TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Raiko Limmart 183153IABM

ANDMEAITADE TESTIMISE AUTOMATISEERIMINE

Magistritöö

Juhendaja: Jaak Tepandi
Professor

Tallinn 2020

Autorideklaratsioon

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

Autor: Raiko Limmart

09.05.2020

Annotatsioon

Lõputöö eesmärkideks on uurida andmekvaliteedi probleeme ja vigade tekkepõhjuseid andmeaitades ning tutvuda erinevate andmeaitade testimisvahenditega. Töö praktilise osa eesmärgiks on luua automaatne andmete testimise vahend, mis juurutatakse suurettevõtte süsteemidesse ning analüüsitakse selle kasutuse tulemusi.

Töö tulemuseks on veebirakendusena kasutatav andmeaitade testimise tööriist, mis võimaldab automatiseerida testide loomist, nende käivitamist ning tulemuste raporteerimist. Loodud rakendus on võetud kasutusele analüüsitavas suurettevõttes ning magistritöö osana kasutati tööriista ühe andmedomeeni andmekvaliteedi kontrollimiseks ja parendamiseks. Rakendust kasutab ettevõttes andmeaida meeskond ning selle eesmärgiks on meeskonnaliikmete töö lihtsustamine ning ettevõtte andmeaida andmekvaliteedi tõstmine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 7 peatükki, 65 joonist, 2 tabelit.

Abstract

Automating data warehouse testing

Data warehouse is the core of business intelligence and the data in it directly defines its value in a company. As companies become more data-driven and the data generated is growing rapidly, it is important to automate data validation to sustain its usability and value.

The purpose of this thesis is to develop an automated testing solution for data warehouses and to implement it into an enterprise system. This is done by studying data quality issues and causes of poor data quality in data warehouses and by comparing different data testing solutions.

The main result of this thesis is an automated testing tool, which is implemented into an enterprises system and used to analyse data quality in Vertica data warehouse. The tool is a web application which can automatically generate test cases from predefined reusable tests and execute these SQL test cases against databases. The results can be automatically reported through email as PDF files, displayed on the tool's dashboard as graphs and viewed from centralized test case page. The tool is used by a team of data warehouse analysts to ease their work and improve the quality of data in enterprise's data warehouse.

The thesis is in Estonian and contains 48 pages of text, 7 chapters, 65 figures, 2 tables.

Lühendite ja mõistete sõnastik

Data Vault	Andmebaaside modelleerimise meetod
DOM	*Document Object Model*, veebilehitseja poolt loodud dokument HTML objektidele ligipääsuks
ETL	*Extract Transform Load*, protsess andmete migreerimiseks ühest andmebaasist teise
HTML	*HyperText Markup Language* ehk hüperteksti märgistuskeel on keel veebilehtede loomiseks
ISO	*International Organization for Standardization*
JDBC	*Java Database Connectivity*, tehnoloogia andmebaasidega suhtlemiseks ning sealt andmete pärimiseks
jQuery	JavaScript funktsioonide kogumik
MVC	*Model View Controller* ehk tarkvara arhitektuur, mis sisaldab andmeid, kasutajaliidest ning loogikat
ODI	*Oracle Data Integrator*, ETL protsesside loomiseks ja käivitamiseks kasutatav tööriist
PDF	*Portable Document Format*, failiformaat tekstide ja piltide kuvamiseks
SQL	*Structured Query Language* ehk struktureeritud päringukeel andmebaasidest andmete pärimiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Magistritöös uuritakse andmekvaliteedi probleeme andmeaidas, nende tekkepõhjuseid, andmekvaliteedi mõju ning võrreldakse olemasolevaid testimisvahendeid. Töö põhiülesandeks on andmeaitade automaatse testimisvahendi loomine, selle juurutamine suurettevõttesse ning tulemuste analüüsimine.

1.1 Taust ja probleem

Tänapäeval toimub ettevõtetes suur osa aruandlusest ning tehtud äriotsustest andemaita laetud andmete põhjal. Andmete konsolideerimisel tehakse tihti ümberarvutusi, andmeid agregeeritakse või viiakse sisse muid muudatusi. Selle probleemiks on testimata andmed ning üldine andmebaasi arhitektuur andmeaidas, mis mõjutab otseselt andmete peale ehitatud aruandlust ning andmebaasi võimekust.

Andmeaida arendus on pidev ning selle testimine on tihti arendaja enda vastutada, mistõttu on vead lihtsasti tekkivad. Üldiselt testitakse ainult arendaja enda arenduses käideldavaid andmeid ja seda vaid näidisandmetega, mitte ei pöörata tähelepanu suurele pildile ning mõju ülejäänud andmetele. Isegi lihtsamate arenduste korral tekkinud vead võivad olla suure mõjuga, mistõttu peaksid need olema kergesti avastatavad läbi automaatse testimise. Andmete testimine peaks olema süstemaatiline ning katma nii andmeaida tehnilise poole kui ka kindlustama andmete vastavuse ärireeglitele.

Ettevõtted kasutavad sageli testimiseks ebapraktilisi lahendusi või kulutavad ressursse poolikutele lahendustele, millel puudub automaatsus või paindlikkus. Selliste lahenduste kasutamine on kulukas ning ebamugav ja ei lahenda tihti esialgset probleemi. Arvestades, et andmeaitades on tavaliselt tuhandeid tabeleid, on kõikide tabelite ja veergude testimiseks tehtav käsitöö mõeldamatu, põhjustades vigaseid andmeid.

1.2 Ülesande püstitus

Töö eesmärgiks on uurida ja analüüsida andmeaitasid, andmekvaliteeti ning kuidas andmeaidas olevate andmete testimist on lahendatud. Analüüsi tulemusena leitakse

erinevaid võimalusi andmeaida testimiseks ja luuakse teste, mida on võimalik automatiseerida ning mille kasutust on võimalik rakendada töö praktilises osas.

Praktilise osa eesmärgiks on luua automatiseeritud andmeaida testimise veebirakendus suurettevõtte andmeaida testimiseks. Rakendus peab suutma luua ühenduse ettevõttes kasutatavate andmebaaside vastu ning omandama sealt metaandmeid. Nende põhjal võimaldab programm luua erinevaid ettemääratud teste ning neid vastavalt kasutaja soovidele seadistada. Kasutajal on võimalik ka ise teste luua, kohandades neid vastavalt ärireeglitele või andmeaida arhitektuurile. Loodud teste on võimalik panna pakettidesse, mille käivitamist saab vastavalt vajadusele automatiseerida. Iga paketi käivituse kohta saab luua automaatse aruandluse ning ilmnenud vigadest on võimalik vastutavat isikut e-maili teel automaatselt teavitada. Rakendus kogub statistikat ja suudab tuvastada erinevusi käivituste vahel, andes kasutajale teada andmete korrapäratusest.

Rakenduse eesmärgiks on tuvastada andmeaidas tekkinud vead, et nendest andmete otsesele kasutajale võimalikult vähe negatiivset mõju oleks. Tulemusena paraneb üldine andmeaida kvaliteet ning suureneb usaldus andmeaidas olevate andmete vastu. Rakenduse kasutamine parandab ka uutele arendustele kuluvat aega ning muudab selle protsessi efektiivsemaks ja ettevõttele odavamaks. Arendatud tööriist juurutatakse analüüsitava ettevõtte süsteemidesse ning analüüsitakse kasutuse tulemusi.

1.3 Metoodika

Töö metoodikaks on *design science*, mille tulemusena valmib kasutatav rakendus. Kavandatavaks rakenduseks on automatiseeritud testimise tööriist, mis on mõeldud eelkõige andmeaitade äriloogika testimiseks. Tööriista vajadus on tingitud probleemidest andmetes, mida käsitsi kontrollida ei ole võimalik. Tulemusena valmivat rakendust hakatakse igapäevaselt kasutama ühe Eesti ettevõtte andmeaida andmete testimiseks, mille tulemust on võimalik hinnata andmete testimise käigus tuvastatud vigade analüüsimisel. Töö loob ettevõttele andmete automaatse testimise lahenduse, millelaadset avalikult kättesaadaval ei ole.

Töö esimeses pooles analüüsitakse, kuidas andmeaitade testimist on siiani lahendatud ja milliseid lahendusi on testimise automatiseerimiseks kasutatud. Uuritakse erinevaid andmete testimise viise ning meetodeid ja kogutud andmete põhjal luuakse testimise

automatiseerimises kasutatavaid SQL testijuhte. Analüüsi tulemuseks on sisend töö praktilisele osale.

Töö praktiline osa on veebirakenduse ehitamine ning selle juurutamine suurettevõtte süsteemidesse. Rakendus ehitatakse *Spring Boot* aplikatsioonina, kasutades Java programmeerimiskeelt *Spring* raamistikus. Kogu arendus toimub kasutades *IntelliJ IDEA* keskkonda, mis võimaldab kompileerida ja käivitada loodud Java rakendust. Rakendus paigaldatakse ettevõtte sisevõrgus asuvale virtuaalmasinale, millele on ligipääs andmeaida arenduse ja testimisega tegelevatel töötajatel. Tööriist ühildub ettevõttes kasutatavate andmebaasidega kasutades JDBC draivereid. Andmebaasidest laetakse rakenduse lokaalsesse SQLite andmebaasi metaandmed, mille pealt võimaldab tööriist ehitada automatiseeritud testijuhte.

1.4 Ülevaade tööst

Kõik magistritöös kasutatavad viitamised on 2020 aasta seisuga ning töö koosneb viiest suuremast sisupeatükist. Peatükis number 2 kirjeldatakse andmeaitade olemust, andmete laadimist ning tutvutakse analüüsitavas suurettevõttes kasutatava andmeaidaga.

Peatükis number 3 räägitakse andmete kvaliteedist, vigaste andmete põhjustajatest ning uuritakse vigaste andmete mõju.

Peatükis number 4 käsitletakse andmeaitade testimise viise ja uuritakse ning võrreldakse testimiseks kasutatavaid tööriistu.

Peatükis number 5 toimub andmeaitade testimise automatiseerimiseks mõeldud rakenduse arendamine ja selle funktsioonide kirjeldamine.

Peatükis number 6 juurutatakse magistritöös arendatud tööriist suurettevõtte süsteemidesse ning analüüsitakse selle kasutuse tulemusi.

2 Andmeaidad

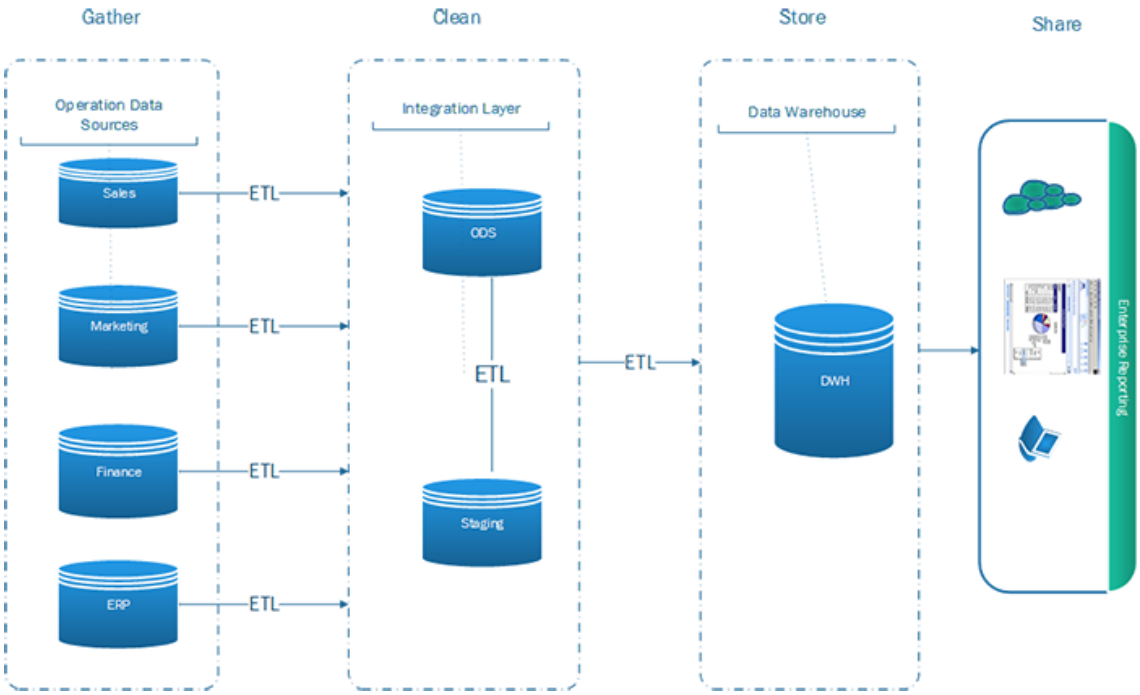
Andmeait on sisule orienteeritud, ühtne, ajast sõltuv ning püsiv kogu andmeid, mille eesmärgiks on anda asjakohast informatsiooni strateegiliste otsuste tegemiseks [1]. Tegu on ettevõttesisese informatsiooni koguga, kuhu laetakse kokku kõik ettevõtte süsteemide andmed. Andmeid hoiustatakse ühes kindlas andmebaasis ning kasutatakse ühist struktuuri. Selle tulemusena on kasutajatel võimalik saada kätte organiseeritud ja struktureeritud andmeid ettevõtte ja selle osakondade erinevatest süsteemidest [2]. Analüütikud kasutavad andmeaidas olevaid andmeid, et leida lahendusi keerulistele küsimustele, millele oleks keeruline vastust leida operatiivsetest andmebaasidest. Sellest tulenevalt ongi andmeaida eesmärgiks pakkuda kasutajatele paremaid võimalusi ettevõtte andmete kasutamiseks [3].

Andmete laadimine on üks kõige ajakulukamaid protsesse andmeaida arendusest. Andmete konsolideerimisel allikatest võib esineda probleeme, mis tuleb lahendada enne andmeaita laadimist. Andmed võivad olla vigased, ebaühtlased, loetamatud või poolikud. Andmete korrastamine laadimisel on oluline, et saavutada kvaliteetne andmeait [4]. See tagab ka andmeaida hooldatavuse, mida peetakse üheks kõige tähtsamaks tarkvarakvaliteedi osaks [1].

2.1 Andmete laadimine

Andmete laadimiseks kasutatakse ETL protsesse, mis laevad andmeid allikatest ning sisestavad need andmeaita kindla formaadi ja struktuuri järgi. ETL protsessi esmane samm on andmete laadimine allikast. Tavaliselt laetakse andmeid, mida ei ole enne laetud või mis erinevad juba andmeaita laetud andmetest. Seejärel sisestatakse andmed vahekihti, kus need puhastatakse ja transformeeritakse. Enamasti andmed filtreeritakse ning kontrollitakse, et need vastaksid reeglitele ning muudele piirangutele. Seejärel laetakse andmed edasi kesksesse andmeaita ja selle osadesse, kus andmeid konsolideeritakse ning agregeeritakse vastavalt vajadustele (Joonis 1) [5]. ETL protsessi disainimine on väga keerukas, kergesti vigu tekitav ning ajakulukas. Öeldakse, et ETL

protsesside arendamine on üks kõige tähtsamaid ning kallimaid andmeaida arenduse osasid, hõivates kolmandiku andmeaida eelarvest [6].



Joonis 1. ETL protsess ning andmeaida üldine arhitektuur [7].

2.2 Andmeait analüüsitavas suurettevõttes

Analüüsitavas ettevõttes on kasutuses andmeait, mis töötab Vertica andmebaasimootoril ning mille suuruseks on ligikaudu 8 terabaiti. Laadimisteks kasutatakse Oracle Data Integrator ETL tööriista, mis võimaldab laadida suuri andmemahte ning rakendada skriptide korduvkasutatavust kasutades *Knowledge Moduleid*. Suurem osa laetavatest andmebaasidest on Oracle, PostgreSQL või Microsoft SQL Server põhised ning enamus nendest on ettevõttesisesed operatiivsed andmebaasid. Laadimised ja transformatsioonid toimuvad iga öö kella 00:00-07:00 vahel. Andmeaida mudeldamiseks kasutatakse Data Vault modelleerimise meetodit, mis lihtsustab andmeaida disaini, toetab suuri andmehulki ning võimaldab olemasolevat disaini lihtsasti muuta [8].

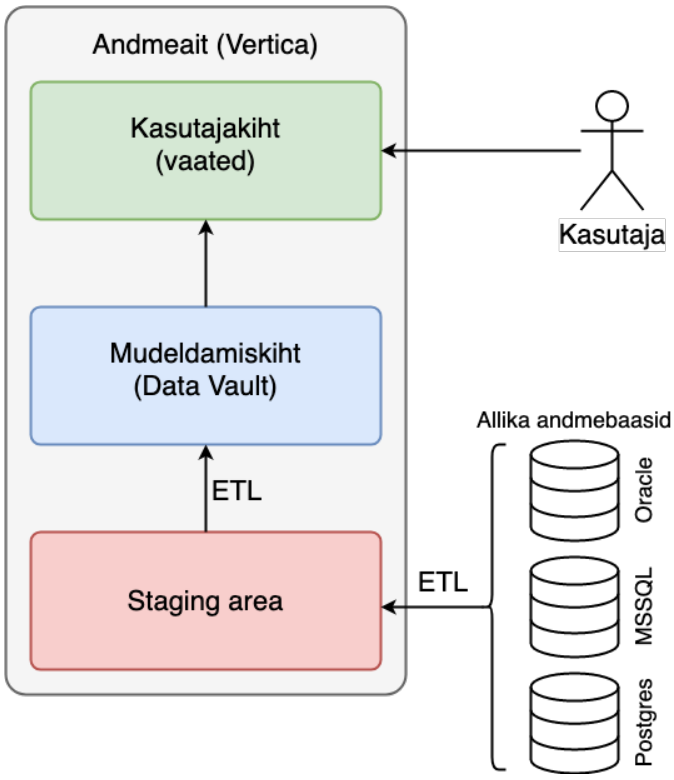
2.2.1 Üldine arhitektuur

Ettevõtte andmeait koosneb kolmest põhilisest kihist, mis omakorda koosnevad andmevaldkondadest ehk andmedomeenidest (Joonis 2):

- *Staging area* ehk laadimiste vahekiht, kuhu laetakse andmeid allika andmebaasidest. Loodavad ja laetavad tabelid nimetatakse allika nime ning laetava tabeli nime kombinatsioonist. Tabelite veergude nimed on samad, mis on allika tabelis, ning laetakse vaid neid veerge, mida reaalselt andmete lõppkasutajatel vaja läheb. Kokku on selles kihis ~1200 tabelit ning enamik nendest tühjendatakse iga öö ja laetakse uued andmed sisse.

- Mudeldamiskiht, kuhu laetakse ja transformeeritakse andmeid *Staging Areast*. Selles kihis on andmed mudeldatud vastavalt kokkulepitud arhitektuurireeglitele ning kasutatakse *Data Vault* mudeldamismeetodit. Mudeldatud andmed on jagatud andmedomeenidesse, kuhu sarnased andmed konsolideeritakse. Selles kihis on ligi 500 tabelit, millest vastavalt *Data Vault* mudelile on ~100 *Hub*, ~200 *Satellite*, ~150 *Link* ning ~50 *Reference* tüüpi tabelit.

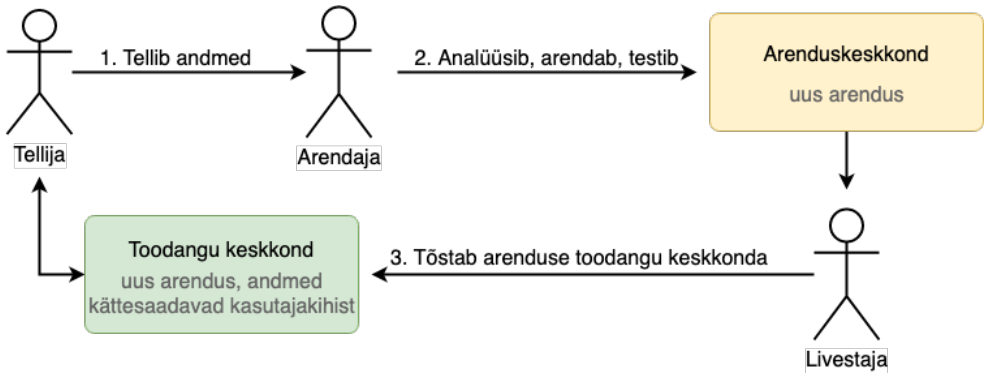
- Kasutajakiht ehk vaadete kiht – koosneb vaadetest, mis on andmete otsestele kasutajatele kättesaadavad. Vaated pannakse kokku mudeldamiskihis olevatest tabelitest ning need vastavad kokkulepitud arhitektuurireeglitele. Kokku on loodud ~900 vaadet.



Joonis 2. Andmeaida loogiline arhitektuur analüüsitavas ettevõttes.

2.2.2 Arendusprotsess ja andmete testimine

Andmeaida arendusprotsessiks analüüsitava ettevõtte mõistes loetakse protsessi, kus vastavalt andmete tellija vajadustele luuakse või muudetakse andmeaida kasutajakihi vaateid. Vastavalt vajadusele laetakse andmed allika andmebaasidest ning seejärel need mudeldatakse, konsolideeritakse, agregeeritakse ja tehakse tellijale kättesaadavaks. ETL protsesse luuakse ning käivitatakse läbi ODI, paigutades need pakettidesse, mida iga öö plaaniliselt käivitatakse. Arendus toimub eraldi keskkonnas, mis on struktuurilt ja olemuselt sarnane toodangu keskkonnaga. Arenduse ja selle testimise eest vastutab arendaja, kes peab olema veendunud, et arenduse ning andmete kvaliteet on korras ja ei põhjusta toodangu keskkonnas vigu. Andmete testimist teeb arendaja käsitsi ning käesoleval hetkel ei ole kasutuses tööriista, mis aitaks seda protsessi automatiseerida. Kui arendus on valmis, siis tõstetakse see toodangu keskkonda teise isiku poolt, kes kontrollib ka üldist vastavust arhitektuurile ning arenduskokkulepetele (Joonis 3). Toodangu keskkonnas tutvub esialgne andmete tellija uute andmetega ning annab teada, kas andmed vastavad tema ootustele.



Joonis 3. Arendusprotsess analüüsitava ettevõtte andmeaidas.

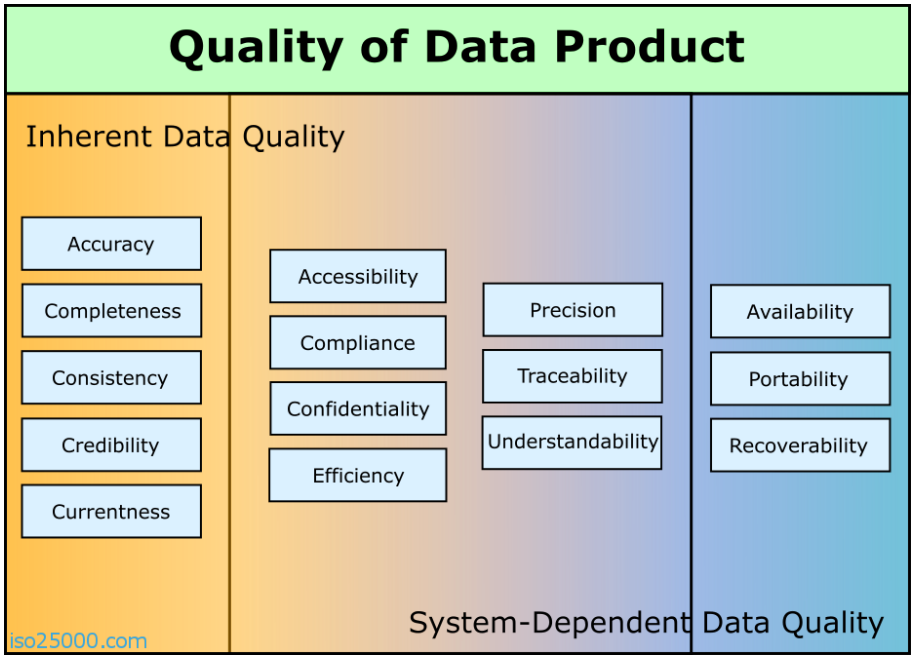
3 Andmete kvaliteet

Andmekvaliteedil on alati olnud oluline roll otsuste tegemisel ning strateegilisel planeerimisel, mistõttu loodi 1963 ütlus *garbage in, garbage out* (prügi sisse, prügi välja), et rõhutada andmekvaliteedi tähtsust [9]. Andmekvaliteedi kontekstis mõiste „kvaliteet“ tähendab taset, mis näitab andmete omaduste vastamist kindlatele nõuetele. Ettevõtte jaoks on nende andmed kvaliteetsed, kui need on põhjalikud, arusaadavad, järjepidevad, asjakohased ning ajakohased [10].

Andmeaitade tulek 80-ndatel tekitas andmekvaliteediga seoses uut laadi probleeme. Strateegiliste otsuste tegemiseks hakati erinevaid andmeid ühte kohta kokku koguma, neid agregeerima ja siduma. See põhjustas puuduolevaid, ebaühtlasi ning ebakorrektseid andmeid, mille tõttu oli vaja hakata sellistele probleemidele keskenduma [9]. Enim hinnatakse andmete juures nende [11]:

- Täpsust (*accuracy*) – kui täpselt andmed kirjeldavad olusid päris maailmas. Ebatäpsed andmed põhjustavad valesid tulemusi ja otsuseid [11].
- Täielikkust (*completeness*) – andmed on täielikud, kui neis ei ole puuduolevaid osasid. Kõik andmed, mida on vaja analüüsida, peavad olema olemas ning kättesaadavad [11].
- Asjakohasust (*relevancy*) – pole mõtet koguda andmeid, mida kunagi ei kasutata. Kui andmed ei loo mingit väärtust, siis ei ole nendest mingit kasu [11].
- Valiidsust (*validity*) – andmed peavad vastama õigele formaadile ning tüübile. Kui andmed on vigaselt sisestatud või laetult, siis on neid raske organiseerida ja analüüsida [11].
- Ajakohasust (*currentness/timeliness*) – ajaga muutuvad andmed vähem täpsemaks ning kasulikumaks. Andmed peavad olema ajaliselt vastavad päris maailma olukorrale [11].
- Järjepidevust (*consistency*) – kui võrrelda samu andmeid erinevates süsteemides, siis peaksid need kattuma. Erinevused nii sisus kui formaadis võivad põhjustada töös vigu, sest süsteemikasutajatel on erinev arusaam andmetest [11].

Need omadused kattuvad osaliselt ka rahvusvahelise andmekvaliteedi standardi ISO 25012-ga, mida iseloomustavad 15 erinevat tunnust (Joonis 4). Näiteks, kui andmebaasist on mingi tabel või andmete kogumik puudu, siis iseloomustab neid andmete kättesaadavus (*availability*). Kui andmed on küll olemas, aga nendele puudub ligipääs, siis iseloomustab neid ligipääsetavus (*accessability*). Andmed võivad vastata korraga mitmele omadusele ehk olla puudulikud mitmes dimensioonis [12].



Joonis 4. ISO/IEC 25012 andmekvaliteedi standardi omadused [13].

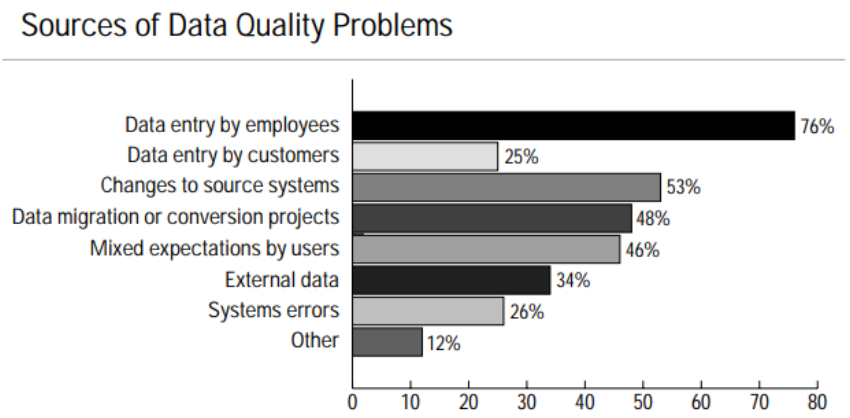
Andmete kvaliteedi hindamisel tuleb kindlasti arvestada ka lõppkasutajate hinnangutega. Lõppkasutajad oskavad hinnata andmete kättesaadavust, kasulikkust, tõlgendatavaust, kehtivust ning usaldatavust. Sellele põhinedes on lõppkasutajad kõige õigemad hindama andmekvaliteeti andmeaidas [5]. Kvaliteedi tagamiseks tuleb andmete konstrueerimisel ja laadimisel tegeleda paralleelselt ka nende testimisega [14].

Andmekvaliteedi tõstmine on parendusprotsessi tulem, kus tehakse kindlaks ning kõrvaldatakse vigaste andmete põhjustajad. Põhiline osa andmekvaliteedi tõstmisel on oskus teha vahet „headel“ ja „halbadel“ andmetel. Probleemiks on andmete erinev sisu, formaat ning raamistik, mille tulemusena on andmete valideerimine väga keeruline protsess. Andmekvaliteedi tõstmiseks tuleb hinnata andmekvaliteedi taset ja defineerida andmete valideerimisreeglid [15].

3.1 Vigaste andmete põhjused

Andmete kvaliteet on ettevõtete jaoks väljakutse, mis muutub ajaga aina keerulisemaks. 91% ettevõtetest kannatavad üldlevinud vigade käes, milleks on ebatäielikud või puuduolevad andmed, vananenud informatsioon ning ebatäpsed andmed. Kõige levinum põhjus vigaste andmete tekkimisel on inimlik eksimus (Joonis 5). Tihti on see tingitud sellest, et erinevates süsteemides sisestatakse nii kliendi kui ka töötajate poolt andmeid käsitsi [16]. Sarnane tulemus oli ka 2001 aasta uuringus, kus leiti, et peamine vigaste andmete põhjustaja on töötajate poolt vigaselt sisestatud andmed (Joonis 6). Sinna kuuluvad näiteks kirjavead, komakohtade vead, valedele väljadele sisestatud andmed, akronüümide kasutamine ja lühendid [17].

Joonis 5. *Reason for data inaccuracy*. Levinumad vigaste andmete põhjustajad [16].



Joonis 6. *Data entry and changes to source systems are the biggest causes of data quality problems for data warehousing managers*. Andmeaida juhtide poolt hinnatud peamised vigaste andmete põhjustajad [17].

Andmeaitades võivad andmevead tekkida erinevates kihtides. Laadimiskihis võivad vead tekkida andmete puhastamisel ja filtreerimisel, mida tehakse selleks, et andmeaidas olevad andmed oleksid täpsemad. Mudeldamiskihis on vead lihtsasti tekkivad, sest

sõltuvad mitmetest erinevatest omadustest: töödeldavate andmete kvaliteedist, kasutatavatest programmidest ning andmebaasi arhitektuurist. Andmeaida arhitektuurist ja mudeldamismeetoditest sõltub mida ja kuidas seal olevate andmetega teha on võimalik ning kuidas neid andmeid sinna laetakse [18]. Kõige rohkem tekitavad andmeaidas vigaseid andmeid probleemid andmete migreerimisel allikatest, mida võivad põhjustada [10]:

- Ebasobilike allikate valik ning ebapiisav teadmine nende sõltuvuste kohta [10].
- Varieeruv andmete ajakohasus ja selle puudulik valideerimine allikas [10].
- Ootamatud muudatused allikas ja andmeaida keerukusest tingitud piirangud [10].
- Erinevad andmete formaadid ning nende mittevastavus [10].
- Samade kirjete olemasolu mitmes erinevas allikas [10].
- Selliste allikate kasutamine, mille üle ettevõttel puudub kontroll [10].
- Puudulik andmete valideerimine allikas [10].
- Allika andmete vastutava omaniku puudumine [10].
- Valesti sisestatud andmed ning katkised vormid andmete sisestamiseks [10].

3.2 Vigaste andmete mõju

Ennustatakse, et aastaks 2025 on maailmas andmeid kümme korda rohkem kui neid oli aastal 2017 – kokku 163 zettabaiti. Sellist kasvu põhjustavad uudsed targad süsteemid, näiteks targad kodud ja isesõitvad autod, mis on pidevalt ühenduses ja suhtlevad erinevate süsteemidega. Selle tulemusena peavad ettevõtted aktiivsemalt tegelema andmehaldusega, et töötada efektiivselt, pakkuda kliendile head kasutajakogemust ning mitte kaotada potentsiaalset tulu [19].

Vigased andmed on ettevõtetele väga kulukad. Puudulikud andmed põhjustavad halbade otsuste tegemist tarneahelates ning lõhuvad kliendisuhteid. Andmed on ettevõtte põhiline ärivara ja selle haldus ning juhtimine on vajalik, et ettevõttele sellest mingit kasu oleks [4]. Umbes 60% ettevõtetest ei ole mõõtnud, kui palju kahju halb andmekvaliteet nende

ettevõttele põhjustab. Seetõttu ei ole sellised ettevõtted ka aktiivsed vigade parandamises ning kaotavad potentsiaalseid kasvuvõimalusi turul. Keskmiselt maksavad aastas halvad andmed ühele ettevõttele 15 miljonit dollarit [20] ning enamikele ettevõtetele ennustatakse kadu 15-25% käibest [21]. Andmeaitades peetakse andmekvaliteediga seoses kõige suuremateks probleemideks andmeaidas olevate andmete kasutatavust ning tööjõu kulu (Joonis 7) [22].

Joonis 7. *Impacts of bad data quality to data warehouse projects*. Ebapiisava andmekvaliteedi mõju andmeaida projektides [22].

Andmetega tegelevad inimesed kulutavad kuni 50% oma ajast andmekvaliteedi probleemidele ning vigaste andmetega tegelemisele. Andmeteadlaste puhul võib see ulatuda kuni 80%-ni. Lisaks ajakulule põhjustavad vigased andmed vigu ettevõtte tegevustes, mõjutavad ärilisi otsuseid ning ärianalüütikat. Selle tulemusena usaldavad täielikult andmeid vaid 16% juhtidest, kes selle põhjal äriotsuseid teevad. Ettevõtetel on keeruline olla andmetel põhinev, kui kasutatavaid andmeid ei ole võimalik usaldada [21]. Andmekvaliteedist tingitud halbade äriotsuste tulemusena võivad ettevõtted näiteks avada poode tänavatel, mis tegelikkuses ei vasta analüüsitud demograafiale, tuues sisse palju vähem tulu kui algselt ennustati. Valed otsused võivad mõjutada ka näiteks ettevõtte poolt pakutava rakenduse optimeerimist, mõjutades programmeerijaid arendama midagi, mis ei täida oma oodatud eesmärki. Tulemusena mõjutavad ettevõtte andmed otseselt nende kliente ja läbi selle ka ettevõtte käivet [23].

4 Andmeaitade testimine

Testimine on tähtis osa tarkvarast ning tarkvaraarendusest. Eriti kriitiline on testimine andmeaidas, sest andmete otsesed kasutajad peavad usaldama neid andmeid, millele neil ligipääs on. Andmeaitade ja tavalise tarkvara testimise peamisteks erinevusteks on [24]:

- Tarkvara testimisel keskendutakse valdavalt koodile. Andmeaida testimine on suunatud andmetele ja informatsioonile. Andmete testimise kõige tähtsam osa on tunda andmeaidas olevaid andmeid ning seda, milliseid andmeid kasutajad pärivad [24].

- Andmeaida testimisel tuleb arvestada ka jõudluse ja produktiivsusega, sest andmete maht on suurem kui tavalistes süsteemides [24].

- Andmete testimisel keskendutakse sellele, et andmed oleks kasutaja jaoks õiged ja kasulikud. Andmete valideerimine on andmeaida testimisel põhiline eesmärk [24].

- Tavalisel tarkvaral on mitmeid kasutusjuhte, kuid nende arv on siiski limiteeritud. Andmeaidad on arendatud toetama ükskõik milliseid andmevaateid ning seetõttu on võimalikke kombinatsioone piiramatult ja neid ei saa täielikult testida [24].

- Kui tarkvara puhul teostatakse enamus testimisest enne juurutamist, siis andmeaida testimine sellega ei lõppe [24].

- Andmeaida projektidel ei ole lõppu, sest on võimatu hinnata tulevikus tekkivaid tingimusi ja nõudeid. Ka kõiki võimalike vigu on võimatu ennustada, sest allika süsteemidest tulevad andmed on pidevalt muutuvad. Seetõttu peab andmeaidas tegelema ka regressioonitestimisega [24].

Andmebaasides olevad vead on oma olemuselt teistsugused kui tavalises tarkvaras. Peamiselt põhjustavad probleeme tabelite omavaheline ühendamine ehk *JOIN*, duplikaatread tabelites ning puuduolevad väärtused. Seega tuleb ka arvestada, et [25]:

- Päringutel võib olla sisenditeks nii parameetrid kui ka alampäringud teistest tabelitest [25].

- Väljundiks on tabel, mitte üks kindel väärtus. Sellest tulenevalt võib testi tulemust olla raske kontrollida [25].

- Päringu töötlemine sõltub väga palju andmebaasist, selle struktuurist ning piirangutest [25].

- Ei kasutata binaarset loogikat, vaid kasutatakse ka tundmatuid ehk NULL väärtuseid, mis võib põhjustada ootamatuid tulemusi [25].

Suurimaks väljakutseks andmeaitade testimisel on tagada paindlik arhitektuur, sest andmeaitadel on ärinõuetest tingitud erinevad funktsionaalsused, arhitektuur ning ka ajalised piirangud. Andmeaidad koosnevad erinevatest sidusatest komponentidest ja igaüht neist tuleks testida eraldi [26].

Ärireeglite järgimine on äriteabe kõige tähtsam osa. Kogu seda teavet ja informatsiooni hoitakse andmeaitades, mis tähendab, et andmeaitade testimine tähendab ka nende kindlate ärireeglite testimist. Kuna selline testimine käsitsi on väga ajakulukas, siis tuleks kasutada automatiseerimist. Eriti tähtis on see siis, kui ärireegleid on palju ning need on keerukad. See loob aga keerulise probleemi testimistarkvara arendajatele, sest tööriist peaks samal ajal suutma töödelda nii ärireegleid kui ka andmeid [27]. Testimiseks kasutatavad päringud tuleks koostada näiteks nii, et eeldame tagastatava tulemusena 0 rida kirjeid. See ei tähenda, et tabelis ei ole andmeid, vaid et kindlatel tingimustel andmete agregeerimisel kirjeid ei leitud. Lisaks saab kontrollida ka väärtuste piire, andmetüüpe ja kuupäevi. Üldiselt tuleks testideks kasutatavad päringud koostada nii, et need kontrolliks tabelis nii õigeid kui ka valesid väärtuseid [25]. Näiteks saame luua testi, kus eeldame, et klientide tabelis ei ole ühtegi sellist kirjet, kus klient on inaktiivne ning omab negatiivset saldot. Eeldades 0 rida tulemust, on võimalik SQLis koostada testijuht (Joonis 8), mis vigaste kirjete leidmisel tagastab arvu kirjetest, kus klient on inaktiivne ning omab negatiivset saldot [28].

```
SELECT COUNT(*) FROM kliendi_tabel WHERE  viimane_tellimus < now()-90 AND
saldo < 0
```

Joonis 8. SQL testijuht inaktiivsete klientide leidmiseks, kelle saldo on negatiivne.

Süsteemi õigsust saab mõõta kasutades etteantud nõudeid ja tingimusi, mida defineeritakse läbi lõppkasutajate etteantud tingimuste. Kuna enamus tingimused tulenevad andmete kvaliteedist, siis keskendutakse andmeaida testimisel ühelt poolt ETL

protsesside testimisele (*back-end*) ning teiselt poolt aruandluse ja ärireeglite testimisele (*front-end*). ETL testimisega tehakse kindlaks, et andmeaita laetud andmed on vastavuses allikas olevate andmetega ning on õigesti transformeeritud. Ärireeglite testimisega veendutakse, et aruandlusesse jõudvad andmed on õigesti agregeeritud ning vastavad kindlatele omadustele [24].

4.1 Ärireeglite testimine

Andmete hindamiseks kasutatakse nii kvantitatiivset kui ka kvalitatiivseid hinnanguid, et aru saada, kas andmeid on mõistlik kasutada või mitte. Hinnatakse vigaste andmete tõsidust ning ulatust, arvestades andmete kasutamise eesmärgiga. Andmete valideerimiseks kontrollitakse, et andmed vastaksid etteantud loogikale. Selline valideerimine on vajalik, sest seda ei ole võimalik teha kasutades tavalisi andmebaasi piiranguid ja kitsendusi [9].

Testitavate andmete maht on suur ning see mõjutab ka andmeaida jõudlust ja produktiivsust. Lisaks lisavad keerukust andmeaidasisesed keerulised transformatsioonireeglid, mille käsitsi testimine on ajakulukas ning pole täpne. Automaatne andmete valideerimine tuleks teha nii, et see vähendaks kulusid ning hoiaks kokku aega. Selleks tuleks esmalt ära liigitada kõik testitavad tingimused. Igale tingimusele tuleb luua käivitatav SQL päring, mis vastab ärireeglitele. Päringud tuleks kirjutada nii, et tulemusest on loetav, kas test käis läbi või ebaõnnestus (tulemus on näiteks „*PASS*“ või „*FAIL*“) [29].

Ärireeglite testimisel on vajalikud teadmised selle kindla valdkonna kohta. Neid teadmisi kasutatakse, et panna paika testitavad reeglid ja tingimused. Ärireeglite testimisel on üheks suurimaks eeliseks nende korduvkasutatavus. Andmeaitades on suure tõenäosusega mitmeid tabeleid, millele saab rakendada sama reeglit, muutes seda vastavalt tabeli omadustele [15].

4.2 ETL testimine

ETL protsess võimaldab andmeaita laadida allikatest andmeid, muuta olemasolevaid andmeid ning muuta andmete struktuuri ja mudelit. ETL transformatsioonid võivad sisaldada üks-ühele, üks-mitmele või mitu-mitmele tüüpi seoseid. Selliste keerukate

transformatsioonide rakendamine võib põhjustada vigu, mis seavad ohtu andmeaidas olevad andmed ning mõjutavad lõppkasutajat. Sellest tulenevalt peab ETL protsesse testima, et veenduda andmeaita laetud andmete õigsuses. Suurte andmemahtude laadimine, nende transformeerimine ja sisestamine andmeaita välistab andmete käsitsi kontrollimise. Lisaks tuleks arvestada, et ETL protsessi testimine ei ole ühekordne, sest andmeaitade arendus on pidev ning andmeid laetakse pidevalt juurde [30].

ETL testimise automatiseerimine on vajalik, et vähendada ressursside kulu ning võimaldada testide korduvkasutust. Selliste testide loomisel tuleb arvestada üldiste andmeaitade omadustega, milleks on täielikkus, terviklikkus ja süntaksile vastavus [31].

4.2.1 Täielikkus

Täielikkus tagab selle, et kõik kirjed, mis peaksid olema transformeeritud ning andmeaita laetud, eksisteerivad ka päriselt andmeaidas. Laadimises tekkinud vigade tõttu võivad mõned kirjed olla andmeaidas mitmekordselt või olla puudulikud. Täielikkuse mõõtmiseks võrreldakse kirjete arvu allika ning andmeaida tabelite vahel, arvestades tabelite omavaheliste seostega (Tabel 1). Näiteks üks-ühele seoste puhul peaks vastavate tingimuste korral andmeaidas olema sama arv kirjeid, mis allikas. Tabelites, kus hoiustatakse andmete ajalugu, tuleks kasutada eristuvate ehk *DISTINCT* kirjete arvu. Ka eristuvate kirjete puhul peab arvestama tabelite ning kirjete vaheliste seostega [31].

Tabel 1. *Transforming Single Source Table to Single Target Table*. Allika ja andmeaida tabelite omavahelised seosed [31].

Allika tabel::veeru nimi	**Andmeaida tabel::veeru nimi**
Address::Address_Key	Location::location_id
Address::Address_string	Location::address_1
Address::City	Location::city
Address::Postal_code	Location::zip

4.2.2 Terviklikkus

Terviklikkus tagab selle, et andmeaita laetud andmete sisu on vastavuses ETL protsessi käigus tehtud transformatsiooni reeglitele ning allika tabelite sisule. Selleks võrreldakse kindla välja väärtust või agregeeritud väärtust, arvestades ka võimaliku veapiiriga (Joonis 9) [31].

```
ABS(allika_väärtus - andmeaida_väärtus) < veapiir
```

Joonis 9. Tulemuste võrdlemine veapiiriga SQLis.

Terviklikkuse tagamiseks testitakse ka allikapoolseid andmebaasi piiranguid. Näiteks, kui allika veerul on piirang *NOT NULL*, siis ei tohiks andmeaita olla sisestatud kirjeid, mille väärtuseks on *NULL* [31]. Sellest võib tuletada SQL testi, mis leiaks selliste kirjete arvu, kus otsitav väärtus on *NULL* (Joonis 10), arvestades, et oodatav tulemus on 0.

```
SELECT COUNT(*) FROM tabel WHERE otsitav_väärtus IS NULL
```

Joonis 10. SQL päring NULL kirjete leidmiseks.

Numbriliste väärtuste terviklikkust on võimalik testida võrreldes minimaalset ja maksimaalset väärtust allikas vastava väärtusega andmeaidas (Joonis 11) või arvutada ning võrrelda väärtuste keskmisi tulemusi (Joonis 12) [31].

```
ABS(MIN/MAX(allika_väärtus) - MIN/MAX(andmeaida_väärtus)) < veapiir
```

Joonis 11. Miinimum või maksimum väärtuste võrdlemine SQLis.

```
ABS(AVG(allika_väärtus) - AVG(andmeaida_väärtus)) < veapiir
```

Joonis 12. Keskmiste tulemuste võrdlemine SQLis.

4.2.3 Süntaksile vastavus

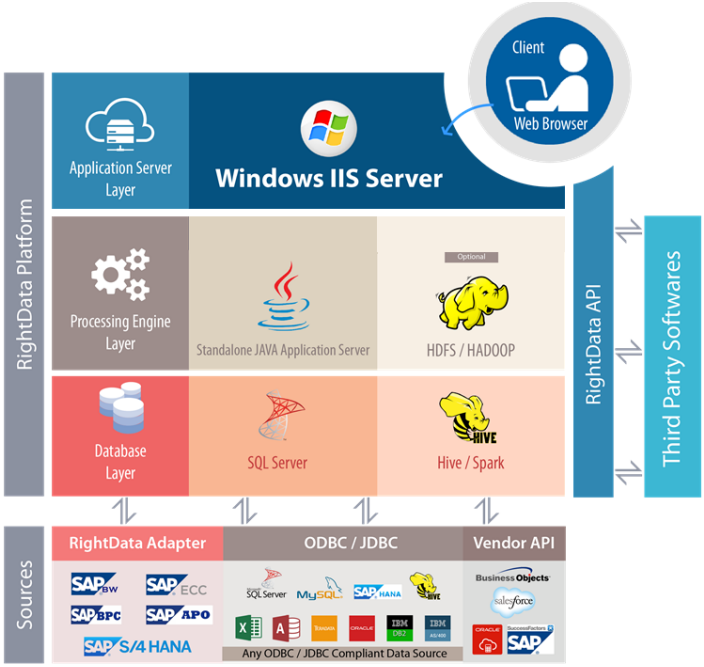
Süntaksile vastavus tagab selle, et andmeaidas oleva väärtuste süntaks vastaks allikas olevale väärtusele. Selleks on võimalik võrrelda andmetüüpe. Näiteks tuleks kontrollida, et allika numbriline väärtus oleks andmeaidas numbriline, mitte tekstiline. Testida saab ka andmebaaside poolt defineeritud andmetüüpide pikkusi. Selleks tuleb võrrelda allika andmete keskmisi pikkusi või suurusi andmeaidas olevate väärtustega. Oluline on ka arvestada andmebaasi tehniliste piirangutega. Näiteks on MS SQL Serveris lubatud VARCHAR suurus 8000, aga Oracle andmebaasis 4000 [31].

4.3 Testimiseks kasutatavad tööriistad

Olemasolevad andmekvaliteedi tööriistad on arendatud väga kindlate valdkondade või väga üldiste omaduste testimiseks. Tihti on sellised tööriistad arendatud ka mingi kindla andmeaida testimiseks ning ei võimalda tööriista kasutada teistes andmeaitades [30]. On arendatud ja turul vabalt kättesaadavaks tehtud ka tarkvarasid, mis on mõeldud kasutamiseks kui universaalsed andmeaitade testimise vahendid. Tuntuimad nendest on RightData, QuerySurge ning DataGaps.

4.3.1 RightData

RightData pakub tööriista andmete testimiseks ja võrdlemiseks, võimaldades seda teha erinevate andmemudelite ja tüüpidega. Kasutajaliideses on võimalik erinevatest allikatest väljad kokku sobitada, nendes olevaid andmeid omavahel võrrelda ning rakendada neile kindlate reeglite testimist [32]. RightDatat on võimalik paigaldada Windows Server 2012 (või uuemasse) keskkonda ning vajab eraldi Microsoft SQL Serveri andmebaasi (2012 või uuem). Andmebaas võib olla paigaldatud ka samasse masinasse, kuhu on paigaldatud RightData rakendus. Tööriistale on võimalik ligi pääseda läbi veebibrauseri ning on seega ka kasutatav meeskonnapõhiselt (Joonis 13) [33].



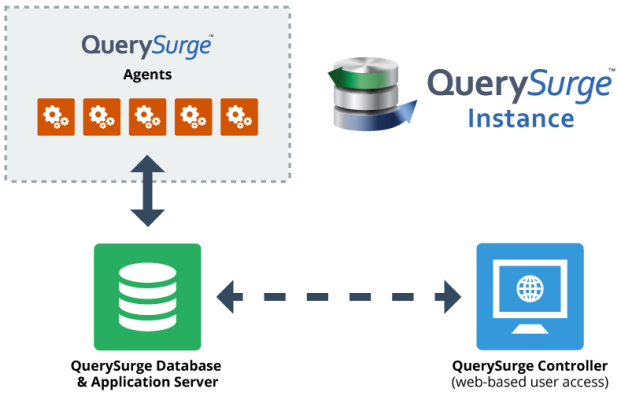
Joonis 13. RightData instantsi arhitektuur [33].

4.3.2 QuerySurge

QuerySurge on tööriist andmete optimiseerimiseks ja vigade tuvastamiseks. See on mõeldud andmete testimise automatiseerimiseks andmeaitades, suurandmetes ning äriteabe aruandluses. QuerySurge võimaldab teste automaatselt käivitada ning tulemustest e-maili teel aru anda. Teste koostatakse läbi kasutajaliidese, kus on võimalik lihtsasti vajalikke komponente kokku lohistada ning selle tulemusena SQL genereerida [34]. Tööriista on võimalik paigaldada nii Windows kui Linux keskkonda ning sellele on võimalik ligi pääseda läbi veebibrauseri. Paigaldus koosneb kolmest osast: agendid,

rakenduse server ja andmebaasi server (Joonis 14). Tööriista on võimalik paigaldada kolmel erineval moel [35]:

- Individuaalsele kasutajale – kogu instants paigaldatakse ühte arvutisse ning tööriistale on võimalik ligi pääseda ainult läbi selle kindla arvuti. Informatsiooni erinevate instantsite vahel on võimalik jagada, kui arvutid on ühendatud samasse võrku [35].
- Väiksele meeskonnale – tööriista instants (andmebaas ja rakendus) paigaldatakse Windows või Linux serverisse ning agendid paigaldatakse eraldi masinatesse. Läbi agentide on võimalik kasutajatel tööriistaga suhelda, kasutades veebibrauserit [35].
- Suurele meeskonnale – andmebaasi server, rakenduse server ja agendid paigaldatakse eraldi masinatesse. Selle tulemusena on võimalik tööriista rakendada ka erinevates keskkondades, võimaldades instantsite vahel teste jagada [35].

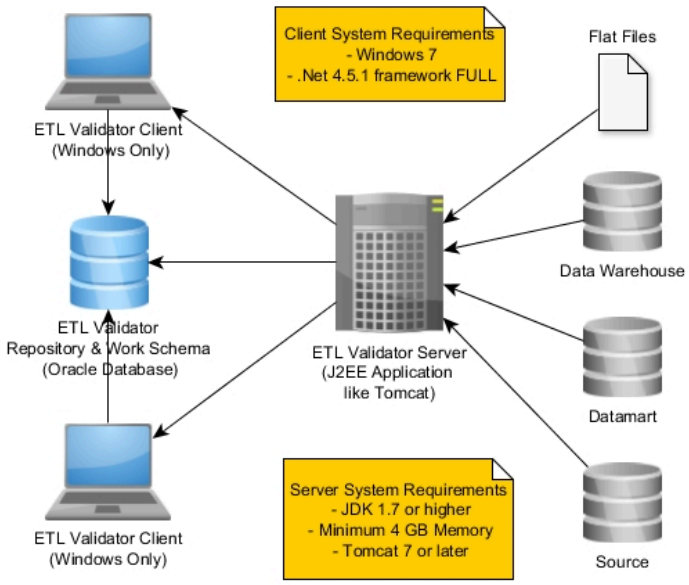


Joonis 14. QuerySurge instantsi arhitektuur [35].

4.3.3 DataGaps ETL Validator

ETL Validator on andmete testimise automatiseerimiseks mõeldud platvorm, mille rõhk on erinevate andmekogumike võrdlemisel. Tööriist kasutab sisseehitatud ETL mootorit, mis võimaldab allikatest korraga pärida miljoneid ridu ning neid omavahel võrrelda. Teste koostatakse kasutades visuaalset kasutajaliidest, et kasutaja ei peaks käsitsi

päringuid looma. Läbi kasutajaliidese on võimalik koostada ka erinevaid ärireegleid, kasutades andmete profileerimist. Profileerimiseks kasutatakse agregeeritud andmeid, milleks on reaarvud, summad ja eristuvad väärtused [36]. Tööriista on võimalik kasutada nii individuaalselt kui ka meeskonnapõhiselt. Individuaalses instantsis saab tööriista kasutada läbi Windows 7 rakenduse ning see vajab eraldi PostgreSQL serverit. Nii rakendust kui ka serverit on võimalik paigaldada samasse masinasse. Meeskonnas kasutamiseks tuleb paigaldada ETL Validatori server, luua eraldi Oracle andmebaas ning paigaldada iga kasutaja Windows 7 keskkonda ETL Validatori rakendus (Joonis 15) [37].



Joonis 15. Datagaps ETL Validator instantsi arhitektuur [37].

4.3.4 Tööriistade võrdlus

Tööriistade võrdlemisel arvestatakse töös analüüsitava suurettevõtte poolt tulenevate vajadustega, tööriistade arhitektuuri ja paigaldusega ning kasutajamugavusega. Ettevõttes toimuks tööriista kasutus meeskonnapõhiselt ning see peaks toetama kõiki ettevõttes kasutatavaid andmebaase. Lisaks peaks tööriista olema võimalik paigaldada Linuxi serverisse ning tööriista haldus ja paigaldus peaks olema lihtne. Tööriistu võrreldes saab analüüsitavas ettevõttes kasutuseks välistada Datagapsi poolt pakutava tööriista, sest ei võimalda veebipõhist ligipääsu ja nõuab selle asemel Windows 7 rakenduse kasutust (Tabel 2). RightData ei võimalda paigaldust Linux serverisse ning puuduvad vajalikud kasutajamugavused. QuerySurge on võrreldatavatest tööriistadest parim valik, kuid kui arvestada kasutajamugavustega (testide korduvkasutatavus ja hulgaline testide genereerimine) ning nende poolt välja toodud avaliku hinnaga (vastavalt ettevõtte vajadustele ~€50000/aastas [38]), siis välistatakse selle kasutus analüüsitavas ettevõttes.

Tabel 2. Andmete testimiseks kasutatavate tööriistade võrdlus lähtudes analüüsitava ettevõtte vajadustest.

	RightData	**QuerySurge**	**Datagaps**
Arhitektuurilised nõuded	Rakenduse server (Windows Server); Andmebaas (MS SQL);	Rakenduse server (Windows/Linux); Andmebaas; Agentide masinad;	Rakenduse server (Windows/Linux); Andmebaas (Oracle); Windows 7 arvutid;
Ligipääs	Veebipõhine	Veebipõhine	Windows rakendus
Andmehulkade võrdlus	Olemas	Olemas	Olemas
REST API liides	Olemas	Olemas	Puudub
Plaanipõhine testide käivitamine	Olemas	Olemas	Olemas
Testide korduvkasutatavus	Puudub	Puudub	Puudub
Hulgaline testide loomine	Puudub	Puudub	Puudub
Reaalajas andmete monitoorimine	Olemas	Olemas	Puudub
Tulemuste raporteerimine e-mailiga	Olemas	Olemas	Olemas
Statistikapõhine vigade ennustamine	Puudub	Puudub	Puudub
Testide loomine kasutajaliideses	Olemas	Olemas	Olemas

5 Loodud testimisvahend

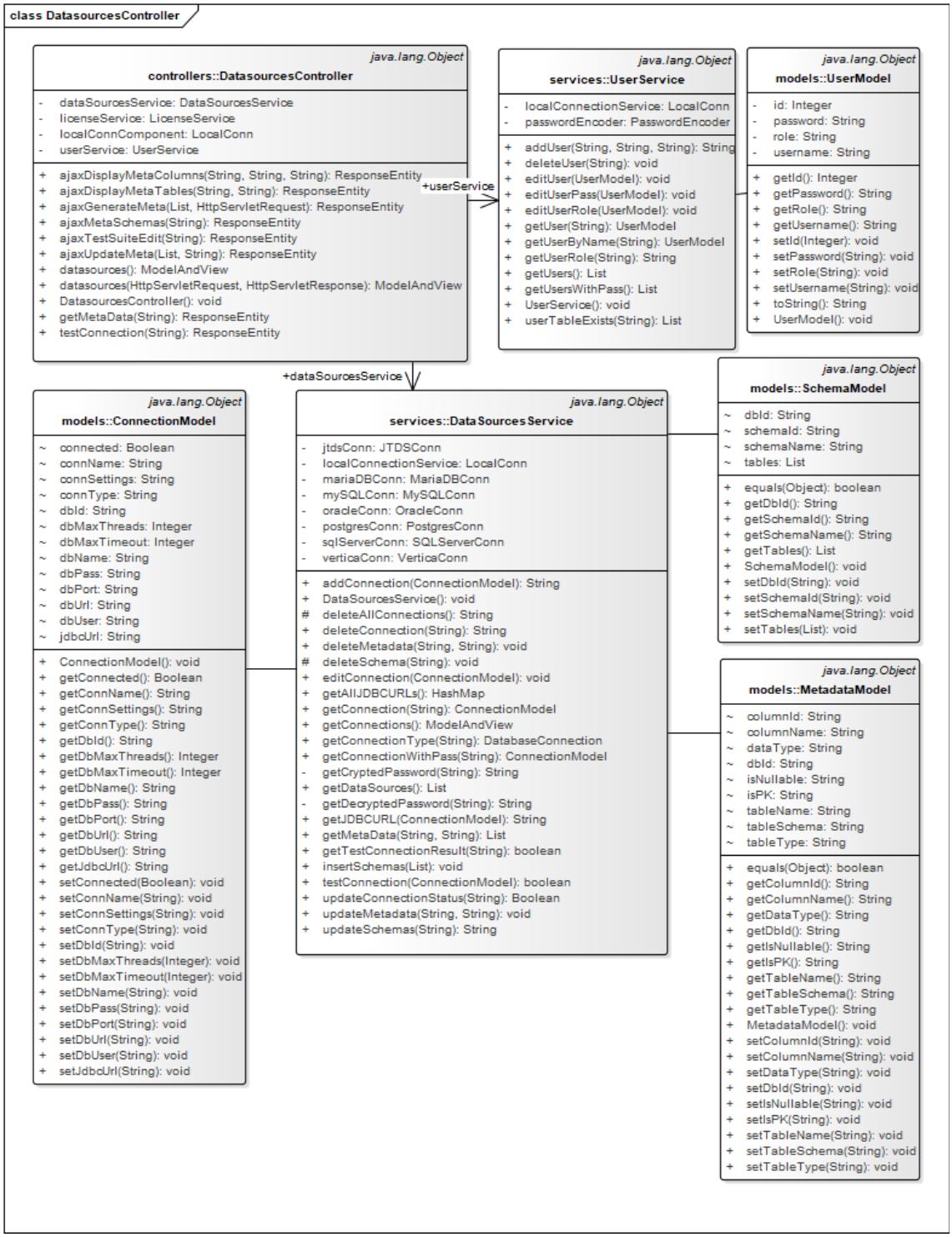
Testimisvahendi arendamisel on kasutatud teoreetilisi teadmisi töö esimesest poolest. Omandatud informatsiooni kasutatakse, et luua sobivate funktsioonide ja omadustega tööriist efektiivseks andmete testimiseks. Arendus toimus kasutades IntelliJ Ultimate tarkvara, et kirjutada funktsionaalsust programmeerimiskeeles Java. Tehnilistes küsimustes ning osaliselt ka koodi kirjutamises oli autorit abistamas kogenud Java arendaja.

5.1 Arendus eesmärk

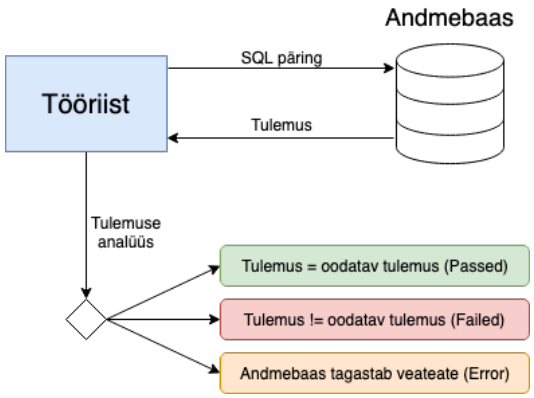
Arenduse eesmärgiks on luua veebirakendus, mis võimaldaks automatiseerida suurettevõtte andmeaida testimist. Arendatav tööriist võetakse suurettevõttes kasutusele, et igapäevaselt monitoorida ja testida ettevõtte andmeaida andmeid ja nende kvaliteeti. Tööriist on mõeldud tiimisiseseks kasutuseks ning võimaldab igal tiimiliikmel individuaalselt oma arendusi või hallatavaid andmedomeene testida. Testimisvahend arendatakse veebirakendusena, et hõlbustada ligipääsu, võimaldada tööriista lihtsasti hallata ja vähendada kasutajate riist- või tarkvarast tingitud probleeme tööriista kasutamisel.

5.2 Arhitektuur

Tööriist on ehitatud Spring Boot raamistikul kasutades MVC arhitektuurimustrit. Kokku on kasutuses 21 mudelit, 11 kontrollerit, 11 vaadet ning 13 teenust. MVC mustrit järgides kasutavad kontrollerid erinevaid teenuseid, mis omakorda kasutavad andmeid läbi mudelite. Näiteks kasutab *DatasourcesController* teenuseid *DataSourcesService* ning *UserService*, et hallata andmebaaside ühendusi (Joonis 16). Andmebaasi ühendustega seotud klasse on 9, millest 7 on väliste ühenduste loomiseks. Väliseid ühendusi teostavad klassid kasutavad ühist liidest *DatabaseConnection*. Tööriista *front-endis* on kasutusel *jQuery* ning *Bootstrap* raamistikud. SQL lugemiseks ja kirjutamiseks kasutatakse JavaScriptil põhinevat *CodeMirror* tekstiredaktorit ning erinevate graafikute kuvamiseks *ChartJS* raamistikku. PDF aruandluse genereerimiseks kasutatakse *PDFbox* raamistikku. Andmete testimine toimub käivitades SQL päringuid ning võrreldes andmebaasi poolt tagastatud väärtust testijuhu oodatava tulemusega (Joonis 17).



Joonis 16. Kontrollerite, teenuste ning mudelite vahelised seosed DatasourcesControlleris.



Joonis 17. Testi käivitamine arendatud tööriistas.

5.2.1 Ühiktestimine

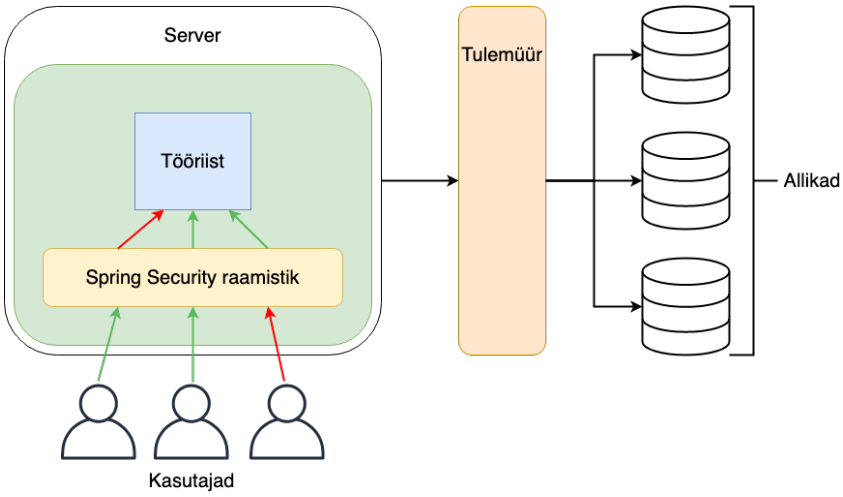
Ühiktestimist teostatakse teenuste klassidel, mis on seotud tööriista põhilise funktsionaalsusega. Selliseid teenuse klasse on 10 ning kokku testitakse nendes 100 meetodit (Joonis 18). Ühiktestimisel rakendatakse *Mockito* raamistikku, et kasutada testimisel teisi vajaminevaid teenuseid ning luua testimiseks kõlblikud olukorrad. Testimine toimub rakenduses IntelliJ Ultimate kasutades *JUnit* ühiktestimise raamistikku. Ühiktestimise eesmärgiks oli testida kõiki klassides kasutatavaid meetodeid ning katta koodiread ära vähemalt 90% ulatuses, mida ka testimise tulemusena saavutati.

Element	Class, %	Method, %	Line, %
VersionControlService	100% (2/2)	100% (11/11)	100% (79/79)
LicenseService	100% (1/1)	100% (5/5)	100% (70/70)
CookieService	100% (1/1)	100% (2/2)	100% (16/16)
UserService	100% (1/1)	100% (12/12)	97% (151/155)
SettingsService	100% (1/1)	100% (5/5)	97% (48/49)
DynamicTestService	100% (1/1)	100% (12/12)	95% (219/229)
TestSuitesService	100% (1/1)	100% (13/13)	94% (224/237)
TestService	100% (1/1)	100% (14/14)	93% (277/296)
DataSourcesService	100% (1/1)	100% (23/23)	93% (274/293)
LogReadService	100% (1/1)	100% (3/3)	92% (38/41)

Joonis 18. Tööriista teenuste klasside ühiktestimisega kaetud osa.

5.2.2 Turvalisus

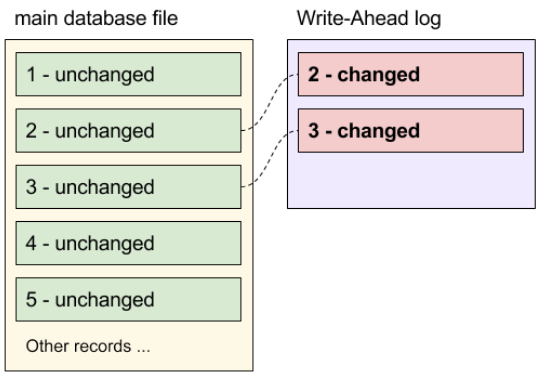
Turvalisuse tagamiseks on tööriista funktsionaalsus kindlustatud ilma internetiühenduseta ehk kõik kasutatavad raamistikud on lokaalselt saadaval. Rakendus on *on-premise* ehk paikneb ettevõtte sisevõrgus olevas serveris, millele on ligipääs määratud kasutajatel (Joonis 19). Lisaks on rakendatud *Spring Security* raamistikku, mis võimaldab tööriista kasutajate autentimist. Kasutajate paroolid on krüpteeritud kasutades *PasswordEncoder* klassi ning samuti on krüpteeritud kasutatavate andmebaasiühenduste delikaatne informatsioon kasutades *Jasypt* (Java Simplified Encryption) krüpteeringut.



Joonis 19. Tööriista loogiline asukoht ettevõtte sisevõrgus.

5.2.3 Andmebaasi arhitektuur

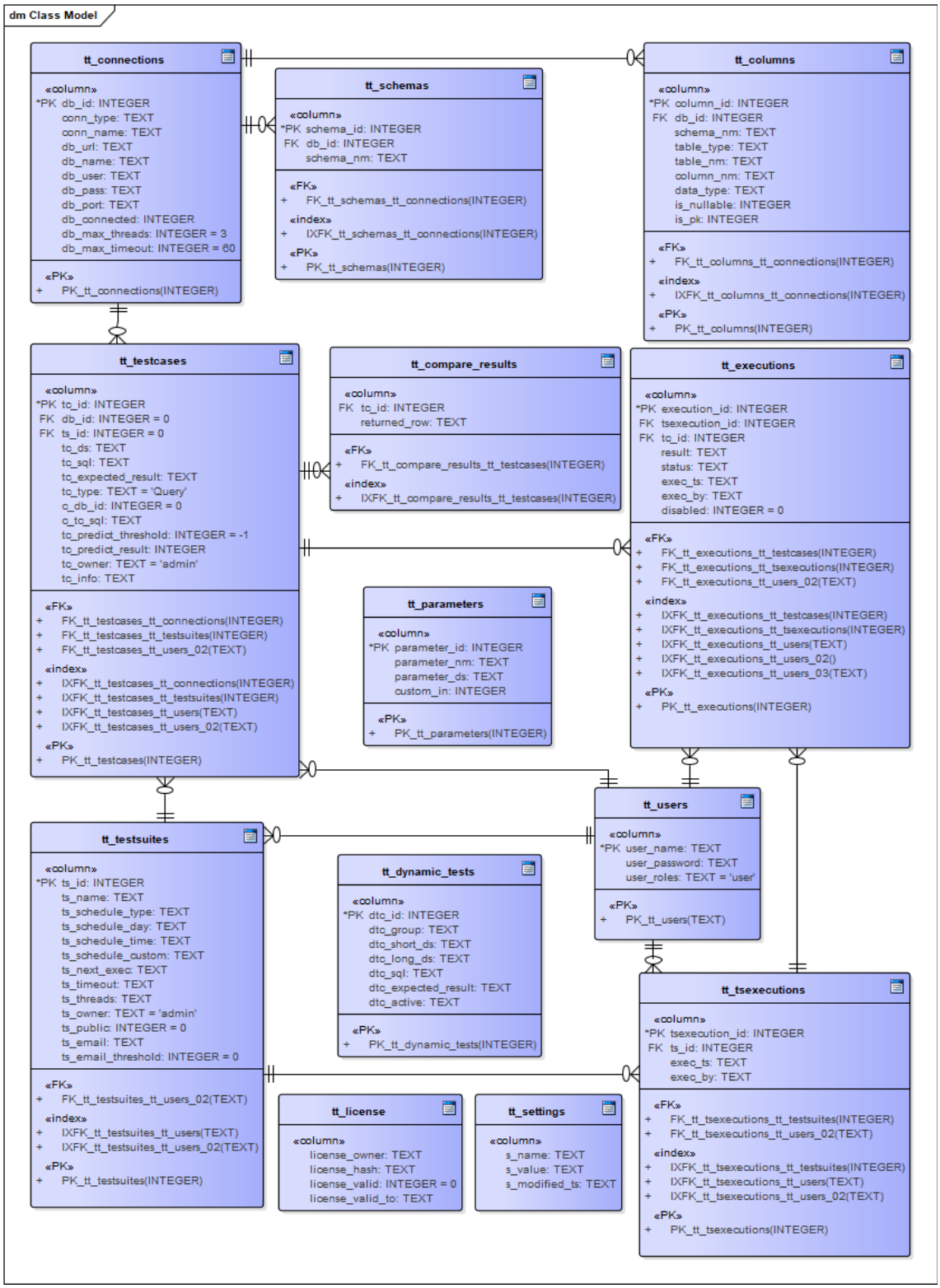
Tööriista lokaalse andmebaasina kasutatakse SQLite failipõhist andmebaasi. Andmebaas on seadistatud kui *Write-Ahead Log*, mis võimaldab mitmel kasutajal samaaegselt tabelitesse kirjutada ja nendest lugeda (Joonis 20). WAL-i põhiliseks piiranguks on see, et seda saab kasutada korraga ainult ühes süsteemis [39]. Arendatava tööriista puhul on see aktsepteeritav, sest ei kasutata väliseid protsesse, mis andmebaasile ligipääsu vajaks. SQLite on antud kontekstis andmebaasina parim valik, sest seda ei ole vaja initsialiseerida eraldi rakendusena eraldi serverisse ning kuna see asub tööriista süsteemifailidega samas kataloogis ja serveris, siis on sealt võimalik kiiresti andmeid lugeda ja kirjutada.



Joonis 20. SQLite Write-Ahead Log tööpõhimõte [40].

Kokku on tööriistas kasutusel 13 süsteemitabelit, millest kõik omavad unikaalset primaarvõtit (Joonis 21). Kasutajatega seotud tabeli primaarvõti on tekstiline, mis on ühtlasi ka unikaalne kasutajatunnus tööriistas autentimiseks. Teiste tabelite võtmed on automaatselt genereeritud numbrid, mis on loodud kasutades tabeli funktsiooni

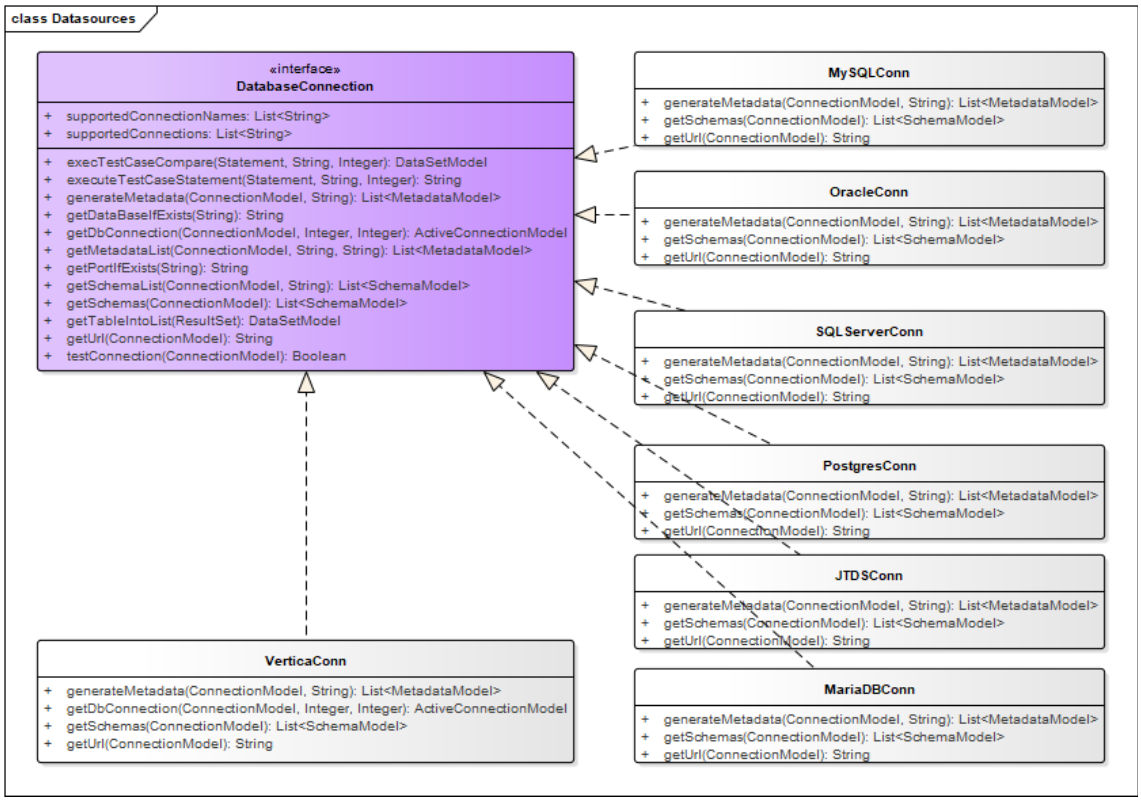
AUTOINCREMENT. Tabelite vahel on kasutatud üks-mitmele (1-0..*) seoseid ning kokku on kasutuses 12 välisvõtit.



Joonis 21. Tööriista lokaalse andmebaasi arhitektuur.

5.3 Andmebaasi ühendused

Arendatav tööriist võimaldab kasutajal lisada erinevaid andmebaase, kuhu tööriist ühenduda saab. Ühendusteks kasutatakse JDBC draivereid ning kokku on võimalik kasutada kuute erinevat tüüpi andmebaase: Vertica, PostgreSQL, Oracle, Microsoft SQL Server, MySQL ja MariaDB (Joonis 22). Toetatud on eelmainitud andmebaasid, sest katavad ära suure osa analüüsitava suurettevõtte andmeaida allikatest. Lisatud andmebaasi ehk allika ühenduse informatsiooni hoiustatakse tööriista lokaalses andmebaasis, hoiustades tüüpi, nimetust, URL-i, porti, kasutajanime ning parooli krüpteeritud kujul. Andmebaasi lisamisel kontrollitakse, kas ühendus on kättesaadav (Joonis 23) ning tulemus kuvatakse kasutajale (Joonis 24). Tööriist kontrollib kõiki lisatud ühendusi automaatselt iga 60 minuti tagant ning uuendab staatust ka kasutajaliideses.

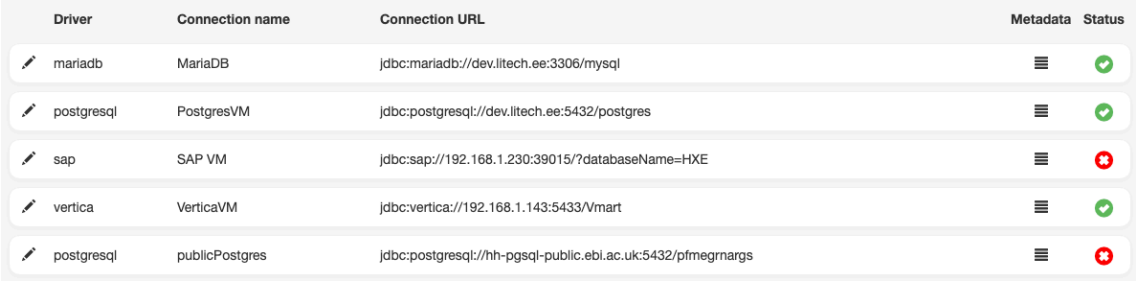


Joonis 22. Tööriistaga liidestatavate andmebaaside klassidiagramm.

```
Boolean connectionAvailable = false;
Properties properties = new Properties();
properties.put("user" , connectionModel.getDbUser());
properties.put("password" , connectionModel.getDbPass());

try (Connection conn = DriverManager.getConnection(getUrl(connectionModel),
properties)) {
    connectionAvailable = true;
} catch (SQLException e) {
    LOG.warn("Connection failed: " + e.getMessage());
}
return connectionAvailable;
```

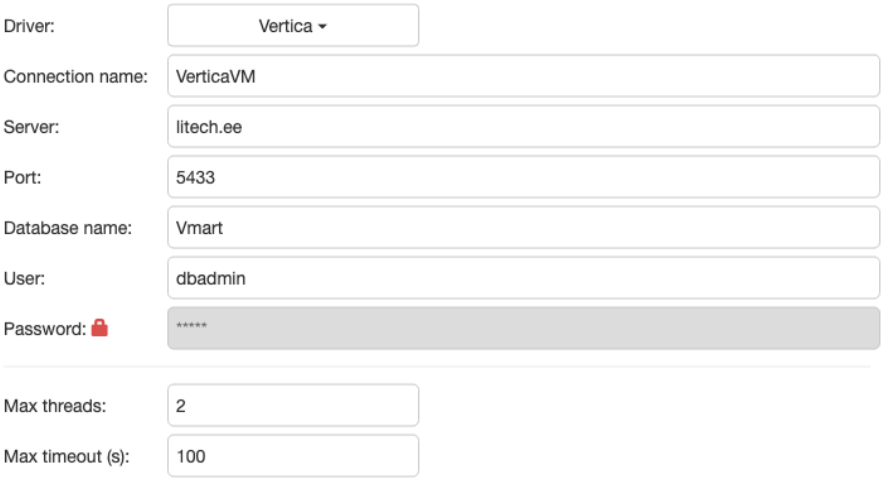
Joonis 23. Andmebaasi ühenduse testimine.



	Driver	Connection name	Connection URL	Metadata	Status
✎	mariadb	MariaDB	jdbc:mariadb://dev.litech.ee:3306/mysql	≣	✓
✎	postgresql	PostgresVM	jdbc:postgresql://dev.litech.ee:5432/postgres	≣	✓
✎	sap	SAP VM	jdbc:sap://192.168.1.230:39015/?databaseName=HXE	≣	✗
✎	vertica	VerticaVM	jdbc:vertica://192.168.1.143:5433/Vmart	≣	✓
✎	postgresql	publicPostgres	jdbc:postgresql://hh-pgsql-public.ebi.ac.uk:5432/pfmegrnargs	≣	✗

Joonis 24. Kasutajatele kuvatavad andmebaasi ühendused ning nende staatused.

Igale lisatud andmebaasile on võimalik määrata maksimaalne *timeout* ehk mitu sekundit üks päring selle baasi vastu aktiivne võib olla, ning maksimaalne *threshold* ehk mitu päringut võib korraga ühe kasutaja poolt selle baasi vastu käivitatud olla (Joonis 25). Sellised seadistused on vajalikud, sest iga allikas on erinev ning tuleb arvestada nende jõudlusega. Kui päring käib üle ettemääratud aja, siis pannakse see andmebaasi poolt kinni ning tagastatakse veateade. Ajalimiit ja paralleelsete päringute piirang seadistatakse ühenduse loomisel ning on seadistatud eraldi iga andmebaasi klassi kohta, sest on igas andmebaasimootoris erinevalt konfigureeritav (Joonis 26).



Joonis 25. Vertica andmebaasi lisamine tööriistas kasutades kasutajaliidest.

```
ActiveConnectionModel model = new ActiveConnectionModel();
connection.setConnSettings("SET SESSION RUNTIMECAP '" + timeout + "
seconds';");
DataSourcePool pool = new DataSourcePool(connection, maxThreads,
getUrl(connection));
model.setDbId(connection.getDbId());
model.setPool(pool);
model.setTimeout(timeout);
return model;
```

Joonis 26. Vertica ühenduse loomisel ajalimiidi ning paralleelsete päringute piirangu seadistamine.

5.3.1 Metaandmete kogumine

Uue ühenduse loomisel päritakse andmebaasist informatsiooni seal eksisteerivate *schemade* kohta. Pärast ühenduse loomist saab kasutaja *schemade* nimede põhjal genereerida metaandmeid andmebaasi veergude kohta, mida omakorda kasutatakse testide automaatseks genereerimiseks. Kogutakse *schema* nime, tabeli nime, veeru nime, andmetüüpi, primaarvõtme indikaatorit ning *null* väärtuse lubamist. Metaandmeid uuendatakse automaatselt kord tunnis, lisades ja kustutades välju, mis on allikas muutunud. Infot nende andmete kohta hoiustatakse lokaalse andmebaasi tabelis *tt_columns* (Joonis 27).

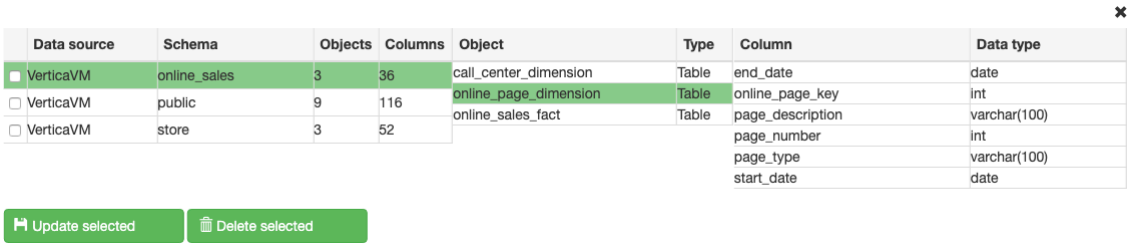
column_id	db_id	schema_nm	table_type	table_nm	column_nm	data_type	is_nullable	is_pk
97848	18	rnacen	Table	xref_p19_deleted	version_i	integer	f	f
97849	18	rnacen	Table	xref_p19_deleted	upi	character varying	f	f
97850	18	rnacen	Table	xref_p19_deleted	last	integer	f	f
97851	18	rnacen	Table	xref_p19_deleted	created	integer	f	f
97852	18	rnacen	Table	xref_p19_deleted	dbid	smallint	f	f
97853	18	rnacen	Table	xref_p18_not_deleted	id	bigint	t	f
97854	18	rnacen	Table	xref_p18_not_deleted	taxid	bigint	t	f
97855	18	rnacen	Table	xref_p18_not_deleted	version	integer	t	f
97856	18	rnacen	Table	xref_p18_not_deleted	ac	character varying	f	f
97857	18	rnacen	Table	xref_p18_not_deleted	userstamp	character varying	f	f
97858	18	rnacen	Table	xref_p18_not_deleted	timestamp	timestamp without time zone	f	f
97859	18	rnacen	Table	xref_p18_not_deleted	deleted	character	f	f
97860	18	rnacen	Table	xref_p18_not_deleted	version_i	integer	f	f
97861	18	rnacen	Table	xref_p18_not_deleted	upi	character varying	f	f

Joonis 27. Metaandmed tööriista lokaalses SQLite andmebaasi tabelis tt_columns.

Metaandmete kogumiseks on iga andmebaasi kohta kirjeldatud eraldi päring (Joonis 28), mis kutsutakse *DatabaseConnection* liidese poolt välja vastavalt andmebaasi tüübile. Päringus asendatakse schema parameetrid ning laetakse uued andmed lokaalsesse andmebaasi. Metaandmeid on võimalik kasutajal vaadata, et nende olemasolus veenduda ning vastavalt vajadusele käsitsi uuendada või kustutada (Joonis 29).

```
SELECT 'Table' AS table_type, a.table_name, a.column_name, a.data_type,
a.is_nullable, CASE WHEN b.constraint_name IS NOT NULL THEN true ELSE false
END AS is_pk FROM v_catalog.columns a LEFT JOIN v_catalog.primary_keys b ON
a.table_schema = b.table_schema and a.table_name = b.table_name AND
a.column_name = b.column_name WHERE a.table_schema = 'store' UNION ALL SELECT
'View' AS table_type, table_name, column_name, data_type, true, false from
v_catalog.view_columns  WHERE table_schema = 'store'
```

Joonis 28. Vertica andmebaasist metaandmete kogumiseks käivitatav SQL päring.



Joonis 29. Metaandmete kuvamine, uuendamine ning kustutamine kasutajaliideses.

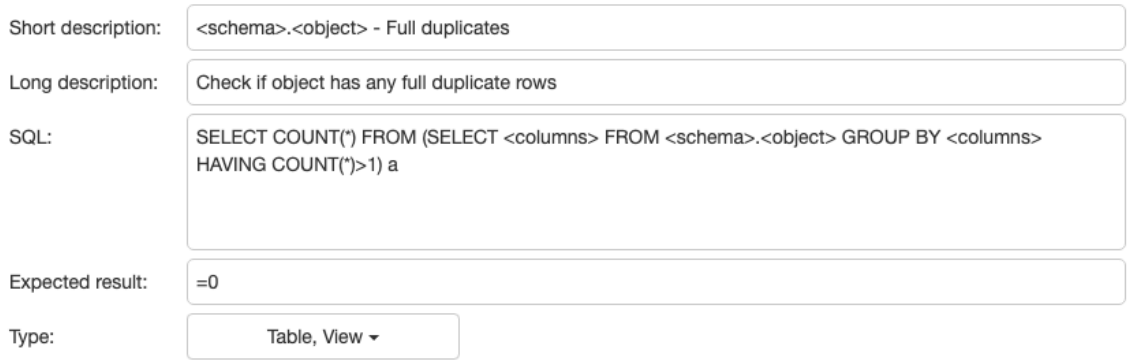
5.4 Dünaamiliste testid

Dünaamilised testid on korduvkasutatavad testid, kus vastajalt kasutaja sisendile asendatakse SQL-s olevaid parameetreid. Dünaamilised testid võivad olla nii objektipõhised, kasutades parameetreid nagu schema nimi (<*schema*>), tabeli nimi (<*object*>) ja veeru nimed (<*columns*>), või tavalised, kasutades kasutaja poolt loodud parameetreid. Dünaamiliste testide loomisel sisestatakse kasutaja poolt SQL koos asendatavate parameetritega, oodatav tulemus, testi kirjeldus ning testi tüüp, mis omakorda sisestatakse lokaalsesse andmebaasi tabelisse *tt_dynamic_tests* (Joonis 30). Testi tüüpe on kolm: *table*, *view* ning *dynamic*. Tüüp määrab ära, millistel juhtudel dünaamilist testi uue testijuhu genereerimisel kasutada on võimalik. *Table* tüüpi teste saab kasutada ainult tabeli objektide testimiseks, *view* tüüpi teste vaadete objektide testimiseks ning *dynamic* tüüpi teste kasutatakse tavaliste dünaamiliste testide loomiseks. Üks dünaamiline test võib olla mitut tüüpi. Näiteks luuakse duplikaatridade kontrollimise test, mis sobib nii tabelite kui ka vaadete testimiseks (Joonis 31).

```
String sql = "INSERT INTO tt_dynamic_tests (dtc_group, dtc_short_ds,
dtc_long_ds, dtc_sql, dtc_expected_result, dtc_active)" +
        " VALUES (?, ?, ?, ?, ?, ?)";
Connection conn = localConnectionService.getConnection();
try {
    if (conn != null) {
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, test.getDtc_group());
        ps.setString(2, test.getDtc_short_ds());
        ps.setString(3, test.getDtc_long_ds());
        ps.setString(4, test.getDtc_sql());
        ps.setString(5, test.getDtc_expected_result());
        ps.setString(6, "true");

        ps.executeUpdate();
        ps.close();
    }
} catch (SQLException e) {
    return null;
}
```

Joonis 30. Dünaamilise testi sisestamine tööriista lokaalse SQLite andmebaasi tabelisse tt_dynamic_tests.



Joonis 31. Duplikaatide kontrollimiseks mõeldud dünaamilise testi loomine vaadetele ja tabelitele.

5.4.1 Loodud dünaamilised testid

Tööriista kasutamiseks analüüsitavas ettevõttes on loodud viis erinevat dünaamilist testi:

- Duplikaatridade kontroll – eeldame, et tabelis on samasuguseid ridu 0. Selleks tuleks päringus valida kõik tabelis olevad veerud ning need grupeerida, otsides selliseid kirjeid, kus ridade arv on suurem kui 1 (Joonis 32).

```
SELECT COUNT(*) FROM (SELECT <columns> FROM <schema>.<object> GROUP
BY <columns> HAVING COUNT(*)>1) a
```

Joonis 32. SQL dünaamiline test duplikaatridade kontrolliks.

- *NULL* veergude kontroll – eeldame, et tabelis ei ole selliseid veerge, kus pole sees ühtegi väärtust (ehk kõikides ridades on selles veerus *NULL* väärtus). Selleks

loeme iga tabelis oleva veeru kohta kokku kõik sellised read, kus väärtuseks on midagi muud kui NULL. Kui kokku loetud veergude summa on 0, siis tähendab, et kõikides ridades on selle veeru väärtus NULL. Sellisel juhul tagastame veeru nime, et testi tulemusena oleks lihtsasti aru saada vea olemusest (Joonis 33).

```
SELECT '' <forEachColumn> || CASE WHEN (SUM(CASE WHEN "<column_name>" IS NULL
THEN 0 ELSE 1 END)=0) THEN '<column_name>;' ELSE '' END </forEachColumn> FROM
<schema>.<object>
```

Joonis 33. SQL dünaamiline test *NULL* veergude kontrolliks.

- Tühjade tekstiliste väärtuste kontroll – eeldame, et tabeli kirjetes ei ole teksti lõpus või alguses tühjasid tekstilisi väärtuseid (näiteks tühikuid). Selleks loeme tabelis iga veeru kohta kokku sellised read, kus teksti pikkus ei ole sama teksti pikkusega, kus on kasutatud *TRIM* funktsiooni. Kui selliseid ridu on rohkem kui 0, siis tagastame jällegi veeru nime, et testi tulemusena oleks lihtsasti aru saada vea olemusest (Joonis 34).

```
SELECT '' <forEachColumn> || CASE WHEN (SUM(CASE WHEN
LENGTH(TRIM(CAST("<column_name>" AS VARCHAR)))<>LENGTH(CAST("<column_name>"
AS VARCHAR)) THEN 1 ELSE 0 END)>0) THEN '<column_name>;' ELSE '' END
</forEachColumn> FROM <schema>.<object>
```

Joonis 34. SQL dünaamiline test tühjade tekstiliste väärtuste kontrolliks.

- Ridade arvu kontroll – eeldame, et tabelid ei ole tühjad. Selleks kontrollime ridade arvu tabelis ning eeldame, et tulemus on suurem kui 0 (Joonis 35).

```
SELECT COUNT(*) FROM <schema>.<object>
```

Joonis 35. SQL dünaamiline test ridade arvu kontrolliks.

- ETL protsessi käivitamise aja kontroll – eeldame, et paketi käivitus pole võtnud kauem kui üks tund. Selleks otsime andmetest konkreetse paketi viimase käivituse aja sorteerides kuupäeva järgi ning eeldame, et tulemus on väiksem kui 3600 sekundit (Joonis 36).

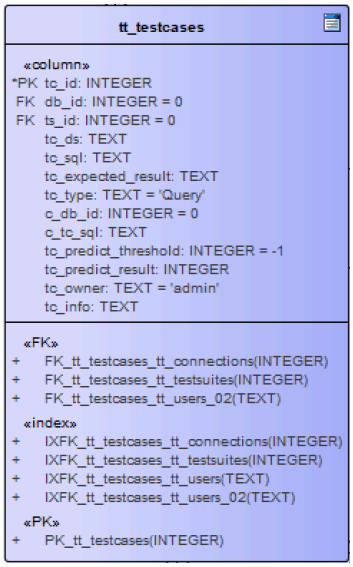
```
SELECT run_time FROM ETL.package_execution WHERE package_name =
<package_name> ORDER BY timestamp DESC LIMIT 1
```

Joonis 36. SQL dünaamiline test ETL protsessi käivituse aja kontrollimiseks.

5.5 Testide genereerimine

Tööriist võimaldab luua SQL teste kolmel erineval viisil: manuaalsed, objektipõhised ning dünaamilised. Teste hoiustatakse lokaalses tabelis (*tt_testcases*), kus iga testi kohta

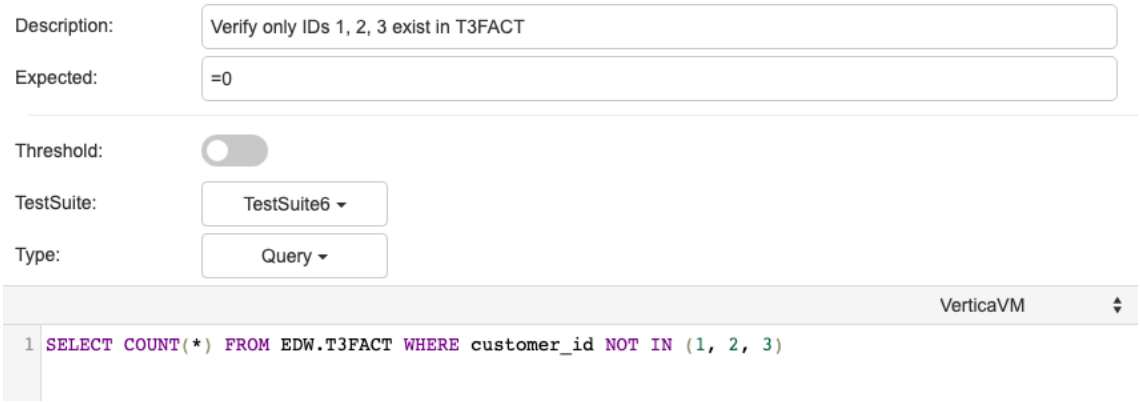
on selle kirjeldus, SQL, oodatav tulemus, viide ühendusele, kuuluvus ja muu testi käivitamisel kasutatav informatsioon (Joonis 37).



Joonis 37. Lokaalse andmebaasi tabeli tt_testcases diagramm.

5.5.1 Manuaalsed testid

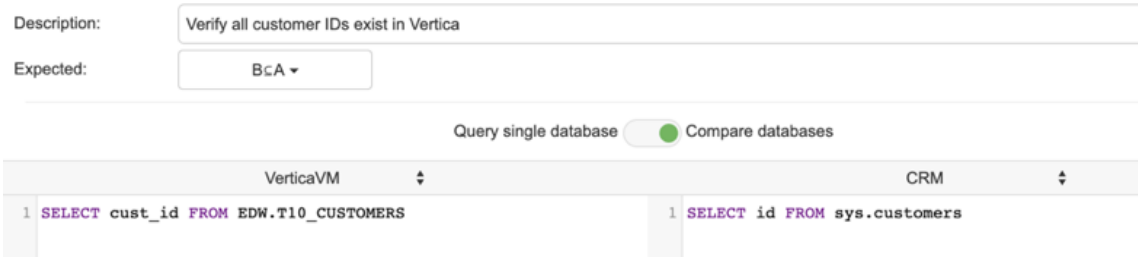
Testi on võimalik kirjutada käsitsi, defineerides käivitatava SQLi ning oodatava tulemuse (Joonis 38). Tööriist eeldab, et SQL tagastab 1*1 ehk ühe veeru ja ühe reaga tulemuse, mis võib olla nii tekstiline kui ka arvuline. Tulemust võrreldakse eeldatava tulemusega (*Expected*), kus pannakse paika, kas oodatav tulemus on mingi numbriga võrdne/suurem/väiksem või on võrdne kindla tekstilise väärtusega.



Joonis 38. Kasutajaliideses testi käsitsi loomine.

Oodatav tulemus võib olla ka SQL-i tulemus mõnest teisest andmebaasist. Sellisel juhul võrreldakse reaalseid andmeid ning oodatavaks tulemuseks võib olla andmete kuuluvus või võrdsus (Joonis 39). Andmete võrdlemiseks loetakse mõlema SQL-i tulemused andmete kogumikku ning seejärel tehakse neist omakorda duplikaadid. Vastavalt

oodatavale tulemusele eemaldatakse kogumikest andmeid, et leida andmetes erinevusi (Joonis 40). Toetudes kogumike suurustele saab järeldada kui palju ridu kummaski kogumikus puudu on.



Joonis 39. Kahe erineva andmebaasi võrdlemine kasutajaliideses.

```
set1 = set1result.getSet();
set2 = set2result.getSet();
HashSet<ArrayList<String>> set1alt = new HashSet<>(set1);
HashSet<ArrayList<String>> set2alt = new HashSet<>(set2);

if (expected.equals("A=B")) {
    set2alt.removeAll(set1);
    set1alt.removeAll(set2);
} else if (expected.equals("B⊆A")) {
    set2alt.removeAll(set1);
} else if (expected.equals("A⊆B")) {
    set1alt.removeAll(set2);
}
```

Joonis 40. Kahe andmekogumiku võrdlemine ning erinevuste leidmine.

5.5.2 Objektipõhised testid

Objektipõhised testid on dünaamilised testid, mida on võimalik genereerida kõikide tabelite või vaadete kohta. Näiteks duplikaatridade kontrollimiseks tuleks kasutajaliidesest valida vaated või tabelid, mille kohta testi genereerida tahetakse. Testide genereerimisel loetakse sisse kõik kasutaja poolt valitud objektid ning valitud testid (Joonis 41). Seejärel kogub tööriist lokaalsest andmebaasist metaandmeid objektide kohta ja kasutab neid dünaamilises testis olevate parameetrite asendamiseks (Joonis 42). Tulemusena luuakse käivitatav SQL iga valitud objekti kohta ning koos dünaamilises testis ette antud kirjelduse ja oodatava tulemusega luuakse uus testijuht (Joonis 43). Genereerimisel tulevad mahulised piirangud veebilehitseja ja kasutaja riistvara jõudlusest, aga katsete tulemusena suudab tööriist korraga genereerida kuni 5000 testi vähem kui paari minutiga.

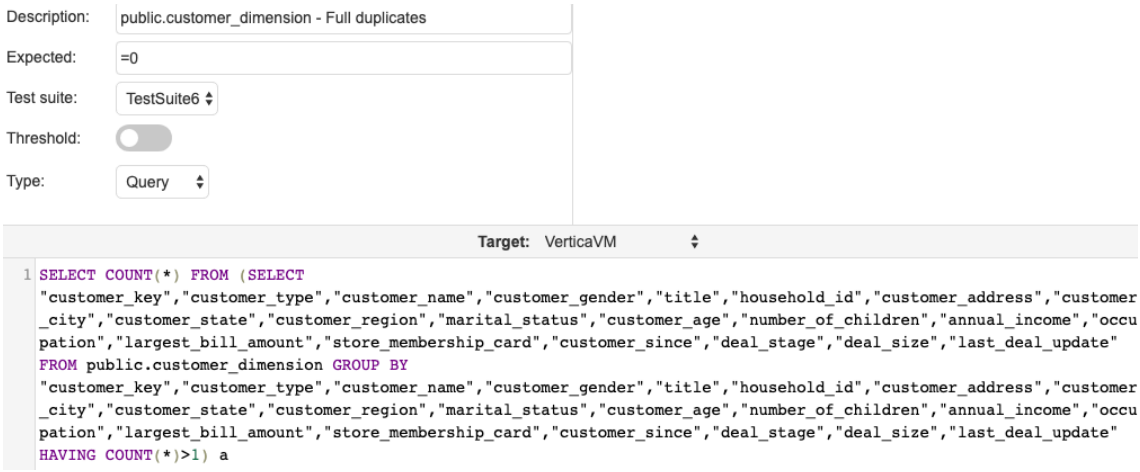
#	Short ds	Long ds	Type
1	<schema>.<object> - Full duplicates	Check if object has any full duplicate rows	Table, View, Dynamic
2	<schema>.<object> - Count of rows	Check if object has any rows inserted	Table, View
3	<schema>.<object> - NULL columns	Check if object has any columns that are NULL in all rows	Table, View
4	<schema>.<object> - White spaces	Check if any columns in a object has values with excess white spaces	Table, View
5	<schema>.<object> - Row count less than 10 000 000	Check if object has more than 10 000 000 rows inserted	Table, View
6	<schema>.<object> - Duplicates by PK	Check if table has duplicate rows by PKs	Table

#	Database	Schema	Object	Type	1	2	3	4	5	6
1	VerticaVM	online_sales	call_center_dimension	Table	☑	☐	☐	☐	☐	☐
2	VerticaVM	online_sales	online_page_dimension	Table	☑	☐	☐	☐	☐	☐
3	VerticaVM	online_sales	online_sales_fact	Table	☑	☐	☐	☐	☐	☐
4	VerticaVM	public	customer_dimension	Table	☑	☐	☑	☐	☐	☐
5	VerticaVM	public	date_dimension	Table	☑	☐	☑	☐	☐	☐
6	VerticaVM	public	employee_dimension	Table	☑	☐	☑	☐	☐	☐
7	VerticaVM	public	inventory_fact	Table	☑	☐	☑	☐	☐	☐
8	VerticaVM	public	product_dimension	Table	☑	☐	☑	☐	☐	☐
9	VerticaVM	public	promotion_dimension	Table	☑	☐	☑	☐	☐	☐
10	VerticaVM	public	shipping_dimension	Table	☑	☐	☑	☐	☐	☐
11	VerticaVM	public	vendor_dimension	Table	☑	☐	☑	☐	☐	☐
12	VerticaVM	public	warehouse_dimension	Table	☑	☐	☑	☐	☐	☐
13	VerticaVM	store	store_dimension	Table	☑	☐	☑	☐	☐	☐
14	VerticaVM	store	store_orders_fact	Table	☑	☐	☑	☐	☐	☐
15	VerticaVM	store	store_sales_fact	Table	☑	☐	☑	☐	☐	☐

Joonis 41. Automaatselt loodavate objektipõhiste testide valimine kasutajaliideses.

```
String dSql = dynamicTest.getDtc_sql();
if(dSql.contains(“<object>”)) {
    dSql = dSql.replace(tableReplace, table.getTableName());
}
if(dSql.contains(“<schema>”)) {
    dSql = dSql.replace(schemaReplace, table.getSchemaName());
}
if(dSql.contains(“<columns>”)) {
    List<String> columns = table.getColumns();
    for (int i = 0; i < columns.size(); i++) {
        columns.set(i, "\"" + columns.get(i) + "\"");
    }
    dSql = dSql.replace(columnsReplace, String.join(",",
table.getColumns()));
}
```

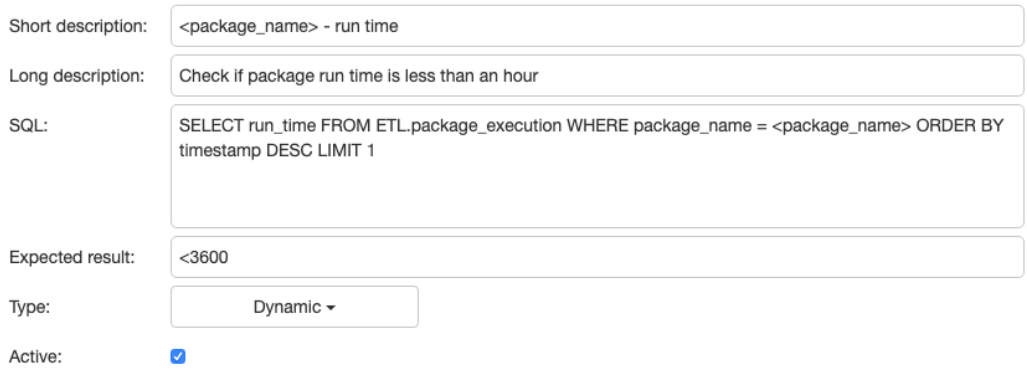
Joonis 42. Parameetrite asendamine dünaamilises testis.



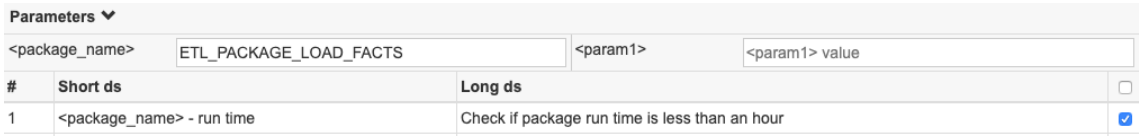
Joonis 43. Automaatselt loodud testijuht kasutajaliideses koos käivitatava SQL päringuga.

5.5.3 Dünaamilised testid

Sarnaselt objektipõhistele testidele on võimalik luua ka tavalisi dünaamilisi teste (Joonis 44). Selliste testide puhul on kasutajal võimalik defineerida parameetrite väärtusi ning seejärel asendatakse need automaatselt testi loomisel (Joonis 45). See võimaldab kasutajal korduvalt kasutada selliseid teste, mis ei ole objektipõhised, vaid nõuavad spetsiifilist kasutajapoolset sisendit. Näiteks kontrollitakse ETL protsessi käivituse pikkust, sest andmete laadimine on tihti ajaliste piirangutega ning sellest tulenevalt peab jälgima protsesside laadimiste aegu.



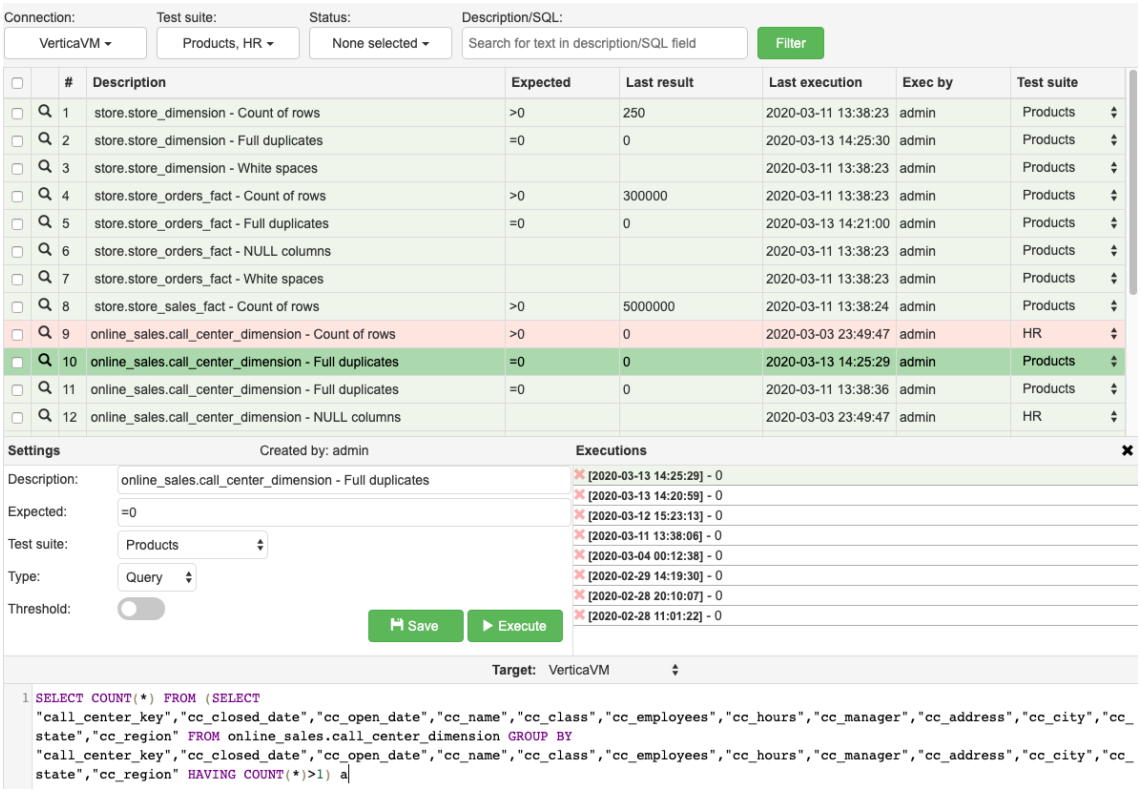
Joonis 44. Dünaamilise testi loomine kasutajaliideses.



Joonis 45. Automaatselt loodavate dünaamiliste testide valimine ning parameetrite sisestamine kasutajaliideses.

5.5.4 Testide haldamine

Teste on võimalik vaadata, muuta ning käivitada eraldi lehelt (Joonis 46). Kuna testide arv on teoreetiliselt piiramatu, siis on võimalik lehel piirata laetavate testide arvu nii arvuliselt kui ka filtreerides testi selle sisu, tulemuste, paketi nime või andmebaaside järgi. Vaikimisi on testide arv piiratud tuhandele reale ehk tuhandele testile, sest veebilehitsejad võivad rohkemate elementide korral muutuda aeglaseks. Ka teste, mis on küll lehel laetud, kuid ei ole kasutajale nähtavad, peidetakse DOM tasemel ära, et lehe jõudlus ja kiirus oleks parem. Selleks lisatakse peidetavatele elementidele külge CSS atribuut *display:none* ning muudetakse üldist tabeli kõrgust, et kasutaja ja veebilehitseja jaoks tunduks tabel sama suur. Tabelis alla või ülespoole kerides võetakse järjest elementidel küljest ära *display:none* atribuut ning muudetakse jällegi vastavalt tabeli kõrgust (Joonis 47). Korraga on DOM tasemel nähtav kuni 200 rida.



Joonis 46. Kasutajaliides testide kuvamiseks, muutmiseks ja käivitamiseks.

```
$("#testTable").scroll(function () {
    var currentRow = parseInt(Math.abs($("#testTable table").offset().top-
68)/29, 10);
    var firstVisible = $('#testsBody tr.visible').first().attr("number");
    var lastVisible = $('#testsBody tr.visible').last().attr("number");

    if ((currentRow-firstVisible<50 && firstVisible != 1) || lastVisible-
currentRow<50) {
        $('#testsBody
tr.visible').removeClass("visible").addClass("notVisible");
        if (currentRow < 100) {
            $('#testsBody tr.notVisible').slice(0,
Number(currentRow)+100).removeClass("notVisible").addClass("visible");
            $("#firstRow").height(0);
        } else {
            $('#testsBody tr.notVisible').slice(currentRow-100,
Number(currentRow)+100).removeClass("notVisible").addClass("visible");
            $("#firstRow").height(($('#testsBody
tr.visible').first().attr("number"))*29-29);
        }

        $("#lastRow").height(($('#testsBody
tr.notVisible').last().attr("number")-$('#testsBody
tr.visible').last().attr("number"))*29);
    }
});
```

Joonis 47. HTML elementide display atribuudi muutmine ning tabeli kõrguse säilitamine lehe kerimisel.

5.6 Käivitamine

Teste käivitatakse nii automaatselt kui ka manuaalselt. Manuaalselt on teste on võimalik käivitada valikuliselt või testiplaanide kaupa, arvestades kasutaja poolt ettemääratud seadistustega. Käivitamisel saadetakse kasutajaliidesest kõik valitud testide unikaalsed ID-d *back-endi*, kus iga ID kohta päritakse lokaalsest tabelist kõik käivituseks vajalik informatsioon. Seejärel luuakse vajalikud ühendused testitavate andmebaaside vastu. Iga unikaalse andmebaasi vastu luuakse N ühendust, mida tööriist päringute käivitamiseks kasutada saab. Ühenduste arv ning Java poolt kasutatavate *threadide* arv on piiratud andmebaasi *threshold* seadistusega. Kõik ühendused lisatakse *connection pooli*, kust tööriist vabasid ühendusi otsimas käib. Vaba ühenduse korral saadab programm testi kohta käiva SQLi vastu andmebaasi ning ootab sealt vastust (Joonis 48).

```
ExecutorService executor = Executors.newFixedThreadPool(maxThreads);
DataSourcePool connectionPool = null;
Integer testTimeout;

for (Testcase testcaseToExec : testcases) {
	for (ActiveConnectionModel connection : connections) {
		if (testcaseToExec.getDb_id().equals(connection.getDbId())) {
			connectionPool = connection.getPool();
			testTimeout = connection.getTimeout();
		}
	}
	Runnable worker = new ExecInParallelManual(testcaseToExec,
connectionPool, tsExecId, sessionId, executedBy, testTimeout);
	executor.execute(worker);
}
```

Joonis 48. Testide connection poolide seadistamine ning käivitamine.

Vastuseks loetakse andmebaasist tagastatav esimese rea esimene veerg. Seejärel võrreldakse vastust oodatava tulemusega. Selleks teeb programm kõigepealt kindlaks, kas tegu on olema numbrilise võrdlusega või võrreldakse tekstilist väärtust. Vastavalt oodatavale tulemusele tagastatakse testi staatuseks kas „ok“ ehk päringu vastus (*lastResult*) vastas oodatavale tulemusele (*expectedResult*) või „nok“, kui test ei vastanud oodatud tulemusele (Joonis 49). Kui päring saab andmebaasi poolt vea ning kutsutakse välja *Java Exception*, siis tagastatakse staatuseks „error“ ning lisatakse andmebaasi poolt tagastatud veateade. Käivituse tulemusena salvestatakse lokaalsesse andmebaasi iga käivituse kohta testi ID, käivituse aeg, tulemus ja kelle poolt test käivitati.

```
if (expectedResult.contains("=") && !lastResult.equals("")) {
	expectedResultClean = expectedResult.split("=")[1];
	try {
		if (Double.parseDouble(expectedResultClean) ==
	Double.parseDouble(lastResult) || expectedResult.equals(lastResult)) {
			status = "ok";
		} else {
			status = "nok";
		}
	} catch (NumberFormatException | NullPointerException nfe) {
		status = "nok";
	}
}
```

Joonis 49. Testi käivituse tulemuse võrdlemine oodatava tulemusega ning staatuse määramine.

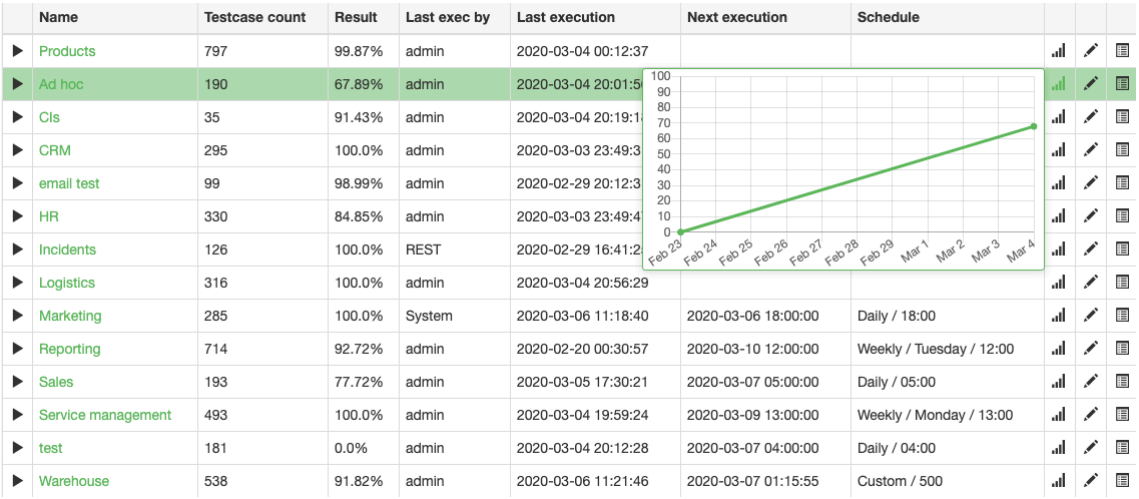
5.6.1 Planeerimine

Teste on võimalik automaatselt käivitada kindlal ajal, paigutades need testipakettidesse ja seadistades need planeeritult käivituma. Testipakette on võimalik automaatselt käivitada kord nädalas, kord päevas või iga N minuti tagant. Käivitamiseks käib tööriist automaatselt iga minut kontrollimas, kas on testipakette, mille järgmine käivitamise aeg (*Next execution*) on varasem kui praegune aeg. Sellisel juhul kõik paketis olevad testid käivitatakse ning peale käivitust arvutatakse välja järgmise automaatse käivituse aeg (Joonis 50). Testipakettide tulemusi on võimalik näha praeguse hetke kohta või kuvada tulemusi iga päeva kohta joondiagrammil (Joonis 51).

```
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
Calendar last = Calendar.getInstance();
Calendar now = Calendar.getInstance();

if (scheduleType.equals("Weekly")) {
	last.set(Calendar.DAY_OF_WEEK,
DayOfWeek.valueOf(testSuiteModel.getTsScheduleDay().toUpperCase()).getValue()
+ 1);
	last.set(Calendar.HOUR_OF_DAY,
Integer.parseInt(testSuiteModel.getTsScheduleTime().split(":")[0]));
	last.set(Calendar.MINUTE,
Integer.parseInt(testSuiteModel.getTsScheduleTime().split(":")[1]));
	last.set(Calendar.SECOND, 0);
	if (last.before(now)) {
		last.add(Calendar.WEEK_OF_YEAR, 1);
		nextExec = dateFormat.format(last.getTime());
	} else {
		nextExec = dateFormat.format(last.getTime());
	}
}
```

Joonis 50. Testi paketi järgmise käivituse kuupäeva ning aja arvutamine nädalapõhiseks käivitamiseks.



Joonis 51. Testi pakettide tulemuste kuvamine kasutajaliideses.

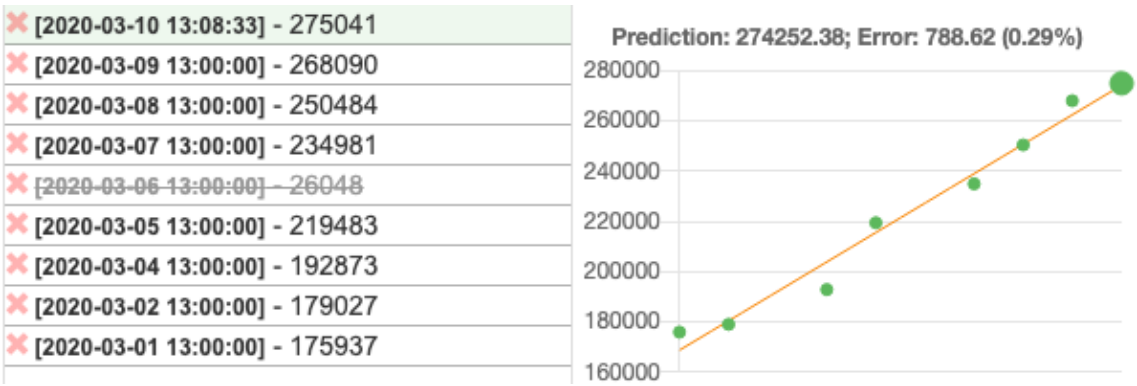
5.6.2 Statistikapõhine vigade tuvastamine

Numbriliste agregeeritud testitulemuste korral on võimalik vigu tuvastada kasutades lineaarset regressiooni. Sellisteks testitulemusteks võivad olla näiteks *SUM*, *AVG* või *COUNT* päringu tulemused. Seda kasutades on võimalik leida ebakorrapärasusi andmete muutumisel, arvestades kasutaja defineeritud veapiiriga. Arvutamiseks laetakse lokaalsest andmebaasist alla testi käivitamise ajalugu (käivitamise aeg ning tulemus). Ajaloo põhjal arvutatakse lineaarset regressiooni kasutades eeldatav numbriline tulemus praeguse hetke seisuga ja võrreldakse seda reaalse käivituse tulemusega (Joonis 52). Kui tulemused erinevad +-n protsenti, siis on tegu ebakorrapärasusega ning testile pannakse külge vastav indikaator ja staatus. Tulemusi kuvatakse kasutajale graafikul, kus on märgitud testide tulemused ning lineaarsus (Joonis 53). Vigaseid tulemusi on võimalik ajaloost eemaldada, et nendega mitte arvestada uute tulemuste analüüsimisel.

```
LinearRegression linearRegression = new LinearRegression(aegList,
tulemusList);

double prediction = linearRegression.predict((double) date.getTime());
double difference = Math.abs((Double.valueOf(result)/prediction - 1)) * 100;
if (difference > threshold) {
	predictResult = true;
} else {
	predictResult = false;
}
```

Joonis 52. Testi arvulise tulemuse ennustamine kasutades lineaarset regressiooni ning tulemuse võrdlemine käivituse tulemusega.



Joonis 53. Testi käivituste ajalugu ning graafik koos ennustatava tulemusega ja veaga 0.29%.

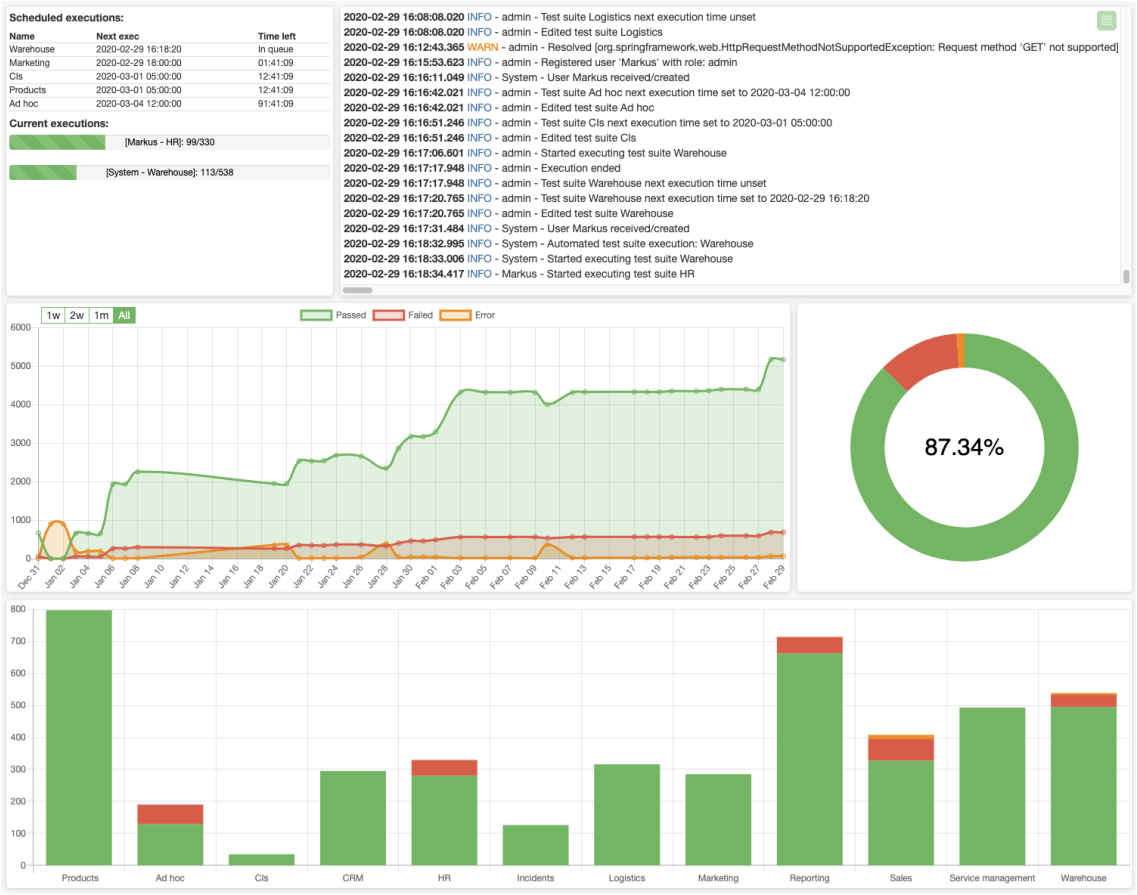
5.7 Tulemused ja aruandlus

Testi tulemusi ja andmete üldist seisu saab jälgida tööriista töölaualt, kus on informatsioon aktiivsete käivituste, planeeritud käivituste, töölogi ning testitulemuste kohta (Joonis 54). Töölaual kuvatavad graafikud on lahendatud kasutades ChartJS JavaScripti põhist raamistikku. Graafikuid on kokku kolm:

- Joondiagramm, millel kujutatakse testi staatuseid ajas. Kokku kuvatakse kolme staatust (Passed, Failed, Error) ning neid iga päeva viimase käivituse tulemuse kohta. Graafikul on võimalik filtreerida tulemusi staatuse ning perioodi põhjal. Maksimaalseks perioodiks on kaks kuud ning minimaalseks üks nädal. Andmed laetakse kokku ühe päringuna lokaalsest SQLite baasist (Joonis 55).

- Sektordiagramm, millel kujutatakse testide staatust hetkeseisuga. Tulemuseks on sama, mis joondiagrammi tulemus viimase päeva kohta.

- Tulpdiagramm, millel kujutatakse testide staatust testipakettide lõikes.



Joonis 54. Tööriista töölaud kasutajaliideses.

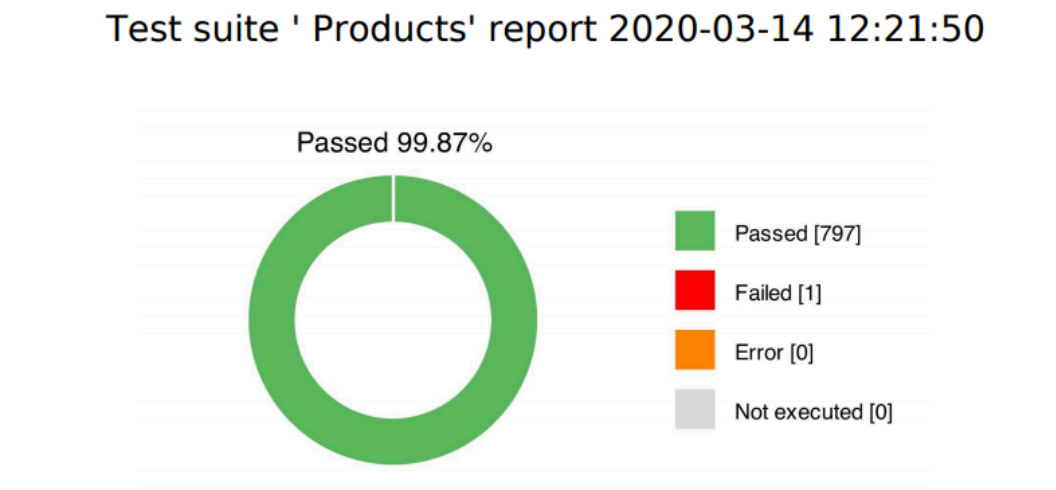
```
SELECT a.kp, a.status, COUNT(*) FROM (
SELECT a.kp, first_value(a.status) over (partition by a.value_partition,
a.tc_id order by a.tc_id, a.kp) status
FROM (SELECT a.kp, a.tc_id, b.status, sum(case when b.status is null then 0
else 1 end) over (order by a.tc_id, a.kp) as value_partition FROM (
SELECT a.kp, b.tc_id FROM (
SELECT DISTINCT SUBSTR(exec_ts,1, 10) kp FROM tt_executions a JOIN
tt_testcases b ON a.tc_id = b.tc_id WHERE exec_ts IS NOT NULL
) a CROSS JOIN (SELECT tc_id FROM tt_testcases a LEFT JOIN tt_testsuites b ON
a.ts_id = b.ts_id WHERE (1=1) ) b ) a LEFT JOIN (
SELECT MAX(exec_ts), tc_id, status, SUBSTR(exec_ts,1, 10) kp FROM
tt_executions GROUP BY tc_id, SUBSTR(exec_ts,1, 10)
) b ON a.tc_id = b.tc_id AND a.kp = b.kp ) a ) a WHERE a.status IS NOT NULL
GROUP BY a.kp, a.status ORDER BY 1 DESC, 2
```

Joonis 55. SQL päring testide käivituste tulemuste kuvamiseks päeva ja staatuse põhjal.

5.7.1 PDF aruandlus ning e-mailide saatmine

Aruandluseks on võimalik raport genereerida või automaatselt e-mailile saata PDF failina. Aruanne on testipaketi põhine ning seal kuvatakse iga testi tulemust ning üldist diagrammi kõigi selle testipaketi tulemuste kohta (Joonis 56). Aruande e-mailile saatmiseks tuleb esmalt seadistada SMTP seaded, mille pealt tööriist e-maile saatma

hakkab. E-maile saadetakse ainult juhul, kui testi tulemused ületavad kasutaja määratud veapiiri, on seadistatud SMTP seaded ning on lisatud saaja e-maili aadress (Joonis 57). E-maili saatmiseks kasutatakse meetodit *sendScheduledEmail*, kus luuakse ning autenditakse SMTP ühendus. Seejärel kasutatakse *MimeMessage* klassi, et luua e-maili pealkiri ning sisu (Joonis 58). E-maili sisuks on PDF fail, mis luuakse kasutades PDFbox raamistikku. Loodud PDF paigutatakse ajutiselt serverisse ning pärast e-maili saatmist see kustutatakse.



#	Description	Expected	Result	Execution time	Executed by
1	pg_catalog.pg_prepared_statements - Count of rows	>0	0	2020-03-11 13:38:33	admin
2	store.store_dimension - Count of rows	>0	250	2020-03-11 13:38:23	admin
3	store.store_dimension - Full duplicates	=0	0	2020-03-13 14:25:30	admin
4	store.store_dimension - White spaces			2020-03-11 13:38:23	admin
5	store.store_orders_fact - Count of rows	>0	300000	2020-03-11 13:38:23	admin
6	store.store_orders_fact - Full duplicates	=0	0	2020-03-13 14:21:00	admin
7	store.store_orders_fact - NULL columns			2020-03-11 13:38:23	admin
8	store.store_orders_fact - White spaces			2020-03-11 13:38:23	admin
9	store.store_sales_fact - Count of rows	>0	5000000	2020-03-11 13:38:24	admin

Joonis 56. Genereeritud PDF aruanne testi paketi tulemustega.

```
if ("System".equals(userName) && testSuiteModel.getTsEmails() != null
&& !testSuiteModel.getTsEmails().isEmpty()) {
      if (testSuiteResponse.getTsEmailThreshold() <= 100-
Double.parseDouble(testSuiteResponse.getTsLastResult())) {
            emailService.sendScheduledEmail(testSuiteResponse);
      }
}
```

Joonis 57. E-maili saatmise tingimuste kontrollimine.

```
MimeMessage msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(emailModel.getSmtpFrom(), "Testing tool"));
msg.setSubject("[Test tool] '" + testSuiteModel.getTsName() + "' report " +
testSuiteModel.getTsLastExec(), "UTF-8");
msg.setSentDate(new Date());

MimeBodyPart textPart = new MimeBodyPart();
textPart.setText("Test suite report added as an attachment", "UTF-8");
MimeBodyPart filePart = new MimeBodyPart();

reportName = "reports/" +
testSuiteService.generateTestSuiteReport(testSuiteModel.getTsId());
FileDataSource fileDataSource = new FileDataSource(reportName);
filePart.setDataHandler(new DataHandler(fileDataSource));
filePart.setFileName(fileDataSource.getName());

Multipart multipart = new MimeMultipart();
multipart.addBodyPart(textPart);
multipart.addBodyPart(filePart);

msg.setContent(multipart);
msg.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(testSuiteModel.getTsEmails(), false));
Transport.send(msg);
```

Joonis 58. E-maili sisu loomine ning selle saatmine.

6 Tööriista kasutus suurettevõttes

Tööriist võetakse ettevõttes kasutusele andmeaida tiimiliikmete poolt, kelleks on arendajad, analüütikud ja arhitektid. Igal tiimiliikmel on tööriistas oma kasutaja, tänu millele on võimalik uusi arendusi testida teistest sõltumata. Testimine toimub domeenipõhiselt ehk iga andmedomeeni kohta luuakse testide pakett. Ühe andmedomeeni kohta võib olla vastavalt keskkondadele ka mitu paketti (näiteks arenduskeskkond ja toodangu keskkond). Plaanipärane testide automaatne käivitamine toimub õhtuti vahemikus 18:00-23:00, et mitte koormata andmebaase tööajal või igaöiste laadimiste ajal.

6.1 Juurutamine ja kasutuselevõtt

Tööriista installeerimiseks loodi esmalt Linuxi põhine virtuaalne masin, mille konfiguratsiooniks on 2-tuumaline protsessor koos 4GB mäluga. Turvalisuse kaalutlustel loodi virtuaalne masin eraldi serverisse, millele on ligipääs vaid valitud kasutajatel. Kuna tööriist töötab Java peal, siis installeeriti masinasse ka OpenJDK8 tasuta Java versioon. Loodud testimisvahend käivitatakse masinas teenusena ning tööriista olekut käiakse automaatselt kontrollimas iga 15 minuti tagant. Kui teenus ei tööta, siis pannakse see automaatselt uuesti tööle – selle tulemusena on lihtsasti võimalik tööriista uuendada, peatades käsitsi teenus ning uuendades masinas paiknevat WAR faili.

Kasutajate kaasamiseks asusid tööriista esmalt kasutama valdkonna arhitektid, kelle kasutuse tulemusena töötati välja head tavad ja nõuded. Seejärel kaasati teisi meeskonnaliikmeid, kes kasutasid tööriista oma andmedomeeni testimiseks ning andmekvaliteedi monitoorimiseks. Esialgse kasutamise tulemusena saadi ka palju tagasisidet tööriista vigadest ning puudustest, mille tulemusena on tööriista edasi arendatud. Tagasiside põhjal on kõige rohkem rõhku pandud turvalisusele ja kasutajamugavusele.

6.2 Tulemuste analüüs

Analüüsiks kasutas autor tööriista ühe andmedomeeni peal uute arenduste testimiseks ning igapäevaste laadimiste kvaliteedi monitoorimiseks. Testiti kasutaja-, mudeldamis- ning laadimiskihti. Kasutajakihti loodi dünaamiliste testide põhjal iga vaate kohta kaks

testi: duplikaatridade ning ridade arvu kontroll. Analüüsitava andmedomeeni vaateid kasutajakihis on 35 ning kokku loodi 76 testi, millest 6 oli käsitsi kirjutatud. Käsitsi kirjutatud testid olid andmete võrdlemiseks allika ja kasutajakihi vaate vahel, et veenduda kõikide kirjete olemasolus andmeaidas. Mudeldamiskihis on kokku 81 tabelit, millele igale ühele loodi ridade arvu test ning osadele tabelitele duplikaatridade test. Kokku loodi mudeldamiskihi peale 112 testi. Laadimiskihis testitakse 65 tabelis ridade arvu, et veenduda andmete laadimises. Kokkuvõtlikult loodi analüüsitava andmedomeeni peale 253 testi, mis lisati ühte testide paketti ning mida käivitatakse automaatselt iga päev kell 20:00 (Joonis 59).

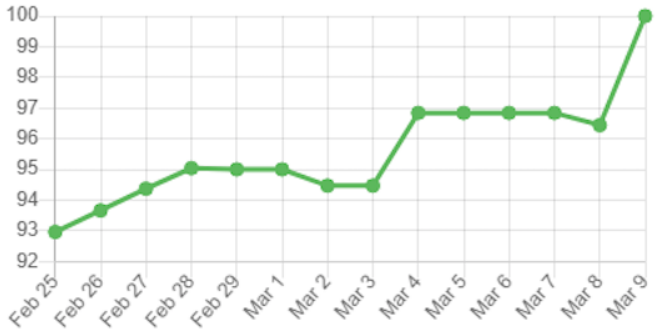
Joonis 59. Analüüsitava andmedomeeni testide pakett ning selle seadistused.

Esmasel käivitamisel leiti vead kasutajakihis ning mudeldamiskihis. Oodatava tulemuse tagastas 235 testi ning mitteoodatava tulemuse tagastas 18 testi. Nendest 12 olid seotud duplikaatidega kasutajakihis ning 6 testi seotud duplikaatidega mudeldamiskihis. Mudeldamiskihis parandati vead viies tabelis, mille oli põhjustanud tabelisse andmete laadimisel kasutatav vale *Knowledge Module* rakenduses ODI. Ühes tabelis oli vea põhjustanud laadimiskihti valesti laetud andmed. Nende kuue tabeli parandamisel paranesid ka oodatavad tulemused seitsmes kasutajakihi vaates, mis eelnevalt sisaldasid duplikaatridu. Viies kasutajakihi vaates olid vead põhjustatud vaadete kokkupanekul. Täpsemalt pandi vaates kokku erinevaid tabeleid unikaalsete koodide põhjal, aga mõnes tabelis olid need koodid väiketähtedega ning teistes suurte tähtedega. Parandamiseks muudeti kõik vastavad *JOIN* tingimused ära selliseks, et need välistaksid erinevusi suurte ja väikeste tähtede vahel. Selleks muudeti kogu kasutatav väärtus suurte tähtede peale ehk kasutati SQL funktsiooni *UPPER* (Joonis 60).

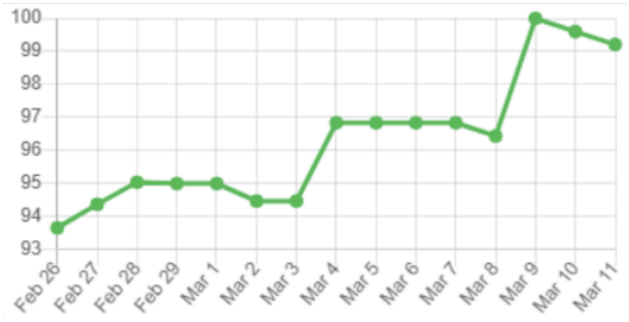
```
UPPER(a.id_code) = UPPER(b.id_code)
```

Joonis 60. SQL JOIN tingimused väike- ja suurtähtede välistamiseks.

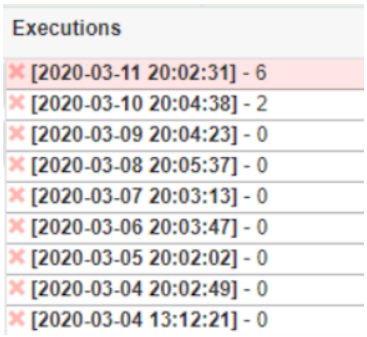
Parandamiste tulemustena saavutati oodatav tulemus 253 testis ehk 100% testidest oli staatuses *passed* (Joonis 61). Järgnevatel päevadel oli näha langust testide tulemustes (Joonis 62). Vigu põhjustasid tekkinud duplikaatread mudeldamiskihid, mis omakorda põhjustasid duplikaate kasutajakihis (Joonis 63). Testi igapäevase käivitamise tulemusena suudeti antud kasutajakihi vaate põhjal viga tuvastada enne kui see andmete kasutajatele negatiivset mõju jõudis avaldada.



Joonis 61. Analüüsitava andmedomeeni testide tulemused päevade lõikes 25.02.2020 - 09.03.2020.

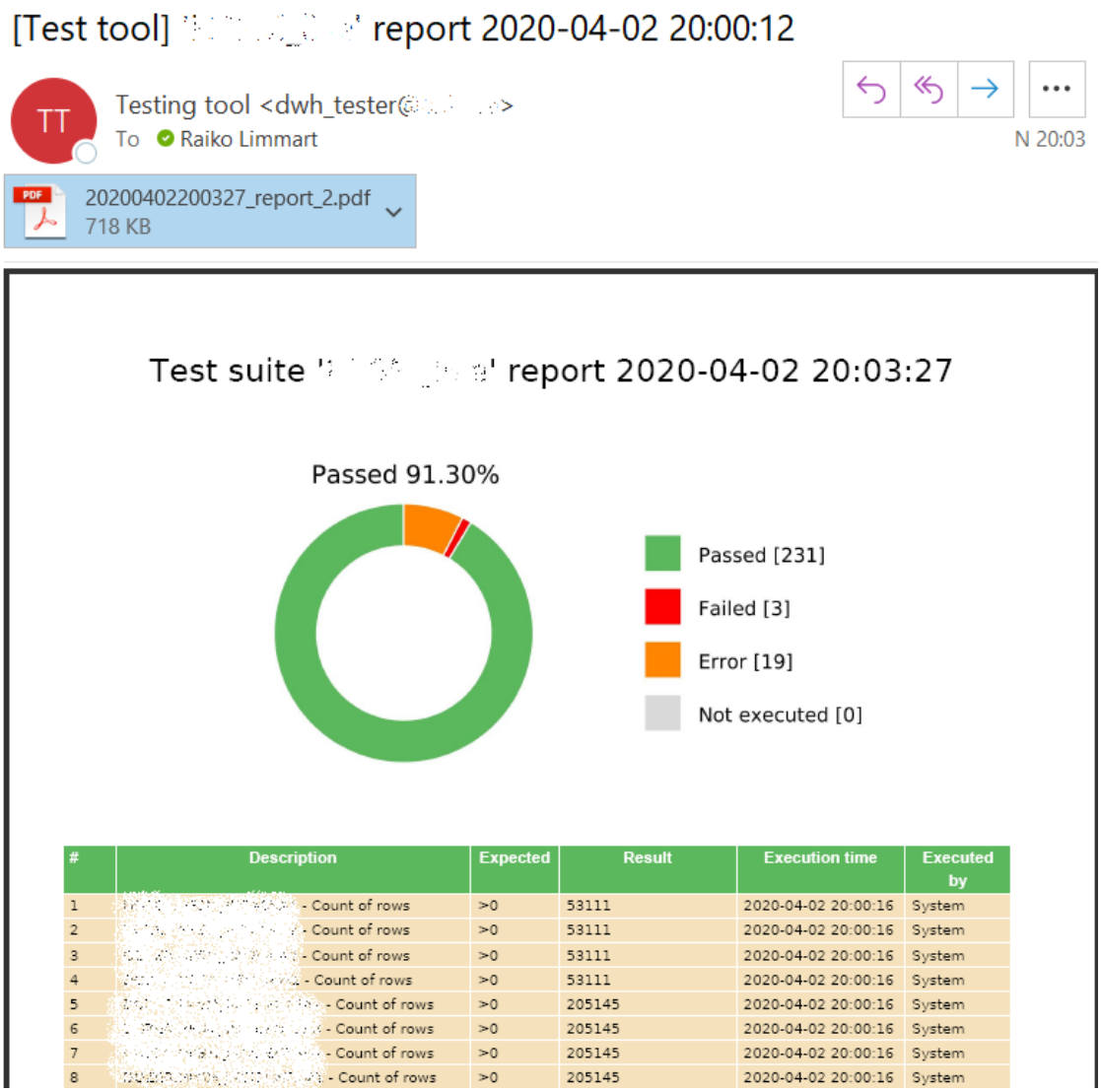


Joonis 62. Analüüsitava andmedomeeni testide tulemused päevade lõikes 26.02.2020 - 11.03.2020.

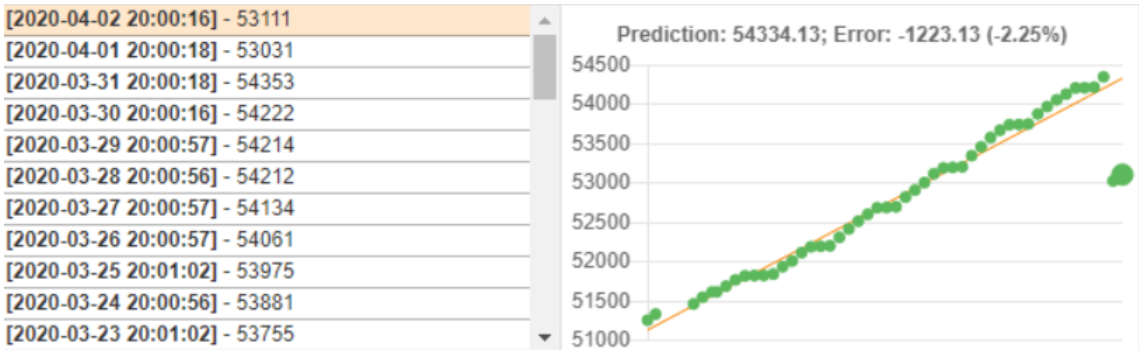


Joonis 63. Kasutajavaate duplikaatridade testi tulemused 04.03.2020 - 11.03.2020.

Kuu alguse tulemusena muudeti analüüsitava andmedomeeni allikabaasis tabeleid, kustutades sealt vanemaid kirjeid. Seda infot ei edastatud andmeaida arendajatele ning seda tuvastas arendatud tööriist kasutades statistikapõhist veatuvastust. Vead tuvastati kokku 19 tabelis ning selle kohta saatis tööriist ka automaatse e-maili koos aruandega (Joonis 64). Testide ajalugu vaadates oli näha, kuidas igapäevaselt laetud ridade arv oli väiksemaks jäänud ning ei olnud enam lineaarne (Joonis 65). Testimise tulemusena hakati ka andmeaidas arvestama allikapoolse andmete säilitamise ajaga.



Joonis 64. Tööriista poolt saadetud automaatne e-mail koos testide tulemuste aruandega.



Joonis 65. Testi tulemuste ajalugu koos visualiseeritud lineaarse kasvuga ning tuvastatud veaga.

7 Kokkuvõte

Töö eesmärkideks oli uurida andmekvaliteedi probleeme, vigade tüüpe ja nende tekkepõhjuseid andmeaitades ning tutvuda erinevate andmeaitade testimisvahenditega. Töö praktilise osa eesmärgiks oli luua automaatne andmete testimise vahend ning juurutada see suurettevõtte süsteemidesse ja analüüsida selle kasutuse tulemusi. Testimisvahend arendati veebirakendusena Spring Boot raamistikul programmeerimiskeeles Java, kasutades *Model-View-Controller* arhitektuuri.

Tulemusena loodi tööriist, mis juurutati analüüsitava suurettevõtte süsteemidesse ja võimaldab testida andmeid ettevõttes kasutatavas Vertica andmebaasimootoril põhineval andmeaidas. Tööriist võimaldab automaatselt genereerida SQL päringuid ning luua testijuhte, kasutades korduvkasutatavaid teste. Testide käivitamist saab automatiseerida, käivitades neid plaanipäraselt vastavalt kasutaja poolt ettemääratud ajale. Tulemuste raporteerimiseks kasutab tööriist PDF aruandlust e-maili teel, kuvab rakenduse töölauale graafikuid ning võimaldab testide tulemusi vaadata ning teste hallata tsentraalsest vaatest.

Tööriist võeti kasutusele andmeaida meeskonnas ning magistritöö osana analüüsiti autori poolt ühte ettevõtte andmeaida andmedomeeni. Analüüsi tulemusena loodi 253 testijuhtu, millest esmasel käivitusel tagastas mitteoodatava tulemuse 18. Testide analüüsimisel tuvastati andmedomeenis erinevaid vigu andmete laadimisel ja kasutajavaadete kokkupanekul. Kõik tuvastatud vead parandati ning tulemusena tagastasid kõik 253 testijuhtu oodatava tulemuse.

Kasutatud kirjandus

[1] A. Gosain, S. Nagpal and S. Sabharwal, "Quality metrics for conceptual models for data warehouse focusing on dimension hierarchies," August 2011. [Online]. Available: https://dl.acm.org/doi/10.1145/1988997.1989015. [Accessed January 2020].

[2] A. Hajmoosaei, M. Kashfi and P. Kailasam, "COMPARISON PLAN FOR DATA WAREHOUSE SYSTEM ARCHITECTURES," 2011. [Online]. Available: https://ieeexplore.ieee.org/document/6108446. [Accessed January 2020].

[3] J. M. Artz, "How Good is that Data in the Warehouse?," 1997. [Online]. Available: https://dl.acm.org/doi/10.1145/272657.272688. [Accessed April 2020].

[4] M. M. Hamad and A. A. Jihad, "An Enhanced Technique to Clean Data in the Data Warehouse," 2011. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6149998. [Accessed January 2020].

[5] Munawar, N. Salim and R. Ibrahim, "Towards Data Quality into the Data Warehouse Development," 2011. [Online]. Available: https://ieeexplore.ieee.org/document/6118854. [Accessed January 2020].

[6] L. Muñoz, J.-N. Mazón and J. Trujillo, "Measures for ETL Processes Models in Data Warehouses," 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1651415.1651422. [Accessed January 2020].

[7] R. Obongo, "Data Warehouse Basics – Do You Need One?," 26 July 2018. [Online]. Available: https://www.analytics8.com/insights/data-warehouse-basics/. [Accessed January 2020].

[8] D. Anderson, "What is "The Data Vault" and why do we need it?," 27 March 2015. [Online]. Available: https://www.talend.com/blog/2015/03/27/what-is-the-data-vault-and-why-do-we-need-it/. [Accessed 2020].

[9] D. Rao, V. N. Gudivada and V. V. Raghavan, "Data Quality Issues in Big Data," 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7364065. [Accessed January 2020].

[10] N. Idris and K. Ahmad, "Managing Data Source Quality for Data Warehouse in Manufacturing Services," 2011. [Online]. Available: https://ieeexplore.ieee.org/document/6021598. [Accessed January 2020].

[11] Lotame, "Why Is Data Quality Important?," 30 April 2019. [Online]. Available: https://www.lotame.com/why-is-data-quality-important/. [Accessed March 2020].

[12] D. P. Ballou and G. K. Tayi, " Enhancing DataQuality in DataWarehouse Environments," January 1999. [Online]. Available: https://dl.acm.org/doi/10.1145/291469.291471. [Accessed April 2020].

[13] ISO25000, "ISO/IEC 25012," [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25012. [Accessed March 2020].

[14] P. Samsuwan and Y. Limpiyakorn, "Generation of Data Warehouse Design Test Cases," 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7292985. [Accessed January 2020].

[15] D. Loshin, "Rule-Based Data Quality," 2002. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/584792.584894. [Accessed January 2020].

[16] Experian, "The state of data quality," September 2013. [Online]. Available: https://www.experian.com/assets/decision-analytics/white-papers/the%20state%20of%20data%20quality.pdf. [Accessed 2020].

[17] W. W. Eckerson, "DATA QUALITY AND THE BOTTOM LINE," 2001. [Online]. Available: http://download.101com.com/pub/tdwi/Files/DQReport.pdf. [Accessed April 2020].

[18] S. Ranjit and S. Kawaljeet, "A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing," May 2010. [Online]. Available: https://www.researchgate.net/publication/46093554. [Accessed April 2020].

[19] J. Paulsen, "Enormous Growth in Data is Coming — How to Prepare for It, and Prosper From It," Seagate, 04 April 2017. [Online]. Available: https://blog.seagate.com/business/enormous-growth-in-data-is-coming-how-to-prepare-for-it-and-prosper-from-it/. [Accessed February 2020].

[20] S. Moore, "How to Stop Data Quality Undermining Your Business," Gartner, 18 January 2018. [Online]. Available: https://www.gartner.com/smarterwithgartner/how-to-stop-data-quality-undermining-your-business/. [Accessed February 2020].

[21] T. C. Redman, "Seizing Opportunity in Data Quality," MITSloan, 27 November 2017. [Online]. Available: https://sloanreview.mit.edu/article/seizing-opportunity-in-data-quality/. [Accessed February 2020].

[22] J. Aunola, "DATA QUALITY IN DATA WAREHOUSES," 2018. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/146311/Aunola_Jere.pdf. [Accessed April 2020].

[23] M. Davie, "Why Bad Data Could Cost Entrepreneurs Millions," Entrepreneur, 15 April 2019. [Online]. Available: https://www.entrepreneur.com/article/332238. [Accessed April 2020].

[24] M. Golfarelli and S. Rizzi, "A Comprehensive Approach to Data Warehouse Testing," 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1651291.1651295. [Accessed January 2020].

[25] M. J. S.-C. C. d. l. R. Javier Tuya, "A practical guide to SQL white-box testing," April 2006. [Online]. Available: https://dl.acm.org/doi/10.1145/1147214.1147221. [Accessed April 2020].

[26] N. ElGamal, A. ElBastawissy and G. Galal-Edeen, "Data Warehouse Testing," 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2457317.2457319. [Accessed January 2020].

[27] H. M. Sneed and K. Erdoes, "Testing big data (Assuring the quality of large databases)," 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7107424. [Accessed January 2020].

[28] D. Willmor and S. M. Embury, "An Intensional Approach to the Specification of Test Cases for Database Applications," May 2006. [Online]. Available: https://dl.acm.org/doi/10.1145/1134285.1134301. [Accessed April 2020].

[29] S. Savanur and K. S. Shreedhara, "Automated Data Validation for Data Warehouse Testing," 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7955219. [Accessed January 2020].

[30] H. Homayouni, "Testing Extract-Transform-Load Process in Data Warehouse Systems," October 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8539188. [Accessed January 2020].

[31] H. Homayouni, S. Ghosh and I. Ray, "An Approach for Testing the Extract-Transform-Load Process in Data Warehouse Systems," June 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3216122.3216149. [Accessed January 2020].

[32] RightData, "Product Overview," RightData, [Online]. Available: https://www.getrightdata.com/product.php. [Accessed April 2020].

[33] RightData, "FAQs," RightData, [Online]. Available: https://www.getrightdata.com/faq.php. [Accessed April 2020].

[34] QuerySurge, "What is QuerySurge?," QuerySurge, [Online]. Available: https://www.querysurge.com/product-tour/what-is-querysurge. [Accessed April 2020].

[35] QuerySurge, "Installation, Architecture, & Supported Environments," QuerySurge, [Online]. Available: https://www.querysurge.com/product-tour/product-architecture. [Accessed April 2020].

[36] Datagaps Inc, "Key Features," Datagaps Inc, [Online]. Available: https://www.datagaps.com/etl-testing-tools/etl-validator/. [Accessed April 2020].

[37] Datagaps Inc, "ETL Validator Setup Guide," 2015. [Online]. Available: https://datagaps.freshdesk.com/support/solutions/articles/1000199572-etl-validator-setup-guide. [Accessed April 2020].

[38] QuerySurge, "Licensing & Pricing Options," QuerySurge, [Online]. Available: https://www.querysurge.com/product-tour/licensing-pricing-options. [Accessed April 2020].

[39] SQLite, "Write-Ahead Logging," SQLite, 21 July 2010. [Online]. Available: https://www.sqlite.org/wal.html. [Accessed January 2020].

[40] V. Skoumal, "Parallel read and write in SQLite," Skoumal, 02 October 2016. [Online]. Available: https://www.skoumal.com/en/parallel-read-and-write-in-sqlite/. [Accessed March 2020].