TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ragnar Pärnamäe 182789IAPM

POSTGRESQL VEEBIPÕHISE VISUAALSE ANDMEKÄITLUSKEELE LAUSETE KOOSTAMISE TARKVARA EDASIARENDAMINE

Magistritöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2020

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: Ragnar Pärnamäe

12.05.2020

Annotatsioon

Käesoleva töö eesmärgiks on edasi arendada PostgreSQL andmebaasides päringute (SELECT lausete) visuaalseks koostamiseks mõeldud avatud lähtekoodiga veebirakendust *Postgres Visual Query Builder*. Ülesandeks on realiseerida uusi funktsionaalsuseid ja ka täiendada graafilist kasutajaliidest. Selleks tuleb leida viisid mitmete SQL andmekäitluskeele lausete konstruktsioonide visuaalseks esitamiseks ning realiseerida see esitus tarkvaras. Lisaks on eesmärgiks kirjeldada mustrite formaadis visuaalselt andmekäitluskeele lausete koostamist võimaldavate vahendite graafilise kasutajaliidese häid praktikaid.

Töös kasutatakse disainiteaduse metoodikat ning töö tulemuseks on tehnilised tehised (täiendatud rakendus, mustrid) ja uus teadmine selle kohta, kuidas oleks graafilisi päringute koostamise vahendeid parem koostada. Töö sisendiks on teadmised valdkonna kohta, mida ammutatakse erinevatest teadusartiklitest ning sarnaste olemasolevate rakenduste vaatlusest. Enne arenduse alustamist tutvutakse rakenduse esimese versiooniga ning töös tuuakse lühidalt välja selle positiivsed küljed ja ka puudujäägid. Arenduse kavandamiseks kasutatakse Normani poolt pakutud agiilse väljalaske planeerimise meetodit, mille alusel koostatakse väljalaske plaan. Arendamine toimub iteratiivselt ning järgitakse paindmetoodikate parimaid praktikaid. Tarkvara arendatakse edasi esimeses versioonis kasutusel olnud vahendites – tagarakenduse realiseerimiseks kasutatakse Node.js ning kasutajaliidese realiseerimiseks kasutatakse React raamistikku. Täienduse tulemusena lisandub tarkvarasse uusi võimalusi nagu näiteks alampäringute kasutamise võimalus päringu otsingutingimustes ja hulgaoperatsioonide tegemise võimalus. Võrreldavates rakendustes on selliseid võimalusi vähe või üldse mitte.

Töö käigus loodud täiendatud rakendust valideeritakse nii autori enda poolt kui ka testkasutajate abil. Saadud tagasiside põhjal hinnatakse rakenduse headust ja tehakse sellesse parandusi.

Töö tulemuseks on *Postgres Visual Query Builderi* uus versioon, mille lähtekood on tehtud MIT litsentsiga kaitstult avalikuks ning mis on kättesaadav GitHubi hoidlas:

https://github.com/rax1110/postgres-visual-query-v2
Veebirakendus asub hetkel aadressil: http://apex.ttu.ee/postgres-query/

Töö tulemusena on ka leitud sobilikud kasutajaliidese disainimustrid visuaalsete andmekäitluskeele rakenduste jaoks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 92 leheküljel, 11 peatükki, 75 joonist, 12 tabelit.

Abstract

Further Development of a PostgreSQL Visual Query Design Software

The goal of this thesis is to extend a web-based open-source application (*Postgres Visual Query Builder*) that is meant for creating queries (SELECT statements) based on PostgreSQL databases. The task is to implement additional functionality as well as improve the graphical user interface. To do so, one has to find ways to visually represent some of the constructs of SQL data manipulation language sentences. In addition, the aim is to describe some of the user interface best practices of visual query tools in a pattern format.

Currently a new trend is gaining momentum. More and more applications are being created that perform the function of a programming language but do not require in-depth programming knowledge. Such applications are called low-code and no-code platforms. In this thesis a similar platform will be further developed that allows creating PostgreSQL database queries through a visual user interface. The application is web-based and its source code is publicly available. Various web-based visual query builder applications were examined in this study. No other such application was found that would be web-based, easy to use, free of charge, and with such functionalities that would offer the same capabilities as *Postgres Visual Query Builder*.

The methodology of research is design science and the results of the work are technical artifacts (improved application, patterns) and a new knowledge about creating visual query tools. The input of the work is knowledge of the field, which is obtained from various scientific articles and by observation of similar existing applications. Before starting the development, the first version of the application is examined and its advantages and disadvantages are briefly highlighted. The agile release planning method proposed by Norman is used to plan the development. Based on this, a release plan is prepared. The development process is iterative and some of the best practices of agile methodologies are followed. The software is further developed in the tools used in the first version – Node.js is used to implement the backend application and the React framework is used to implement the user interface. The resulting software has new

functionalities like, for instance, possibility to use subqueries in search conditions or possibility to use set operations. Similar visual query construction programs lack these features entirely or support these in a limited manner.

The resulting new version of the software that is created during the work is validated both by the author himself and with the help of test users. Based on the received feedback, the goodness of the application is assessed and corrections are made.

As a result of this work, the source code of the new version of *Postgres Visual Query Builder*, protected by MIT license, has been made publicly available through GitHub repository:
https://github.com/rax1110/postgres-visual-query-v2
The web application is currently available at: http://apex.ttu.ee/postgres-query/

Moreover, a set of suitable user interface design patterns has been found for visual query construction applications.

The thesis is in Estonian and contains 92 pages of text, 11 chapters, 75 figures, 12 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
CSV	*Comma Separated Values,* komaeraldusega väärtused
GET-päring	HTTP meetod, mis küsib serverist andmeid
HTML	*Hypertext Markup Language*, hüperteksti märgistuskeel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
IBM	*The International Business Machines Corporation,* USA rahvusvaheline infotehnoloogia ettevõte
JSON	*JavaScript Object Notation*, JavaScripti objektide notatsioon
JSX	*JavaScript Extensible Markup Language,* Reactis kasutatav JavaScripti XML süntaks
lint, linter	Tööriist programmikoodi analüüsimiseks, tuvastab programmeerimisvigu ning stiilivigu
POST-päring	HTTP meetod, mis saadab andmeid kliendilt serverile
päring	*Query* Andmebaasikeele lause andmebaasist andmete otsimiseks. SQLis on selleks SELECT lause.
QBE	*Query By Example*, päring näite järgi, graafiline andmebaasikeel
SQL	*Structured Query Language*. Standardiseeritud keel, mis sisaldab nii andmekäitluse, andmekirjelduse, tehingute juhtimise kui õiguste haldamise lauseid. Keel realiseerib relatsioonilist andmemudelit ja on kasutusel SQL-andmebaasisüsteemides.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Enamik ettevõtteid kasutab või pakub tänapäeval arvutipõhiseid teenuseid, mis sisemiselt vajavad toimimiseks andmebaase. Paljud andmebaasid on realiseeritud kasutades SQL-andmebaasisüsteeme, kus andmete haldamiseks on kasutusel SQL (*Structured Query Language*) andmebaasikeel. 2020. aasta mai seisuga on kümnest kõige populaarsemast andmebaasisüsteemist seitse SQL-andmebaasisüsteemid [1]. Sellest tingituna on vaja SQLi (sh selle süntaksi) tundvaid inimesi, kes suudaksid andmebaase hallata ning kasutada. Lisaks teenuste tööd tagavatele IT-töötajatele on ka aina rohkematel äripoole inimestel soov ja vajadus kasutada andmebaasidesse kogunenud andmeid kiiresti ja võimalik, et ilma vahendajateta. Alati ei ole aga pädevaid spetsialiste käepärast võtta või puuduvad vahendid vastavate oskustega inimeste palkamiseks. Alternatiivina kasutatakse olemasolevate töötajate väljaõpetamist, kuid paraku osutub see tihtipeale liiga keerukaks ja aeganõudvaks, eriti mittetehnilise taustaga inimeste jaoks. Andmebaasidest andmete otsimine tuleks teha võimalikult lihtsaks. Tekstilise SQL koodi kirjutamise asemel oleks osadel inimestel lihtsam õppida ja kasutada graafilist kasutajaliidesest. Näiteks see [2] õpilaste põhjal läbi viidud uuring näitas, et läbi viidud eksperimendi käigus uuritutest umbes kolmandik eelistas tekstilist programmeerimiskeelt, kolmandik visuaalset keelt ja kolmandikul puudus arvamus. Kuigi konkreetses uuringus ei olnud vanemas vanusegrupis visuaalse keele eelistajaid oleks siiski hea, kui kasutajatel oleks valikuvõimalus.

Lahendus sellise probleemi jaoks oleks kasutada arendusplatvormi, kus kasutaja ei pea ise programmikoodi kirjutama (*no-code development platform*) [3]. Sellised platvormid võimaldavad nii programmeerijatel kui ka mitteprogrammeerijatel luua rakendusi graafiliste kasutajaliideste abil. Taolised platvormid on mõeldud tarkvaraarenduse protsessi kiirendamiseks. Selliste platvormide loomine on hetkel kasvav trend, sest ettevõtete käsutuses olevate digitaalsete andmete hulga kiire kasv ja vajadus teha muutuvatest ärioludest tulenevalt kiireid otsuseid või kiireid muudatusi tarkvaras nõuab kiiresti valmivaid ning paindlikke lahendusi, mis on arusaadavad nii IT-spetsialistidele kui ka mittetehnilise taustaga inimesele. Seega võiks loodavate süsteemide kasutajad ise olla kaasatud endale vajalike rakenduste loomisesse. See on kooskõlas kaasava disaini [4]

põhimõtetega, mille kohaselt peaks süsteemi tulevased kasutajad olema kaasatud süsteemide arendamisse. Samuti peaks pakkuma abi väiksema kogemustepagasiga alles alustavatele arendajatele või huvilistele, kes andmebaasidega esmatutvust teevad.

SQL lausete graafilise koostamise programme on mitmeid, kuid enamikke nendest on võimalik kasutada ainult arvutisse installeerimise teel. Tänapäeva veebilehitsejad ja võrgukiirused on piisavad, et sellised ülesandeid ka veebirakenduse abil lahendada, sest see teeb rakenduse kasutamise mugavamaks, parandab rakenduse kättesaadavust ja võimaldab rakenduses tehtud muudatused koheselt kõigile kasutajatele kättesaadavaks teha. Veebipõhiseid visuaalseid SQL päringute (SELECT lausete) genereerimise rakendusi on vähe ning üldjuhul on need tasulised ja suletud lähtekoodiga. Antud lõputöö käigus arendatakse edasi vabavaralist tarkvara, mis arvestab kasutusmugavust ning on kõigile tasuta kättesaadav.

Käesoleva töö eesmärgiks on edasi arendada PostgreSQL andmebaasides päringute (SELECT lausete) visuaalseks koostamiseks mõeldud avatud lähtekoodiga veebirakendust *Postgres Visual Query Builder*. Ülesandeks on realiseerida uusi funktsionaalsuseid ja ka täiendada graafilist kasutajaliidest. Selleks tuleb leida viisid mitmete SQL andmekäitluskeele lausete konstruktsioonide visuaalseks esitamiseks. Teiseks eesmärgiks on kirjeldada mustrite formaadis visuaalselt andmekäitluskeele lausete koostamist võimaldavate vahendite kasutajaliidese parimaid praktikaid. Lõputöö käigus keskendutakse pigem veebipõhistele lahendustele ning mustritele, kuid see ei välista sarnaste mustrite kasutamist ka arvutisse installeeritavate rakenduste korral.

Magistritöö jätkab Dzotsenidze [5] poolt alustatud tööd. Ta lõi esimese versiooni veebipõhisest graafilise kasutajaliidesega rakendusest SQL SELECT lausete koostamiseks PostgreSQL andmebaaside põhjal. Käesolevas töös nimetatakse seda edaspidi *Postgres Visual Query Builder*. Vastavalt keelte kvaliteedi raamistikule [6] tuleb mõelda läbi visuaalse keele süntaks, semantika ja pragmaatika. Töö aluseks olevas bakalaureusetöös on seda juba jõudumööda tehtud ning kõik käesolevas töös tehtud täiendused peavad samuti selle raamistikuga arvestama.

- Süntaks:
    - Tuleb koostada graafiline esitusviis.

- Semantika:
    - Paikapidavus (*validity*): Tuleb tagada, et kõik keeles koostatud laused oleks võimalik tõlkida mingi andmekäitluse probleemi lahenduseks SQLis.
    - Täielikkus (*completeness*): Tuleb tagada, et kõiki SQLis lahendatavaid andmekäitluse probleeme saaks lahendada ka visuaalses keeles.
- Pragmaatika:
    - Tuleb tagada, et keel oleks hästi hoomatav (arusaadav, õpitav).

Mis puudutab täielikkust, siis käesolevas töös arendatava rakenduse eelmine versioon ei toetanud kõiki võimalusi, mida PostgreSQL SELECT lause päringute kirjutajale pakub [7]. Neid puuduseid nimetatakse Dzotsenidze [5] lõputöös ja tuuakse ka välja järgnevalt. Näiteks ei saanud koostada koondandmete päringutele piiravaid tingimusi (tulemuseks olevas SQL lauses oleks HAVING klausel või koondandmeid leidev alampäring, mille tulemust piiratakse), luua rekursiivseid päringuid, kasutada ühiseid tabeli avaldisi, alampäringuid, aknafunktsioone, tabelite ühendi/ühisosa/vahe leidmise operaatoreid ning 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. Rakendus toetas ainult SELECT lauseid. Edasiarendusena saaks rakendusse lisada näiteks teiste andmekäitluse lausete (UPDATE, DELETE ja INSERT) koostamise võimaluse.

Pragmaatika kohapealt tuleb arvestada sellega, kes on kasutatava rakenduse peamine sihtgrupp. Autori hinnangul on Dzotsenidze poolt loodud rakendus suunatud eeskätt SQLi tundvale (või vähemalt väheselgi määral kokku puutunud) kasutajale. Käesoleva töö puhul püütakse järgida sama joont, et mitte kogu rakendust ümber kirjutada. See tähendab, et töö eesmärgiks *ei ole* luua rakendust, mis on suunatud kasutajale, kes SQLi üldse ei tunne.

Kõige eelneva jaoks oleks vaja luua komponendid ning funktsionaalsus täiendavate konstruktsioonidega SQL lausete genereerimiseks. Käesoleva töö ülesandeks on selliste võimaluste lisamine. Kuna kõige selle tegemiseks vajaliku töö maht ületaks ilmselt magistritöö mahu, siis on vajalik väljalaske planeerimine, et valida töö tulemusena

tehtavad täiendused. Rakendus tuleb luua viisil (puhas kood, laiendatavus), et puuduvaid osi oleks hiljem võimalikult lihtne lisada.

Töös kasutatakse disainiteaduse metoodikat [8], mis tähendab et lähtuvalt kasutajate (antud juhul juhenda ja töö autori) nõuetest realiseeritakse tehniline tehis ehk artefakt, mille headuses veendutakse läbi testkasutajate poolt saadud tagasiside.

Projekti uue versiooni lähtekood on kättesaadav GitHubi hoidlas: https://github.com/rax1110/postgres-visual-query-v2

Peatükis 2 antakse lühiülevaade teoreetilisest taustast ja sarnastest programmidest. Peatükis 3 kirjeldatakse tarkvara uue versiooni loomise protsessi. Peatükis 4 kirjeldatakse lühidalt olemasolevat tarkvara, mida käesolevas töös edasi arendatakse. Peatükis 5 esitatakse hulk olemasolevate süsteemide vaatluse ning olemasolevate kasutajaliidese disainimustrite kohandamise tulemusena koostatud disainimustreid, mis kirjeldavad visuaalselt andmekäitluskeele lausete koostamist võimaldavate keskkondade kasutajaliidese parimaid praktikaid. Peatükis 6 esitatakse nõuded süsteemi uuele versioonile ja nende realiseerimist kirjeldav väljalaske plaan. Peatükis 7 analüüsitakse seda, kuidas töös edasiarendatav programm vastab eelnimetatud mustritele ja kuidas neid mustreid arvestada uue funktsionaalsuse realiseerimisel. Peatükis 8 kirjeldatakse ära programmis tehtud muudatused ning seejärel valideeritakse peatükis 9 tehtud tööd, et veenduda tulemuse headuses ja leida parandamise kohti. Peatüki 10 moodustavas arendusvaates kirjutatakse, millised võiksid olla edasised tarkvara täiendused. Lõpuks esitatakse lühikokkuvõte saavutatud tulemustest.

2 Teoreetiline taust

Selles peatükis antakse ülevaade töö teoreetilisest taustast. Kirjutatakse visuaalsetest programmeerimiskeeltest ja kodeerimisvajaduseta arenduskeskkondadest (sh SQL lausete graafilisest koostamisest), sest edasiarendatav tarkvara pakub võimaluse luua SQL päringuid (sisuliselt andmete otsimise miniprogramme) graafilise kasutajaliidese kaudu. Samuti kirjutatakse väljalaske planeerimisest, sest töö käigus tuleb valida, millised funktsionaalsused sellele tööle vastavas väljalaskes realiseerida.

2.1 Visuaalsed programmeerimiskeeled

Programmeerimiskeeli saab liigitada erinevalt. Näiteks üks võimalik liigitus on see, kas keeles kirja pandud programmis kirjeldatakse programmi täitmiselt soovitud lõpptulemust (deklaratiivsed keeled) (jättes täitmiseks vajalike sammude jada täitjale valida) või kirjeldatakse täpset tegevuste jada (algoritmi), kuidas soovitud tulemuseni jõuda (imperatiivsed keeled). Veel üks võimalik liigitus on see, millisel viisil (graafiliselt *vs.* tekstiliselt) esitatakse inimkasutajale keele elemendid ja nendest kokku pandud programmid (arvutile täitmiseks mõeldud spetsifikatsioonid). Nii võib olla näiteks imperatiivseid visuaalseid keeli (näiteks Scratch [9]) ja deklaratiivseid visuaalseid keeli (näiteks töös käsitletav *Query by Example*).

Visuaalseks programmeerimiskeeleks nimetatakse programmeerimiskeelt, mis võimaldab kasutajal programmeerida graafilises kasutajaliideses visuaalsete elementidega, kasutades traditsioonilisi joonistamise operatsioone ning teksti ja graafiliste sümbolitega töötamist. Sageli on üheks sellist keelt pakkuva programmeerimiskeskkonna funktsionaalsuseks komponentide asukoha muutmine ruumis ja visuaalsete elementide mingil viisil ühendamine. Kogu sellise funktsionaalsuse eesmärk on peita kasutaja eest ära tema jaoks liiga tehnilise (madala) tasemega programmikood (kas mõnes kõrgtaseme programmeerimiskeeles või masinkoodis) ning pakkuda talle selle asemel programmeerimist lihtsustavat liidest, kus saab töötada kõrgemal abstraktsioonitasemel, kasutades graafiliste elementide töötlemist graafilises kasutajaliideses [10].

Visuaalseid programmeerimiskeeli on palju ning neid kasutatakse mitmete erinevate valdkondada jaoks. Tulenevalt keelte omadustest saab neid [11] kohaselt liigitada järgnevalt.

1. Puhtalt visuaalsed keeled (*Purely visual languages*).
2. Teksti ja visuaali hübriidid (*Hybrid text and visual systems*).
3. Programmeerimine näite baasil (*Programming-by-example systems*).
4. Kitsendustele orienteeritud (*Constraint-oriented systems*).
5. Vormipõhised (*Form-based systems*).

Siinkohal tuleb silmas pidada, et välja toodud kategooriad ei ole üksteist välistavad, mis tähendab, et üks visuaalne programmeerimiskeel saab kuuluda ka mitmesse kategooriasse. Lisaks liigitatakse mõnes kirjandusallikas visuaalseid programmeerimiskeeli ka ikoonilisteks (rakenduse käitumist mõjutatakse ikoonide liigutamise, muutmise, defineerimise abil) ja mitteikoonilisteks (tekstipõhisteks).

Puhtalt visuaalsete keelte puhul koostab kasutaja visuaalseid elemente kasutades visuaalse esitusega spetsifikatsiooni, mida silutakse ja mida käivitatakse samas keskkonnas, ilma, et seda vahepeal tekstipõhisesse keelde tõlgitakse. Selliste visuaalsete süsteemide näited on VIPR [12] ja Prograph.

Hübriidsete keelte puhul on võimalus, et programm koostatakse visuaalses keskkonnas ja siis tõlgitakse kõrgtaseme tekstilisse keelde. Teine võimalus on, et üldiselt tekstilises keeles saab kasutada graafilisi elemente. Sellisesse liigitusse kuuluva programmeerimiskeskkonna näiteks on Rehearsal World [13].

Programmeerimine näite baasil lubab süsteemi graafilisi objekte töödeldes „õpetada“, kuidas konkreetset ülesannet täita. Sellise süsteemi näide on Pygmalion [14].

Kitsendustele orienteeritud keele puhul saab programmeerija näiteks modelleerida füüsilisi objekte visuaalses keskkonnas, määrates neile näiteks loodusseadusi jäljendavaid piiranguid. See võimaldab luua simulatsioone pärismaailma reegleid järgides. Sellise süsteemi näide on ARK [15].

Vormipõhine keel kujutab programmeerimist kui omavahel ühendatud lahtrite rühma muutmist. See võimaldab visualiseerida programmi täitmist erinevate lahtrite olekute jadana, mis kulgevad läbi aja. Näide sellisest süsteemist on Forms/3 [16]. Samuti saab selle kategooria alla liigitada vorm-orienteeritud analüüsi jaoks kasutatavat keelt, mis võimaldab süsteeme kirjeldada kasutades kolme tüüpi diagramme [17].

1. Olekumasina mudelil põhinev vormide diagramm (rakenduse seisundit kujutatakse läbi kasutajapoolse päringu esitamise *Submit* ja serveri poolse vastuse *Response*).

2. Hierarhilisel kujul lehtede laadimisel saadetavad teated (andmesõnastik).

3. Andmemudel, mille alusel on võimalik luua andmebaas, mis pakub vajalikke andmeid teadete loomiseks ja salvestamiseks.

Autori hinnangul kuulub edasiarendatavas rakenduses realiseeritav keel, Query By Example, teksti ja visuaali hübriidide kategooriasse.

SQLi ning ka käesolevas rakenduses kasutatava visuaalse keele näol on tegemist valdkonnaspetsiifilise programmeerimiskeelega (vastandina üldotstarbelisele programmeerimiskeelele, mida saab kasutada kõikvõimalike programmeerimisülesannete lahendamiseks). Seega uuris autor valdkonnapõhiste tarkvarakeelte (sh ka modelleerimiskeelte) loojate kogemusi [18].

Uurimise tulemusena saab välja tuua mõned punktid, millele analoogsete rakenduste arendamisel keskenduda, et teiste valdkonnapõhiste tarkvarakeelte loojate vigu vältida.

- Loodav rakendus ei tohiks olla liiga üldine (kasutatav ka mõne muu valdkonna probleemi jaoks) ega ka liiga spetsiifiline (liiga mahukas ja keeruline õppida).

- Eeskätt tuleks mõelda kõige rohkem kasutust leidva funktsionaalsuste realiseerimisele.

- Tuleb välja selgitada, kes on põhilised süsteemi kasutajad. Vastavalt sellele tuleb kasutada kas graafilist või tekstilist esitusviisi või mõlemat.

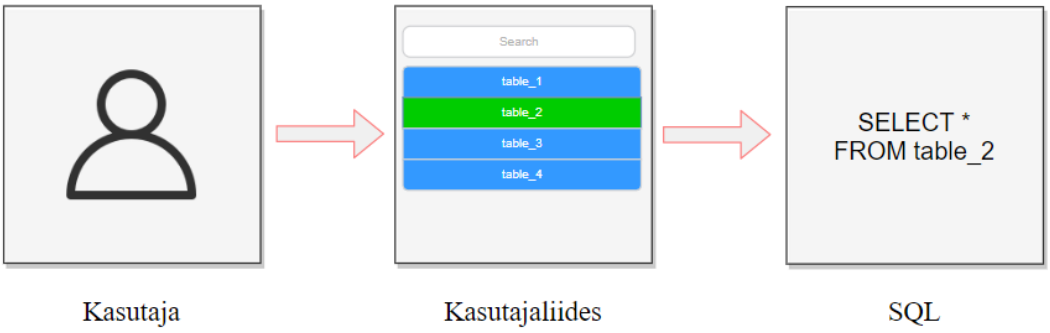
- Mitte ignoreerida reaalseid süsteemi kasutusjuhte (võimaldada komponentide taaskasutust ja eemaldamist).

Samuti on mõistlik järgida valdkonnapõhiste tarkvarakeelte disaini juhiseid [19], mis ütlevad, et valdkonnapõhise keele disaini puhul on oluline saavutada keele lihtsus. Lihtsuse saavutamiseks tuleks disainida ainult olulist funktsionaalsust, piirata keele elementide arvu ja vältida keele kontseptuaalset liiasust (võimalust saavutada samu tulemusi erineval moel).

Käesolevas magistritöös realiseeritakse sisuliselt QBE (*Query By Example*) visuaalne keel [20], milles koostatud laused tõlgitakse SQLi. QBE on andmebaasi päringute tegemise meetod, põhiliselt relatsiooniliste andmebaaside jaoks, mis loodi Moshe Zloofi poolt IBMis (*The International Business Machines Corporation)* 1970. aastatel paralleelselt SQLi arendamisega. QBE üldine idee on, et kasutaja koostab malli selle kohta, millised andmed tuleb andmebaasist leida või milliseid muuta. Enamasti on malli koostamiseks graafiline kasutajaliides, kuid põhimõtteliselt saab seda realiseerida ka teksti töötlemisena nagu seda tehakse objektorienteeritud andmebaasisüsteemi db4o korral [21]. See on ühtlasi näide, et QBE idee on üldisem kui SQL või relatsiooniline mudel ja seda saab kasutada erinevate andmemudelite ja andmebaasikeelte korral.

QBE põhimõte seisneb selles, et kasutaja saab lihtsa ja arusaadava graafilise või tekstilise liidese abil andmebaasi andmeid kuvada ja töödelda (Joonis 1). Tegelikult on QBE pelgalt abstraktsioon kasutaja ja tekstilise andmebaasikeele vahel. Taustal muudetakse visuaalselt tehtud valikud, ehk andmebaasi mõistes kasutaja päringud, andmebaasikeele (antud juhul SQL) lauseteks ja andmebaasiga suhtlus ning andmete kasutamine toimub tegelikult läbi selle programmeerimiskeele [22].

SQL on QBE väljamõtlemise ajast palju edasiarenenud [23] ja seega on üheks käesoleva töö väljakutseks leida sobivad viisid SQLi uuemate konstruktsioonide QBE keskkonnas väljendamiseks.



Joonis 1. Abstraktne näide kasutaja suhtlusest süsteemiga QBE keskkonnas.

2.2 Kodeerimisvajaduseta arenduskeskkond

Kodeerimisvajaduseta arenduskeskkond (*No-code development platform)* võimaldab programmeerijatel ja mitteprogrammeerijatel luua tarkvara traditsioonilise lähtekoodi teksti töötlemise asemel graafiliste kasutajaliideste (sh visuaalsete tekstilahtrite või valikute) abil. Eeldusel, et kodeerimisvajaduseta platvorm on veebipõhine ja keegi on selle üles seadnud, ei ole seda platvormi kasutades vaja ise keerulist töökeskonda seadistada ega lisatarkvara installeerida. See annab võimaluse tarkvara senisest kiiremini ja lihtsamini luua. Sellised arendusplatvormid on tihedalt seotud madala kodeerimisvajadusega arendusplatvormidega (*Low-code development platform)*, sest mõlema lahenduse puhul on võimalik rakendusi kiiremini luua, omamata (vähemalt ideaalis) spetsiaalset tarkvaarenduse alast väljaõpet või põhjalikke teadmisi sellest [3]. Siiski tuleb silmas pidada, et madala kodeerimisvajadusega arendusplatvormide puhul võib olla oluline ja vajalik mõningane arusaam tarkvaraarendusest, tulenevalt kasutatavast platvormist ja rakenduse eripärast.

Vähese kodeerimise idee pärineb neljanda generatsiooni programmeerimiskeelte ja kiirprogrammeerimise keskondade tööriistadest ning esmakordselt tutvustati seda ideed 2011. aastal. Tehnoloogilised lähenemised/paradigmad, mis viisid madala kodeerimise mõiste loomiseni on järgmised [24].

- Mudelipõhine tarkvaraarendus.
- Kiirprogrammeerimine.
- Automaatne koodi genereerimine.
- Visuaalne programmeerimine.

Ärilise ning tööstusliku poole pealt kasvab kodeerimisvajaduseta platvormide populaarsus kiirelt, kuna ettevõtted peavad tulema toime kaadri voolavuse ning heade oskustega tarkvaraarendajate puudusega. Samuti tuleb konkurentidega sammu pidada ja arvestada klientide soovide ning (äri- ja seadusandliku) keskkonna muudatustega, mis samuti nõuab pidevaid muutusi tarkvaras [3]. Teisest küljest on selliste platvormide kasutamine abiks ka alustavatele arendajatele või huvilistele, kellel on programmeerimisega vähe kokkupuude olnud, kuid kes soovivad katsetada programmeerimise võimalusi või avastada enda jaoks midagi uut.

Madala kodeerimisvajadusega keskkonnad keskenduvad üldjoontes järgnevatele valdkondadele [24].

- Andmebaasirakendused.
- Äriprotsesside automatiseerimine.
- Kasutajaliidesed (veebipõhised rakendused).

Kodeerimisvajaduseta arenduskeskkonnad pakuvad traditsiooniliste arenduskeskkondadega võrreldes küll eeliseid, kuid need toovad ka kaasa riske [25].

Eelistest ja nende riskidest saab välja tuua järgmised näited [26].

- Suurem paindlikkus – tänu visuaalsetele elementidele on tarkvara loomine kiirem. Valkonna jaoks parimat tarkvara oskaksid luua valdkonna eksperdid, kes aga ei pruugi olla IT spetsialistid. Nendel võiks olla sellistest arenduskeskkondadest abi. Kuid kui tarkvara loob töötaja, kellel puuduvad konkreetse valdkonna (mille jaoks tarkvara luuakse ning ka programmeerimise valdkonna) teadmised, siis paratamatult võib ta luua tarkvara, mis tegelikult ei tee seda, mida see oli mõeldud tegema või teeb seda ebakorrektselt.

- Vähendatud kulud – kuna arendajate tööjõukulud on kõrged, siis saab raha kokku hoides olemasoleva töötaja vajalikku tarkvara ise looma panna. Kui rakendust loob vastava kogemuseta töötaja, siis lõpptulemusena võib see tuua ettevõttele kasu asemel kahju. Näiteks võib see juhtuda kui loodav tarkvara ei tööta korrektselt või töötaja kulutab kogu aja selle loomisele ning tavapärased tööülesanded jäävad täitmata.

- Suurem tootlikkus – kuna kodeerimisvajaduseta keskkonnas saab tarkvara luua kiiremas tempos, siis pole ettevõte IT-töötajad ülekoormatud kõigi teiste osakondade probleemidega. Siin kehtib sama risk, mis vähendatud kulude puhul.

- Kergesti muudetav – kui väljatöötatud tarkvaras on vaja midagi muuta, siis tuleb muuta vaid visuaalsete komponentide loogikat või asetust, mille teostamine võiks olla kiirem, kui traditsioonilise programmeerimise puhul. Ka väikese muudatusega võib tarkvara tegelik funktsionaalsus muutuda. Väljaõppeta töötaja ei oska selliseid vigu alati ette näha. Samuti, kui on vaja tarkvaras realiseerida mõnda ebastandardset või spetsiifilist lahendust, siis ei pruugi läbi madala kodeerimise keskkonna see võimalik olla.

- Erinevatele inimestele sobivad erinevad tööviisid ja vahendid. Kui arendamiseks saab kasutada vaid vähese koodi kirjutamisega graafilist keskkonda, siis pärsib see nende inimeste tööd, kes eelistaksid kirjutada koodi tekstiliselt (sh võiks olla valikuvabadus).

Ameerika turu-uuringute ettevõte Forrester [27], mis annab nõu tehnoloogia olemasoleva ja võimaliku mõju kohta, on ennustanud, et madala kodeerimisvajadusega arendusplatvormide turg kasvab 2022. aastaks 21,2 miljardi dollarini, võrreldes 2017. aasta 3,8 miljardi dollariga [26].

2.3 SQL lausete graafiline koostamine

Selles peatükis kirjeldatakse, kuidas saab SQL lauseid graafiliselt koostada. Muuhulgas nimetatakse ka programme, mis seda võimaldavad ja antakse lühiülevaade nende funktsionaalsusest ning kasutajaliidese ülesehitusest.

SQL lausete graafiliseks koostamiseks on erinevaid võimalusi. Üldjuhul sõltub see konkreetsest programmist ja selle ülesehitusest. Kogu suhtlus programmi ja kasutaja vahel toimub läbi graafilise liidese (visuaalsed objektid/komponendid, tekstilahtrid). Näiteks põhilisteks tabelite valimise viisideks (SELECT lause koostamisel) on pukseerimise (*Drag and Drop)* meetod, samuti kasutatakse lihtsalt hiirega tabelil klõpsamist. Ühendamisoperatsioonide (*join*) kirjeldamiseks saab kasutada joontega

ühendamist või vastavate veergude märgistamist. Erinevate filtrite või lisatingimuste määramiseks saab kasutada rippmenüüd (*dropdown*), raadio-nuppe (*radio button*) või märkeruute (*checkbox)*.

Nagu eelnevalt mainitud, on visuaalseid SQL lausete koostamise rakendusi küllaltki palju. Probleemiks on see, et enamjaolt on need tasulised või kasutatavad ainult arvutisse installeerimise teel. Kuna antud lõputöö keskendub veebipõhise päringute koostamise rakenduse edasiarendamisele, siis vaadeldakse ka selle lõputöö puhul ainult veebipõhiseid lahendusi. Rakenduste valikul lähtuti sellest, et rakendus oleks tasuta kättesaadav või veebipõhise ajutise demoversiooniga. Võrdlusmomendi huvides lisati võrdlusesse ka Dzotsenidze poolt loodud tarkvara [4], mis on käesoleva projekti sisendiks. Veel detailsemalt käsitletakse eelmise autori poolt loodud tarkvara puudujääke ja positiivseid külgi peatükis 4.

Iga väljatoodud rakenduse puhul tuuakse lisaks lühitutvustusele välja ka rakenduse poolt toetatud andmebaasisüsteemid ja ekraanipildid. Täpsem võrdlustabel, mis vastab alljärgnevatele punktidele, on välja toodud selle jaotise lõpus (Tabel 1). Tabeli pealkirjas on sulgudes vaadeldud programmi versioon.

- **Toetatud andmebaasisüsteemid (arvuliselt):** Andmebaasisüsteemide arv, mida rakendus toetab.
- **Toetatud SQL lausete tüübid:** SQL lausete tüübid, mille koostamist rakendus toetab.
- **Esituse tüüp:** Rakenduse esituse tüüp (visuaalne, tekstiline või hübriid).
- **Kuvab SQLi tõlkimise tulemust:** Jah/Ei vastus küsimusele, kas rakendus näitab genereeritud SQL lauset?
- **Võimaldab kirjutada tekstilise SQLi ja seda käivitada:** Jah/Ei vastus küsimusele, kas rakendusel on olemas tekstiredaktor, milles tekstilist SQL koodi kirjutada ja käivitada?
- **Võimaldab graafiliselt koostatud lauset enne käivitamist muuta:** Jah/Ei vastus küsimusele, kas visuaalsete komponentide abil koostatud SQLi lauset on võimalik enne käivitamist tekstiredaktoris muuta?
- **Toetab kahesuunalist esituse loomist:** Jah/Ei vastus küsimusele, kas rakendus võimaldab SQL lause kirjutamise järel genereerida selle visuaalse esituse?

- **Võimalus kuvada tulemusi graafikul:** Jah/Ei vastus küsimusele, kas rakendus võimaldab vaadata tulemusi graafiku kujul?
- **Kas on tasuta:** Jah/Ei?
- **Kas on avatud lähtekoodiga:** Jah/Ei?

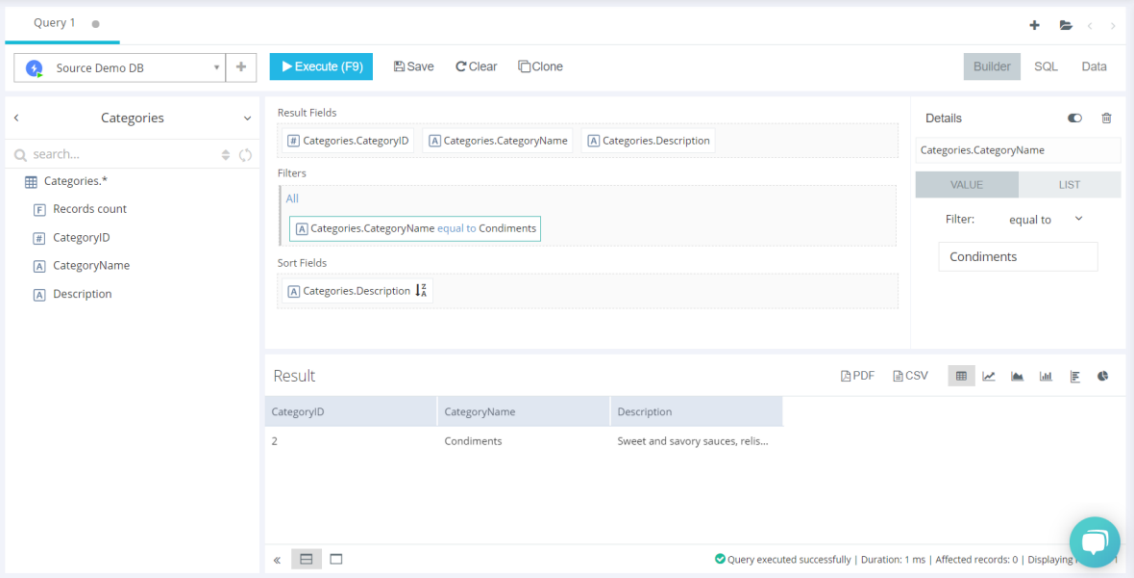
Tabel 1. Graafiliste SQL lausete koostamise rakenduste võrdlustabel.

	Skyvia (2020.4.22)	**Active (3.6.1.401)**	**Dev Express (v2020 vol 1.3)**	**Datapine (2020)**	**Apex (4.2.0.00.27)**	**Postgres Visual Query Builder (1)**
Toetatud andmebaasisüsteemid (arvuliselt)	7	30	17	13	1	1
Toetatud SQL lausete tüübid	SELECT, INSERT, UPDATE, DELETE, DLL. (graafiliselt ainult SELECT)	SELECT	SELECT	SELECT	SELECT	SELECT
Esituse tüüp	Hübriid	Hübriid	Hübriid	Hübriid	Hübriid	Hübriid
Kuvab SQLi tõlkimise tulemust	**Jah**	**Jah**	**Jah**	Ei	**Jah**	**Jah**
Võimaldab kirjutada tekstilise SQLi ja seda käivitada	**Jah**	**Jah**	Ei	**Jah**	Ei	Ei
Võimaldab graafiliselt koostatud lauset enne käivitamist muuta	**Jah**	**Jah**	Ei	Ei	Ei	Ei
Toetab kahesuunalist päringute loomist	Ei	**Jah**	Ei	Ei	Ei	Ei
Võimalus kuvada tulemusi graafikul	**Jah**	Ei	Ei	**Jah**	Ei	Ei
Pakutakse tasuta	Ei	Ei	Ei	Ei	Ei	**Jah**
On avatud lähtekoodiga	Ei	Ei	Ei	Ei	Ei	**Jah**

2.2.1 Skyvia Query Builder

Skyvia Query Builder (Joonis 2, Joonis 3) on veebipõhine SQLi andmekäitluskeele lausete koostamise tööriist. [28] Vahend võimaldab teha andmekäitlust nii pilvepõhiste andmebaaside kui ka kasutaja enda serverites olevate andmebaaside põhjal.

Toetatud andmebaasisüsteemid: Amazon Redshift, Google BigQuery, MySQL, Oracle, PostgreSQL, SQL Data Warehouse, SQL Server.



Joonis 2. Rakenduse *Skyvia Query Builder* päringu loomise vaade.

```
SELECT t.CategoryID, t.CategoryName, t.Description
  FROM Categories AS t
WHERE (t.CategoryName = 'Condiments')
ORDER BY t.Description DESC
```

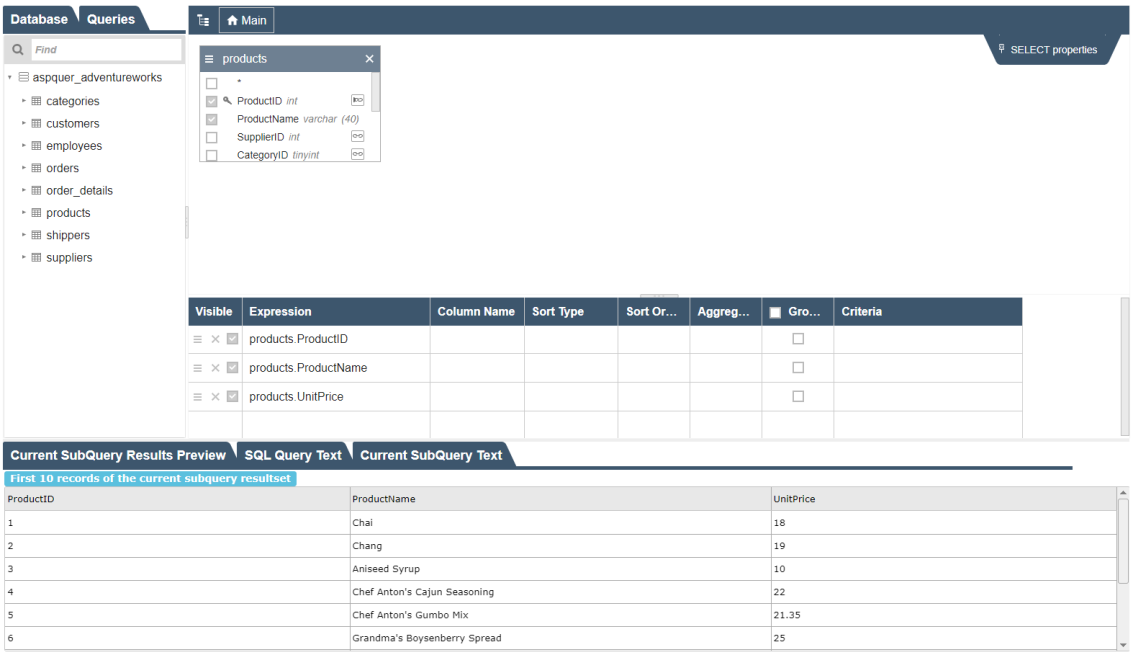
Joonis 3. Rakenduses *Skyvia Query Builder* loodud päringu SQLi tõlkimise tulemus.

2.2.2 Active Query Builder

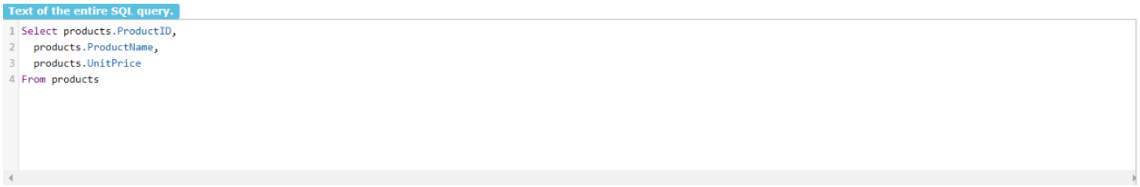
Active Query Builder (Joonis 4, Joonis 5) on graafiliste päringute koostamise komponentide komplekt, mis võimaldab lõppkasutajatel luua SQL päringuid visuaalse liidese kaudu. Antud rakenduse programmiliides on kujul, mis tuleb kõigepealt vastava programmeerimiskeelega siduda. Olemas on ka veebipõhine näidisrakendus. [29].

Toetatud andmebaasisüsteemid: Oracle, MS SQL Server,MS Azure, MySQL (MariaDB), PostgreSQL (Redshift), SQLite, MS Access, InterBase, Firebird, IBM DB2

(incl. AS/400), Informix, SAP IQ, SAP SQL Anywhere, SAP ASE, SAP Hana, SAP Advantage DB, Netezza, Teradata, Pervasive (Actian Zen), Vista DB, Nexus, Elevate DB, Vertica, Progress (OpenEdge), Apache Spark, Amazon Aurora, Amazon Athena, Google Big Query, Cassandra, Impala.



Joonis 4. Rakenduse *Active Query Builder* päringu loomise vaade.



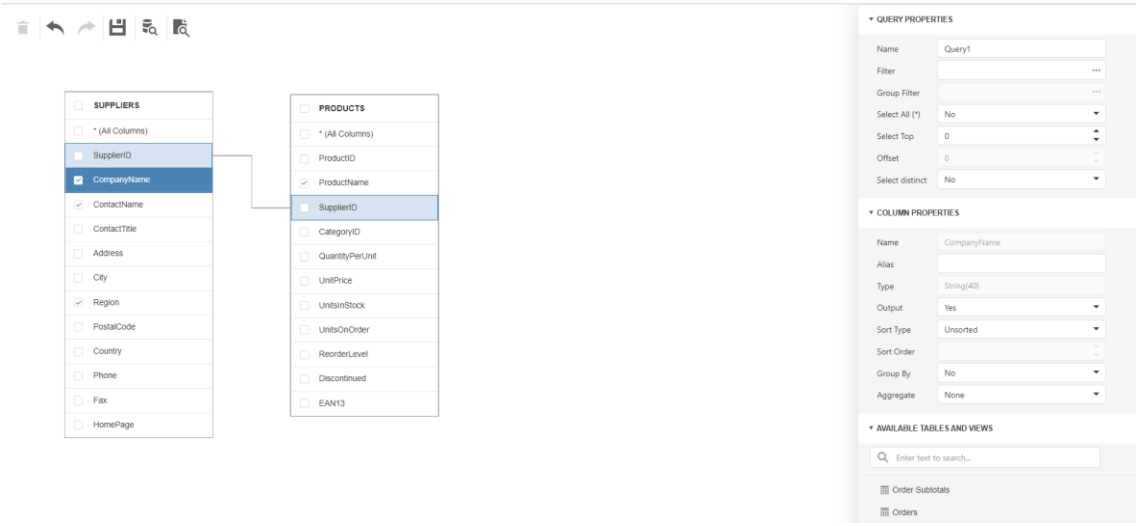
Joonis 5. Rakenduses *Active Query Builder* loodud lause SQLi tõlkimise tulemus.

2.2.3 DevExpress Query Builder

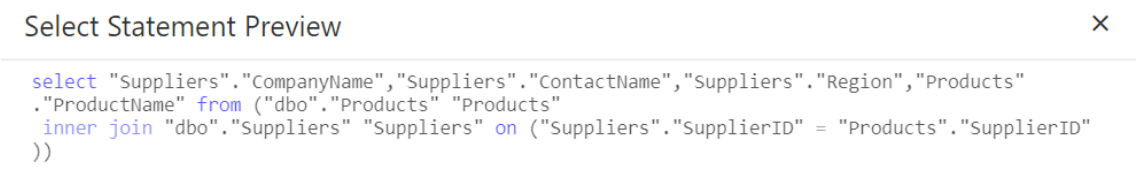
DevExpress Query Builder (Joonis 6, Joonis 7) on ASP.NET baasil ehitatud lihtsamat laadi graafiliste päringute koostamise moodul. Moodulil on veebipõhine näidisrakendus. Seda moodulit saab kasutada ainult läbi ASP.NET raamistiku ning see on mõeldud olemasoleva rakendusega sidumiseks [30].

Toetatud andmebaasisüsteemid: MS SQL Server, MS Access, MS SQL Server CE, Oracle, Amazon Redshift, Google Big Query, Teradata, SAP Advantage, SAP ASE, SAP

SQL Anywhere, IBM DB2, Firebird, MySQL, Pervasive PSQL, PostgreSQL, Vista DB, SQLite.



Joonis 6. Rakenduse *DevExpress Query Builder* päringu loomise vaade.

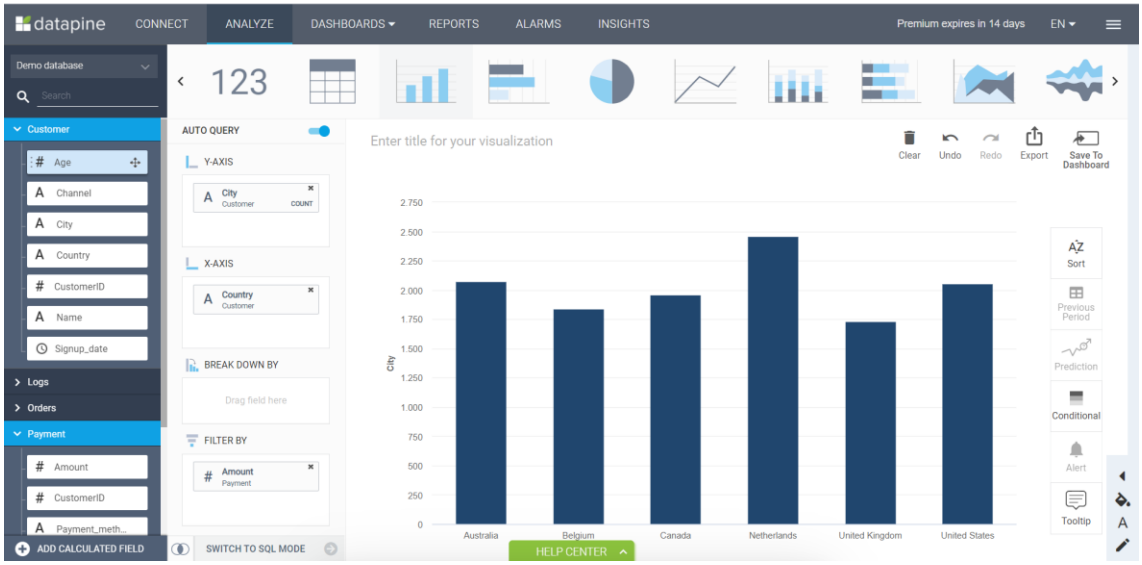


Joonis 7. Rakenduses *DevExpress Query Builder* loodud lause SQLi tõlkimise tulemus.

2.2.4 Datapine Query Builder

Datapine Query Builder (Joonis 8) on veebipõhine andmebaasi päringute ning analüüsi platvorm, mis lubab teha visuaalsete komponentide abil andmebaasipäringuid ning genereerib andmete põhjal ka graafikuid [31].

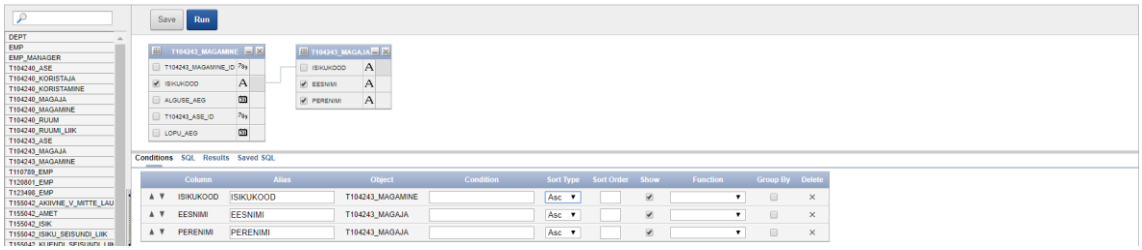
Toetatud andmebaasisüsteemid: MySQL, Oracle, PostgreSQL, AWS, Amazon Aurora, Amazon S3, Amazon Redshift, Google Cloud SQL, MS SQL Server, SAP Hana, MariaDB, MS Azure, Percona.



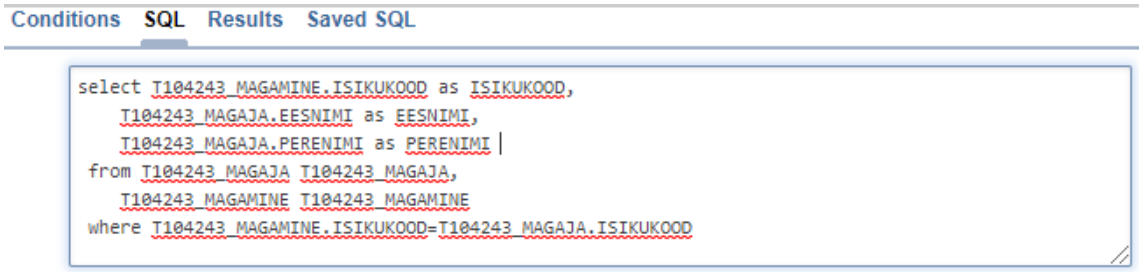
Joonis 8. Rakenduse *Datapine Query Builder* päringu loomise vaade.

2.2.5 Oracle Application Express (APEX) Query Builder

APEX Query Builder (Joonis 9, Joonis 10) on graafilise kasutajaliidesega SQL päringute loomise rakendus, mis kuulub Oracle Application Express veebirakenduste kiirprogrammeerimise keskkonna koosseisu [32].



Joonis 9. Rakenduse *Oracle APEX* päringu loomise vaade.



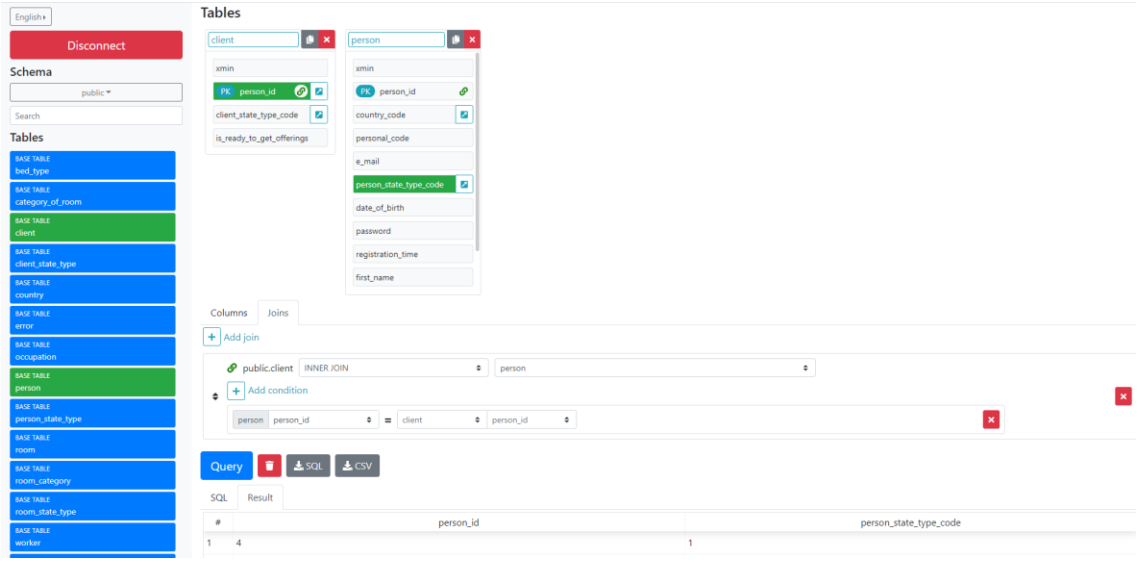
Joonis 10. Rakenduses *Oracle APEX* loodud lause SQLi tõlkimise tulemus.

2.2.6 Postgres Visual Query Builder

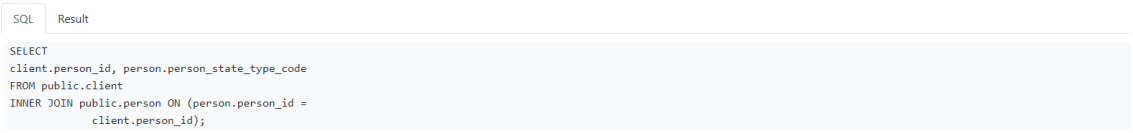
Postgres Visual Query Builder (Joonis 11, Joonis 12) on graafiline päringute tegemise keskkond PostgreSQL andmebaaside jaoks [4]. Rakendust on võimalik kasutada läbi

veebilehitseja ning ühenduda kasutaja poolt sisestatud andmeid kasutades vajaliku andmebaasi külge.

Toetatud andmebaasisüsteemid: PostgreSQL.



Joonis 11. Rakenduse *Postgres Visual Query Builder* päringu loomise vaade.



Joonis 12. Rakenduses *Postgres Visual Query Builder* koostatud lause SQLi tõlkimise tulemus.

2.4 Väljalaske planeerimise meetod

Selles lõputöös kasutatakse agiilset väljalaske planeerimist. Agiilne väljalaske planeerimine on tarkvaraarenduses üks lähenemisviisidest tootehaldusele. Selline lähenemisviis paneb põhirõhu kohanemisvõimelisele planeerimisele, evolutsioonilisele arengule, kiirele väljastamisele ning pidevale täiustamisele [33], võttes seehulgas arvesse tarkvara ärilisi väärtuseid ja paindlikkust. Selle asemel, et proovida arendada kõiki pakutud funktsioone ühes suures, korrapärases projektis, jagab paindlik planeerimine arendusprotsessi etappideks, mida nimetatakse väljalaseteks. Tänu sellele suudavad projektijuhid töövoogu paremini hallata ja anda kliendile pidevalt ülevaadet loodavast tootest [34].

Antud lõputöö puhul kasutatakse T. Normani poolt välja pakutud agiilset väljalaske planeerimise meetodit, mis on kirjeldatud artiklis „Agile Release Planning 101“ [35]. Selles artiklis kirjeldatud meetod on ideelt sarnane *Scrum'*i kasutamisele [36]. Antud meetod valiti:

- selle realiseerimise lihtsuse poolest,
- seetõttu et autor arendab üksinda ja seega pole funktsionaalsuse keerukuse hinnanguid vaja erinevate arendajate vahel kokku leppida,
- autori varasema kokkupuute tõttu *Scrum* metoodikaga.

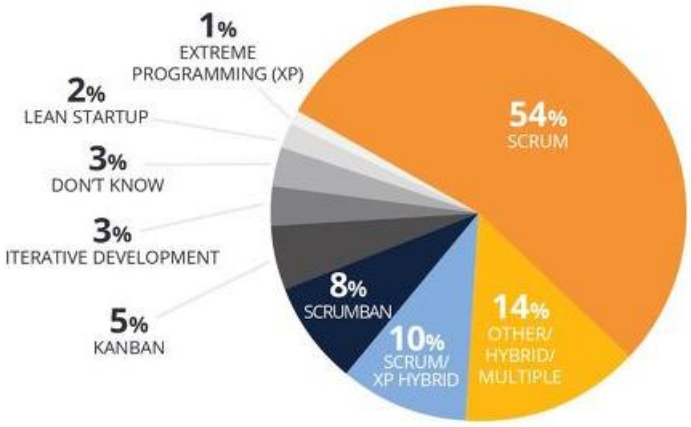
Lisaks sellele on ka teisi väljalaske planeerimise meetodeid, nagu näiteks geneetilisel algoritmil põhinev *EVOLVE*, mis valib algoritmiliselt igasse iteratsiooni optimaalsed ülesanded/nõuded (prioritiseerimine põhineb algoritmil) [37]. Teine näide on *Release Iteration Planning* [38], kus on suur rõhk koostööl. Planeerimisel osalevad kõik arendusprojekti erinevad tiimid (näiteks tootehalduse tiim, eesrakenduse tiim, tagarakenduse tiim või mõni muu). Iga tiimi toote omanik või juht prioritiseerib ülesanded ning seejärel viib oma tiimiliikmed ülesannetega kurssi. Seejärel koostab iga tiim kasutuslood ja hinnangud ning paneb paika järgmise iteratsiooni. Hiljem osalevad tiimi esindajad/tiimijuhid ühisel koosolekul ning presenteerivad tiimi poolt loodud plaani ja kui on murekohti, siis leitakse nendele ühiselt lahendused. Pärast lahenduste ja lõppotsuste tegemist alustab iga tiim oma plaani realiseerimist. Samuti on olemas *Continuous Release Planning* [38], kus keskendutakse rohkem pidevale arenduse jätkule, kui optimaalsematele, prioriteetsematele ülesannetele. Otsuseid tehakse pigem arendusressursside võimekuste põhjal. See meetod põhineb erinevatel otsustusvoorudel, kus tooteomanikud esitlevad oma ideid portfelli juhtidele (otsustusgrupp). Portfelli juhtidel on võimalik idee tagasi või edasi lükata või aktsepteerida. Kui idee jõuab järgmisesse vooru, siis seda täiendatakse, mille põhjal teeb otsustusgrupp uue otsuse. Kui idee läbib portfelli juhtide erinevad etapid, siis jõuab idee arendus- ja turundustiimini ja sealt edasi realiseerimiseni.

Eelnevalt kirjeldatud meetodid eeldavad üldjuhul suurema meeskonna olemasolu. Seetõttu otsustati ka T. Normani poolt pakutud meetodi kasuks.

3 Arendusprotsess

Selles peatükis kirjeldatakse, kuidas olemasolevat tarkvara edasi arendati ning kuidas arenduses ellu viidud ideedeni jõuti. Samuti tuuakse välja kasutusele võetud uued vahendid/tööriistad. Realiseeritud funktsionaalseid ning mittefunktsionaalseid nõudeid kirjeldatakse peatükis 6.

Agiilseid arendusmetoodikaid on mitmeid, nagu näiteks *Scrum* [36], *XP* (ekstreemprogrammeerimine) [39] ja *Kanban* [40]. Käesoleva arenduse käigus täpset *Scrum* meetodit ei järgitud (st ei olnud päevaseid koosolekuid või eraldi meeskonda), aga järgiti paljusid *Scrum* põhimõtteid (*sprindid,* juhendajaga koosolekud ning plaani ning tööde ülevaatamised), mis võiks kokkuvõttes vastata hübriidsele lähenemisele (alloleval joonisel 14%). *Scrum* metoodika on ka 2019. aasta *State of Agile* raporti [41] küsitluste tulemuste põhjal ettevõtetes enim kasutatav agiilne metoodika. Sellele järgnevad erinevad hübriidid eelnevalt välja toodud metoodikatest (Joonis 13).



Joonis 13. Ettevõtetes enim levinud agiilsed arendusmetoodikad [41].

Arendamisel järgiti ka paindmetoodikate parimaid praktikaid. Sellest tulenevalt arvestati suletud akna reegliga (*Closed-window rule)* [42], mis keelab käimasoleva sprindi perioodil (aken) uute ülesannete lisamist või muutmist. Küll aga lubab see väljalaske plaani pärast iga iteratsiooni uuesti üle vaadata. Samuti kasutati ajakarbi (*Timebox)* meetodit [43], mis tähendab, et igale planeeritud tegevusele oli eraldatud kindel

ajavahemik. Konkreetse tegevusega töötati selle jaoks määratud aja jooksul ning lõpetati töötamine, kui aeg oli täis. Seejärel hinnati, kas kavandatud eesmärgid olid saavutatud. Kui mõnda ülesannet ei jõutud iteratsiooni jooksul täita, siis ei muudetud iteratsiooni pikkust, vaid tõsteti see ülesanne mõnesse järgmisesse iteratsiooni.

3.1 Ideed funktsionaalsuste realiseerimiseks

Esialgselt tulid ideed, mida võiks *Postgres Visual Query Builder* rakenduse edasiarendamisel realiseerida, eelmise autori ning juhendaja poolt välja toodud mõtetest. Sellest tulenevalt hakkas autor uurima ka sarnaseid olemasolevaid programme ning nende poolt pakutavaid võimalusi, et näha kuidas teised loojad on selliseid funktsionaalsusi realiseerinud. Erinevate rakenduste võrdlemine andis küll parema arusaama, milliseid funktsionaalsuseid antud rakendused võiksid toetada, aga *kõigi* nende võimaluste realiseerimine polnud töö mahtu arvestades selle töö eesmärk. Siiski sai autor tänu nende rakenduste kasutajaliideste uurimisele mõningad ideed.

Arendamisel jälgiti põhimõtet, et ei hakata kogu rakendust otsast peale tegema (kuigi arenduse käigus tuli see ka aeg-ajalt mõttesse). Sellest tulenevalt tuli püsida etteantud raamides ja prooviti ära kasutada võimalikult palju olemasolevaid komponente ja lahendusi.

Mittefunktsionaalsed nõuded tekkisid autori poolt pärast esialgset rakendusega tutvumist, mille käigus avastati mõningad detailid/vead (jaotis 4.4), mis vajasid parandamist või muutmist.

3.2 Kasutatud töövahendid

Selles jaotises tuuakse välja arenduse käigus kasutusele võetud uued raamistikud ja teegid ning samuti olemasolevate uuendused.

3.2.1 Tagarakenduse vahendid

Koodivigade ja stiili parandamiseks võeti kasutusele eslint-config-airbnb-standard (versioon 3.1.0) [44]. Uusimate ECMAScript võimaluste kasutamiseks paigaldati JavaScripti kompilaator Babel (versioon 7.8.6) [45].

3.2.2 Eesrakenduse vahendid

Reacti koodivigade ja stiili parandamiseks võeti kasutusele eslint-plugin-react (versioon 7.19.0) [46]. SQLi kirjutamise võimaldamiseks võeti kasutusele tekstiredaktor react-codemirror2 (versioon 5.52.2) [47]. Uuendati React teeki (versioon 16.8.4 → 16.8.5), React Redux teeki (versioon 6.0.1 → 7.2.0) ning React Scripts teeki (2.1.8 → 3.4.1).

4 Tarkvara olukord enne arendust

Selleks, et luua andmebaasipäringuid *Postgres Visual Query Builder* keskkonnas, ei ole vaja teada SQL süntaksi – päringute loomine käib läbi visuaalsete komponentide ning SQL laused genereeritakse kasutaja eest automaatselt, mis on koheselt nähtavad. Küll aga peaks kasutajal olema üldine ettekujutus SQLi võimalustest ning sellest, milliseid konstruktsioone seal saab päringu koostamisel kasutada ja kuidas need omavahel seostuvad (st autori arvates pole tegemist keskkonnaga, mis oleks väga mugav kasutajale, kes pole üldse SQLiga tuttav). Päringu tulemus esitatakse tabelina.

Järgnevalt kirjeldatakse *Postgres Visual Query Builder* põhilist funktsionaalsust, kasutajaliidest ning süsteemi arhitektuuri. Kogu järgnev tekst selles peatükis kirjeldab ainult selle programmi esimest versiooni, mille autoriks oli Erik Dzotsenidze ja ei kirjelda selle lõputöö tulemusena lisatud funktsionaalsust ega kasutajaliidese muudatusi.

4.1 Funktsionaalsus

Järgnevalt kirjeldatakse lühidalt rakenduse *Postgres Visual Query Builder* funktsionaalsust [5].

Rakendus võimaldab ühenduda endale sobiva PostgreSQL andmebaasiga (millele ühendujale peab olema antud juurdepääs). Rakendus võimaldab graafilise liidese abil SELECT lausete koostamist. Pärast andmebaasiga ühendumist on võimalik vahetada andmebaasi skeemi, et näha teiste skeemide tabeleid. Lisaks pakub rakendus tabelite otsimise võimalust ning näitab ära tabelite tüübid. Rakendus lubab tabeleid omavahel päringu kontekstis ühendada (realiseerida ühendamise operatsiooni) ning määrata nii tabelitele kui ka veergudele varjunimesid e aliasi. Samuti näidatakse ära välisvõtmed, sest sageli ühendatakse tabeleid üle välisvõtme veergude. Rakendus võimaldab päringut tabeli veergude põhjal grupeerida, sorteerida, kasutada päringus reataseme funktsioone ja tulemuses ridu mingi tingimuse alusel piirata. Lisaks on kasutajal võimalus genereeritud SQL kood .SQL või .CSV (*Comma Separated Values*) failina arvutisse salvestada.

4.2 Kasutajaliides

Postgres Visual Query Builder (Joonis 11, Joonis 12) kasutajaliides on värvikasutuse mõttes küllaltki kirju. Üldine taustavärv on küll valge, kuid erinevate värvikombinatsioonidega on esile toodud vastavaid tegevusi/komponente. Kasutajaliidese ülesehituse puhul on järgitud Oracle Application Express (APEX) alamprogrammi Query Builder.

4.3 Tehniline arhitektuur

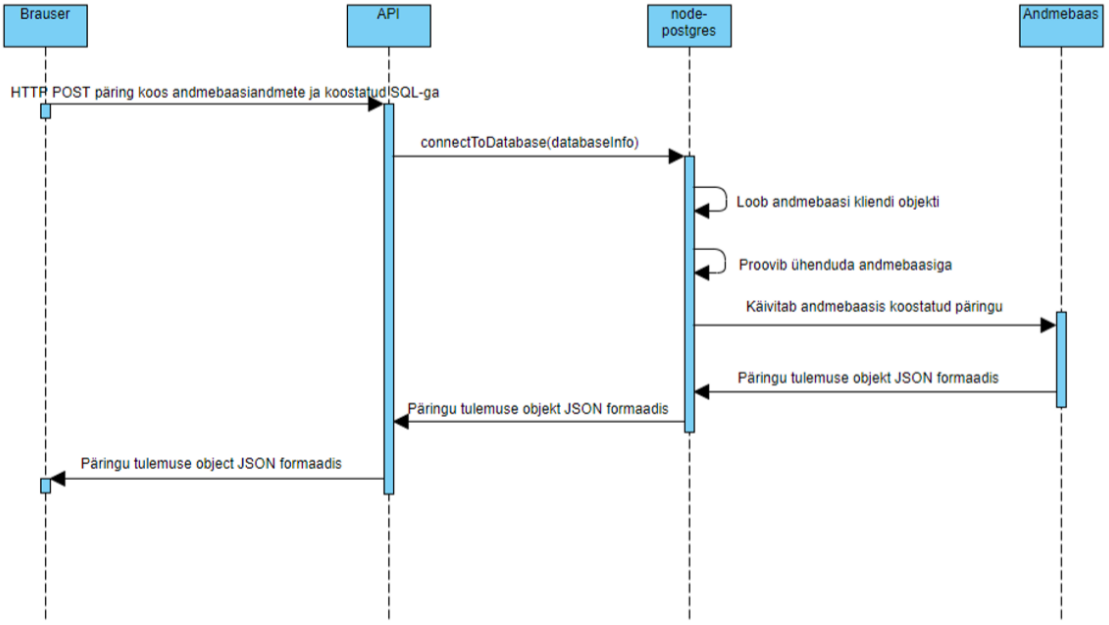
Käesolev jaotis kirjeldab lühidalt olemasoleva rakenduse arhitektuuri (Joonis 14). Tehnilises mõttes jaguneb rakendus kaheks osaks, eesrakendus (*front-end)* ja tagarakendus (*back-end)*. Süsteemis kasutatakse klient-server mudelit (*client-server model).*

3.3.1 Tagarakenduse vahendid

Tagarakenduse realiseerimiseks on kasutatud Node.js [48], mis on JavaScripti [49] käituskeskkond (*runtime environment)*. Rakendusliideste (*Application Programming Interface* e API) loomiseks on kasutatud Express [50] raamistikku. Andmebaasiga suhtlus käib läbi node-postgres [51] moodulite.

3.3.2 Eesrakenduse vahendid

Eesrakenduse realiseerimiseks on kasutatud JavaScripti teeki React [52], kus rakenduse olekut hallatakse läbi Redux [53] teegi. Kasutajaliidese komponentide disaini jaoks on kasutusel raamistik Bootstrap [54]. Tagarakendusega suhtlemiseks kasutatakse veebilehitseja HTTP (*Hypertext Transfer Protocol)* klienti axios [55] ning SQL lausete loomiseks veebilehitseja poolel on kasutatud Squel.js teeki [56].



Joonis 14. Ees-ja tagarakenduse suhtlust kirjeldav jadadiagramm [5].

4.4 Positiivsed ja negatiivsed küljed

Tagarakenduse puhul on positiivne see, et kasutatakse modulaarset marsruutimise struktuuri. See tähendab, et kõik rakendusliidese sõlmede alamteed (*sub paths)* pannakse kokku mooduliteks, ning paigutatakse nende jaoks eraldatud põhiteede külge. Selline lähenemine tagab selgema arusaamise, kuidas on lõppsõlmede päringud omavahel seotud ja lihtsustab edaspidist arendust. Lisaks on tagarakenduse üldine kaustade struktuur hea.

Miinusena saab välja tuua, et hetkel luuakse tagarakenduses iga POST-päringu korral uus andmebaasiühendus, mis ei ole mõistlik ei ressursi ega koodistiili poolest. Mõistlikum oleks teha andmebaasiga ühendamiseks eraldi klassi meetod, mis kasutab vastavat konfiguratsiooni ja loob ühenduse ainult ühel korral ning hoiab seda aktiivsena. Samuti, kui on ühendus olemas ja on teada ühenduse parameetrid, siis saab andmebaasis andmete küsimiseks kolme POST-päringu asemel teha ühe GET-päringu.

Samuti kasutatakse hetkel tagarakenduses vananenud moodulite importimise ja eksportimise lahendusi (Joonis 15) ning ka asünkroonseid meetodeid. Tänapäeval võiks Node.js'is kirjutatud rakendus kasutada juba vähemalt ES6 (*ECMAScript 6*) [57] spetsifikatsiooni poolt pakutavaid võimalusi ja järgida parimaid praktikaid [58], sh puhta koodi põhimõtteid [59]. Lisaks on koodistiil kohati lohakas – palju liigseid reavahesid ja

impordid ja ekspordid on igalpool laiali, puudub korrektne koodi struktuur (Joonis 16). Mõistlik oleks siinkohal rakendada *linter'*it (tööriist programmikoodi analüüsimiseks), mis hoiatab ja vajadusel keelab (vastavalt seadistusele saab määrata, kas hoiatatakse või takistatakse rakendusel käivitumast) selliste lohakuste tekkimist. JavaScripti puhul on selleks kõige levinumad tööriistad *JSLint* [60] või *ESLint* [61].

```
const queries = require("../queries").querys;
module.exports = router;
```

Joonis 15. Vananenud moodulite importimise ja eksportimise koodifragment failist *database.js*.

```
router.post('/tables', (req, res) => {
    const db = utils.connectToDatabase(req, res);

    db.query(queries.postgre.tables, (err, queryRes) => {


        res.json(queryRes);
        db.end()
    })
});
```

Joonis 16. Stiiliprobleemidega koodifragment failist *database.js*.

Eesrakenduse puhul on positiivne, et kaustade struktuur on hea ja selge. Koodis on kasutatud arusaadavaid ning selgitavaid funktsioonide ning muutujate nimesid. Kasutajaliidese komponendid on enamjaolt korralikult eraldatud. Miinusena tuleks tuua esile, et iga kord kui rakenduse veebilehekülge värskendatakse, logitakse süsteemist välja. Kriitiline viga on see, et kui lisada kasutajaliidese kaudu tabelile ühendamise operatsiooni väljakutse ja seejärel tabel päringust ära kustutada, läheb rakendus katki. Lisaks on rakenduses kasutatud arusaamatult Reacti funktsionaalseid ning klassi komponente. Klassi komponentide kasutus on antud juhul õigustatud, kui komponent vajab oleku haldamist (kuigi tänapäeval soovitatakse Reacti puhul juba täielikult funktsionaalsete komponentide kasutust). Samuti on koodistiil kohati lohakas, mille puhul aitaks jällegi *linter'i* kasutuselevõtt.

5 SQL lausete graafilise koostamise keskkonna kasutajaliidese disainimustrid

Graafiliste SQL lausete koostamise keskkondade loomise kohta ei leidnud autor häid allikaid, kuidas selliseid rakendusi peaks looma. Selles peatükis tuuakse välja erinevad olemasolevate rakenduste analüüsil põhinevad kasutajaliidese disainimustrid, mida võiks analoogse rakenduse loomisel järgida. Olgu mainitud, et disainimustrite valikul on lähtutud ainult rakenduste põhivaadetest (päringu loomine ja päringuga seotud tegevused) – andmebaasi sisselogimise, kasutaja registreerimiste ja muude toetavate funktsionaalsuste realiseerimise mustreid siin ei käsitleta. Mustri all peetakse silmas struktuurse kirjutamise viisi, mille alusel on võimalik mingi valdkonna (antud juhul kasutajaliidesed) probleeme ja lahendusi kirjeldada ühesugust (ja seega valikut ning võrdlemist soodustavat) struktuuri kasutades. Mustrite nimedest moodustub sõnastik, mida kasutades saavad valdkonna eksperdid probleemidele viidata.

Töös esitatava kasutajaliidese disainimustrite kataloogi koostamise jaoks uuriti kataloogi *UI-patterns* [62] ning valiti nende hulgast välja need, mida autori hinnangul tuleks pidada taoliste rakenduste loomisel oluliseks. Kokku vaadati läbi 66 mustrit ja valiti välja 12 mustrit. Mustrite valikuni jõuti uurides jaotises 2.3 välja toodud rakenduste kasutajaliidese disaine ning valiti sellised mustrid, mille puhul oli seda mustrit rakendatud vähemalt *kolme* rakenduse puhul. Kolme rakenduse nõue tuleneb üldisest mustrite kohta käivast reeglist, mille kohaselt peab probleemilahenduse mustriks osutumiseks olema selle lahenduse kasutamise kohta vähemalt kolm näidet [63]. Nende mustrite sõnastamisel võeti aluseks kirjeldused, mis on *UI-patterns* kataloogis, aga mõningate autoripoolsete muudatustega. Lisaks mõeldi autori poolt välja mustri eestikeelne nimetus ning toodi välja päringute koostamise rakendused, kus seda mustrit järgitakse. Samuti lisati ekraanipildina ja sõnalise selgitusena näide kasutatavast mustrist.

Mustrite kataloogi koostamisel analüüsiti ka paralleelselt edasiarendatavat rakendust (nii seisu enne arenduse algust kui ka plaanitavaid täiendusi) ja analüüsi tulemustest kirjutatakse peatükis 7.

Iga sobiliku ning töösse valitud kasutajaliidese disainimustri kohta tuuakse välja järgnevad alampunktid.

Alternatiivsed nimed eesti keeles: Alternatiivsed eestikeelsed nimed kui neid on.

Nimed inglise keeles: Võib olla mitu tükki.

Viited rakendustele: Viited päringute koostamise rakendustele (jaotisest 2.3), kus on disainimustrit järgitud.

Probleem: Millist probleemi üritatakse selle mustri abil lahendada.

Kasutusjuhud: Olukorrad, millal disainimuster on kasulik.

Lahendus: Kuidas realiseerida vajalik funktsionaalsus.

Põhjendus: Seletus, millised eelised disainimustri kasutuselevõtt kaasa toob.

Lahenduse probleemid (valikuline): Probleemid mis võivad tekkida selle lahenduse kasutamisel.

Näide: Ekraanipilt rakenduses kasutatavast disainimustrist ja selle sõnaline selgitus (sõnaline selgitus kirjeldab konkreetset rakenduse näidet, see tähendab et realisatsioon võib mõneti rakenduste vahel erineda, aga idee jääb samaks).

Iga mustri kohta käiva jaotise nimeks on selle mustri eestikeelne nimi.

5.1 Pukseeri

Nimed inglise keeles: Drag and Drop.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab objekti või objekte ühest kohast teise liigutada.

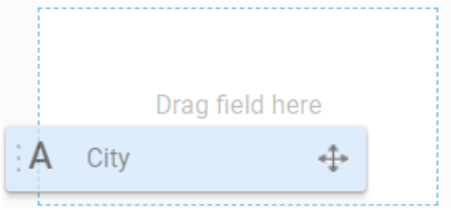
Kasutusjuhud: Kui on vajadus lasta kasutajal täita keerulisemat ülesannet läbi graafiliste elementide otsese töötlemise. Kui on soov vältida kasutaja liikumist järgmisele veebilehele, et leheküljel olevate elementide paigutust muuta.

Lahendus: Lasta kasutajal objekt valida (üles tõsta) ning objektide ümberpaigutamiseks mööda ekraani lohistada.

Põhjendus: Füüsilise maailma harjumustest tulenevalt proovivad kasutajad instinktiivselt objekte liigutada – see on üks tõhusamaid võimalusi objektide ümberpaigutamiseks.

Lahenduse probleemid: Võivad tekkida juurdepääsetavuse (*accessibility)* probleemid (puuetega inimestele keerulisem kasutada), tuleks kaaluda ka alternatiivsete võimaluste pakkumist.

Näide: Kasutaja soovib andmebaasi veergu valida. Selleks vajutab ta hiirega tabeli veerul (*City)* ning lohistab selle vastavasse alasse/aknasse (Joonis 17).



Joonis 17. Pukseerimise disainimustri kasutamise näide rakenduses *Datapine Query Builder*.

5.2 Sisestusviip

Alternatiivsed nimed eesti keeles: Kohahoidja.

Nimed inglise keeles: Input Prompt. Placeholder.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31], Postgres Visual Query Builder.

Probleem: Kasutaja tahab andmeid süsteemi sisestada.

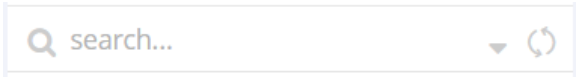
Kasutusjuhud: Kui sisestusvälja silt ei anna piisavalt palju informatsiooni, mida sisestada. Kui soovitakse sildi arvelt ruumi kokku hoida. Kui sisestusvälja silt tundub liiga pikk ja seletav.

Lahendus: Sisestusväli täidetakse eelnevalt selgitusega (võib olla küsimuse vormis) või näiteväärtusega, mis annab kasutajale infot, mida tuleb väljale sisestada.

Põhjendus: Kasutaja soovib üldjuhul täita vajalikke sisestusvälju võimalikult kiiresti ja efektiivselt. Sellest tulenevalt ei pöörata suurt tähelepanu vormi siltidele. Sisestusviip jääb aga kasutajale koheselt silma, mida nähes ei lähe ta sellest mööda.

Lahenduse probleemid: Tuleb meeles pidada, et kui fookus on sisestusviipel või kui väljas on juba mingi väärtus, siis kaob ka abistav tekst (seega ei pruugi siltide eemaldamine alati mõistlik olla). Näiteväärtuse korral võib kasutajale jääda mulje, et ta peaks välja selle sama väärtuse sisestama või isegi, et väli on juba täidetud.

Näide: Kasutaja soovib märksõna alusel tabelit otsida. Selleks, et otsingukasti paremini üles leida või paremini aru saada, milleks see kast mõeldud on, kuvatakse talle sisestusviip (Joonis 18).



Joonis 18. Sisestusviiba disainimustri kasutamise näide rakenduses *Skyvia Query Builder*.

5.3 Eelvaade

Nimed inglise keeles: Preview. Live Preview.

Viited rakendustele: Active Query Builder [29], Datapine Query Builder [31], APEX Query Builder [32], Postgres Visual Query Builder.

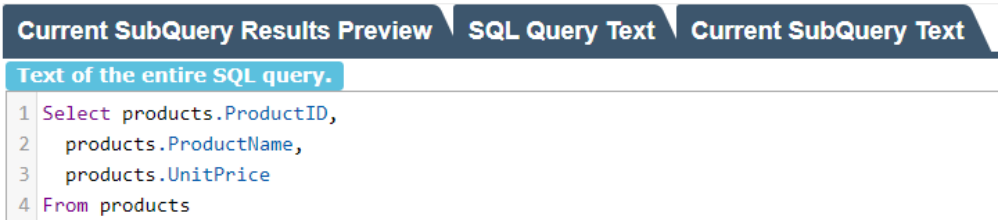
Probleem: Kasutaja tahab võimalikult kiiresti näha, kuidas tema poolt tehtud valikud muudavad lõpptulemust.

Kasutusjuhud: Kui soovitakse kasutajale reaalajas näidata, kuidas tema poolt tehtud valikud mõjutavad tulemusi. Kui ilma eelvaateta on keeruline aru saada, milline võiks lõpptulemus välja näha.

Lahendus: Näidata kasutajale enne käivitatavat tegevust, milline tema antud sisendi alusel väljund välja näeb. Pakkuda reaalajas nähtavat eelvaadet, mis reageerib koheselt igale kasutaja poolt tehtud muudatusele.

Põhjendus: Eelvaade annab kasutajale mängimisruumi ja lubab turvalisemalt erinevaid võimalusi läbi proovida. Selle abil on kasutajal parem mõista, kas ta soovib süsteemis sellist muudatust teha. Tulemusena saavutatakse parem interaktiivsus ning kasutaja ei pea ootama veebilehekülje laadimiste taga, et näha, kas muudatused olid sobilikud või mitte.

Näide: Kasutaja koostab SQL lauseid graafilist liidest kasutades. Liideses erinevaid valikuid tehes kuvatakse eelvaatena vahekaardis (*SQL Query Text)* SQL kood, mis annab kasutajale ülevaate kogu SQL lausest enne selle käivitamist (Joonis 19).



Joonis 19. Eelvaate disainimustri kasutamise näide rakenduses *Active Query Builder*.

5.4 Otsene redaktor

Nimed inglise keeles: Inplace Editor. Direct Manipulation.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], Datapine Query Builder [31], DevExpress Query Builder [30].

Probleem: Kasutaja soovib lihtsalt ja kiiresti muuta leheküljel mingit väärtust.

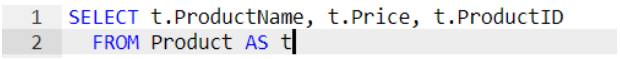
Kasutusjuhud: Kui soovitakse pakkuda kasutajale lisavõimalust muuta/kohandada mingisugust tekstilises formaadis väärtust. Kui kasutaja soovib muuta väärtust ilma administratiivseid liigutusi tegemata (vajaduseta vahetada näiteks süsteemis oma õiguseid või administraatori kontoga sisse logida või lahkuda olemasolevast vaatest).

Lahendus: Lasta kasutajatel muuta väärtusi sealsamas, kus neid kuvatakse. Teha redaktori komponent kursori jaoks nähtavaks, millele vajutades muutub see sisetusväljaks. Muudatuste tegemisel ei ole vajadust lehekülge värskendada (Joonis 20).

Põhjendus: Otsene redaktor võimaldab kergesti leheküljel olevaid väärtusi muuta, ilma, et peaks lehekülge värskendama või liikuma muudatuste vaatesse.

Lahenduse probleemid: Kasutaja võib otsest redaktorit kasutades proovida käivitada keelatuid või rakendusi lõhkuvaid käskluseid või ta võib proovida õngitseda süsteemist infot, millele tal pole juurdepääsuõigust.

Näide: Kasutaja soovib graafiliste tegevuste tulemusena genereeritud SQL lauset muuta. Selleks antakse kursori abil märku, et tegemist on redaktoriga, millele peale vajutades saab kasutaja otse SQL koodi muuta, tegemata graafilisi muudatusi.



Joonis 20. Otsese redaktori disainimustri kasutamise näide rakenduses *Skyvia Query Builder*.

5.5 Head vaikimisi valikud

Alternatiivsed nimed eesti keeles: Head vaikimisi väärtused.

Nimed inglise keeles: Good Defaults.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab sisestada süsteemi andmeid (sh anda näiteks juhiseid edasisteks tegevusteks), kuid mõne atribuudi (olemi nimelise omaduse puhul) on suure tõenäosusega sisestamist vajavaks väärtuseks mingi kindel väärtus.

Kasutusjuhud: Kui kasutaja peab valima erinevate väärtuste vahel, millest ühe kasutamine on teistest märgatavalt tõenäolisem. Kui kasutatavast süsteemist või kasutaja eelnevast käitumisest tulenevalt saab eeldada, millist valikut kasutaja võiks eelistada. Kui mõne välja valimisel on keeruline sobilikku valikut teha või pakutakse liiga palju valikuid, mis võivad kasutaja segadusse ajada.

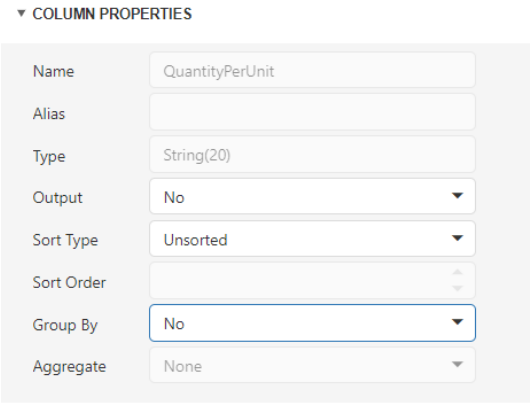
Lahendus: Eeltäita vormiväljad valikutega, mida tõenäoliselt kasutaja võiks valida. Kasutada neid rippmenüüde, tekstiväljade ja märkeruutude puhul ning arvestada just kasutajate jaoks oluliste valikutega.

Põhjendus: Pakkudes vaikeväärtusi keeruliste valikute juures hoidutakse kasutaja segadusse ajamisest. Kasutaja ei pea ilma vajaduseta ise konteksti liialt süvenema. Alati ei pruugi vaikeväärtus õige olla, aga vähemalt saab seda hiljem kerge vaevaga muuta.

Lahenduse probleemid: Kui ei osata arvestada kasutaja põhivajadustega võib vale vaikeväärtus hoopis lisatööd tekitada, sest hiljem tuleb registreeritud andmeid või

süsteemi koostatud tulemust parandada või hoopis valikute tegemine ja käitumise algatamine uuesti läbi teha. Kasutaja ei pruugi vaikeväärtuse tõttu enam väljale tähelepanu pöörata ja nii satuvad süsteemi valed andmed või tehakse süsteemi poolt valesid tegevusi. Vaikeväärtuse hilisem eemaldamine või muutmine võib kasutaja segadusse ajada, sest ta on harjunud ootama süsteemilt midagi muud.

Näide: Kasutaja soovib uue veeru valimisel päringusse, et veeru alusel ei toimuks automaatselt grupeerimist. Selleks on kasutusel hea vaikimisi valik (*No)*, mis eeldab, et esmasel veeru valimisel ei soovi kasutaja veeru alusel grupeerimist, vaid lastakse vajadusel seda kasutajal ise teha (Joonis 21).



Joonis 21. Heade vaikimisi valikute disainimustri kasutamise näide rakenduses *DevExpress Query Builder.*

5.6 Modulaarsed vahekaardid

Alternatiivsed nimed eesti keeles: Struktureeritud sakid.

Nimed inglise keeles: Module Tabs.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab näha ühtse temaatika/kategooria alla kuuluvat sisu erinevate osadena/vahekaartidena, kus osad peavad olema ligipääsetavad ühise komponendi kaudu, kasutades selleks ühetasemelist navigeerimisstruktuuri.

Kasutusjuhud: Kui visuaalne ruum on piiratud ja lehekülje sisu tuleb eraldada osadeks. Kui ühetasemelist navigeerimist vajavaid osi on vahemikus kaks kuni üheksa. Kui ilma

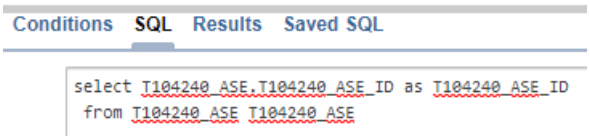
lehekülge värskendamata tuleb hoida kasutaja tähelepanu. Kui erinevate vahekaartide nimetused on suhteliselt lühikesed. Kui erinevate vahekaartide sisu on on mõistetav ilma samal ajal teisi vahekaarte vaatamata. Kui iga vahekaardi sisu struktuur on sarnane. Kui soovitakse näidata, millise vahekaardi sisu (millist alamosa lehel esitatud andmetest ja käitumisest) parajasti vaadatakse.

Lahendus: Näidata kasutajale korraga ühe vahekaardi sisu ja esitada see eraldi kastina. Sisu kohale tuleb lisada horisontaalne riba, mis pakub vahekaartide vahel valimise võimalust. Horisontaalsele ribale tuleb lisada rohkem kui üks vahekaart. Kasutada visuaalseid vahendeid, et tõsta valikuribal esile valitud vahekaart. Horisontaalse riba struktuur peab vahekaardi vahetamisel säilima. Vältida lehekülje värskendamist vahekaardi vahetamisel.

Põhjendus: Modulaarsed vahekaardid pakuvad lihtsat viisi, kuidas näidata kasutajale sarnase struktuuriga sisu, mis on kategooriate põhjal eraldatud.

Lahenduse probleemid: Vahekaartide suur hulk, ebaloogiline järjekord ja halvasti valitud nimed võivad muuta soovitud sisu leidmise raskemaks. Lehel esitatud andmed ja väljendatud käitumine ei ole korraga nähtavad.

Näide: Kasutaja tahab navigeerida erinevate rakenduse osade vahel. Selleks pakutakse talle horisontaalset vahekaartide riba, kus hetkel aktiivne olek (*SQL)* on visuaalselt eristatud (valitud vahekaardi pealkirja all asuv paksem joon). Vahekaardil vajutades kuvatakse sellele vahekaardile vastavat sisu, antud juhul genereeritud SQL koodi (Joonis 22).



Joonis 22. Modulaarsete vahekaartide disainimustri kasutamise näide rakenduses *APEX Query Builder.*

5.7 Vertikaalne rippmenüü

Nimed inglise keeles keeles: Vertical Dropdown Menu.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab liikuda lehekülje erinevate osade vahel või valida süsteemi funktsionaalsuste hulgast täitmiseks endale vajaliku, kuid iga eraldi valiku või osa esitamise jaoks on lehekülje ruum piiratud.

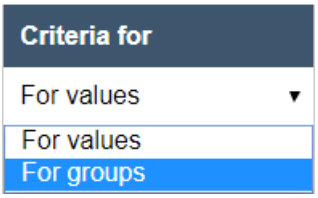
Kasutusjuhud: Kui leheküljel on kaks kuni üheksa osa/sektsiooni või valikut, mis vajavad hierarhilist struktuuri. Kui soovitakse pakkuda kasutajale kiiret ülevaadet erinevatest võimalustest valiku tegemiseks.

Lahendus: Pakkuda kasutajale rippmenüüd, kus on valikud toimingute tegemiseks, kuvades kõik valikud järjendina teineteise all. Kui kasutaja liigub kursoriga rippmenüü peale, siis avada see automaatselt või anda visuaalselt märku (kursori, sümboliga), et elementi saab valida. Kui kasutaja eemaldab kursori või vajutab hiirega mujale, siis tuleb rippmenüü sulgeda.

Põhjendus: Rippmenüüde kasutamise põhiline eelis on see, et nad hoiavad lehekülje ruumi kokku ja koondavad kõik valikud ühe visuaalse komponendi alla.

Lahenduse probleemid: Puhtalt JavaScriptis tehtud rippmenüüd kasutatavat veebilehte ei indekseerita hästi otsingumootorite poolt. Rippmenüüsid on tihtipeale tülikas kasutada. Kui menüü on visuaalselt väike ja kursor on sellest kaugel, siis võtab valiku tegemine suhteliselt palju aega (vt Fitti seadus [64]). Kõik valikud pole korraga nähtavad. Kasutajad kulutavad valiku otsimiseks aega. Erinevad kasutajad võivad eelistada/eeldada erinevat menüüsüsteemi, st mis ühele sobib, see tundub teisele ebaloogiline. Menüüsüsteemi muutmine tekitab segadust, sest kasutaja ei leia valikut harjumuspärasest kohast.

Näide: Kasutaja soovib valida, mille jaoks veeru filtrit rakendatakse. Selleks pakutakse talle rippmenüüd, millele vajutades kuvatakse nimekiri erinevatest valikutest (Joonis 23).



Joonis 23. Vertikaalse rippmenüü disainimustri kasutamise näide rakenduses *Active Query Builder*.

5.8 Akordionmenüü

Alternatiivsed nimed eesti keeles: Lõõtsmenüü.

Nimed inglise keeles: Accordion Menu.

Viited rakendustele: Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31].

Probleem: Kasutaja tahab navigeerida lehekülje põhisektsioonide vahel ning samaaegselt näha ka alamsektsioonide sisu või nende peale liikuda.

Kasutusjuhud: Kui soovitakse kasutada tavapärast külgmenüüd, kuid pole ruumi kõigi valikute korraga kuvamiseks. Kui on rohkem kui kaks põhisektsiooni, millel mõlemal on lisaks kaks või rohkem alamsektsiooni. Kui on vähem kui kümme põhisektsiooni. Kui põhisektsioon on ainult kahetasemeline.

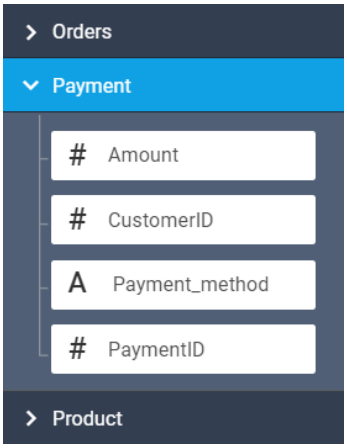
Lahendus: Igal sektsioonil või pealkirjal on olemas paneel, mida vajutades avanevad kas horisontaalselt või vertikaalselt alamsektsioonid. Kui vajutada ühel paneelil, siis see avatakse ning kõik teised paneelid suletakse.

Põhjendus: Tihtipeale kasutatakse akordionimenüüsid lehekülje põhinavigatsioonina, et kergesti sektsioonide vahel liikuda. Üldiselt kasutatakse neid vertikaalsete põhimenüüdena, kuid teatud juhtudel saab ka alammenüüdena kasutada.

Lahenduse probleemid: Kõik valikud pole korraga nähtavad. Kasutajad kulutavad valiku otsimiseks aega. Erinevad kasutajad võivad eelistada/eeldada erinevat menüüsüsteemi, st mis ühe sobib tundub teisele ebaloogiline. Menüüsüsteemi muutmine tekitab segadust, sest kasutaja ei leia valikut harjumuspärasest kohast. Kui menüü on

visuaalselt väike ja kursor on sellest kaugel, siis võtab valiku tegemine suhteliselt palju aega (vt Fitti seadus [64]).

Näide: Kasutaja soovib näha andmebaasi tabelisse kuuluvaid veerge. Selleks pakutakse akordionmenüüd, kus põhisektsioonideks on tabelid. Põhisektsioonil vajutades (*Payment)* kuvatakse alamsektsioonidena kõik selle tabeli veerud (Joonis 24).



Joonis 24. Akordionmenüü disainimustri kasutamise näide rakenduses *Datapine Query Builder*.

5.9 Kaardid

Alternatiivsed nimed eesti keeles: Kaardipakk.

Nimed inglise keeles: Cards.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], DevExpress Query Builder [30], Datapine Query Builder [31], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab sirvida erinevate pikkusete ja tüüpidega sisu.

Kasutusjuhud: Kui soovitakse kuvada sisu, mis on kokku pandud erinevatest elementidest, mis võib olla moodustunud erinevat tüüpi väärtustest. Kui soovitakse näidata elemente, mille suurus ja võimalikud seotud funktsionaalsused erinevad (näiteks fotod, millel on erinevate pikkustega pealkirjad). Kui kuvatakse elemente, mis ei vaja otsest võrdlemist või on interaktiivsed. Kui soovitakse pakkuda sisu lühikirjeldust ning samas võimalust detailvaatesse minemiseks. Kui tahetakse esitada ühe temaatika kohta

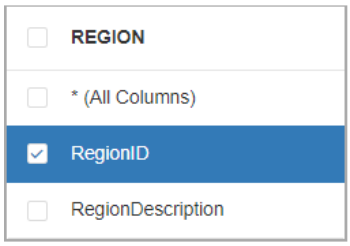
käivat mingil määral erinevat informatsiooni ühtse visuaalse komponendina. Kui kasutaja sirvib informatsiooni, mitte ei otsi vastust mingile täpsele küsimusele.

Lahendus: Kuvada kasutajale ligipääsu erinevate sisude ning detailvaadete juurde sarnaste komponentidena. Kaart võib sisaldada nii pilti, video, teksti kui ka linke ühe teema kohta. Pakkuda kasutajale kaartidest koosnevat kollektsiooni, mida saab kerida ainult ühes suunas (horisontaalselt või vertikaalselt). Disaini puhul kasutada ümardatud nurki, et toonitada päriselus kasutatavat reaalse kujuga kaarti. Lisada vari, et tuua esile sügavust ja anda märku, et kogu kaart on klõpsatav.

Põhjendus: Kasutaja tahab kiiresti ülevaadet suurest hulgast sisuelementidest ning vastavalt vajadusele sobiva elemendi peale pidama jääda. Kaardid on hea võimalus näidata erineva suuruse ja funktsionaalsusega osadest kokkupandud elemente. Kaardid käituvad konteinerina, mille sees on kasutajale olulised osad, mida saab vajadusel detailsemalt uurida.

Lahenduse probleemid: Kaarte kasutatakse tihtipeale veebilehtedel, kus on palju informatsiooni. See võib tekitada visuaalse ülekoormuse, mis teeb sisu hoomamise keerulisemaks.

Näide: Kasutaja soovib näha andmebaasi tabeli nime ja selle veerge ühtse komponendina ning vajadusel veerge valida. Lahendusena kuvatakse vastava tabeli valimisel selle tabeli nimi (näites *Region)* koos selle veergudega ühtse komponendina, kus päringusse veergude valimisel on võimalik valida neid ühekaupa või kõik korraga (märgistades tabeli nime ees asuva märkeruudu) (Joonis 25).



Joonis 25. Kaartide disainimustri kasutamise näide rakenduses *DevExpress Query Builder*.

5.10 Lehekülgedeks jagamine

Alternatiivsed nimed eesti keeles: Paginatsioon.

Nimed inglise keeles: Pagination. Paging.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab näha sorteeritud andmete alamhulka arusaadaval kujul.

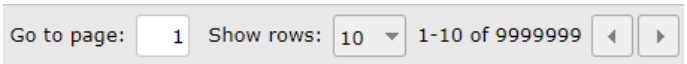
Kasutusjuhud: Kui kõigi andmete kuvamine korraga ühel leheküljel ei ole mõistlik, sest andmeid on liiga palju. Kui andmed on mingil kasutajale arusaadaval viisil järjestatud.

Lahendus: Tükeldada üks suur andmehulk väiksemateks osadeks ning pakkuda linkidena võimalust iga osa eraldi leheküljena vaatamiseks. Pakkuda lehekülje valimiseks eraldi menüüd või kontrollkasti, mis sisaldab linke nii järgmise, eelneva, esimese kui ka viimase alamhulga vaatamiseks. Võimalusel pakkuda linki ka konkreetsel leheküljel asuvale alamhulgale. Kui andmehulga suurus on muutlik, siis viimase lehekülje linki pole vaja näidata. Paigutada lehekülje valimise kontrollkast komponendi või lehekülje alamossa.

Põhjendus: Suurte andmehulkade osadeks jaotamine muudab informatsiooni paremini hoomatavaks ja hallatavaks. Kogu andmehulga asemel alamhulkade tagastamine aitab saavutada ka paremat tehnilist jõudlust. Lehekülgedeks jagamine annab kasutajale informatsiooni, kui palju on andmeid vaadatud ning palju on veel vaadata. Samamoodi, võimaldab see teha korraks andmete lugemisest pausi ning võimaluse mõtlemiseks, kas soovitakse edasi uurida või leheküljelt lahkuda. See on ka põhjus, miks paginatsioon paigutatakse üldjuhul lehekülje alaossa.

Lahenduse probleemid: Kui süsteemis puudub (hea) otsingu võimalus või ridade sorteerimise võimalus (vt muster Veeru järgi sorteerimine), siis on hädapäraseks võimaluseks vaadata korraga kõigi olemite andmeid ja teha otsing kasutades veebilehitseja võimalusi. Lehekülgedeks jagamine muudab selle tülikamaks. Selleks, et lehtedelt vajalike andmetega rida ilma kõiki lehekülgi läbikäimata üles leida, peavad read olema mingi kasutajale teada oleva tingimuse alusel sorteeritud.

Näide: Kasutaja soovib näha mahukat andmebaasi päringu tulemust erinevate osadena. Selleks pakutakse talle paginatsiooni komponenti, et paremini osade vahel liikuda (Joonis 26).



Joonis 26. Lehekülgedeks jagamise disainimustri kasutamise näide rakenduses *Active Query Builder*.

5.11 Veeru järgi sorteerimine

Nimed inglise keeles: Sort by Column.

Viited rakendustele: Active Query Builder [29], Datapine Query Builder [31], Postgres Visual Query Builder.

Probleem: Kasutaja tahab sorteerida tabeli andmeid veerus olevate väärtuste põhjal.

Kasutusjuhud: Kui tabelis on palju ridu (rohkem kui kümme) ja on keeruline välja tuua ühte rida ning selle suhet teiste ridadega. Kui visuaalselt esitatud tabelis on kasutusel lehekülgedeks jagamine ja andmed on jaotatud mitme lehekülje vahel. Kui soovitakse tabeli ridu omavahel võrrelda.

Lahendus: Muuta tabeli funktsionaalsust selliselt, et iga veeru päis on link. Kui vastava veeru päisel klõpsata, siis sorteeritakse read selles veerus olevate andmete alusel kasvavalt. Kui samal lingil korra veel klõpsata, siis sorteerida read selles veerus olevate andmete aluselt kahanevalt. Kui tabeli read on sorteeritud mingi konkreetse veeru põhjal, siis tavaliselt on päises ka sorteerimise suunda näitav visuaalne indikaator (näiteks nool).

Põhjendus: Veeru järgi sorteerimine pakub lihtsat võimalust suurte andmehulkade organiseerimiseks, olemite andmete võrdlemiseks ja vajalike andmete leidmiseks.

Lahenduse probleemid: Kuidas anda kasutajatele võimalus sorteerida mitme veeru järgi.

Näide: Kasutaja soovib päringu tulemusi veeru põhjal sorteerida. Selleks pakutakse veeru järgi sorteerimist, kus veeru nimele (*Amount)* vajutades sorteeritakse see vastavalt. Sorteeringu suunast annab märku visuaalne nool, antud näite puhul sorteeritakse kahanevalt (lisaks kuvatakse ka tööriistavihje (*tooltip)* võimalustest) (Joonis 27).

Joonis 27. Veeru järgi sorteerimise disainimustri kasutamise näide rakenduses *Datapine Query Builder*.

5.12 Koopiakast

Nimed inglise keeles: Copy Box.

Viited rakendustele: Skyvia Query Builder [28], Active Query Builder [29], APEX Query Builder [32], Postgres Visual Query Builder.

Probleem: Kasutaja tahab kergesti näha ja kopeerida eelvormindatud teksti.

Kasutusjuhud: Kui soovitakse kuvada vormindatud teksti, mida saab kergesti ja ilma vormingut kaotamata kopeerida. Kui on vaja tuua näidisena välja programmikoodi. Kui soovitakse pakkuda kasutajale võimalust programmikoodi kopeerimiseks ning järeltöötlemiseks muus rakenduses.

Lahendus: Luua eraldi kast, millel on parema eristatavuse huvides ülejäänud rakendusest erinev stiil. Kasutada üheruumilist (*mono-space)* ning kindla laiusega fonti, et visuaalselt paistaks tekstikast terminali aknana või tekstiredaktorina. Lisada HTMLi (*Hypertext Markup Language)* <pre> märgendid (*tag*) teksti ümber, mille vormingut soovitakse säilitada või kasutada <textarea> elementi.

Põhjendus: Koopiakast pakub võimalust säilitada programmikoodi vorming. Kui kasutada tavalist ilma märgenditeta teksti, mis on küll rakenduse HTML failis reavahedega, siis veebilehitseja vormi ei säilita, vaid paigutab kõik ühele reale.

Näide: Kasutaja soovib kopeerida rakenduse poolt genereeritud SQL koodi ning säilitada selle struktuur, et sellega näiteks mõnes muus rakenduses edasi töötada. Selleks pakutakse talle koopiakastis võimalust vorminguga teksti valimiseks ja kopeerimiseks (Joonis 28).

Joonis 28. Koopiakasti disainimustri kasutamise näide rakenduses APEX Query Builder.

6 Tarkvara uue versiooni nõuded ja väljalaske plaan

Selles peatükis kirjeldatakse, kuidas toimus väljalaske planeerimise protsess ning antakse ülevaade koostatud plaanist ja selle koostamise sisendiks olnud nõuetest.

Nagu jaotises 2.4 mainiti, kasutati planeerimise jaoks Tommy Normani poolt väljapakutud meetodit [35]. Planeerimise protsess tuuakse välja etappide kaupa ning kirjeldatakse tehtud tegevusi.

T. Norman mainib oma artiklis, et planeerimise alustamiseks on vaja toote täitmata tööde nimekirja e toote tööjärge e soovilogi (*product backlog)*. Selleks tuleb kindlaks teha nõuded süsteemile. Funktsionaalsete nõuete, mis ühtlasi on väljalaske plaani koostamise sisendiks, esitamiseks kasutatakse kasutuslugusid (*user stories*) [65] (vt jaotis 6.1). Mittefunktsionaalseid nõudeid, mida tuleb arvestada kõigi funktsionaalsuste realiseerimisel ning millega tegeleti ka enne arendust (iteratsioon 0), esitatakse tabeli kujul (vt jaotis 6.2). Kuna töö autor oli antud projektis ainuke arendaja, siis koostas autor iseseisvalt esialgse väljalaske plaani koos vajalike prioriteetidega (vt jaotis 6.3). Hiljem vaadati see koos juhendajaga üle ning tehti vajalikud muudatused.

6.1 Funktsionaalsed nõuded

Esimese asjana koguti kokku esialgne hulk kasutuslugusid (sellel ajal kõige olulisemana tundunud puuduvate funktsionaalsuste kohta) ning lisati need Exceli tabelisse. Selles etapis ei mõeldud veel prioritiseerimiste ega hinnanguliste pingutuste peale, vaid pandi kõik ideed kasutuslugudena kirja sellises järjekorras nagu need sõnastati (Tabel 2). Nõuded leiti koostöös toote omanikuga (selles rollis on antud töö kontekstis juhendaja). Nõude juures on sulgudes viide SQLi võtmesõnale või mõistele, mille kasutamise soovile see nõue osutab.

Tabel 2. Süsteemi kasutuslood.

ID	Kasutuslugu
1000	Kasutajana tahan koondandmete päringute tulemustele rakendada koondandmete alusel piiravaid tingimusi, et küsida ainult neid andmeid mida vaja (HAVING).
1001	Kasutajana tahan piirata tagastatud ridade hulka, et näha tulemuses ainult valitud arvu ridu (FETCH, LIMIT).

ID	Kasutuslugu
1002	Kasutajana tahan luua rekursiivseid päringuid, et pärida hierarhilisi andmeid (ühised tabeli avaldised – *common table expressions*, CTE).
1003	Kasutajana tahan koostada virtuaalseid tuletatud tabeleid, mille põhjal saan teha järgnevaid päringuid ja niimoodi suuremat andmete otsimise probleemi väiksemate sammudena lahendada (ühised tabeli avaldised – *common table expressions*, CTE).
1004	Kasutajana tahan arvutada iga rea jaoks koondtulemuse ridade põhjal, mis on mingil viisil selle reaga seotud, et näiteks arvutada asukohta pingereas (OVER, PARTITION BY).
1005	Kasutajana tahan kasutada tabelite ühendi operaatorit, et leida erinevate sama struktuuriga tulemuse andvate päringute tulemuste ühend (UNION, UNION ALL, *set operator*).
1006	Kasutajana tahan kasutada tabelite ühisosa operaatorit, et leida erinevate sama struktuuriga tulemuse andvate päringute tulemuste ühisosa (INTERSECT, *set operator*).
1007	Kasutajana tahan kasutada tabelite vahe operaatorit, et leida erinevate sama struktuuriga tulemuse andvate päringute tulemuste vahe (EXCEPT, *set operator*).
1008	Kasutajana tahan tabeli ridade väljade väärtuseid uuendada, et andmebaasis oleksid soovitud ja asjakohased andmed (UPDATE).
1009	Kasutajana tahan tabeli ridu kustutada, et andmebaasis oleksid soovitud ja asjakohased andmed (DELETE).
1010	Kasutajana tahan tabelisse uusi ridu lisada, et andmebaasis oleksid soovitud ja asjakohased andmed (INSERT).
1011	Kasutajana tahan koostada liit-otsingutingimusi, mitmest liht-otsingutingimusest, sidudes need loogikaoperaatoritega AND või OR, selleks, et leida keerukatele tingimustele vastavaid ridu.
1012	Kasutajana tahan arvuliselt näha mitu tabeli rida on päringu tulemuses, et oleks kiire ülevaade tulemuste mahust.
1013	Kasutajana tahan näha veeru tüüpi, et kiiremini aru saada, milliste andmetega on tegu ja milliseid operatsioone saab nende andmetega teha.
1014	Kasutajana tahan piirata tulemuses olevate ridade hulka sama tabeli teiste ridade või teiste tabelite ridade põhjal kontrollitud tingimuste alusel (alampäringud WHERE klauslis).
1015	Kasutajana tahan rakenduse veebilehte värskendades jääda sisselogituks, et ei peaks iga kord uuesti andmeid sisestama.
1016	Kasutajana tahan tabelilt ühendamise operatsiooni eemaldada ning seejärel tabelit päringust kustutada, ilma, et rakendus katki läheks, et jätkata rakendusega töötamist.
1017	Kasutajana tahan tulemuste tabelis näha ka väärtuseid tõeväärtustüüpi veergudest, et saada soovitud ja asjakohast informatsiooni.

ID	Kasutuslugu
1018	Kasutajana tahan GROUP BY päringus kasutada kaamliküüru (camelCase) stiilis nimedega veerge, et pärida andmeid ka selliste nimedega veergudest.
1019	Kasutajana tahan näha tulemuste tabelis vahelduvate värvidega ridu, et ridu üksteisest visuaalselt paremini eristada.
1020	Kasutajana tahan genereeritud SQLi ise tekstiväljas muuta või käsitsi kirjutada, et saaks mugavalt lauses muudatusi teha või luua lauseid, mille koostamise võimalust visuaalsed komponendid ei paku.

6.2 Mittefunktsionaalsed nõuded

Eraldi tabeli kujul pandi kirja kõik tarkvara mittefunktsionaalsed nõuded (Tabel 3). Nende saavutamiseks vajalikke tegevusi tehti enne arendust (iteratsioon 0) ja nendele pöörati tähelepanu ka kogu arenduse kestel. Nende nõuete täpsem sisu on avatud jaotises 4.4, kus kirjeldatakse olemasolevat rakendust.

Tabel 3. Süsteemi mittefunktsionaalsed nõuded.

ID	Nõude kirjeldus	Vastutaja
1.1	Tagarakendus peab kasutama linterit, et tagada tagarakenduse koodipuhtus ja korrektne stiil.	Arendaja
1.2	Eesrakenduse peab kasutama linterit, et tagada eesrakenduse koodipuhtus ja korrektne stiil.	Arendaja
1.3	Eesrakendus peab kasutama React teegi funktsionaalseid komponente seal, kus klassikomponent pole mõistlik, et vältida segaduste tekkimist ning järgida Reacti kasutamise soovitusi.	Arendaja
1.4	Rakenduses peab kasutama uusimaid ja soovitatumaid programmeerimiskeele võimalusi, et rakendus järgiks kaasaegseid ning parimaid praktikaid.	Arendaja
1.5	Andmebaasist andmete küsimisel peab kasutaja parool olema krüpteeritud, et tagada parooli turvalisus.	Arendaja
1.6	Rakendus peab kasutama andmebaasist andmete küsimiseks GET-päringuid, mitte POST-päringuid, et mitte ilma asjata saata iga kord päringuga kaasa JSON (*JavaScript Object Notation)* objekti.	Arendaja
1.7	Kasutajaliides peab järgima kasutajaliidese disaini mustreid (vt peatükk 5).	Arendaja

6.3 Väljalaske plaan

Selles jaotises esitatakse väljalaske plaan.

6.3.1 Kasutuslugude prioritiseerimine ja hinnangute määramine

Kui Exceli tabel koos kasutuslugudega oli olemas, pandi paika prioriteedid ning hinnangud vastavalt sellele, mida autor koos juhendajaga kõige tähtsamateks pidasid. Plaani esimese versiooni koostas autor ja siis vaadati see koos juhendajaga üle ning täiendati. Töös esitatakse täiendatud plaan (Tabel 4). Kõrgem prioriteet määrati ülesannetele, mis parandavad olemasolevat funktsionaalsust või rakendusest leitud vigu. Prioriteetide puhul kasutati lihtsat arvulist meetrikat: 1, 2, 3. Pingutuse hinnangute (*estimates)* esitamiseks kasutati Fibonacci arve, mis on sellist tüüpi ülesande puhul populaarseim hinnangute esitamise viis. Lõpuks sorteeriti tööde nimekiri prioriteetide põhjal.

Täitmata tööde tabelis on kolm veergu.

- ID (annotatsioon) – unikaalne identifikaator iga töö jaoks – viide kasutusloole (vt Tabel 2), koos sisu avavate märksõnadega.
- Täitmata töö prioriteet – arv, mis esitab täitmata töö prioriteedi (mis järjekorras tööd teostada), tulenevalt toote omaniku (*Product Owner)* soovist.
- Hinnanguline pingutus – autori hinnang tööülesande täitmiseks vajalikule pingutusele.

Tabel 4. Kasutuslugude prioriteedid ning hinnangud.

ID (annotatsioon)	Täitmata töö prioriteet	Hinnanguline pingutus
1012 (ridade arv)	1	3
1013 (veergude tüübid)	2	5
1016 (ühendamine ei tee katki)	3	3
1015 (lehe värskendamine)	4	5
1017 (tõeväärtustüüpi veerud)	5	2
1018 (tõstutundlikud nimed)	6	3
1020 (SQLi kirjutamine)	7	8
1019 (ridade värvimine)	8	3

ID (annotatsioon)	Täitmata töö prioriteet	Hinnanguline pingutus
1011 (liit-tingimused)	9	8
1014 (alampäringud WHERE)	10	8
1000 (HAVING)	11	8
1001 (LIMIT)	12	5
1005 (UNION, UNION ALL)	13	5
1006 (INTERSECT)	14	5
1007 (EXCEPT)	15	5
1004 (OVER, PARTITION BY)	16	13
1003 (CTE)	17	40
1002 (rekursiivsed päringud)	18	20
1010 (INSERT)	19	20
1008 (UPDATE)	20	20
1009 (DELETE)	21	20

6.3.2 Töökiiruse määramine

Töökiirus (*velocity)* on kogu meeskonnatöö pingutuste summa, mis tehakse ühe iteratsiooni jooksul. Kuna autor oli selles projektis ainus arendaja, siis määras ta esialgu töökiiruseks 16.

6.3.3 Väljalaske plaani loomine

Olles saanud prioritiseeritud ja hinnatud tegemata tööde nimekirja ning määranud esialgse töökiiruse, sai autor paika panna väljalaske plaani. Alustati tegemata tööde nimekirja tipust ja liiguti allapoole, summeerides pingutuse hinnanguid. Kui pingutuse hinnangute summaarne väärtus jõudis töökiiruseni, siis lõpetati ja seni summeeritud ülesanded läksid ühte iteratsiooni. Kui tööülesanded olid iteratsioonide vahel ära jagatud, sai paika panna ka projekti alustuskuupäeva ja iga iteratsiooni lõppkuupäeva.

Tabel 5 on välja toodud kogu väljalaske plaan koos kasutuslugude identifikaatorite, nende prioriteetide ning hinnanguliste pingutustega. Samuti on välja toodud iteratsioonide lõppkuupäevad ning iga iteratsioon on tähistatud erineva värviga. Projekti alustuskuupäev oli 11.02.2020. Projekt algas iteratsiooniga 0, esimene iteratsioon algas 11.03.2020. Väljalaskeplaani koostamisel valiti töökiiruseks 16 ning iga järgneva iteratsiooni

pikkuseks kaks nädalat. Põhjalikumalt kirjutatakse arendusprotsessist peatükis 3. Töö mahukusest tulenevalt ei olnud võimalik kõiki soovitud funktsionaalsuseid selles väljalaskes realiseerida. Realiseerimata funktsionaalsused on nähtavad Tabel 5 lõpus valgel taustal.

Tabel 5. Lõputööna valmiva väljalaske plaan.

ID (annotatsioon)	**Täitmata töö prioriteet**	**Hinnanguline pingutus**	**Iteratsiooni lõpp**
1012 (ridade arv)	1	2	
1013 (veergude tüübid)	2	3	
1016 (ühendamine ei tee katki)	3	3	
1015 (lehe värskendamine)	4	5	
1017 (tõeväärtustüüpi veerud)	5	3	25.03.2020
1018 (tõstutundlikud nimed)	6	2	
1020 (SQLi kirjutamine)	7	5	
1019 (ridade värvimine)	8	1	
1011 (liit-tingimused)	9	8	08.04.2020
1014 (alampäringud WHERE klauslis)	10	8	
1000 (HAVING)	11	8	22.04.2020
1001 (LIMIT)	12	2	
1005 (UNION, UNION ALL)	13	8	
1006 (INTERSECT)	14	3	
1007 (EXCEPT)	15	3	06.05.2020
1004 (OVER, PARTITION BY)	16	13	
1003 (CTE)	17	40	
1002 (rekursiivsed päringud)	18	20	
1010 (INSERT)	19	20	
1008 (UPDATE)	20	20	
1009 (DELETE)	21	20	

6.4 SQL lausete koostamise võimalused võrreldavates päringute koostamise programmides

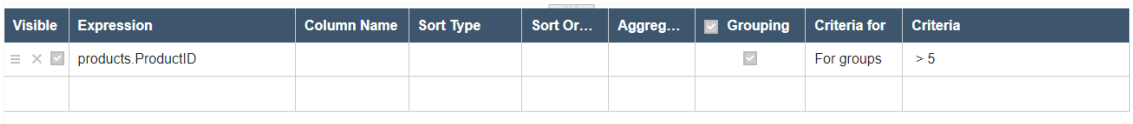
Peale seda, kui väljalaske plaanis pandi paika käesoleva töö tulemusena valmivas väljalaskes realiseeritavad SQL lausete koostamisega seotud funktsionaalsused, on põhjust uurida, kas ja kui hästi on see funktsionaalsus realiseeritud programmides, mida kirjeldati jaotises 2.3. See võib anda ideid, kuidas sellist funktsionaalsust käesolevas rakenduses realiseerida ning samuti näitab, kui levinud on tugi lausekonstruktsioonidele, mida käesolev töö käsitleb. Tabel 6 esitatakse SQL lausekonstruktsioonide toe lisamist nõudvad kasutuslood. Iga võrreldava rakenduse puhul öeldakse Jah/Ei vormis kas vastav rakendus võimaldab seda või mitte. Peale tabelit esitatakse näiteid nende rakenduse kasutajaliidestest.

Tabel 6. Toetatud lausekonstruktsioonide võrdlustabel uuritud rakenduste põhjal.

	Skyvia (2020.4.22)	**Active (3.6.1.401)**	**Dev Express (v2020 vol 1.3)**	**Datapine (2020)**	**Apex (4.2.0.00.27)**
1014 (alampäringud WHERE klauslis)	Ei	Ei	Ei	Ei	Ei
1000 (HAVING)	Ei	**Jah**	Ei	Ei	Ei
1001 (LIMIT)	Ei	**Jah**	**Jah *** (toetatud TOP klausel, kuid pole päris sama, mis LIMIT)	**Jah**	Ei
1005 (UNION, UNION ALL)	Ei	**Jah**	Ei	Ei	Ei
1006 (INTERSECT)	Ei	Ei	Ei	Ei	Ei
1007 (EXCEPT)	Ei	Ei	Ei	Ei	Ei

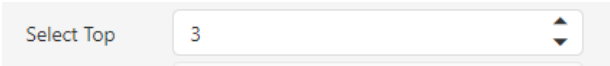
Koondandmeid piirava tingimuse lisamist (*HAVING)* võimaldas uuritud rakendustest ainult Active Query Builder. Selle realiseerimiseks on kaks võimalust, näiteks valida veerg, lisada sellele grupeeringu märgistus ning seejärel määrata kriteerium, mille alusel piirata ning lisaks tuleb valida ka et soovitakse piirata gruppe (Joonis 29). Teiseks

võimaluseks on sisestada kriteerium ning kasutada lihtsalt agregeerimisfunktsiooni, mida valides tekib automaatselt koondandmeid piirav tingimus.



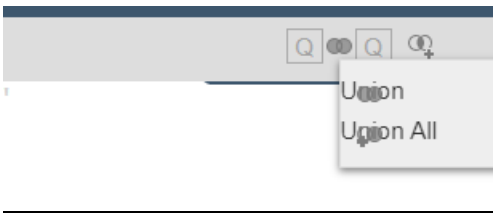
Joonis 29. Koondandmeid piirava tingimuse kasutamine rakenduses *Active Query Builder.*

Päringu ridade arvu piiramist pakub rakendus DevExpress. Tulemuse piiramist tehakse kasutades *TOP* klauslit (Joonis 30), mis on kasutusel SQL Serveri ja MS Access andmebaasisüsteemides. Kuigi ka selle abil saab piirata ridade arvu pole see päris sama mis LIMIT, sest näiteks TOP 1 tulemuses koos sorteerimisega võib olla ka rohkem kui üks rida (leitakse read, milles on sorteerimise tulemusena esimesel kohal olev väärtus – selliseid ridu võib olla mitu).



Joonis 30. Ridade arvu piiramine rakenduses *DevExpress Query Builder*.

Erinevate päringute ühendi (*UNION, UNION ALL*) leidmise võimalust pakub uuritud rakendustest ainult Active Query Builder. Ühendi leidmiseks tuleb lisada navigatsioonirealt uus ühendi tüüpi päring, mille tulemusel tekivad ikoonid, kus ühendi operatsiooni ikoon esitatakse Venni diagrammina [66]. Kahe päringu vahel asuvale Venni diagrammi ikoonile vajutades saab määrata, kas soovitakse kasutada *UNION* või *UNION ALL* operaatorit (Joonis 31). Vajutades lisatud päringu ikoonile/nupule, kuvatakse tühi ala, kus saab päringut graafiliselt koostada.



Joonis 31. Päringu lisamine ühisosa leidmiseks rakenduses *Active Query Builder.*

Autori hinnangul pakub Active Query Builder küll uuritud rakendustest kõige rohkem võimalusi, kuid on ka kõige rohkem segadust tekitava ja arusaamatu kasutajaliidesega.

Kokkuvõttes võib öelda, et käesoleva tööga *Postgres Visual Query Builder* vahendisse lisatavad võimalused SQL SELECT lausete koostamiseks tõstavad selle vahendi teiste

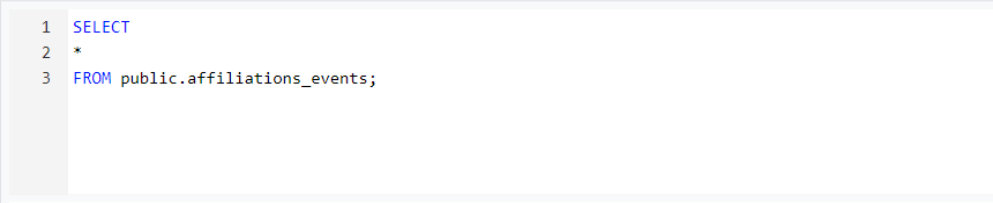
võrreldavatest vahendite seast esile. Nende konstruktsioonide visuaalse esitamise kohta on eeskujuks võtta vähe näiteid. Samas saaks selle töö tulemus olla eeskujuks teistele sarnaste vahendite arendajatele.

7 Arendatava tarkvara vastavus kasutajaliidese disainimustritele

Selles peatükis hinnatakse, kuidas vastab Dzotsenidze poolt loodud rakendus (vt peatükk 5) peatükis 5 välja toodud disainimustritele ning kirjeldatakse lähemalt, kuidas ja kui suures ulatuses on neid mustreid selles rakenduses kasutatud. Alapeatükkidena tuuakse ükshaaval välja iga eelnevalt kirja pandud disainimuster ning nende realiseerimise kirjeldus ning ekraanipilt mustri rakendamisest (mustri olemasolul). Kui rakenduses on ühte mustrit rakendatud mitmes kohas siis tuuakse näitena välja ekraanipilt ühe näite kohta (kui visuaalselt on mustri kasutuste vahel märkimisväärseid erinevusi, siis ka rohkem). Hindamiseks kasutatakse kuue-palli skaalat (hinded 0–5). Skaala elementide defineerimisel on võetud aluseks Tallinna Tehnikaülikooli õppekorralduse eeskiri paragrahv 16 [67]. Hinnete täpne sõnastus on esitatud Lisa 1. Hinded on pandud autori poolt tema subjektiivsete arvamuste alusel. Mustrite peatükid järjestatakse täiendatava rakenduse hinde alusel kasvavalt (alustatakse mustrist, millele täiendatav rakendus autori hinnangul kõige vähem vastab). Samuti kirjutatakse sellest, kuidas nimetatud mustreid kasutatakse ära töös plaanitava uue funktsionaalsuse realiseerimiseks ning tuuakse võimalusel välja ka ekraanipildid lahendusest. Samuti hinnatakse selle sama skaala alusel täiendatud rakendust. Mustrite kirjeldus on peatükis 5.

7.1 Otsene redaktor

- **Täiendatava rakenduse hinne:** 0
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses puudub võimalus genereeritud SQL koodi tekstiredaktori abil muuta.
- **Rakenduse täiendused:** Võetakse kasutusele muudetava kõrgusega, reanumbritega ja sulgudepaare visuaalselt esiletoov tekstiredaktor. Tekstiredaktorit saab kasutada koostatud SQL koodi parandamiseks ning manuaalseteks päringute kirjutamiseks (Joonis 32).



Joonis 32. Otsese redaktori disainimustri kasutamise näide *Postgres Visual Query Builder* teises versioonis.

7.2 Akordionmenüü

- **Täiendatava rakenduse hinne:** 0
- **Täiendatud rakenduse hinne:** 0
- **Täiendatav rakendus:** Rakenduses puudub akordionmenüü komponent. Üldjuhul kasutatakse visuaalsete päringute loomise rakendustes akordionmenüüd tabelite ning tabeli veergude valimiseks.
- **Rakenduse täiendused:** *Postgres Visual Query Builder* puhul kasutakse tabelite esitamiseks nuppe. Autor leiab, et tegelikult on rakenduses *Postgres Visual Query Builder* tabeli valimine hetkel isegi paremini realiseeritud, sest akordionmenüüga võivad kaasneda ka probleemid (vt jaotis 7.2). Seega ei plaanita akordionmenüü mustrit kasutada.

7.3 Koopiakast

- **Täiendatava rakenduse hinne:** 2
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on olemas koopiakast, kust saab formaati säilitava SQL koodi kopeerida. Probleemiks on see, et kogu koopiakasti sisu kopeerimine on küllaltki tülikas. Näiteks ei saa seda teha klahvikombinatsiooniga *CTRL + A*, ilma, et kogu lehe sisu ei valitaks. Samuti ei ole koopiakast visuaalselt väga esiletõusev ning teistest komponentidest eristatav. Tundub nagu oleks tegemist mingisuguse tekstiga teist värvi taustal (Joonis 33).
- **Rakenduse täiendused:** Võetakse kasutusele koopiakast, mis on ühtlasi ka tekstiredaktor. Lisaks võimaldab uus koopiakast genereeritud SQL koodi

klahvikombinatsiooniga *CTRL+A* mugavalt kopeerida. Samuti on uus koopiakast visuaalselt paremini esile toodud (Joonis 32).

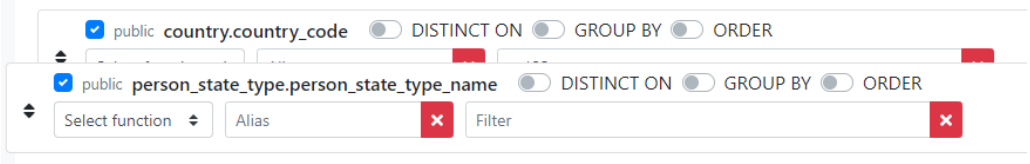
```
SELECT
country.country_code
FROM public.country
WHERE (country.country_code = 'EST');
```

Joonis 33. Koopiakasti disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis.

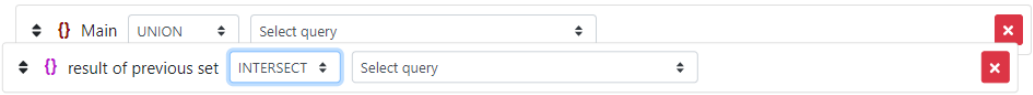
7.4 Pukseeri

- **Täiendatava rakenduse hinne:** 3
- **Täiendatud rakenduse hinne:** 3
- **Täiendatav rakendus:** Rakenduses on rakendatud pukseerimise võimalust kahes kohas, mille abil saab tõsta ümber päringus olevate veergude või ühenduste järjekorda (vajutades hiirega veeru või ühenduse komponendil ning lohistades selle üle olemasolevate komponentide). Autori arvates on sellise pukseerimise otstarve veeru otsingutingimuste (mis on veeruga ühises visuaalses kogumis) puhul arusaamatu. Selle lahenduse abil saab küll veergude järjekorda vahetada (kuvades tulemuste veerge ka vastavalt), kuid erinevate veergude alusel koostatud otsingutingimused ühendatakse kõik *AND* operaatoriga. (Joonis 34).

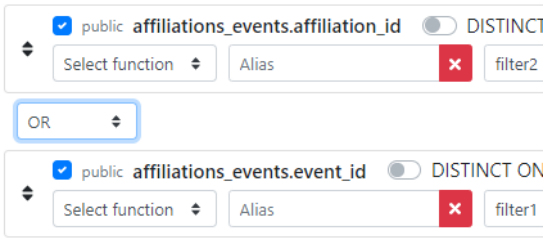
- **Rakenduse täiendused:** Pukseerimise võimalus võetakse sarnaselt ühendamise (*JOIN)* komponendile kasutusele ka hulga (*SET)* operatsioonide ümberjärjestamisel (Joonis 35). Samuti lisatakse olemasolevate veergude iga kahe järjestikuse otsingutingimuse vahele operaatori valikukast, mis annab filtrite ümberjärjestamiseks parema võimaluse (Joonis 36).



Joonis 34. Pukseerimise disainimustri kasutamise näide veergude ümberjärjestamiseks *Postgres Visual Query Builder* esimeses versioonis.



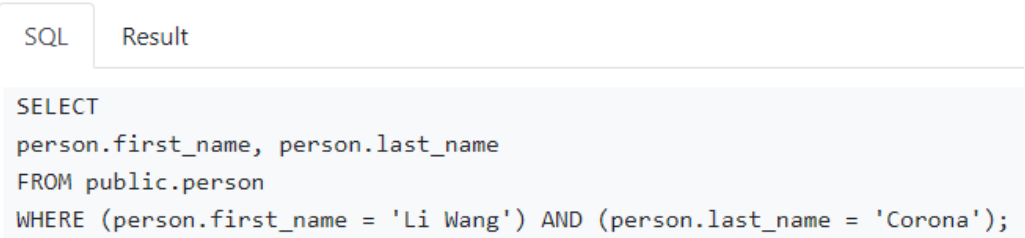
Joonis 35. Pukseerimise disainimustri kasutamise näide hulgaoperatsioonide ümberjärjestamiseks *Postgres Visual Query Builder* teises versioonis.



Joonis 36. Pukseerimise disainimustri kasutamise näide veergude ümberjärjestamiseks koos operaatori valikukastiga *Postgres Visual Query Builder* teises versioonis.

7.5 Eelvaade

- **Täiendatava rakenduse hinne:** 4
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on olemas eelvaade, mis näitab pärast iga graafilist muudatust, kuidas genereeritav SQL kood muutub. Miinusena saab välja tuua, et eelvaatel ei ole värvidega eristatud SQL keele klausleid/märksõnu, mis annaks visuaalselt parema ülevaate päringust (Joonis 37).
- **Rakenduse täiendused:** Täiendusena tuuakse eelvaate puhul värvidega esile ka SQL lausete klauslid/märksõnad (Joonis 38).



Joonis 37. Eelvaate disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis.

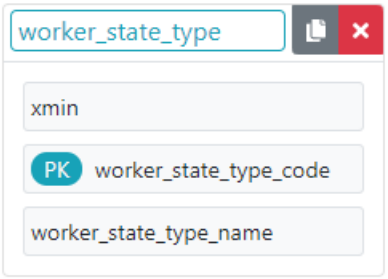
SQL Result (0)

```
1  SELECT
2  addresses.street
3  FROM public.addresses
4  WHERE (addresses.street = 'Wuhan Street');
```

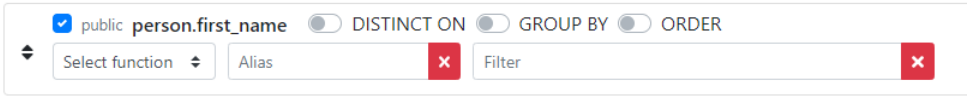
Joonis 38. Eelvaate disainimustri kasutamise näide *Postgres Visual Query Builder* teises versioonis.

7.6 Kaardid

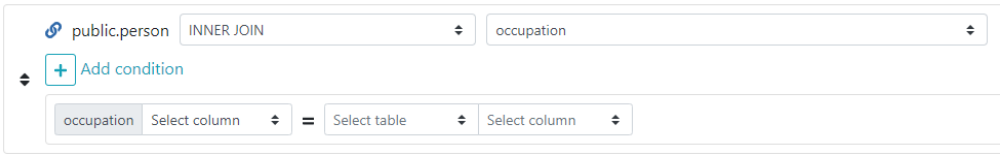
- **Täiendatava rakenduse hinne:** 4
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on kasutatud kaarte kolme komponendi jaoks. Esiteks tabelite nimede ja veergude ühtse komponendina kuvamiseks ning veergude valimiseks (kahjuks ei ole aga veergude puhul visuaalselt välja toodud veeru tüüpe, mis aitaks tabeli veergudest kiiremini aru saada) (Joonis 39). Teiseks veergudele rakendatava funktsionaalsuse kuvamiseks (Joonis 40) ning kolmandaks ka ühendamisoperatsiooni funktsionaalsuste kuvamiseks (Joonis 41).
- **Rakenduse täiendused:** Tabelite nimede ja veergude ühtse komponendina kuvamiseks mõeldud kaardil tuuakse ikoonide kujul välja ka veergude tüübid (Joonis 42).



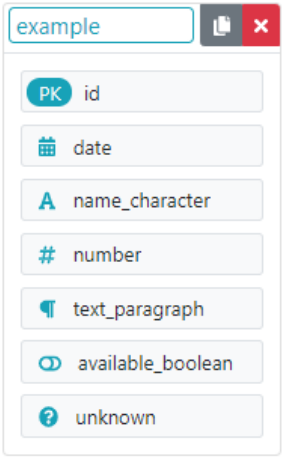
Joonis 39. Kaartide disainimustri kasutamise näide tabeli nime ja veergude kuvamiseks *Postgres Visual Query Builder* esimeses versioonis.



Joonis 40. Kaartide disainimustri kasutamise näide veerule rakendatava funktsionaalsuse kuvamiseks *Postgres Visual Query Builder* esimeses versioonis.



Joonis 41. Kaartide disainimustri kasutamise näide ühendamisoperatsiooni funktsionaalsuste kuvamiseks *Postgres Visual Query Builder* esimeses versioonis.

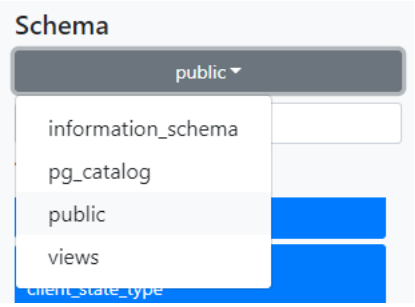


Joonis 42. Kaartide disainimustri kasutamise näide tabeli nime ja veergude kuvamiseks *Postgres Visual Query Builder* teises versioonis.

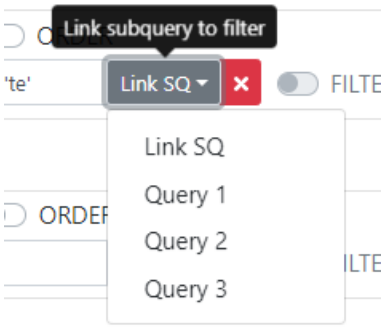
7.7 Vertikaalne rippmenüü

- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on kasutusel vertikaalne rippmenüü. Rippmenüüd on kasutatud andmebaasi skeemi valikul (Joonis 43).
- **Rakenduse täiendused:** Vertikaalne rippmenüü võetakse kasutusele alampäringute linkimiseks/ühendamiseks päringu tingimusega (Joonis 44) ja samuti kõikide päringute ja alampäringute kuvamiseks (Joonis 45). Teisel juhul võetakse rippmenüü kasutusele ühetasemeliselt, aga realiseeritakse selliselt, et

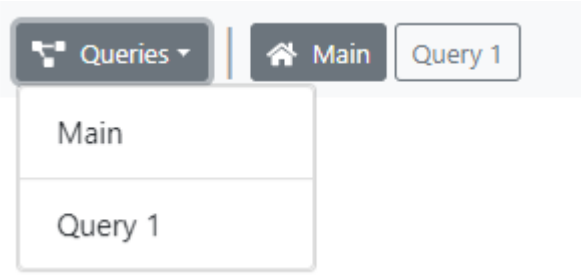
rakenduse edasiarendamisel saab ka alampäringuid mitmetasemelises hierarhias kuvada.



Joonis 43. Vertikaalse rippmenüü disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis.



Joonis 44. Vertikaalse rippmenüü disainimustri kasutamise näide alampäringu linkimiseks päringu tingimusega *Postgres Visual Query Builder* teises versioonis.



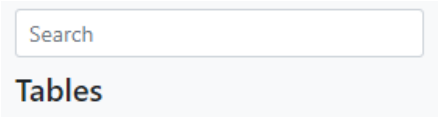
Joonis 45. Vertikaalse rippmenüü disainimustri kasutamise näide kõigi hetkel koostatud päringute kuvamiseks *Postgres Visual Query Builder* teises versioonis.

7.8 Sisestusviip

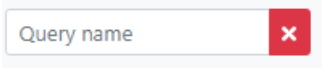
- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5

- **Täiendatav rakendus:** Rakenduses on olemas neli sisestusviipa/kohahoidjat. Sisestusviipa on kasutatud andmaks märku tabelite otsingu võimalusest (Joonis 46). Samuti kasutatakse seda märguandeks ning selgituseks veeru varjunime sisestamisel, otsingutingimuse sisestamisel ning ka tabeli varjunime sisestamisel. Hea on see, et kursori fookuse viimisel sisetusviibale jääb sisestusviip esialgu alles.

- **Rakenduse täiendused:** Sisestusviip võetakse kasutusele andmaks märku alampäringu nime muutmise võimalusest (Joonis 47).



Joonis 46. Sisestusviiba disainimustri kasutamise näide tabelite otsimiseks *Postgres Visual Query Builder* esimeses versioonis.

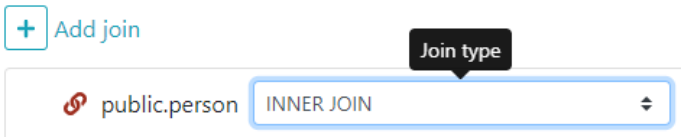


Joonis 47. Sisestusviiba disainimustri kasutamise näide päringu nime muutmiseks *Postgres Visual Query Builder* teises versioonis.

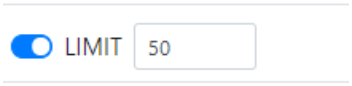
7.9 Head vaikimisi valikud

- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on kasutusel ühendamisoperatsiooni lisamisel hea vaikimisi valik *INNER JOIN* (Joonis 48). See on lihtsaim ja levinuim ühendamise tüüp.

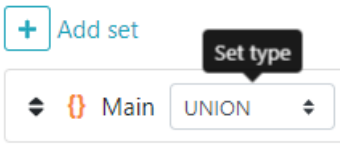
- **Rakenduse täiendused:** Võetakse kasutusele uus hea vaikimisi valik päringu ridade arvu piiramise operatsiooni jaoks. Autori arvates on 50 optimaalne arv, mida saab vajadusel kasutaja ise suurendada või vähendada (Joonis 49). Samuti võetakse hulgaoperatsiooni lisamisel kasutusele hea vaikimis valik *UNION* (Joonis 50). Autori arvates on see üks levinumaid hulgaoperatsiooni tüüpe, mida tuuakse ka erinevates SQL keele hulgaoperatsioonide kohta käivates materjalides esimesena välja.



Joonis 48. Heade vaikimisi valikute disainimustri kasutamise näide ühendamisoperatsiooni valimisel *Postgres Visual Query Builder* esimeses versioonis.



Joonis 49. Heade vaikimisi valikute disainimustri kasutamise näide ridade arvu piiramise operatsiooni kasutamisel *Postgres Visual Query Builder* teises versioonis.



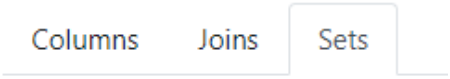
Joonis 50. Heade vaikimisi valikute disainimustri kasutamise näide hulgaoperatsiooni valimisel *Postgres Visual Query Builder* teises versioonis.

7.10 Modulaarsed vahekaardid

- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on erinevate päringuga seotud sektsioonide kuvamiseks kasutusel modulaarsed vahekaardid. Modulaarseid vahekaarte on rakendatud kahel korral, veergude ning ühendamisoperatsioonide kuvamiseks (päringu moodul) ja genereeritud SQL koodi ning päringutulemuse kuvamiseks (tulemuste moodul) (Joonis 51).
- **Rakenduse täiendused:** Olemasoleva päringu mooduli juures võetakse kasutusele uus vahekaart (*Sets*), millele vajutades kuvatakse hulgaoperatsioonidega seotud sisu (Joonis 52).

(a) (b)

Joonis 51. Modulaarsete vahekaartide disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis: (a) veerud ja ühendamised, (b) SQL kood ja päringu tulemus.



Joonis 52. Modulaarsete vahekaartide disainimustri kasutamise näide päringu moodulis *Postgres Visual Query Builder* teises versioonis.

7.11 Lehekülgedeks jagamine

- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on päringu tulemuste erinevate osade/lehekülgede sirvimiseks kasutusel lehekülgedeks jagamise komponent. See võimaldab liikuda nii järgimisele, eelmisele kui ka vabalt valitud leheküljele. Samuti näidatakse ära kogu lehekülgede arv ning komponent võimaldab määrata, mitu tulemuste rida korraga kuvatakse (Joonis 53).
- **Rakenduse täiendused:** Lehekülgedeks jagamise osas muudatusi ei tehta.

Joonis 53. Lehekülgedeks jagamise disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis.

7.12 Veeru järgi sorteerimine

- **Täiendatava rakenduse hinne:** 5
- **Täiendatud rakenduse hinne:** 5
- **Täiendatav rakendus:** Rakenduses on olemas võimalus päringu tulemustes olevaid veerge sorteerida. Sorteerimise suunast annab märku visuaalne tumedam joon. Ekraanipildil sorteeritakse veergu *country_code* tähestiku järgi kahanevalt (alustades tagantpoolt) (Joonis 54).

- **Rakenduse täiendused:** Veergude sorteerimise osas muudatusi ei tehta.

#	country_code
3	LAT
2	FIN
1	EST

Joonis 54. Veeru järgi sorteerimise disainimustri kasutamise näide *Postgres Visual Query Builder* esimeses versioonis.

Rakenduse *Postgres Visual Query Builder* esimese versiooni mustritele vastavuse keskmine hinne oli autori hinnangul 3.91 ning uue versiooni keskmine hinne 4.82 (mõlema arvutuse puhul ei võetud arvesse akordionmenüü disainimustrit*)*. Tuleb silmas pidada, et see on siiski subjektiivne hindamine.

8 Muudatused SQL lausete koostamise programmis

See peatükk kirjeldab muudatusi, mida tehti uue *Postgres Visual Query Builder* versiooni loomisel nii kasutajaliideses (eesrakenduses) kui ka tagarakenduses. Kõigepealt tuuakse punktide kaupa välja realiseeritud funktsionaalsused ja nendest tulenevad muudatused kasutajaliidese väljanägemises. Seejärel tuuakse välja muudatused tagarakenduses ning viimaseks muudatused eesrakenduse realisatsioonis. Eesrakenduse realisatsiooni puhul tuuakse lisaks tabelite kujul välja uued ja muudetud tegevuste loojad (*action creators)*, lao muudatused (*store)*, ülekandjad (*reducers)* ja uued komponendid ning vanade muudatused.

8.1 Muudatused funktsionaalsuses ja kasutajaliideses

Järgnevalt tuuakse välja realiseeritud funktsionaalsused punktide kaupa ning kirjeldatakse, milliseid otsuseid iga ülesande puhul tehti. Muudatusi illustreeritakse ekraanipiltidega. Ekraanipildid esitatakse ainult nende muudatuste kohta, mida mustrite põhjal valideerimise puhul (vt peatükk 7) eraldi välja ei toodud. Muudatused, mis on seotud kasutajaliidese mustrite (vt peatükk 5) paremaks kasutusele võtmiseks, on esitatud peatükis 7.

8.1.1 Kiirülevaade päringu tulemuste mahust

Tulemuste mahtu otsustati näidata otse päringu tulemuste vahekaardil (Joonis 56), sest siis näeb kasutaja kohe pärast päringu käivitamist, kas tulemusi leiti või mitte (olenemata sellest, kas vahekaart on avatud või mitte).

8.1.2 Veeru tüüpi iseloomustava sümboli kuvamine

Veeru tüüpi iseloomustav sümbol otsustati lisada, sarnaselt olemasolevale privaatvõtme sümbolile, kaardil asuva veeru nime ette (Joonis 42). Sümbolite valikul lähtuti uuritud rakendustes kasutuselolevatest sümbolitest ning lisaks püüdis autor valida kasutatavast sümbolite teegist veeru tüüpi enim iseloomustava sümboli.

8.1.3 Tabeli päringust kustutamise võimalus pärast ühendamise operatsiooni eemaldamist

Selle võimaluse realiseerimine nõudis koodibaasis muudatuse tegemist ja lisa kontrolltingimuse lisamist ühendamise komponendis, mis kontrolliks, et päringu olekus olev tabelite list poleks tühi.

8.1.4 Andmebaasiühenduse säilitamine pärast lehekülje värskendamist

See ülesanne otsustati lahendada nii, et kogu rakenduse olekut säilitatakse veebilehitseja sessiooni mälus. See tähendab, et kasutaja loodud päringud säilivad seni, kuni ta andmebaasiga ühenduse katkestab või veebilehitseja sulgeb.

8.1.5 Tõeväärtusi omavate ridade kuvamine

Tõeväärtuste kuvamiseks oli vaja teha koodibaasis muudatusi, et ka selliseid väärtusi tulemustes kuvatakse.

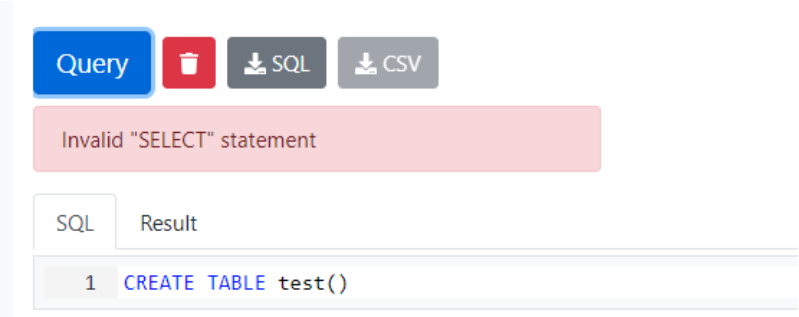
8.1.6 Päringu loomise võimalus ka kaamliküüru stiilis nimega veergudega

Muudatusi tuli teha koodibaasis, täpsemalt päringu koostamise funktsioonis.

8.1.7 SQL päringu käsitsi sisestamine ning muutmine

Selle ülesande lahendamiseks prooviti kõigepealt ise luua tekstivälja komponent, mis võtab graafilise päringu abil loodud SQL koodi ning kuvab seda tekstiväljas ning lubab ka hiljem muuta. Hiljem leiti, et selleks on olemas ka sobilikud teegid. Kõigepealt prooviti Monaco Editori teeki [68], aga see ei võimaldanud nii palju kohandamist ega funktsionaalsust, kui autor oleks soovinud. Selle tõttu võeti kasutusele CodeMirror (vt jaotis 3.2.2), mille funktsionaalsust ja välimust ka autori poolt kohandati.

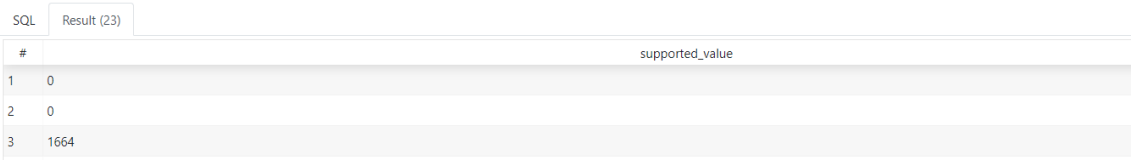
Samuti tuli arvestada sellega, et ei oleks lubatud SQL süstimine ja keelatud käskluste (näiteks andmekirjelduse laused) käivitamine. Selleks loodi eraldi validaator (analoogselt esimeses versioonis tehtud valideerimisele), mis kontrollib keelatud käskluste olemasolu. Illegaalse päringu käivitamisel kuvatakse kasutajale veateade (Joonis 55).



Joonis 55. Keelatud käskluse veateade *Postgres Visual Query Builder* teises versioonis.

8.1.8 Päringu tulemuse ridade kuvamine vahelduvate värvidega

Selle ülesande lahendamiseks tuli teha muudatusi koodibaasis, täpsemalt rakenduses kasutatava tulemuste tabeli komponendis (Joonis 56).



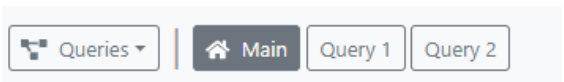
Joonis 56.Vahelduvate värvidega tulemuste tabel *Postgres Visual Query Builder* teises versioonis.

8.1.9 Liit-otsingutingimuse koostamise võimalus läbi otsingutingimuste kombineerimise

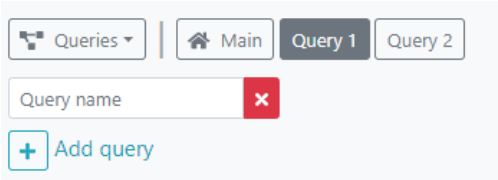
Selle ülesande puhul tuli välja mõelda, kuidas oleks seda olemasoleva kasutajaliidese puhul kõige mõistlikum lahendada. Rakenduses on iga veeru jaoks oma filter e piirang e otsingutingimus ning iga veerg on eraldi komponent. Selle tõttu otsustati teha nii, et kui lisada pärast esimest tingimust veel üks tingimus, siis kasutajaliideses ilmub kahe veeru vahele väike vertikaalne rippmenüü, kust saab valida kahte tingimust ühendava loogikaoperaatori (Joonis 36). Alternatiiviks, mis oleks nõudnud rakenduse päris palju ümbertegemist, oli eemaldada filtri kastid veergude juurest ning lisada veergude kohal asuvale alale kogu päringut mõjutava filtri koostamist võimaldav visuaalne komponent. Filtri komponent oleks sarnane olemasolevale, aga selle komponendi lõpus oleks väike nupp, millele vajutades ilmuks rohkemate võimalustega modaalaken, nii-öelda *„filter builder/editor“*. Selle modaali sees saaks ehitada liitotsingutingimust läbi erinevate liht-otsingutingimuste lisamise. Realiseeritud lahenduse puuduseks on, et kui kasutaja soovib määrata tingimuste rakendamise järjekorra, siis tuleb vastavalt vajadusele lisada vajalikud sulud genereeritud SQL lauset sisaldavasse otsesse redaktorisse või piirangu kastide algusesse või lõppu.

8.1.10 Alampäringud *WHERE* klauslis

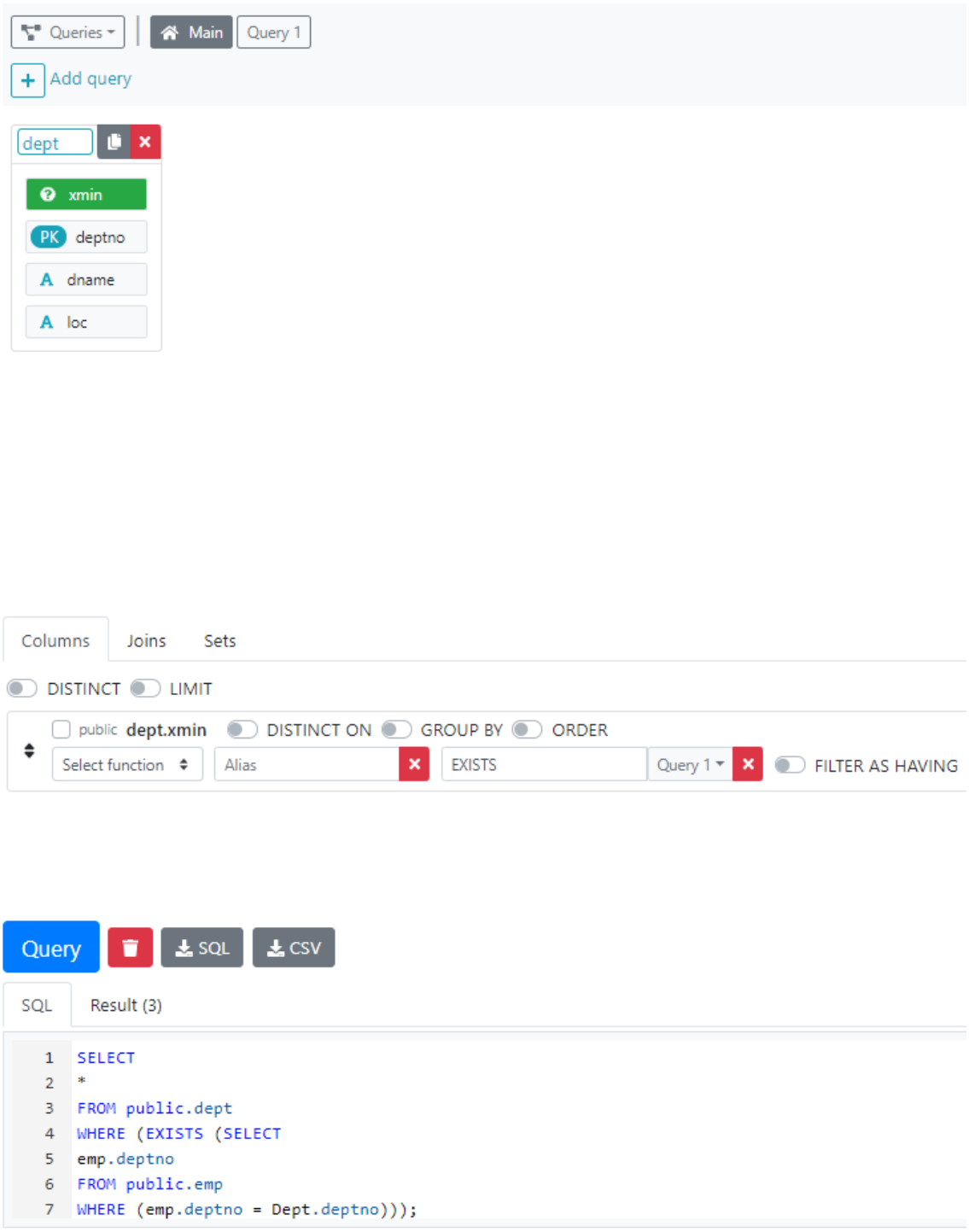
Alampäringute kasutamist *WHERE* klauslis ei paku mitte ükski jaotises 2.3 uuritud rakendustest. Seega ei olnud selle ülesande lahendamisel kuskilt eeskuju võtta ja kõigepealt tuli välja mõelda, kuidas oleks mõistlik rakenduses alampäringut koostada nii, et oleks võimalik kasutada ära olemasolevat funktsionaalsust ja et alampäring oleks taaskasutatav. Samuti tuli arvestada olemasoleva tarkvara arhitektuuriga. Murekohaks oli see, et olemasolev rakendus oli loodud ainult ühe päringu vaate jaoks. Seega tuli üsna pikalt kaaluda ja proovida variante rakenduse laiendamiseks alampäringute jaoks, et kõike mitte ümber kirjutada. Üks variant oleks olnud muuta kogu rakenduse lao olekut (päringu laos oleks ühe päringu asemel hoitud kõikide päringute olekuid). See oleks nõudnud praktiliselt kogu rakenduse muutmist. Teine variant oli luua olemasoleva lao juurde järjend, mille sees teisi päringuid hoida. See oleks tähendanud rekursiivselt päringute vahel liikumist. Lõpplahendusena otsustati luua hoopis uus päringute ülekandja, mis hoiab oma laos kõiki päringuid, mis pole aktiivsed päringud. Visuaalseks realiseerimiseks tundus kõige mõistlikum võtta kasutusele navigatsiooniriba (Joonis 57) koos uue päringu lisamise võimalusega. Samuti lisati iga alampäringu jaoks võimalus päringu nime muuta, et päringuid paremini hallata (Joonis 58). Iga päringut kujutatakse navigatsiooniribal nupu komponendina, mis täidab oma olemuselt vahekaardi rolli. Nupule vajutades ilmub samasugune vaade nagu algse/põhipäringu vaate puhul. Päringu ühendamiseks filtriga e otsingutingimusega otsustati lisada filtri külge rippmenüü, millega saab alampäringu filtri külge linkida (Joonis 44). Joonis 59 esitab ekraanipildi sellest, kuidas näeb välja korreleeruva alampäringu loomine. Põhipäringus (aktiveeritud, navigatsioonireal halli taustaga) koostatakse välimine *SELECT* lause, lisatakse piirangu tingimus *EXISTS* ning piirangu kasti paremas otsas lingitakse piirangu külge alampäring. Alampäring on koostatud navigatsioonireal nähtava Päring 1 vaates (vaade analoogne põhipäringule, aga valitud vastavad veerud ja sisestatud vajalikud piirangu tingimused). Genereeritud SQL kood on nähtav ekraanipildi allosas, tekstiredaktoris. Ekraanipildilt on näha ka enamjaolt kogu eestikeelset kasutajaliidest.EXI



Joonis 57. Navigatsiooniriba (aktiivne on põhipäring) *Postgres Visual Query Builder* teises versioonis.



Joonis 58. Navigatsiooniriba koos menüükomponendiga (aktiivne on alampäring *Query 1*) *Postgres Visual Query Builder* teises versioonis.

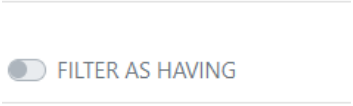


Joonis 59. Korreleeruva alampäringu loomise vaade *Postgres Visual Query Builder* teises versioonis.

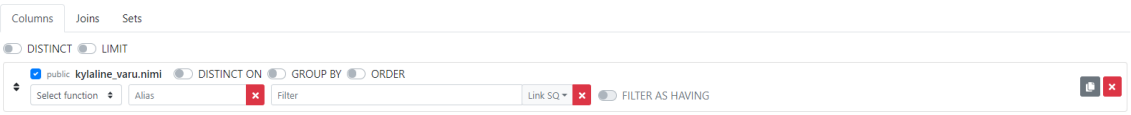
Alles hiljem selgus dokumentatsiooni uurides [69], et alampäringute koostamise võimalus puudub *Active Query Builderi* veebipõhises demoversioonis, kuid on olemas täisversioonis. Huvitaval kombel sarnaneb see antud töös loodud lahendusele.

8.1.11 Koondandmeid piirav tingimus (*HAVING* klausel)

Selle ülesande lahendamisel prooviti jällegi hoida rakendus kompaktsena ning taaskasutada olemasolevaid elemente. Kuna koondandmeid piirav tingimus saab olla veeru küljes (Joonis 61), siis otsustati taaskasutada filtri välja, et mitte luua uuesti sarnast funktsionaalsust. Koondandmeid piirava tingimuse kasutamiseks tuleb see lülitist aktiveerida (Joonis 60) ning piirangu tingimus kirjutada filtri väljale.



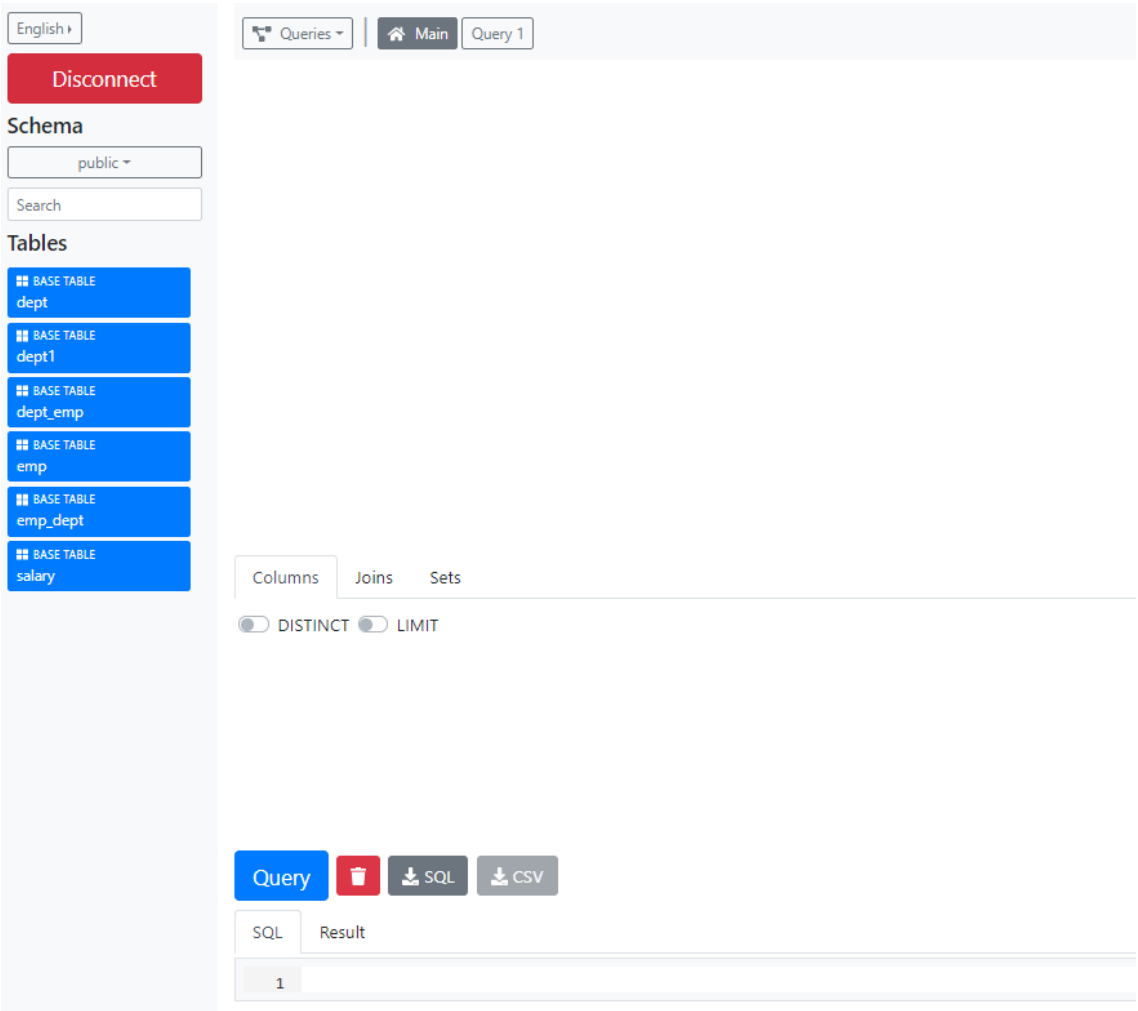
Joonis 60. Lüliti filtri kasutamiseks *HAVING* klausli jaoks *Postgres Visual Query Builder* teises versioonis.



Joonis 61. Veergude vahekaardi vaade *Postgres Visual Query Builder* teises versioonis.

8.1.12 Tagastatud ridade piiramine (*FETCH, LIMIT)*

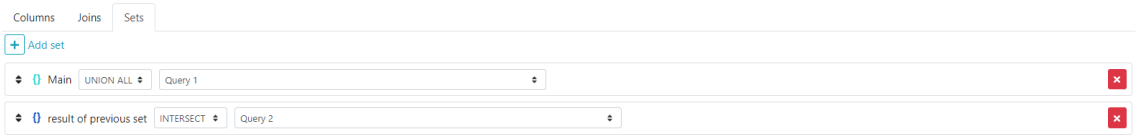
Tulemuses olevate ridade arvu piiramise võimalust otsustati realiseerida lüliti ja numbrilise sisendvälja abil (Joonis 49) ning kuvada neid elemente kohe veergude sektsiooni ülaosas (Joonis 62). Kuna kasutusel olev teek Squel.js pakkus genereerimiseks ainult *LIMIT* klauslit, siis otsustati selle kasuks.



Joonis 62. Päringu loomise vaade *Postgres Visual Query Builder* teises versioonis.

8.1.13 Hulgatöötluse operaatorid (*UNION, UNION ALL, INTERSECT, EXCEPT)*

Esialgselt oli idee paigutada hulgaoperatsioonide valiku võimalus navigatsioonirea menüü külge ning samuti võtta kasutusele päringu tüübi valik. See tähendaks lisa valikukasti loomist, kust saab valida päringu tüübi (näiteks alampäring *WHERE* klausili jaoks või mõni hulgaoperaatori tüüp). Päringu tüübi alusel oleks hiljem vastav SQL kood kokku pandud. Siin tekkisid aga mõningad probleemid – näiteks oleks muutnud see päringute taaskasutamise tülikamaks ja samuti võtaks need navigatsioonireal omakorda ruumi, mis muudaks rakenduse visuaalselt vähem hoomatavaks. Lõpuks otsustati võtta kasutusele uus vahekaart ning esitada sisu sarnaselt ühenduse operaatorite jaoks loodud viisil (Joonis 63).



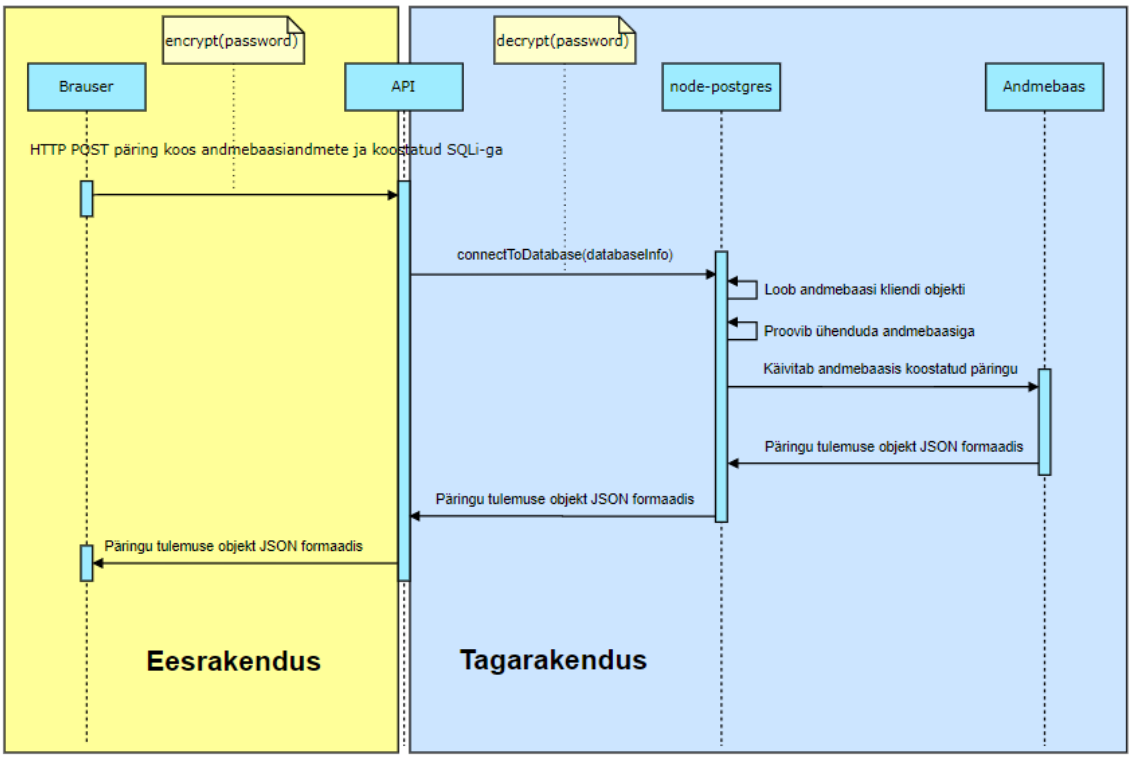
Joonis 63. Hulkade vahekaardi vaade *Postgres Visual Query Builder* teises versioonis.

8.2 Muudatused tagarakenduses

Järgnevates alapeatükkides kirjeldatakse kuidas muutus tagarakenduse arhitektuur, struktuur ning milliseid stiiliparandusi ning muudatusi koodibaasis tehti.

8.2.1 Arhitektuur

Algne idee oli muuta tagarakenduse arhitektuuri selliselt, et kliendi ja andmebaasi omavaheline suhtlus käiks läbi ühenduspuuli (*connection pool)*, mis oleks jätnud ühenduse aktiivseks ning seejärel oleks saanud andmeid küsida kasutades GET-päringut. See oleks aga nõudnud lisaks autoriseerimise realiseerimist, et mitu klienti saaksid korraga ühenduses olla. Selle tõttu otsustati, et praegune lahendus siiski sobib ja seni kuni jõudluse või muid probleeme ei teki, pole vajadust seda muuta. Seega tagarakenduse arhitektuuris suuri muudatusi ei tehtud, küll aga lisati parooli dekrüpteerimine. Krüpteerimine toimub kliendi (eesrakenduse) poolel, sellepärast et varjata POST-päringuga saadetava andmebaasi kasutaja parooli ning selle talletamist veebilehitseja mälus. Võib tekkida küsimus, et milleks see vajalik? Autori hinnangul on võimalikeks rakenduse kasutajateks ka andmebaase õppivad IT-tudengid, kes omavahel arvuteid jagavad. Krüpteering loodi selleks, et osaliselt vältida paroolide leket või andmebaasi kasutajate väärkasutamist. Antud lahendus puhul ei olnud eesmärk tagada rakenduses 100% turvalisust, vaid vältida veebilehitseja kaudu otsest ligipääsu andmebaasi kasutaja paroolile. Joonis 64 esitab eesrakenduse ja tagarakenduse suhtlust kirjeldava jadadiagrammi, kus tagarakenduse moodulite suhtlust on kujutatud sinisel taustal.



Joonis 64. Eesrakenduse ja tagarakenduse suhtlust kirjeldav jadadiagramm.

8.2.2 Struktuur

Tagarakenduse kaustade struktuuri jaoks loodi uus kaust *src* (*source)*. Kõik failid, mis ei ole serveri metaandmeid hoiustavad või konfiguratsioonifailid, tõsteti eraldi *src* kausta. See ühtlustab rakenduse kaustade paigutust (sarnaselt eesrakenduse struktuurile) ning muudab failides orienteerumise lihtsamaks ja kataloogide struktuuri loogilisemaks.

8.2.3 Koodi- ja stiilivigade parandused ning parimate praktikate rakendamine

Koodi- ja stiilivigade leidmiseks/parandamiseks lisati ESLint, mis kasutab Airbnb poolt loodud JavaScripti stiilijuhendit (üks levinumaid stiilijuhendeid JavaScriptis) [70]. Stiili ja koodivigu parandati kokku viie faili puhul.

Kõigis tagarakenduse failides (kokku kuus) asendati olemasolevad import/eksport lahendused uuemate lahendustega. Failis *utils.js* asendati olemasolev anonüümse funktsiooni väljendus noolfunktsiooniga. Failis *query.js* asendati olemasolev asünkroonne lahendus uuemaga.

8.2.4 Uued funktsioonid ning vanade muudatused

Failis *utils.js* loodi uus funktsioon *decrypt()*, mis võtab sisendiks krüpteeritud andmebaasi parooli ning dekrüpteerib selle vastavalt võtmele.

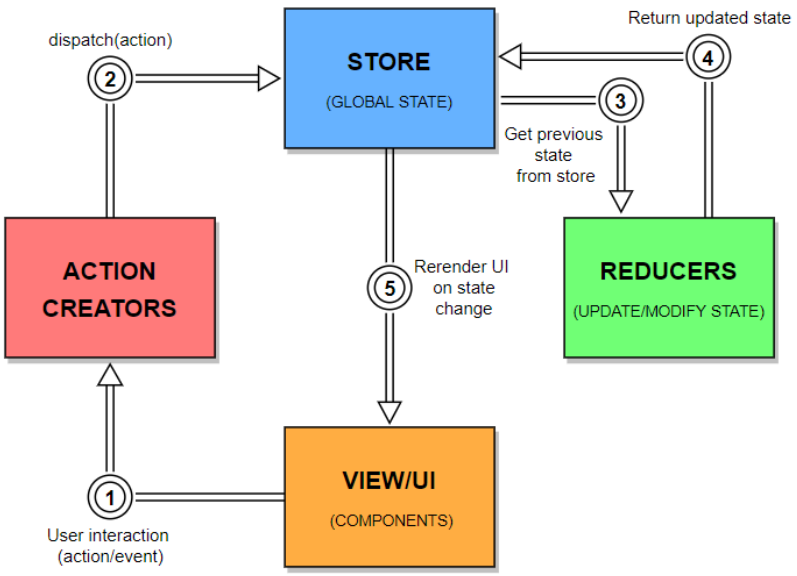
8.3 Muudatused eesrakenduse realisatsioonis

Järgnevates alapeatükkides kirjeldatakse kuidas muutus eesrakenduse arhitektuur, struktuur ning milliseid stiiliparandusi ning muudatusi koodibaasis tehti.

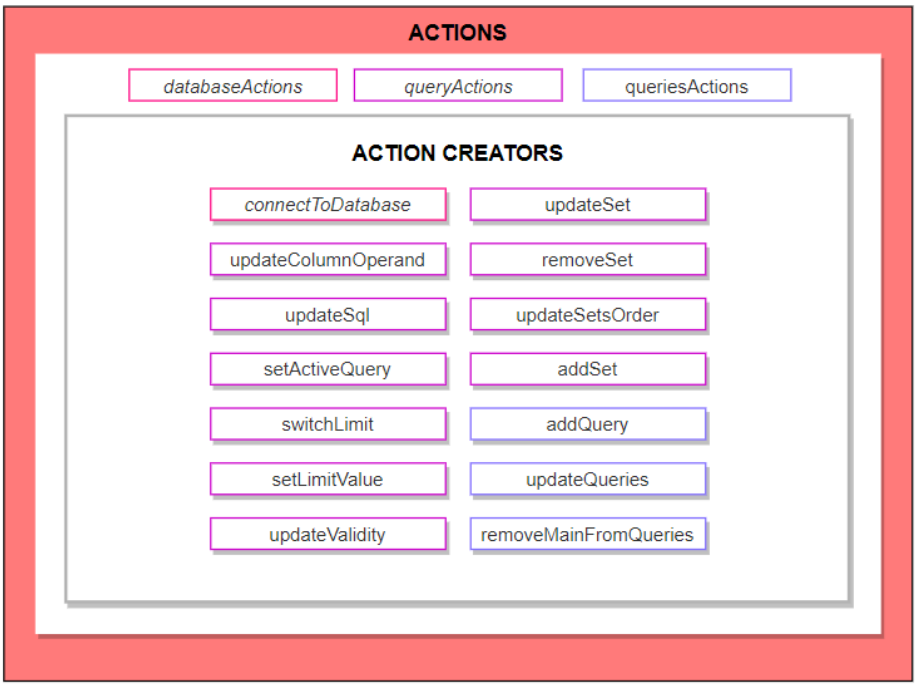
8.3.1 Arhitektuur

Eesrakendus järgib Reduxi arhitektuuri, mis koosneb neljast põhielemendist: laost (*store)*, ülekandjatest (*reducers)*, vaatest/komponentidest (*view)* ja tegevuste loojatest *(action creators)* (Joonis 65). Järgnevalt tuuakse diagrammide kujul iga põhielemendi kohta eraldi välja rakenduses tehtud muudatused (need ei kajasta kogu rakenduse süsteemi, vaid ainult teises versioonis tehtud muudatusi). Diagrammidel on proovitud värvide abil tuua esile nende seoseid.

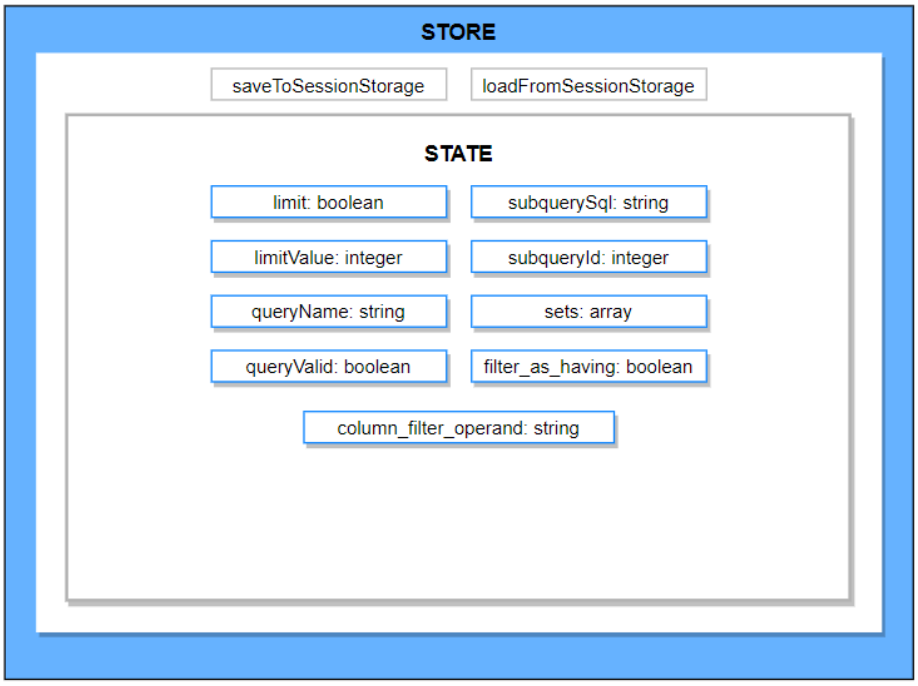
Joonis 66 näitab rakendusse lisandunud ning muudetuid tegevusi ja nende loojaid (detailsema kirjelduse jaoks vt jaotiseid 8.3.6 ja 8.3.7). Joonis 67 toob välja muudatused laos ja selle olekus (detailisema kirjelduse jaoks vt jaotis 8.3.8). Joonis 68 näitab rakendusse lisandunud ning muudetuid ülekandjaid ning neid mõjutavate tegevuste/käskluste tüüpe (detailsema kirjelduse jaoks vt jaotis 8.3.9). Joonis 69 toob välja kasutajaliidese muudetud ning uued komponendid (kujutatud on ainult need komponendid, mille puhul tehti muudatusi rohkem kui funktsionaalseks komponendiks muutmine ja omaduste tüüpide väljatoomine. Detailsema kirjelduse jaoks vt jaotiseid 8.3.10 ja 8.3.11). Noolte ja värvide abil on toodud esile (seal kus võimalik) otsesed või kaudsed emakomponentide *(parent component)* ja laste *(children)* ühendused.



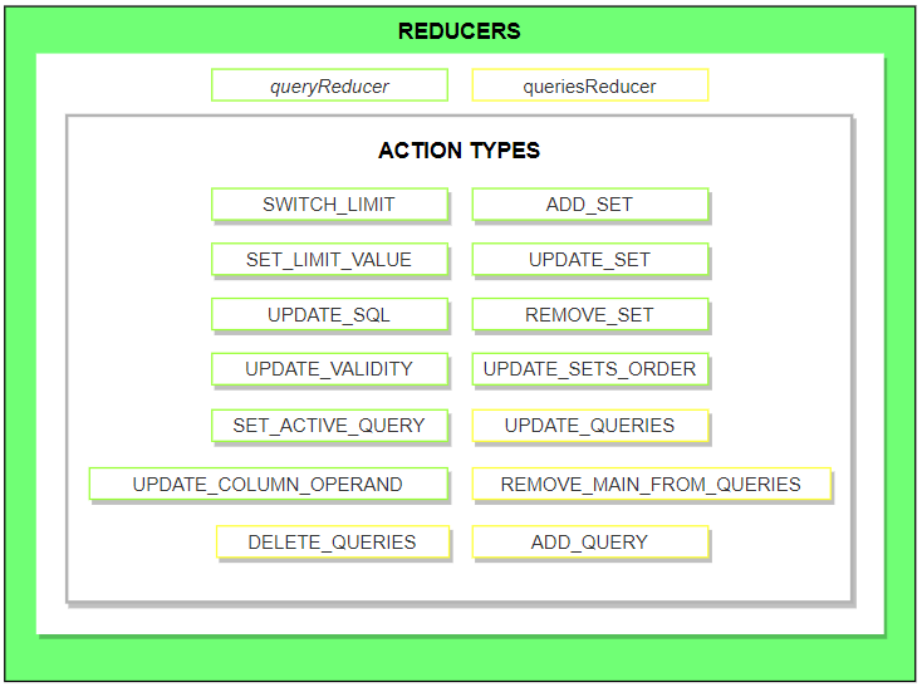
Joonis 65. Reduxi arhitektuur.



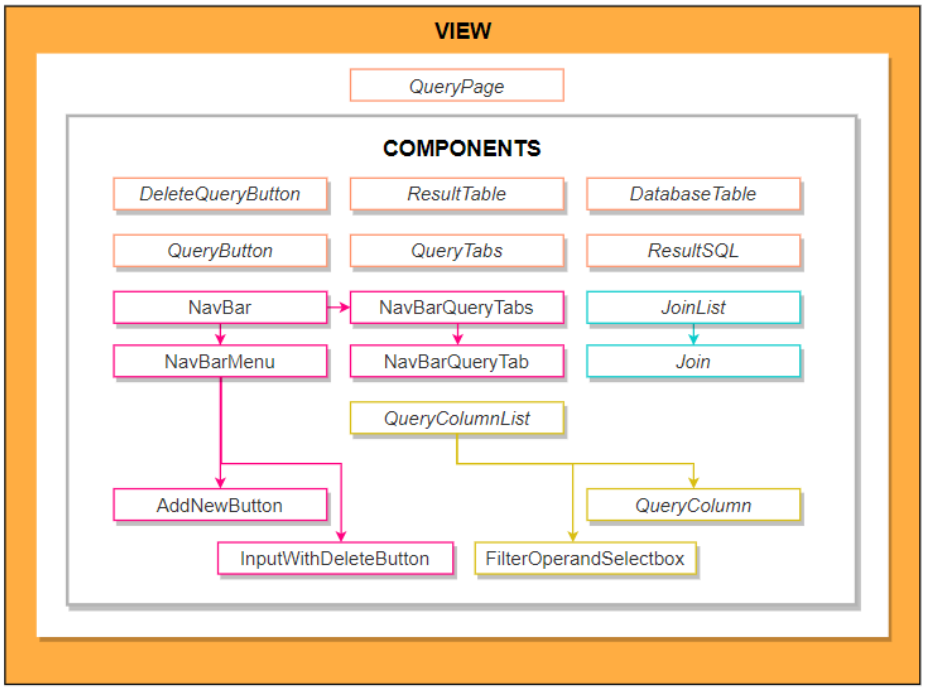
Joonis 66. Muudetud (kaldkirjas) ja uued tegevused ning nende loojad.



Joonis 67. Muudatused laos ja lao olekus.



Joonis 68. Muudetud (kaldkirjas) ja uued ülekandjad ning nende käskluste/tegevuste tüübid.



Joonis 69. Muudetud (kaldkirjas) ja uued kasutajaliidese komponendid.

8.3.2 Struktuur

Eesrakenduse kõikide komponentide ning testide jaoks, kus oli kasutusel JSX (*JavaScript Extensible Markup Language)* süntaks, muudeti faililaiend ka vastavaks, ehk .jsx laiendiks. Samuti nimetati ümber kõik kasutatavad ülekandjate (*reducer*) funktsioonid (kooskõlastades need failinimedega). Selle tõttu tuli ülekandjate kombineerimisel muuta ka nende esitus objektina võtme -väärtuse kujule.

8.3.3 Koodi- ja stiilivigade parandused ning parimate praktikate rakendamine

Koodi- ja stiilivigade leidmiseks/parandamiseks lisati jällegi ESLint, kuid seekord Reacti jaoks mõeldud reeglitega. Reegleid laiendati Airbnb stiilijuhendiga. Stiili- ja koodivigu parandati 24-s Reacti põhi kasutajaliidese komponendis, ühes kõrgema järgu komponendis (*higher-order component)* ning kahes põhimarsruute (*routes)* renderdavas komponendis. Reduxi puhul parandati stiili- ja koodiviguneljas tegevuses (*action)* ja neljas ülekandjas (*reducer)*. Samuti tehti seda ka põhikomponendi failis *App.js* ning *config.js* ning *index.js*. Iga kasutajaliidese komponendi puhul, kus oli kasutatud tegevuste lähetamist omadusteks (*mapDispatchToProps)*, lühendati lähetamise esitust.

Samuti parandati koodi- ja stiilivigu 24-s kasutajaliidese komponendi testis. Reduxi puhul neljas ülekandja testis ning neljas tegevuste testis. Lisaks ka *queryBuilder.test.js* failis.

Lisaks võeti iga omadusega (*property)* kasutajaliidese komponendi puhul kasutusele *React PropTypes*, mille abil toodi välja iga komponendi lõpus tema omaduste tüübid. PropTypes kasutamine on hea praktika, mis annab koheselt ülevaate komponendis kasutavatest omadustest ning nende tüüpidest.

Kõik klassikomponendid, kus ei kasutatud olekut, kirjutati ümber funktsionaalseteks komponentideks. Kokku muudeti funktsionaalseks 19 kasutajaliidese komponenti.

8.3.4 Muutunud eesrakenduse utiliidid

Päringu ehitamise failis *queryBuilder.js* lisati olemasoleva *addColumnsToQuery* funktsiooni uus funktsioon *buildFilter*, mis ehitab vastavalt kasutajaliideses tehtud valikutele SQL lause *WHERE* ja *HAVING* klauslid. Samuti lisati funktsioon *buildSetQuery*, mis ehitab vastavalt kasutajaliideses valitud hulgaoperaatoritele SQL lause. Lisati ka *limit* funktsionaalsus. *DatabaseViewer* kasutajaliidese komponendis olnud staatiline meetod *filterTable* tehti eraldi utiliit-funktsiooniks. Kasutajaliidese komponentide jaoks loodi ka vajadusel uued tõlked.

8.3.5 Uued eesrakenduse utiliit-funktsioonid

Tabel 7 loetleb kõik uued utiliit-funktsioonid ja nende kirjeldused.

Tabel 7. Uued utiliit-funktsioonid *Postgres Visual Query Builder* teises versioonis koos kirjeldusega.

Nimi	**Kirjeldus**
encrypt	Krüpteerib sisendteksti, kasutades aes-256-cbc algoritmi ja tagastab krüpteeritud kujul andmeobjekti.
iconPicker	Võtab sisendiks andmetüübi ja tagastab vastavalt andmetüübile ikooni tüübi.
buildSetQuery	Ehitab vastavalt hulga operaatori tüübile SQL lause.
validateQuery	Valideerib sisendiks antava SQL lause ja tagastab tõeväärtuse.

8.3.6 Muutunud eesrakenduse oleku haldamise tegevused

Andmebaasiga ühenduse loomise tegevuste (*actions)* failis *databaseActions.js* lisati *connectToDatabase* funktsioonis olevale andmeobjektile parooli krüpteering. Lao külge lisati kaks funktsiooni. Nendest üks (*saveToSessionStorage)* võtab rakenduse oleku ja salvestab selle JSON kujul veebilehitseja sessiooni mällu ning teine

(*loadFromSessionStorage)* loeb selle sessiooni mälust ning muudab JavaScripti objektiks. Lao loomisel antakse säilitatud rakenduse olek parameetrina kaasa.

8.3.7 Uued eesrakenduse oleku haldamise tegevuste loojad

Tabel 8 loetleb kõik uued oleku haldamise tegevuste loojad (funktsioonid) ja nende kirjeldused.

Tabel 8. Uued oleku haldamise tegevuste loojad *Postgres Visual Query Builder* teises versioonis koos kirjeldusega.

Nimi	**Kirjeldus**
updateColumnOperand	Saadab laole veeru operaatori uuendamise käskluse koos valitud liht-tingimuse operaatori ja veeru identifikaatoriga ning SQL lause genereerimise käskluse.
updateSql	Saadab laole SQL lause uuendamise käskluse koos SQL lausega.
setActiveQuery	Saadab laole aktiivse päringu määramise käskluse koos päringu andmetega.
switchLimit	Saadab laole limiteerimise ümberlülituse käskluse ning SQL lause genereerimise käskluse.
setLimitValue	Saadab laole limiteerimise väärtuse määramise käskluse koos limiteerimise väärtusega ning SQL lause genereerimise käskluse.
updateValidity	Saadab laole päringu kehtivuse/õigsuse tõeväärtuse uuendamise käskluse koos tõeväärtusega.
addSet	Saadab laole uue hulgaoperatsiooni objekti lisamise käskluse.
updateSet	Saadab laole hulkade järjendi uuendamise käskluse koos hulgaoperatsiooni objektiga ning SQL genereerimise käskluse.
removeSet	Saadab laole hulgaoperatsiooni objekti kustutamise käskluse koos hulgaoperatsiooni objektiga ning SQL genereerimise käskluse.
updateSetsOrder	Saadab laole hulgaoperatsioonide järjendi järjekorra uuendamise käskluse koos hulgaoperatsiooni objektiga ning SQLi genereerimise käskluse.
addQuery	Saadab laole uue päringu lisamise käskluse.
updateQueries	Saadab laole päringute järjendi uuendamise käskluse koos viimase aktiivse päringu objektiga ning uue aktiivse päringu identifikaatoriga.
removeMainFromQueries	Saadab laole päringute järjendist põhipäringu eemaldamise käskluse.

8.3.8 Muutunud eesrakenduse ülekandjad

Tabel 9 loetleb päringu ülekandja (*queryReducer)* esialgse lao olekuobjekti muudatused.

Tabel 9. Muutunud päringu ülekandja esialgse lao olekuobjekti muudatused *Postgres Visual Query Builder* teises versioonis koos lisatud võtmete, nende esialgsete väärtuste, tüüpide ja kirjeldusega.

Võtme nimi	**Esialgne väärtus**	**Väärtuse tüüp**	**Kirjeldus**
limit	False	Tõeväärtus	Esialgse lao aktiivse päringu limiteerimise operatsiooni tõeväärtus.
limitValue	50	Täisarv	Esialgse lao aktiivse päringu limiteerimise väärtus.
queryName	'Main'	Sõne	Esialgse lao aktiivse päringu nimi.
queryValid	True	Tõeväärtus	Esialgse lao aktiivse päringu kehtivuse/õigsuse tõeväärtus.
sets	Tühi järjend	Järjend	Esialgse lao aktiivse päringu hulkade järjend.
column_filter_operand	Tühi sõne	Sõne	Esialgse lao aktiivse päringu veeru filtri operaator.
filter_as_having	False	Tõeväärtus	Esialgse lao aktiivse päringu veeru *HAVING* klausli tõeväärtus.
subquerySql	Tühi sõne	Sõne	Esialgse lao aktiivse päringu veeru alampäringu SQL lause.
subqueryId	0	Täisarv	Esialgse lao aktiivse päringu veeru alampäringu identifikaator.

8.3.9 Uued eesrakenduse ülekandjad

Tabel 10 loetleb kõik uued tegevuste tüüpidest/käsklustest sõltuvad ülekandjate kirjeldused. Rakenduse lattu lisati uus päringute ülekandja (*queriesReducer)*, mis haldab kõikide mitteaktiivsete päringute olekuid. Päringute ülekandja esialgne olek on tühi järjend.

Tabel 10. Uued tegevuste tüüpidest sõltuvad ülekandjate kirjeldused *Postgres Visual Query Builder* teises versioonis.

Tegevuse tüüp/käsklus	**Kirjeldus**
UPDATE_COLUMN_OPERAND	Uuendab laos veeru filtri operaatori väärtust tulenevalt tegevuse saadetisest.
SWITCH_LIMIT	Muudab laos aktiivse päringu limiteerimise tõeväärtuse vastupidiseks.
SET_LIMIT_VALUE	Määrab laos aktiivse päringu limiteerimise väärtuse tulenevalt tegevuse saadetisest.
UPDATE_SQL	Uuendab laos aktiivse päringu SQL koodi sõne vastavalt tegevuse saadetisele.
SET_ACTIVE_QUERY	Määrab laos saadetisena tuleva päringu aktiivseks ning muudab ka päringu nime.
UPDATE_VALIDITY	Uuendab laos aktiivse päringu õigsuse/kehtivuse tõeväärtust vastavalt tegevuse saadetisele.
ADD_SET	Lisab laos aktiivse päringu hulgaoperatsioonide järjendisse uue hulgaoperatsiooni objekti.
UPDATE_SET	Uuendab laos aktiivse päringu hulgaoperatsiooni objekti.
REMOVE_SET	Eemaldab laos hoitava aktiivse päringu hulgaoperatsioonide järjendist hulgaoperatsiooni objekti (vastavalt identifikaatorile).
UPDATE_SETS_ORDER	Uuendab laos aktiivse päringu hulgaoperatsioonide järjendi järjekorda.
ADD_QUERY	Lisab laos hoitava päringute järjendisse uue päringu objekti.
UPDATE_QUERIES	Uuendab laos hoitava päringute järjendit.
REMOVE_MAIN_FROM_QUERIES	Eemaldab loas hoitava päringute järjendist põhipäringu.
DELETE_QUERIES	Kustutab laost kõik olemasolevad päringud.

8.3.10 Muutunud kasutajaliidese komponendid

Tabel 11 loetleb kõik komponendid, mida selle töö käigus muudeti ning kirjeldab ka tehtuid muudatusi (eraldi ei tooda välja komponendi koodistiili muudatusi ja põhikomponentide sees olevate alamkomponentide/HTML elementide muudatusi ning sündmusetöötlejate muudatusi (*event handler*)). Funktsionaalne komponent on tegelikult tavaline JavaScripti funktsioon, mis võtab sisendiks komponendi omadused (*properties)* ja tagastab Reacti elemendi (objekt komponendi parameetritega, mille alusel ehitatakse ekraanil nähtav HTML). Omaduste tüüpide väljatoomine tähendab, et iga Reacti komponendis kasutatava omaduse jaoks tuuakse selle sama komponendi all või üleval välja tema tüübid (näiteks id: PropTypes.number, mis annab informatsiooni, et komponendis kasutatava omaduse id tüüp on number). See lähenemine on analoog tüübitud programmeerimiskeelele, mis aitab vältida vigade tekkimist (kuvades arendamisel veebilehitseja konsoolis veateateid) ja annab edaspidiseks arendamiseks komponendist parema arusaama. Tegevuse lähetamine (*action dispatch)* tähendab, et Reacti kasutajaliidese komponendis kutsutakse läbi mingisuguse interaktiivse tegevuse (näiteks hiireklõps komponendil) välja tegevuste loojate funktsioon, mis saadab vastava käskluse Reduxi lattu. Reduxi laos tehakse sõltuvalt käsklusest vajalikud lao oleku muudatused ning tulenevalt lao muudatest, uuendatakse ka kasutajaliidese komponenti.

Tabel 11. *Postgres Visual Query Builder* muudetud kasutajaliidese komponendid ja nende kirjeldused.

Komponendi nimi	**Kirjeldus**
DatabaseTable	Kuvab andmebaasi tabeli nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati tabeli ikoon.
DatabaseViewer	Kuvab andmebaasi tabelite listi. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
DeleteQueryButton	Kuvab päringu kustutamise nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati kaks tegevuste lähetamist, millest ühe abil leitakse nupu vajutusel põhipäring ja teisega määratakse leitud päring aktiivseks. Lisati omaduste külge lao päringud, mille abil filtreeritakse välja põhipäring.
DisconnectButton	Kuvab andmebaasiga ühenduse katkestamise nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.

Komponendi nimi	Kirjeldus
DownloadCSVButton	Kuvab CSV faili allalaadimise nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
DownloadSQLButton	Kuvab SQL faili allalaadimise nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
Join	Kuvab ühendamisoperatsiooni kaardi. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati tingimus, mis kontrollib tabelite järjendi olemasolu, et vältida rakenduse katki minemist.
JoinCondition	Kuvab ühendamisoperatsiooni tingimuse kaardi. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
JoinList	Kuvab ühendamisoperatsioonide listi. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati omaduste külge aktiivse päringu identifikaator, et hoida komponendi võtmeid unikaalsena.
LanguageSwitcher	Kuvab keele vahetamise rippmenüü. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
LoginFormContainer	Kuvab sisselogimise vormi konteineri. **Muudatused:** toodi välja omaduste tüübid.
QueryButton	Kuvab päringu käivitamise nupu. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati tegevuse lähetamine, mille abil uuendatakse päringu käivitamisel päringu kehtivuse/õigsuse tõeväärtust.
QueryColumn	Kuvab päringu veeru kaardi. **Muudatused:** toodi välja omaduste tüübid. Lisati omaduste külge kõik lao päringud. Päringute filtreerimisega lingitakse õige alampäring veeru filtri külge. Lisati lüliti *HAVING* klausli kasutamiseks ja rippmenüü alampäringuga linkimiseks. Muudeti kõigi alamkomponentide võtmeid ja identifikaatoreid, et need oleksid unikaalsed.
QueryColumnList	Kuvab päringu veergude listi. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati omaduste külge lao limiteerimise oleku tõeväärtus, limiteerimise väärtus ja aktiivse päringu identifikaator. Samuti lisati kaks tegevuste lähetamist, millest ühega muudetakse limiteerimise olekut ja teisega limiteerimise väärtust. Lisati lüliti ja sisestusväli limiteerimise jaoks.

Komponendi nimi	Kirjeldus
QueryTable	Kuvab päringu tabeli kaardi. **Muudatused:** toodi välja omaduste tüübid.
QueryTablePopover	Kuvab päringu tabeli aliase akna. **Muudatused:** toodi välja omaduste tüübid.
QueryTabs	Kuvab päringu mooduli vahekaardid. **Muudatused:** toodi välja omaduste tüübid. Lisati uus vahekaart hulkade jaoks.
ResultSQL	Kuvab SQL koodi tekstiredaktori. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Asendati olemasolev SQL koodi kuvav element interaktiivse tekstiredaktoriga. Lisati tegevuse lähetamine, mis uuendab SQL lauset vastavalt sisestatud tekstile.
ResultTable	Kuvab päringu tulemuse tabeli. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati tabeli ridadele vahelduvad värvid ja esiletõstmine.
ResultTabs	Kuvab tulemuste mooduli vahekaardid. **Muudatused:** toodi välja omaduste tüübid.
SchemaSelector	Kuvab tabeli skeemi valimise rippmenüü. **Muudatused:** muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid.
SearchBar	Kuvab tabeli otsingukasti. **Muudatused:** toodi välja omaduste tüübid.
TableColumn	Kuvab tabeli veeru nupu. **Muudatused:** toodi välja omaduste tüübid.
TableColumnPopover	Kuvab tabeli veeru välisvõtmete akna. **Muudatused:** toodi välja omaduste tüübid.
withTabSwitcher	Taaskasutab aktiivse vahekaardi loogikat ja kuvab sisendiks antud komponendi. **Muudatused:** muudeti funktsionaalseks komponendiks.
withToggle	Taaskasutab ümberlülituse loogikat ja kuvab sisendiks antud komponendi. **Muudatused:** muudeti funktsionaalseks komponendiks.
LandingPage	Kuvab kogu sisselogimise vaate. **Muudatused:** muudeti funktsionaalseks komponendiks.
QueryPage	Kuvab kogu päringu loomise vaate.

Komponendi nimi	Kirjeldus
	Muudatused: muudeti funktsionaalseks komponendiks, toodi välja omaduste tüübid. Lisati omaduste külge päringu valiidsuse tõeväärtus, mille abil kuvatakse veateadet keelatud päringu korral.

8.3.11 Uued kasutajaliidese komponendid

Tabel 12 loetleb kõik uued komponendid ja nende kirjeldused.

Tabel 12. Uued kasutajaliidese komponendid *Postgres Visual Query Builder* teises versioonis.

Komponendi nimi	Kirjeldus
AddNewButton	Kuvab uue komponendi lisamise nupu (loodud taaskasutamiseks).
FilterOperandSelectbox	Kuvab filtri operaatori valikukasti.
InputWithDeleteButton	Kuvab sisestusvälja koos kustutamise nupuga (loodud taaskasutamiseks).
NavBar	Kuvab navigatsioonirea.
NavBarMenu	Kuvab navigatsioonirea menüü.
NavBarQueryTab	Kuvab navigatsioonirea päringu vahekaardi nupu.
NavBarQueryTabs	Kuvab kõik navigatsioonirea päringute vahekaardid.

8.4 Ühiktestid

Koodi kvaliteedi parandamiseks ning vigade vältimiseks muudeti ning kirjutati arenduse käigus juurde ühikteste. Iga klassikomponendi muutmine funktsionaalseks komponendiks ning uued funktsionaalsused nõudsid ka olemasolevate testide muutmist. Testide kirjutamist jätkati kasutades Jest [71] raamistikku.

Kokku kirjutati juurde 103 testi (testide osakaal suurenes 50%) ning kogu rakenduse testide lõplik reakattuvus on 89%.

9 Valideerimine

Käesolevas peatükis kirjeldatakse töö tulemuste valideerimist. Esiteks valideeris töö autor rakenduse toimimist kogu arenduse vältel ise. Ta tegi seda nii testide abil (vt jaotis **Error! Reference source not found.**) kui ka läbi kasutajaliidese päringute koostamist katsetades. Lisaks sellele koostati ka koos juhendajaga päringuid ning kontrolliti tulemusi. Peale viimase iteratsiooni lõppu leiti rakenduse valideerimise jaoks viis testkasutajat, et saada tagasisidet rakenduse headuse kohta. Igale testkasutajale esitati Google Formsi veebirakenduse abil loodud küsimustik, mis koosnes kolmest SQL päringu koostamise ülesandest ja samuti küsimustest üldise rakenduse kasutusmugavuse ja headuse kohta (küsimuste tekstid on nähtavad Lisa 2). Küsimustiku koostamisel võeti aluseks eelmise autori poolt loodud küsimused [5], kuid selle asemel, et uurida kui palju on küsitletavad kokku puutunud PostgreSQLiga, küsiti kuidas osalejad ise oma oskuseid hindavad.

9.1 Katsetamine kasutajate poolt

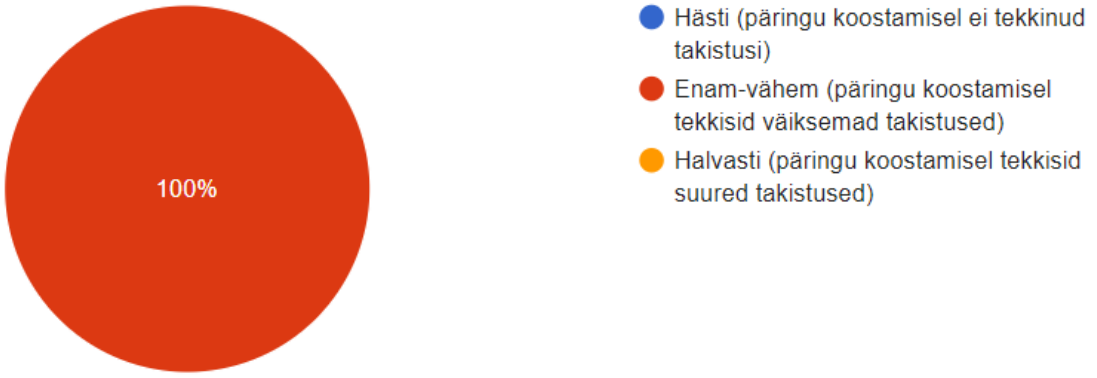
Küsitlus viidi läbi ajaperioodil 03.05.2020–04.05.2020 ning küsitluses osalejad olid küllaltki erineva taustaga. Kaks osalenutest hindasid oma PostgreSQL SQL mägimurraku tundmise taset heaks, kaks keskmiseks ning üks küsitletutest polnud PostgreSQL-iga varem kokku puutunud. Eelnevat lühikoolitust rakenduse kasutamise kohta ei tehtud. Kaks inimest küsisid jooksvalt abi, kui olid veidi kauem hätta jäänud – näiteks alampäringute leidmise ja agregeerimisfunktsiooni lisamisega.

Järgnevalt tuuakse välja kuidas testkasutajad päringute loomisega hakkama said.

```
1  SELECT
2  hotell.hotelli_nr, hotell.nimi, COUNT(hotell.nimi) AS arv
3  FROM public.hotell
4  INNER JOIN public.reserveerimine ON (reserveerimine.hotelli_nr = hotell.hotelli_nr)
5  GROUP BY hotell.hotelli_nr, hotell.nimi
6  HAVING (Count(hotell.nimi) > 3);
```

Joonis 70. Esimene päring.

Esimese päringu (Joonis 70) koostamisega said kõik enam-vähem hakkama, tekkisid mõned väiksemad takistused (Joonis 71).



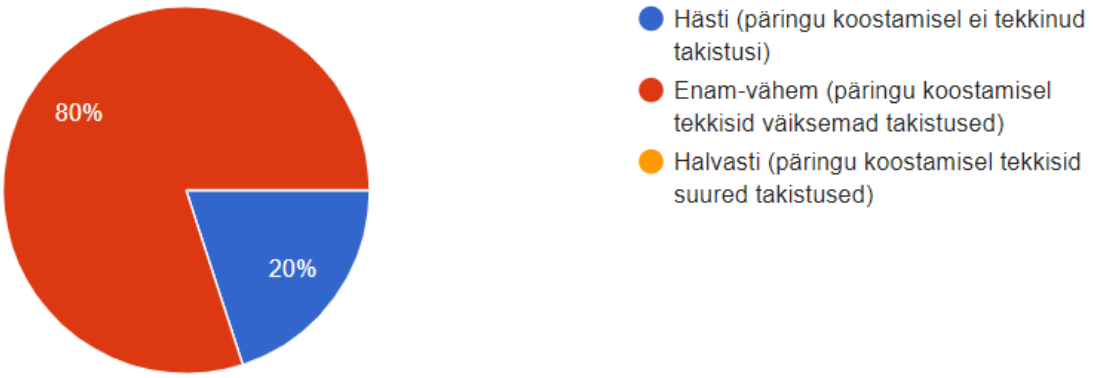
Joonis 71. Esimese päringu tagasiside ringdiagramm.

Toodi välja, et filtreerimisel oli lihtne lisada filtrit, aga kokkuvõttefunktsiooni (*Count)* lisamine filtri juurde võttis natuke aega. Polnud selge kas seda saab teha kuidagi läbi kasutajaliidese või otse filtri kasti sisse kirjutades.

```
1  SELECT
2  hotell.hotelli_nr, hotell.nimi
3  FROM public.hotell
4  WHERE (hotell.hotelli_nr NOT IN (SELECT
5  reserveerimine.hotelli_nr
6  FROM public.reserveerimine));
```

Joonis 72. Teine päring.

Teise päringu (Joonis 72) koostamisega sai üks inimene hästi hakkama ja ülejäänud enam-vähem. Päringute koostamisel tekkisid väiksemad takistused (Joonis 73).



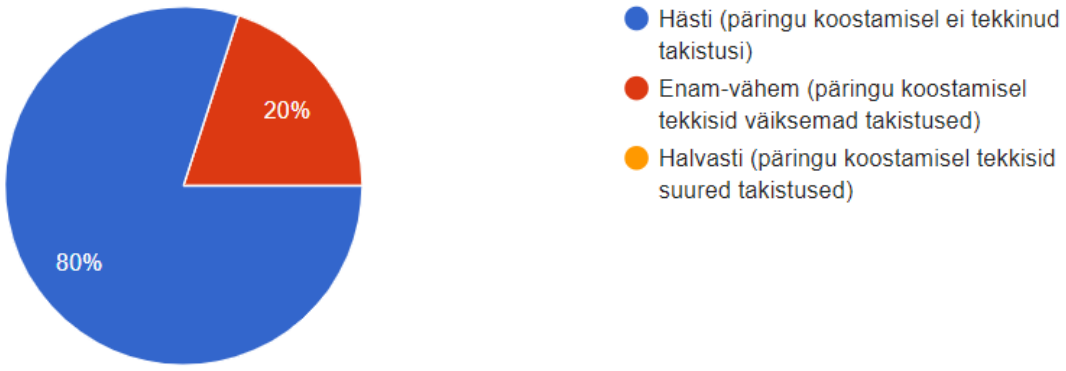
Joonis 73. Teise päringu tagasiside ringdiagramm.

Vastuseid põhjendati sellega, et keeruline oli esialgu leida, kust saab alampäringuid lisada, aga edaspidised sammud olid lihtsasti mõistetavad.

```sql
SELECT
ruum.ruumi_nr, ruum.hotelli_nr, ruum.ruumi_tyyp, ruum.hind
FROM public.ruum
UNION
SELECT
ruum_koopia.ruumi_nr, ruum_koopia.hotelli_nr, ruum_koopia.ruumi_tyyp, ruum_koopia.hind
FROM public.ruum_koopia;
```

Joonis 74. Kolmas päring.

Kolmanda päringu (Joonis 74) koostamisega said juba enamik vastajaid hästi hakkama. Ainult ühel inimesel tekkisid väiksemad takistused (Joonis 75).



Joonis 75. Kolmanda päringu tagasiside ringdiagramm.

Küsitletud põhjendasid, et kuna eelnevalt oli juba asju läbi tehtud, siis kolmanda päringu koostamine raskuseid ei valmistanud. Ühe inimese jaoks oli keeruline leida, kust *UNION* operatsiooni valida.

9.2 Üldine tagasiside

Neli küsitluses osalenud inimest arvasid, et rakendust on mugav kasutada ning ühe inimese jaoks oli rakendus enam-vähem mugav.

Küsitletutele meeldis, et rakenduses on erinevate värvidega elemente rõhutatud ning et saab näha koostatava päringu SQL kuju. Lisaks meeldis rakenduse üldine idee, et saab visuaalselt kasutajaliidese abil päringut koostada ning vajadusel käsitsi SQL lauset muuta. Üks küsitletutest tõi välja, et rakendust on lihtne kasutada ning pole segavaid disainielemente ning kõik sektsioonid on loogiliselt eraldatud.

Küsitletutele ei meeldinud, et rakenduses on „hüplevad nupud“, mille all peeti silmas navigatsioonirea kuvamist ainult hiirega selle peale liikudes. Samuti toodi välja, et võiks

lisada igale elemendile tööriistavihjed (*tooltips).* Lisaks, tõi üks inimene välja, et ühendamise *(Join)* operatsioonide tegemisel võiks olla kohe tabeli valimisel võimalus valida sarnaselt külgribale kõiki andmebaasi tabeleid koos otsinguga, ilma et selle peaks eraldi päringusse valima. Samuti tõi ta välja, et päringu nime muutus võiks koheselt toimida, ilma, et peaks vahepeal mujale vajutama. Lisaks mainis ta, et põhipäringus võiks prügikasti nupp olla peidetud seni kuni ühtegi tabelit pole valitud, sest muidu tekib tunne, et saab ka esialgset põhipäringut kustutada. Üks küsitletutest arvas, et tulemuste tabelist võiks kaotada tühjad read, kuna need võivad jätta mulje, et tabel sisaldab ridu, kus väärtused puuduvad (väljades on *NULL* markerid).

Üks küsitletutest tõi välja ka rakendusest leitud vea. Vea taasesitamiseks tuli luua uus päring, lehekülge värskendada ning seejärel jälle uus päring lisada. Selle tulemusel läksid päringute identifikaatorid segamini ja ühe ja sama nimega päringut näidati mitu korda.

Pärast rakenduse ülesseadmist serverisse andis ka juhendaja omapoolse tagasiside (väljaspool küsitlust). Tulenevalt sellest, et rakendus hakkab kasutama sama domeeni, mis esimene versioon, soovis juhendaja, et rakenduses võiks olla mingisugune vaheleht või aken, mis kuvaks rakenduse autorid. Lisaks mainis ta, et rakenduse elementide tõlked võiksid olla mitte nii tehnilised ja suurema hulga inimeste jaoks arusaadavamad ning soovitas omapoolseid tõlkemuudatusi.

9.3 Tagasiside alusel tehtud parandused

Rakenduses tehti tagasiside põhjal järgnevad muudatused.

- Lisati rakenduse maandumislehele teabe nupp, mis kuvab informatsiooni rakenduse autorite kohta.
- Parandati rakendusest leitud viga, muutes unikaalsete identifikaatorite genereerimise loogikat.
- Muudeti navigatsioonirea kuvamise loogikat. Nüüd on rakendusse sisselogimisel kohe näha navigatsioonirea menüüd, et kasutaja leiaks kohe, kuidas uut päringut lisada. Samuti lisati nupp menüü sulgemiseks ning uuesti avamiseks.
- Muudetud päringu nime kuvatakse nüüd koheselt, ilma et peaks mujale vajutama.

- Lisati päringute nimedele tõlked ja muudeti need dünaamiliseks (kui päringule lisada omapoolne nimetus, siis jääb see kehtima. Kui see kustutada, siis lisatakse automaatselt vastava tõlke ja järjekorranumbriga päringu nimetus).
- Muudeti kõigi rakenduses olevate lülitite tõlkeid.
- Nüüd kuvatakse kustutamise nuppu põhipäringus ainult siis kui vähemalt üks tabel on valitud või uus päring lisatud.
- Põhipäringu kustutamine kustutab nüüd ka kõik alampäringud.
- Tulemuste tabelis ei kuvata enam tühjasid ridu ning lehtede valiku komponenti näidatakse ainult siis kui ridade arv on suurem kui 20.
- Filtrile lisati sisestusviip, mis viitab veeru filtris kasutatavale lühendile.
- Parandati viga nii, et nüüd on alampäringu muudatus kohe näha ka põhipäringus.
- Nüüd lisatakse päringule automaatselt kokkuvõttefunktsioon, kui veerul on rakendatud grupeerimist ning on valitud kokkuvõttefunktsioon. See tähendab, et kokkuvõttefunktsiooni pole vaja enam käsitsi filtri kasti sisestada.

10 Arendusvaade

Käesoleva lõputöö autori hinnangul oleks järgmistes väljalasetes mõistlik keskenduda kõigepealt olemasoleva rakenduse parandamisele, mitte kohe uute funktsionaalsuste loomisele. Näiteks tuleks mõelda juba esimeses versioonis realiseeritud võimaluste paremaks muutmisele. Võttes arvesse lisaks oma mõtetele ka esimese versiooni ja uue versiooni tagasisidet, pakub autor välja järgmised ideed.

- Kui WHERE klauslis on liit-otsingutingimus (vt jaotis 8.1.9), siis hetkel ei ole võimalik graafiliselt määrata alamtingimuste kontrollimise järjekorda. SQLis rakendatakse AND operatsiooni enne OR operatsiooni ning tingimuste kontrolli järjekorda saab mõjutada sulge kasutades. Tingimus a=1 AND b=2 OR c=3 on samaväärne tingimusele (a=1 AND b=2) OR c=3. Kui aga on soov kasutada tingimust a=1 AND (b=2 OR c=3), siis pole praegu muud midagi teha, kui vajalikud sulud genereeritud SQL lausesse käsitsi sisestada või lisada need piirangu algusesse või lõppu, vastavalt vajadusele (vt jaotis 8.1.7).

- Filtri (*WHERE* klausli) lisamise loogika tuleks teha kasutaja jaoks lihtsamaks (näiteks pakkuda kasutajale filtri juures eraldi nimekirja kõikidest võimalikest operaatoritest).

- Tabelite ning veergude valimisel võiks kasutusmugavuse mõttes olla võimalik neid ka samamoodi eemaldada nagu neid lisatakse (hiirevajutusega tabelil või veerul, lisaks võimalus elementi fokusseerida ja kustutada näiteks *Delete* klahviga). Hetkel tehakse sellise protseduuri puhul tabelist näiteks koopia, kuigi see võimalus on juba eraldi nupu näol olemas.

- Kõikide sisestusväljade muudatusi võiks kuvada koheselt muudatuse tegemise ajal SQL redaktoris, ilma, et peaks vahepeal mujale vajutama.

- Lisada ühendamise lisamisel võimalus tabeleid kohe läbi valikukasti valida (näiteks koos otsimisvõimalusega), ilma et neid peaks külgribalt valima.

- Tuleks lisada võimalus läbi graafiliste komponentide filtris kokkuvõttefunktsioone kasutada.

- Lisada alampäringu linkimise ja hulgaoperaatorite juurde otseteed uute päringute loomiseks.

- Viia kogu rakendus funktsionaalsete komponentide kujule (näited on uue versiooni näol olemas).

- Kui kõik eelnevad parandused on tehtud võib keskenduda uue funktsionaalsuse realiseerimisele, näiteks alustades funktsionaalsustest, mida selle lõputöö käigus valmis ei jõutud (vt jaotis 6.3.3).

Siinkohal tuleb silmas pidada, et need on kõigest autori poolsed ideed, mis ei pruugi sobida teiste inimeste nägemusega.

11 Kokkuvõte

Käesoleva töö eesmärgiks oli edasi arendada PostrgreSQL andmebaasides SQL päringute (SELECT lausete) visuaalseks koostamiseks mõeldud avatud lähtekoodiga veebirakendust *Postgres Visual Query Builder*, mille esimene versioon valmis hiljuti lõputöö tulemusena. Töö käigus realiseeriti rakenduse eelmises versioonis puuduvat funktsionaalsust ja tehti tarkvarasse ka muid parandusi. Otsiti ja leiti viisid mitmete SQL andmekäitluskeele lausete konstruktsioonide visuaalseks esitamiseks ning realiseeriti see esitus tarkvaras. Veel üheks eesmärgiks oli kirjeldada mustrite formaadis visuaalse andmekäitluskeele lausete koostamise vahendi graafilise kasutajaliidese häid praktikaid ning ka see saavutati.

Kõigepealt tutvuti erinevate teadusartiklitega ja saadi kindlust, et visuaalsed programmeerimiskeskkonnad, mille kasutajad peavad kirjutama vähe või üldse mitte tekstilist lähtekoodi, on aktuaalne teema. Samuti analüüsiti ja võrreldi olemasolevaid veebipõhiseid SQL päringute koostamise rakendusi, et saada aru nende kasutajaliidese headest praktikatest. Samaaegselt uuriti ka töö sisendiks olevat rakendust ning leiti funktsionaalsused, mida võiks järgnevalt realiseerida. Lisaks rakendustest ammutatud ideedele võeti arvesse ka juhendaja poolt esitatud nõudeid.

Kuna puuduvat funktsionaalsust oli palju, siis tuli koostada väljalaske plaan, et realiseerida lõputöö käigus juhendaja määratud prioriteetsemad ülesanded. Väljalaske plaani koostamiseks kasutati T. Normani poolt välja pakutud väljalaske planeerimise meetodit ning arendamisel kasutati paindmetoodikate parimaid praktikaid.

Autori hinnangul puuduvad nõuded või soovitused, kuidas selliste rakenduste kasutajaliidesed võiksid välja näha. Uuritud rakenduste alusel pandi kirja 12 erinevat kasutajaliidese disainimustrit, mida võiks/tuleks rakendada selliste rakenduste loomisel. Nende mustrite alusel hinnati päringute tegemise rakendust enne ja pärast täienduste tegemist.

Käesoleva tööga *Postgres Visual Query Builder* tarkvarasse lisatud SQL SELECT lausete koostamise võimalustele ei olnud võrreldavate rakenduste hulgas üldse analooge (alampäringud WHERE klauslis) või oli neid vähestes (HAVING klausel, hulgaoperatsioonid). See tõstab arendatud tarkvara teiste võrreldavatest tarkvarade seast

esile. Nende konstruktsioonide visuaalse esitamise kohta oli eeskujuks võtta vähe näiteid. Samas saab selle töö tulemus olla eeskujuks teistele sarnaste vahendite arendajatele.

Uue funktsionaalsuse realiseerimise ideed ja lähenemised toodi välja nii eesrakenduse kui ka tagarakenduse jaoks ning toodi detailselt välja tehtud uuendused ja muudatused.

Lõputöö käigus täiendatud rakendust valideeriti nii autori enda kui ka testkasutajate poolt, et veenduda selle headuses. Testkasutajatelt ja juhendajalt saadud tagasiside põhjal tehti rakendusse parandusi. Samuti kirjutati ja käivitati hulgaliselt ühikteste.

Postgres Visual Query Builderi uue versiooni lähtekood on kättesaadav GitHubi hoidlast: https://github.com/rax1110/postgres-visual-query-v2 Rakenduse lähtekood on kaitstud MIT litsentsiga.

Selle tööga ei ole rakenduse arendus kindlasti veel lõppenud. Palju funktsionaalsust ning muudatusi tuleks veel realiseerida ning mõned ideed on toodud välja töö lõpus esitatud arendusvaates. Igal juhul võib öelda, et inimesel, kes plaanib käesolevat rakendust edasi arendada, võiksid olla head teadmised nii kasutajaliidese disainist, tarkvara arhitektuurist kui ka programmeerimisest ja SQLi võimalustest.

Kasutatud kirjandus

[1] „DB-engines,“ 13 02 2020. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking.

[2] M. Noone ja A. Mooney, „First Programming Language: Visual or Textual?,“ %1 *International Conference on Engaging Pedagogy*, Maynooth, 2017.

[3] „No-code development platform,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/No-code_development_platform. [Kasutatud 25 01 2020].

[4] „Participatory design,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Participatory_design. [Kasutatud 16 11 2019].

[5] E. Dzotsenidze, PostgreSQL veebipõhise visuaalse päringute koostamise tarkvara arendamine: bakalaureusetöö, Talllinn: Tallinna Tehnikaülikool, 2019.

[6] O. I. Lindland, G. Sindre ja A. Solvberg, „Understanding quality in conceptual modeling.,“ *IEEE software,* kd. 11, nr 2, pp. 42-49, 1994.

[7] „Postgresql,“ [Võrgumaterjal]. Available: https://postgresql.org/. [Kasutatud 14 11 2019].

[8] 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.

[9] „Scratch Programming,“ Lifelong Kindergarten Group, [Võrgumaterjal]. Available: https://en.scratch-wiki.info/wiki/Programming. [Kasutatud 27 02 2020].

[10] G. Karsai, „A configurable visual programming environment: a tool for domain-specific programming,“ *Computer,* kd. 28, nr 3, pp. 36 - 44, 1995.

[11] M. Boshernitsan ja M. S. Downes, „Visual Programming Languages: A Survey,“ Computer Science Division, EECS, Berkeley, 2004.

[12] W. Citrin, M. Doherty ja B. Zorn, The Design of a Completely Visual Object-Oriented Programming Language, Colorado: University of Colorado, 1994.

[13] W. F. Finzer ja L. Gould, „Rehearsal World: Programming by Rehearsal,“ *Byte,* kd. 9, nr 6, 1984.

[14] D. C. Smith, „PYGMALION: A creative Programming Environment,“ Stanford Artificial Intelligence Laboratory, Stanford, 1975.

[15] R. B. Smith, „Experiences with the alternate reality kit: an example of the tension between literalism and magic,“ *ACM SIGCHI Bulletin,* kd. 17, nr SI, pp. 61-67, 1986.

[16] M. Burnett, J. Atwood, R. W. Djang, J. Reichwein, H. Gottfried ja S. Yang, „Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm,“ *Journal of Functional Programming,* kd. 11, nr 2, pp. 155-206, 2001.

[17] E. Eessaar, „Andmebaaside projekteerimiseks kasutatavad mudelid,“ Erki Eessaar, Tallinn, 2020.

[18] S. Kelly ja R. Pohjonen, „Worst practices for domain-specific modeling,“ *IEEE software,* kd. 26, nr 4, pp. 22-29, 2009.

[19] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler ja S. Völkel, „Design Guidelines for Domain Specific Languages,“ %1 *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling* , Orlando, Florida, 2009.

[20] M. M. Zloof, "Query by example: a database language.," *IBM Systems Journal,* vol. 16, no. 4, 1977.

[21] T. Neward, „Queries, updates, and identity,“ *The busy Java developer's guide to db4o,* pp. 1-13, 27 March 2007.

[22] „Query by Example,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Query_by_Example. [Kasutatud 25 01 2020].

[23] „Modern-SQL,“ [Võrgumaterjal]. Available: https://modern-sql.com/. [Kasutatud 06 11 2019].

[24] R. Waszkowski, „Low-code platform for automating business processes in manufacturing,“ *IFAC-PapersOnLine,* kd. 52, nr 10, pp. 376-381, 2019.

[25] C. Richardson ja J. R. Rymer, „Vendor Landscape: The Fractured, Fertile Terrain. The Landscape Reflects A Market In Its Formative Years,“ Forrester Research, Cambridge, 2016.

[26] „No-code,“ Kissflow, [Võrgumaterjal]. Available: https://kissflow.com/no-code/. [Kasutatud 06 11 2019].

[27] „The Growing Importance Of Process To Digital Transformation,“ Forrester, [Võrgumaterjal]. Available: https://www.forrester.com/report/The+Growing+Importance+Of+Process+To+Digital+Transformation/-/E-RES143158. [Kasutatud 25 01 2020].

[28] „Skyvia,“ [Võrgumaterjal]. Available: https://skyvia.com/query/sql-query-builder. [Kasutatud 06 11 2019].

[29] „Active Query Builder,“ ActiveDBSoft, [Võrgumaterjal]. Available: https://www.activequerybuilder.com/. [Kasutatud 04 02 2020].

[30] „ASPx Query Builder Control,“ Developer Express Inc, [Võrgumaterjal]. Available: https://demos.devexpress.com/ASPxGridViewDemos/DataBinding/QueryBuilderControl.aspx. [Kasutatud 16 02 2020].

[31] „Datapine SQL Query Builder,“ Datapine, [Võrgumaterjal]. Available: https://www.datapine.com/sql-query-builder. [Kasutatud 16 02 2019].

[32] „Application Express SQL Workshop Guide: Using Query Builder,“ Oracle, [Võrgumaterjal]. Available: https://docs.oracle.com/database/apex-5.1/AEUTL/using-query-builder.htm#AEUTL405. [Kasutatud 16 02 2020].

[33] „Agile software development,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Agile_software_development. [Kasutatud 02 02 2020].

[34] „Agile release planning,“ Lucidchart, [Võrgumaterjal]. Available: https://www.lucidchart.com/blog/agile-release-planning. [Kasutatud 02 02 2020].

[35] „Agile Release Planning 101,“ T.Norman, [Võrgumaterjal]. Available: http://tommynorman.blogspot.com/2012/09/agile-release-planning-101.html. [Kasutatud 06 11 2019].

[36] „Scrum-guide,“ scrumguides, [Võrgumaterjal]. Available: https://www.scrumguides.org/scrum-guide.html. [Kasutatud 12 04 2020].

[37] D. Greer ja G. Ruhe, „Software release planning: an evolutionary and iterative approach,“ *Information and Software Technology*, kd. 46, nr 4, pp. 243-253, 2004.

[38] V. T. Heikkilä, Case studies on release planning in agile development organizations, Helsinki: Aalto University, School of Science, 2015.

[39] D. Wells, „Extreme Programming,“ [Võrgumaterjal]. Available: http://www.extremeprogramming.org/. [Kasutatud 12 04 2020].

[40] „Kanban (development),“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Kanban_(development). [Kasutatud 12 04 2020].

[41] C. VersionOne, „13th Annual State Of Agile Report,“ CollabNet VersionOne, California, 2019.

[42] B. Meyer, Agile! : The Good, the Hype and the Ugly. 2014th Edition., Springer, 2014.

[43] „Timebox,“ Agile Alliance, [Võrgumaterjal]. Available: https://www.agilealliance.org/glossary/timebox/. [Kasutatud 06 10 2019].

[44] doasync, „eslint-config-airbnb-standard,“ [Võrgumaterjal]. Available: https://github.com/doasync/eslint-config-airbnb-standard. [Kasutatud 03 04 2020].

[45] „Babel,“ [Võrgumaterjal]. Available: https://babeljs.io/. [Kasutatud 03 04 2020].

[46] yannickcr, „eslint-plugin-react,“ [Võrgumaterjal]. Available: https://github.com/yannickcr/eslint-plugin-react. [Kasutatud 03 04 2020].

[47] scniro, „react-codemirror2,“ [Võrgumaterjal]. Available: https://github.com/scniro/react-codemirror2. [Kasutatud 03 04 2020].

[48] „Nodejs,“ [Võrgumaterjal]. Available: http://nodejs.org/. [Kasutatud 14 11 2019].

[49] „JavaScript,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Kasutatud 09 02 2020].

[50] „Express,“ [Võrgumaterjal]. Available: http://expressjs.com/. [Kasutatud 14 11 2019].

[51] „node-postgres,“ [Võrgumaterjal]. Available: http://node-postgres.com/. [Kasutatud 14 11 2019].

[52] „React,“ [Võrgumaterjal]. Available: http://reactjs.org/. [Kasutatud 14 11 2019].

[53] „Redux,“ [Võrgumaterjal]. Available: http://redux.js.org/. [Kasutatud 14 11 2019].

[54] „Bootstrap,“ Bootstrap team, [Võrgumaterjal]. Available: https://getbootstrap.com/. [Kasutatud 14 11 2019].

[55] „axios,“ axios, [Võrgumaterjal]. Available: https://github.com/axios/axios. [Kasutatud 14 11 2019].

[56] „Squel,“ Hiddentao, [Võrgumaterjal]. Available: https://hiddentao.github.io/squel/. [Kasutatud 14 11 2019].

[57] „Standard ECMA-262 6th Edition,“ Ecma International, [Võrgumaterjal]. Available: http://www.ecma-international.org/ecma-262/6.0/. [Kasutatud 12 02 2020].

[58] Y. Goldberg, „Node.js Best Practices,“ [Võrgumaterjal]. Available: https://github.com/goldbergyoni/nodebestpractices. [Kasutatud 29 02 2020].

[59] R. C. Martin, Clean Code, New Jersey: Prentice Hall, 2008.

[60] D. Crockford, „JSLint,“ [Võrgumaterjal]. Available: http://jslint.com/. [Kasutatud 12 02 2020].

[61] „ESLint,“ [Võrgumaterjal]. Available: https://eslint.org/. [Kasutatud 12 02 2020].

[62] „UI Patterns,“ [Võrgumaterjal]. Available: http://ui-patterns.com/. [Kasutatud 06 11 2019].

[63] „Rule Of Three,“ Wiki, [Võrgumaterjal]. Available: https://wiki.c2.com/?RuleOfThree. [Kasutatud 12 04 2020].

[64] „Fitts Law,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Fitts%27s_law. [Kasutatud 13 04 2020].

[65] „User-stories,“ Mountain Goat Software, [Võrgumaterjal]. Available: https://www.mountaingoatsoftware.com/agile/user-stories. [Kasutatud 17 11 2019].

[66] „Venni diagramm,“ Wikipedia, [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Venni_diagramm. [Kasutatud 15 05 2020].

[67] „Õppekorralduse eeskiri,“ Tallinna Tehnikaülikool, 01 11 2019. [Võrgumaterjal]. Available: https://www.ttu.ee/tudengile/oppeinfo/oppekorraldus/oppetegevuse-juhendid-ja-oigusaktid/oppee/. [Kasutatud 05 11 2020].

[68] Microsoft, „Monaco Editor,“ Microsoft, [Võrgumaterjal]. Available: https://microsoft.github.io/monaco-editor/. [Kasutatud 11 05 2020].

[69] „User's guide,“ Active Query Builder, [Võrgumaterjal]. Available: https://www.activequerybuilder.com/hs90.html. [Kasutatud 15 05 2020].

[70] Airbnb, „JavaScript (style guide),“ Airbnb, [Võrgumaterjal]. Available: https://github.com/airbnb/javascript. [Kasutatud 04 04 2020].

[71] „Jest,“ Facebook Inc, [Võrgumaterjal]. Available: http://jestjs.io/. [Kasutatud 14 11 2019].

Lisa 1 – disainimustritele vastavuse hindamisskaala

„5“ („A”) – „suurepärane” – silmapaistev ja eriti laiapõhjaline mustrile vastavuse tase, mida iseloomustab väga head taset ületav visuaalsete komponentide vaba ning loov kasutamine;
„4” („B”) – „väga hea” – väga heal tasemel mustrile vastavus, mida iseloomustab visuaalsete komponentide eesmärgipärane ja loov kasutamine. Komponentide detailide osas võivad ilmneda mittesisulised ja mittepõhimõttelised eksimused;
„3” („C”) – „hea” – heal tasemel mustrile vastavus, mida iseloomustab visuaalsete komponentide eesmärgipärane kasutamine. Komponendi või selle kasutamise detailide osas avaldub ebaotstarbekus ja ebatäpsus;
„2” („D”) – „rahuldav” – piisaval tasemel mustrile vastavus, mida iseloomustab visuaalsete komponentide kasutamine tüüpolukordades, erandlikes olukordades avalduvad puudujäägid ja ebaotstarbekus;
„1” („E”) – „kasin” – minimaalselt lubataval tasemel olulisemate mustrile vastavuse nõuete saavutamine, mida iseloomustab visuaalsete komponentide kasutamine tüüpolukordades piiratud viisidel, erandlikes olukordades avalduvad märgatavad puudujäägid ning ebaotstarbekus;
„0” („F”) – „puudulik” – visuaalne komponent on puudulik.

Lisa 2 – Tagasiside küsimustik

Kuidas hindate oma PostgresSQL-i kasutamise oskust?

- ○ Hea (olen kirjutanud keerulisemaid PostgreSQL lauseid, teinud päringute optimeerimisi, loonud enda Postg...
- ○ Keskmine (olen kirjutanud lihtsamaid ja keskmise raskusega PostgreSQL lauseid, kuid kõiki detaile ei tea)
- ○ Vähene (olen kirjutanud mõne lihtsama PostgreSQL lause)
- ○ Puudulik (pole PostgresSQL-iga kokku puutunud)
- ○ Other...

PÄRING 1: Leia hotellid, milles reserveerimiste arv on suurem kui kolm. Väljasta hotelli number, nimi ja reserveerimiste arv.

```
SELECT
hotell.hotelli_nr, hotell.nimi, COUNT(hotell.nimi) AS arv
FROM public.hotell
INNER JOIN public.reserveerimine ON (reserveerimine.hotelli_nr = hotell.hotelli_nr)
GROUP BY hotell.hotelli_nr, hotell.nimi
HAVING (Count(hotell.nimi) > 3);
```

Kuidas saite hakkama esimese päringu koostamisega? *

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

Palun põhjendage eelmist vastust. *

Long answer text

PÄRING 2: Leia hotellid, milles ei ole tehtud ühtegi reserveerimist kasutades alampäringut. Väljasta hotelli number ja nimi.

```
SELECT
hotell.hotelli_nr, hotell.nimi
FROM public.hotell
WHERE (hotell.hotelli_nr NOT IN (SELECT
reserveerimine.hotelli_nr
FROM public.reserveerimine));
```

Kuidas saite hakkama teise päringu koostamisega? *

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

Palun põhjendage eelmist vastust. *

Long answer text

PÄRING 3: Leia tabelites Ruum ja Ruum_koopia olevate andmete ühend (union).

```
1  SELECT
2  ruum.ruumi_nr, ruum.hotelli_nr, ruum.ruumi_tyyp, ruum.hind
3  FROM public.ruum
4  UNION
5  SELECT
6  ruum_koopia.ruumi_nr, ruum_koopia.hotelli_nr, ruum_koopia.ruumi_tyyp, ruum_koopia.hind
7  FROM public.ruum_koopia;
```

Kuidas saite hakkama kolmanda päringu koostamisega? *

- ○ Hästi (päringu koostamisel ei tekkinud takistusi)
- ○ Enam-vähem (päringu koostamisel tekkisid 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 teile rakenduse juures meeldis? *

Long answer text

Mis teile rakenduse juures ei meeldinud? *

Long answer text

Kuidas võiks rakendust parandada? *

Long answer text

Kui leidsite rakendusest mõne vea, siis kirjutage sellest siia.

Long answer text