TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karel Markus Mulk 185279IAIB

ANDMEMUUDATUSTE TOE LISAMINE POSTGRESQL VEEBIPÕHISESSE VISUAALSESSE ANDMEKÄITLUSE LAUSETE KOOSTAMISE TARKVARASSE

Bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2024

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: Karel Markus Mulk

11.01.2024

Annotatsioon

Veebipõhine visuaalselt andmekäitluse lausete koostamist võimaldav tarkvara *Postgres Visual Query Builder* on avatud lähtekoodiga tarkvara, mis on loodud Tallinna Tehnikaülikooli üliõpilaste poolt. Eelnevalt on selles tarkvaras realiseeritud andmete otsimiseks mõeldud SELECT lausete koostamine ja käivitamine. Käesoleva töö eesmärgiks oli tarkvaras realiseerida PostgreSQL andmemuudatuste lausete INSERT, UPDATE ja DELETE graafilise koostamise võimalus. Lisaks sellele tuli parandada rakenduse kasutusmugavust ja kõrvaldada kõik arenduse käigus leitud rakenduse töötamist pärssivad vead. Töö alguses mindi üle Node.js versioonilt 8 versioonile 18.

Lõputöös järgitakse disainiteaduse metoodikat. Lõputöö tulemusena valmib tehniline artefakt e tehis, mis on realiseeritud kasutaja nõuete ja valdkonna parimate praktikate põhjal. Enne arendusega alustamist tutvuti lähemalt rakenduse viimase versiooniga ning analüüsiti seda kasutajaliidese, tagarakenduse ja eesrakenduse vaatest. Seejärel uuriti ülesande teoreetilist tausta, tuues välja asjakohased ja olulisemad teadusartiklid ning nendes pakutud lahendused. Lisaks analüüsiti olemasolevat tarkvara, mis võimaldavad andmemuudatuste lausete visuaalset koostamist. Analüüsi tulemusena valminud teadmiste kogum oli sisendiks tööle. Arendusprotsessis kasutati agiilse arenduse põhimõtteid ja parimaid praktikaid ning väljalaske planeerimiseks kasutati T. Normani poolt kirjeldatud agiilset väljalaske planeerimist. Töö tulemust valideeriti testkasutajatele loodud tagasiside küsimustiku abil ning samuti võrreldi loodud rakendust eelnevalt analüüsitud olemasolevate programmidega. Autor ei leidnud ühtegi visuaalset SQL lausete koostamise programmi PostgreSQL jaoks, mis võimaldaks koostada andmemuudatuse lauseid ning üldse on andmemuudatuse lausete graafilist koostamist võimaldavaid programme palju vähem kui otsingulausete (SELECT lausete) graafiliseks koostamiseks mõeldud tarkvara.

Tarkvara eesrakenduse realiseerimiseks kasutati React raamistikku. Tagarakendus realiseeriti Node.js JavaScript käitluskeskkonnas. Rakenduse lähtekood on kaitstud MIT litsensiga ja on avalikult kättesaadav GitHubi koodihoidlas: `https://github.com/v4hukomm/postgresql-visual-query-v3`.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 47 leheküljel, 8 peatükki, 39 joonist, 4 tabelit.

Abstract

Adding Support of Data Manipulation to a PostgreSQL Web-based Visual Query Design Software

The goal of this thesis is to add support for data modification to a PostgreSQL web-based visual query design software *Postgres Visual Query Builder*. The application is open source software that has been developed by students of Tallinn University of Technology. Previous developers have implemented the functionality of constructing and executing SELECT statements. The main result of the work is additional functionality that was implemented in the visual query building interface of the application. The new functionality allows users to construct and execute PostgreSQL INSERT, UPDATE, and DELETE statements, including construction of the PostgreSQL-specific RETURNING clause and PostgreSQL-specific joining of tables in case of UPDATE and DELETE statements (FROM and USING clauses, respectively). In addition, the generation of SQL SELECT statements was improved by replacing LIMIT with the FIRST FIRST n ROWS clause. Moreover, improvements were made to the graphical user interface for the sake of better user experience. All the bugs that were found during development were fixed. At the beginning of the work the system was migrated from Node.js 8 to Node.js 18.

In the modern world our need to store digital data is increasing every day. A popular way of storing digital data is by using SQL-databases, i.e., databases that have been created by using a SQL Database Management System (DBMS). As a result there is an ever-increasing demand of people with knowledge of the SQL database language. PostgreSQL is among the most popular SQL DBMSs. This application lets the users to manipulate PostgreSQL databases by creating SQL data manipulation language statements of varying complexity through a graphical user interface. It offers users an alternative way of using databases and can aid in learning of the SQL language. The application implements the visual programming language Query By Example, devised by M. Zloof at IBM parallel to the development of the SQL language.

The work uses design science methodology. The result of the work is a technical artefact, that is implemented based on the requirements of users and the best practices of the domain.

Prior to the development process, the previous version of the software was analysed. Next the theoretical background of the problem was examined through relevant academic papers and articles. In addition, existing software with the functionality to visually create data manipulation SQL statements was analysed. The result of these analyses became the input to the development. The development process used the agile release planning method proposed by T. Norman. Firstly, the results of the work were validated through the feedback of three test users. The test users had to solve nine tasks and answer to a questionnaire. The test users were generally satisfied with the user interface. Secondly, the results of the work were validated by comparing it with two existing applications (MS Access and dbForge Studio For MySQL) that also allow us to visually construct SQL data modification statements. The author did not find any tools with such functionality for PostgreSQL.

The application uses the React framework to implement the frontend and Node.js JavaScript runtime environment to implement the backend. The source code of the application is protected by the MIT licence and is available at the GitHub repository: `https://github.com/v4hukomm/postgresql-visual-query-v3`.

The thesis is written in Estonian and is 47 pages long, including 8 chapters, 39 figures and 4 tables.

Lühendite ja mõistete sõnastik

API	Application Programming Interface, rakendusliides
CRA	Create React App on ametlikult toetatud viis React rakenduste loomiseks, mis loob arendajale mugava ja mitmekülgse arenduskeskkonna
CTE	Common Table Expression, ühine tabeli avaldis
HTTP	Hypertext Transfer Protocol, hüperteksti edastuse protokoll
Linter	Arendustööriist koodi analüüsimiseks, aitab tuvastada stiilivigu ja programmeerimisvigu
LTS	Long-term support, kestustugi
QBE	Query by Example, graafiline andmebaasikeel
SQL	Structured Query Language, tekstiline andmebaasikeel
Tellija	Antud töö kontekstis töö juhendaja
VSC	Visual Studio Code, arendajale mõeldud tekstiredaktor

Sisukord

Jooniste loetelu

Tabelite loetelu

1. Sissejuhatus

Meid ümbritsevas üha enam digitaliseeruvas maailmas kasvab igapäevaselt andmete hulk, mida on vaja digitaalselt talletada. Kõikvõimalikud erinevad digitaalsed süsteemid, olenemata nende keerukusest või suurusest, kasutavad oma töös andmeid, mida vajavad nende süsteemide kasutajad.

Andmete digitaalseks hoidmiseks ja haldamiseks on mitmeid erinevaid tehnoloogilisi võimalusi. Üks võimalus on hoida andmeid spetsiaalse tarkvara - andmebaasisüsteemide - abil loodud andmebaasides. Väga hästi on ajaproovile vastu pidanud ja ennast tõestanud SQL-andmebaasisüsteemid, kus andmete töötlemiseks ning andmebaasi haldamiseks kasutatakse andmebaasikeelt SQL (*Structured Query Language*). 2023. aasta detsembri seisuga on kümnest kõige populaarsemast andmebaasisüsteemist seitse SQL-andmebaasisüsteemid. [1] IEEE Spectrum 2023. aasta programmeerimiskeelte populaarsuse indeksis oli SQL seitsmendal kohal [2] (2022. aastal kuuendal kohal [3]) ja tööandjate ootuses esikohal (2022. aastal samuti esikohal [3]). Tööandjad ootavad, et arendajad oskaks *lisaks* muudele keeltele kindlasti ka SQLi. [4, 2]

Aina rohkem on ka äripoole inimestel soov ja vajadus andmebaasisüsteemide abil loodud andmebaasidesse talletatud andmeid otsida või hallata. Olenemata sellest, et SQL keele lausete struktuur ei ole lihtsamate ülesannete korral ülemäära keerukas, siis selles keeles andmekäitluse (andmete otsimise ja muutmise) ülesannete lahendamine ei ole triviaalne ettevõtmine. Sellest tulenevalt on töölise väljakoolitamine nii ajaliselt kui ka rahaliselt kulukas. Siinkohal on ühe võimalusena abi visuaalsetest andmekäitluse lausete koostamise töövahenditest, mille kasutaja ei pea tundma SQL keele süntaksi, vaid saab lause koostada visuaalselt, kasutades graafilisi elemente.

Lõputöö arendab edasi avatud lähtekoodiga veebirakendust *Postgres Visual Query Builder*. See tarkvara on mõeldud PostgreSQL andmebaasides [5] päringute (SELECT lausete) visuaalseks koostamiseks. Rakenduse esimene väljalase loodi E. Dzotsenidze bakalaureusetöös [6] ning järgmine väljalase loodi R. Pärnamäe magistritöö tulemusel [7]. Käesoleva töö eesmärk on realiseerida funktsionaalsuseid, mis jäid tegemata eelmistes väljalasetes ning teha ühtlasi täiendusi Pärnamäe töös välja toodud vigade kõrvaldamiseks. Edasiarenduste puhul arvestatakse Pärnamäe töös väljatoodud ettepanekutega, neid omalt poolt edasi arendades.

Kui küsida, miks sellist programmi vaja on, siis selline vahend suurendab andmebaasi kasutajate valikuvõimalusi. Täpselt nii nagu on palju erinevaid andmebaasisüsteeme, programmeerimiskeeli, kontoripakette või teenuse X pakkujaid ei tee halba ka see, kui on võimalik kasutada erinevaid mooduseid andmebaasikeele lausete koostamiseks. Nii on suurem võimalus, et igaüks leiab endale kõige meelepärasema lahenduse. Programm, mis on mõeldud spetsiifiliselt ühe andmebaasisüsteemi jaoks (antud juhul PostgreSQL), võimaldab paremini arvestada selle andmebaasisüsteemi andmebaasikeele erisustega võrreldes üldotstarbelise programmiga, mis on mõeldud paljudele erinevatele andmebaasisüsteemidele. Veebipõhise programmi kasutamiseks on vaja internetiühendust ja veebilehitsejat ning pole vaja oma arvutisse ise täiendavat tarkvara lisada. Täiendava tarkvara installeerimiseks võib kasutajal olla liiga vähe õiguseid, arvuti võib olla liiga vana ning eriti algajate puhul võib ka lihtsalt jääda puudu oskustest. Eriti just praeguse arendusega programmi lisatavad täiendused on sellised, mida konkureerivad programmid peaaegu ei paku - seega pakub töös arendatav programm ka lisandväärtust puhtalt funktsionaalsuse seisukohalt.

Töö peamine eesmärk on lisada andmete muutmise lausete (INSERT, DELETE, UPDATE) loomise funktsionaalsus. SQL andmekäitluskeelde kuulub ka MERGE lause, mis kombineerib INSERT, UPDATE ja DELETE lause. PostgreSQL toetab lisaks ka lauset, mis kombineerib INSERT ja UPDATE (nn upsert operatsioon [8]) ja mis on vanem süntaks sellise ülesande lahendamiseks, mida lahendab MERGE. [9] MERGE ja upsert tuge töö mahtu arvestades selles arendustsüklis ei lisata. Olemasolevates tarkvarades on andmete muutmise lausete koostamise võimalus üldiselt haruldane või on saadaval ainult tasulistes versioonides. Lisaks uue funktsionaalsuse lisamisele parandatakse olemasoleva süsteemi kasutusmugavust. Edasiarenduses lähtutakse Dzotsenidze ja Pärnamäe töödes paika pandud põhimõttest, et rakenduse sihtgrupiks on SQLi juba veidi oskavad kasutajad ehk rakendus ei ole mõeldud kasutajatele, kellel ei ole mingeid teadmisi SQL lausete koostamisest.

Lõputöös järgitakse disainiteaduse metoodikat. [10] See tähendab, et töö tulemusena luuakse tehniline artefakt e tehis, mis realiseeritakse kasutaja nõuete ja valdkonna parimate praktikate põhjal ja lõpuks valideeritakse valminud tulemust. Töö tulemuse kontrollimiseks koostatakse erineva raskusastmega ülesanded, mis antakse testkasutajatele kasutajaliideses lahendamiseks. Lisaks küsitakse tagasisidet üldise kasutajakogemuse ja -mugavuse kohta. Saadud tagasiside põhjal tehakse rakenduses vajalikud muudatused ja parandused.

Rakenduse uue versioon lähtekood on saadaval GitHubi hoidlas: `https://github.com/v4hukomm/postgresql-visual-query-v3`.

Lõputöö koosneb kaheksast peatükist. Teises peatükis tutvustatakse arendatavat rakendust

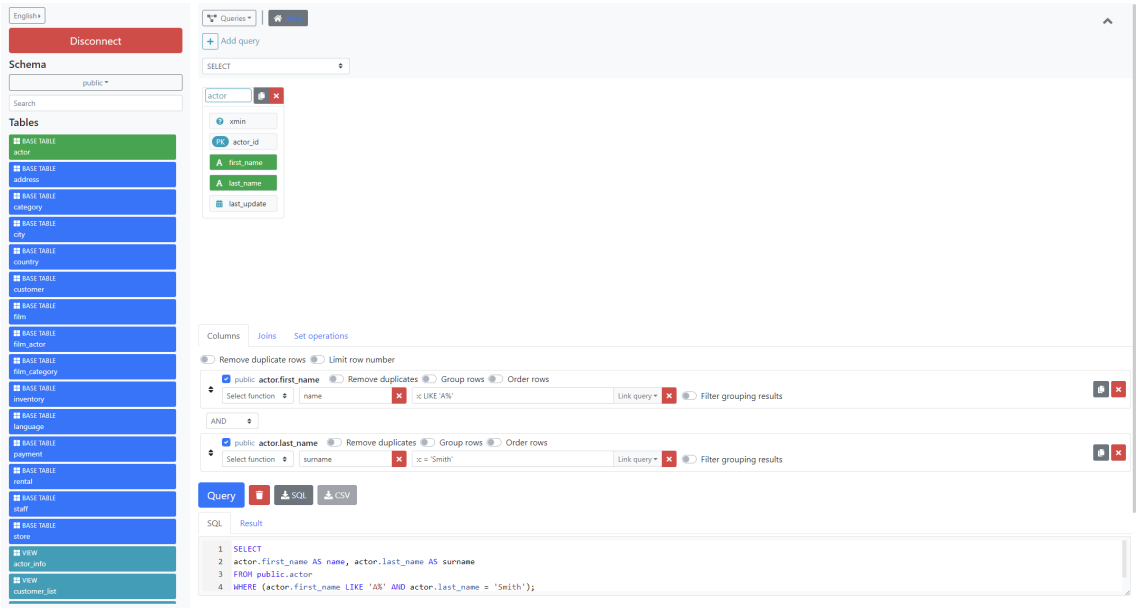
ning tehakse ülevaade arendusprotsessist ja kasutatud tööriistadest. Kolmandas peatükis kirjeldatakse tarkvara seisu enne arenduse algust. Neljandas peatükis kirjeldatakse teoreetilist tausta ja analüüsitakse juba olemasolevat tarkvara. Viiendas peatükis esitatakse rakendusele uued funktsionaalsed nõuded, kirjeldatakse väljalaske plaani ning kasutajaideses tehtud täiendusi. Kuuendas peatükis valideeritakse rakendust läbi testkasutajate tagasiside. Seitsmendas peatükis kirjutatakse, millised võiksid olla järgmised täiendused rakendusele. Kaheksandas peatükis esitatakse kokkuvõte lõputööst.

2. Metoodika

Selles peatükis antakse lühiülevaade arendsuse objektist e arendatavast tarkvarast ning kirjeldatakse arendusprotsessi ja selle käigus kasutatud tööriistu.

2.1 Ülevaade objektist

Postgres Visual Query Builder on PostgreSQL andmebaasides päringute (SELECT lausete) graafiliseks koostamiseks mõeldud avatud lähtekoodiga veebirakendus, mis on loodud Tallinna Tehnikaülikooli üliõpilaste poolt lõputööde tulemusena. Peale sisselogimist ja andmebaasiga ühendumist kuvatakse kasutajale päringute loomise kasutajaliides (vt Joonis 1). Vaikeväärtusena valitakse *public* andmebaasi skeem, mida kasutaja saab vahetada, et näha teiste skeemide tabeleid. Tabelite juures on ära näidatud nende liik (baastabel, vaade, materialiseeritud vaade e hetktõmmis ning väline tabel) ning neid on võimalik otsinguriba abil nime järgi otsida. Rakendus võimaldab koostada andmete otsingu lauseid (SELECT lauseid), sh tabelites olevaid andmeid ühendada ning kasutada alampäringuid ja viia läbi hulgateoreetilisi operatsioone. Tabelitele ja veergudele on võimalik määrata aliasi e varjunimesid. Lisaks on võimalus päringu tulemusi tabeli veergude põhjal grupeerida ja sorteerida ning piirata tulemuste arvu või eemaldada tulemustest korduvad read. Rakendust on võimalik kasutada nii inglise kui ka eesti keeles.



Joonis 1. *Postgres Visual Query Builder veebipõhine kasutajaliides.*

2.2 Arendusprotsess

Arendusprotsessis kasutatakse agiilse arenduse põhimõtteid ja parimaid praktikaid.

Realiseeritav funktsionaalsus lepitakse kokku tellijaga (juhendajaga) ja arendus jaotatakse iteratsioonideks ehk *sprintideks*. Rakenduse nõuded (funktsionaalsed nõuded) kirjutatakse üles kasutuslugudena ning igale kasutusloole määratakse olulisus ja raskusaste. [11] Nõuded järjestatakse vastavalt olulisusele ning jagatakse iteratsioonide vahel selliselt, et iga iteratsiooni raskusastmete summad oleksid sarnase suurusega. Peale iga iteratsiooni lõppu kohtutakse tellijaga (juhendajaga), et näidata ette valminud funktsionaalsus ning saada temalt tagasisidet.

Arendusprotsess sarnaneb kõige enam agiilsele arendusmetoodikale *Scrum* [12]. Kasutusel on iteratsioonid e *sprindid* ning iteratsiooni järgsed kohtumised lõppenud iteratsiooni tulemuste ülevaatamiseks ja järgmise plaanimiseks. Kuna autor arendab rakendust üksinda, siis on välja jäetud meeskondadele mõeldud põhimõtted nagu igapäevased koosolekud ja iteratsiooni järgsed *Retro* koosolekud.

Töö tulemuse kontrollimiseks koostatakse erineva raskusastmega ülesanded, mis antakse testkasutajatele kasutajaliideses lahendamiseks. Lisaks küsitakse neilt tagasisidet üldise kasutajakogemuse ja -mugavuse kohta.

2.3 Kasutatud tööriistad

Arendamisel kasutas autor integreeritud arenduskeskkonda *Visual Studio Code* [13]. VSC on optimiseeritud kaasaegsete veebi- ja pilverakenduste loomiseks ja silumiseks. Samuti on Google'i otsingutulemuste põhjal (2023 november seisuga) tegu populaarsuselt teisel kohal oleva arenduskeskkonnaga. [14] Autor valis selle kasutamiseks aga peamiselt enda kasutusharjumuste ja -mugavuse tõttu. Koodihaldussüsteemina kasutati arenduse ajal ülikooli poolt pakutatavat *GitLab* lahendust. Kohtumisteks ja koosolekuteks juhendajaga kasutati ärisuhtluse platvormi *Microsoft Teams* [15]. Testkasutajatelt tagasiside saamiseks kasutati küsitluste haldustarkvara *Google Forms* [16]. Nii nagu seda tehti ka varasemate versioonide korral, avaldatakse valminud tarkvara uue versiooni lähtekood maailmale GitHub keskkonnas.

3. Tarkvara seis enne arendust

Selles peatükis kirjeldatakse tarkvara seisu peale eelnevalt tarkvara arendanud üliõpilaste poolt kahe arendustsükli läbimist ning enne seda kui käesoleva töö autor seda edasiarendama hakkas.

3.1 Kasutajaliides

Rakenduse kasutajaliides võib olla esimest korda kasutajale kirju ja natuke segane. Rakenduse taustavärvina on kasutatud valget ning komponendid ja tegevused on esile toodud erinevate värvide abil.

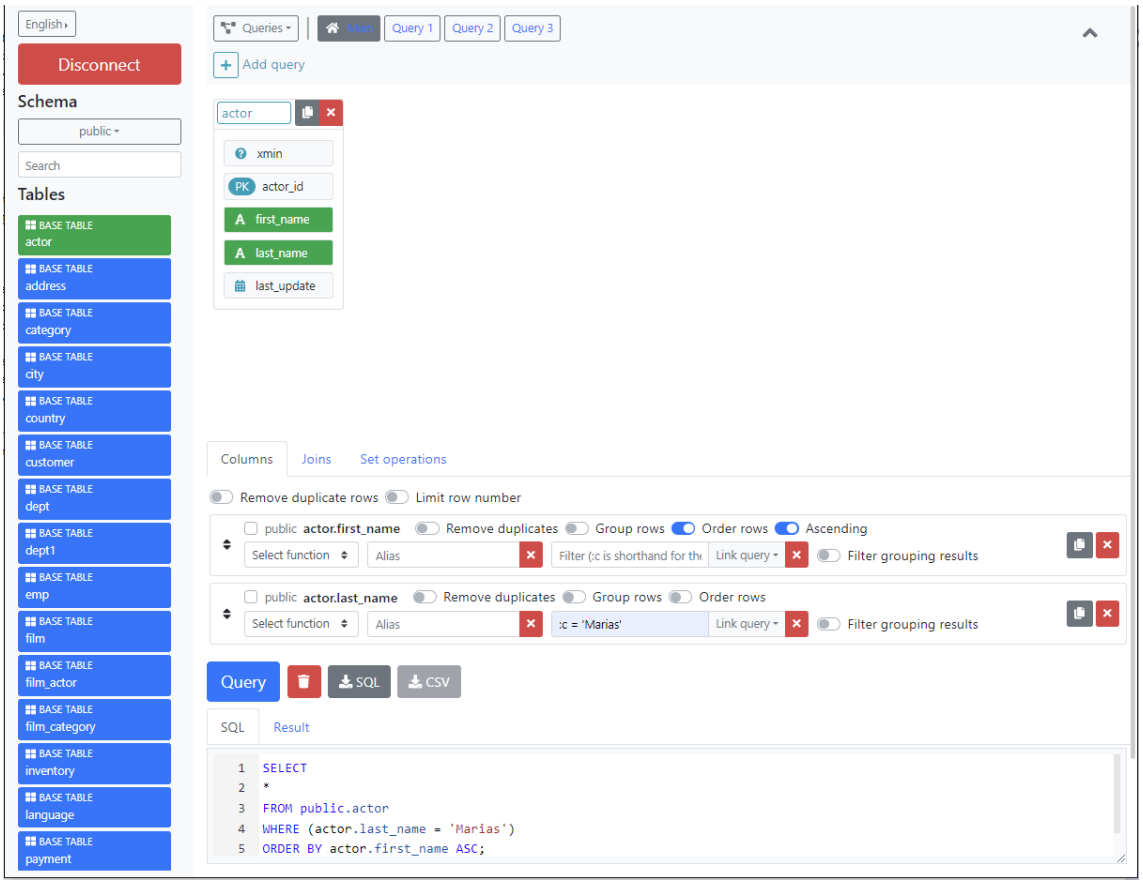
Mitmete komponentide funktsionaalsus või eesmärk on ebamäärane ja võib tekitada segadust. Selle näiteks on punane prügikasti märgiga päringu kustutamiseks mõeldud nupp (vt. Joonis 2) Kasutajale ei ole täielikult selge, mida nupule vajutamisega kustutatakse. Samuti tekitab segadust, mis eesmärki täidab päringute listi rippmenüü. Kasutajakogemust parandavad mitmel pool kasutusel olevad sisestusviibad ja kohtspikrid, mis annavad kasutajale selgelt teada, millist funktsionaalsust komponendid pakuvad. Samuti säilitavad visuaalselt sarnased või ühesugused komponendid ühese funktsionaalsuse, mis tagab kasutajakogemuse järjepidevuse.

3.2 Tagarakendus

Tagarakendus on realiseeritud Node.js [17] JavaScript käituskeskkonnas ning kasutab Express [18] raamistikku rakendusliideste (API e *Application Programming Interface*) loomiseks. Andmebaasisüsteemiga ühendumiseks ja suhtlemiseks on kasutusel raamistik node-postgres. [19]

3.3 Eesrakendus

Eesrakendus on ehitatud React JavaScript [20] raamistiku abil ja rakenduse olekut hallatakse läbi Redux [21] teegi. Tagarakendusega suhtlemiseks on kasutusel veebilehitseja jaoks mõeldud HTTP klient axios [22]. Kasutajaliidese komponentide disainimiseks on kasutatud Bootstrap [23] ja Reactstrap [24] raamistikke. Ka andmebaasi päringud koostatakse eesrakenduses - selleks on kasutusele võetud Squel.js [25] teek.



Joonis 2. *Postgres Visual Query Builder kasutajaliides. Prügikasti nupp on näha Query nupu kõrval, päringu loomise vaate alumises osas.*

Eesrakenduse kood on lihtsasti loetav ja on kasutatud piisavalt selgitavaid funktsioonide ja muutujate nimesid. Samuti on kaustade struktuur arusaadav ja loogiline. Negatiivse poole pealt võiks välja tuua koodi käekirja ebaühtluse. See tähendab, et rakenduses on kasutatud segamini Reacti funktsionaalseid ja klassi komponente. Üldiselt kasutatakse 2023. aasta sügise seisuga Reacti klassi komponente väga harva kuna tegemist on pärand-funktsionaalsusega. React versioonis 16.8 lisati funktsionaalsetele komponentidele erisus *React Hooks*, millega on võimalik rakenduse olekut hallata. [26] Enne seda uuendust oli see funktsionaalsus võimalik vaid klassi komponentides. Soovituse funktsionaalseid komponente kasutada leiab ka Reacti ametlikust dokumentatsioonist. [27] Kui puudub mõjuv põhjus nende kasutamiseks, siis tuleks klassi komponendid ümber teha funktsionaalseteks komponentideks. Funktsionaalsed komponendid vähendavad komponendi keerukust ja on lihtsamini loetavad.

4. Teoreetiline taust

Selles peatükis kirjutatakse käesoleva töö paremaks mõistmiseks olulisest teoreetilisest taustast.

4.1 Visuaalsed programmeerimiskeeled

Tänapäeval on programmeerimiskeeled põhiliselt tekstipõhised. Nende kasutamisel peab kasutaja kirjeldama tekstiliselt programmi tegevuste jada, mis on vajalikud soovitud tulemuseni jõudmiseks (imperatiivsed programmeerimiskeeled) või programmi täitmiselt oodatavat tulemust (deklaratiivsed programmeerimiskeeled). SQLi andmekäitluskeel on tekstiline ja deklaratiivne. Visuaalsed programmeerimiskeeled võimaldavad kasutajal programmeerida kasutades graafilisi elemente. Ka visuaalsed programmeerimiskeeled võivad olla imperatiivsed või deklaratiivsed.

Visuaalsete programmeerimiskeelte üldine eesmärk on lihtsustada programeerimise õppimist algajatele ning teha see huvitavamaks ja kaasahaaravamaks. Visuaalsed programmeerimiskeeled aitavad programmeerijad kolme takistuse ületamisel [28]:

- **Süntaks**: Vähendada või täielikult kõrvaldada süntaksi vigade tekkimist, kasutades graafilisi elemente. Tekstipõhistes programmeerimiskeeltes võib ühe semikooloni puudumine takistada terve programmi toimimist, mis võib olla algajale programmeerijale ärritav ja segadust tekitav. Muidugi on ka graafilisel keelel süntaks, kuid lootus on, et graafiline keskkond võimaldab selle võimalike vigade hulka vähendada.
- **Semantika**: Läbi visuaalsete abifunktsioonide kirjeldada programmi graafiliste elementide tähendust ja funktsionaalsust. Enamasti on see lahendatud kohtspikrite abil või üldisema visuaalse abipaneeliga, kus on kirjeldatud elementide omadused.
- **Pragmatika**: Visuaalselt kasutajale väljendada programmi täitmist või tulemuse muutumist erinevates olukordades.

Kõige populaarsem visuaalne programmeerimiskeel on MIT poolt loodud Scratch [29], mis lubab kasutajal graafiliste elementide abil rakendusi luua. See keel oli TIOBE programmeerimiskeelte populaarsuse indeksis 2024. aasta jaanuari seisuga kümnendal kohal ja tõusis võrreldes 2023. aasta jaanuariga kümme kohta. [30] Scratch loodi õppevahendina peamiselt lastele ja noortele vanuses 8 kuni 16 aastat, kuid on ajaga kasvatanud ka arvestava täiskasvanutest kasutajaskonna. Scratchi kasutatakse peamiselt põhi- ja keskkoolides

õpilastele programmeerimise põhiteadmiste õpetamiseks ning on kõigile tasuta saadaval. [29] Visuaalsed programmeerimiskeeled on tõestanud ennast positiivsete tulemustega arvutiteaduste õpetamisel ja õppehuvi säilitamisel. [31] Seda eriti nooremate inimeste puhul ja õppehuvi osas kuni kõrgkooli kursusteni välja. Graafilised keeled loovad unikaalse keskkonna loovuse probleemide lahendamiseks, mis erineb teksipõhiste keelte kasutamisest. Kasutajad saavad lihtsamalt ja rohkem eksperimenteerida oma lahendustega, kui neil on visuaalne esitus programmi toimimisest. [31]

Lisaks õppevahendina kasutamisele saab tuua välja järgmised visuaalsete keelte eelised. [32]

- **Kiirem arendusprotsess** - Arendajad saavad programme ehitada kiiremini, ühendades graafiliselt eeldisainitud komponente. See kiirus on eriti kasulik lihtsate rakenduste, prototüüpide või korduvate ülesannete automatiseerimisel.
- **Lihtsustab koostööd ja suhtlust** - Graafiline esitus programmi toimimise loogikast aitab arendusmeeskondade liikmetel lihtsamalt ideid täpsustada ja probleeme lahendada, vaatamata liikme arenduskogemuse tasemele.
- **Innovatsioon ja loovus** - Kiire prototüüpimine lubab testida palju erinevaid ideid ja lahendusi, suurendades innovatsiooni ja loovust.
- **Alternatiivne tööviis** - Erinevatele kasutajatele sobivad erinevad tööviisid. Kasutaja võib koodi tekstilise kirjutamise asemel eelistada graafilise kasutajaliidese kasutamist. Visuaalsete programmeerimiskeelte abil saab igaüks kasutada endale meelepärast varianti.

4.2 Query by Example

Postgre Visual Query Builder realiseerib graafilise programmeerimiskeele *Query by Example* e QBE, mis loodi paralleelselt SQL keele arendusega. Tegemist on valdkonnapõhise programmeerimiskeelega, mida saab kasutada relatsiooniliste andmebaaside programmeerimisülesannete lahendamiseks. Esimest korda kirjeldati *QBE*d M.M.Zloofi 1975.aasta artiklis *"Query by Example"* [33] ning hiljem täiendatud kirjelduste ja edasiarendustega 1977.aasta artiklis *"Query by Example: a database language"* [34].

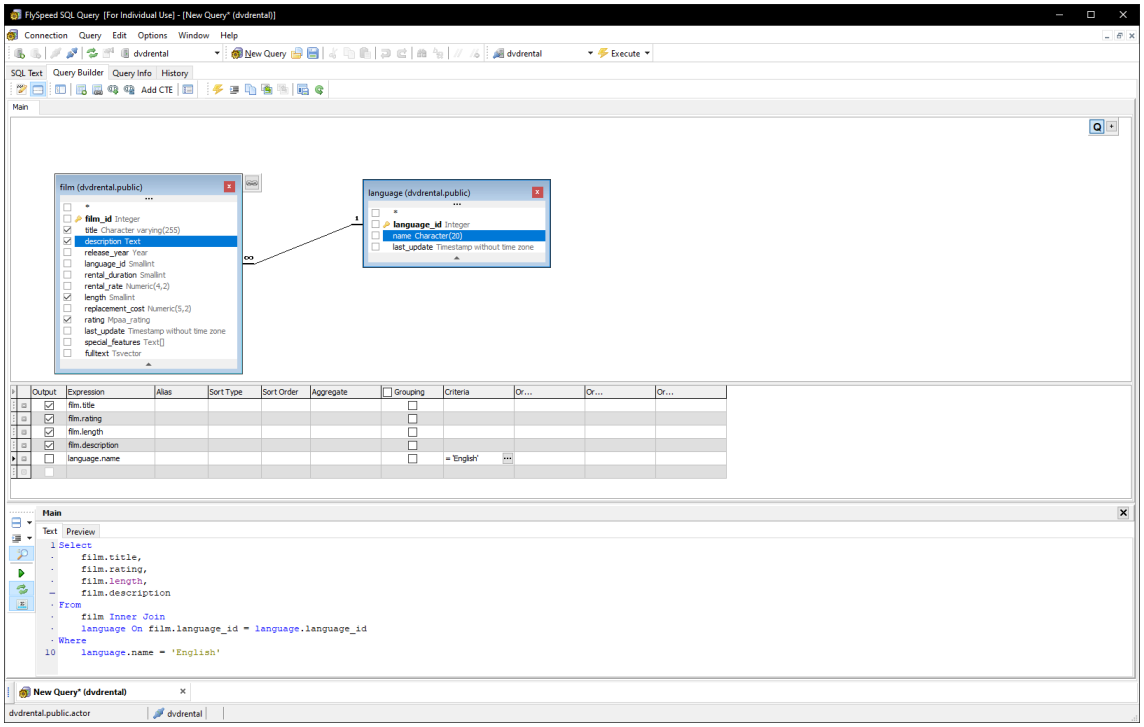
Algselt võimaldas *QBE* koostada ainult andmete otsimise lauseid. Hiljem täiendati keelt võimalustega koostada andmete muutmise e manipuleerimise ja andmestruktuuride haldamise lauseid. 1977.aasta artiklis on kirjeldatud andmete muutmise lauseid kahes erinevas vormis - lihtsad operatsioonid ja *Query-dependent* (päringust-sõltuvad) operatsioonid. Lihtsad operatsioonid kasutavad sisendiks ainult staatilisi andmeid, *Query-dependent* operatsioonid võimaldavad sisendina kasutada lisaks staatilistele andmetele ka teiste päringute

tulemusi.

INSERT, DELETE ja UPDATE laused kirjeldatakse *QBE*s sarnaselt SELECT lausetele. Ainus erinevus on unikaalne graafiline indikaator, mille järgi on võimalik lausetüüpe üksteisest eristada. Kuigi lauseid esitatakse visuaaselt sarnaselt, ei tähenda see, et samasugused graafilised esitused tagaksid alati ühesuguse käitumise. Näiteks tähendab tühi väli DELETE lauses, et välja väärtust ei kontrollita, UPDATE lauses, et see väli ei vaja uuendamist ning INSERT lauses asendatakse tühi väli NULL'iga.

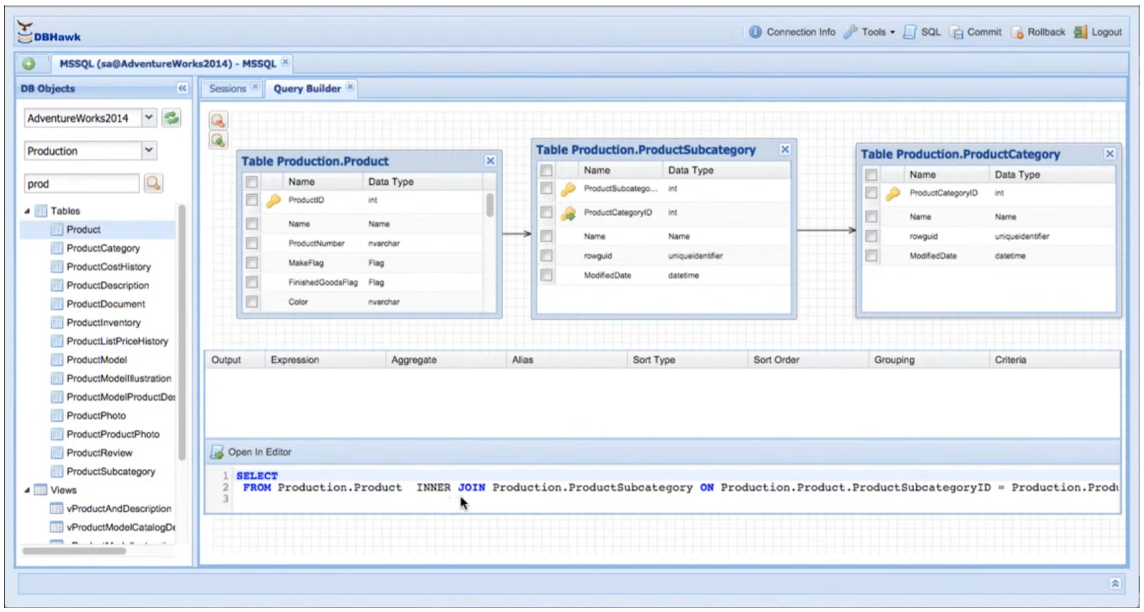
4.3 SQL lausete graafilise koostamise vahendid

Leidub palju erinevaid rakendusi, mis võimaldavad või ongi mõeldud graafiliselt SQL päringute koostamiseks. Kasutaja saab koostada päringu graafilises keskkonnas ja tulemuse otse käivitada või lasta tõlkida tulemuse SQL keelde. Peamiselt on tegu tasuliste töölauarakendustega nagu näiteks töölaua andmebaasisüsteem *MS Access* [35], mille üks funktsionaalsus on *Query Designer*, *FlySpeed* [36] (vt Joonis 3), *dbForge* [37], *DbVisualizer* [38] või *Active Query Builder* [39]. Samas on ka veebipõhiseid rakendusi nagu *Oracle APEX* [40], *Skyvia Query Builder* [41], *DBHawk* [42] (vt Joonis 4) või *Draxlr* [43]. Üldiselt on ka veebipõhised rakendused tasulised.



Joonis 3. *Rakenduse FlySpeed kasutajaliidese päringu loomise vaade.*

Põhifunktsionaalsuse poolest võimaldavad kõik rakendused koostada lihtsaid SELECT



Joonis 4. *Rakenduse DBHawk veebiliidese päringute loomise vaade.*

lauseid. Tasuta rakenduste puhul on keerulisemad funktsionaalsused enamasti tasulised. Enamik rakendustest toetavad mitmeid erinevaid populaarseid SQL-andmebaasisüsteeme nagu *Oracle*, *MySQL* ja *PostgreSQL*, kuid on ka ainult kindlatele andmebaasisüsteemidele mõeldud rakendusi nagu *dbForge Studio for MySQL*. Enamasti võimaldavad kõik rakendused visuaalselt koostatud päringud ka tekstilisteks SQL lauseteks ümber tõlkida. Vastupidist tõlget tekstilisest graafiliseks toetavad vähesed vahendid, üheks neist *MS Access*.

Käesoleva töö eesmärgist tulenevalt otsis autor analüüsimiseks rakendusi, mis lubaksid visuaalselt koostada lisaks SELECT lausetele ka andmemuudatuse lauseid INSERT, DELETE ja UPDATE. SQL standard toetab ka MERGE lauset, mis kombineerib INSERT, UPDATE ja DELETE lause, kuid sellele antud töös ei keskendutud. Lisaks sellele otsiti veel rakendusi, mis toetaks ühiste tabeli avaldiste e *common table expressions* (CTE) graafilisel viisil loomist, sest kuigi käesolevas töös ei jõuta nende tuge lisada oleks see üks olulisemaid järgmisi täiendusi, mida tarkvarasse tuleks teha. Selliste rakenduste otsimine ja analüüsimine osutus üpris vaevaliseks, kuna visuaalsed SQL lausete koostamise vahendid on enamasti moodulid tasulistes andmebaasihalduse tööriistades. Paljud neist rakendustest ei paku tasuta prooviperioodi või -versiooni.

4.3.1 MS Access

Microsoft Access on põhiliselt üksikkasutaja andmebaaside loomiseks mõeldud andmebaasisüsteem, millel on SQL lausete loomiseks ja kohandamiseks mitmeid kastuajaliidese vaateid. Üheks nendest vaadetest on *Query Design View*, (Joonis 5) mis võimaldab ka-

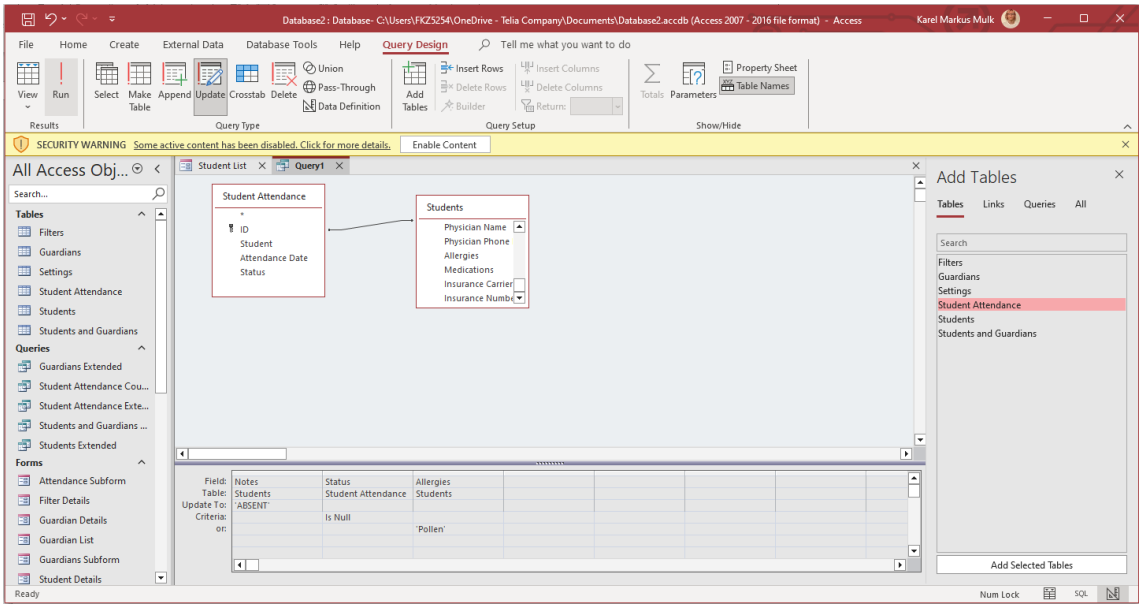
sutajal graafiliselt koostada lisaks SELECT lausetele ka kõiki andmebaasisüsteemi poolt toetatud andmete muutmise lauseid (INSERT, DELETE ja UPDATE). *Query Design View* kasutajaliides on oma kujult väga sarnane *QBE*s kirjeldatud lausete visuaalse esitusega.

INSERT lauset graafiliselt koostades saab *Access*is kasutada alampäringuga sisestamist ehk kasutades SELECT lauset ja väärtustena kirja pandud ridade ühe kaupa lisamist, mille tulemusel tekib ka SELECT lause, kuid ilma FROM klauslita. SELECT lausega päritud väärtuste read lisatakse sihttabelisse valitud veergudesse. Kasutajakogemuse vaatest ei ole üksiku rea lisamine intuitiivne ja nõuab sügavamat teadmist SQList ja selle võimalustest.

DELETE lause koostamisel saab visuaalselt luua liht- ja liitotsingutingimusi WHERE klausliga. Otsingutingimuste lisamine on disainitud väga sarnaselt QBEs välja pakutud lahendustega. Lisaks on võimalik kasutada tabelite ühendamisoperatsioone, et kontrollida otsingutingimusi teiste tabelite põhjal. Seda tehakse JOIN operaatori abil.

UPDATE lause visuaalsel koostamisel on otsingutingimuste lisamine samasugune nagu DELETE lause puhul. Ka UPDATE lauses on võimalik kasutada tabelite ühendamisoperatsioone. Ühendamisoperatsioonidega on *Access*is võimalik ka teha keerulisemaid lauseid, millega saab mitmes tabelis korraga andmeid muuta.

UPDATE ja DELETE laused toetavad ka nö mitmesuunalist lausete koostamist. See tähendab, et neid on võimalik teha ümber SELECT lauseteks koos otsingutingimuste säilimisega, et kontrollida, kas kustutatakse või uuendatakse just need read, mis soovitud.



Joonis 5. *MS Access Query Design View vaade.*

4.3.2 dbForge Studio for MySQL

dbForge Studio for MySQL on graafiline SQL päringute loomise töölauarakendus, mis on mõeldud spetsiifiliselt *MySQL* andmebaasidele. Antud tööriistaga on võimalik koostada kõiki SQL lauseid, sealhulgas ka andmete muutmiseks mõeldud lauseid. Antud vahendit autor ise katsetada ei saanud kuna seda saab ainult tasuliselt - seega põhineb ülevaade demovideotel tarkvara kohta [44].

*dbForge Studio*s saab luua kahte põhilist INSERT lausete variatsiooni - lause, kus lisatav rida konstrueeritakse VALUES klausli abil või lause, kus tabelisse lisatakse SELECT lause tulemus. VALUES klausli kasutamisel on rakenduses aga piirang, mis lubab sisestada tabelisse korraga ainult ühe rea. Sellest tulenevalt on korraga mitme rea lisamine tabelisse võimalik ainult SELECT lausega. Kui kasutajal on soov lisada mitu rida kasutaja poolt defineeritud väärtusi, siis tuleb koostada iga rea jaoks uus lause.

*dbForge Studio*s saab luua lihtsaid DELETE lauseid. Lausesse saab lisada WHERE klausli, mille abil saab piirata kustutatavaid ridu. Rakendus lubab DELETE lausete WHERE klauslis määrata piiravaid tingimusi ainult veergudele, mis on sihttabelis. See tähendab, et WHERE klauslis ei ole võimalik kasutada alampäringuid. Visuaalselt ei ole võimalik ka luua ühendamisoperatsioone kasutavaid DELETE lauseid (kustutamine läbi ühendamise), mida saab MySQLis kasutada.

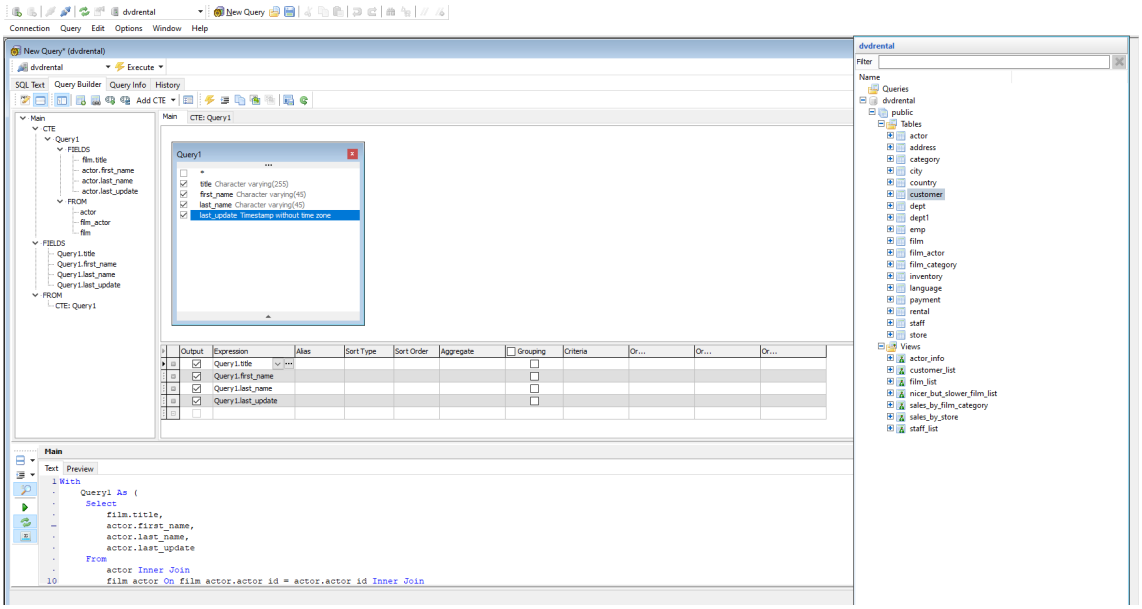
*dbForge Studio*s UPDATE lausete loomise puhul valib kasutaja tabeli ja soovitud read, kus andmeid muuta ning seejärel määrab valitud ridade valitud väljadele uued väärtused. Sarnaselt DELETE lausetele, saab ka UPDATE lausete WHERE klausli jaoks kirjeldada ainult lihtsaid piiravaid tingimusi sihttabeli veergudele ning seega ei saa kasutada alampäringuid. Visuaalselt ei ole võimalik ka luua ühendamisoperatsioone kasutavaid UPDATE lauseid (uuendamine läbi ühendamise), mida saab MySQLis kasutada.

Lisaks eelpool väljatoodud WHERE klausli koostamise piirangutele, ei ole võimalik ka kasutada keerukamaid liit-otsingutingimusi, kus kasutatakse alamtingimuste ühendamiseks nii AND kui OR operaatoreid. Kõik määratud tingimused liidetakse kokku AND operatsiooniga ehk kui kasutajal on soov rakendada mitut otsingutingimust OR operatsiooniga, siis peab ta selle jaoks looma eraldi laused. See piirang ei kehti ainult andmete muutmise lausetele, vaid ka SELECT lausetele. Positiivne külg on see, et rakendus salvestab WHERE klausli tingimused päringutüübi vahetamisel ehk kasutaja saab enne DELETE või UPDATE lause käivitamist SELECT lause abil veenduda, et lausega kustutatakse või muudetakse soovitud andmeid. Sellist lähenemist soovitatakse ka *dbForge Studio* ametlikes videojuhendites.

4.3.3 FlySpeed

Flyspeed SQL Query lubab koostada erineva keerukusega SELECT lauseid. Toetatud on tabelite ühendamisoperatsioonid, alampäringute kasutamine, grupeerimine, agregeerimisfunktsioonid ning tabelite ühendi, ühisosa ja vahe operaatorite kasutamine.

FlySpeed SQL Query oli ainus autori poolt analüüsitud rakendustest, mis toetas ühiste tabeli avaldiste (CTE, *Common Table Expression*) loomist graafilises keskkonnas (Joonis 6). Kuna *FlySpeed* ei toeta rekursiivseid CTE päringuid (WITH RECURSIVE), siis ei erine rakenduses CTEde loomine tavaliste alampäringute loomisest. Selleks, et kasutaja suudaks päringutüüpe eristada, on CTE päringutele lisatud visuaalsed identifikaatorid.



Joonis 6. *CTE loomine rakenduses FlySpeed.*

CTE päringud lisatakse kohe nende loomisel põhipäringusse WITH klausliga. *Flyspeed* lubab korraga kasutada ka mitut CTEd. Ühe märkusena võib välja tuua, et ühine tabeli avaldis jääb päringusse ka siis, kui see põhipäringust kustutatakse. See tähendab, et isegi kui midagi CTEst ei küsita, siis WITH klausel koos kõikide loodud CTEdega jääb päringusse alles. Klausel kaob päringust alles CTE enda kustutamisel.

5. Arendustöö ja selle tulemused

Selles peatükis antakse ülevaade tarkvara uue väljalaske tulemustest alatest kavandamisest kuni realiseerimiseni.

5.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded koguti kokku koostöös juhendajaga peale autori poolset rakenduse analüüsi. Algselt ei määratud nõuetele olulisust ja raskusastet. Nõuete kogumisel lähtuti juhendaja soovidest, autori poolt analüüsi käigus tehtud tähelepanekutest ning eelnevas magistritöös [7] realiseerimata jäänud funktsionaalsustest ja edasiarenduse ideedest.

Funktsionaalsed nõuded on kirja pandud kasutuslugudena (*user story*) (Tabel 1). Kasutuslood on kirjeldatud kasutaja perspektiivist ning võimaldavad lühidalt ja konkreetselt edasi anda kasutaja soovid funktsionaalsusele. Kasutuslood järgivad formaati "<kasutajatüübina> tahan teha <tegevust>, et <põhjus>". [45] Kasutuslugudele määrati unikaalsed identifikaatorid, et neid oleks lihtne üksteisest eristada. Kasutuslood ei ole tabelis olulisuse järgi sorteeritud. Kasutuslugude juures on viited SQL lausete tüüpidele või SQL konstruktsioonidele, mis sellele funktsionaalsusele vastavad.

Tabel 1. Tarkvara kasutuslood.

ID	Kasutuslugu
1001	Kasutajana tahan ühendamise lisamisel tabeleid kohe läbi valikukasti valida, ilma, et neid peaks külgribalt valima, et lausete koostamine oleks lihtsam ja kiirem.
1002	Kasutajana tahan, et päringute eemaldamine oleks loogilisem ja lihtsam, et saaksin lauseid kiiremini koostada.
1003	Kasutajana tahan tabelisse lisada enda kirja pandud väärtustest koosnevaid ridu, et andmebaasis oleks vajalik ja piisav hulk tõeseid väited reaalse maailma olemite kohta. (INSERT).
1004	Kasutajana tahan tabelisse lisada uusi alampäringuga leitud ridu, et andmebaasis oleks vajalik ja piisav hulk tõeseid väited reaalse maailma olemite kohta. (INSERT).
1005	Kasutajana tahan tabeli ridade väärtusi uuendada, et andmebaasis oleks ainult tõesed väited reaalse maailma olemite kohta. (UPDATE).
1006	Kasutajana tahan tabeli ridu kustutada, et andmebaasis oleks ainult tõesed väited reaalse maailma olemite kohta. (DELETE).
1007	Kasutajana tahan arvutada iga rea jaoks koondtulemuse ridade põhjal, mis on mingil viisil selle reaga seotud. (OVER, PARTITON BY).
1008	Kasutajana tahan koostada ühised tabeli avaldisi (CTE), et loodud andmebaasikeele lause oleks parema loetavusega.
1009	Kasutajana tahan andmete kustutamise lausetes kasutada tabelite ühendamisoperatsiooni (USING), et saaksin andmete kustutamisel kontrollida tingimuste täidetust teiste tabelite põhjal. (DELETE)
1010	Kasutajana tahan ridade uuendamise lausetes kasutada tabelite ühendamisoperatsiooni (FROM), et saaksin andmeid uuendada teiste tabelite abil. (UPDATE)
1011	Kasutajana tahan tabeli andmete muutmise lausetes (INSERT, DELETE, UPDATE) näha päringu tulemusena muudetud andmeid. (RETURNING)

5.2 Väljalaske plaan

Väljalaske plaaneerimiseks kasutatakse T. Normani poolt kirjeldatud agiilset väljalaske planeerimist [11]. Normani välja pakutud väljalaske planeerimise meetodit on lihtne järgida kuna see katab peamised agiilse arenduse põhitõed ja -praktikad. Selline lähenemine sobib erineva keerukuse, projekti suuruse ja meeskonna suurusega arenduste jaoks. See oli ka peamine põhjus selle väljalaske planeerimise meetodi valikus.

Esimese asjana kirjeldatakse artiklis tegemata tööde nimekirja e *product backlog* kirja panemist. Artiklis on selline nimekiri kirjeldatud lihtsalt ja ainult nelja väljaga:

- **ID** – Kasutusloo (töö) unikaalne identifikaator.
- **Kasutuslugu** – Kasutuslugu kirjutatud lihtsa pealkirjana. Mahukamas nimekirjas võib kirjas olla ka terve kasutuslugu, täpsemad kriteeriumid või kategoriseerimised

jne.

- **Töö olulisus** – Võimaldab järjestada kasutuslood tooteomaniku või tellija jaoks tähtsuse järjekorras.
- **Pingutuse hinnang** – Ligikaudne hinnang kui suurt pingutust kasutusloo realiseerimine nõuab. Selle määramise sisendid võivad olla meeskondadel ja projektidel erinevad, olenevalt nende keerukusest ja suurusest. Üldiselt on need töö ajaline mahukus, keerukus, riskid, ebakindlad sõltuvused jms.

Tegemata tööde nimekirjas kasutatakse jaotises 5.1 kirjeldatud kasutuslugusid koos nendele määratud unikaalsete identifikaatoritega.

5.2.1 Kasutuslugude olulisuse määramine ja hinnangute määramine

Kui kasutuslood olid paigas ja tabelisse pandud, siis lisati neile pingutuse hinnang (Tabel 2). Hinnangu sisendiks oli autori üldine tunnetus töö realiseerimiseks eelnevalt rakendusele tehtud analüüsi põhjal. Hinnangute mõõdikuks kasutati Fibonacci jadasse kuuluvaid arve. [46] Fibonacci jada on arvujada, kus esimesed kaks arvu on 0 ja 1 ning iga järgmine arv on jada kahe eelmise arvu summa. See on kõige levinum viis hinnangute esitamiseks ning oli kasutusel ka Normani artiklis. [11]

Järgmisena määrati kasutuslugudele koos juhendajaga olulisuse hinnangud (Tabel 2). Olulisuse määramisel lähtuti peamiselt sellest, mis tüüpi tööga oli tegu. Kõige olulisemad olid olemasolevate funktsionaalsuste ja kasutusmugavuse parandamised ning sellele järgnesid täiesti uute funktsionaalsuste lisamised. Uute funktsionaalsuste puhul arvestati mitme töö omadusega. Kõige tähtsam oli juhendaja hinnang funktsionaalsuse tähtsusele rakenduse vaatest. Edasi peeti oluliseks töid, mille tulemust sai kasutada ka teistes töödes. Näiteks DELETE lause koostamisel realiseeritakse WHERE klausli koostamine, mida saaks kasutada ka UPDATE lause realiseerimisel. Kuna UPDATE lause on keerulisem, saab DELETE suurema olulisuse hinnangu. Kui tööd olid võrdse tähtsusega, siis eelistatud olid väiksema pingutuse hinnanguga tööd. Kui töö sõltus mõnest teisest tööst, siis oli see, millest sõltuti, suurema prioriteediga. Näiteks DELETE lause USING klausli kasutusloo (ID 1009) saab realiseerida alles peale DELETE lause kasutusloo (ID 1006) realiseerimist.

Tabel 2. Tarkvara kasutuslood koos olulisuse ja pingutuse hinnangutega.

ID	Kasutuslugu	Olulisus	Pingutuse hinnang
1002	Lihtsam ja loogilisem eemaldamine	1	3
1001	Tabelite valimine otse valikukastist	2	8
1006	DELETE lause	3	21
1005	UPDATE lause	4	21
1003	INSERT lause	5	21
1004	INSERT lause alampäringuga	5	8
1011	RETURNING klausel	6	8
1009	DELETE USING klausel	7	8
1010	UPDATE FROM klausel	8	8
1007	OVER ja PARTITION BY	9	13
1008	CTE	10	55

5.2.2 Töökiiruse määramine

Projekti alguses on töökiiruse e *velocity* määramine puhtalt meeskonna tunnetuslik hinnang selle kohta kui palju töid iga iteratsiooniga valmis saadakse. Selle projekti puhul põhineb töökiiruse hinnang ainult autori hinnangul. Esimene hinnang on suure tõenäosusega vale ning seda võib järgmistes iteratsioonides korrigeerida, olles saanud parema tunnetuse tööde tegelikest mahtudest. Autor määras algseks töökiiruseks 21. See tähendab, et ühes iteratsioonis arendatakse 21 pingutuse punkti võrra töid. Töökiirus määrati autori esialgse tunnetuse järgi, mille sisendiks oli eelnev rakenduse analüüs.

5.2.3 Esialgse väljalaske plaani loomine

Peale töökiiruse määramist sai kokku panna esialgse väljalaske plaani (Tabel 3). Erinevad iteratsioonid on tähistatud erinevate värvidega. Tööd jaotati iteratsioonide vahele ära vastavalt töökiirusele. Tööde listi tipust allapoole liikudes liidetakse tööde pingutuse hinnangud kuni jõutakse töökiiruseni või täpselt enne sellest üle minemist. Kõik liidetud tööd pannakse ühte iteratsiooni. Tegevust korratakse tabeli lõpuni. Esimeses iteratsioonis võeti tegemata tööde nimekirjast e *backlog*ist käimasolevasse iteratsiooni uusi töid, et suur osa iteratsioonist raisku ei läheks. Kuid edaspidi osutus arendamine oodatust keerukamaks, töökiirus osutus eeldatust väiksemaks ning seetõttu lisandus võrreldes algse plaaniga uusi iteratsioone ning kasutuslood 1007 ja 1008 jäidki lõpuks ootama järgmist arendustsüklit.

Tabel 3. Esialgne väljalaske plaan.

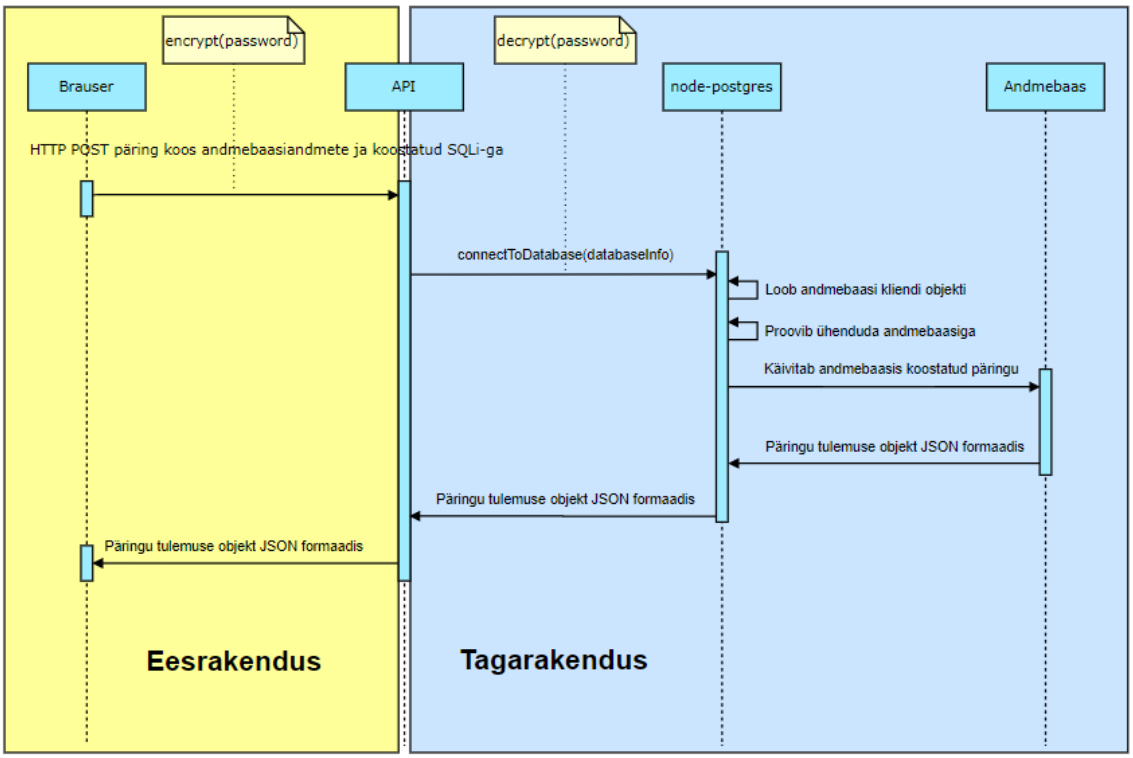
ID	Kasutuslugu	Olulisus	Pingutuse hinnang	Iteratsiooni lõpp
1002	Lihtsam ja loogilisem eemaldamine	1	3	
1001	Tabelite valimine otse valikukastist	2	8	1.03.2023
1006	DELETE lause	3	21	15.03.2023
1005	UPDATE lause	4	21	29.03.2023
1003	INSERT lause	5	21	14.04.2023
1004	INSERT lause alampäringuga	5	8	
1011	RETURNING klausel	6	8	28.04.2023
1009	DELETE USING klausel	7	8	
1010	UPDATE FROM klausel	8	8	15.11.2023
1007	OVER ja PARTITION BY	9	13	
1008	CTE	10	55	

5.3 Tarkvara sisemine ülesehitus

Selles jaotises kirjutatakse sellest, kuidas tarkvara sisemiselt toimib ja on ülesehitatud.

5.3.1 Tagarakenduse arhitektuur

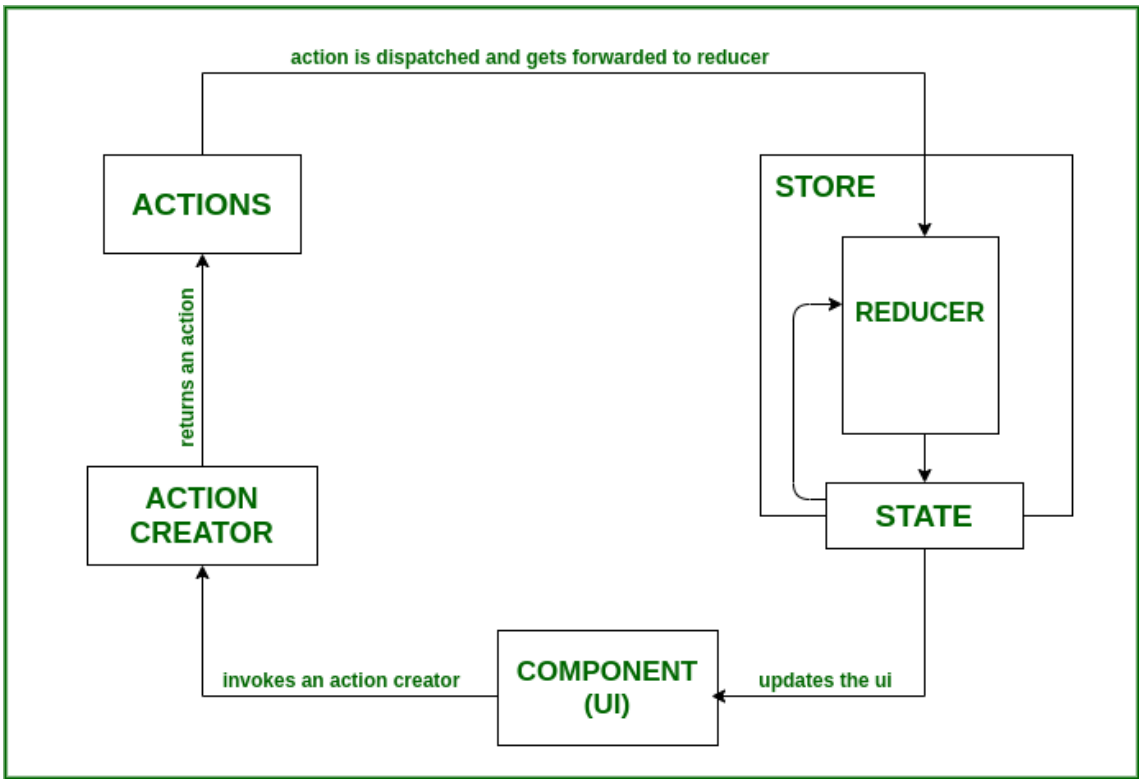
Tagarakendus suhtleb andmebaasiga node-postgres [19] teegi abil. Peale eesrakendusest tulnud päringut loob node-postgres uue andmebaasi kliendi objekti ja proovib ühenduda andmebaasiga. Eduka ühenduse puhul käivitatakse andmebaasis eesrakendusest tulnud SQL lause ja tagastatakse tulemuse objekt JSON formaadis. Obejkt saadetakse tagasi eesrakendusse päringu vastuseks. Samuti toimub tagarakenduses parooli dekrüpteerimine, mis krüpteeritakse eesrakenduses andmebaasi logimisel. Eesrakendus suhtleb tagarakendusega läbi axios HTTP klienti [22]. Tagarakenduse ja eesrakenduse suhtlust kirjeldab joonisel 7 esitatud jadaskeem.



Joonis 7. *Tagarakenduse objektide suhtluse stsenaarium jadaskeemina. [7]*

5.3.2 Eesrakenduse arhitektuur

Eesrakenduses on kasutatud React Reduxi [21] arhitektuuri, mis koosneb neljast põhiosast (vt Joonis 8). Ladu e *store*, kus hoitakse tervet rakenduse olekut. Tegevuste loojad e *action creators* on funktsioonid, mis edastavad tegevusi e *actions*. Tegevuste loojaid kutsutakse välja erinevate kasutaja interaktsioonide kaudu kasutajaliideses. Tegevused kirjeldavad sündmusi, mis rakenduses juhtuvad. Viimaseks põhiosaks on ülekandjad e *reducers*, mis uuendavad rakenduse olekut vastavalt saadetud tegevusele. Rakenduse komponendid ühenduvad lao külge kasutades neile vajalikke olekuid ja tegevuste funktsioone. Tulemusena on rakendusel üks globaalne ladu, mille muutmisel järgitakse väga spetsiifilisi mustreid. Lao olekut muutvad funktsioonid koondatakse ühte kohta kokku, mis muudab rakenduse koodi lihtsasti arusaadavaks ja hallatavaks.



Joonis 8. *React Redux rakenduse tsükkel [47].*

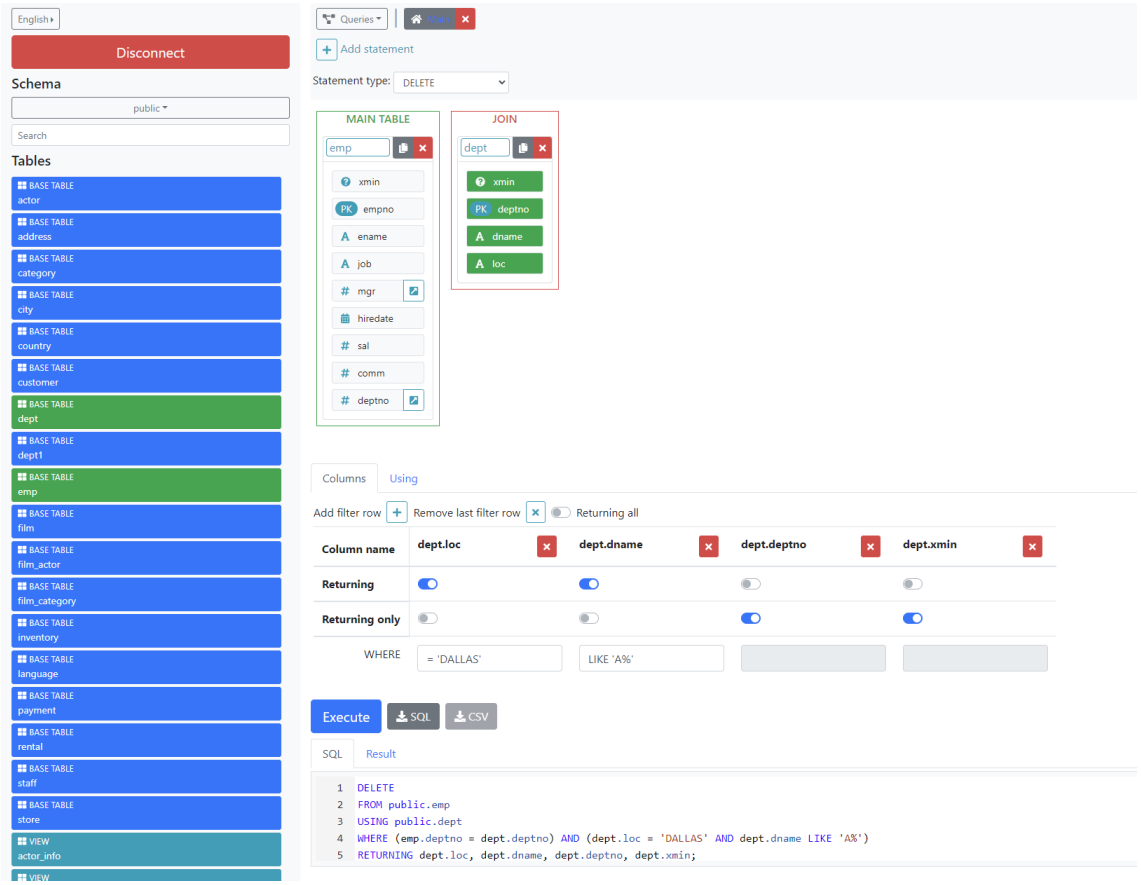
5.4 Uued funktsionaalsused

Selles jaotises kirjutatakse käesolevas töös realiseeritud uutest funktsionaalsustest. Kasutajaliidese koostamisel lähtuti (olulisuse järgi kahanevalt) töö autori enda eelistustest, võrreldavates programmides realiseeritud sarnastest funktsionaalsustest, teadusartiklites väljatoodust ning juhendaja soovitustest. Tuleb arvestada, et RETURNING klausli kasutamine ning UPDATE või DELETE lause puhul teiste tabelite kaasamine tingimuste kontrolli FROM või USING klausli abil on PostgreSQL-spetsiifilised keele võimalused ja seega pole ka vastava kasutajaliidese kavandamisel kuskilt eeskuju võtta.

5.4.1 DELETE laused

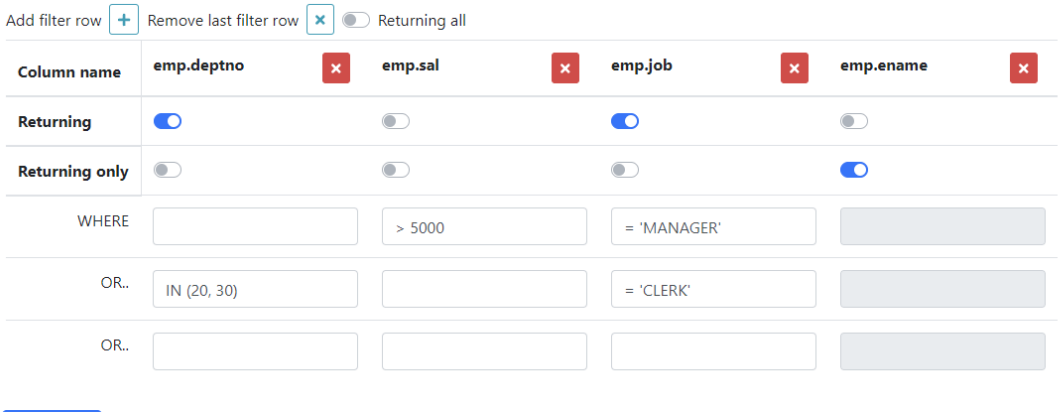
Kuna DELETE lause oli esimene realiseeritud uus lausetüüp, siis sellega koos lisati kasutajaliidesesse ka rippmenüü, kust kasutaja saab soovitud lausetüübi valida. Rippmenüü paigutati lause koostamise vaate esimesse ossa kõige alla. Nii on kasutajale selgelt näha, mis tüüpi lauset hetkel koostatakse. Vaikimisi lausetüübiks on rippmenüüs "SELECT". DELETE lause koostamiseks tuleb lausetüübi rippmenüüst valida "DELETE". Seejärel saab kasutaja valida tabeli kust andmeid kustutatakse ja veerud, millele soovitakse WHERE klausli abil kitsendusi lisada. Kõikide lisatud veergude kohta on näha veeru nimi, RE-

TURNING lüliti, RETURNING *only* lüliti ja kustutamise nupp. DELETE lause koostamise kasutajaliidest illustreerib joonis 9.

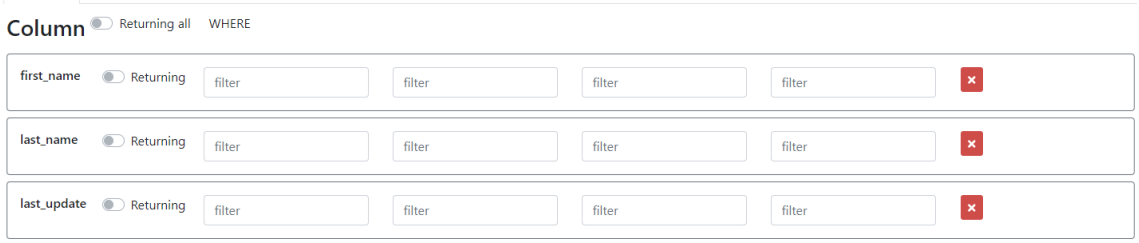


Joonis 9. *Postgres Visual Query Builder DELETE lause loomise vaade.*

WHERE klausli filtri e otsingutingimuse koostamise liidese kavandamisel võeti eeskuju QBE'st. Veergude kitsendused moodustavad rea, mis liidetakse kokku AND operatsiooniga ning kõik read siis omakorda liidetakse kokku OR operatsiooniga (vt Joonis 10). Nii on võimalik kasutajal koostada lihtsamaid lihttingimusi kui ka keerulisemaid liittingimusi. Sarnast lähenemist kasutatakse ka MS Accessis (vt Lisa 2). Algselt prooviti samal loogikal põhinevat lahendust selliselt, et AND'iga liideti kokku kõik veeruga seotud kitsendused ja OR'iga liideti kokku kõik veerud, kuid see variant oli autori ja juhendaja hinnangul kasutajale visuaalselt ebaselgem (vt Joonis 11). Samuti sobitub realiseeritud versioon paremini rakenduse ekraanipiirkondade vaatesse. Kuna ruumi on horisontaalselt rohkem, siis on kõik veerupõhised lülitid kasutajale kompaktselt näha ja ka erinevaid filtri ridasid mahub ekraanile korraga rohkem.



Joonis 10. *Postgres Visual Query Builder DELETE lause WHERE filtri loomise vaade.*



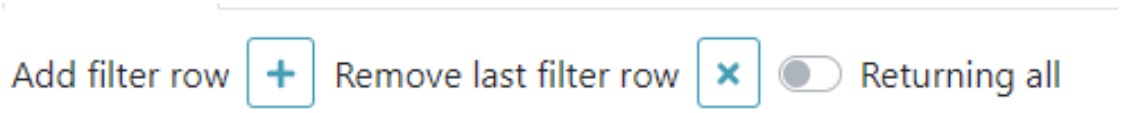
Joonis 11. *Postgres Visual Query Builder DELETE lause WHERE filtri loomise vaate esimese versiooni prototüüp.*

Filtri väljale tuleb sisestada tingimuses kasutatav väärtus koos operaatoriga. Näiteks kui soovitakse lisada tingimus, kus väljas olev väärtus peab olema võrdne arvuga 3, siis peab filtrisse sisestama "= 3". Rakendus lisab koostatud SQL lauses tingimusse veeru nime automaatselt. Operaatori ise sisestamisega on kasutajal filtri loomisel laiemad võimalused võrreldes sellega, kui operaator kasutajaliideses väljapakutud operaatorite hulgast valida. Juhul kui väli jäetakse tühjaks, siis ei lisata vastavat tingimust ka lausesse. Sellisel viisil veerule filtri lisamine on identne QBE's kirjeldatud lahendustega. [34]

RETURNING lüliti abil saab kasutaja lisada selle veeru koostatava lause RETURNING klauslisse. RETURNING klausel võimaldab andmemuudatuse järel tagastada andmeid muudetud ridadest. Kustutamise korral võimaldab see tagastada andmeid kustutatud ridade kohta. RETURNING ALL lüliti lisab lausesse RETURNING klausli kõikidest veergudest andmete tagastamisega (Joonis 12). Juhul kui see lüliti on aktiivne, siis muutuvad kõik veerupõhised RETURNING lülitid mitteaktiivseks, säilitades oma väärtused. See tähendab, et RETURNING ALL väljalülitamisel taastub kasutaja poolt eelnevalt valitud RETURNING lülitite kombinatsioon.

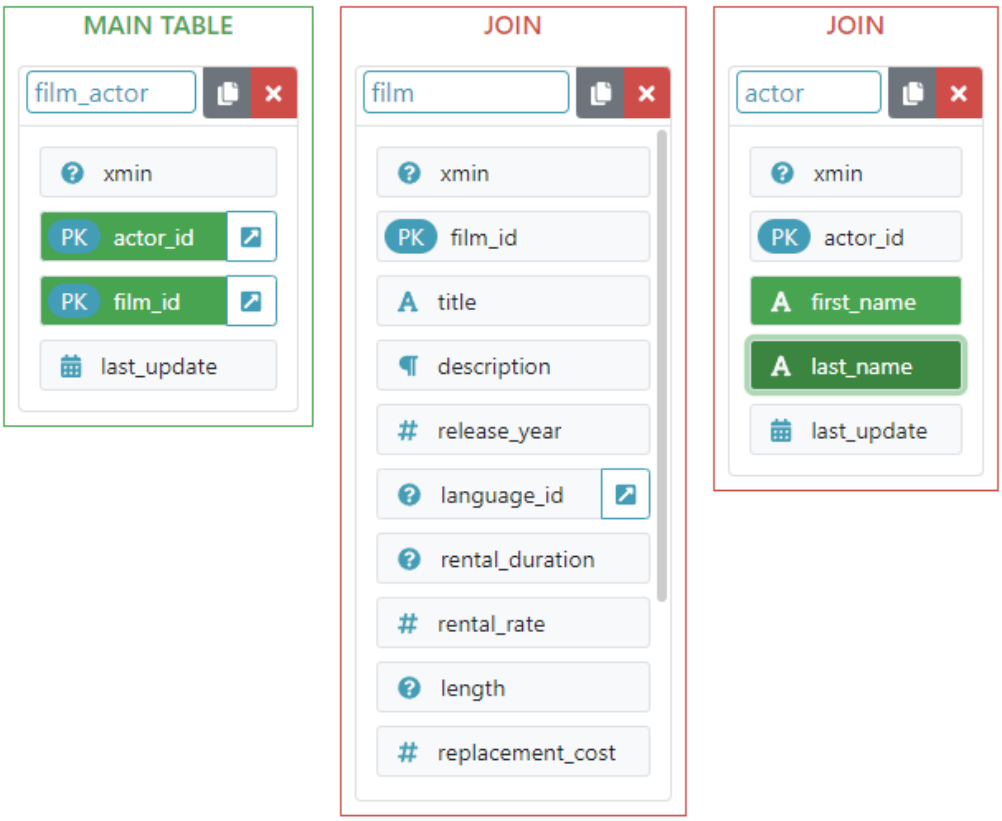
RETURNING *only* lülitiga saab kasutaja lisada veeru ainult lause RETURNING klauslisse

(st seda veergu ei kasutata tingimuses, kuid lause tagastab andmeid sellest veerust). Kõik veeruga seoses kirjeldatud kitsendused eemaldatakse lausest. Lüliti aktiveerimisel muutuvad konkreetse veeru väljad nii visuaalselt kui ka funktsionaalselt mitteaktiivseks, et kasutajal oleks lihtsam aru saada, millistele veergudele rakendatud tingimustest otsingutingimuse kitsendus moodustub.

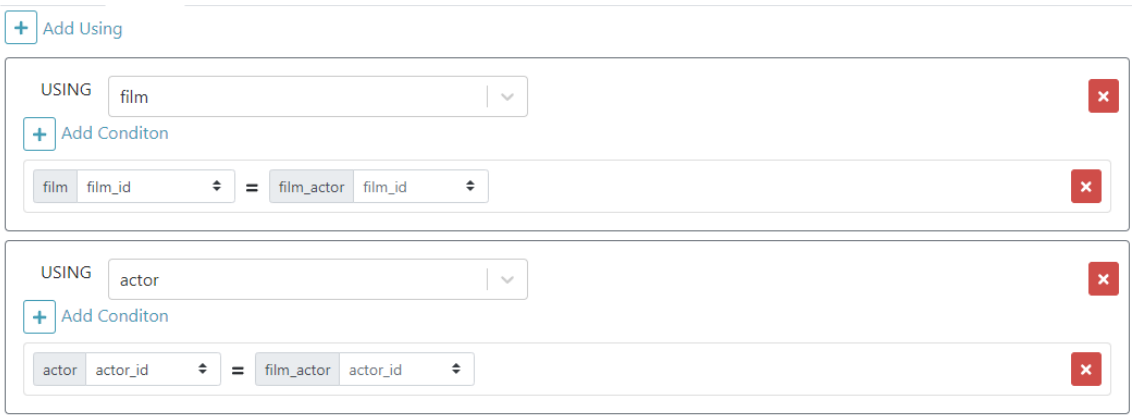


Joonis 12. *DELETE lauses tingimuste ridade lisamise ja eemaldamise nupud, RETURNING lüliti.*

PostgreSQLis on võimalik DELETE lauses tabeleid ühendada USING klausli abil, et kontrollida tingimuste täidetust teiste tabelite põhjal. Selle kasutamise võimalus realiseeriti ka käesolevas rakenduses. USING klausli kasutamiseks peab kasutaja valima päringu koostamise alumises osas "Join" vahekaardi. Ühtse kasutajakogemuse tagamiseks on tabelite ühendamise kasutajaliides sarnane SELECT lausete omaga. Kasutaja saab valida soovitud tabeli ning siis seada paika tabelite ühendamise tingimused (vt Joonis 13). Lisatud tingimused lisatakse filtrile AND operatsiooniga. Kasutaja saab filtrisse lisada tingimusi ka ühendatud tabelite veergudele. Põhitabel, kust andmeid kustutatakse, on märgitud rohelise kasti ja tekstilise viitega, kõik ühendatud tabelid aga punase kasti ja tekstilise viitega. Nii eristab kasutaja, millisest tabelist täpselt ridu kustutatakse ning millised on lausesse kaasatud ainult abistamiseks (vt Joonis 14).



Joonis 13. *DELETE lauses kasutatud põhitabel ja ühendatud tabelid.*



Joonis 14. *DELETE lause USING klausli koostamise vaade.*

Põhitabel on tabel, mis valitakse lausesse esimesena. See tähendab, et kui see lausest eemaldatakse siis nullitakse terve lause. Selline lahendus valiti, sest põhitabeli vahetamisel peaks suure osa juba koostatud lausest kustutama - kustutada tuleks eelnevalt koostatud USING klausli ühendamiste tingimused ja ühendatud tabelite veergude filtrid. Nii võib kasutajal tekkida segadus, mis osa lausest täpselt kustutati ja mis on vaja uuesti koostada. See võib omakorda viia lauseteni, kus lausesse jääb tingimus, mida kasutaja tegelikult ei

soovinud ning tabelist kustutatakse valed andmed.

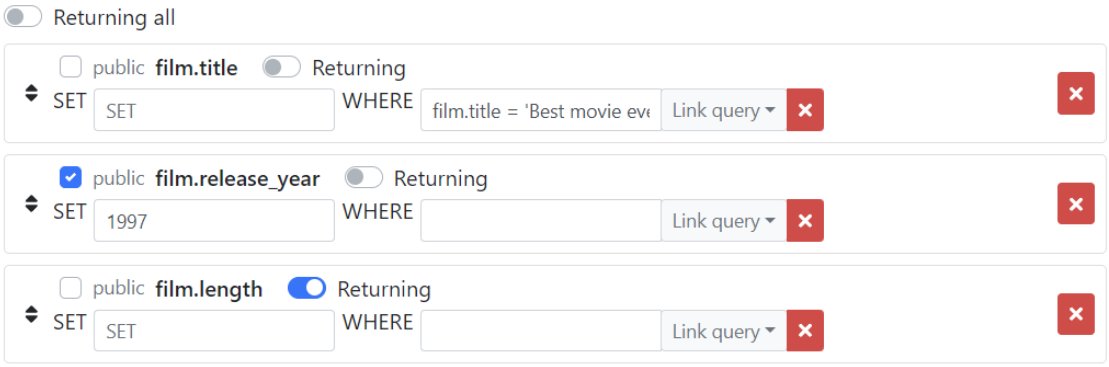
5.4.2 UPDATE laused

UPDATE lause koostamiseks tuleb valida lausetüübi rippmenüüst "UPDATE". Järgmisena valitakse tabel, kus soovitakse andmeid uuendada. Sihttabel märgitakse ära rohelise kastiga ja tekstilise viitega, et kasutajal oleks lihtne aru saada, millises tabelis andmeid muudetakse.

Kui kasutaja on sihttabelist valinud uuendatavad veerud, siis saab ära kirjeldada nendele vastavatesse väljadesse lisatavad uuendatud väärtused. Tühi väli kasutajaliideses asendatakse automaatselt NULLiga. Selline lahendus on välja pakutud ka QBE artiklis. [34] Kuna valitud väljad lisatakse automaatselt ka WHERE filtri otsingutingimuse koostamise vaatesse, siis on iga veeru juures ka lüliti, millega saab veeru väärtuse uuendamise lausest välja võtta. Nii on võimalik kasutada veergu WHERE filtris ilma veeru väärtusi uuendamata (vt Joonis 15).

Joonis 15. *UPDATE lause uuendatud väärtuste lisamise vaade.*

Algselt kasutati UPDATE lause puhul SELECT lausetes realiseeritud veergude lisamisele sarnast disaini, kus sai veerule korraga määrata nii uue väärtuse kui ka WHERE filtri kitsenduse (vt Joonis 16). Hiljem aga tehti nende jaoks kasutajaliidesesse eraldi vaated. Selle tulemusena saab kasutaja koostada WHERE filtris keerukamaid liitotsingutingimusi. Lisaks paranes tugevalt kasutusmugavus, kuna üks väga keerukas mitme funktsiooniga visuaalne komponent tükeldati väiksemateks ja konkreetsema funktsiooniga komponentideks.

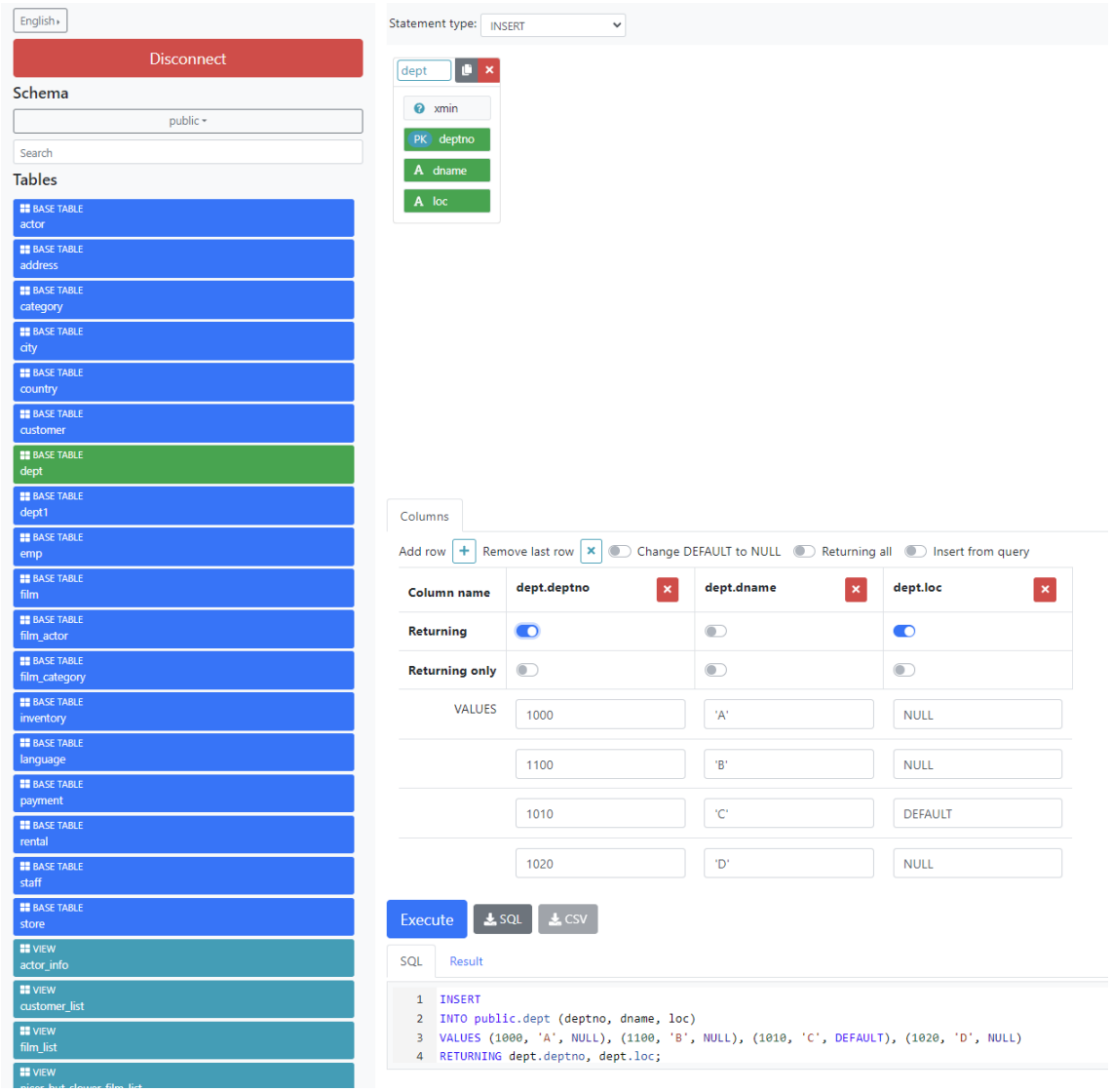


Joonis 16. *UPDATE lause uuendatud väärtuste lisamise vaate prototüüp.*

Ka UPDATE lauses on võimalik PostgreSQLis tabeleid ühendada, et kontrollida tingimuste täidetust teiste tabelite põhjal. Seda saab teha FROM klausli abil. Otsingutingimuste filtri ja tabelite ühendamiste koostamise vaates on kasutatud samasuguseid komponente nagu DELETE lauses ehk need on disainilt ja funktsionaalsuselt identsed. Samuti on sarnaselt DELETE lausele põhitabel esimene tabel, mis lausesse valitakse ja selle kustutamine nullib samadel põhjustel terve lause.

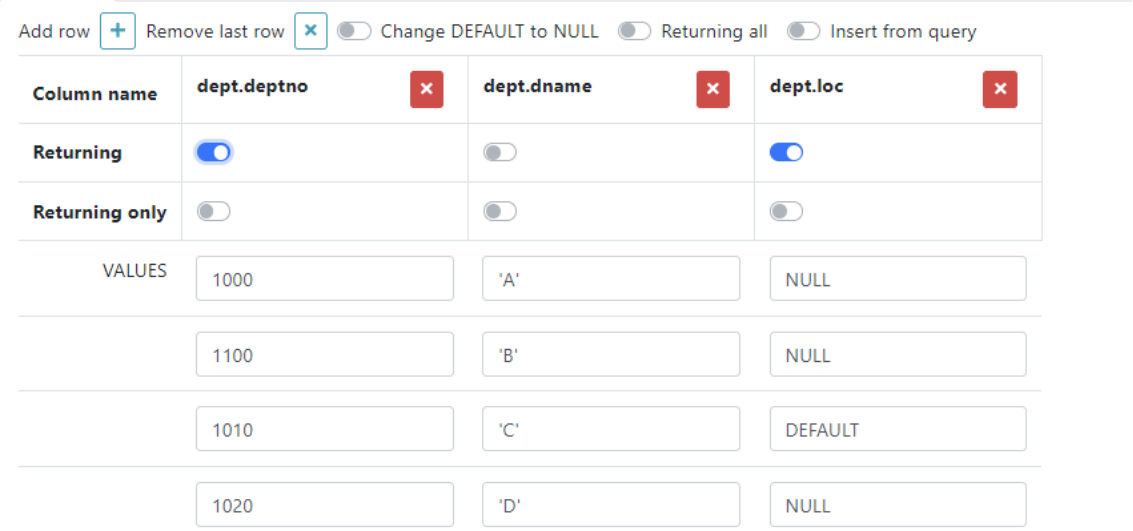
5.4.3 INSERT laused

INSERT lause koostamiseks tuleb valida lausetüübi rippmenüüst "INSERT". Järgmisena saab valida tabeli, kuhu soovitakse uusi ridu lisada ja veerud, mille jaoks antakse väärtused. Kasutaja saab andmeid lisada ühe või rohkema rea kaupa (VALUES klausel) (vt Joonis 17). Samuti on kasutajal võimalik lisada andmeid alampäringu abil.



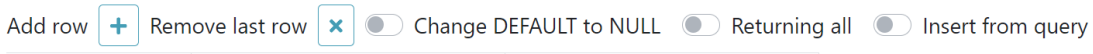
Joonis 17. *Postgres Visual Query Builder INSERT lause loomise vaade.*

Iga veeru kohta on kasutajaliideses kirjas veeru nimi, RETURNING ning RETURNING *only* lülitid ja vähemalt üks väli kuhu saab sisestada tabelisse lisatava väärtuse. Vaikimisi on selles väljas väärtuseks DEFAULT, millega sisestatakse veerule vastavasse välja veerule deklareeritud vaikeväärtus või kui see puudub, siis lisatakse NULL (vt Joonis 18). Ka tühi väli asendatakse automaatselt NULL'iga. Tühja välja NULL'iga asendamisel on eeskuju võetud QBE artiklis kirjeldatud INSERT lausete esitusest. [34] Lisatud väärtus pannakse lausesse täpselt nii nagu kasutaja on selle kirja pannud. See tähendab, et näiteks sõne lisamisel tuleb küll kasutada apostroofe e ülakomasid, aga see lubab lihtsasti kasutada väärtuse lisamisel ka funktsioonide väljakutseid või NULL'i. Kustutamise nupu abil saab kasutaja veeru ja selle väärtused lausest eemaldada.



Joonis 18. *Postgres Visual Query Builder INSERT lause ridade väärtustest koostamise vaade.*

Veergude alajaotuse ülaosas on kaks nuppu vastavalt väärtuste sisestamise ridade lisamiseks ja eemaldamiseks, RETURNING ALL lüliti, vaikeväärtuse vahetuse lüliti ja alampäringu kasutamise lüliti (vt Joonis 19).



Joonis 19. *INSERT lauses väärtustega kasutajaliidese ridade lisamise ja eemaldamise nupud, RETURNING ALL, vaikeväärtuse ja alampäringu kasutamise lüliti.*

Uute ridade lisamisel puudub piirang ridade arvu ülapiirile, aga neid peab olema alati vähemalt üks. Kui kasutajaliideses on üks rida, siis ei tee kasutajaliidesest rea eemaldamise nupp midagi. RETURNING ja RETURNING *only* lülitid töötavad sama loogika järgi nagu DELETE lauses. Vaikeväärtuse lülitiga saab muuta algväärtust, mis uue veeru või uue väärtuste rea lisamisel väljadele määratakse. Vaikeväärtus saab olla kas DEFAULT või NULL (viimane pole rangelt võttes väärtus, vaid väärtuse puuduse tähis). Alampäringu kasutamise lüliti aktiveerimisel ilmub nähtavaks rippmenüü alampäringu valimiseks. Lisaks sellele muudetakse ridade lisamise ja kustutamise nupud mitteaktiivseks ning peidetakse kasutaja eest väärtuste sisestamise read. Alampäringu rippmenüüst saab kasutaja valida endale soovitud päringu.

Kui kasutaja soovib muuta tabelit, kuhu andmeid sisestatakse, saab ta tabelite menüüst valida uue tabeli.

5.5 Muud täiendused

Selles jaotises kirjutatakse muudest käesoleva töö tulemusena rakenduses tehtud parandustest ja uuendustest.

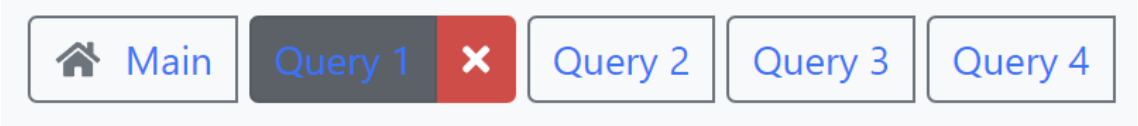
5.5.1 Eesrakenduse täiendused

Selles jaotises kirjutatakse eesrakenduses tehtud täiendavatest muudatustest.

Alampäringute kustutamise loogika

Rakendus võimaldab koostada päringuid, mida saab kasutada teistes lausetes alampäringutena. Päringute kustutamise loogika muudeti kasutajale paremini arusaadavaks. Päringu kustutamiseks mõeldud nupp oli algselt paigutatud päringu käivitamise nupu kõrvale. Nupul oli kasutatud prügikasti ikooni ehk oli aru saada, mis eesmärki nupp täidab, kuid jäi ebaselgeks, mida täpselt kustutada soovitakse. Samuti võis nupu paigutuse tõttu päringu käivitamise asemel hoopis kogemata kustutada.

Kustutamise nupp paigutati ümber päringute navigatsioonireale. Kustutamise nupp ilmub aktiivseks muudetud päringu saki kõrvale. Nupp on näha ainult aktiivse päringu juures. Prügikasti ikoon asendati "X" ikooniga (vt Joonis 20).



Joonis 20. *Päringu kustutamise nupp ilmub valitud päringu kõrvale.*

Uute tabelite valimine tabelite ühendamisel

Eelnevalt pidi tabelite ühendamisoperatsiooniks kasutatav tabel olema juba lausesse valitud. Ühendamisoperatsiooni tabeli valiku rippmenüüd täiendati, et kasutaja saaks valida uue tabeli ka rippmenüüst. Uus tabel lisatakse automaatselt ka lausesse.

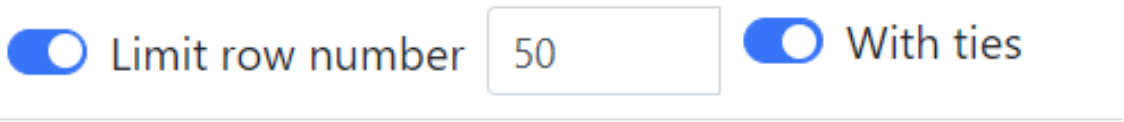
LIMIT klausli asendamine FETCH FIRST klausliga ja WITH TIES klausli tugi

SELECT lausete koostamisel oli eelnevalt realiseeritud LIMIT klausli lisamise funktsionaalsus, mis lubas kasutajal piirata SELECT lauses tagastatud ridade arvu. LIMIT klausli näol on tegemist mittestandardse SQL süntaksiga.

Tellija (juhendaja) soovil asendati koostatud SQL koodis LIMIT klausel FETCH FIRST

klausliga (vt Joonis 21).[48] Klauslid erinevad funktsionaalsuse mõttes ainult süntaksi poolest. Erinevalt LIMIT klauslist on FETCH FIRST klausel kasutusel SQL standardis ja tuli kasutusele SQL:2008 versioonis. [49]

Juurde lisati ka WITH TIES määrangu lisamise funktsionaalsus. [50] WITH TIES määrang on kasutatav ainult üheskoos ORDER BY klausliga (sorteerimisega) ning tagab, et kui väljastada tuleb n esimest rida ning nende hulgas on sorteerimise aluseks olevad väärtused mitmes reas ühesugused, siis on kõikide selliste väärtustega read tulemuses - isegi kui selle tulemusel on väljastatud ridade arv suurem kui n. Funktsionaalsus on realiseeritud täiendava lülitiga, mis muutub nähtavaks ainult siis, kui kasutaja on aktiveerinud FETCH FIRST lüliti (vt Joonis 21) ja lauses on määratud sorteerimine. Kui sorteerimine lausest eemaldada, siis deaktiveeritakse ka WITH TIES valik.



Joonis 21. *Ridade arvu piiramise lüliti.*

Veaparandused

Töö tulemusena parandati päringute kustutamise protsessis ilmnenud kriitiline viga. Nimelt kui lauses kasutati alampäringut ning see kustutati, siis lõpetas rakendus funktsioneerimise. Töötavat seisu ei olnud seepeale võimalik taastada ning kasutusseanss tuli hüljata. Parandusena lisati kontroll alampäringu sõne muutujale. Juhul kui muutuja on tühja väärtusega e *undefined* siis asendatakse see tühja sõnega. Paranduse tulemusena eemaldatakse alampäringu kustutamisel see põhipäringust ning alampäringu valik muutub vaikimisi väärtuseks ehk tühjaks sõneks.

5.5.2 Node.js versiooniuuendus

Enne arenduse alustamist kohandati rakendust nii, et see töötaks töö alguse seisuga kõige viimase Node.js versiooniga. Sellel hetkel (2023. aastal) oli kestustoe e LTS versiooniks v18.12.0. Rakendus oli eelnevalt arendatud versiooni v8.4.0 peale.

Tagarakenduse muudatused

Tagarakenduses uuendati järgmised teegid ja raamistikud :

- Express raamistik (ver. 4.16.2 → 4.18.2),

- Express Promise Router raamistik (ver. 3.0.3 → 4.1.1),
- Helmet raamistik (ver. 3.22.0 → 6.0.1),
- Node Postgres raamistik (ver. 7.8.2 → 8.2.0),
- Nodemon tööriist (ver. 2.0.2 → 2.0.20).

Samuti tehti muudatused metaandmete failis *package.json*, et lubatud oleks ES6 (*ECMAScript 6*) [51] moodulite impordi ja ekspordi süntaks. Sellele vastavalt kohandati ka moodulite importe kuues failis.

Eesrakenduse muudatused

Eesrakenduses uuendati või võeti kasutusele järgmised teegid:

- React Scripts (ver. 3.4.1 → 5.0.0),
- Buffer (ver. 6.0.3),
- Crypto Browserify (ver. 3.12.0),
- Stream Browserify (ver. 3.0.0),
- React App Rewired (ver. 2.2.1).

Node.js 18 jaoks on vaja kasutada vähemalt React Scripts versiooni v5.0.0. Selles versioonis võetakse kasutusele Webpack 5 [52]. Webpack < 5 versioonides oli kaasas mitmete *node.js core module*'ite jaoks loodud polyfill'id, mis on versioonist 5 eemaldatud. [53] Veebiarenduse vaatest on polyfill kooditükk, mis realiseerib mingi rakendusliidese funktionaalsuse, mida veebilehitseja ei toeta. Mitmeid neid mooduleid on kasutatud eesrakenduses andmebaasi sisselogimisel salasõnade räsimiseks. Webpacki ametlikus dokumentatsioonis pakutud lahenduse jaoks oleks vaja teha rakendusele *eject*, mis on CRA'ga (*Create React App*) loodud rakendustele mõeldud skript, millega muutub rakendus täielikult konfigureeritavaks ja kontrollitavaks. *Eject*'imine on aga ühekordne tegevus, mis ei ole tagasikeeratav ning muudab rakenduse haldamist tunduvalt keerulisemaks. [54] Et vältida *eject* skripti kasutamist, võeti alternatiivse lahendusena kasutusele teek *React App Rewired*. Selle teegi abil on võimalik muuta Webpack konfiguratisoone ilma *eject* skriptita. Puudu olevad polyfill'id lisatakse *config-overrides.js* failis.

Node.js uuendamise käigus läks katki ka eelnevalt konfigureeritud linter (ESLint). Puudu olid erinevad sõltuvused (*peer dependencies*), mis takistasid Airbnb poolt loodud stiilijuhendi kasutamist. [55] Kuna uuendamise käigus hüpati üle mitme põhiversiooni (*major version*), siis on raske öelda, mis täpselt põhjustas linteri probleeme. Kõik leitud lahendused viitasid aga sellele, et tuleb installeerida vajalikud sõltuvused. Muudatused tehti järgmistes sõltuvustes:

- ESLint Plugin JSX a11y (ver. 6.2.3 → 6.5.1),
- ESLint Plugin Import (ver. 2.20.1 → 2.25.3),
- ESLint Plugin React (ver. 7.19.0 → 7.28.0).

Juurde lisati sõltuvused:

- Prop Types (ver. 15.8.1),
- ESLint (ver. 8.2.0),
- ESLint Config Airbnb (ver. 19.0.4),
- ESLint Plugin React Hooks (ver. 4.3.0).

Uutes ja uuendatud linteri moodulite versioonides lisatud reeglite tõttu tekkis peale uuendamist ka mitmeid uusi stiilivigu. Enamus neist olid parandatavad automaatselt, kuid tuli konfigureerida ka mõned uued reeglid ja parandada koodi käsitsi nagu näiteks ringsõltuvuste eemaldamine. Kõige ajamahukam stiilivigade parandamise juures oli erinevate reeglitega tutvumine ja analüüsimine, kas nende kasutamine on mõistlik või mitte. Kokku tehti muudatusi 23 failis ja juurde lisati neli reeglit.

5.6 Testimine

Rakenduse testimiseks ja koodi kvaliteedi tagamiseks loodi ühikteste. Testide kirjutamiseks kasutati JavaScript raamistikku Jest [56]. Eesrakenduses on testimiseks kasutusel teegid moxios [57] ja redux-mock-store [58]. Moxiosega saab luua axiose libaobjekte, et testida axiose teel tagarakendustest saadud vastuste korrektset kasutamist. Redux-mock-store võimaldab luua Redux lao libaobjekti, et kontrollida Redux tegevuste käivitamisel korrektset lao oleku muutusi.

Töö jooksul koostati 30 ühiktesti. Olemasolevate komponentide muudatustest tulenevalt parandati 13 olemasolevat ühiktesti ja kolm *snapshot*i.

5.7 Tulemuste avaldamine

Töö tulemused on avatud lähtekoodiga avaldatud GitHubi koodihoidlas: `https://github.com/v4hukomm/postgresql-visual-query-v3`. Rakenduse lähtekood on kaitstud MIT litsentsiga. Samasugune väheste piirangutega litsents oli ka tarkvara kahel esimesel versioonil. Ka näiteks Node.js on MIT litsentsiga [59].

6. Valideerimine

Selles peatükis kirjeldatakse loodud töö tulemuste valideerimist.

Kõigepealt valideeris rakendust autor ise terve arendusprotsessi vältel. Selleks proovis ta luua uut tüüpi toetatud SQL lauseid (INSERT, UPDATE ja DELETE). Sellele lisaks kontrolliti kasutajaliidese toimimist ja lausete koostamist koos tellijaga (juhendajaga) nende regulaarsetel kohtumistel iteratsioonide vältel ja lõpus. Peale arendusprotsessi lõppu koostas autor tagasiside küsimustiku Google Forms veebirakenduse abil. [16] Küsimustik koosnes üheksast lause koostamise ülesandest. Samuti küsiti üldisemalt rakenduste positiivsete ja negatiivsete külgede ning kasutusmugavuse kohta. Testkasutajate ülesannete lahendamiseks mõeldud laused on nähtavad Lisa 3. Ülesanded on nähtavad Lisa 4. Küsimustik on nähtav Lisa 5. Küsimustiku koostamisel võeti eeskuju eelmiste autorite koostatud küsimustest [6], [7].

6.1 Testkasutajatega testimine

Testkasutajatega testimine viidi läbi ajavahemikul 07.01.2024-08.01.2024. Testkasutajaid oli kolm, kõik neist erineva SQLi kasutamise kogemusega. Üks kasutajatest hindas oma oskusi SQLis heaks, üks kesmiseks ja viimane minimaalseks. PostgreSQLi-spetsiifilisi oskuski hindas üks kasutaja heaks ja kaks kasutajat minimaalseks. Eelnevat rakendust kasutajatele ei tutvustatud. Ainsa eeltööna tutvustati, millised tabelid on andmebaasis olemas. Kasutajatele anti ka SQL-lausete kujul vastuste nimekiri, mille vastu said kasutajad peale igat ülesannet oma tulemust kontrollida. Lausete korrektsust kontrollis ka autor. Kasutajad läbisid testimise iseseisvalt, kuid võisid küsida abi juhul kui nad olid pikemalt mõne probleemiga hätta jäänud. Seda võimalust paar korda ka kasutati. Üks kord, et saada aru, kuidas käib alampäringu koostamine ja teine kord, et saada aru, kuidas käib SELECT lauses WHERE kitsenduse e filtri koostamine.

6.1.1 Ülesannete lahendamine

Järgmisena tuuakse välja testkasutajate ülesannete tulemused ja küsimustiku vastused.

INSERT lause 1

Sisestage uus osakond, kus osakonna number on 1010, nimeks 'Osakond A' ja asukohaks on NULL.

```
1  INSERT
2  INTO public.osakond (osakond_nr, osakond_nimi, asukoht)
3  VALUES (1010, 'Osakond A', NULL)
4  ;
```

Joonis 22. *Esimene ülesanne.*

Esimese ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 22). Ühel kasutajal esinesid väiksemad takistused (vt Joonis 23). Kasutaja ei saanud kohe aru, millisel kujul väärtus lausesse pannakse. Tal tuli natuke katsetada, enne kui ta sai aru, et sõne lisamiseks tuleb väärtus ise apostroofide vahele panna.

Joonis 23. *Esimese ülesande tagasiside.*

INSERT lause 2

Sisestage kaks uut osakonda. Esimese number on 1000 ja nimeks 'Osakond B'. Teise number on 1100 ja nimeks 'Osakond C'. Tagasta tulemuses osakonna number.

```
1  INSERT
2  INTO public.osakond (osakond_nr, osakond_nimi)
3  VALUES (1000, 'Osakond B'), (1100, 'Osakond C')
4  RETURNING osakond.osakond_nr;
```

Joonis 24. *Teine ülesanne.*

Teise ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 24). Ühel kasutajal esinesid väiksemad takistused (vt Joonis 25). Kasutaja põhjendas, et RETURNING klausli lisamine võttis natuke aega.

Joonis 25. *Teise ülesande tagasiside.*

INSERT lause 3

Sisestage kõik osakonnad tabelist **osakond**, mille osakonna number on suurem kui 100, tabelisse **osakond1**. Sisestage ainult osakonna number ja nimi. Tulemuses tagastage kõik veerud.

```
1  INSERT
2  INTO public.osakond1 (osakond_nr, osakond_nimi)
3  SELECT
4  osakond.osakond_nr, osakond.osakond_nimi
5  FROM public.osakond
6  WHERE (osakond.osakond_nr > 100)
7  RETURNING *;
```

Joonis 26. *Kolmas ülesanne.*

Kolmanda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 26). Kahel kasutajal esinesid väiksemad takistused ja ühel kasutajal suured takistused (vt Joonis 27). Kasutajate peamine murekoht oli alampäringu loomise protsess. Ei olnud selgelt arusaadav, kust ja kuidas selle loomine toimub. Lisaks oli kahel kasutajal raskusi SELECT lauses WHERE filtri leidmisega ja selle koostamisega.

Joonis 27. *Kolmanda ülesande tagasiside.*

UPDATE lause 1

Suurendage kõikide töötajate palkasid 100 võrra. Tulemuses tagastage palgad.

```
1  UPDATE
2  public.tootaja
3  SET palk = palk + 100
4  RETURNING tootaja.palk
5  ;
```

Joonis 28. *Neljas ülesanne.*

Neljanda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 28). Takistusi kasutajatel ei esinenud (vt Joonis 29).

Joonis 29. *Neljanda ülesande tagasiside.*

UPDATE lause 2

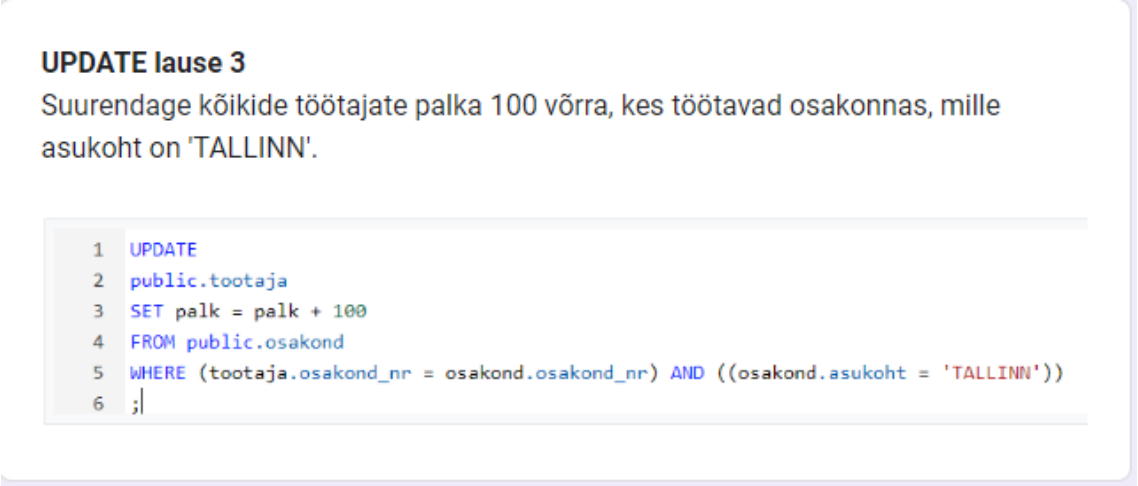
Suurendage kõikide töötajate palka 100 võrra, kelle praegune palk on alla 2000 või kelle töökoht on 'MANAGER' ja boonus NULL.

```
1  UPDATE
2  public.tootaja
3  SET palk = palk + 100
4  WHERE ((tootaja.palk  < 2000) OR (tootaja.tookoht = 'MANAGER' AND tootaja.boonus = NULL))
5  ;
```

Joonis 30. *Viies ülesanne.*

Viienda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 30). Kahel kasutajal esinesid väiksemad takistused (vt Joonis 31). Toodi välja, et kuna SET ja WHERE olid eraldi, siis pidi natukene õppima ja koostamise protsessiga harjuma.

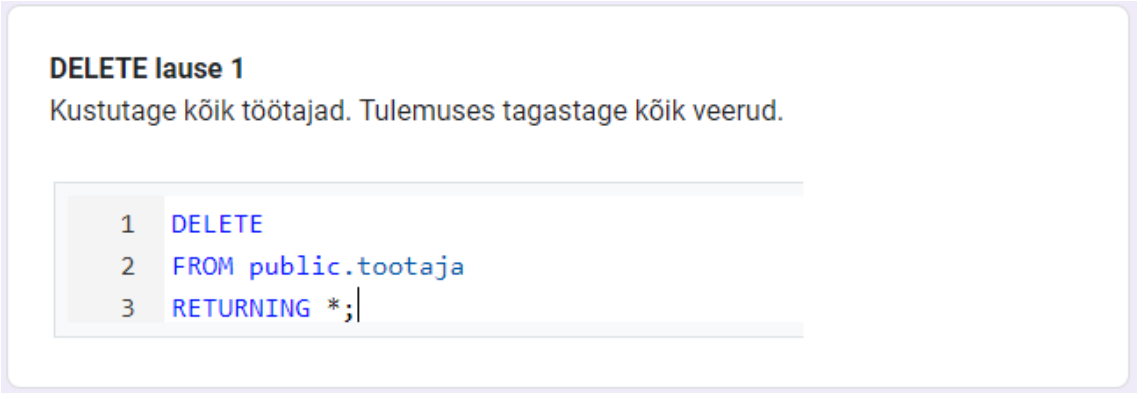
Joonis 31. *Viienda ülesande tagasiside.*



Joonis 32. *Kuues ülesanne.*

Kuuenda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 32). Ühel kasutajal tekkisid väiksemad takistused (vt Joonis 33). Vastuses põhjendati, et läks aega, leidmaks, kuidas ühendatud tabelist veergu otsingutingimusse lisada.

Joonis 33. *Kuuenda ülesande tagasiside.*



Joonis 34. *Seitsmes ülesanne.*

Seitsmenda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 34). Takistusi kasutajatel ei esinenud (vt Joonis 35).

Joonis 35. *Seitsmenda ülesande tagasiside.*

DELETE lause 2

Kustutage kõik töötajad, kelle töötaja number on 7369 või kelle palk on alla 1500. Tulemuses tagastage töötaja palk ja nimi.

```
1  DELETE
2  FROM public.tootaja
3  WHERE ((tootaja.tootaja_nr = 7369) OR (tootaja.palk < 1500))
4  RETURNING tootaja.palk, tootaja.tootaja_nimi;
```

Joonis 36. *Kaheksas ülesanne.*

Kaheksanda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 36). Takistusi kasutajatel ei esinenud (vt Joonis 37). Toodi välja, et iga uue lause koostamisega saab rakenduse käitumine selgemaks ja muutub lihtsamaks.

Joonis 37. *Kaheksanda ülesande tagasiside.*

DELETE lause 3

Kustutage kõik töötajad, kes töötavad osakonnas asukohaga 'TALLINN' või kelle osakonna nimi hakkab 'C' tähega. Tulemuses tagastage kõik veerud.

```
1  DELETE
2  FROM public.tootaja
3  USING public.osakond
4  WHERE(tootaja.osakond_nr = osakond.osakond_nr) AND ((osakond.asukoht = 'TALLINN') OR (osakond.osakond_nimi LIKE 'C%'))
5  RETURNING *;
```

Joonis 38. *Üheksas ülesanne.*

Üheksanda ülesande lahendamisega said hakkama kõik kasutajad (vt Joonis 38). Kahel kasutajal esinesid väiksemad takistused (vt Joonis 39). Üks kasutaja eeldas, et LIKE predikaati saab lisada kuskilt nupust ja ei pea ise väljale kirjutama. Teine kasutaja ütles, et vähese SQL kogemuse tõttu oli tal ununenud LIKE predikaadi süntaks.

Joonis 39. *Üheksanda ülesande tagasiside.*

6.1.2 Üldine tagasiside

Üldise tagasiside vaatest oli kõikide testkasutajate arvates rakendust mugav kasutada. Kasutajatele meeldis rakenduse juures selle intuitiivsus ning märgiti ära, et rakenduse toimimise loogikast saab kiiresti aru. Positiivsena toodi välja ka erinevaid kasutajaliidese külgi nagu värviliste kastidega märgitud põhi- ja abitabelid ja tabelite ülevaade ekraani vasakul küljel. Ühele kasutajale meeldis kasutajaliidese minimaalsus.

Rakenduse juures oli ebameeldiv alampäringute loomine. Kasutajate jaoks oli protsess keeruline ja frustreeriv. Üks kasutaja tõi välja ka SELECT lause koostamise keerukuse.

Kasutajamugavuse parandamiseks pakuti välja tööriistavihjete e *tooltip* kasutamist erinevate elementide juures. Üks kasutaja arvas, et SELECT lause kasutajakogemuse (UX) osas võiks parandusi teha.

Üks kasutaja leidis rakenduse testimisel ka vea. Vea kordamiseks pidi kasutaja päringut muutma samaaegselt, kui aktiivne oli vastuste tabel ("Result" sakk päringu koostamise vaate alumises osas). Käesoleva töö raames ei jõudnud autor viga parandada. Viga kajastatakse arendusvaates.

6.1.3 Nõrkused

Tehtud valideerimise nõrkuseks on kaasatud inimeste väikene arv ning samuti nende tutvus töö autoriga, mis võib mõjutada tagasiside andmist.

6.2 Võrdlus olemasolevate rakendustega

Selles jaotises võrreldakse töö käigus valminud rakendust jaotises 4.3.1 ja 4.3.2 analüüsitud tarkvaraga.

Tabelis 4 on võrreldud valminud uusi funktsionaalsusi kahe olemasoleva tarkvaraga, kus saab koostada andmete muutmise lauseid. Kasutatud tähistus:

- Linnuke – funktsionaalsuse olemasolu,
- "X" – funktsionaalsuse puudumine,
- miinus – vastava SQL konstruktsiooni toe puudumine andmebaasisüsteemis.

Tuleb rõhutada, et kumbki võrreldavatest tarkvaradest ei võimalda koostada SQL lauseid PostgreSQL jaoks. Autor ei leidnud programme, mis võimaldaks koostada graafilisel viisil andmete muutmise lauseid PostgreSQL jaoks. Selles mõttes on tegemist unikaalset funktsionaalsust pakkuva tarkvaraga.

Vaadeldud 11-st funktsionaalsusest, mida toetab PostgreSQL Visual Query Builder (v3) toetab MS Access üheksat ja dbForge Studio For MySQL nelja. Valminud rakendus on paindlikum ja suuremate võimalustega kui võrreldavad tarkvarad.

Tabel 4. PostgreSQL Visual Query builder võrdlus olemasolema tarkvaraga

	PostgreSQL Visual Query Builder (v3)	**MS Access (v2311)**	**dbForge Studio For MySQL (v10)**
DELETE lause koostamine	✓	✓	✓
Liitotsingutingimused WHERE filtris DELETE lauses	✓	✓	✗
Tabelite ühendamisoperatsioonid DELETE lauses	✓	✓	✗
UPDATE lause koostamine	✓	✓	✓
Liitotsingutingimused WHERE filtris UPDATE lauses	✓	✓	✗
Tabelite ühendamisoperatsioonid UPDATE lauses	✓	✓	✗
INSERT lause koostamine	✓	✓	✓
INSERT lauses alampäringu kasutamine	✓	✓	✓
INSERT lauses ühe rea lisamine ilma alampäringuta	✓	✓	✗
INSERT lauses mitme rea lisamine ilma alampäringuta	✓	-	✗
RETURNING klausel (ainult PostgreSQL)	✓	-	-

7. Arendusvaade

Selles peatükis kirjutatakse edasistest ideedest visuaalse SQL lausete koostamise tarkvara arendamiseks.

Autori hinnangul võiks edasiarenduses kõigepealt tegeleda rakenduses olemasoleva funktsionaalsuse parendamise ja täiendamisega.

7.1 Vigade parandused

1. Parandada viga - kui päringu koostamise vaates tulemuste osas on aktiivne "Result" sakk, siis päringu muutmisel "SQL" sakis koodikastis olev kood ei muutu.

7.2 Olemasoleva kasutajaliidese ja funktsionaalsuse täiendamine

1. Kuna andmemuudatuse lausete INSERT, UPDATE ja DELETE tugi realiseeriti SELECT lausete toest eraldi arendustsüklis ja erineva inimese poolt, siis võiks mõelda lausete koostamise ühtlustamisele. Käesoleva töö tagasisides esines mitmel kasutajal ka takistusi SELECT lausete koostamisel. Autori hinnangul tuleneb see SELECT lause visuaalsete veeru komponentide keerukusest, mida võiks proovida järgmises rakenduse arenduse iteratsioonis lihtsustada.
2. SELECT lause koostamisel võiks kasutada sarnast WHERE filtri loomist nagu on käesolevas töös realiseeritud UPDATE ja DELETE lausete koostamisel. Sellisel juhul ei erine sama funktsionaalsus erinevat tüüpi lausetes ning paraneks kasutusmugavus.
3. UPDATE ja DELETE lause WHERE filtri loomises võiks saada kasutada alampäringuid. Hetkel on võimalik alampäring manuaalset filtri väljale kirjutada, kuid seda võiks saada teha läbi alampäringute visuaalse liidese nagu SELECT lause filtri puhul.
4. INSERT, UPDATE ja DELETE lausetes võiks SET, VALUES ja WHERE klausli koostamisel saada kustutada kasutajaliidesest konkreetseid ridu, mitte viimast koostatud rida nagu on praegu võimalik.
5. UPDATE lauses võiks veergude lisamine SET ja WHERE klauslisse olla eraldi. Hetkel lisatakse veerg mõlemasse korraga. See elimineeriks ka väärtuste uuendamisel "Enabled" lüliti vajaduse.
6. UPDATE ja DELETE lause WHERE filtris võiks liitotsingutingimuste loomine olla

veel paindlikum. Näiteks saab hetkel tingimust (x = 3 OR y = 2) AND (z = 1) luua ainult kujul (x = 3 AND z = 1) OR (y = 2 AND z = 1) vastavalt jaotises 5.3.1 kirjeldatud loogikale. Võiks olla võimalus lisada visuaalselt juurde sulgusid või muuta ridade sees ja vahel olevaid operaatorid (AND ja OR).
7. INSERT lausele tuleks lisada ON CONFLICT UPDATE ja ON CONFLICT DO NOTHING klauslite tugi.

7.3 Refaktoreerimine

1. Ümber kirjutada kõik klassi komponendid funktsionaalseteks komponentideks.
2. Rakenduses kasutatav SQL lausete ehitamise teek *squel.js* [25] on aegunud ja võiks kaaluda nt teegi *knex.js* [60] kasutusel võtmist SQL lausete ehitamiseks.

7.4 Uued funktsionaalsused

1. Realiseerida uusi funktsionaalsuseid, mis selles töös tegemata jäid nagu OVER ja PARTITION BY klauslid ja ühiste tabeli avaldiste (CTE e *common table expressions*) kasutamine kõikides andmekäitluse lausetes (SELECT, INSERT, UPDATE, ja DELETE) kuna PostgreSQL seda toetab. [61] Ühised tabeli avaldised on SQL koodis hea viis lausete keerukuse haldamiseks ja loetavuse parandamiseks.
2. Võimaldada koostada lauseid, kus INSERT, UPDATE ja DELETE laused koos RETURNING klausliga on ühises tabeli avaldises (WITH klausel).
3. Võimaldada muuta UPDATE ja DELETE lause käigult ümber SELECT lauseks.
4. Võimaldada koostada MERGE lauseid, mis kombineerivad INSERT, UPDATE ja DELETE lauseid.
5. Võimaldada koostada VALUES lauseid, mis konstrueerivad päringu tulemusena tabeli kasutaja etteantud väärtustest (alternatiiv SELECT <väärtuste list> UNION SELECT <väärtuste list> ... kasutamisele).
6. Võimaldada koostada andmete kiireks kustutamiseks mõeldud TRUNCATE lauseid.
7. Võimaldada koostada SELECT lauseid, kus puudub FROM klausel.
8. Lisada kasutajaliideses erinevate elementide juurde tööriistavihjeid e *tooltip*'e.
9. Realiseerida vahendis kontrollid, mis hoiataksid enne käivitamist võimalik, et ebaõigesti koostatud lausete eest või siis ei laseks selliseid lauseid üldse koostada. Näiteks praegu saab koostada SELECT lause, kus puudub sorteerimine, kuid kus piiratakse ridade arvu FETCH FIRST n ROWS ONLY klauslit kasutades. Kuigi see lause on süntaktiliselt korrektne ei ole sorteerimise puudumise tõttu ennustatav, mis read on tulemuses.

8. Kokkuvõte

Lõputöö eesmärgiks oli lisada andmemuudatuste lausete koostamise tugi PostgreSQL veebipõhisesse visuaalsesse andmekäitluse lausete koostamise tarkvarasse *Postgres Visual Query Builder*. See avatud lähtekoodiga tarkvara on loodud Tallinna Tehnikaülikooli üliõpilaste poolt ning eelnevalt on seal realiseeritud SELECT lausete koostamise ja käivitamise funktsionaalsus. Töö käigus realiseeriti PostgreSQL andmemuudatuste lausete INSERT, UPDATE ja DELETE koostamise ja käivitamise funktsionaalsus. Eesmärgi saavutamiseks analüüsiti asjakohased teadusartikleid ja olemasolevat tarkvara, mis toetas visuaalselt andmemuudatuste lausete koostamist, et tutvuda erinevate pakutud ja realiseeritud lahendustega.

Kõigepealt analüüsiti tarkvara viimast versiooni, et tutvuda lähemalt selle kasutajaliidese, eesrakenduse ja tagarakenduse lahendustega. Koos teadusartiklite ja olemasolevate rakenduste analüüsi tulemustega moodustati sisend tööle. Arendusprotsessile eelnevalt loodi väljalaske plaan T. Normani pakutud meetodi abil ning tööd jaotati iteratsioonide vahel ära. Arendustegevuse vältel lähtuti agiilse arenduse parimatest põhimõtetest.

Töö tulemusena on rakenduses võimalik visuaalselt koostada andmemuudatuste lauseid (INSERT, UPDATE ja DELETE). Kõikide nende lause tüüpide puhul saab koostada ka PostgreSQLile omase RETURNING klausli, mis võimaldab kasutajale tagastada andmeid muudetud ridadest. Samuti saab visuaalselt koostada ühendamisoperatsioonid UPDATE ja DELETE lausetes - vastavalt FROM ja USING klausli. Täiendati ka SELECT lause koostamise funktsionaalsust, asendades genereeritud SQL koodis LIMIT klausli FETCH FIRST klausliga, mis on kasutusel SQL standardis. Kasutaja saab lisada ka WITH TIES määrangu. Samuti parandati töö käigus tarkvara kasutusmugavust ja parandati leitud vigu. Töö alguses mindi üle Node.js versioonilt 8 versioonile 18.

Töö tulemusi valideeriti nii autori enda poolt töö vältel kui ka kolme testkasutaja poolt. Testkasutajatele loodi küsimustik üheksa lause koostamise ülesandega, lisaks küsiti üldisemaid küsimusi rakenduse kasutamise kohta. Testkasutajate hinnang kasutajaliidesele oli üldiselt positiivne. Rakendust võrreldi ka kahe olemasoleva tarkvaraga (MS Access ja dbForge Studio For MySQL). Need programmid ei ole mõedud PostgreSQLi jaoks ning autoril ei õnnestunud leida tarkvara, mis võimaldaks PostgreSQL andmemuudatuste lausete visuaalset koostamist.

Lisaks toodi välja nimekiri autori poolt välja pakutud võimalikest edasiarendustest, mida järgmistes rakenduse versioonides võiks realiseerida.

Rakenduse uue versiooni lähtekood on kaitstud MIT litsensiga ja on saadaval GitHubi koodihoidlas: `https://github.com/v4hukomm/postgresql-visual-query-v3`.

Kasutatud kirjandus

[1] *DB-Engines Ranking*. [Kasutatud: detsember 2023]. URL: https://db-engines.com/en/ranking.

[2] IEEE. *IEEE Spectrum*. [Kasutatud: detsember 2023]. Aug. 2023. URL: https://spectrum.ieee.org/the-top-programming-languages-2023.

[3] IEEE. *IEEE Spectrum*. [Kasutatud: veebruar 2023]. Aug. 2022. URL: https://spectrum.ieee.org/top-programming-languages-2022.

[4] IEEE. *IEEE Spectrum*. [Kasutatud: veebruar 2023]. Aug. 2022. URL: https://spectrum.ieee.org/the-rise-of-sql.

[5] *PostgreSQL*. [Kasutatud: veebruar 2023]. URL: https://postgresql.org/.

[6] E. Dzotsenidze. "PostgreSQL veebipõhise visuaalse päringute koostamise tarkvara arendamine". BS thesis. Tallinn: Tallinna Tehnikaülikool, 2019.

[7] R. Pärnamäe. "PostgreSQL veebipõhise visuaalse andmekäitluskeele lausete koostamise tarkvara edasiarendamine". MSc thesis. Tallinn: Tallinna Tehnikaülikool, 2020.

[8] PostgreSQL Tutorial. *PostgreSQL Upsert Using INSERT ON CONFLICT statement*. [Kasutatud: detsember 2023]. URL: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-upsert/.

[9] PostgreSQL. *MERGE*. [Kasutatud: detsember 2023]. URL: https://www.postgresql.org/docs/current/sql-merge.html.

[10] J. Park ja S. Ram A. R. Hevner S. T. March. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[11] T. Norman. *Agile Release Planning 101*. [Kasutatud: veebruar 2023]. Sept. 2012. URL: http://tommynorman.blogspot.com/2012/09/agile-release-planning-101.html.

[12] *Scrum*. [Kasutatud: märts 2023]. URL: https://www.scrum.org/learning-series/what-is-scrum.

[13] *Visual Studio Code*. [Kasutatud: märts 2023]. URL: https://code.visualstudio.com/.

[14] P.Carbonnelle. *TOP IDE index*. [Kasutatud: märts 2023]. URL: https://pypl.github.io/IDE.html.

[15] Microsoft. *Microsoft Teams*. [Kasutatud: märts 2023]. URL: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software.

[16] Google. *Google Forms*. [Kasutatud: märts 2023]. URL: https://www.google.com/forms/about/.

[17] *Node*. [Kasutatud: veebruar 2023]. URL: https://nodejs.org/en/.

[18] *Express*. [Kasutatud: märts 2023]. URL: https://www.npmjs.com/package/express.

[19] *Node-postgres*. [Kasutatud: märts 2023]. URL: https://node-postgres.com/.

[20] *React*. [Kasutatud: veebruar 2023]. URL: https://reactjs.org/.

[21] *React Redux*. [Kasutatud: märts 2023]. URL: https://react-redux.js.org/.

[22] *Axios*. [Kasutatud: märts 2023]. URL: https://axios-http.com/docs/intro.

[23] *Bootstrap*. [Kasutatud: märts 2023]. URL: https://getbootstrap.com/.

[24] *Reactstrap*. [Kasutatud: märts 2023]. URL: https://github.com/reactstrap/reactstrap.

[25] *Squel.js*. [Kasutatud: märts 2023]. URL: https://hiddentao.github.io/squel/.

[26] *Introducing Hooks*. [Kasutatud: jaanuar 2024]. URL: https://legacy.reactjs.org/docs/hooks-intro.html.

[27] *React Documentation*. [Kasutatud: april 2023]. URL: https://react.dev/reference/react.

[28] A. Repenning. "Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets". In: *Journal of Visual Languages and Sentient System* 3 (2017), pp. 68–91.

[29] *Scratch*. [Kasutatud: november 2023]. URL: https://scratch.mit.edu/about.

[30] *TIOBE Index for January 2024*. [Kasutatud: jaanuar 2024]. URL: https://www.tiobe.com/tiobe-index/.

[31] M.Noone ja A.Mooney. "Visual and textual programming languages: a systematic review of the literature". In: *Journal of Computers in Education* 5 (2018), pp. 149–174.

[32] *Innovating with Ease: How Visual Programming Transforms Software Development?* [Kasutatud: november 2023]. URL: https://quixy.com/blog/visual-programming-in-software-development/.

[33] M. M. Zloof. "Query by example". In: *National Computer Conference* 44 (1975), pp. 431–438.

[34] M. M. Zloof. "Query by example: a databases language". In: *IBM Sysstems Journal* 16.4 (1977).

[35] Microsoft. *MS Access*. [Kasutatud: veebruar 2023]. URL: https://www.microsoft.com/en-us/microsoft-365/access.

[36] Active Database Software. *FlySpeed SQL Query*. [Kasutatud: veebruar 2023]. URL: https://www.activedbsoft.com/overview-querytool.html.

[37] *dbForge Studio for MySQL*. [Kasutatud: märts 2023]. URL: https://www.devart.com/dbforge/mysql/querybuilder/.

[38] *DbVisualizer*. [Kasutatud: märts 2023]. URL: https://www.dbvis.com/feature/query-builder/.

[39] *Active Query Builder*. [Kasutatud: märts 2023]. URL: https://www.activequerybuilder.com/.

[40] Oracle. *Oracle Apex*. [Kasutatud: veebruar 2023]. URL: https://apex.oracle.com/en/.

[41] *Skyvia Query Builder*. [Kasutatud: märts 2023]. URL: https://skyvia.com/query/.

[42] *DbHawk*. [Kasutatud: märts 2023]. URL: https://www.datasparc.com/.

[43] *Draxlr*. [Kasutatud: märts 2023]. URL: https://www.draxlr.com/.

[44] *Devart Youtube*. [Kasutatud: märts 2023]. URL: https://www.youtube.com/@DevartSoftware/featured.

[45] M.Cohn. *User Stories*. [Kasutatud: märts 2023]. URL: https://www.mountaingoatsoftware.com/agile/user-stories.

[46] *Fibonacci Sequence*. [Kasutatud: jaanuar 2024]. URL: https://en.wikipedia.org/wiki/Fibonacci_sequence.

[47] *What's the typical flow of data like in a React with Redux app ?* [Kasutatud: jaanuar 2024]. URL: https://www.geeksforgeeks.org/whats-the-typical-flow-of-data-like-in-a-react-with-redux-app/.

[48] *PostgreSQL FETCH*. [Kasutatud: april 2023]. URL: https://www.postgresql.org/docs/current/sql-fetch.html.

[49] *SQL Fetch*. [Kasutatud: november 2023]. URL: https://www.sqltutorial.org/sql-fetch/.

[50] *PostgreSQL SELECT*. [Kasutatud: november 2023]. URL: https://www.postgresql.org/docs/current/sql-select.html.

[51] *Standard ECMA-262 6th Edition*. [Kasutatud: april 2023]. URL: https://262.ecma-international.org/6.0/.

[52] *Webpack v4 to v5*. [Kasutatud: märts 2023]. URL: https://webpack.js.org/migrate/5/.

[53] *Webpack 5 release*. [Kasutatud: märts 2023]. URL: https://webpack.js.org/blog/2020-10-10-webpack-5-release/#automatic-nodejs-polyfills-removed.

[54] A. Laycock. *Don't eject your Create React App*. [Kasutatud: märts 2023]. URL: https://medium.com/curated-by-versett/dont-eject-your-create-react-app-b123c5247741.

[55] *Airbnb JavaScript Style Guide*. [Kasutatud: november 2023]. URL: https://airbnb.io/javascript/react/.

[56] *Jest*. [Kasutatud: märts 2023]. URL: https://jestjs.io/.

[57] *Moxios*. [Kasutatud: märts 2023]. URL: https://github.com/axios/moxios.

[58] *Redux-mock-store*. [Kasutatud: märts 2023]. URL: https://github.com/reduxjs/redux-mock-store.

[59] *Node.js Wikipedia*. [Kasutatud: jaanuar 2024]. URL: https://en.wikipedia.org/wiki/Node.js.

[60] *Knex.js SQL Query Builder for JavaScript*. [Kasutatud: jaanuar 2024]. URL: https://knexjs.org/.

[61] *PostgreSQL WITH Queries (Common Table Expressions)*. [Kasutatud: jaanuar 2024]. URL: https://www.postgresql.org/docs/current/queries-with.html.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

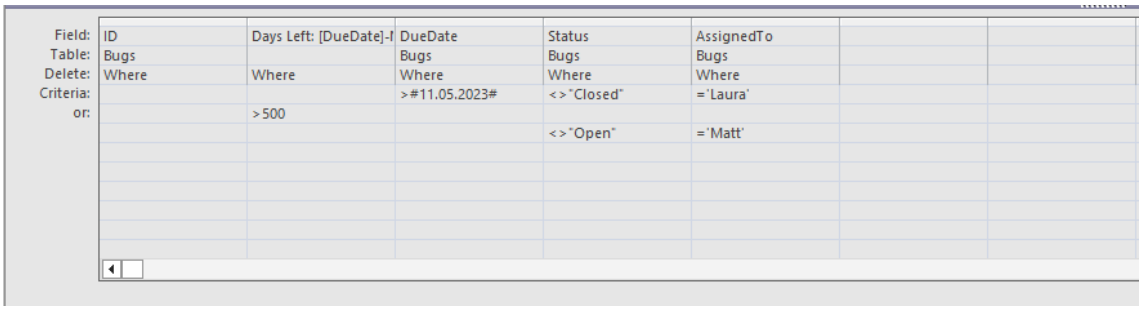
Mina, Karel Markus Mulk

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Andmemuudatuste toe lisamine PostgreSQL veebipõhisesse visuaalsesse andmekäitluse lausete koostamise tarkvarasse", mille juhendaja on Erki Eessaar
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

11.01.2024

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – MS Access filtri koostamise vaade



Field:	ID	Days Left: [DueDate]-[	DueDate	Status	AssignedTo		
Table:	Bugs		Bugs	Bugs	Bugs		
Delete:	Where	Where	Where	Where	Where		
Criteria:			>#11.05.2023#	<>"Closed"	='Laura'		
or:		>500					
				<>"Open"	='Matt'		

Lisa 3 – Testkasutajatele antud andmebaasi loomise laused

```
CREATE TABLE osakond (
osakond_nr INTEGER NOT NULL,
osakond_nimi VARCHAR(14),
asukoht VARCHAR(13),
CONSTRAINT pk_dept2 PRIMARY KEY (osakond_nr));

CREATE TABLE osakond1 (
osakond_nr INTEGER NOT NULL,
osakond_nimi VARCHAR(14),
asukoht VARCHAR(13),
CONSTRAINT pk_dept3 PRIMARY KEY (osakond_nr));

CREATE TABLE tootaja (
tootaja_nr INTEGER NOT NULL,
tootaja_nimi VARCHAR(10),
tookoht VARCHAR(9),
juht INTEGER,
hiredate DATE,
palk DECIMAL(20,4),
boonus DECIMAL(20,4),
osakond_nr INTEGER,
CONSTRAINT pk_emp2 PRIMARY KEY (tootaja_nr));

ALTER TABLE tootaja ADD CONSTRAINT fk_emp_deptno FOREIGN KEY (
   osakond_nr) REFERENCES osakond(osakond_nr) ON UPDATE CASCADE;
CREATE INDEX idx_emp_deptno1 ON tootaja(osakond_nr);
```

Lisa 4 – Testkasutajatele antud ülesanded

INSERT lause 1

Sisestage uus osakond, kus osakonna number on 1010, nimeks 'Osakond A' ja asukohaks on NULL.

```
INSERT
INTO public.osakond (osakond_nr, osakond_nimi, asukoht)
VALUES (1010, 'Osakond A', NULL)
;
```

INSERT lause 2

Sisestage kaks uut osakonda. Esimese number on 1000 ja nimeks 'Osakond B'. Teise number on 1100 ja nimeks 'Osakond C'. Tagasta tulemuses osakonna number.

```
INSERT
INTO public.osakond (osakond_nr, osakond_nimi)
VALUES (1000, 'Osakond B'), (1100, 'Osakond C')
RETURNING osakond.osakond_nr;
```

INSERT lause 3

Sisestage kõik osakonnad tabelist **osakond**, mille osakonna number on suurem kui 100, tabelisse **osakond1**. Sisestage ainult osakonna number ja nimi. Tulemuses tagastage kõik veerud.

```
INSERT
INTO public.osakond1 (osakond_nr, osakond_nimi)
SELECT
(osakond_nr, osakond_nimi)
FROM public.osakond
```

```
WHERE (osakond.osakond_nr > 100)
RETURNING *;
```

UPDATE lause 1

Suurendage kõikide töötajate palkasid 100 võrra. Tulemuses tagastage palgad.

```
UPDATE
public.tootaja
SET palk = palk + 100
RETURNING palk;
```

UPDATE lause 2

Suurendage kõikide töötajate palka 100 võrra, kelle praegune palk on alla 2000 või kelle töökoht on 'MANAGER' ja boonus NULL.

```
UPDATE
public.tootaja
SET palk = palk + 100
WHERE ((tootaja.palk  < 2000) OR
(tootaja.tookoht = 'MANAGER' AND tootaja.boonus IS NULL))
;
```

UPDATE lause 3

Suurendage kõikide töötajate palka 100 võrra, kes töötavad osakonnas, mille asukoht on 'TALLINN'.

```
UPDATE
public.tootaja
SET palk = palk + 100
FROM public.osakond
WHERE (tootaja.osakond_nr = osakond.osakond_nr)
```

```
AND ((osakond.asukoht = 'TALLINN'))
;
```

DELETE lause 1

Kustutage kõik töötajad. Tulemuses tagastage kõik veerud.

```
DELETE
FROM public.tootaja
RETURNING *;
```

DELETE lause 2

Kustutage kõik töötajad, kelle töötaja number on 7369 või kelle palk on alla 1500. Tulemuses tagastage töötaja palk ja nimi.

```
DELETE
FROM public.tootaja
WHERE ((tootaja.tootaja_nr = 7369)
OR (tootaja.palk < 1500))
RETURNING tootaja.palk, tootaja.tootaja_nimi;
```

DELETE lause 3

Kustutage kõik töötajad, kes töötavad osakonnas asukohaga 'TALLINN' või kelle osakonna nimi hakkab 'C' tähega. Tulemuses tagastage kõik veerud.

```
DELETE
FROM public.tootaja
USING public.osakond
WHERE(tootaja.osakond_nr = osakond.osakond_nr)
AND ((osakond.asukoht = 'TALLINN')
OR (osakond.osakond_nimi LIKE 'C%'))
RETURNING *;
```

Lisa 5 – Testkasutajate küsimustik

Kuidas hindate oma PostgreSQL oskusi?

- ◯ "Hea" - Oskan koostada väga keerulisi lauseid, olen loonud enda Postgres andmebaasi
- ◯ "Keskmine" - Oskan koostada lihtsamaid ja keerukamaid lauseid
- ◯ "Minimaalne" - Oskan koostada lihtsaid lauseid
- ◯ "Puudulik" - Pole varem PostgreSQL-iga kokku puutunud.

INSERT lause 1

Sisestage uus osakond, kus osakonna number on 1010, nimeks 'Osakond A' ja asukohaks on NULL väärtus

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

INSERT lause 2

Sisestage kaks uut osakonda. Esimese number on 1000 ja nimeks 'Osakond B'. Teise number on 1100 ja nimeks 'Osakond C'. Tagasta tulemuses osakonna number.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

INSERT lause 3

Sisestage kõik osakonnad tabelist **osakond**, mille osakonna number on suurem kui 100, tabelisse **osakond1**. Sisestage ainult osakonna number ja nimi. Tulemuses tagastage kõik veerud.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

UPDATE lause 1

Suurendage kõikide töötajate palkasid 100 võrra. Tulemuses tagastage palgad.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

UPDATE lause 2

Suurendage kõikide töötajate palka 100 võrra, kelle praegune palk on alla 2000 või kelle töökoht on 'MANAGER' ja boonus NULL.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

UPDATE lause 3

Suurendage kõikide töötajate palka 100 võrra, kes töötavad osakonnas, mille asukoht on 'TALLINN'.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

DELETE lause 1

Kustutage kõik töötajad. Tulemuses tagastage kõik veerud.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

DELETE lause 2

Kustutage kõik töötajad, kelle töötaja number on 7369 või kelle palk on alla 1500. Tulemuses tagastage töötaja palk ja nimi.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

DELETE lause 3

Kustutage kõik töötajad, kes töötavad osakonnas asukohaga 'TALLINN' või kelle osakonna nimi hakkab 'C' tähega. Tulemuses tagastage kõik veerud.

Kas saite lause koostamisega hakkama?

- ◯ Jah
- ◯ Ei

Kuidas saite hakkama lause koostamisega?

- ◯ "Hästi" - Ei esinenud takistusi
- ◯ "Keskmiselt" - Esinesid väiksemad takistused
- ◯ "Halvasti" - Esinesid suured takistused
- ◯ Ei suutnud lauset koostada

Põhjendage eelmist vastust.

Teie vastus

Kas rakenduse kasutamine oli mugav

- ◯ Jah
- ◯ Ei

Mis teile rakenduse kasutamise juures meeldis?

Teie vastus

Mis teile rakenduse kasutamise juures ei meeldinud?

Teie vastus

Kuidas võiks rakendust paremaks teha?

Teie vastus

Kui leidsite rakenduse kasutamisel mõne vea, kirjutage see siia.

Teie vastus