine

- Kasutajana tahan lisada päringusse tabeli, et selle alusel päring koostada.
- Kasutajana tahan eemaldada päringust tabeli, et seda enam päringus ei oleks.

Projektsioon (veergude valimine)

- Kasutajana tahan päringusse lisada veerge, et saaksin nendele vastavaid andmeid hiljem päringu tulemusest vaadata.
- Kasutajana tahan muuta päringus olevate veergude järjekorda, et tulemuses oleks veerud teisiti järjestatud ja seega ehk tulemus paremini arusaadav.
- Kasutajana tahan päringust eemaldada veerge, et neid ei kajastataks päringu tulemuses.
- Kasutajana tahan rakendada veerule reataseme funktsiooni, et teisendada päringu tulemuses olevaid andmeid.
- Kasutajana tahan eemaldada veerult sellele rakendatud reataseme funktsiooni, et näha andmeid algsel kujul.
- Kasutajana tahan lisada päringusse veeru nii, et selles olevaid andmeid tulemuses ei näidata, et saaksin seda siiski kasutada piiravas tingimuses või sorteerimiseeskirjas.

Ridade piiramine

- Kasutajana tahan lisada päringu veergudele piiranguid, et tulemuses oleks ainult soovitud tingimusele vastavad read.
- Kasutajana tahan muuta päringus olevaid piiranguid, et päringut parandada ja muuta päringu tulemuseks olevat ridade hulka.
- Kasutajana tahan eemaldata päringust piiranguid, et muuta päringu tulemuseks olevat ridade hulka.

Tabelite ühendamine

- Kasutajana tahan ühendada päringus tabeleid, et luua päringuid mitme tabeli põhjal.
- Kasutajana tahan vahetada päringus tabelite ühenduse tüüpi, et muuta päringu tulemust.
- Kasutajana tahan muuta tabeleid, mis on omavahel päringus kokku ühendatud, et muuta päringu tulemust.
- Kasutajana tahan eemaldada päringust tabelite ühendamise operatsioone, et muuta päringu tulemust.

Sorteerimine

- Kasutajana tahan lisada päringule sorteerimiseeskirja, et muuta ridade järjekorda tulemuses.
- Kasutajana tahan muuta päringu sorteerimiseeskirja, et muuta ridade järjekorda tulemuses.
- Kasutajana tahan eemaldada päringust sorteerimiseeskirja, et tulemuses ridu enam ei sorteeritaks.

Veergude ümbernimetamine

- Kasutajana tahan päringu veerge ümbernimetada (anda alias), et tulemuses oleksid veerud uue nimega.
- Kasutajana, tahan muuta veeru aliast, et tulemuses oleksid veerud uue nimega.
- Kasutajana, tahan veeru aliase kustutada, et tulemuses oleks algne veeru nimi.

Tabelite ümbernimetamine

- Kasutajana, tahan tabeleid ümbernimetada (anda alias), et muuta näiteks päringut ja selle tulemust loetavamaks.
- Kasutajana, tahan muuta tabeli aliast, et päringus oleks tabel teistmoodi nimetatud.
- Kasutajana tahan tabeli ümbernimetust aliast kustutada, et päringus kasutataks tabelile viitamiseks selle algset nimetust.

Koondandmed

- Kasutajana tahan päringu veerule lisada grupeerimise, et leida koondandmeid.

- Kasutajana tahan päringu veerult eemaldada grupeerimise, et muuta päringu tulemust.
- Kasutajana, tahan rakendada veerule kokkuvõttefunktsiooni, et muuta päringu tulemust.
- Kasutajana tahan vahetada veerule rakendatavat kokkuvõttefunktsiooni, et muuta päringu tulemust.
- Kasutajana, tahan eemaldada veerule rakendatud kokkuvõttefunktsiooni, et muuta päringu tulemust.

Korduvad read

- Kasutajana tahan lisada päringule korduvate ridade eemaldamise klausli, et vähendada tulemuses andmete liiasust.
- Kasutajana tahan eemaldada korduvate ridade eemaldamise klausli, et näha kõiki ridu, mida päring välja toob.

SQL koodi vaatamine

- Kasutajana tahan näha koostatud päringu SQL kuju, et kontrollida oma päringut.
- Kasutajana, tahan salvestada koostatud päringu SQL kuju eraldi faili, et seda hiljem kasutada.

Tulemuse vaatamine

- Kasutajana tahan käivitada koostatud päringut, et näha tulemust.
- Kasutajana soovin suure hulga päringu tulemuses olevate ridade puhul vaadata tulemust osade (lehekülgede) kaupa, et mitte ennast infoga ülekoormata.
- Kasutajana tahan peale koostatud päringu käivitamist näha päringu tulemust, et leida otsitav informatsioon.
- Kasutajana tahan päringu tulemust allalaadida CSV failina, et tulemusi hiljem kasutada.
- Kasutajana tahan kustutada kogu koostatud päringu, et alustada algusest.

4.2 Mittefunktsionaalsed nõuded

Tabel 1 esitatab ülevaate rakenduse mittefunktsionaalsetest nõuetest.

Tabel 1. Veebirakenduse mittefunktsionaalsed nõuded

Tüüp	Nõude kirjeldus
Kasutajaliides	Rakenduse kasutajaliides peab olema veebipõhine. Kasutajaliides peab olema selge ja arusaadav. Rakendus peab olema kasutatav ning kõik informatsioon nähtav nii suurte lauaarvutite monitoridega kui ka sülearvutide ekraanidega.
Arendusvahendid	Veebirakenduse tagarakendus peab olema Node.js põhine.
Keel	Kasutajaliides peab olema kasutatav inglise kui ka eesti keeles. Uute keelte lisamine peaks olema võimalikult lihtne.
Veebibrauserite tugi	Veebirakendus peab töötama enimkasutatavatel veebibrauseritel: Google Chrome, Internet Explorer ja Mozilla Firefox [18].
Laiendatavus	Rakenduse arhitektuur peab võimaldama tulevikus rakendusele lihtsalt funktsionaalsusi lisada.
Litsents	Loodav tarkvara peab olema avatud lähtekoodiga, mida kaitseb MIT (Massachusetts Institude of Technology) litsents. See litsents valiti, sest see on lühike vabavara litsents, mis ei piira lähtekoodi kasutamist, kui on tagatud, et teave autoriõiguse tingimuste kohta on lisatud uuesti publitseeritud tarkvarale [19]. Litsents ei piira lähtekoodi kasutajatel koodi kasutamist, muutmist ega müümist. Samuti eemaldab litsents vastutuse lähtekoodi autorilt selle eest kuidas teised lähtekoodi kasutavad.
Avaldamine	Lähtekood tuleb maailmale avaldada GitHubi hoidlas.

5 Tagarakendus

Käesolev peatükk kirjeldab tagarakenduse (*back-end*) loomiseks kasutatud vahendeid.

5.1 Tagarakenduse loomise vahendid

Tagarakendus töötab Node.js käitussüsteemis, kasutades raamistiku Express, mis võimaldab kiiresti luua võimekaid rakendusliideseid (API-sid). Andmebaasiga ühenduse loomiseks kasutatakse raamistiku node-postgres. Node-postgres raamistiku valimisel mängis suurt rolli, et node-postgres tegeleb ainult andmebaasisüsteemiga ühenduse loomisega ja andmebaasist päringute tegemisega, võrreldes mõne teise raamistikuga (näiteks Knex.js) , mis peale selle tegeleb ka päringute koostamisega.

5.2 Andmebaasisüsteem

Rakendus töötab ainult PostgreSQL andmebaasisüsteemiga. Rakenduse puhul testiti seda ühendustega PostgreSQL (11) abil loodud andmebaasidega.

5.3 Rakendusliides

Tagarakenduse rakendusliides jaguneb kahte ossa – andmebaasilst informatsiooni küsimine ja päringu käivitamine.

Andmebaasilst informatsiooni küsimiseks on rakendusliidesel kolm lõppsõlme:

- *postgres-query/api/database/tables*,
- *postgres-query/api/database/columns*,
- *postgres-query/api/database/constraints*.

Nende lõppsõlmede pihta HTTP POST päringut tehes tuleb sisendiks anda andmebaasisüsteemiga ühenduse loomiseks vajalikud andmed JSON objektina, mis koosneb järgnevatest väljadest:

- host – hosti nimi,
- port – pordi number,
- database – andmebaasi nimi,

- user – andmebaasi kasutajanimi,
- password – andmebaasi kasutaja parool.

postgres-query/api/database/tables proovib käivitada kaasa antud andmebaasi informatiooni alusel andmebaasi süsteemikataloogi põhjal SQL päringu (Lisa 1), mis leidab kõik andmebaasi tabelid.

Süsteemikataloog on andmebaasi alamosa, mis sisaldab infot andmebaasi struktuuri ja käitumise kohta. PostgreSQL realiseerib SQL standardis ettenähtud skeemi *information_schema*, mis sisaldab standardiseeritud struktuuriga vaateid süsteemikataloogi tabelite põhjal. Kuid kuna SQL standard (SQL:2016) ei kirjelda hetktõmmiseid e materialiseeritud vaateid, siis ei saa nende kohta andmeid standardiseeritud vaadetest, vaid selle asemel tuleb teha päring otse süsteemikataloogi tabelite põhjal ja leida tulemuse ühend ülejäänud tabelite kohta käivate andmetega.

Päringu tulemus tagastatakse list JSON objektidest, mis koosvevad kolmest väljast:

- tabeli skeem (table_schema),
- tabeli nimi (table_name),
- tabeli tüüp (table_type).

List on sorteeritud tabeli skeemi, tabeli tüübi ja tabeli nime järgi. Juhul kui ei ole võimalik andmebaasiga ühendust saada, siis tagastatakse vastuseks veateade.

postgres-query/api/database/columns proovib käivitada kaasa antud andmebaasi informatiooni alusel andmebaasi süsteemikataloogi põhjal SQL päringu (Lisa 2), mis leidab kõik andmebaasi tabelite veerud.

Päringu tagastab listi JSON objektidest, mis koosnevad viiest väljast:

- tabeli skeem (*table_schema*),
- tabeli nimi (*table_name*),
- veeru nimi (*column_name*),
- veeru järjekorranumber (*ordinal_position*),
- veeru andmetüüp (*column_type*)

List on sorteeritud tabeli skeemi, tabeli nime ja veeru järjekorranumbri järgi. Juhul kui ei ole võimalik andmebaasiga ühendust saada, siis tagastatakse vastuseks veateade.

postgres-query/api/database/constraints proovib käivitada kaasa antud andmebaasi informatiooni alusel andmebaasi süsteemikataloogi põhjal SQL päringu (Lisa 3), mis leiab kõik primaarvõtme, unikaalsuse ja välisvõtme kitsendused.

Päringu tulemus tagastatakse JSON objektide listina, mis koosnevad kaheksast veerust:

- piirangu nimi (*constraint_name*),
- piirangu tüüp (*constraint_type*),
- tabeli skeem (*table_schema*),
- tabeli nimi (*table_name*),
- veeru nimi (*column_name*),
- välise tabeli skeem (*foreign_table_schema*),
- välise tabeli nimi (*foreign_table_name*),
- välise tabeli veeru nimi (*foreign_column_name*).

Juhul kui ei ole võimalik andmebaasiga ühendust saada tagastatakse vastuseks veateade.

Andmebaasis kasutaja päringu käivitamiseks on lõppsõlm *postgres-query/api /query/query*, mille pihta on kliendil võimalik teha HTTP POST päring. Sisendiks tuleb anda JSON objekt järgnevate väljadega:

- host – hosti nimi,
- port – pordi number,
- database – andmebaasi nimi,
- user – andmebaasi kasutajanimi,
- password – andmebaasi kasutaja parool,
- sql – andmebaasis käivitatav SQL päring.

Seejärel proovitakse käivitada päring andmebaasis. Kui päring on korrektse süntaksiga, siis tagastatakse kasutajale päringu tulemus JSON formaadis. Vea korral tagastatakse kasutajale veateade JSON objekt, mis sisaldab veasisu, veakoodi ja juhul kui on tegu SQL süntaksi veaga, siis ka numbri, mis viitab vea tekitanud kohale SQL lauses.

Lisa 1-Lisa 3 ülesannete lahendamiseks mõeldud päringute esimese versiooni koostasin iseseisvalt, kuid lõpuks kasutasin juhendajalt saadud päringuid, mis annavad infot täpsemalt ja suurema hulga andmebaasiobjektide kohta.

6 Veebiliides

Järgnev peatükk kirjeldab vahendeid, mille abil loodi veebirakenduse kasutajaliides. Rakendusele on võimalik ligi pääseda leheküljelt http://apex.ttu.ee/postgres-query.

6.1 Veebirakenduse loomise vahendid

Rakenduse kasutajaliides on ehitatud raamistiku React abil. React on komponentidel põhinev deklaratiivne raamistik, mille tõttu ei pea kasutajaliideste loomisel ja sündmuste loomisel dokumendiobjekti mudelit ehk DOM-i pärima. See tähendab, et React ütleb brauserile täpselt mida brauser tegema peab, mitte ei küsi brauserilt vajaminevaid elemente [20].

Reacti kaudu on võimalik igale komponendile anda olek ja funktsionaalsus kasutades JavaScript-i. Kuna React on deklaratiivne, siis oskab see komponendi oleku muutumisel efektiivselt komponente uuendada ja esitada.

Reactis kasutajaliidese struktuuri loomisel pannakse komponendid hierarhiasse ehk igal komponendil on üks vanemkomponent (välja arvatud baaskomponent, millel on null vanemkomponenti) ning null või rohkem alamkomponenti. Kuna Reactis on andmete liiklus ühesuunaline, siis liiguvad andmed vanemkomponendidelt alamkomponentidele, mis vähendab rakenduses tekkivate vigade arvu, kergendab rakendusest vigade leidmist ning muudab rakenduse efektiivsemaks [21]. Kuid selle tõttu muutub rakenduse terve oleku haldamine raskeks.

Rakenduse erinevate vaadete haldamiseks kasutatakse raamistiku React Router (versioon 4.3.1) [22], mille abil on võimalik kirjeldada, milliseid vaateid näidatakse erinevate URL radade puhul.

Komponentide kujundamiseks kasutatakse Bootstrapi (versioon 4.3.1) [23] ning selle põhinevat raamistiku Reactstrap (7.1.0) [24], mis muudab Bootstrapi komponendid Reacti komponentideks ning aitab seeläbi koodi kirjutamise ja lugemise mugavamaks teha. Projektis kasutatakse Bootstrapi, sest võimaldab luua kasutaja seadme

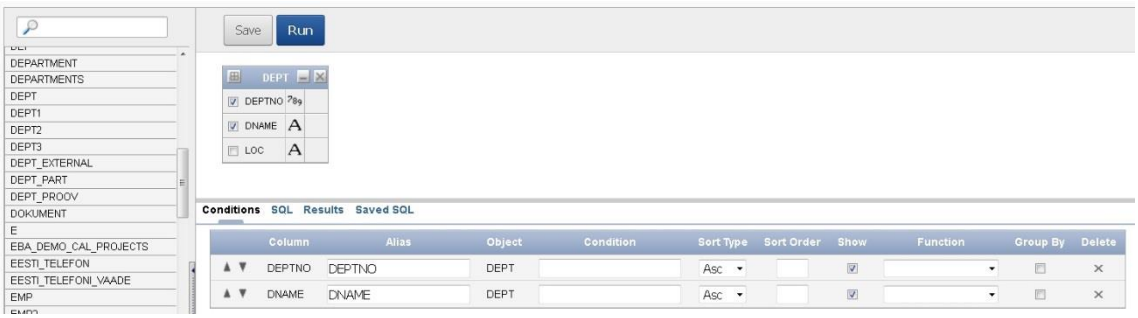
ekraanisuurusele vastavalt reageerivat disaini, mis lihtsustab kasutajaliidese loomist. Samuti dokumenteerib Bootstrap projekti kasutajaliidese disaini koodi. Reageeriva disaini tugi on kasutatud selleks, et veebibrauseri akna suurust muutes ei läheks kasutajaliides katki. Samas ei kasutata seda tuge et muuta rakendus kasutatavaks nutiseadmete jaoks. Autori hinnangul tegelevad enamus inimestest andmebaasidest andmete otsimisega laua- või sülearvutit kasutades.

Kasutajaliideses on kasutusel ka välised komponendid React Custom Scrollbars (versioon 4.2.1) [25] kohandatud kerimisribade jaoks, React Table (versioon 6.10) [26] andmete tabelis näitamiseks ja React FontAwesome (versioon 0.1.4) [27] erinevate ikoonide näitamiseks rakenduses. Väliseid komponente kasutatakse, et saaks kulutada rohkem aega rakenduse funktsionaalsuse arendamisele ja mitte kulutada aega erinevate utiliit-komponentide arendamisele. Lisaks tuleb väliste komponentidega kaasa dokumentatsioon, mis omakorda kirjeldab programmi koodi.

Rakenduses kasutatakse ka JavaScripti teeki Lodash (versioon 4.17.11) [28], mis pakub arendamiseks utiliit-funktsioone mugavamaks arendamiseks ja koodi loetavuse tõstmiseks.

6.2 Veebiliidese funktsionaalsus ning väljanägemine

Järgnevalt kirjeldatakse vaadete kaupa veebirakenduse funktsionaalsust. Kasutajaliidese ülesehitusel on võetud eeskuju Oracle veebirakenduste kiirprogrammeerimise vahendis Oracle Application Express (APEX) olevast alamprogrammist Query Builder (Joonis 6).



Joonis 6. Oracle APEX Query Builder.

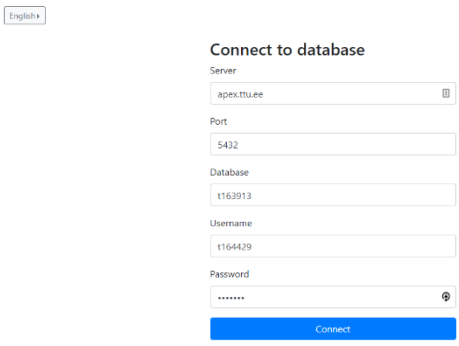
Akna vasakus servas on tabelite nimekiri koos võimalusega tabeleid otsida ja valida. Töölaua osa on jaotatud kaheks – ülemises osas näeb päringusse valitud tabeleid ning saab valida nende veerge ning alumises osas on ruudustik, milles saab kasutada valitud

veerge päringu koostamiseks. Selles osas on ülesehitus ka sarnane MS Accessi Query Designer vahendile (Joonis 1).

Kasutajaliideses kasutatakse punast värvi kasutajale teatamiseks, et mingi nuppu vajutamisel toimub tegevus, mille tagajärjel katkestatakse ühendus andmebaasiga või kustutatakse ära mingi informatsioon. Samuti kasutatakse punast värvi veateadetes. Sinise värviga tõstetakse esile tähtsaid funktsionaalsuseid nagu päringu käivitamine ning tuuakse esile lüliti elementide olek. Rohelise värviga kuvatakse veebirakenduses kasutaja õnnestunud tegevuste tulemusi – näiteks veeru päringusse lisamisel muutub veerg roheliseks, et kasutajale teaks, millised veerud on juba päringusse lisatud ilma, et peaks veeru nime veergude nimekirjast otsima.

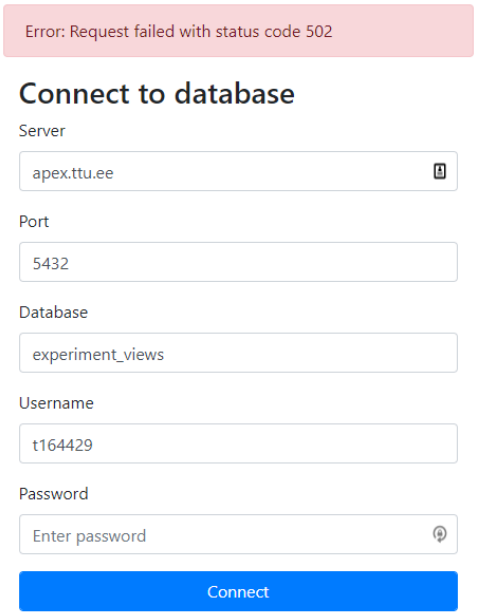
6.2.1 Sisselogimise vaade

Sisselogimise vaade (Joonis 7) on ühtlasi ka veebirakenduse avaleht. Siin kuvatakse kasutajale ankeet, kus kasutajal on võimalik sisestada oma andmebaasi serveri aadress, serveri port, andmebaasi nimi, andmebaasi kasutajanimi ja parool. Igale väljale on lisatud ka lühike seletus selle välja kohta, et kasutaja saaks peale vaadates aru, mis funktsionaalsust need väljad täidavad [29]. Kui tarkvara on mingil serveril üles seatud, siis on selle kaudu võimalik teha päringuid mistahes serveris olevate andmebaaside suhtes (eeldusel muidugi, et kasutajal on selleks õigus ja teine server lubab selliselt aadressilt tulevaid sisselogimise katseid).



Joonis 7. Sisselogimise vaade.

Kõik ankeedi väljad on kohustuslikud. Ankeedi lõpus on nupp, mille vajutamisel proovib veebirakendus saada ühendust andmebaasiga (vt jaotis 5.3). Juhul kui andmebaasiga ühendust ei saadud kuvatakse ankeedi kohale teade selle ebaõnnestumisest (Joonis 8). Kui kõik kohustuslikud väljad ei ole täidetud, siis kuvatakse välja juurde selle kohta teade.



Joonis 8. Veateade sisselogimise ebaõnnestumisel.

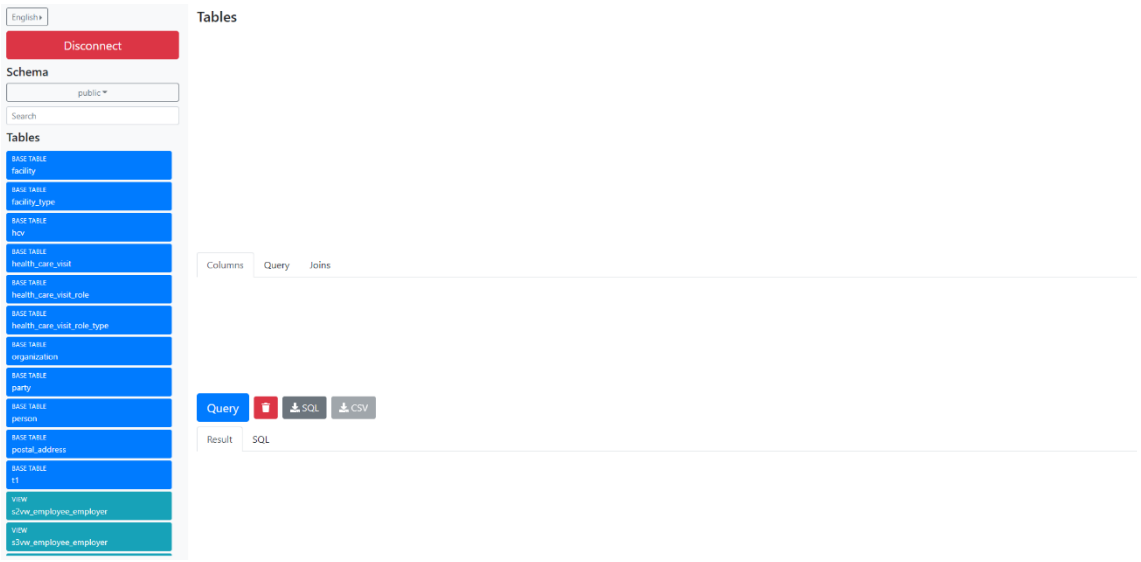
Veebilehe üleval on veel ka nupp, millele vajutades avanevast rippmenüüst on võimalik valida veebirakenduse keelt. Praegusel hetkel toetab rakendust kahte keelt: inglise ja eesti. Veebirakenduses kasutatakse rippmenüüsid, et hoida kokku ruumi, mida läheks vaja kõikide valikute korraga kuvamiseks [30].

Andmebaasiga ühendumise õnnestumisel küsitakse andmebaasisüsteemi käest andmebaasi struktuuri ehk tabeleid, veerge ja kitsendusi (vt jaotis 5.3). Seejärel liigub kasutaja edasi päringu koostamise vaatesse ning saab hakkata oma valitud andmebaasi põhjal päringut koostama.

6.2.2 Päringu koostamise vaade

Selles vaates toimub päringu koostamine. Vaade on jaotatud kaheks suuremaks osaks (Joonis 9). Vaate vasakus servas on külgriba. Seal saab valida andmebaasi skeemi, valida päringusse tabeleid, valida rakenduse keelt ning väljuda andmebaasist. Ülejäänud

osa aknast on päringu koostamise ala, kus kasutaja saab teha erinevaid tegevusi päringu loomiseks ning päringut käivitada.



Joonis 9. Päringu koostamise vaade.

Külgriba on alati kasutaja veebiakna kõrgune ja muudab veebiakna suurust muutes oma kõrgust.

Külgriba kõige tähtsam osa ehk andmebaasi tabelite kuvaja võimaldab kasutajal lisada oma päringusse tabeleid (Joonis 10). Tabelid on kuvatud nuppude listina üksteise alla ja iga tabeli kohta andakse kasutajale teada, millist tüüpi tabeliga on tegu ja tabeli nimi. Juhul kui tabeli nimi on liiga pikk ja seda enam ei ole võimalik näha, siis on kasutajal võimalik hiir liigutada nuppu peale, mille tulemusena kuvatakse tabeli nupu kõrvale tabeli terve nimi. Igale tabeli tüübile on määratud vastav värv, et kasutajal oleks kergem tabelitel vahet teha. Värvid on järgmised:

- Baastabel – sinine,
- Vaade – helesinine,
- Hetktõmmis e materialiseeritud vaade – helehall,
- Väline tabel – tumehall.

Tabeli nupule vajutades muutub nupu värv roheliseks ja see lisatakse päringusse. Uuesti nupule vajutades tabel eemaldatakse päringust ja nupu värv muutub algseks. Juhul kui tabelite nimekiri on liiga pikk ja ei mahu enam ekraanile ära, siis tekib nimekirja juurde kerimisriba, millega on võimalik nimekirjas edasi ja tagasi liikuda.

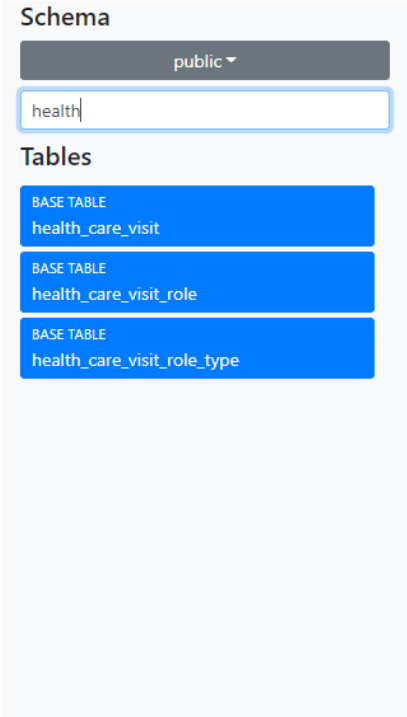
Joonis 10. Andmebaasi tabelite kuvaja.

Tabelite nimekirjas näidatakse korraga tabeleid ühest skeemist. Vaikimisi näidatakse tabeleid PostgreSQL poolt automaatselt loodavast skeemist *public*. Tabelite nimekirja kohal on nupp, mille kaudu saab kasutaja vahetada skeemi ja selle tulemusel muutub ka näidatavate tabelite nimekiri. Nupp näitab kasutajale ka hetkel valitud skeemi. Andmebaasi tabelite leidmiseks otsib tarkvara üles kõik andmebaasi skeemid ja kui nende seas on skeem nimega *public*, siis määrab selle valituks. Kui andmebaasis sellist skeemi ei ole, siis võetakse esimene skeem tähestikulises järjekorras.

Skeemi valiku ja tabelite nimekirja vahel on väli, mille kaudu on kasutajal võimalik otsida tabelite nimekirjast tabeleid nime või tüübi järgi. Ainult nime järgi otsimiseks peab kasutaja sisestama tabeli terve nime või ühe osa sellest (Joonis 11). Kui kasutaja soovib otsida konkreetse nimega tabelit, siis peab ta välja lõppu lisama tühiku. Näiteks kui kasutaja sisestab „hotell“, siis leitakse kõik tabelid, mis sisaldavad sõne „hotell“. Kui aga kasutaja sisestab „hotell “, siis otsitakse tabeleid, mille nimi on „hotell“.

Kui kasutaja soovib otsida tabeli tüübi järgi, siis peab ta kasutama märki # (trellid) ning sellele järgi lisama tabeli tüübi ingliskeelse nime (tabeli tüübi nimes tuleb asendada tühik alakriipsuga ehk nime BASE TABLE asemel tuleb kirjutada BASE_TABLE). Näiteks kui kasutaja soovib näha tabelite nimekirjas ainult andmebaasi vaateid, siis peaks ta sisestama „#VIEW“.

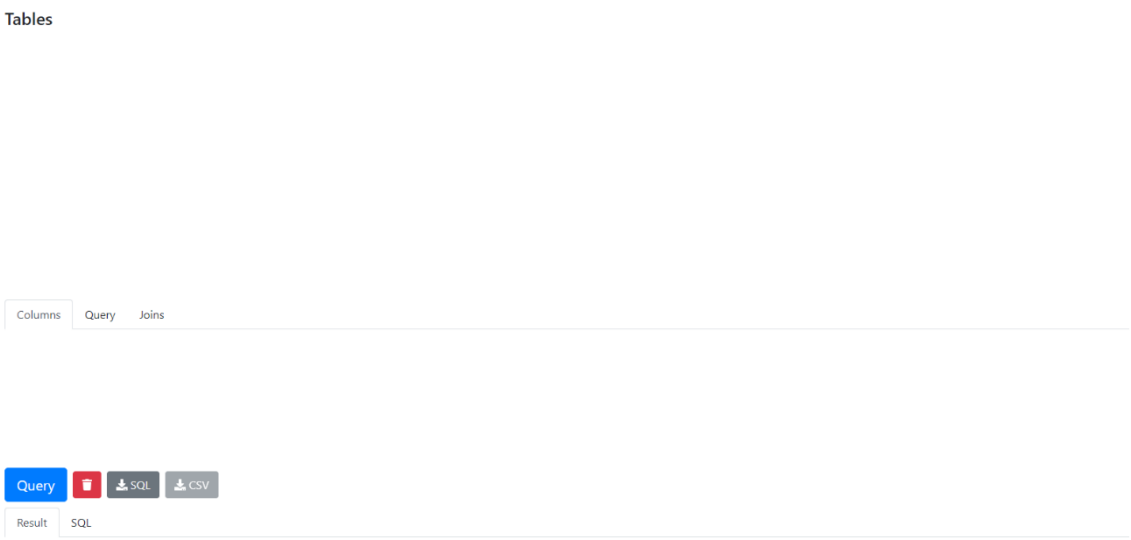
Otsing toetab ka tabeli tüübi ja tabeli nime järgi koos otsimist. Selle jaoks tuleb kirjutada tühikuga eraldatult tabeli tüübi nimi ja tabeli nimi nagu näiteks „#BASE_TABLE hotell“.



Joonis 11. Tabeli otsimine.

Külgribal on andmebaasist väljumise nupp, millele vajutades suunatakse kasutaja tagasi avavaatesse ja kustutatakse kogu informatsioon andmebaasi ja koostatud päringu kohta. Samuti on külgribal nupp rakenduse kasutajaliidese keele vahetamiseks.

Ülejäänud osa vaatest on ala kus toimub päringu koostamine (Joonis 12). Ala koosneb kolmest suuremast osast.



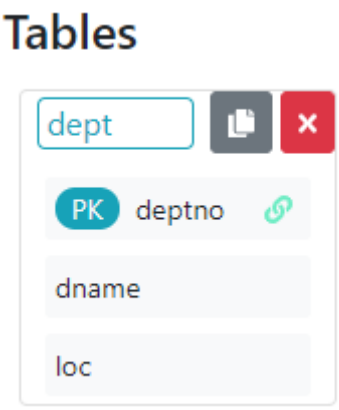
Joonis 12. Päringu koostamise ala.

Esimene osas kuvatakse kasutajale külgribalt valitud tabelid. Iga tabel kuvatakse eraldi tabeli komponendina (graafiliselt eraldi kast).

Tabeli komponent koosneb päisest ja sisust (Joonis 16). Päises näeb tabeli nime ja aliast. Hoides hiirt tabeli nime peal kuvatakse kasutajale tabeli skeemi nimi ja tabeli nimele klõpsates avaneb aken kust on võimalik kasutajal lisada, muuta või kustudata tabeli aliast. Samuti on võimalik päisest valida, et tabel lisatakse päringusse mitmekordselt. Mitmekordsel lisamisel määratakse tabeli uuele esinemisele automaatselt alias (X_1, X_2, … X_n – kus X on tabeli nimi), mida saab hiljem muuta (Joonis 14).

Tabeli sisu osas kuvatakse nimekirjana tabelis olevad veerud. Veerud kuvatakse samas järjekorras nagu see on määratud tabeli loomisel (SQLis on tabeli veerud järjestatud ja igal veerul on järjekorranumber). Iga veeru kohta kuvatakse veeru nimi. Hoides kursorit veeru kohal kuvatakse selle veeru andmetüüp. Kui veerg osaleb välisvõtmes, siis kuvatakse selle juures nupp, millele vajutades avaneb aken, kus näeb infot välisvõtme kaudu viidatud tabeli ja veergude kohta. Juhul kui veerg kuulub primaarvõtmesse, siis kuvatakse veeru nime ette lühend „PK“ (nagu *Primary Key*). Veerule vajutades lisatakse see veerg päringusse ning kasutajale andakse lisamisest teada veeru värvi muutumisega roheliseks. Uuesti veerule vajutades eemaldatakse veerg päringust.

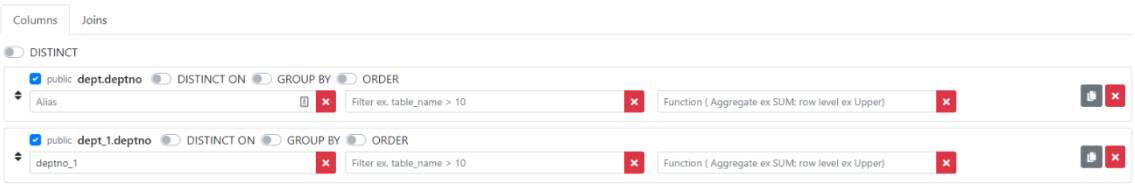
Tabeli kuvamiseks kasutatakse Bootstrapi elementi kaart. Sellise esituse abil on kasutajal lihtne aru saada tabeli struktuurist ehk millised veerud on selles tabelis. Kuna iga tabeli kohta on eraldi kaart, siis aitab see tabeleid visuaalselt üksteisest eristada [31].



Joonis 13. Tabeli komponent.

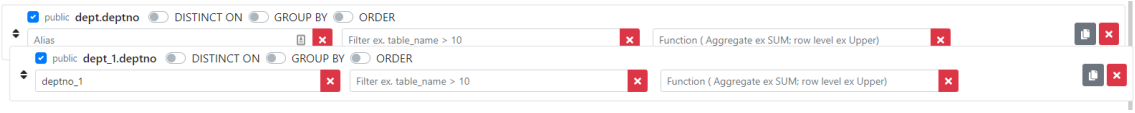
Joonis 14. Tabeli lisamine päringusse mitmekordselt.

Teine osa on jaotatud kaheks alammenüüks (Joonis 15). Alammenüüdeks jaotamist kasutatakse sellepärast, et kasutaja saaks teha kergesti vahet, mida talle hetkel ekraanil kuvatakse ning see annab asutajale hea ülevate, kus kohast ta endale vajalikud funktsionaalsused leiab [32]. Selle osas on kasutajal võimalik muuta päringusse lisatud veergude järjekorda ning teostada ridade piiramist mingi tingimuse alusel, veergude ümbernimetamist, tabelite ühendamist, grupeerimist, sorteerimist ja korduvate ridade eemaldamist. Samuti saab valida, et sama veerg lisatakse päringusse mitmekordselt. Mitmekordsel lisamisel määratakse veeru uuele esinemisele automaatselt alias (X_1, X_2, … X_n – kus X on veeru nimi), mida saab hiljem muuta. Igale väljale on lisatud ka lühike seletus selle välja kohta, et kasutaja saaks peale vaadates aru, mis funktsionaalsust need väljad täidavad [29].



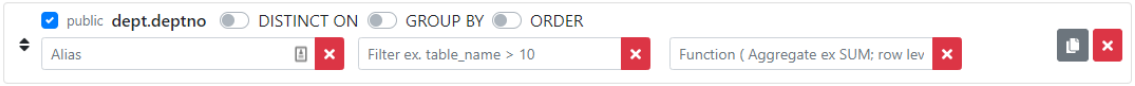
Joonis 15. Veeru lisamine päringusse mitmekordselt.

Päringusse lisatud veerud on kuvatud vertikaalses nimekirjas ja kasutajal on võimalik veergude järjekorda veergude komponente lohistades muuta (Joonis 16). Järjekorra muutmisel järgiti kasutajaliidese ülesehituse mustrit *Drag and Drop* [33]. Niiviisi on kasutajal lihtne veergude järjekorda ümber tõsta ja kerge aru saada kuidas veergude ümber järjestamine töötab. Komponendi kõige alguses on ja kahe nuulega ikoon, mis viitab kasutajale, et veergude järjekorda saab ümber tõsta.



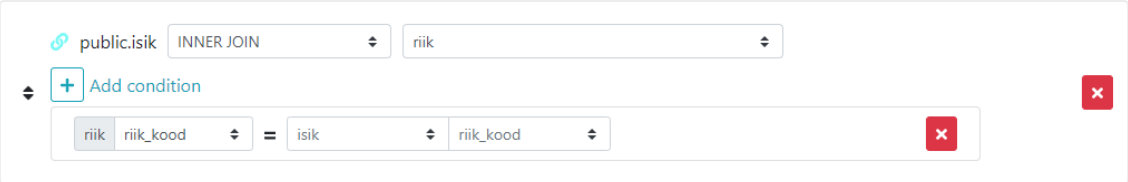
Joonis 16. Veergude järjekorra muutmine.

Igale veerule vastavas komponendis on kirjas veeru identifikaator kujul tabeli_nimi.veeru_nimi (Joonis 17). Veeru komponendis on kasutajal võimalik lisada, muuta ja kustutada veeru aliast, veeru alusel ridadele rakendatud piiranguid ja veerule rakendatud funktsiooni. See funktsioon võib olla kokkuvõttefnktsioon (näiteks Sum või Count) või reataseme funktsiooni (näiteks Upper). Kasutaja saab lisada, muuta ja kustutada sorteerimiseeskirja, grupeerimist ja korduvate ridade eemaldamist. Kasutaja saab ka määrata, et veerg on päringus kasutusel (näiteks ridu piiravas tingimuses), kuid päringu tulemuses ei näidata andmeid sellest veerust (selleks on sinine märkeruut). Samuti on kasutajal võimalik veerg päringust täielikult eemaldada.



Joonis 17. Veeru komponent.

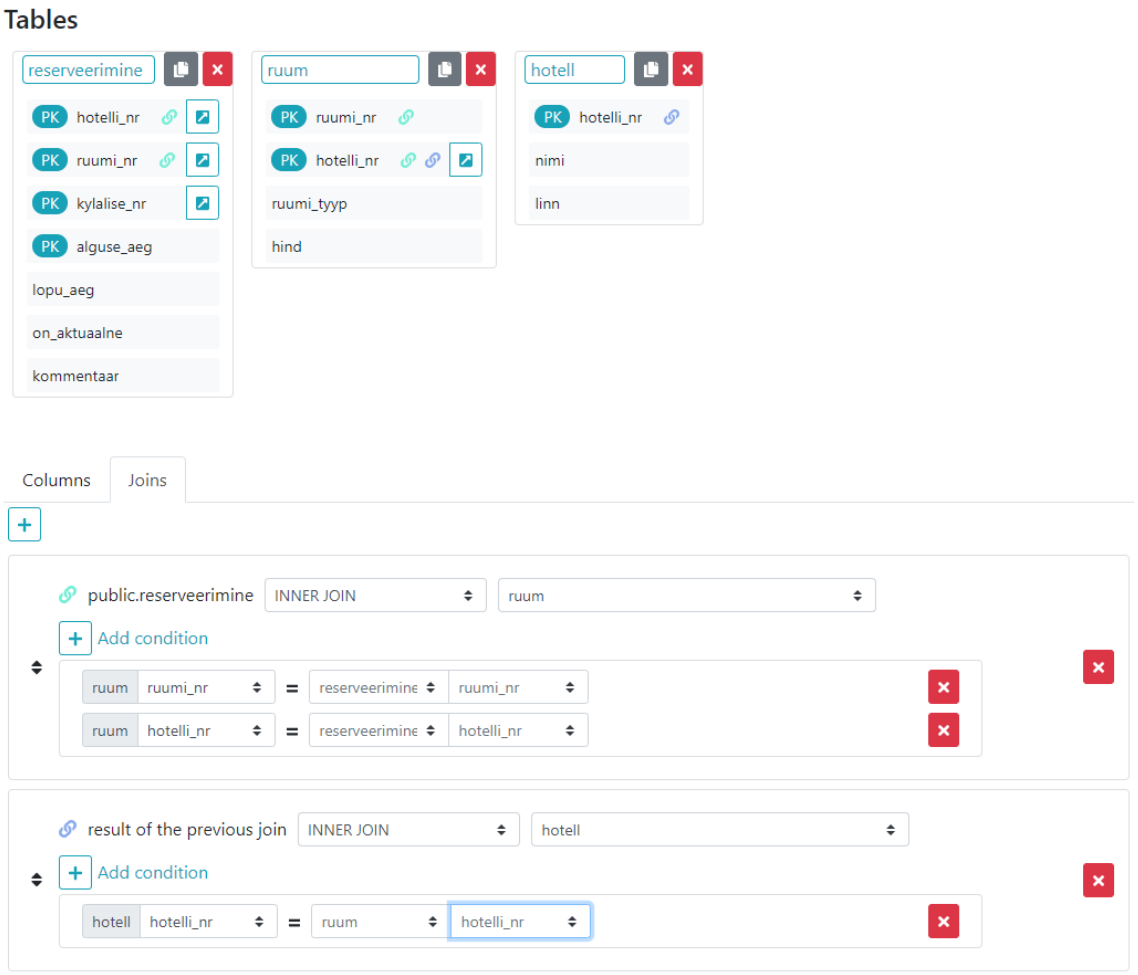
Päringus tabelite ühendamiseks on eraldi alammenüü (Joonis 18).



Joonis 18. Tabelite ühendamise komponent.

Uue ühendamisoperatsiooni defineerimiseks peab kasutaja vajutama alammenüüs olevat plussmärgiga nuppu. Seejärel tekib menüüse uus ühendamise komponent. Komponendis saab kasutaja valida ühendatava tabeli ning rippmenüüst ühendamisoperatsiooni tüübi. Seejärel saab kasutaja kirjeldada tingimused, mille põhjal toimub tabelites olevate andmete ühendamine. Tingimuste lisamiseks on igas ühendamise komponendis plussmärgiga nupp. Mitut tingimust on vaja, kui näiteks välisvõti ja sellele vastav primaarvõti on liitvõtmed (Joonis 19). Tingimuse komponendis saab kasutaja valida veerud ja tabelid, üle mille peab ühendamine toimuma. Iga ühendamise komponendi sees on ka unikaalse värviga ikoon, mis peale ühendatavate tabelite ja veergude valimist, kuvab tabeli komponenti veeru juurde samasuguse värviga ikooni. Ikoonid kuvatakse selle pärast, et kasutajal olek mugavam näha, millised tabelid on ühenduses ning ühendusi kiiremini üles leida (Joonis 19).

Kõik kasutaja poolt loodud ühendamisoperatsioonide kirjeldused kuvatakse vertikaalse nimekirjana ning kasutajal on võimalik järjekorda muuta lohistades vastavaid komponente. Nagu ka veergude ümberjärjestamisel võeti aluseks disainimuster *Drag and Drop* [33].



Joonis 19. Tabelite ühendamine mitme tingimusega ja kolme tabeli ühendamine.

Kasutajal on võimalik päringule lisada korraldus eemaldada tulemusest garanteeritult korduvad read. Selleks saab valida päringuga üldiselt seotud DISTINCT korralduse või konkreetsele veerule rakenduva DISTINCT ON korralduse. DISTINCT korralduse lisamine tervele päringule tõkestab DISTINCT ON kasutamise üksikute veergude korral. DISTINCT korraldus elimineerib korduvad read tulemusest. DISTINCT ON võimaldab moodustada ridade grupid ja jätta iga grupi kohta alles mingi ühe rea, kusjuures sorteerimise abil saab häälestada seda, millisel põhimõttel see rida valitakse [34].

Selle osa järel kuvatakse kasutajale reas neli nuppu (Joonis 20): päringu käivitamise, päringu kustutamise, päringu SQL tõlke allalaadimise ja päringu tulemuse allalaadimise nupp. Päringu SQL kuju laetakse alla .sql laiendiga failina ning päringu tulemused pannakse .csv tüüpi faili.

Joonis 20. Nupud päringutega seotud tegevuste jaoks.

Päringu koostamise akna kõige alumine osa koosneb kahest alammenüüst: päringu tulemuste vaatamise menüü ja koostatud päringu SQL kuju vaatamise menüü (Joonis 21).

Päringu koostamisel on kasutajal võimalik vaadata oma koostatava päringu tõlget SQLi. Iga kord kui kasutaja päringus midagi muudab, siis kuvatakse see koheselt selles alammenüüs. See muudab kasutajale kergemini arusaadavamaks kuidas tema tegevused mõjutavad genereeritavat SQL koodi ning annab kasutajale julgust rakendusega katsetada [35].

Result SQL

```
SELECT facility.facility_id AS id, facility.facility_name AS name, facility_type.name AS type
FROM public.facility
INNER JOIN facility_type ON (facility_type.facility_type_code = facility.facility_type_code)
```

Joonis 21. Päringu SQL kuju.

Peale päringu koostamist ja selle käivitamist on kasutajal võimalik vaadata ka päringu tulemust (Joonis 22). Tulemus kuvatakse kasutajale tabelis, mille päises on kirjas veergude nimed ja esimeses veerus rea number. Kasutaja saab tabelis lohistades muuta veergu laiust, et informatsiooni mugavamalt lugeda. Suurema ridade hulga puhul jagatakse tulemused lehekülgede vahel ära ning kasutaja saab valida kui palju ridu ühel leheküljel kuvatakse (maksimaalselt 100, minimaalselt 20). Tabeli loomiseks kasutatakse React-Table komponenti (versioon 6.10).

#	facility_id	facility_type_code	facility_name
1	98	ASC001	Yamia1072
2	99	HOS01	Zoomcast6220
3	100	HF09	Innotype82
4	101	MEDOFFICE03	Feednation790
5	102	ASC001	Avavee9398
6	103	MEDOFFICE03	Feedbug4130
7	104	CLI06	Jaloo1602
8	105	HOS01	Janyx8401
9	106	HOS01	Gabspot6711
10	107	CLI06	Yadel4181
11	108	HF09	Realbuzz7321
12	109	HF09	Zooxo4553
13	110	HOS01	Gabvine3750
14	111	HF09	Yakijo3223
15	112	ASC001	Gevee7621
16	113	HOS01	Photobug7091
17	114	ASC001	Agimba4500
18	115	MEDOFFICE03	Trunyx4479
19	116	HF09	Midel7642
20	117	MEDOFFICE03	Twitterlist9550

Previous Page 1 of 2500 20 rows Next

Joonis 22. SQL päringu tulemus.

7 Rakenduse arhitektuur

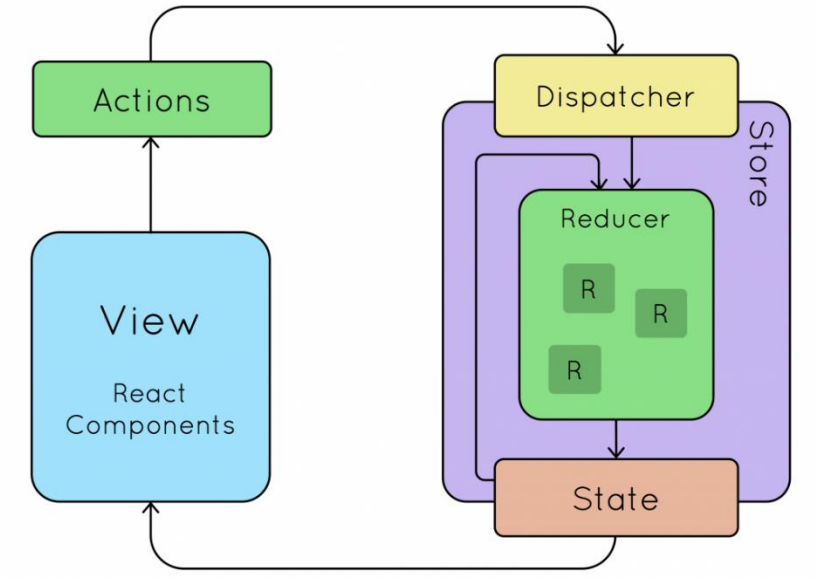
Selles peatükis antakse ülevaade rakenduse arhitektuurist.

7.1 Kasutajaliidese oleku haldamine

Veebirakenduse oleku haldamiseks kasutatakse raamistiku Redux (4.0.1) [36] ja selle Reacti implementatsiooni React Redux (6.0.1) [37], mis võimaldab terve rakenduse olekut hoida ühes staatilises objektis mida nimetatakse laoks (*store*). Lao oleku muutmiseks kasutatakse tegevusi (*action*), mille kaudu kirjeldatakse, mis rakenduses juhtub. See kuidas rakenduse olek tegevuste tagajärjel hakkab muutuma täpsustatakse ülekandjate (*reducer*) abil. Iga komponent ühendub vajaminevate rakenduse olekute ja tegevustega läbi lao, mis võimaldab igal komponendil teada rakenduse olekust täpselt nii palju kui on vajalik (Joonis 23). Seetõttu muutub rakenduses erinevate komponentide vahel informatsiooni jagamine kordades lihtsamaks ning rakenduse olekut muutvad funktsioonid koondatakse ühte kohta kokku.

Reduxil on kolm põhitõde.

- Terves rakenduses on ainult üks ladu.
- Oleku muutmiseks on vaja kasutada tegevusi.
- Oleku muutumiste kirjeldamiseks on ülekandefunktsioonid.



Joonis 23. React Reduxi elutsükkel [38].

Rakenduses on jaotatud ladu kolmeks osaks – andmebaasi, päringu ja seadete osa. Andmebaasi osa hoiab endas informatsiooni andmebaasi kõigi tabelite, veergude ja huvipakkuvate kitsenduste kohta. Päringu osas on kogu informatsioon koostatud päringu kohta, sh kõik päringus olevad tabelid, veerud ja tabelite ühendamised koos nendele määratud piirandutega. Seadete osa hoiab hetkel ainult informatsiooni rakenduses keele kohta. Ladu on niimoodi osadeks jaotatud, et koodis oleks kergem erinevatel oleku osadel vahet teha.

7.2 PostgreSQL lausete genereerimine

SQL lausete genereerimiseks kasutatakse JavaScripti teeki Squel.js [39] (versioon 5.12.2), mis sisaldab objektorjenteeritud rakendusliidest. See võimaldab lihtsalt ja mugavalt SQL lauseid koostada. Lausete genereerimiseks valiti see teek, sest see on kasutajasõbralik ning paindlik. Squel.js lubab ka muuta genereeritud SQL koodi vormistust, näiteks reavahetuste lisamine ja jutumärkide lisamist andmebaasi skeemide, tabelite ja veergude nimedele. Samuti toetab teek erinevaid andmebaasisüsteeme. Seetõttu on tulevikus lihtsam lisada rakendusse ka teiste andmebaasisüsteemide tuge.

Rakendus käivitab SQL lause koostamise peale igat päringu loomisega seotud tegevust – näiteks veeru või tabeli lisamine. Päringu koostamise tulemuseks on JSON objekt, mis sisaldab kogu informatsiooni päringu koha. Peale objekti loomist muudetakse see Squel.js abil SQL lauseks, mida on võimalik andmebaasi põhjal käivitada.

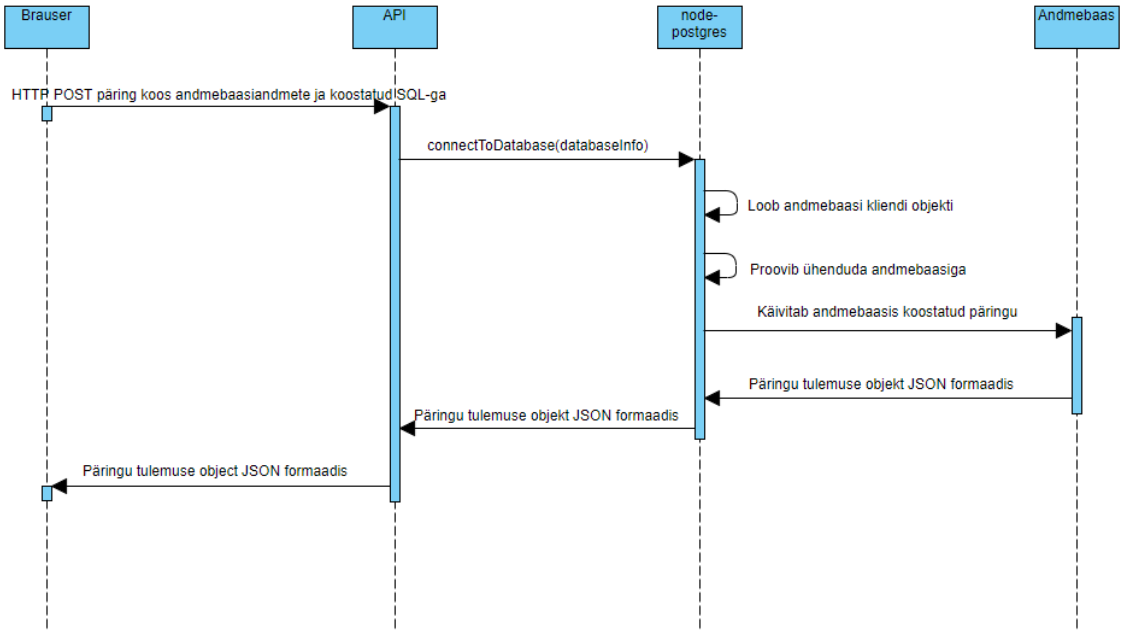
Päringu koostamisel võetakse ka arvesse ka identifikaatorite (skeemide, tabelite, veergude nimed) võimalik tõstutundlikust, et rakendust saaks ka kasutada andmebaaside puhul, milles on tõstutundlikud skeemi, tabeli või veeru nimed. SQL identifikaatorid jagunevad regulaarseteks ja piiritletuteks. Regulaarne identifikaator ei ole jutumärkides, piiritletud on. Piiritletud identifikaatori näited on "Isik" ja "Kliendi tellimus". Regulaarne identifikaator on tõstutundetu, kuid piiritletud on tõstutundlik.

7.3 Andmebaasist andmete küsimine rakendusliidese vahendusel

Rakenduse töötamiseks on vaja kuidagi kätte saada informatsioon andmebaasi kohta ning kuidagi käivitada koostatud päring andmebaasis .

Tagarakendusega suhtlemiseks kasutab rakendus brauseri jaoks mõeldud HTTP klienti axios (versioon 0.18.0) [40]. Axios on AJAX ja Promise tehnoloogiatel põhinev teek.

Päringu käivitamiseks tehakse JavaScriptis axiose abil päring tagarakendusele ning vastuseks tagastatakse päringu tulemus JSON formaadis (Joonis 24). Andmebaasiga ühendamiseks on kasutusel node-postgres teek, mis võimaldab luua andmebaasi kliente ja neid andmebaasiga ühendada. Mõne teise andmebaasisüsteemi toe lisamiseks peaks leidma leidma mõne teise teegi, mis võimaldaks ühendust luua ja see node-postgres-i asemel kasutusele võtma.



Joonis 24. Koostatud päringu käivitamise jadadiagramm.

Nagu jooniselt näha, siis peale tagarakendusele päringu saatmist loob node-postgres andmebaasiühenduse objekti ja proovib selle abil ühenduda andmebaasiga. Kui ühendumine oli edukas, siis käivitatakse koostatud päring andmebaasis. Vastuseks saadakse JSON formaadis objekt, mis sisaldab päringu tulemust ning see objekt saadetakse brauserile vastuseks. Andmebaasilt tabelite, veergude ja kitsenduste kohta andmete küsimine käib sama loogika järgi, kuid brauser ei saada POST päringut tehes rakenduselte SQL lauset, sest need laused on defineeritud tagarakenduses (vt jaotis 5.3). Mõne teise andmebaasisüsteemi toe lisamisel tuleks koostada nende jaoks sobivad SQL laused andmebaasist samasuguse informatsiooni saamiseks.

7.4 Mitmekeelsuse tugi

Mitmekeelsuse toe lisamiseks on loodud JSON objekt, kus on kirjapandud erinevate veebilehel asuvate tekstide tõlked (Joonis 25). Veebilehel kuvatav keel hoitakse meeles Reduxi abil. Tõlgitav tekst sisestatakse veebilehe HTML-i JSON objekti väljale viidates (Joonis 26).

```
{eng: {
  loginForm: {
    title: „Connect to database“,
    ...
    },
  ...
  },
est: {
  loginForm: {
    title: „Ühendu andmebaasiga“,
    ...
    },
  ...
  }
}
```

Joonis 25. Veebirakenduse tõlgete JSON objekti struktuuri näide

```
<Label htmlFor="hostAddress">
  {translations[props.language.code]['loginFrom']['title']}
</Label>
```

Joonis 26. Veebirakenduse lähtekoodi HTML-is tõlgete JSON objektile viitamine

7.5 SQL süstimine

SQL-süstimine tähendab seda, et kurjade kavatsustega kasutaja annab programmile spetsiaalselt valitud sisendi, mille alusel genereerib ja käivitab programm SQL lause, mida programmi kirjutaja polnud ette näinud. Käesolevas päringute koostamise programmis peab kasutaja paljud valikud tegema läbi graafilise kasutajaliidese (nt tegema valiku liitboksist). Siiski leidub kasutajaliideses ka kohti, kuhu kasutaja peab ise midagi sisestama:

- tabeli alias,
- veeru alias,
- piirav tingimus,
- veerule rakendatav funktsioon.

SQL süstimist saab piirata parameetriseeritud päringute abiga. Nende kasutamine aitab koostatud päringus käsitleda paomärke sõne tüübina (Joonis 27, Joonis 28), seda nimetatakse *escape*-imiseks, ning seetõttu vähendada SQL süstimise võimalusi. Joonis 27 olev päring tagastab kõik kasutajad, kuid Joonis 28 olev päring otsib kasutajanimesid, mis on võrdsed tühja stringiga.

```
SELECT * FROM Users WHERE username=’’ OR ’1’=’1’
```

Joonis 27. Päring enne *escape*-imist

```
SELECT * FROM Users WHERE username=’\’ OR \’1\’=\’1\’’;
```

Joonis 28. Päring peale *escape*-imist

Parameetriseeritud päringute abil on võimalik ka kontrollida kasutaja poolt sisestatud väärtuseid ning selle abil vältida ebasobivate väärtuste sattumist päringusse.

Korralduslikult poolelt aitab SQL süstimise vastu võidelda see, kui päringute tegijate jaoks luuakse PostgreSQLi andmebaasis võimalikult piiratud õigustega kasutaja, kellel pole õigust teha skeemimuudatusi, andmemuudatusi ja küsida andmeid talle mitte-ettenähtud tabelitest.

7.6 Installeerimine

Eelnevalt rakenduse ülesseadmist peab serveris olema ülesseadistatud vähemalt Node.js tarkvara. Lisaks võiks olla serveris protsesside haldamise vahend pm2 [41].

Rakenduse arendamisel kasutati npm paketihaldamissüsteemi [42]. Npm võimaldab kergesti hallata projektis vaja minevaid teeke ja laiendusi ning konfigureerida erinevaid projektiga seotud andmeid nagu projekti versioon, autor ja litsents. Npm konfiguratsioon on kirjeldatud failis *package.json*. Nii ees- kui ka tagarakenduse jaoks on loodud eraldi *package.json* fail, et hoida rakenduste sõltuvused üksteisest lahus.Eesrakenduse üles seadmine toimub järgnevalt.

- Esimese sammuna tuleb allalaadida eesrakenduse sõltuvused. Seda saab teha käivitadades terminalis käsu *npm install* (eeldatakse, et arvutis on npm olemas). Käsku tuleb käivitada lähtekoodi kaustas *client*.

- Teiseks tuleb *config.js* failis, mis on lähtekoodi kasutas *client*, määrata ära tagarakenduse baas-URL. apex.ttu.ee serverisse ülespandud rakenduse puhul oli see *http://apex.ttu.ee:8080/postgres-query/api*.
- Kolmandaks tuleb luua eesrakendusest kompileeritud versioon. Selleks tuleb käivitada lähtekoodi kaustas *client* käsk *npm run build*. Seejärel tekib, kausta *client*, *build* nimeline kaust. Selle sees on eesrakenduse kompileeritud versioon.
- Neljandaks tuleb eesrakenduse kompileeritud versioon paigutada serverisse. Selleks tuleb võtta kaustast *build* kõik failid ja kopeerida need serveri juurdomeeni kataloogi (Apache serveri puhul võib selle kasuta nimi olla näiteks www või htdocs, oleneb serveri konfiguratsioonist, apex.ttu.ee serveris on selle kasuta nimi htdocs) loodud kausta (apex.ttu.ee serveris on selleks kaustaks *postgres-query*).

Tagarakenduse üles seadmine toimub järgnevalt.

- Esiteks tuleks kopeerida lähtekoodi kaustas *server* olevad failid serverisse. apex.ttu.ee serveris on selleks kaustas *postgres-query* kaust *api*.
- Teiseks tuleb allalaadida tagarakenduse sõltuvused. Seda tuleb teha serveris selles kaustas kuhu failid on kopeeritud.
- Kolmandaks tuleb käivitada tagarakendus. Selle jaoks on *package.json* failis defineeritud kaks käsku. Käivitades serveri terminalist *npm start* pannakse tagarakendus käima kuid terminali kinni pannes lakkab ka tagarakendus töötamast. Selle probleemi lahendamiseks on defineeritud käsk *npm run pm2-start*. Käsu käivitamine eeldab, et serverisse on paigaldatud protsesside haldamise vahend pm2. Peale selle käsu käivitamist töötab tagarakendus konstantselt.
- Lisaks on veel võimalik seadistada üles tagarakenduse taaskäivitumine peale serveri taaskäivitumist. Selle jaoks on vaja terminalis käivitada käsk *pm2 startup*, mille järel genereeritakse serverile vastav konfiguratsiooni käsk ning see kuvatakse terminalis (näide genereeritud käsu kujust, ei pruugi täpselt kattuda, *sudo su -c "env PATH=$PATH:/home/unitech/.nvm/versions/node/v4.3/bin pm2 startup <distribution> -u <user> --hp <home-path>*). Genereeritud käsk tuleb kopeerida ning terminalis käivitada. Seejärel tuleb veel terminalis käivitada käsk *pm2 save*, et salvestada protsesside list.

7.7 Ühiktestid

Ühiktestide koostamiseks kasutati projektis JavaScriptile mõeldud testimise raamistikku jest (versioon 23.6.0) [43]. Ühiktestid aitavad tõsta koodi toimimise usaldavust ning muudab koodi taaskasutatavamaks, sest testide kirjutamiseks peab kood olema modulaarne. Seega muutub ka koodi loetavus palju paremaks ning vigade parandamine lihtsamaks.

Projektis kasutatakse ühikteste, et veenduda genereeritud SQL lause õigsuses, kontrollitakse rakenduse oleku haldamise funktsioonide korrektset toimimist ning kasutjaliidese funktsionaalsuse korrektset toimimist.

Eesrakenduse testimisel kasutatakse veel ka teeki moxios (versioon 0.4.0) [44] ja redux-mock-store (versioon 1.5.3) [45]. Moxios võimaldab luua axiose libaobjekti, mille abil saab testida axiose teel saadud tagarakenduse vastuste korrektset käsitlemist. Redux-mock-store abil luuakse testides Reduxi lao libaobjekt. Selle abil testitakse, et lao tegevuste käivitamisel toimub laos korrektne oleku muutus.

Kokku kirjutati projekti käigus 60 ühiktesti, reakattuvusega 20%.

8 Kasutajate tagasiside

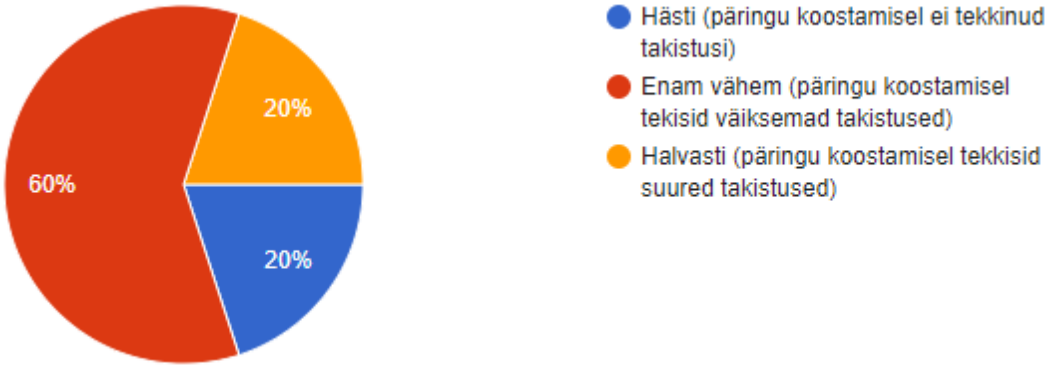
Kasutajatelt tagasiside saamiseks koostasin tagasiside vormi Google Formi veebirakenduse abil ja kolm päringut mida lasin kasutajate rakendust kasutades luua. Tagasiside vormis andsin ette SQL laused ning palusin neil luua vahendi abil samasugused laused. Seejärel küsisin kasutajatelt iga lause kohta kas nad said selle koostamisega hakkama ning kuidas see edenes. Lõpus küsisin paar üldisemat küsimust rakenduse kohta. Kõik tagasiside käigus esitatud küsimused on leitavad lisast Lisa 4. Tagasiside vorm saatsin viiele inimesele, kes kõik olid infotehnoloogia valdkonna kaastudengid. Kolmel tudengil oli vähe teadmisi PostgreSQL kohta, sest nad olid varem kasutanud PostgreSQLi ainult lihtsamate lausete loomiseks. Ülejäänud kaks tudengit teadsid PostgreSQL kohta rohkem, sest nad olid loonud varem PostgreSQL abil andmebaasi. Tagasiside viisin läbi ajavahemikul 10.05.2019–11.05.2019. Seega ei kajastu tagasisides peale selle perioodi lõppu tehtud muudatused nagu näiteks ühendamisoperatsiooni kirjeldamisel osalevate tabelite selgem väljatoomine.

8.1 Tagasiside päringute koostamise kohta

```
SELECT isik.eesnimi AS eesnimi, isik.perenimi AS perenimi, isik.elukoht AS elukoht
FROM public.isik
WHERE (eesnimi LIKE 'L%')
ORDER BY isik.isik_id DESC;
```

Joonis 29. Esimene päring.

Esimese päringu (Joonis 29) koostamisega said kõik tagasiside vormi täitnud vastajad hakkama. Enamus vastajaid ütlesid, et neil tekkisid väiksed takistused esimese päringu koostamisel ning ühel vastajal tekkisid päringu koostamisel suuremad takistused (Joonis 30).



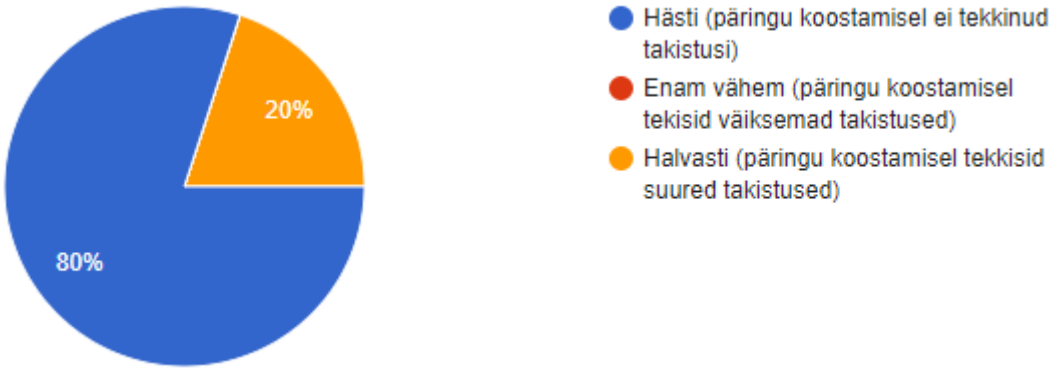
Joonis 30. Esimese päringu koostamise ringdiagramm.

Toodi välja, et vihjed nuppudel aitaksid aru saada, mida mingile nupule vajutus teeb. Samuti ei saadud kohe aru kuidas piirangute ja funktsioonide väljad töötavad ning arusaamiseks oli vaja katsetada. Kasutajate arvates läks veebilehel orienteerumiseks natukene aega.

```
SELECT i.riik_kood, COUNT(riik.nimetus) AS isikuid_riigist_kokku
FROM public.isik AS i
INNER JOIN riik ON (riik.riik_kood = i.riik_kood)
WHERE (i.riik_kood LIKE 'USA')
GROUP BY i.riik_kood;
```

Joonis 31. Teine päring.

Teise päringu (Joonis 31) koostamisega said samuti kõik hakkama ning kõik peale ühe vastaja ütlesid, et päringu koostamine läks neil sujuvalt (Joonis 32).



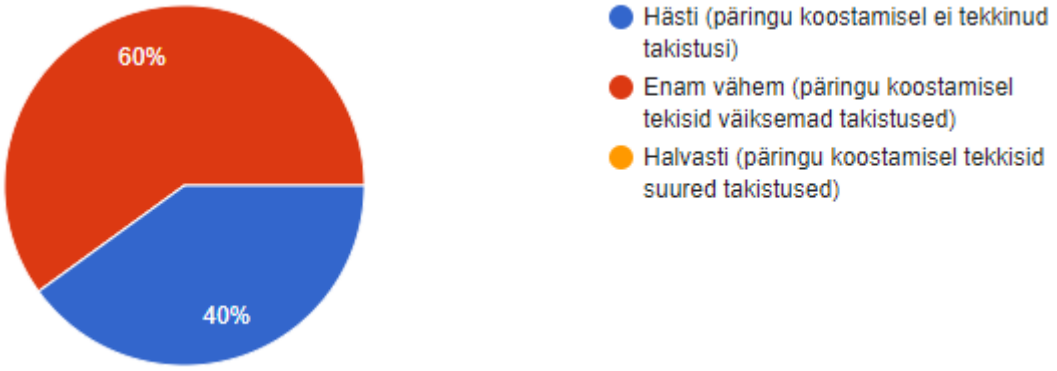
Joonis 32. Teise päringu koostamise ringdiagramm.

Vastajad mainisid, et teise päringu koostamisel oli rakendus arusaadavam ning tekkis vähem probleeme. Kõige rohkem kurdeti, et tabeli aliase muutmise funktsionaalsus ei olnud ilmne ning selle leidmine võttis aega.

```
SELECT DISTINCT ON (parkla.nimi)
parklakoht.parklakoha_kood,
parkla.nimi AS parkla_nimi,
teekate.nimetus AS teekate,
parkla_seisundi_liik.nimetus AS parkla_seisund,
isik.eesnimi,
isik.perenimi
FROM public.parklakoht
INNER JOIN parkla ON (parkla.parkla_kood = parklakoht.parkla_kood)
INNER JOIN teekate ON (teekate.teekatte_kood = parklakoht.teekatte_kood)
INNER JOIN parkla_seisundi_liik ON (parkla_seisundi_liik.parkla_seisundi_liik_kood = parkla.parkla_seisundi_liik_kood)
INNER JOIN tootaja ON (tootaja.isik_id = parklakoht.reg_isik_id)
INNER JOIN isik ON (isik.isik_id = tootaja.isik_id);
```

Joonis 33. Kolmas päring.

Kolmanda päringu (Joonis 33) koostamisega said ka kõik hakkama. Kolm vastajat ütlesid, et päringu koostamisel tekkistid väiksed probleemid, ülejäänutel probleeme ei tekkinud (Joonis 34).



Joonis 34. Kolmanda päringu koostamise ringdiagramm.

Mainiti, et päringusse tuleb lisada palju tabeleid ning tabelit järjekorda ei ole võimalik muuta, kuid muidu oli päringut mugav luua. Ühe inimese arvates on rakendus õpitav. Samuti, kui päringus on palju veerge, siis on raske aru saada, millisesse kasti on juba midagi kirjutatud kuna väljas olev kohahoidja tekst oli väga sarnast värvi.

8.2 Üldine tagasiside

Enamuste kasutajate arvates oli rakendust mugav kasutada. Üks tagasisidet andnud vastaja ütles peale teise päringu koostamist, et tema arvates on rakendus õpitav ning kõige rohkem häiris teda, et päringu SQL kuju ei olnud kogu aeg näha.

Tagasiside andnud vastajatele meeldis rakenduse kujundus ning nende arvates oli seda mugav kasutada. Samuti arvati, et rakendusega oleks teatud tüüpi SQL päringuid kiirem teha võrreldes nende SQL koodi kirjutamisega. Toodi välja, et suuremates SQL päringutes oleks selle abil kergem muuta näiteks DISTINCT ON, GROUP BY ja ORDER klausleid.

Rakenduse juures ei meeldinud neile, et piirangute sisestamisel peab kogu piirangu ise sisestama. Lisaks oleks võinud nuppude juures olla rohkem vihjeid ning lehe elemendid oleks võinud võtta vähem ruumi. Mõned arvasid, et SQL peaks olema koguaeg nähtavas kohas, ning kasutaja peaks seal saama ise muudatusi teha.

8.3 Tagasiside alusel rakenduse parandamine

Saadud tagasiside alusel tegin ma rakenduse kasutajaliidesesse järgmised muudatused:

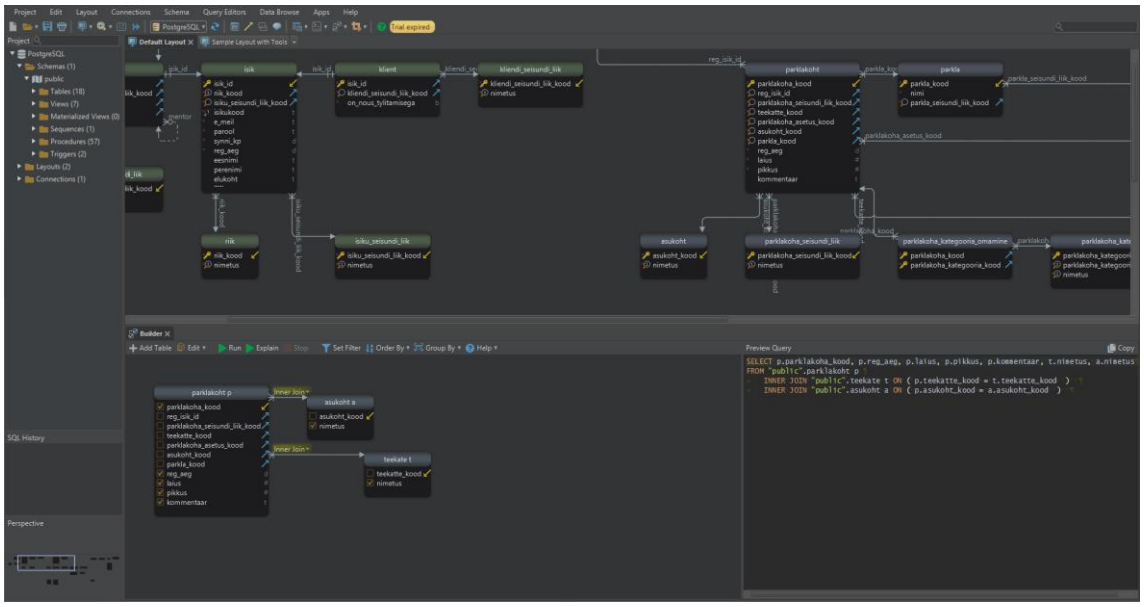
- muutsin tabelite, veergude komponendid visuaalselt kompaktsemaks,
- SQL alamaken kuvatakse koheselt peale andmebaasiga ühendumist,
- genereeritud SQL on esitatud kasutajale loetavamal kujul,
- nupu ja välja peal hiirt hoides kuvatakse kasutajale nuppu või välja selgitav tekst,
- veeru piirangute välja tegin suuremaks, et oleks mugavam sisestatud piirangut lugeda ja piirangut ennast sisestada.

9 Võrdlus olemasoleva tarkvaraga

Enda loodud tarkvara olemasoleva võrdlemiseks võtan võrdlusesse kaks rakendust – ühe kasutaja arvutisse installeeritava rakenduse ja ühe veebipõhise rakenduse. Kasutaja arvutisse installeeritavaks rakenduseks valin DbSchema (versioon 8.1.3) ja veebirakenduseks valin Skyvia Query Builderi. Valisin võrdlusesse need kaks rakendust, sest DbSchema oli kasutajaliidese vaatenurgast kõige sarnasem installeeritav programm, mille leidsin ning Skyvia Query Builder oli oma kasutajaliidese poolest kõikidest programmidest kõige unikaalsem.

9.1 DbSchema

Siin alampeatükis võrreldakse loodud tarkvara programmiga DbSchema (Joonis 35).



Joonis 35. Rakenduse *DbSchema* päringu koostamise vaade

Lõputöö rakenduse puudused võrreldes DbSchemaga.

- DbSchemas andmebaas kuvatakse päringute koostajale diagramina, kust on kohe näha, millised tabelid on omavahel välisvõtmete kaudu ühendatud.
- DbSchemas päringu koostamisel kuvatakse päring diagrammina, kus näidatakse ühendamisoperatsioone ning tabeli lisamisel päringusse pakutakse automaatselt välja ühendamisoperatsioon, ilma, et kasutaja peaks selleks midagi tegema.
- DbSchema kasutajaliides on kompaktsem.

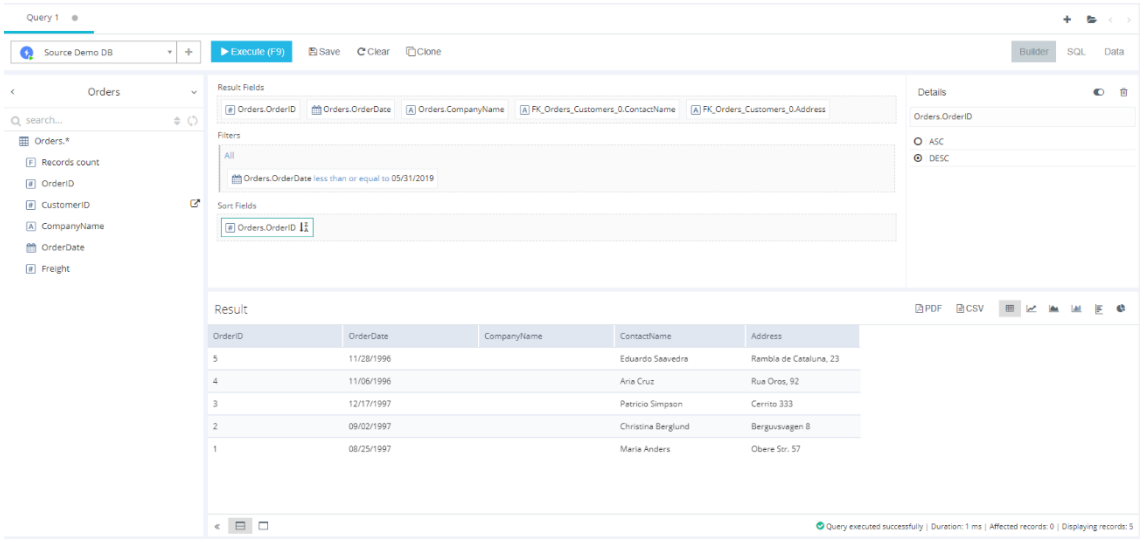
- DbSchemas pakutakse tabelitele välja aliased, et vähendada lause pikkust.
- DbSchemas saab teha peale SELECT päringute ka teisi opertsioone andmebaasiga.
- DbSchemat saab kasutada võrku ühendamata.
- DbSchemas on lihtsam veergudele piiranguid seada.
- DbSchemas on mõeldud andmebaasi üldisemaks haldamiseks seega on sel hulk funktsionaalsusi sellle jaoks nagu tugi erinevate andmebaasisüsteemide jaoks ja päringu täitmisplaani vaatamine.

Lõputöö rakenduse eelised võrreldes DbSchemaga.

- DbSchemas vahendis saab päringus tabeleid ühendada ainult välisvõtmete alusel – kui neid ei ole siis ei saa ka päringus nende veergude põhjal tabelite ühendamise operatsiooni läbi viia.
- DbSchema rakendus ei luba teha päringuid andmebaasi süsteemsete tabelite põhjal nagu näiteks tabelid, mis asuvad skeemis *information_schema*.
- DbSchema ridade hulka piiravad tingimused esitatakse graafilises vaates tabeli all ja kui neid on palju, siis on raske leida kõiki ühe veeru kohta käivaid tingimusi.
- DbSchemas on ühe tabeli ridu võimalik sorteerida ainult ühe veeru järgi.
- DbSchema on tasuline programm.
- DbSchema on vaja kasutajal allalaadida ja installerida oma arvutisse.

9.2 Skyvia Query Builder

Siin alampeatükis võrreldakse loodud tarkvara programmiga Skyvia Query Builder (Joonis 36).



Joonis 36. Rakenduse *Skyvia Query Builder* päringu loomise vaade

Lõputöö rakenduse puudused võrreldes Skyvia Query Builderiga.

- Skyvia Query Builderil on tugi mitmete erinevate andmebaasisüsteemide jaoks (Oracle, MySQL, Google BigQuery, Amazon Redshift).
- Skyvias saab kasutaja salvestada oma andmebaaside ühendusi ja hiljem neid lihtsalt kasutada.
- Skyvias on lihtsam lisada päringu veerule piiranguid.
- Skyvias saab päringu tulemusi peale tavalise tabeli vaate vaadata ka erinevate graafidena.

Lõputöö rakenduse eelised võrreldes Skyvia Query Builderiga.

- Skyvia kasutajaliideses tegeleb kasutaja ainult tabeli veergudega ning kasutaja ise ei lisa ega ühenda päringus tabeleid. Seetõttu on raske öelda, millised tabelid on päringusse lisatud.
- Andmebaasi tabelite vahel navigeerimine on segane, sest korraga näidatakse kasutajale ainult ühe tabeli veerge.
- Skyvia tasuta versioon võimaldab teha ainult viis päringut päevas.
- Skyvias peab enne andmebaasiga ühendamist ütlema, millist skeemi kasutatakse, ning teise skeemi peale minemiseks peab looma andmebaasiga uue ühenduse.

10 Kokkuvõte

Käesoleva töö eesmärgiks oli luua avatud lähtekoodiga, veebipõhise kasutajaliidesega rakendus, mis võimaldaks kasutajal luua päringuid (SELECT lauseid) PostgreSQL andmebaaside põhjal.

Töö tulemusena loodi eesmärgiks seatud rakendus. Rakenduse lähtekood kaitsti MIT litsentsiga ja tehti maailmale avalikuks GitHubi hoidlas: https://github.com/Erikdzo/postgres-visual-query-app. Tulemuse headuse hindamiseks paluti seda rakendust kasutada teistel inimestel (kaasüliõpilastel), kes andsid rakenduse kohta tagasisidet. Tagasiside alusel tehti rakenduse kasutajaliidesesse parandusi. Samuti võrreldi loodud rakendust kahe teise samalaadse rakendusega.

Rakenduse loomisel kasutati mitmeid erinevaid raamistikke ja tehnoloogiaid (React, Redux, axios, lodash, Squel.js, Bootstrap, Node.js, Express, node-postgres), mis peaks tagama rakenduse laiendatavuse ja arhitektuuri arusaadavuse. Rakenduses on ka mitmekeelse kasutajaliidese tugi.

Hetkel ei toeta rakendus kõiki võimalusi, mida PostgreSQL SELECT lause päringute kirjutajale pakub. Näiteks ei saa koostada koondandmete päringutele piiravaid tingimusi HAVING klausli näol, kasutada ühiseid tabeli avaldisi, aknafunktsioone, kasutada tabelite ühendi/ühisosa/vahe leidmise operaatoreid või piirata tagastatud ridade hulka (FETCH FIRST n ROWS ONLY klausel). Pärimisseosega tabelite korral ei saa küsida ülatabelist neid andmeid, mis on ülatabelis, kuid ei ole alamtabelis. Töö edasiarenduseks oleks selliste võimaluste lisamine. Selle juures vajab eraldi uurimist, kuidas oleks kõige kasutajasõbralikum selliseid valikuid läbi graafilise kasutajaliidese teha.

Hetkel toetab rakendus ainult PostgreSQL andmebaase ning SELECT lauseid. Edasiarendusena saaks lisada teiste andmekäitluse lausete (UPDATE, DELETE või INSERT) koostamise võimaluse. Selle jaoks oleks vaja luua vajalikud komponendid ning funktsionaalsus lausete genereerimiseks. Samuti saaks lisada toe kasutada teisi andmebaasisüsteeme nagu Oracle või MySQL. Teiste andmebaasisüsteemide toetamiseks oleks vaja luua vastavad süsteemikataloogi päringud andmebaasist

informatsiooni saamiseks, vastavad komponendid ja kohandada päringu genereerimise koodi vastavalt konkreetsele andmebaasisüsteemile. Siiski olgu öeldud, et PostgreSQL andmebaasis saab luua väliseid tabeleid ning nende kaudu muuta andmebaasis kasutatavaks andmed, mis ei ole selles samas PostgreSQL andmebaasis, vaid on, näiteks, mõnes MySQL või Oracle andmebaasis hoopis teises serveris. Selle mehhanismi kaudud on juba praegu teiste andmebaasisüsteemide hoole all olevad andmed ka selle vahendi abil kasutatavad, sest rakendus võimaldab teha päringuid ka väliste tabelite põhjal.

Kasutatud materjalid

[1] „phpPgAdmin,“ [Võrgumaterjal]. Available: http://phppgadmin.sourceforge.net/doku.php. [Kasutatud 24 02 2019].

[2] „postgreSQL,“ [Võrgumaterjal]. Available: https://www.postgresql.org/. [Kasutatud 24 02 2019].

[3] A. R. Hevner, S. T. March, J. Park ja S. Ram, „Design Science in Information Systems Research,“ *MIS Quarterly,* kd. 28, nr 1, pp. 75-105, 2004.

[4] „Lähtekood,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/L%C3%A4htekood. [Kasutatud 24 02 2019].

[5] M. M. Zloof, „Query by Example,“ %1 *AFIPS '75 Proceedings of the May 19-22, 1975, national computer conference and exposition,* Anaheim, 1975.

[6] M. M. Zloof, „Query by example: a database language,“ *IBM Systems Journal,* kd. 16, nr 4, 1977.

[7] „DbSchema,“ [Võrgumaterjal]. Available: https://www.dbschema.com/index.html.

[8] „Active Database Software,“ [Võrgumaterjal]. Available: https://www.activedbsoft.com/overview-querytool.html.

[9] „RazorSQL,“ [Võrgumaterjal]. Available: https://razorsql.com/updates.html.

[10] „Aquafold,“ [Võrgumaterjal]. Available: https://www.aquafold.com/aquadatastudio_whats_new.

[11] „Skyvia,“ [Võrgumaterjal]. Available: https://skyvia.com/query/sql-query-builder.

[12] „Datasparc,“ [Võrgumaterjal]. Available: https://www.datasparc.com/.

[13] „Web-based Visual Query Designer,“ [Võrgumaterjal]. Available: http://web-vqd.sourceforge.net/.

[14] „Active Query Builder Web API,“ [Võrgumaterjal]. Available: https://webapi.activequerybuilder.com/.

[15] K. B. e. al., „Agiilse tarkvaraarenduse manifest,“ 2001. [Võrgumaterjal]. Available: http://agilemanifesto.org/iso/et/manifesto.html. [Kasutatud 17 05 2019].

[16] T. Norman, „Agile Release Planning 101,“ 06 09 2012. [Võrgumaterjal]. Available: http://tommynorman.blogspot.com/2012/09/agile-release-planning-101.html. [Kasutatud 06 05 2019].

[17] Moutain Goat Software, „User Stories,“ [Võrgumaterjal]. Available: https://www.mountaingoatsoftware.com/agile/user-stories. [Kasutatud 06 05 2019].

[18] statcounter GlobalStats, „Desktop Browser Market Share Worldwide,“ [Võrgumaterjal]. Available: http://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201903-201905. [Kasutatud 06 05 2019].

[19] milmartin, „MIT litsents,“ 24 04 2014. [Võrgumaterjal]. Available: https://milmartin.wordpress.com/2014/04/24/mit-litsents/. [Kasutatud 19 05 2019].

[20] flaviocopes, „React concepts: declarative,“ 03 11 2018. [Võrgumaterjal]. Available: https://flaviocopes.com/react-declarative/. [Kasutatud 17 05 2019].

[21] Flaviocope, „Unidirectional Data Flow in React,“ 09 12 2018. [Võrgumaterjal]. Available: https://flaviocopes.com/react-unidirectional-data-flow/. [Kasutatud 11 05 2019].

[22] React Training, „React Router,“ [Võrgumaterjal]. Available: https://github.com/ReactTraining/react-router. [Kasutatud 20 05 2019].

[23] Bootstrap team, „Bootstrap,“ [Võrgumaterjal]. Available: https://getbootstrap.com/. [Kasutatud 20 05 2019].

[24] reactstrap, „reactstrap,“ [Võrgumaterjal]. Available: https://reactstrap.github.io/. [Kasutatud 20 05 2019].

[25] M. Wessel, „react-custom-scrollbars,“ [Võrgumaterjal]. Available: https://github.com/malte-wessel/react-custom-scrollbars. [Kasutatud 20 05 2019].

[26] tannerlinsley, „React Table,“ [Võrgumaterjal]. Available: https://github.com/tannerlinsley/react-table. [Kasutatud 20 05 2019].

[27] FontAwesome, „react-fomtawesome,“ [Võrgumaterjal]. Available: https://github.com/FortAwesome/react-fontawesome. [Kasutatud 20 05 2019].

[28] Lodash Utilities, „Lodash,“ [Võrgumaterjal]. Available: https://lodash.com/. [Kasutatud 20 05 2019].

[29] UI patterns, „Input Prompt,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/InputPrompt. [Kasutatud 20 05 2019].

[30] UI Patterns, „Vertical Dropdown Menu,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/VerticalDropdownMenu. [Kasutatud 20 05 2019].

[31] UI Patterns, „Cards,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/cards. [Kasutatud 20 05 2019].

[32] UI Patterns, „Navigation Tabs,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/NavigationTabs. [Kasutatud 20 05 2019].

[33] UI Patterns, „Drag and Drop,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/drag-and-drop. [Kasutatud 20 05 2019].

[34] The PostgreSQL Global Development Group, „SELECT,“ [Võrgumaterjal]. Available: https://www.postgresql.org/docs/current/sql-select.html. [Kasutatud 20 05 2019].

[35] UI Patterns, „Preview,“ [Võrgumaterjal]. Available: http://ui-patterns.com/patterns/LivePreview. [Kasutatud 20 05 2019].

[36] Redux, „Redux,“ [Võrgumaterjal]. Available: https://redux.js.org/. [Kasutatud 09 05 2019].

[37] React Redux, „React Redux,“ [Võrgumaterjal]. Available: https://github.com/reduxjs/react-redux. [Kasutatud 09 05 2019].

[38] M. V. d. Bergh, „React Redux: Building Modern Web Apps with the ArcGIS JS API,“ 08 09 2017. [Võrgumaterjal]. Available: https://www.esri.com/arcgis-blog/products/js-api-arcgis/3d-gis/react-redux-building-modern-web-apps-with-the-arcgis-js-api/. [Kasutatud 09 05 2019].

[39] hiddentao, „Squel.js,“ [Võrgumaterjal]. Available: https://hiddentao.github.io/squel/. [Kasutatud 11 05 2019].

[40] axios, „axios,“ [Võrgumaterjal]. Available: https://github.com/axios/axios. [Kasutatud 20 05 2019].

[41] KeyMetrics, „PM2,“ [Võrgumaterjal]. Available: http://pm2.keymetrics.io/. [Kasutatud 19 05 2019].

[42] npm, „npm,“ [Võrgumaterjal]. Available: https://www.npmjs.com. [Kasutatud 19 05 2019].

[43] jest, „jest,“ [Võrgumaterjal]. Available: https://jestjs.io/. [Kasutatud 20 05 2019].

[44] axios, „moxios,“ [Võrgumaterjal]. Available: https://github.com/axios/moxios. [Kasutatud 20 05 2019].

[45] dmitry-zaets, „redux-mock-store,“ [Võrgumaterjal]. Available: https://github.com/dmitry-zaets/redux-mock-store. [Kasutatud 20 05 2019].

Lisa 1 – Andmebaasi tabelite leidmine

```
SELECT table_schema, table_name, table_type
FROM information_schema.tables
UNION SELECT n.nspname AS table_schema, c.relname AS table_name,
'MATERIALIZED VIEW' AS table_type
    FROM pg_catalog.pg_class AS c
    INNER JOIN pg_catalog.pg_namespace AS n ON c.relnamespace=n.oid
    WHERE c.relkind='m'
ORDER BY table_schema, table_type, table_name;
```

Lisa 2 – Andmebaasi tabelite veergude leidmine

```
SELECT table_name,
    column_name,
    ordinal_position,
    data_type
FROM information_schema.columns
UNION SELECT n.nspname AS table_schema, c.relname AS table_name,
a.attname AS column_name, a.attnum AS ordinal_position, t.typname AS
data_type
    FROM pg_catalog.pg_class AS c
    INNER JOIN pg_catalog.pg_namespace AS n ON c.relnamespace=n.oid
    INNER JOIN pg_catalog.pg_attribute AS a ON c.oid=a.attrelid
    INNER JOIN pg_catalog.pg_type AS t ON a.atttypid=t.oid
    WHERE c.relkind='m' AND a.attnum>0
    UNION SELECT n.nspname AS table_schema, c.relname AS table_name,
a.attname AS column_name, a.attnum AS ordinal_position, t.typname AS
data_type
        FROM pg_catalog.pg_class AS c
        INNER JOIN pg_catalog.pg_namespace AS n ON
c.relnamespace=n.oid
        INNER JOIN pg_catalog.pg_attribute AS a ON c.oid=a.attrelid
        INNER JOIN pg_catalog.pg_type AS t ON a.atttypid=t.oid
        WHERE c.relkind='r' AND a.attnum=-2
ORDER BY table_schema, table_name, ordinal_position;
```

Lisa 3 – Andmebaasi primaarvõtme, välisvõtme ja unikaalsuse kitsenduste leidmine

```
WITH keys AS
  (SELECT o.conname,
     (SELECT nspname
      FROM pg_namespace
      WHERE oid=m.relnamespace) AS key_schema,
         m.relname AS key_table,
         m.oid AS key_table_oid,
         o.conkey AS key_col,
         CASE
             WHEN o.contype='p' THEN 'PRIMARY KEY'
             ELSE 'UNIQUE'
         END AS contype
   FROM pg_constraint o
   INNER JOIN pg_class c ON c.oid = o.conrelid
   INNER JOIN pg_class m ON m.oid = o.conrelid
   WHERE o.contype IN ('p',
                       'u')
     AND o.conrelid IN
       (SELECT oid
        FROM pg_class c
        WHERE c.relkind = 'r')),
     keys_unnest AS
  (SELECT conname,
         key_schema,
         key_table,
         key_table_oid,
         key_col,
         key_col_num,
         ordin,
         contype
   FROM keys,
        unnest(keys.key_col) WITH
   ORDINALITY AS k(key_col_num, ordin)),
     keys_with_names AS
  (SELECT conname,
         key_schema,
         key_table,
         contype,
         array_agg(a_key.attname
                   ORDER BY ordin) AS key_col
   FROM keys_unnest k
   INNER JOIN pg_attribute a_key ON k.key_col_num = a_key.attnum
   AND k.key_table_oid = a_key.attrelid
   AND a_key.attisdropped = FALSE
   GROUP BY conname,
            key_schema,
            key_table,
            contype),
     fk AS
  (SELECT o.conname,
     (SELECT nspname
      FROM pg_namespace
      WHERE oid=f.relnamespace) AS foreign_schema,
```

```
          f.relname AS foreign_table,
          f.oid AS foreign_table_oid,
          o.confkey AS foreign_col,
     (SELECT nspname
      FROM pg_namespace
      WHERE oid=m.relnamespace) AS target_schema,
          m.relname AS target_table,
          m.oid AS target_table_oid,
          o.conkey AS target_col
   FROM pg_constraint o
   INNER JOIN pg_class c ON c.oid = o.conrelid
   INNER JOIN pg_class f ON f.oid = o.confrelid
   INNER JOIN pg_class m ON m.oid = o.conrelid
   WHERE o.contype = 'f'
     AND o.conrelid IN
       (SELECT oid
        FROM pg_class c
        WHERE c.relkind = 'r')),
     fk_unnest AS
  (SELECT conname,
          foreign_schema,
          foreign_table,
          foreign_table_oid,
          foreign_col,
          foreign_col_num,
          target_schema,
          target_table,
          target_table_oid,
          target_col,
          target_col_num,
          ordin
   FROM fk,
        unnest(fk.foreign_col, fk. target_col) WITH
   ORDINALITY AS f(foreign_col_num, target_col_num, ordin)),
     fk_with_names AS
  (SELECT conname,
          foreign_schema,
          foreign_table,
          array_agg(a_foreign.attname
                    ORDER BY ordin) AS foreign_col,
          target_schema,
          target_table,
          array_agg(a_target.attname
                    ORDER BY ordin) AS target_col
   FROM fk_unnest fk
   INNER JOIN pg_attribute a_foreign ON fk.foreign_col_num =
a_foreign.attnum
   AND fk.foreign_table_oid = a_foreign.attrelid
   AND a_foreign.attisdropped = FALSE
   INNER JOIN pg_attribute a_target ON fk.target_col_num =
a_target.attnum
   AND fk.target_table_oid = a_target.attrelid
   AND a_target.attisdropped = FALSE
   GROUP BY conname,
            foreign_schema,
            foreign_table,
            target_schema,
            target_table)
SELECT conname AS CONSTRAINT_NAME,
```

```
       'FOREIGN KEY' AS constraint_type,
       target_schema AS table_schema,
       target_table AS TABLE_NAME,
       target_col AS table_column,
       foreign_schema AS foreign_table_schema,
       foreign_table AS foreig_table_name,
       foreign_col AS foreign_column_name
FROM fk_with_names
UNION
SELECT conname,
       contype,
       key_schema,
       key_table,
       key_col,
       NULL,
       NULL,
       NULL
FROM keys_with_names
ORDER BY table_schema,
         TABLE_NAME,
         constraint_type,
         table_column;
```

Lisa 4 – Tagasiside küsimustik

Kui palju olete kokkupuutunud PostgreSQL-iga?

- ◯ Palju (olen kirjutanud keerulisemaid PostgreSQL lauseid või loonud oma andmebaasi kasutades PostgreSQLi)
- ◯ Natukene (olen kirjutanud lihtsamaid PostgreSQL lauseid)
- ◯ Mitte üldse (ei ole PostgreSQL lauseid kirjutanud)
- ◯ Other...

Päring 1

```
SELECT isik.eesnimi AS eesnimi, isik.perenimi AS perenimi, isik.elukoht AS elukoht
FROM public.isik
WHERE (eesnimi LIKE 'L%')
ORDER BY isik.isik_id DESC;
```

Kas saite esimese päringu koostamisega hakkama? *

- ◯ Jah
- ◯ Ei

Kuidas edenes esimese päringu koostamine? *

- ◯ Hästi (päringu koostamisel ei tekkinud takistusi)
- ◯ Enam vähem (päringu koostamisel tekisid väiksemad takistused)
- ◯ Halvasti (päringu koostamisel tekkisid suured takistused)
- ◯ Other...

Palun põhjendage eelmist vastust *

Long answer text

Päring 2

```
SELECT i.riik_kood, COUNT(riik.nimetus) AS isikuid_riigist_kokku
FROM public.isik AS i
INNER JOIN riik ON (riik.riik_kood = i.riik_kood)
WHERE (i.riik_kood LIKE 'USA')
GROUP BY i.riik_kood;
```

Kas saite teise päringu koostamisega hakkama? *

- ○ Jah
- ○ Ei

Kuidas edenes teise päringu koostamine? *

- ○ Hästi (päringu koostamisel ei tekkinud takistusi)
- ○ Enam vähem (päringu koostamisel tekisid väiksemad takistused)
- ○ Halvasti (päringu koostamisel tekkisid suured takistused)
- ○ Other...

Palun põhjendage eelmist vastust *

Long answer text

Päring 3

```
SELECT DISTINCT ON (parkla.nimi)
parklakoht.parklakoha_kood,
parkla.nimi AS parkla_nimi,
teekate.nimetus AS teekate,
parkla_seisundi_liik.nimetus AS parkla_seisund,
isik.eesnimi,
isik.perenimi
FROM public.parklakoht
INNER JOIN parkla ON (parkla.parkla_kood = parklakoht.parkla_kood)
INNER JOIN teekate ON (teekate.teekatte_kood = parklakoht.teekatte_kood)
INNER JOIN parkla_seisundi_liik ON (parkla_seisundi_liik.parkla_seisundi_liik_kood = parkla.parkla_seisundi_liik_kood)
INNER JOIN tootaja ON (tootaja.isik_id = parklakoht.reg_isik_id)
INNER JOIN isik ON (isik.isik_id = tootaja.isik_id);
```

Kas saite kolmanda päringu koostamisega hakkama? *

- ◯ Jah
- ◯ Ei

Kuidas edenes kolmanda päringu koostamine? *

- ◯ Hästi (päringu koostamisel ei tekkinud takistusi)
- ◯ Enam vähem (päringu koostamisel tekisid väiksemad takistused)
- ◯ Halvasti (päringu koostamisel tekkisid suured takistused)
- ◯ Other...

Palun põhjendage eelmist vastust *

Long answer text

Kas teie arvates oli rakendust mugav kasutada?

- ◯ Jah
- ◯ Ei
- ◯ Other...

Mis meeldis rakenduse juures? *

Long answer text

Mis ei meeldinud rakenduse juures? *

Long answer text

Mida oleks rakenduses paremini saanud teha? *

Long answer text

Kui leidsite päringute koostamisel rakendusest mingi vea, siis palun kirjutage sellest siia.

Long answer text