TARTU ÜLIKOOL
Loodus- ja täppisteaduste valdkond
Arvutiteaduse instituut
Andmeteaduse õppekava

Kristjan Lõhmus

Avaandmeait

Magistrtitöö (15 EAP)

Juhendajad: Kristo Raun, PhD
Priit Kongo, MSc

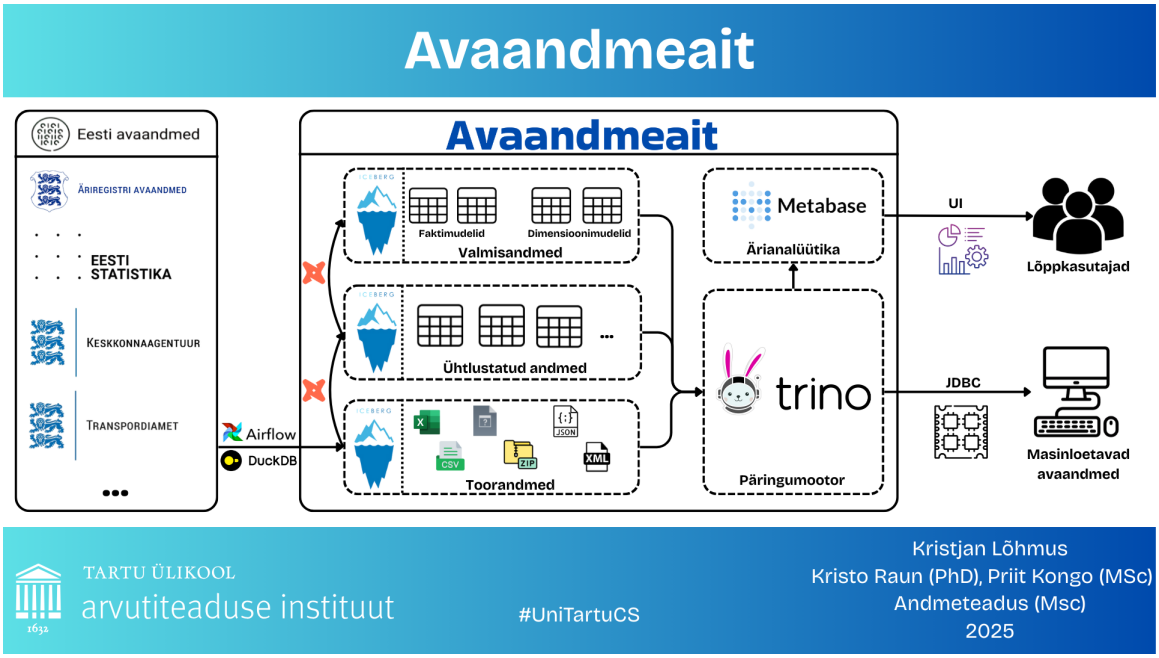
Tartu 2025

Avaandmeait

Lühikokkuvõte:
Eestis avaldavad erinevad riigiasutused arvukalt avaandmestikke, kuid nende kasutuspotentsiaali piiravad andmete killustatus, ebaühtlane kvaliteet ja puudulik masinloetavus. Käesolev magistritöö tegeleb nende probleemide lahendamisega, kujundades ja juurutades keskse avaandmete analüüsiplatvormi nimega Avaandmeait. Platvorm on üles ehitatud kaasaegse andmejärvemaja arhitektuuri alusel, ühendades andmejärvede paindlikkuse andmeladude struktureeritud päringuvõimekusega. Platvormi loomisel on kasutatud modernseid andmeinseneeria tehnoloogiaid: Apache Iceberg ühtse tabeliformaadi tagamiseks, Trino hajusate päringute teostamiseks, Apache Airflow andmetöövoogude orkestreerimiseks ning dbt andmetransformatsioonide haldamiseks. Ühtlustades andmeformaate ja koondades varem killustunud andmestikud ühtsesse hoidlasse, tõstab süsteem oluliselt andmete kvaliteeti, võrreldavust ja kättesaadavust, võimaldades andmekasutajatel keskenduda sisulisele analüüsile andmete puhastamise asemel. Platvormi praktilist väärtust demonstreeritakse näidete abil, sealhulgas kombineerides Transpordiameti liiklusloenduse andmed Maa-ameti geograafiliste ja Keskkonnaagentuuri ilmastikuandmetega ning sidudes Riigihangete registri hanked Äriregistri majandusaruannete andmetega. Need juhtumid illustreerivad, kuidas Avaandmeait võimaldab andmestike rikastamist läbi ristkasutuse.

Võtmesõnad:
Andmeinseneeria, andmeait, avaandmed

CERCS:
P175 Informaatika, süsteemiteooria



Joonis 1. Graafiline lühikokkuvõtte eesti keeles.

Estonian Open Data Warehouse

Abstract: Estonia has numerous open datasets published by different public sector agencies, but their impact is limited by fragmentation, inconsistent quality, and a lack of machine-readability. This thesis addresses these issues by designing and implementing a centralized open data analytics platform called Avaandmeait. The platform is built on a modern data lakehouse architecture, combining the flexibility of data lakes with the structured querying capabilities of data warehouses. It leverages technologies such as Apache Iceberg for a unified table storage format, Trino for distributed querying, Apache Airflow for workflow orchestration, and dbt for managing data transformations. By standardizing formats and consolidating previously siloed datasets into a single repository, the system enhances data quality, comparability, and accessibility, allowing analysts to focus on insights rather than data cleaning. The thesis demonstrates the platform's practical benefits through use cases, including combining traffic information from Transport Administration with geospatial data from the Land Board and meteorological data from the Environmental Agency, and linking procurement data from Public Procurement Register with economic data from the Business Register. These scenarios illustrate how Avaandmeait enables complex cross-domain analyses that were previously cumbersome to perform.

Keywords:
Data engineering, data warehouse, open data

CERCS:
P175 Informatics, systems theory

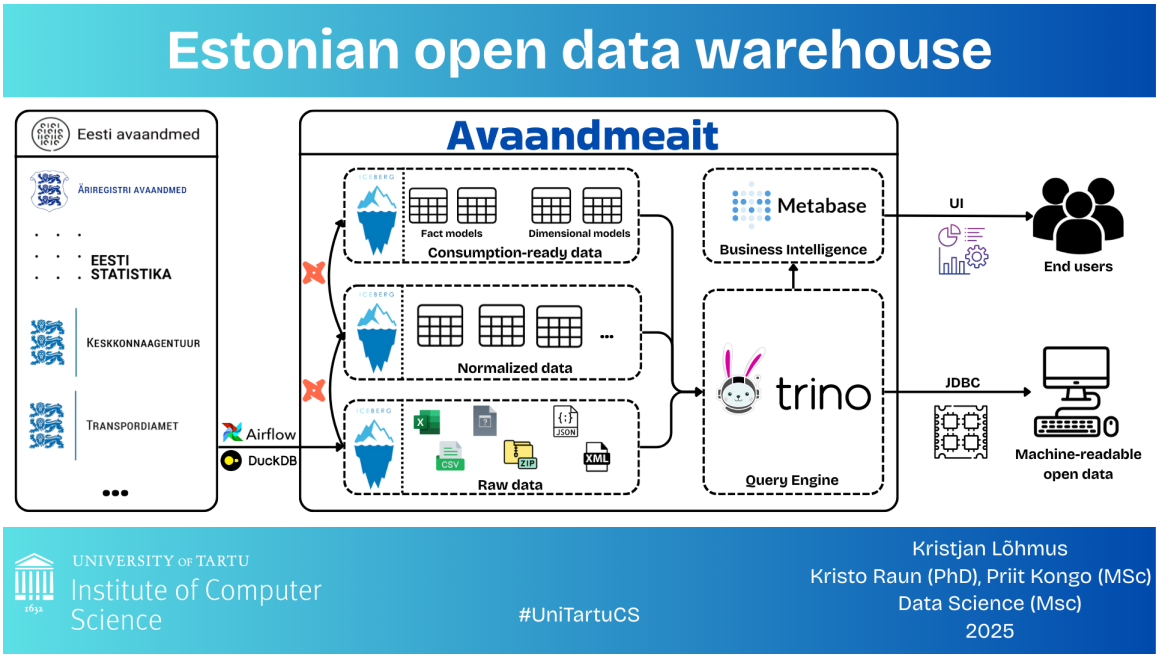


Figure 2. Graphical abstract in english.

Sisukord

1 Sissejuhatus

Avaliku sektori andmete vaba kasutamise idee on muutunud viimase kümnendi jooksul üha olulisemaks, kujunedes oluliseks teguriks Euroopa Liidu digipöörde ja andmemajanduse edendamisel [1]. Avaandmete turu väärtuseks ennustatakse 2025. aastaks 199.51 kuni 334.21 miljardit eurot [2]. Avaliku sektori teabe taaskasutamine soodustab uute teenuste, rakenduste ja ideede teket. Seda ilmestab ka Euroopa Liidu laiem strateegia andmepõhise majanduse arendamiseks: Euroopa Komisjon näeb avaandmetes märkimisväärset potentsiaali innovatsiooni soodustamisel, ettevõtete konkurentsivõime tõstmisel ja üldise ühiskondliku väärtuse loomisel [1]. Euroopa Liidu tasandil on kehtestatud mitmeid meetmeid ja suuniseid, et julgustada liikmesriike avaldama oma valduses olevat teavet võimalikult laialdaselt ja kvaliteetselt. Üheks olulisemaks õiguslikuks raamistikuks on Direktiiv (EL) 2019/1024 „Avaandmed ja avaliku sektori valduses oleva teabe taaskasutamine“ [3], millega seati eesmärk suurendada avaliku sektori andmete sotsiaal-majanduslikku mõju, eriti toetades idufirmasid ning väike- ja keskmise suurusega ettevõtteid.

Direktiiv sätestab, et liikmesriigid peavad oma avaliku sektori andmestikud tegema tasuta kättesaadavaks avatud litsentside alusel ja masinloetavates formaatides, et hõlbustada nende laialdasemat kasutuselevõttu. See nõue on osa Euroopa Liidu laiemast paketist, mis soodustab andmemajanduse kasvu. Paljud riigid on selle tulemusel oma seadusandlust ja praktikaid juba muutnud, kohandades olemasolevaid lahendusi uute nõuetega [4]. Avaandmete kui ühise hüve ärakasutamine toob märgatavat kasu nii avaliku sektori läbipaistvuse, erasektori uuenduslike rakenduste loomise kui ka teadusasutuste uurimistegevuse seisukohalt. Samas ei tähenda õigusliku raamistiku olemasolu veel automaatselt kvaliteetseid ja kergesti kasutatavaid andmeid.

Eesti on tõusnud Euroopa avaandmete küpsuse hindamistes keskmisest kõrgemale. Ometi tuuakse välja kitsaskohti andmete küpsuse ja kvaliteedi vallas [5]. Probleemid andmete formaadi, struktureerituse ja ühtsete standardite puudumisega takistavad avaandmetest loodavat lisandväärtust, olgu selleks siis uute digitaalsete teenuste arendamine või poliitika efektiivne kujundamine. Sellega piiratakse avaandmete ärilist ja teaduslikku potentsiaali, kuna ettevõtjad, riigiametnikud ja teadlased vajavad otsuste tegemiseks usaldusväärset ja kergesti kasutatavat infot. Kitsaskohtade lahendamine aitaks kiirendada andmemajanduse arengut, soodustada innovatsiooni ning toetada läbipaistvust ja andmepõhiseid otsuseid nii riigi kui ka erasektori tasandil.

Käesoleva lõputöö eesmärk on kavandada ja rakendada lahendus, mis tsentraliseerib Eesti avaandmed ning parandab nende kvaliteeti ja masinloetavust. Töö teises peatükis kaardistatakse avaandmete hetkeseis, hinnates andmekvaliteeti ja masinloetavust, tuues esile peamised kitsaskohad ning praktilised soovitused nende lahendamiseks. Kolmas peatükk pakub ülevaadet moodsatest analüütilistest andmesüsteemidest, mis sobivad avaandmete tsentraliseerimiseks. Neljandas peatükis kirjeldatakse loodud infrastruktuuri — kasutatud tehnoloogiaid ja nende omavahelist ühildumist. Viiendas peatükis keskendutak-

se süsteemi disainile, selgitades detailsemalt, kuidas andmed laetakse, transformeeritakse ja edastatakse lõpptarbijale. Kuuendas peatükis demonstreeritakse lahenduse väärtust näidiskasutusjuhtude kaudu, kus eri allikatest pärinevaid andmestikke ristkasutatakse.

2 Eesti Avaandmed

Eestil on erinevaid avaandmete hoiustamise ja avaldamise süsteeme. Nendest suurematteks on E-äriregister [1], Maa- ja Ruumiameti geoportaal [2] ning Statistikaameti andmebaas [3]. Neid ja teisi väiksemaid andmestikke koondab avaandmete portaal [4], millega kaardistatakse ja avaldatakse suurem osa andmetest.
Andmepõhise majanduse ja avaandmete kohta avaldab EL ka iga aasta avaandmete raporti, milles hinnatakse liikmesriikide avaandmete pakkumise ja kasutamise olukorda. Üle-euroopaline võrdlusuuring "Open Data Maturity Landscaping"mõõdab riikide edusamme nelja mõõdikuga:

- **Poliitika** – hinnatakse avaandmete õigusraamistikku, juhtimist ja rakendamist;
- **Portaal** – analüüsitakse avaandmete portaali funktsionaalsust, kasutajasõbralikkust, haldust ja jätkusuutlikkust;
- **Kvaliteet** – hinnatakse metaandmete täielikkust ja asjakohasust, andmete monitoorimist, DCAT-AP standardi rakendamise taset ning juurutuse kvaliteeti ja seotud andmete olemasolu;
- **Mõju** – käsitletakse strateegilist teadlikkust avaandmetest, nende taaskasutust ning saavutatud majanduslikku ja ühiskondlikku mõju.

Tabelis 1 on toodud raportis avaldatud Eesti avaandmete skoorid ja paigutus pingereas aastatel 2018 kuni 2024. 2018. aastal oli Eesti tulemus üks Euroopa viimaseid – Eesti üldine küpsusskoor oli vaid 44%, jäädes alla ELi keskmisele (65%) ning paigutudes 27. kohale “järgijate” ehk madala küpsusega riikide sekka. Samal aastal tellis Majandus- ja Kommunikatsiooniministeerium (MKM) Open Knowledge Estonia käest uuringu selgitamaks avaandmete hetkeseisu ja pakkumaks lahendusi olukorra parandamiseks. Uuringust selgus, et andmete avalikustamise vajadusse suhtusid riigiasutused leigusega, kuna juba eksisteeris X-tee, mis lubas teiste riigiasutustega lihtsasti andmeid vahetada. Lisaks puudus hea ülevaade saadavalolevatest andmestikest ja nende kvaliteedist, polnud võimalik otsest tagasisidet anda ning üldine avalik teadlikkus avaandmetest oli madal [6]. Uuringu tulemustest tehti järeldused ja järgneval aastal toimus oluline edasiminek – 2019. aastaks paranes Eesti avaandmete küpsus oluliselt (üldskoor 67%) ning riik tõusis edetabelis 14. kohale. 2020. aasta raportis saavutas Eesti esmakordselt “suunanäitaja” staatuse, platseerudes Euroopa riikide hulgas 5. kohale. Seda positsiooni hoiti kuni 2024. aastani, kui langeti 9. kohale, sest langesid portaali, mõju ja kvaliteedi hinnangud [7].
Käesolevas peatükis tutvustatakse avaandmete portaali kui ökosüsteemi keskset platvor-

[1] https://www.rik.ee
[2] https://geoportaal.ee/
[3] https://andmed.stat.ee
[4] https://avaandmed.eesti.ee

Tabel 1. Eesti paigutus Euroopa Liidu liikmesriikide avaandmete järjestuses [7].

Aasta	Poliitika	Portaal	Mõju	Kvaliteet	Koguskoor	Koht edetabelis
2018	550	310	135	105	1100	27.
2019	540	435	350	425	1750	14.
2020	645	570	630	516	2361	5.
2021	635	620	650	545	2450	5.
2022	612	609	600	530	2351	6.
2023	640	630	600	570	2440	4.
2024	640	605	580	535	2360	9.

mi ning selle rolli avalike andmete vahendamisel. Seejärel antakse ülevaade portaalis leiduvate andmestike olemusest – nende struktuuridest, formaatidest ning andmete avaldajatest. Lõpuks analüüsitakse peamisi probleeme, mis piiravad avaandmete tõhusat kasutamist, pöörates eraldi tähelepanu andmete sisuga seonduvatele väljakutsetele ja masinloetavusele.

2.1 Avaandmete portaal

Avaandmete portaal on Eesti avaandmete ökosüsteemi tuumik. See täidab kataloogi rolli, kuhu on koondatud ülevaade kogu avalikust andmevarast, muutes eri allikad kättesaadavaks ühest kohast. Portaali kaudu on tänaseks kasutusele võetud hulgaliselt andmeid, mis varem olid peidus asutuste sisesüsteemides või laiali eri veebilehtedel. Nii riigi kui ka kohalike omavalitsuste jaoks pakub portaal platvormi, kuhu oma andmeid üles laadida ning seeläbi täita avaliku teabe jagamise kohustust efektiivsemalt [8].
Portaal on integreeritud ka rahvusvahelisse andmevahetusse: selle metaandmeid korjab automaatselt Euroopa andmeportaal data.europa.eu, mis tähendab, et Eesti andmekirjed on leitavad ka üle-Euroopalises kataloogis, kus 06.04.2025 seisuga on ligi 2 miljonit andmestikku. Samuti vastab portaali arhitektuur Euroopa DCAT-AP andmestandardi nõuetele, mis on eelduseks riikidevahelisele andmevahetusele. Rakendusprofiilis on andmevahetuse hõlbustamiseks ja liikmesriikide portaalide ühilduvuseks määratletud klassid ja atribuudid, millega andmestikke kirjeldada [9]. Avaandmete portaali kasutajaliides ja funktsioonid on loodud selliselt, et andmete leidmine ja kasutamine oleks võimalikult mugav. Portaal pakub igale huvilisele mitmeid viise andmetega tutvumiseks: on olemas otsingu- ja filtreerimisvõimalused, mis lubavad andmekogumeid leida märksõnade, teemavaldkondade, asutuste või märksiltide alusel. Iga andmekogumi juures on metaandmed – kirjeldus, formaadid, uuendamise sagedus, avaldaja ja litsentsitingimused. Andmeid saab portaali kaudu alla laadida erinevates formaatides (nt CSV, JSON, XML, RDF jt, olenevalt andmest) või kasutada läbi API liideste. Portaalil on olemas masinloetav liides andmekogude metaandmete päringuteks – see tähendab, et arendajad

saavad programmiliselt portaalist andmeid pärida ja alla laadida. Lisaks pakub portaal andmete visualiseerimise võimalusi: tabelandmete eelvaateid, ruumiandmeid kaardil. RIA kinnitusel on teabevärava eesmärk lisaks andmete tarbimisele pakkuda ka visualiseerimise tööriistu ning tuua esile avaandmetel põhinevaid lugusid. Seetõttu kogub portaal ka kasutuslugusid – eraldi rubriikides tuuakse välja näiteid rakendustest või uuringutest, mis on tehtud avaandmeid kasutades.

2.1.1 Andmestike kirjeldus

2025. aasta 31. märtsi seisuga on avaandmete portaalis 5690 avalikku andmestikku. Iga andmestikku iseloomustavad metaandmete väljad koos nende selgitustega on välja toodud tabelis 2. Põhiväljadeks igal andmestikul on identifikaator, nimi, kategooria, kirjeldus, looja, haldaja, kättesaadavus, terviklikkus, uuendamise sagedus ja viimase uuendamise aeg. Lisaväljadena kirjeldavad andmestikke erinevad ruumilised ja ajalised väljad ning võtmesõnad.

Väli	**Kirjeldus**
id	unikaalne identifikaator
name	Andmestiku nimi (name_et ja name_en eesti- ja inglisekeelseteks nimedeks)
description	Andmestiku kirjeldus (description_et ja description_en eesti- ja inglisekeelseteks kirjelduseks)
created_at	Andmestiku loomise aeg
published_at	Andmestiku avaldamise aeg
translated_at	Andmestiku tõlkimise aeg
updated_at	Andmestiku viimane uuendamise aeg
to_be_deleted_at	Andmestiku kustutamise aeg tulevikus
user	Andmestiku lisanud kasutaja
organization	Andmestiku lisanud kasutaja organisatsioon
maintainer	Andmestiku haldaja
citations	Viited andmestiku algallikatele
files	Andmestikuga seotud failid
conformities	Andmestiku puudutavad määrused
south_latitude, south_latitude, west_longitude, east_latitude	Kaardiandmestike ruumilised piirangud
language	Andmestiku keel
data_from, data_to	Andmete ajaline algus ja lõpp
update_interval_unit	Andmete uuenemissageduse ühik (päev, nädal, kuu, aasta)

update_interval_frequency	Andmete uuenemissagedus (mitu korda ühikus)
access	Andmestikku kättesaadavus (avalik või privaatne)
available_to	Andmestiku kättesaadavuse ajaline piirang
landing_page	Lauter
was_generated_by	Andmestikku genereerija
geoportal_identifier	Geportaalis andmestiku identifikaator
geoportal_keywords	geportaalis andmestiku võtmesõnad
lineage	Andmestikuga seotud teised andmestikud (on osa või teine andmestik on selle osa)
pixel_size	Kaardiandmete pikslisuurus meetrites
resource_type	Kaardiandmete allika liik (andmestik või teenus)
topic_categories	Kaardiandmete kategooria
spatial_resolution_type	Ruumilise esituse tüüp
spatial_data_service_type	Kaardiandmete teenuse tüüp (allalaadimine, vaade)
temporal_resolution	Kaardiandmete uuendamise vahemik
maturity	Andmestiku terviklikkus (kas on valmis või mitte)
version	Andmestiku versioon
version_notes	Märkmed versiooni kohta
is_actual	Andmestikku aktuaalsus
sync_id	Andmestiku sünkroneerimistöö ID
sync_date_stamp	Andmestiku viimase sünkroniseerimise hetk
licence	Andmestiku kasutamise litsents
coordinate_reference_system	Kaardiandmete koordinaatsüsteem
keywords	Andmestikuga seotud võtmesõnad
parent_datasets	Andmestik on osa nendest andmestikest
child_datasets	Andmestikud on osa sellest andmestikust
map_regions	Ruumiandmestiku kaetavad alad
is_content_allowed	Piiratud ligipääsuga andmestik

Tabel 2. Avaandmestikke kirjeldavad väljad

Avaandmete portaalis on registreeritud 2232 andmevaldajat ehk organisatsiooni, kes omavad mingisuguseid andmeid. Siia kuuluvad kõik ministeeriumid, ametid, kohalikud omavalitsused, haridusasutused, riigiasutuste allasutused ja ka hulk era- ja vabasektori organisatsioone, kes on end andmevaldajaks registreerinud. Neist 104 (ligikaudu 4.7%) on avaldanud vähemalt ühe andmestiku. Suurim avaandmete jagaja on Statistikaamet, kellele kuulub ligi 80% avaldatud andmestikest. Enim andmestikke avaldanud organisatsioonid on toodud tabelis 3.

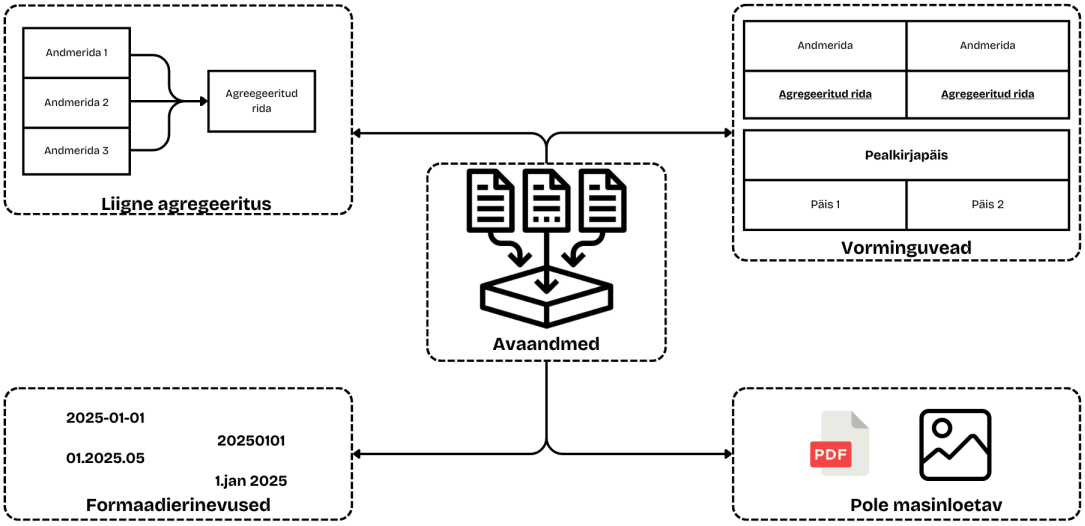
Teabevaldaja	Andmestike arv
Statistikaamet	4448
Maa-ja ruumiamet	288
Eesti Keele Instituut	179
Tartu Ülikool	94
Regionaal- ja põllumajandusministeerium	93

Tabel 3. Rohkeimate andmestike valdajad avaandmete portaalis.

Andmed ise on kaasa pandud failidena või on viidatud andmete algallikale veebis, kust neid võib leida. Algallikatele viidetega andmestikke on 5616 (98.7%) ja otse portaali failidest on avaandmestikest kätte saadavad 74 (1.3%) andmestikku 229 erineva failina. Andmestikke on failideks jaotatud erinevalt – osa on partitsioneeritud ajaliselt, et vältida ühe faili liiga suureks paisumist, nagu näiteks Transpordiameti liiklusloenduse andmestik, kus on iga aasta liiklusloenduse andmed eraldi failis. Teised andmestikud on jaotatud failideks tunnuste järgi, nagu näiteks Töötukassa registreeritud töötute andmestik, kus on 13 erinevat faili kirjeldamaks töötute statistikat maakonniti, vanuseti, haridustaseme ja muude tunnuste järgi.

2.1.2 Andmestike probleemid

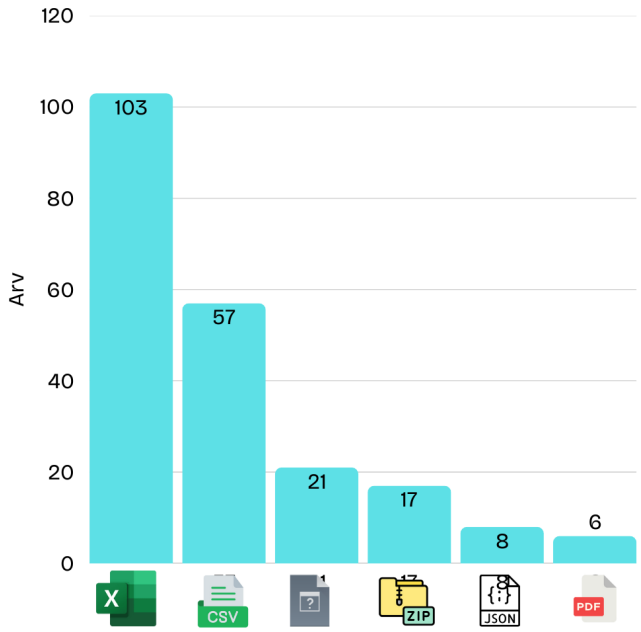
Avaandmete kogumi loomisel tekivad olulised väljakutsed andmestike masinloetavuse, formaatide, sisu ja vorminduse tõttu.
Esimeseks probleemiks on failiformaatide masinloetavus. Euroopa Liidu avatud andmete direktiivis on „masinloetav vorming" defineeritud järgmiselt: „ .. failivorming, mis on struktureeritud selliselt, et tarkvararakendused suudavad spetsiifilisi andmeid, sealhulgas üksikuid faktiväiteid, ja nende sisemist struktuuri kergelt tuvastada, ära tunda ja teha andmetest väljavõtteid [3]". Näiteks on masinloetavad formaadid nagu CSV, JSON või XML, kus andmete struktuur on selgelt määratletud, võimaldades tarkvaral sealt andmeid hõlpsalt kätte saada. Seevastu vormingud nagu skaneeritud pildid või PDF-failid, kust andmeid ei saa masinaliselt lihtsasti välja lugeda, ei kvalifitseeru masinloetavate andmetena [10]. On ka vahepealseid formaate, mille masinloetavus sõltub andmete struktuurist. Nendest on kõige levinum Excel, kust on võimalik andmeid lugeda, kuid väga palju oleneb andmete struktuurist. Avaandmete portaalis leidub faile kolmeteistkümnes erinevas formaadis, millest suurem osa on Exceli või komaeraldusega väärtuste (CSV) formaadis, mõned kokkupakituna zip failideks (enamasti keeletekstid) või JSON formaadis keerukama struktuuriga andmed. Leiduvad ka portaalile tundmatu formaadiga failid (application/octet-stream), milleks on näiteks .md lõpuga markdown, .jl formaadis Julia ja .tmx formaadid. Koheselt masinloetavad on failidest 29% ehk CSV, JSON ja laiendatava märgistuskeelega (XML) formaadis failid. Exceli failidest tuli hilisema töö käigus välja, et umbes pooled neist pole mõistliku vaevaga masinloetavad, kuna andmete



Joonis 3. Avaandmete portaalis leiduvate andmestike probleemid.

struktuur polnud korrektne. Histogramm avaandmete portaalist enimleiduvatest failitüüpidest on toodud joonisel 4 ja nende jaotumisest masinloetavuse järgi joonisel 5.
Teiseks suuremaks probleemiks on andmestike sisu. Andmekaitse seadusega on kehtestatud, et avalikustada ei tohi andmeid, millega on võimalik tuvastada üksikisikuid [11]. Esimeseks variandiks isikuandmeid kaitsta on eemaldada kõik otseselt või kaudselt isikut tuvastada võimaldavad andmed. Näitena anonüümimisest saab välja tuua Transpordiameti juhilubade andmestiku [5], kus Transpordiamet on teinud kättesaadavaks detailsema andmestiku sõidueksamite tulemuste kohta, milles iga eksami kirje on pseudonüümseks muudetud kujul olemas (sisaldades näiteks eksami aastat ja kuud, asukohta, eksamineerija koodi jms, aga mitte eksamineeritava isikuandmeid). Näidis sõidueksamite andmetest on leitav tabelist 4. Sellisel kujul andmestik (kokku üle 118 tuhande sõidueksami kirje) võimaldab läbi viia põhjalikumaid analüüse – näiteks tuvastada tegureid, mis mõjutavad eksami edukust. Teiseks isikuandmete kaitsmise võimaluseks on agregeerimine, kus võetakse mõned tunnused ja väljastatakse nende omavaheliste seoste summad. See aga piirab analüüsi võimalusi, kuna võivad välja jääda teised tunnused, mis aitaks andmeid paremini mõista ja analüüsida. Näiteks ei saa agregeeritud andmete puhul teha mikrotasandi seoseanalüüsi ega tuletada individuaalseid käitumismustreid, mis paljude rakenduste ja teadusuuringute jaoks on vajalik [12]. Näiteks agregeerimisest on Riigikohtu põhimenet-

[5] https://avaandmed.eesti.ee/datasets/toimunud-soidueksamid-eestis

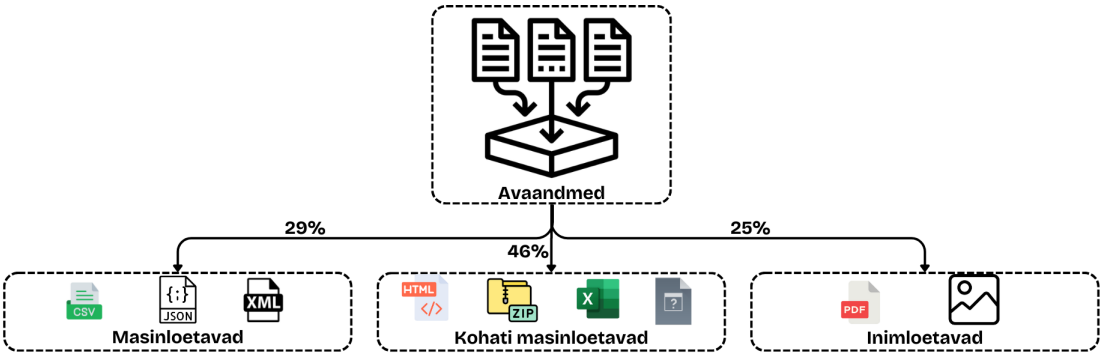


Joonis 4. Avaandmete portaalis enimleiduvate failitüüpide histogramm.

luste statistika [6], mida avaldatakse avaandmetena üksnes koondtulemustena aasta lõikes. Andmestik sisaldab näiteks 2021. aastal Riigikohtu menetletuste arvu liikide kaupa, kuid puuduvad kirjed üksikjuhtumite kohta. Ühest küljest on hea, kui mingi statistika on juba ära tehtud, kuid kui andmete tarbija sooviks teha ise mingit statistikat – näiteks andmekaitsega seotud lahendite kohta – siis hetkel avaldatud andmestiku põhjal ei ole see võimalik. Kõikide riigikohtu lahendite info on kättesaadav Riigiteataja lehelt kättesaadav [7] ja tuleks statistika tegemisel kaasa panna toorandmetena.

Soovitatavalt võiks andmevaldajad leida tasakaalu andmekaitse ja andmete kasulikkuse vahel. Euroopa andmekaitsemäärus (GDPR) võimaldab kasutada andmete pseudonümiseerimist või anonümiseerimist, et muuta isikuandmed avaandmetena avaldamiseks sobilikuks. Eesti avaandmete juhistes rõhutataksegi, et kui andmestikus leidub üksikandmeid, millele kehtiks muidu juurdepääsupiirang, tuleks need pseudonümiseerida või anonüümida, et andmestiku saaks siiski avalikuks teha [13]. See tähendab praktikas, et andmekirjetest eemaldatakse nimed, isikukoodid ja muud otsesed identifikaatorid ning vajadusel asendatakse need koodidega. Samuti võib rakendada täiendavaid privaatsuskaitse tehnikaid – näiteks agregeerimist või sünteetiliste andmete genereerimist – vähenda-

[6] https://avaandmed.eesti.ee/datasets/riigikohtu-pohimenetluse-statistika-2021
[7] riigiteataja.ee/kohtulahendid/koik_menetlused.html



Joonis 5. Avaandmete jaotus masinloetavuse järgi.

EKSAMI_SOORITAJA	KUUPAEV	BYROO	EKSAMINEERIJA	SEISUND
578cb547abb8c476	2025-01	Tallinn	01be6f2aa5cba2cd	MITTE_SOORITATUD
136c09cbc796f60f	2025-01	Tallinn	01be6f2aa5cba2cd	MITTE_SOORITATUD
24cbf537f0653ea8	2025-01	Paide	213d8a3fa533a2ce	MITTE_SOORITATUD
8779a924392d4c75	2025-01	Haapsalu	12ed0758e403a7d6	MITTE_SOORITATUD
d4f85e337149c34c	2025-01	Paide	844452b4e4bac2ef	MITTE_SOORITATUD

Tabel 4. Näidisandmed Transpordiameti sõidueksamite tulemuste andmestikust.

maks taasidentifitseerimise riski isegi andmete ristkasutuse korral. Nende meetodite rakendamine võimaldaks avaldada detailsemaid andmeid nii, et üksikisikute privaatsus jääb kaitstuks. Kokkuvõtlikult on andmestike olemusest tulenev põhiprobleem dilemma, kuidas tagada andmete väärtusliku detailsuse säilimine ilma isikute õigusi rikkumata – lahenduseks on robustne anonüümimine, mis lubab avaandmetena pakkuda võimalikult rikka sisuga andmestikke.

Kolmas suur väljakutsete blokk on seotud andmestike tehnilise struktuuri ja vorminguga. Isegi juhul, kui andmed on sisuliselt avalikustamiseks sobilikud, võivad kehvad vormistusvõtted muuta nende masinlugemise vaevaliseks. Avaandmete ideaal on, et need oleksid selge struktuuriga ja automaatselt töödeldavad, kuid reaalsuses seda nõuet tihti ei täideta [14]. Levinud probleem on andmete esitamine inimesele loetaval kujul viisil, mis pole masinloetav. Näiteks avaldatakse andmestik Exceli failina, mis esmapilgul on struktureeritud tabel, kuid sisaldab mitut päise rida, ühendsalve või vorminduslikke elemente (värvid, ridade vahetihedused jms), mis pole masinale loetavad. Selline andmestik on küll tehniliselt avatud vormingus, ent nõuab kasutajalt märkimisväärset eeltöötlust, et see muutuks analüüsikõlbulikuks. Juba varasemast on näiteid, kus masinloetavaks peetav Exceli formaadis andmestik vajas palju käsitsitööd enne, kui sellest sai struktureeritud andmestik [15] – probleemid tekkisid näiteks kuupäeva veergude ebaühtlase vormingu

tõttu ning tulenevalt sellest, et tabelites oli mõeldud pigem visuaalse esituse kui masinloetavuse peale. Samuti esineb juhtumeid, kus andmeid jagatakse PDF-formaadis või HTML-lehtedel, mis on mõeldud inimesele lugemiseks; selliste failide automaatne lugemine on väga keeruline. Kui andmekogu pakutakse ainult inimloetaval kujul, siis pole avaandmete põhimõte – masinloetav vabakasutus – sisuliselt täidetud.
Struktuuriprobleemide hulka kuulub ka standardiseeritud vormingu puudumine. Igas andmestikus võivad samad väärtused olla esindatud eri viisil, mis takistab andmekogude ristkasutust ja integratsiooni. Näiteks võivad erinevad avaldajad kasutada erinevaid vorminguid ja tähistusi: ühes andmestikus on kuupäevad formaadis YYYY-MM-DD, teises aga tekstina (nt "1. jaanuar 2025"); ühes andmestikus on maakonnad märgitud täispikkade nimedega, teises aga lühenditega [16]; või on ühes andmestikus märgitud koordinaadid GCS süsteemis ja teises EPSG:3301 süsteemis. Standardstruktuuride puudumine sunnib andmekasutajaid iga konkreetse andmestiku jaoks eraldi käsitsi andmestruktuuri lahti mõtestama ja ümber teisendama, mis on aeganõudev ning aldis vigadele.
Samuti kohtab probleeme, kus andmestik sisaldab inimliku loetavuse parandamiseks tehtud kõrvalekaldeid struktureeritud vormist. Näiteks võib Exceli failis olla esimese rea asemel pikk pealkiri või selgitav tekstilõik, tegelikud veerupäised aga alles mitmendal real. Samuti lisatakse tihti tühje ridu või vahepealkirju, et inimesele rühmitada andmeid loogiliselt (nt kvartalite kaupa). Need elemendid teevad masinliku lugemise raskemaks, sest programmid eeldavad reeglina, et tabeli struktuur algab kohe esimesest reast, kus on kirjas veergude nimed. Riigieelarve täitmise andmestikud (näidis joonisel 6) on üheks näiteks, kus andmed esitatakse küll tabelina, kuid sisaldavad mitmeid inimesele suunatud struktuurielemente – summade vahekokkuvõtted jaotiste lõpus, mitmerealised päised erinevate eelarveosade jaoks jmt. Masin, mis sellist faili loeb, peab need mittestandardseid read eraldi käsitlema või välja filtreerima. Kui puuduvad ühtsed reeglid, kuidas andmeid struktureerida (nt CSV vormingus, kus iga veerg on fikseeritud tähendusega ja esimesel real on veerunimed), siis iga andmestiku kasutuselevõtt nõuab eraldi kodeerimist ja eeltööd.
Kokkuvõttes raskendavad andmestike struktuursed puudujäägid avaandmete automatiseeritud töötlemist ja analüüsi. Kui andmed pole esitatud selgel ja ühtsel kujul, kaob suur osa avaandmete eelistest – nende praktiline kasutamine nõuab liigselt käsitööd ja tehnilist vaeva. Selliste probleemide lahendamiseks võiks andmevaldajad järgida parimaid tavasid: esitada avaandmed võimalusel lihtsas tabelilises vormis (nt CSV), vältida mitmerealisi päiseid ja muud mitmeselt mõistetavat struktuuri, dokumenteerida andmeväljade tähendused ning kasutada rahvusvahelisi standardkoode (nt kuupäevade jaoks ISO 8601 vormingut). Nagu avaandmete põhimõtetes rõhutatud [13], peab avaldatud andmestik olema masinloetav ehk ühtlase ja arusaadava struktuuriga – selle saavutamine eeldab, et tehnilised detailid nagu failivorming, kodeering, veerunimed ja andmetüübid on läbimõeldud ja kooskõlas üldiste standarditega.

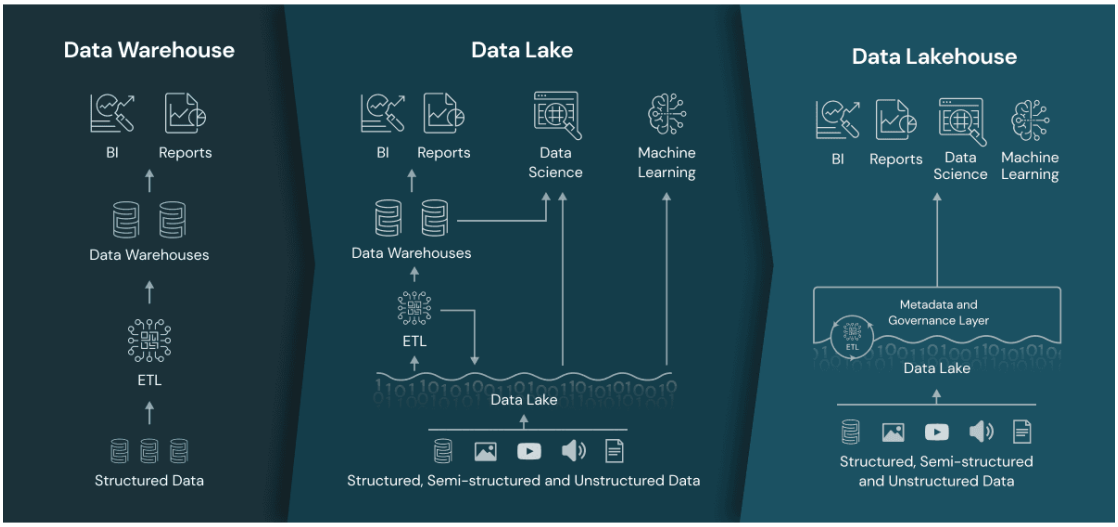
2020. aasta riigieelarve tulud ja kulud
(eurodes)

	täitmine jaanuar 2020	täitmine veebruar 2020	täitmine märts 2020	täitmine aprill 2020	täitmine mai 2020	täitmine juuni 2020
Riigieelarve tulud	**839,608,796**	**691,554,764**	**696,988,951**	**776,295,979**	**832,213,962**	**1,031,893,411**
MAKSUTULUD	**784,851,768**	**646,476,238**	**593,261,232**	**713,374,745**	**771,983,589**	**819,647,936**
Tulumaks	**220,537,733**	**117,536,449**	**49,756,061**	**158,735,034**	**177,641,421**	**190,079,132**
Füüsilise isiku tulumaks	162,858,177	96,707,851	22,468,089	124,131,414	147,813,932	147,404,300
Juriidilise isiku tulumaks	57,679,556	20,828,598	27,287,972	34,603,620	29,827,489	42,674,832
Sotsiaalmaks ja sotsiaalkindlustusmaksed	**316,448,529**	**313,865,462**	**286,308,764**	**307,543,344**	**312,446,358**	**324,960,970**
Sotsiaalmaks	281,349,214	281,733,922	254,252,556	277,569,336	280,558,937	292,762,262
Töötuskindlustusmakse	19,412,863	17,546,409	17,489,831	16,251,418	17,307,705	17,546,255
Kogumispensionimakse	15,686,451	14,585,131	14,566,377	13,722,590	14,579,716	14,652,453
Omandimaksud	**1,292,595**	**-228,303**	**26,709,826**	**4,950,218**	**1,021,820**	**855,052**
Raskeveokimaks	1,190,020	56,646	16,996	1,014,883	102,956	62,788
Maamaks	102,575	-284,949	26,692,830	3,935,335	918,863	792,264
sh riigisisene maamaks	*-395*	*-95*	*584,055*	*516,398*	*6,359*	-
Maksud kaupadelt ja teenustelt	**242,810,194**	**211,513,438**	**226,903,136**	**238,128,000**	**277,592,186**	**300,515,491**
Käibemaks	190,061,222	156,426,399	159,254,532	169,711,415	196,140,421	209,362,599
sh riigisisene käibemaks	*8,815,058*	*8,593,786*	*15,192,166*	*10,466,987*	*14,209,248*	*14,249,088*
Aktsiisid	49,348,948	51,981,572	65,536,088	66,321,085	79,516,241	89,413,409
Alkoholiaktsiis	14,488,888	12,826,273	16,416,817	12,987,694	16,254,393	25,033,220
Tubakaaktsiis	15,871,663	15,887,785	18,234,941	17,803,176	19,586,742	22,906,704
Kütuseaktsiis	15,672,590	20,038,597	27,444,044	32,819,247	43,053,094	40,902,227
Pakendiaktsiis	17,883	15,491	121,128	18,006	14,585	18,320
Elektriaktsiis	3,297,924	3,213,426	3,319,158	2,692,961	607,427	552,939
Hasartmängumaks	2,720,879	2,784,329	1,791,137	1,657,769	1,909,905	1,540,946
Muud kohalikud maksud	679,146	321,138	321,380	437,731	25,619	198,537
Maksud väliskaubanduselt ja-tehingutelt	**3,762,717**	**3,789,191**	**3,583,444**	**4,018,149**	**3,281,804**	**3,237,291**
Tollimaks	3,762,717	3,789,191	3,583,444	4,018,149	3,281,804	3,237,291
MITTEMAKSULISED TULUD	**54,757,028**	**45,078,526**	**103,727,720**	**62,921,235**	**60,230,373**	**212,245,475**
Kaupade ja teenuste müük	**26,584,716**	**23,662,651**	**21,576,023**	**17,856,110**	**16,092,388**	**21,531,886**
Riigilõivud	8,960,992	6,502,345	5,670,715	4,186,716	5,288,740	6,082,049
Tulu majandustegevusest	3,570,378	3,327,680	3,010,145	1,956,259	2,296,239	2,886,174
Muu kaupade ja teenuste müük	14,053,345	13,832,626	12,895,163	11,713,135	8,507,409	12,563,664
Saadud toetused	**20,540,133**	**13,608,328**	**58,039,942**	**21,235,941**	**37,740,725**	**92,876,258**
Välistoetus	19,890,125	12,818,718	57,434,354	20,328,150	36,983,324	91,376,004
Sisetoetus	650.007	789.609	605.588	907.791	757.401	1.500.254

Joonis 6. Lõik riigieelarve täitmise andmetest 2020.

3 Analüütilised andmesüsteemid

Üheks olulisimaks varaks igas organisatsioonis on tema andmed. Suurandmete ajastul on kasvanud nii struktureeritud kui ka struktureerimata andmete hulk plahvatuslikult ning üha enam organisatsioone teevad oma strateegilisi otsuseid andmepõhiselt [17]. Seetõttu käsitletakse andmeid üha enam strateegilise varana, mille tõhus haldamine ja analüüs võimaldab saada väärtuslikke teadmisi ning luua uut äri- ja innovatsioonivääртust [18]. Et organisatsioonid saaksid andmetest maksimaalset kasu, on tarvis sobivaid infosüsteeme. Andmete kogumine on küll hädavajalik, ent veelgi olulisem on nende kättesaadavus, kvaliteet ning õigeaegne kasutamine otsuste tegemisel [19]. Järgnevalt antakse ülevaade operatiiv- ja analüütiliste infosüsteemide erinevustest ning nende põhjal kujunenud andmearhitektuuridest (andmeaitadest, andmejärvedest ja andmejärvemajadest).



Joonis 7. Andmeaida, -järve ja -järvemaja arhitektuurid [20].

3.1 Analüütiliste süsteemide eripära

Tavapäraselt jagunevad organisatsioonide andmesüsteemid kaheks: operatiivseteks ja analüütilisteks. Operatiivsed infosüsteemid tegelevad igapäevaste tehingute töötlemisega. Neid nimetatakse OLTP-süsteemideks (*Online Transaction Processing*) ning nad haldavad näiteks tellimuste, maksete, kliendiandmete jms sisestamist ja ajakohastamist [21]. OLTP-süsteemides on andmebaasid üldjuhul normaliseeritud, et vältida andmeduplikatsiooni ja tagada tehingute terviklikkus. Iga operatiivne tehing on väike, lühiajaline ja puudutab ainult mõnda kirjet, kuid nõuab ranget korrektset töötlemist. Oluline on, et kõik

tehingud vastaksid ACID-omadustele – atomaarsusele, konsistentsusele, isoleeritusele ja püsivusele [22] – mis tagavad andmebaasi järjepidevuse ja veakindluse. Peamine jõudlusmõõdik OLTP puhul on tehingute läbilaskvus (tehingute arv ajaühikus), sest süsteem peab suutma teenindada samaaegselt paljusid kasutajaid ja päringuid reaalajas [23]. Eestis on operatiivseid süsteeme mitmeid, nagu näiteks Karistusregister, Digilugu või Päästeameti väljakutsete süsteem.

Analüütilised infosüsteemid seevastu on suunatud läbi ajalooliste andmete analüüsi juhtimisotsuste toetamiseks. Tüüpiline analüütiline keskkond on andmeait, mis võimaldab teostada keerukaid päringuid suurtele andmemahtudele. Seda tuntakse OLAP-süsteemina (*Online Analytical Processing*). OLAP süsteemide päringud on sageli ad-hoc iseloomuga ja hõlmavad tabelite liitmisi ning agregatsioone paljudelt andmeüksustelt. Erinevalt OLTP-st on siin tähtsam päringute läbivusaeg ja tulemuste saamise kiirus kui üksiktehingu töötlemise kiirus. Kuna analüütilised päringud võivad hõlmata andmeid mitme aasta või kogu organisatsiooni tegevuse kohta, ulatuvad analüütiliste andmehoidlate mahud tavaliselt gigabaitidest terabaitideni – suurusjärgu võrra mahukamad kui operatiivandmebaasid [23]. OLAP-süsteemides kasutatakse tihti denormaliseeritud andmemudeleid (nt täht- või lumehelbemudelid), mis on optimeeritud lugemispäringute kiirendamiseks [24]. Samuti integreeritakse andmeaita andmeid mitmest allikast, et luua organisatsiooniülene "ühtne tõeallikas", mis väldib info killustatust [24].

Lihtsustatult võib öelda, et operatiivsetesse süsteemidesse pannakse andmed sisse ja analüütilistest süsteemidest võetakse andmed välja. OLTP ja OLAP süsteemide erinevuste tõttu ei ole otstarbekas teha mahukaid analüütilisi päringuid otse operatiivsetest andmebaasidest – see koormaks operatiivsüsteeme liigselt ning aeglustaks tehingute töötlemist. Seetõttu on välja kujunenud eraldatud andmeanalüüsi keskkonnad, kus operatiivandmetest koostatakse koondandmestik hilisemaks analüüsiks.

Hea analüütiline andmesüsteem peab vastama mitmele olulisele nõudele, et pakkuda organisatsioonile maksimaalset väärtust. Peamised kvaliteedikriteeriumid Kimballi järgi on järgmised:

- **Informatsioon peab olema kergesti kättesaadav** ehk andmed peavad olema intuitiivsed kasutajale, mitte ainult arendajale.
- **Informatisioon peab olema järjepidev** ehk andmed peavad olema usaldusväärsed ja ilma lünkadeta.
- **Süsteem peab olema muutlik**. Kuna kasutajate nõuded, andmed ja tehnoloogia on muutlikud, siis peab analüütiline süsteem olema võimeline nende muutustega kohanema. Olemasolevad süsteemid ei tohiks muutuda, kui kasutaja uurib andmeid uuel viisil või lisatakse uusi andmestikke.
- **Andmed peavad olema ajakohased** ehk vananenud andmete põhjal on keeruline otsuseid teha ja seetõttu on hea, kui analüütilise süsteemi andmed peegeldavad operatiivsüsteemides toimuvat võimalikult reaalajas.

Analüütilise süsteemi arendamine on võrreldav ajakirja avaldamisega, kus peatoimetaja peab mõistma, kes on ajakirja lugejad, mida neile lugeda meeldib ja kuidas panna neid ajakirjast loetut usaldama [24].

3.2 Andmeait

Esimene populaarseks saanud analüütilise süsteemi formaat oli andmeait (*data warehouse*). Andmeaidad (andmelaod) baseeruvad relatsioonilistel andmebaasidel, kasutades tabeleid ning seeläbi garanteerides andmete atomaarsuse, konsistentsuse, isoleerituse ja püsivuse ehk ACID-omadused [22]. Bill Inmon on defineerinud andmeaida kui "teemapõhise, integreeritud, ajas muutuva ja püsiva andmekogu, mis toetab organisatsiooni otsustusprotsessi" [25]. Andmeaidas on andmed organiseeritud viisil, mis toetab tõhusat pärimist ja aruandlust: sageli kasutatakse dimensioonilist mudelit (fakti- ja dimensioonitabelid), mis lihtsustab ärianalüütikute jaoks päringute koostamist [24]. Enne andmeaida täitmist läbivad andmed töötluse, mille käigus need puhastatakse vigadest, muundatakse ühisesse vormingusse ning rikastatakse kontekstiga. Andmeaita salvestatud andmed on ajaloolised ning tüüpiliselt mittemuudetavad – uusi andmeid lisatakse regulaarselt (näiteks igapäevaselt või -nädalaselt), kuid varasemaid kirjeid tavaliselt ei uuendata, vaid vajadusel säilitatakse parandused eraldi kannetena. See tagab ajaloolise "mälu" säilimise. Andmeait peab tagama ka head jõudlusnäitajad mahukate päringute teenindamiseks; seetõttu kasutatakse tihti indekseerimist, eel-agregeerimist ja muid tehnikaid päringute kiirendamiseks [23].
Andmeaitade kasutegur on laialdaselt tõestatud – need on olnud edukalt kasutusel paljudes valdkondades (tervisesektor [26], tootmine [27], finantssektor [28], telekom [29] jm) pakkudes paremat ülevaadet äritegevusest. Samas on traditsioonilistel andmeaitadel ka piiranguid. Esiteks on andmeait enamasti struktureeritud andmete jaoks; uute andmetüüpide (näiteks pildid, logifailid, sotsiaalmeedia andmed) integreerimine võib olla keeruline. Teiseks võib väga suurte andmemahtude puhul andmeaida lahendus muutuda kulukaks nii salvestusmahu kui ka arvutusressursside poolest. Sellised väljakutsed sillutasid teed uuele lähenemisele – andmejärvedele [23].

3.3 Andmejärv

Andmejärve (*data lake*) kontseptsiooni tutvustas 2010. aastate alguses Pentaho tehnoloogiajuht James Dixon, kes kasutas "andmejärve" metafoori rõhutamaks erinevust traditsiooniliste andmeaitadega. Kui andmeait on nagu filtreeritud ja pudelitesse villitud vesi (andmed on struktureeritud kindlal eesmärgil), siis andmejärv sarnaneb looduslikule järvele, kuhu voolab kokku mitmesugust päritolu töötlemata vett [30]. Andmejärv on suure mahuga hajussalvestus, kuhu kogutakse kõik toorandmed nende originaalses vormingus [31]. Tavaliselt kasutatakse andmejärvede realiseerimiseks hajusfailisüsteeme

või odavaid pilvepõhiseid objektisalvestusi (nt Hadoop HDFS [8], Amazon S3 [9] jmt), mis võimaldavad salvestada nii struktureeritud kui ka struktureerimata andmeid. Andmejärve peamine eelis on paindlikkus: andmeid ei pea enne salvestamist transformeerima ega struktureerima, struktuur loetakse alles päringu ajal (*schema-on-read*). See teeb andmete allalaadimise ja talletamise väga kiireks ning võimaldab organisatsioonil säilitada potentsiaalselt kasulikku infot isegi enne, kui on selge, milleks seda täpselt vaja võib minna. Näiteks võib andmejärve koguda toorfailid sensoritelt, logifaile, pildi- ja helifaile jms, et neid hiljem vajadusel analüüsida. Siiski kaasneb andmejärvedega ka väljakutseid. Kuna andmeid säilitatakse nende algkujul ja minimaalsete eeltöötlustega, lasub suurem vastutus andmete korrastamisel ja integreerimisel tarbijate (analüütikute, andmeteadlaste) õlul. Ilma piisava halduse ja metaandmeteta võib andmejärv muutuda "andmesooks" (*data swamp*), kus andmete leidmine ja mõistmine on keeruline. Seetõttu rõhutatakse andmejärvede juures tugevat andmehaldust ja kataloogide (metaandmete) olulisust. Sellest hoolimata on andmejärved osutunud väärtuslikuks suurandmete ajastul, pakkudes skaleeritavust ja võimekust talletada erinevat tüüpi andmeid. Andmejärved ja andmeaidad teenivad osaliselt sarnast eesmärki – koondada andmed analüüsi jaoks – kuid erinevate meetoditega. Praktikas kasutatakse neid tihti kõrvuti: toorandmed hoitakse esmalt andmejärves ning seejärel valitud ja töödeldud andmed liigutatakse andmelao struktuuri, et pakkuda stabiilsemat ja kiiremat päringukeskkonda kriitilistele aruannetele [32]. Selline lahendus aga tähendab, et organisatsioon peab haldama kahte eraldiseisvat andmeplatvormi ning seda, et andmed on suures osas dubleeritud.

3.4 Andmejärvemaja

Uuema põlvkonna andmearhitektuurina on välja kujunenud andmejärevemaja (*data lakehouse*), mis püüab ühendada andmeaida ja andmejärve parimad omadused ühtseks platvormiks [31]. Andmejärvemaja kontseptsioon on saanud alguse 2020. aastate paiku vastusena praktilisele vajadusele lihtsustada andmete ökosüsteemi – hallata nii struktureeritud kui ka struktureerimata andmeid ühes keskses keskkonnas ilma topeltandmekogumiteta. Andmejärvemaja lähenemine ühendab andmeaitade ja andmejärvede tugevused, võimaldades korraga nii suurte kiirustega toorandmete sisse laadimist kui ka nende kiiret töötlemist ja analüüsi ühtses süsteemis. Teisisõnu pakub andmejärvemaja arhitektuur andmejärve paindlikkust, kuid lisab sellele andmeaidale omased andmehalduse mehhanismid (andmete kvaliteedikontroll, struktuuri rangem rakendamine, ACID-toega tehingukihid). Järvemaja on kontseptuaalselt keskne andmeplatvorm. Erinevalt klassikalisest andmejärvest on andmejärvemaja puhul rõhutatud ka andmete struktuuri ja kvaliteedi tagamist. Andmejärvemajas rakendatakse andmetele metaandmete kihte, ka-

[8] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[9] https://aws.amazon.com/s3/

sutades avatud tabeliformaate, millest tuntumad on Delta Lake [10], Apache Iceberg[11] ja Apache Hudi [12]. Avatud tabeliformaadid võimaldavad hoida andmefailid hallatuna nagu relatsioonilises andmebaasis. Selline lähenemine vähendab andmejärvede puhul esinevat andmesoostumise riski, sest andmed on paremini organiseeritud ja nende kasutamine standardiseeritud. Andmejärvemaja arhitektuur on kiiresti pälvinud suurt tähelepanu nii tööstuses kui ka akadeemilises uurimistöös, kuna see lubab lahendada senise andmeinfrastruktuuri killustatuse probleemi [33, 20, 31]. Hoolimata tehnoloogilisest erinevusest, täidab korralikult realiseeritud andmejärvemaja samuti Inmoni määratletud neli põhitunnust: see on integreeritud, püsiv, teemakohane ja ajas muutuva sisuga andmekogu, mis toetab organisatsiooni otsustamist [25]. Seega on ka andmejärvemaja Inmoni definitsiooni järgi andmeait.

[10] https://delta.io/
[11] https://iceberg.apache.org/
[12] https://hudi.apache.org/

4 Avaandmeaida infrastruktuur

Avaandmeait on implementeeritud kasutades andmejärvemaja lähenemist. Seda seetõttu, et andmete formaadid varieeruvad palju ning lõppkasutajate soovid andmete kasutamiseks on töö tegemise hetkel kaardistamata. Saadaval on mitmeid tuntud andmejärvemaja lahendusi, mida pakuvad näiteks Snowflake [13], Databricks [14], Google BigQuery [15] ja Dremio [16]. Avaandmete ühele platvormile koondamisel võiks esmapilgul lihtsaim lahendus olla mõne sellise kommertslahenduse kasutamine. Kuid käesolevas töös peeti oluliseks lahenduse uudsust ning kulutõhusust, mistõttu otsustati eelistada vabavaralist lähenemist. Hetkel puudub samaväärne vabavaraline valmislahendus ja seetõttu on tarvis kombineerida mitmeid olemasolevaid avatud lähtekoodiga tehnoloogiaid, et luua platvorm, mille funktsionaalsus on võrreldav suurte teenusepakkujate pakutavate lahendustega.
Kontseptuaalselt koosneb andmejärvemaja, sarnaselt andmelaole, mitmest olulisest komponendist:

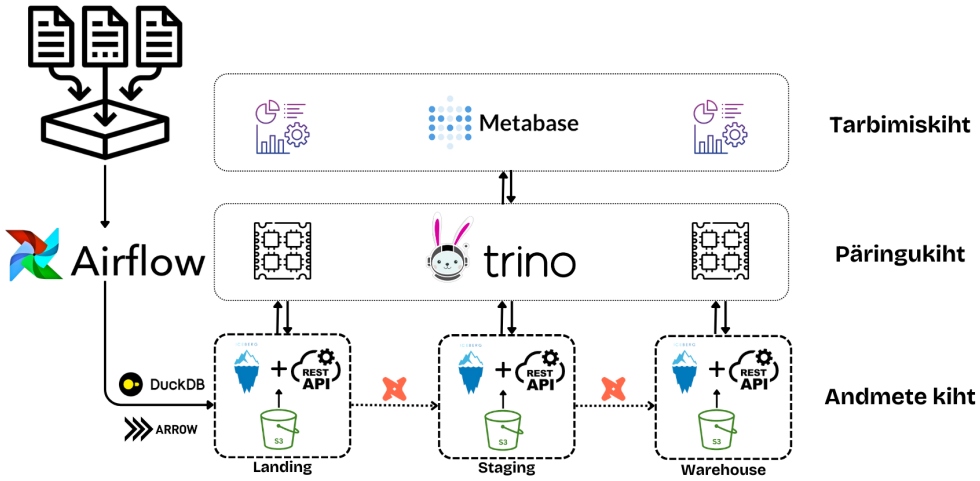
- **Keskne andmebaas või andmesalvestus:** Andmejärvemaja tuum on skaleeruv andmesalvestus, mis talletab kogu andmemassiivi. Traditsiooniliselt on selleks olnud relatsiooniline andmebaas, kuid modernses andmejärvemajas võib keskne andmeait baseeruda hajussalvestusel (pilveobjektide hoidla) koos kihiga, mis toetab ACID-tehinguid ja skeemi rakendamist. Oluline on, et see keskus võimaldab nii toorandmete kui ka töödeldud andmete hoidmist ühes kohas.

- **Metaandmete haldus:** Metaandmed ehk andmeid kirjeldav info on kriitiline komponent, mis tagab, et kasutajad leiaksid järvemajas vajalikud andmekogumid üles ja mõistaksid nende sisu. See hõlmab andmekatalooge, andmemudelite kirjeldusi, infoallikate päritolu (andmete päritolujälitus) jms. Metaandmete süsteem aitab hoida ära andmejärve muutumise "soiseks", pakkudes läbipaistvust ja korraldust. Praktikas on metaandmete haldust võimalik realiseerida läbi avatud tabeliformaatide.

- **Andmete laadimis- ja integreerimisvahendid:** Nagu andmeaidas, peab ka andmejärvemaja platvorm saama andmeid kätte erinevatest allikatest. See tekitab vajaduse tööriistade järele, mis suudavad andmeid sisse laadida ja järvemajas ringi liigutada.

- **Arvutus -ja ligipääsu tööriistad:** Andmejärvemaja väärtus realiseerub läbi tööriistade, mis võimaldavad lõppkasutajatel andmeid pärida ja analüüsida. Nendeks on

[13] https://www.snowflake.com/en/
[14] https://www.databricks.com/
[15] https://cloud.google.com/bigquery?hl=en
[16] https://www.dremio.com/



Joonis 8. Avaandmeaida infrastruktuur.

päringutööriistad, analüütika- ja visualiseerimisvahendid ning masinõppe platvormid, mis kasutavad järvemajas talletatud andmeid. Oluline on, et need tööriistad saavad andmetele ligi ühtsete liideste kaudu ning et andmete kasutusõigused ja turvalisus on kekselt hallatud [34].

Avaandmeait implementeerib kõik need komponendid kasutades eraldi vabavaralisi tehnoloogiaid nii nagu näidatud joonisel 8. Käesolev peatükk annab ülevaate avaandmeaida infrastruktuurist ja selle loomisel kasutatud tehnoloogiatest. Infrastruktuuri realiseerimisel kasutati teenuste hõlpsamaks haldamiseks ja ülesseadmiseks virtuaalkonteinereid Dockeris. Andmesalvestuseks ja metaandmete haldamiseks kasutatakse Apache Icebergi. Andmete laadimist ja integratsiooni tehakse kasutades Apache Airflow'd orkestreerijana, DuckDB'd ja PyArrow'it andmete sisse laadimiseks ning dbt-d süsteemisiseselt andmete liigutamiseks ja transformeerimiseks. Arvutustööriistana on kasutatud Trinot ja lõppkasutajatel on ligipääs andmetele läbi Trino JDBC draiveri või Metabase'i veebirakenduse.

4.1 Andmete kiht

Andmete kihi implementimiseks andmejärvemajas on vajalik avatud tabeliformaat, mis lisab failipõhistele andmetele andmebaasidele omase metaandmete kihi. Iga tabeli juurde kuulub metaandmestik, mis kirjeldab selle struktuuri (atribuutide nimesid ja tüüpe), partitsioone ja eelnevaid versioone. See võimaldab hallata suuri andmehulkasid struk-

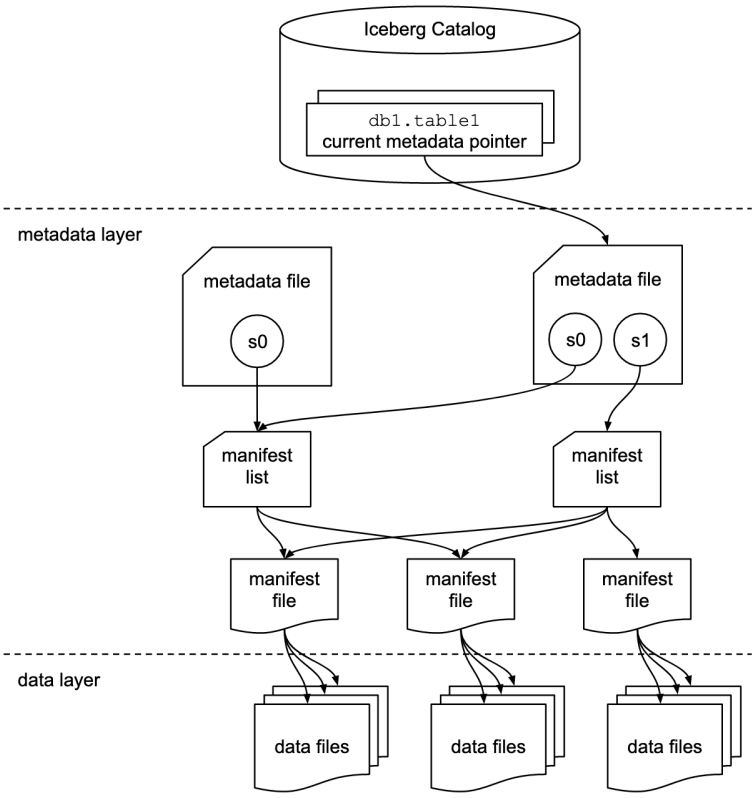
tuurselt ning tagab ACID-omadused ka hajusatel failisüsteemidel. Metaandmete kaudu on võimalik jälgida kõiki tabelis tehtud muudatusi: uute failide lisamist, vanade eemaldamist, ridade lisamisi ja kustutamisi jms. See tähendab, et andmekihi päringud ei pea pöörduma otseselt toorfailide poole ega skaneerima kogu failisüsteemi – vajalikud failid leitakse metaandmete abil kiiresti üles. Metaandmete olemasolu võimaldab ka ajaloolisi päringuid, kuna säilitatakse tabeli eelmised hetktõmmised ning on võimalik vaadata andmestiku seisundit mistahes varasemal ajahetkel. Samuti toetavad metaandmed skeemimuudatusi (veeru lisamine, eemaldamine, ümber nimetamine) ilma, et peaks terve andmehulga ümber kirjutama – uus skeem registreeritakse metaandmetes ja andmefailid jäävad füüsiliselt muutmata. Kokkuvõttes toimib metaandmete kiht "andmeladestusena" andmejärvel, võimaldades korraga nii andmete terviklikkust kui ka paindlikku töötlemist.

4.1.1 Apache Iceberg

Antud lahenduses kasutatakse Apache Icebergi avatud tabeliformaati andmete kihi implementeerimiseks. Icebergi andmefaile hoiustatakse Parquet formaadis Amazoni poolt pakutavas S3 failihoidlas, kuhu on võimalik avaliku ja salajase võtmega ligi pääseda. Andmefaile kaardistava metaandmete haldust on võimalik implementeerida mitmel viisil: REST-teenusena, Hive Metastore'ina, JDBC andmebaasina või Nessie kataloogina [35]. Käesolevas töös on valitud REST API-l põhinev lähenemine, kus Icebergi kataloog (tabelite metaandmete hoidla) toimib eraldiseisva veebiteenusena. See tähendab, et süsteem pöördub andmetabelite loomiseks, muutmiseks ja päringute ettevalmistamiseks API liidese poole. Tänu programmeerimisliidesele kapseldub metaandmete haldus omaette teenusesse, lihtsustades kogu süsteemi ülesehitust – näiteks puudub vajadus käitada eraldi Hive Metastore'i teenust või hoida katalooge lahti eraldi andmebaasis, kuna Icebergi REST API katab selle funktsionaalsuse. Samuti teeb REST-liides andmekihi kättesaadavaks ka väljaspool konkreetsest programmeerimiskeelest: võimalik on suhelda metaandmetega üle võrgu standardsete HTTP päringutega, mis teeb andmete kihi ülejäänud kihtide suhtes agnostiliseks. Illustratsioon Icebergi tabelite ülesehitusest on toodud joonisel 9.

Apache Iceberg on valitud andmekihi tehnoloogiaks peamiselt tänu selle agnostilisele ülesehitusele, lihtsale integreeritavusele ja tugevale arendajaskonnale. Iceberg defineerib avatud tabeliformaadi, mis on sõltumatu töötlemismootorist – seda toetavad mitmed suurandmete tööriistad (sh Apache Spark, Flink, Trino/Presto, Hive, Impala jm), võimaldades pärida sama andmestikku erinevate mootoritega ühtse SQL-liidese kaudu. Icebergi integratsioon teiste süsteemidega on otsene ja standardne – projekt pakub Java-põhist API-t ja teeke (PyIceberg), mis hõlbustavad Iceberg-tabelite kasutamist eri platvormidel. See mootoritest sõltumatu ja pilv-agnostiline lähenemine tähendab, et andmetaristu saab olla paindlikum ning vältida lukustumist ühe konkreetse teenuspakkuja ökosüsteemi.

Alternatiividena on olemas ka teisi avatud tabeliformaate, nendest populaarsemateks on Icebergi kõrval Delta Lake ja Apache Hudi. Kõik formaadid on funktsionaalsuse



Joonis 9. Icebergi tabelite struktuur [35].

poolest peaaegu samaväärsed ning arenevad kiires tempos. Iceberg ja Delta Lake on optimiseeritud analüütiliste päringute jaoks. Teistsuguse arhitektuuriga Hudi on aga optimiseeritud tegema andmetes kiiresti palju uuendusi ja seega töötlema efektiivsemalt reaalajas andmeid [36]. Kuna aga avaandmed pole suuremas osas reaalajas ja Hudi eelis jääks realiseerimata. Delta Lake on optimiseeritud kasutamaks Sparki ja DataBricksi ja integreerub kõige paremini nendega [37]. Käesolevas töös pole aga plaanis kumbagi tehnoloogiat kasutada, siis kaob sellega Delta Lake'i eelis teiste formaatide ees. Nende põhjuste tõttu on Avaandmeaidas eelistatud Icebergi teistele avatud tabeliformaatidele. Alljärgnevalt on esitatud väljavõte docker–compose.yml failist, mis konfigureerib Icebergi REST teenuse konteineri:

```
iceberg:
    image: tabulario/iceberg-rest
    container_name: iceberg
    ports:
      - "8181:8181"
    environment:
      AWS_ACCESS_KEY_ID: {aws_access_key}
```

```
    AWS_SECRET_ACCESS_KEY: {aws_secret_access_key}
    AWS_REGION: eu-north-1
    CATALOG_WAREHOUSE: s3://{s3_bucket_name}/iceberg
    CATALOG_IO__IMPL: org.apache.iceberg.aws.s3.S3FileIO
    CATALOG_URI: jdbc:sqlite:file:/home/iceberg/iceberg.db
volumes:
  - ./iceberg/:/home/iceberg
networks:
  iceberg_network:
```

Dockerit luues tekitatakse ka volüüm majutavasse süsteemi, kuna REST teenus talletab metaandmeid SQLite andmebaasis, mis läheksid konteineri taaskäivitumisel kaotsi.

4.2 Päringukiht

Andmejärvemaja arhitektuuris täidab päringukiht kriitilist rolli, muutes andmejärves talletatud suurandmed päringutega kättesaadavaks analüütikaks. Andmeid on võimalik Apache Icebergist pärida mitmete kasutaja arvutisse paigaldatavate tööriistade abil, nagu näiteks DuckDB [17] või PyIceberg [18]. Suuremad andmejärvemaja pakkujad, nagu näiteks Snowflake, pakuvad enda poolt päringusüsteeme, mis jooksevad pilves ja seeläbi eemaldavad kliendilt riistvaralised piirangud.
Riistvaraliste piirangute eemaldamiseks on loodud hajusad päringumootorid (*distributed query engine*), mis on mõeldud otse andmehoidlatest suuremahuliste SQL-päringute tegemiseks. Selliseid süsteeme iseloomustab massiline paralleeltöötlus (MPP – *Massively Parallel Processing*). Massilises paralleltöötluses kasutatakse koordinaatorit ja mitmeid töösõlmesid. Koordinaator tegeleb päringu vastuvõtmise, päringu SQL-süntaksi analüüsimise, täitmisplaani koostamise ja üldise orkestreerimisega, samal ajal kui töötajad täidavad plaani alusel tegelikku andmetöötlust [38]. Selline mudel on levinud paljudes MPP-süsteemides, tagades tsentraalse planeerimise ja hajusalt paralleelse täitmise. Hajusad SQL-päringumootorid on tavaliselt loodud voogedastava täitmise arhitektuuriga, kus andmeid töödeldakse torustikuna läbi mitme etapi ilma tarbetute vahepealsete salvestusteta. Sellises torustiku-stiilis mootoris "lükatakse" andmed läbi mitme järjestikuse täitmisetapi nii, et ühe etapi tulemusi hakatakse järgmises etapis tarbima kohe nende genereerimisel. See erineb etapiviisilisest (*stage-by-stage*) mudelist, mida kasutasid esimesed suurandmemootorid (nt MapReduce-põhine Hive, varasem Spark SQL), kus iga etapp pidi täielikult lõppema enne järgmise alustamist. Voolupõhine mitme-etapiline paralleeltöötlus suurendab oluliselt samaaegset paralleelsust ja ressursside kasutust, vähendades latentsust võrreldes etapiviisilise mudeliga [39].

[17] https://duckdb.org/
[18] https://py.iceberg.apache.org/

4.2.1 Trino

Trino (varasemalt PrestoSQL) on avatud lähtekoodiga hajus SQL-päringumootor, mille arhitektuur on kujundatud suure jõudlusega interaktiivsete päringute tarbeks üle andmejärve. Trino klaster koosneb ühest koordinaatorist ja mitmest töösõlmest. Koordinaator haldab sissetulevate päringute järjekorda ja käitlemist. Seejärel toimub SQL-süntaksi töötlemine (Trino kasutab ANTLR-põhist parserit süntaksi puu loomiseks) ning loogilise päringuplaani genereerimine. Loogiline plaan on päringuoperaatorite puu, mis on algul sõltumatu füüsilisest täitmisviisist. Koordinaatori planeerija/optimeerija rakendab reeglipõhiseid transformaatori-reegleid, et muuta loogiline plaan efektiivseks füüsiliseks täitmisplaaniks, kuni edasised parandused pole võimalikud. Tulemuseks on jaotatud täitmisplaan, mis koosneb mitmest etapist jaotatuna ülesanneteks erinevatel töötajatel [38].
Trino täitmisetapid on loogilised alamplaanid, mis võivad toimida paralleelselt. Iga etapp jaotatakse ülesanneteks konkreetsetel töösõlmedel – kõik ühe etapi ülesanded teostavad sama operatsioonide jada, kuid erinevate andmete peal. Etappide vahelisel andmevahetusel kasutab mootor segamis-mehhanismi (*shuffle-mechanism*): töötajad vahetavad andmeplokkide kaupa tulemusi üle võrgu. Trino rakendab seda mälusiseste puhverdatud andmevoogudena – vaheandmed edastatakse otse töötajalt töötajale üle HTTP päringu, hoides andmeid vahemälus kuni tarbimiseni. Selline voogedastuslik töötlus minimiseerib kettakirjutuse latentsust, kuid nõuab, et kogu vaheolek mahuks mällu. Trino lahendab mälu haldust mälukvootide jaotamisega: kogu mälu jagatakse loogiliselt mälu puhvritsoonideks (*memory pools*). Trino kehtestab igale päringule nii üle-klastrilise mälu piirangu kui ka sõlme-põhise piirangu. Kui päringu kasutus ületab lubatud piiri (näiteks kogub liitmine liiga suure hash-tabeli), siis päring katkestatakse veaga. Samuti, kui mõni sõlm ammendab oma füüsilise mälu, peatatakse seal ajutiselt päringu täitmine, et vältida protsessi kokku jooksmist [38].
Praktikas on Trinot võimalik jooksutada läbi Java, Dockeri ja Kubernetese. Käesolevas projektis jooksutatakse Trinot ühe-lõimelisena läbi Dockeri, millega tulevad kaasa mitmed alammoodulid, nagu näiteks integratsioonid erinevate failisüsteemidega, autentimine ja ligipääsude reguleerimine [40]. Alljärgnevalt on esitatud väljavõte docker–compose.yml failist, mis konfigureerib Trino teenuse:

```
services:
  trino:
    image: "trinodb/trino"
    container_name: trino
    ports:
      - "8082:8082"
    volumes:
      - ./trino/config/config.properties:/etc/trino/config.properties
      - ./trino/auth/password-authenticator.properties:/etc/trino/
          password-authenticator.properties
```

```
    - ./trino/auth/:/etc/trino/auth/
    - ./trino/catalog/:/etc/trino/catalog/
networks:
  iceberg_network:
```

Trino konteineri korrektseks toimimiseks on vaja talle ette anda konfiguratsioonifailid. Olulisemad neist on config.properties (üldseaded ja võrgustik) ning kataloogi konfiguratsioonifailid. Alljärgnevalt on üldseadete sisu olulisemad osad. Trino config.properties konfigureerib Trino serveri käitumise:

```
internal-communication.https.required=true
http-server.https.port=8082
http-server.https.enabled=true
http-server.process-forwarded=true
http-server.authentication.type=PASSWORD
```

Ülaltoodud konfiguratsioonis on lubatud HTTPS protokoll pordil 8082 ning määratud, et Trino nõuab kasutajatelt autentimist parooli abil. Autentimise täielikuks implementeerimiseks on ka vaja sätestada password-authenticator.properties failis võtmed:

```
password-authenticator.name=file
file.password-file=/etc/trino/auth/passwords.db
```

Selline konfiguratsioon võimaldab kasutajate haldamist läbi faili ja neid lisada sisestades käsureale: htpasswd -B -C 10 password.db kasutaja.

4.2.2 Trino + Apache Iceberg

Andmejärvemaja arhitektuuri kontekstis on oluline Trino võimekus suhelda avatud tabeliformaatidest Apache Icebergiga. Pilveobjektide (nt S3) metaandmetele otseligipääs on tihti pudelikael: näiteks failide listimise S3-s tagastab maksimaalselt 1000 kirjet ja miljonite failide korral võib juba see minuteid aega võtta. Iceberg lahendab selle, hoides metaandmeid ise failidena struktureeritult (manifestidena), nii et Trino saab lugeda võrdlemisi väikest arvu metaandmefaile, selle asemel et küsida objektisalvestilt iga andmefaili eraldi. Trino koordinaator saab tänu sellele kiiresti välja selgitada, millised failid konkreetse päringu jaoks vajalikud on. Iceberg-tabeli päringu planeerimine toimub Trinos tihti ühe koordinaatori protsessi sees tervikuna – koordinaator loeb läbi ülemise taseme manifesti ning sealt suunatult vajalikud osamanifestid, moodustades vajadusel filtritega piiratud faililoendi, mida töötajad seejärel läbi töötavad [41].
Iceberg kataloogi konfiguratsiooni (iceberg.properties) fail on paigutatakse Trino konteinerisse kausta catalog ning faili nime esimene osa (iceberg) defineerib kataloogi nime, mida SQL-päringutes kasutatakse. Selles failis on määratud, et kasutatakse Icebergi moodulit ja selle REST API teenust:

```
connector.name=iceberg
iceberg.catalog.type=rest
```

```
iceberg.rest−catalog.uri=http://iceberg:8181
iceberg.rest−catalog.warehouse=s3a://{s3_bucket_name}/iceberg
fs.native−s3.enabled=true
s3.path−style−access=true
s3.endpoint=https://s3.{aws_region}.amazonaws.com
s3.region={aws_region}
s3.aws−access−key={AWS_ACCESS_KEY}
s3.aws−secret−key={AWS_SECRET_KEY}
```

Antud töös paiknevad Iceberg REST teenus ja Trino samas Dockeri võrgus, mis võimaldab neid ühendada kasutades konteinerite nimesid (kohalikku võrguaadressi kasutades töötab ühendamine ainult Docker Desktopis, aga mitte Linuxis).

4.3 Andmete laadimine

Traditsiooniliselt on andmete integreerimisel kasutatud ETL (*Extract, Transform, Load*) protsessi, kus andmed transformeeritakse enne laadimist sihtsüsteemi. See eeldab, et andmete lõplik struktuur ja kasutusotstarve on juba ette teada, kuna transformatsioon rakendatakse enne andmete salvestamist. ETL lähenemine tagab üldjuhul kõrgema andmekvaliteedi ja ühtluse, kuid võib suurte andmemahtude ja -voogude korral muutuda pudelikaelaks [42]. Uuemas ELT (*Extract, Load, Transform*) lähenemises laaditakse andmed esmalt toorandmetena andmehoidlasse ning transformatsioonid tehakse seejärel sihtplatvormil, pakkudes suuremat paindlikkust ja skaleeritavust. Pilvetehnoloogiate laialdane kasutuselevõtt on oluliselt soodustanud ELT levikut – pilvepõhised andmeaidad ja andmejärved pakuvad suuremat mastaapsust ja kuluefektiivsust, muutes ELT lähenemise atraktiivseks ja teostatavaks. Sihtandmeait või -järv toimib ELT-lähenemises ühtaegu andmete hoidla kui ka töötlemismootorina, erinevalt ETL-st, kus töötlemine toimus eraldiseisval ETL-tööriistal enne andmete laadimist [43, 44].
Eeltoodud põhjustel on ka käesoleva magistritöö raames loodud andmete töötlus kavandatud ELT lähenemise järgi. Andmed ekstraktitakse allikatest ja laaditakse esmalt toorandmetena Apache Iceberg formaadis andmehoidlasse. Seejärel rakendatakse vajalikud transformatsioonid dbt abil Avaandmeaidas endas. Selline disain võtab arvesse kaasaegse andmejärvemaja võimalusi ning tagab, et andmeanalüüsi ja modelleerimise sammud toimuvad toorandmetele lähedal.

4.3.1 Airflow

Andmetöötlusprotsessi orkestreerimiseks kasutatakse avaandmeaidas Apache Airflow'd, mis on laialdaselt tunnustatud töövoogude orkestreerimise platvorm. Airflow võimaldab defineerida andmetöötluse töövooge DAG-idena (*Directed Acyclic Graph*) ning ajastada ja monitoorida ülesannete täitmist. Airflow modulaarne arhitektuur orkestreerib ülesandeid sõnumijärjekorra abil, võimaldades suurt hulka töid samaaegselt käivitada. Airflow

kasutajaliides (veebipõhine UI) pakub täielikku ülevaadet töövoo täitmisest, võimaldades jälgida iga ülesande staatust ja logisid ning vajadusel käsitsi DAG-e käivitada või peatada. [45].
Käesolevas töös kasutati Airflow käivitamiseks Astronomeri [19], mis pakub mugavat viisi Airflow haldamiseks ja konteineriseerimiseks. Astro CLI genereerib ja käivitab vajalikud Docker konteinerid (andmebaas, veebiserver, planeerija ja täitja) ühtse klastrina. Praktikas saab Airflow projekti jooksutada käsuga `astro dev start`, mis loob taustal vajalikud konteinerid. [46].

4.3.2 dbt

ELT protsessi transformeerimise etapi teostamiseks kasutatakse üha enam spetsiaalseid tööriistu. Üks laialdaselt levinud vahend on dbt (*Data Build Tool*), mis on avatud lähtekoodiga andmetransformatsiooni raamistik. dbt võimaldab luua andmemudeleid (nt vaated või tabelid) otse andmeaitades, kirjutades transformatsiooniloogika SQL päringutena. Sisuliselt kombineerib dbt SQL-koodi kasutamise tarkvaraarenduse parimate praktikatega nagu näiteks versioonihaldus, mallid ja modulaarsus. dbt muudab andmete ettevalmistamise usaldusväärseks ja korduvkasutatavaks protsessiks. Oluline on, et dbt ise ei oma iseseisvat töötlemismootorit; ta on mootori-agnostiline tööriist, mis kasutab sihtplatvormi (nt Google BigQuery, Trino või Snowflake) enda SQL-mootorit transformatsioonide läbiviimiseks. dbt kompileerib kasutaja kirjeldatud päringupõhised mudelid sihtandmebaasi jaoks sobivasse SQL-dialekti ning käivitab need andmeaidas. Tänu SQL-kesksele lähenemisele on dbt kergesti omandatav – vaja on vaid SQL-i tundmist [47].

4.3.3 Airflow + dbt

Andmejärvemaja arhitektuuri ELT-torustiku realiseerimisel integreeritakse Airflow orkestreerimine dbt transformatsioonidega, et saavutada sujuv ja automatiseeritud töövoog. Üldpõhimõte on, et Airflow's defineeritud DAG sisaldab ülesandeid, mis käivitavad dbt-projekti transformatsioonid sobival hetkel.
Dbt transformatsioonide käivitamine Airflow's eeldab õiget seadistust. Esimeseks seadistuseks Avaandmeaidas on dbt jaoks eraldi virtuaalkeskkonna loomine. Kuna nii Airflow kui ka dbt on Pythonil põhinevad tööriistad, võib nende teekide vahel tekkida versioonikonflikte ja seetõttu on dbt jaoks loodud eraldi virtuaalkeskkond. Teiseks seadistuseks on dbt koodi Airflow jaoks kättesaadavaks tegemine. See on lahendatud dbt koodi volüümina Airflow konteineri lisamisega.
Kui keskkond on ette valmistatud, tuleb lahendada dbt transformatsioonide käivitamine Airflow's. Airflow pakub ametlikult ainult dbt Cloud'i operaatorit, mis sellesse projekti ei sobi, kuna üritame dbt-d jooksutada lokaalselt. Üks võimalus lokaalseks dbt jooksutamiseks on kasutada Airflow'sse sisseehitatud BashOperatorit, mis võimaldab täita

[19] https://www.astronomer.io/docs/astro/

käsureakäske. Kuna dbt primaarne liides on käsurea-põhine, saab BashOperatoriga hõlpsasti käivitada näiteks dbt run või dbt build käske. Suuremate projektide puhul ilmnevad aga BashOperatori otsese kasutamise piirangud. Airflow ei oma sisemist nähtavust dbt protsessi detailidesse – terve dbt projekt jooksutatakse ühe musta kastina, mille edukust või ebaedu saab jälgida ainult väliste logide kaudu. Lisaks tuleb dbt jooksutamiseks kaasa anda palju käsurea argumente (projekti asukoht, logifailide asukoht, profiil jpm.), mille iga kord koodi uuesti defineerimine on ajamahukas. Nende kitsaskohtade leevendamiseks ja sujuvama integratsiooni saavutamiseks loodi käesolevas lahenduses Airflow jaoks kohandatud dbt operaator [20], mis kasutab dbt.cli moodulit dbt käskude jooksutamiseks. Pärast käsu täitmist loeb operaator dbt väljundit ja tagastab õnnestumise puhul dbt tulemuse ja ebaõnnestumise puhul veateate. Operaatori kasutamine on lihtne, tuleb ainult loomisel kaasa anda vastav projekt, jooksutatavate mudelite võtmesõnad ja kas laadimist teha otsast peale või mitte. Operaatorit on võimalik Airflow's kasutada järgmise näite järgi:

```
from operators.dbt import dbtOperator
dbtOperator('staging').build(models=['avaandmed'])
dbtOperator('staging').run(models=['avaandmed'], full_refresh=False)
```

4.4 Tarbimiskiht

Andmete tarbimisel Avaandmeaita platvormil on keskne roll, võimaldades salvestatud andmetest reaalset väärtust luua. Tarbimiskiht on andmeplatvormi arhitektuuri osa, mis vahendab lõppkasutajatele juurdepääsu andmetele, peites andmeallikate keerukuse ning võimaldades sujuvat päringute tegemist. See toimib ka kaitsekihina, eraldades lõppkasutajate tegevused andmete töötlemise sisemistest protsessidest [48].
Tänapäeval eristatakse tihti kahte sorti andmeanalüütika tarbijaid: tehnilised ja tavakasutajad. Tehnilised kasutajad oskavad andmeid ühendada, koostada päringuid ja aruandeid ning teha keerukamat analüüsi iseseisvalt. Seevastu tavalistel kasutajatel napivad samasugused oskused ja nad toetuvad tööriistade intuitiivsusele või tehnilisematele inimeste abile, et andmetest endale kasulikku teavet leida. Mõlema grupi teenindamiseks on kaasaegne suund iseteeninduslikku ärianalüütika poole. Kaasaegsed ärianalüütika tööriistad pakuvad graafilisi liideseid, mille abil saab päringuid koostada ilma SQL-i tundmata ning võimaldavad kasutajatel andmemudeleid omal käel uurida [49].

4.4.1 Trino JDBC

Tehnilistele kasutajatele (nt. andmeanalüütikud, andmeteadlased) on võimalus saada otsene ligipääs masinloetavatele andmetele programmeerimisliideste või SQL-päringute kaudu. Avaandmeaita platvormil on selle eesmärgi täitmiseks Trino päringumootor

[20] https://github.com/shanazar/data_lakehouse/blob/main/airflow/operators/dbt.py

koos JDBC (*Java Database Connectivity*) liidesega. Praktikas saavad tehnilised kasutajad Trino JDBC ühendust kasutada erinevate vahendite kaudu. Esimeseks variandiks on SQL-klientrakendused ja andmebaasi haldusvahendid nagu DBeaver või JetBrains DataGrip. Teiseks variandiks on analüütika- ja programmeerimiskeeled (näiteks R ja Python), mis saavad Trinoga ühenduda JDBC/ODBC draiverite või spetsiaalsete teekide abil, võimaldades andmete pärimist otse koodist ja integreerimist andmetöötluse töövoogudesse. Ühendada saab enda valitud tööriista vastu Avaandmeaita läbi aadressi jdbc:trino://trino.avaandmeait.ee:443 ning andes argumentidena kaasa kasutajanime ja parooli.

4.4.2 Metabase

Tavakasutajad vajavad andmetele ligipääsu viisil, mis ei eelda programmeerimisoskust ega detailseid teadmisi SQL-ist. Avaandmeait pakub neile iseteeninduslikku ärianalüütika lahendust Metabase'i abil. Tegemist on veebipõhise rakendusega, mis kasutab SQL-päringuid tulemuste toomiseks. Metabase'is on võimalik nii ise SQL-i kirjutada, kui ka läbi visuaalsete abitööriistade päringuid kujundada. Seetõttu saavad ka vähese tehnilise ettevalmistusega inimesed Avaandmeaida andmeid analüüsida ja visualiseerida. Lisaks saab Metabase's hõlpsasti koostada töölaudu, graafikuid ja erinevaid näidikuid, mille abil andmetest tehtavaid järeldusi edasi anda.
Avaandmeaidas on Metabase juurutatud läbi Docker'i, mis on ühendatud ülejäänud teenustega samasse võrku. Metabase'il puudus esialgu otsene tugi Trino päringumootorile, mistõttu tuli platvormi seadistamisel lisada Metabase'ile täiendav JDBC draiver. Konkreetselt kasutati Starbursti poolt arendatud Trino draiverit [21].

[21] https://github.com/shanazar/data_lakehouse/tree/main/metabase/plugins

5 Andmeaida disain

Järgnevas peatükis kirjeldatakse, kuidas andmed lõpptarbijani jõuavad. Avaandmeait on rajatud medaljoni arhitektuuri põhimõtetele [50], mis jagab töötlustaristu kolmeks loogiliseks kihiks:

1. Maandumiskiht (*landing layer*)
2. Vahekiht (*staging layer*)
3. Andmeaida kiht (*warehouse layer*)

Kihiline lähenemine võimaldab töövoogude selget piirimääratlust, andmekvaliteedi järkjärgulist tõstmist ning skaleeritavat haldust [24].

5.1 Maandumiskiht – andmete laadimine

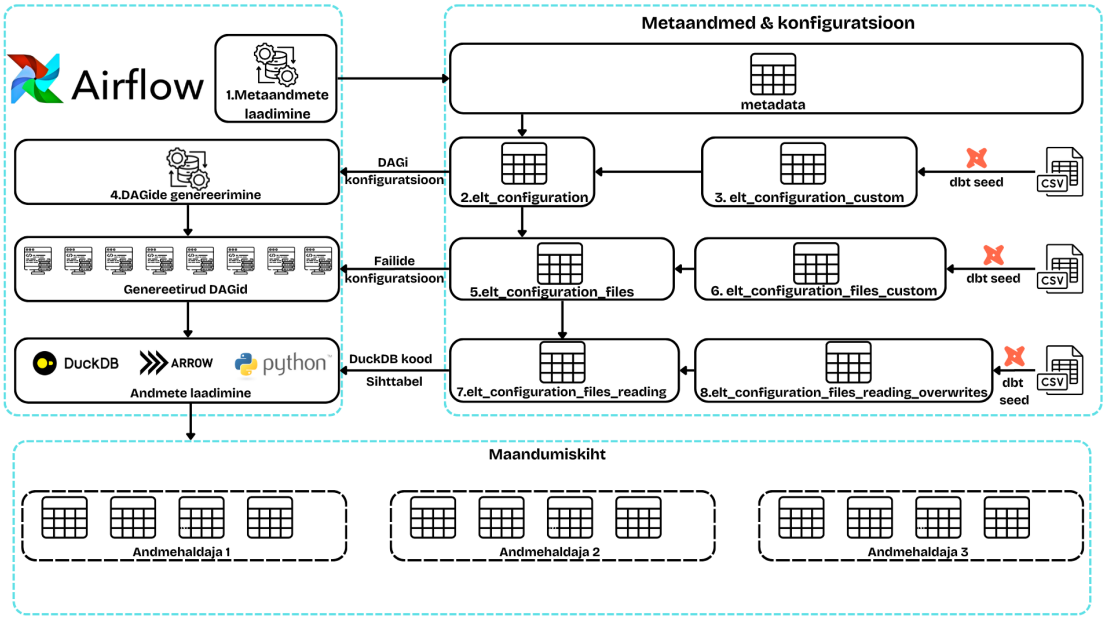
Andmeallikaid võib liigitada nende muutumiskiiruse järgi kaheselt: staatilised ja dünaamilised.

- **Staatilised andmeallikad** on pidevalt kättesaadavad sama aadressi kaudu ja muutuvad harva. Nende metaandmete lugemiseks on võimalik kasutada lihtsamaid laadimismeetodeid, kuna nende uuenemissagedused on teada. Oluline on, et staatilised andmed on versioonihalduse all ning igasugused muutused on jälgitavad. Avaandmeaidas on sellisteks allikateks näiteks Riigihangete register ja Äriregister.
- **Dünaamilistele allikatele** võib andmeid pidevalt lisanduda ettearvamatutel hetkedel. Selleks, et kinni püüda muudatusi dünaamilistes andmeallikates, peab jälgima nende metaandmeid. Ainsaks dünaamiliseks andmeallikaks Avaandmeaidas on hetkel avaandmete portaal ja seal sisalduvad failid. Avaandmete portaalis avaldatakse pidevalt läbi failide uusi andmestikke ning muudetakse eelnevaid. Muudatused on etteaimamatud ja seega jälgitakse igapäevaselt metaandmeid nende püüdmiseks.

Andmete jõudmine maandumiskihti on kirjeldatud joonisel 10.

5.1.1 Andmestike laadimine

Avaandmeaias luuakse Airflow DAG-id dünaamiliselt, tuginedes metaandmetes talletatud teabele iga andmeallika kohta. Airflow uuendab kord päevas maandumiskihi metaandmeid (joonisel 10 nr. 1); seejärel koostatakse inkrementaalselt laadimiskonfiguratsiooni tabel, kuhu lisatakse ainult andmestikud, mille allika uuenemisaeg on hilisem kui eelmise laadimise aeg. Päring, millega luuakse avaandmete portaali metaandmetest laadimiskonfiguratsiooni tabel elt_configuration (joonisel 10 nr. 2) on toodud järgnevalt:



Joonis 10. Avaandmeaida andmete laadimise arhitektuur.

```
select
    cast(id as varchar) as dataset_id,
    cast(name as varchar) as dataset_name,
    regexp_replace(
        regexp_replace(
            regexp_replace(
                regexp_replace(
                    regexp_replace(lower(slug), '\.[^.]*$', ''),
                    '[^a-zA-Z0-9 -]', ''),
                ' ','_'),
            '-', '_'),
        '\.[^.]*$','') as dag_name,
    cast(update_interval_frequency as int) as
       update_interval_frequency,
    cast(update_interval_unit as varchar) as update_interval_unit,
    case
    when cardinality(files) = 0 then 'web-scraping'
        else 'files'
    end as source_type,
    case
        when update_interval_frequency = 0 then '@once'
        when update_interval_unit in ('minute', 'hour') then
            case
                when update_interval_unit = 'minute' then concat('*/'
```

```
                    , cast(update_interval_frequency as varchar), ' *
                    * * *')
                when update_interval_unit = 'hour' then concat('0 */'
                    , cast(update_interval_frequency as varchar), ' *
                    * *')
            end
    else '@once'
    end as cron_expression,
    'direct_download.j2' as jinja_template,
    case
    when cardinality(files) = 0 then false
        else true
    end as enabled,
    replace(organization.slug, '-', '_') as schema_name,
    cast(null as varchar) as comment,
    updated_at as last_update_timestamp,
    cast(from_iso8601_timestamp('1800-05-01') as timestamp(6)) as
        last_ingestion_timestamp
from {{ ref("datasets") }}
{% if is_incremental() %}
    where updated_at > (select max(last_update_timestamp) from {{
        this }})
{% endif %}
```

Ülal toodud päringus on igal andmestikul unikaalne ID ja DAGi nimi. Uuendamisgraafik on andmestikel sätitud ühekordseks, kui nende uuendamistihedus on harvem kui päev, sest siis haldab käitamist DAGide generaatori kood vastavalt uuenemisajale. Laadimise tüüpe on kaks: failipõhine ja veebikraapimine (mida pole veel implementeeritud). Lisaks on tabelis kirjas veel laadimise lubatus, laadimise alusmall (Jinja faili nimi), skeema nimi, viimase laadimise ajahetk süsteemis sees ja uuendamise aeg metaandmetes ning on võimalus igale andmestikule lisada ka kommentaar (näiteks tema kvaliteedi kohta).
Staatiliste allikate konfiguratsioon hallatakse dbt seed funktsionaalsusega. See tähendab, et defineerides csv faili vajalikud tulbad, tehakse csv failist tabel nimega elt_configuration_custom (joonisel 10 nr. 3) ja seejärel ühendatakse see dünaamilise konfiguratsiooniga, lisades ta union käsu abil elt_configuration tabelile.
Pärast konfiguratsioonitabeli koostamist jooksutatakse DAGi dag_generator (joonisel 10 nr. 4), kus päritakse konfiguratsioonitabelist kõik andmestikud, mille Avaandmeaida viimane laadimisaeg on vanem kui nende uuendamisaeg. Kui DAGi ei eksisteeri, võetakse konfiguratsioonis kirjas olev Jinja mall ja asendatakse seal sees ID, nimi ja laadimisgraafik ning kirjutatakse Pythoni faili DAGina. Kui DAG juba eksisteerib, siis teda lihtsalt jooksutatakse. Selline lähenemine võimaldab suuremat konfigureeritavust, kuna malle saab alati vastavalt uutele juhtumitele juurde teha.

5.1.2 Failide lugemine

Igal andmestikul võib olla mitu faili. Nende kohta talletatakse teave tabelis elt_configuration_files (joonisel 10 nr. 5), mis koostatakse järgneva päringuga:

```
select
    id as dataset_id,
    file_id,
    regexp_replace(file_name, ' ', '_') as file_name,
    concat('https://avaandmed.eesti.ee/api/datasets/',id, '/files/',
        file_id,'/download-s3') as file_url,
    'POST' as request_method,
    file_mimetype,
    concat('landing/raw_data/',cast(ec.schema_name as varchar),'/{
        yyyy}/{mm}/{dd}/',cast(file_name as varchar)) as s3_path,
    false as compressed,
    case
        when file_mimetype in ('text/xml',
                               'application/json',
                               'text/csv',
                               'application/vnd.openxmlformats-
                                   officedocument.spreadsheetml.sheet'
                                   ,
                               'application/vnd.ms-excel')
                                then true
        else false
    end as load_to_iceberg,
    ec.schema_name
from {{ ref('datasets') }}
cross join
    unnest(files) as t (
        file_id,
        file_name,
        file_mimetype,
        file_size,
        file_dataset_id,
        file_metadata,
        file_processing_status,
        file_storage_filename
    )
left join {{ ref('elt_configuration') }} as ec on id = ec.dataset_id
```

Failide konfiguratsioonitabelis on kirjas faili kättesaamise aadress, päringutüüp (GET/POST), faili asukoht S3-s, kas fail on kokkupakitud või mitte, faili tüüp ning kas ta on masinloetav ehk kas teda peaks laadima Iceberg'i või mitte. Andmete laadimisel arvestatakse nende väljadega ja protsessi kohaldatakse vastavalt nendele. Esmalt laetakse fail alla baasis täpsustatud aadressilt. Seejärel laetakse fail toorkujul S3-e, partitsioneerides andmestiku nime ja kuupäeva järgi. Mõnede failiformaatide puhul, kuigi iseenesest

masinloetavad, puudub duckDB-l võimekus neid lugeda. See tähendab, et need vajavad eeltöötlust enne lugemist. XML failid lamendatakse csv failideks, kasutades rekursiivseid funktsioone ning xls lõpuga failidest tehakse xlsx failid, kasutades Pandase ExcelWriterit. Pärast eeltöötlust jõuab programm faili lugemiseni DuckDB-ga. DuckDB valiti lugemiseks oma kiiruse, funktsioonide rohkuse ja veergude automaatse tuvastamise võimekuse pärast [51]. Pärast DuckDB-s faili lugemist tehakse andmetest PyArrow'i skeem, mida kasutatakse Icebergi tabeli tekitamiseks. Faili lugemise konfiguratsioon koos sihttabeliga tuleb kaasa eraldi tabeliga elt_configuration_files_reading (joonisel 10 nr. 7), mis koostatakse järgneva päringuga:

```
select distinct
    dflt.dataset_id,
    dflt.file_id,
    overwrites.sheet_name,
    dflt.file_mimetype,
    coalesce(
        overwrites.duckdb_read_command,
        case
            when dflt.file_mimetype = 'text/csv'
                then 'select *, row_number() over () as
                    file_row_number from read_csv("{local_file_path
                    }", sample_size=-1)'
            when dflt.file_mimetype = 'text/xml'
                then 'select *, row_number() over () as
                    file_row_number from read_csv("{local_file_path
                    }", sample_size=-1)'
            when dflt.file_mimetype = 'application/json'
                then 'select * from read_json("{local_file_path}",
                    sample_size=-1)'
            when dflt.file_mimetype = 'application/vnd.openxmlformats
                -officedocument.spreadsheetml.sheet'
                then 'select *, row_number() over () as
                    file_row_number from read_xlsx("{local_file_path
                    }", all_varchar = true)'
            when dflt.file_mimetype = 'application/vnd.ms-excel'
                then 'select *, row_number() over () as
                    file_row_number from read_xlsx("{local_file_path
                    }", all_varchar = true)'
            else null
        end
    ) as duckdb_read_command,
    schema_name,
    coalesce(
        overwrites.table_name,
        case
            when dflt.file_mimetype in (
                'text/xml',
                'application/json',
```

```
                'text/csv',
                'application/vnd.openxmlformats-officedocument.
                    spreadsheetml.sheet',
                'application/vnd.ms-excel'
            ) then regexp_replace(
                    regexp_replace(
                      regexp_replace(
                        regexp_replace(
                          regexp_replace(
                            lower(dflt.file_name),
                            '\.[^.]*$', ''
                          ),
                          '[^a-zA-Z0-9 -]', ''
                        ),
                        ' ', '_'
                      ),
                      '-', '_'
                    ),
                    '\.[^.]*$', ''
                  )
            else null
        end
    ) as table_name

from {{ ref('elt_configuration_files') }} as dflt
left join {{ source('overwrites', '
    elt_configuration_files_reading_overwrites') }} as overwrites
    on dflt.dataset_id = overwrites.dataset_id
    and dflt.file_id = overwrites.file_id
where dflt.load_to_iceberg = true
```

Tabeli koostamisel üritatakse vastavalt faili tüübile koostada duckDB lugemiskäsk, mis faili loeb. See on eraldiseisev failide tabelist, kuna mõnele failile tuleb anda kohaldatud käsk või mitu lugemiskäsku. Mitut käsku on enamasti tarvis Excelis olevate erinevate vahelehtede lugemiseks ja kohaldatud käske suurte JSON-failide haldamiseks. Selliste erandjuhtude käsitlemiseks on ka lisatabel elt_configuration_files_reading_overwrites (joonisel 10 nr. 8), kus saab andmestiku ja faili ID järgi defineerida lugemiskäske, mida seejärel saab dbt seed käsu abil andmebaasi ja dbt run –-select elt_configuration_files_reading käsuga lugemisseadete tabelisse laadida.

5.2 Vahekiht – andmete ühtlustamine

Vahekihi ülesandeks on andmete normaliseerimine ja ettevalmistamine andmeaida kihi jaoks. Kogu töö toimub siin dbt inkrementaalsete mudelite abil, mis eeldab iga andmestiku puhul kirjete unikaalse kombinatsiooni leidmist, mille alusel uusi kirjeid tuvastatakse. Esimeseks ülesandeks on loogiliste andmekogude ühendamine. Kuna ühel andmestikul

võib olla mitu faili ja igal failil mitu tabelit, on vaja need vahekihis ühendada ühte loogilisse andmekogusse. Selle protsessi automatiseerimiseks kasutatakse dbt dünaamilisi makrosid. Sageli on andmestiku failid ajaliselt partitsioneeritud (nt riigihankednotice2018, riigihankednotice2019 jne), mistõttu tekib vajadus uusi tabeleid dünaamiliselt laadida. Avaandmeaidas kasutatakse selleks kohandatud dbt makro get_tables_by_pattern, mis otsib andmebaasist mustrile vastavaid tabeleid. Kuigi dbt-utils teegis on sarnane funktsionaalsus olemas, puudub Trinol ILIKE funktsioon, mille tõttu oli vajalik iseseisev lahendus. Makrole tuleb ette anda andmebaasi nimi, skeema, tabelinimemuster ja vajadusel mustrid, mida ignoreerida. Näidis makro kasutamiseks leidmaks kõik riigihangete tabelid on toodud järgnevalt:

```
{% set relations = get_tables_by_pattern('landing','riigihanked', '
    riigihankednotice%', 'riigihankednoticeaward%') %}
{% for rel in relations %}
select * from {{ rel }}
{% if not loop.last %}
    union all
{% endif %}
{% endfor %}
```

Teiseks ülesandeks on andmete tüüpimine ja korrastamine. Nende sammudega tagatakse, et kõik andmed oleksid õiges tüübis ja formaadis. Selleks kasutatakse Trino try_cast(veerg as type), mis võimaldab veerge turvaliselt soovitud tüübiks teisendada. Eraldi väljatoomist väärib probleem, kus Excelist pärit kuupäevad olid esitatud täisarvudena. Uurimisel selgus, et Exceli ajaarvestus algab kuupäevast 1899-12-30; täisarvulise päevade arvu lisamine sellele kuupäevale annab tegeliku kuupäeva. Näide teisenduskäsust:

```
date_add('day', cast("alguskuupaev" as integer), date '1899-12-30')
    as alguskuupaev
```

Kolmandaks ülesandeks oli veergude ühildamine ja veerunimede normaliseerimine. Nimede normaliseerimine seisneb tühikute ja sidekriipsude asendamises alakriipsudega ning mitte-inglise tähtede standardiseerimises. Selle tulemusel võib veergudele viidata ilma jutumärkideta. Lisaks esines juhtumeid, kus andmed olid jaotatud mitme erineva nimega veeru vahel. Üheks selliseks olid riigihangete andmestikud vahemikus 2017. aasta septembrist kuni 2023. aasta maini, kus olid samad väärtused ära jaotatud 10 erineva nimega veeru vahel ja sellest tulenevalt sisaldas keskmine tabel endas umbes 600 veergu. Selle probleemi lahendamiseks loodi dbt makro, mis koos Trino coalesce funktsiooniga leiab sarnaste nimedega veerud ja tagastab esimese mittetühja väärtuse. Makro kasutamine riigihangete tabelist veergude otsimiseks nägi välja selline:

```
{% set relations = get_tables_by_pattern('landing','riigihanked', '
    riigihankednotice%', 'riigihankednoticeaward%') %}

{% for rel in relations %}
select distinct
```

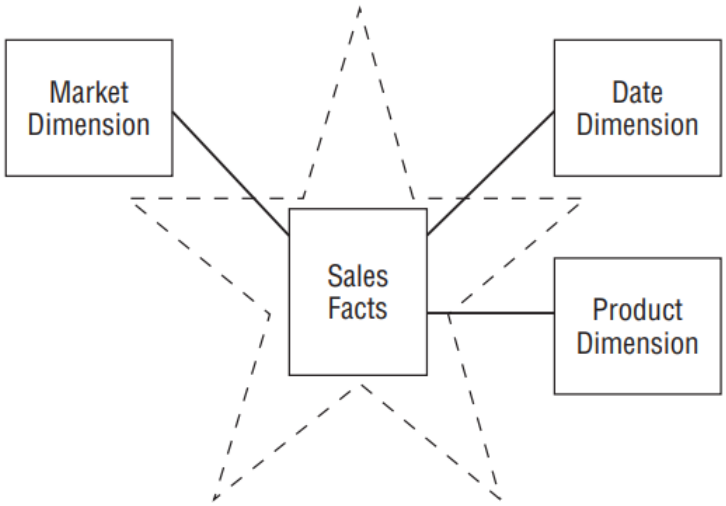
```
    coalesce({{ get_columns_with_pattern(rel, '%
        contract_reference_number', 'varchar') }}, null) as id
from {{ rel }}
{% if not loop.last %}
    union all
{% endif %}
```

Viimaseks ülesandeks vahekihis oli andmete lamendamine. Mitmes andmestikus sisaldasid veerud JSON-formaadis objektide massiive. Sellise andmeesituse puhul on andmebaasi normaliseerimise huvides otstarbekas lamendada need eraldi tabelitesse. DuckDB tuvastab automaatselt sellised massiivid array tüübina, mida saab Trino cross join unnest() funktsiooni abil lahti parsida.

5.3 Andmeaida kiht – andmete modelleerimine

Andmeaida kiht on viimane ja seega on siia jõudnud andmed tarbimiseks valmis ning kihi ülesandeks on need kasutajale presenteerida arusaadaval viisil. Selle jaoks on tähtskeem eelistatud traditsioonilistele relatsioonilistele mudelitele, sest tähtskeem on kasutajale arusaadavam, võimaldab visualiseerimistööristades andmeid lihtsamini grupeerida ning võimaldab OLAP-päringuid efektiivsemalt teostada. Tähtskeem on dimensioonilise disaini keskne skeem, kus üks keskne faktitabel on seotud mitme dimensioonitabeliga, moodustades skeemi, mis meenutab tähte – faktitabel asub skeemi keskmes ning dimensioonitabelid kiirduvad sellest välja. Näide tähtskeemist on toodud joonisel 11. Faktitabel sisaldab tavaliselt äriliste sündmuste või tehingute detailseid kirjeid ning mõõdikunäitajaid (näiteks müügisumma, kogus, sündmuse arv). Dimensioonitabel kirjeldab üht analüüsi mõõdet – see sisaldab atribuute, mis annavad faktile konteksti (näiteks aja dimensioon sisaldab kuupäeva, nädalanumbrit, kvartalit jne; asukoha dimensioon sisaldab linnanimesid, maakondi, riike). Dimensioonitabelid on faktitabeliga üks-mitmele seoses: ühte dimensioonikirjet võib faktitabelis esineda palju kordi [24].

Avaandmete puhul on dimensionaalne lähenemine keeruline. Andmete kvaliteet ja sisu varieeruvad ja häid faktimudeleid on vähe, kuna suur osa andmestikest on eelaggregeeritud. Selleks, et oleks võimalik Avaandmeaida andmeid rikastada, on lisatud kolm dimensionaalset tabelit. Esimeseks on ajalise dimensiooni lisamiseks mõeldud kuupäevade tabel d_date, kus on vahemikus 1900-01-01 kuni 2030-12-31 kõik kuupäevad koos muu lisainfoga (kas on tööpäev, nädalapäev jne.). Geograafilisteks dimensiooniks on Maa-ja Ruumiameti katastriandmetetest koosnev tabel d_kataster, kus on kõik katastrid kuuluvusega riigi tasandist asutusüksuseni koos koordinaatide ja muu täpsustava infoga. Lisaks on veel katastriandmetest välja võetud d_maakonnad, d_omavalitsused ja d_asustusuksused tabelid, mis kaardistavad Eesti jaotust erineva täpsusega. Kolmandaks on Keskkonnaagentuuri ilmastikuandmete tabel d_ilm, kus on ilmajaamade info tunni täpsusega. Ilmavaatlusandmetele on lisatud ka jaamade asukohad asustusüksuse täpsusega. Kuna ilmastikujaamu pole igas asustusüksuses ega omavalitsuses, siis on tehtud nen-



Joonis 11. Näide täheskeemist [24].

dest keskmistatud kokkuvõtted maakonniti ja tekitatud erineva ajalise täpsusega tabelid d_ilm_maakond_kuu, d_ilm_maakond_paev ja d_ilm_maakond_tund. Dimensioone lisatakse veel vastavalt andmetarbijate vajadustele.

6 Näidiskasutusjuhud

6.1 Vihma mõju liiklemiskiirusele

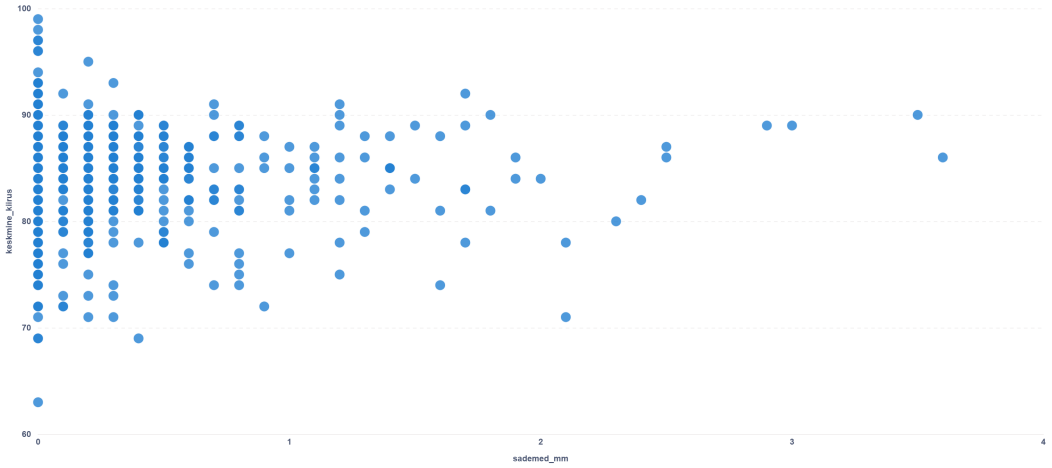
Transpordiameti liiklusloenduse andmestik on hea näide detailsest faktimudelist, mida on võimalik rikastada ja kombineerida teiste andmestikega. Näiteks võib vaadata, kuidas sademed mõjutavad liiklejate keskmist kiirust.
Liiklusloenduse andmestikus on kirjas mõõteseadme ID, vaadeldav sõidurida ja iga tunni kohta loetud ära erinevate sõidukite tüübid ja nende kiirusvahemikud. Seda täiendab mõõteseadmete andmestik, kus on kirjas seadme täpsem asukoht ja muud metaandmed. Nende omavahel liitmist saab teostada järgneva päringuga:

```
SELECT
        f.*,
        d.*
    FROM warehouse.transpordiamet.liiklusloenduse_andmed f
    LEFT JOIN warehouse.transpordiamet.liiklusloenduse_jaamad d
    ON f.seadme_id = d.id
    WHERE d.id IS NOT NULL
```

Liiklusloendusjaamade andmestikes on ära toodud ka maakond, kus mõõteseade asub. Kuna meil on tabel d_ilm_maakond_tund, siis on võimalik läbi maakonna nime ja mõõtmisaja järgi neid omavahel liita. Seega eelmist päringut täiendades saame lisada liiklusandmete juurde ka ilmastikuandmed:

```
select
    f.*,
    d.maakond,
    i.sademed_mm
from "staging"."transpordiamet"."liiklusloenduse_andmed" f
left join "staging"."transpordiamet"."liiklusloenduse_jaamad" d
on f.seadme_id = d.id
left join "warehouse"."keskkonnaagentuur"."d_ilm_maakond_tund" i
on
    d.maakond = i.maakond and
    f.mootmise_aeg = i.aeg
where
    d.id is not null and
    i.maakond is not null and
    i.aeg is not null
```

Valmis mudeli jaoks tuleb teha paar sammu veel. Andmestikus pole toodud täpseid kiirusid, vaid tunni kohta sõidukite arv teatud kiirusvahemikus. Kiiruse muutumise jälgimise lihtsustamiseks on otstarbekas võtta kasutusele mõõtetulemuse umbkaudne keskmine kiirus. Seda saab arvutada, kasutades vahemike keskmised kiiruseid (nt. 40 ja 50 km/h vahemiku keskmine on 45 km/h), korrutades selle vahemikus kokku loetud sõidukite arvuga ja jagades nende summad sõidukite koguarvuga. Lisaks tasub välja



Joonis 12. Sõidukiiruste sõltuvus sademetest.

jätta andmepunktid, kus autosid on liiga vähe ning võtta loendusandmed, kus loeti kokku vähemalt 5 autot. Rakendades seda kõike päringus ja kasutades Metabase'i andmete visualiseerimiseks, on tulemuseks graafik 12. Kahjuks Metabase'i 0.53.4 versioonis veel trendijooned korralikult ei tööta, seega järeldusi graafikult teha ei saa.

6.2 Riigihangete võitjate majandusnäitajad

Riigihangete register pakub avaandmetena infot riigihangete ja nende raames sõlmitud lepingute kohta. Riigihangetel on vahel üles seatud mõni majandusnäitaja kriteerium (ettevõtte müügitulu peab möödunud aastal ületama X €). Seetõttu oleks huvitav teada, millised on keskmise hanke võitja eelmise aasta majandusnäitajad ja kui palju riigihanketest saadav raha panustab tema aasta tulusse.
Esimeseks sammuks selle uurimisel on teha väljavõte riigihangete lepingutest, mis sobivad uurimiseks. Filtreerime välja veel lõpetamata ja ülesöeldud päringud ning summeerime lepingute summad ettevõtte ja aasta järgi:

```
select
    year(solmimise_kuupaev) as aasta,
    toovotja_kood,
    sum(tegelik_maksumus) as lepingute_summa
from warehouse.riigihanked.f_lepingud
    where rikkumised != true and toovotja_kood is not null
    group by year(solmimise_kuupaev), toovotja_kood
```

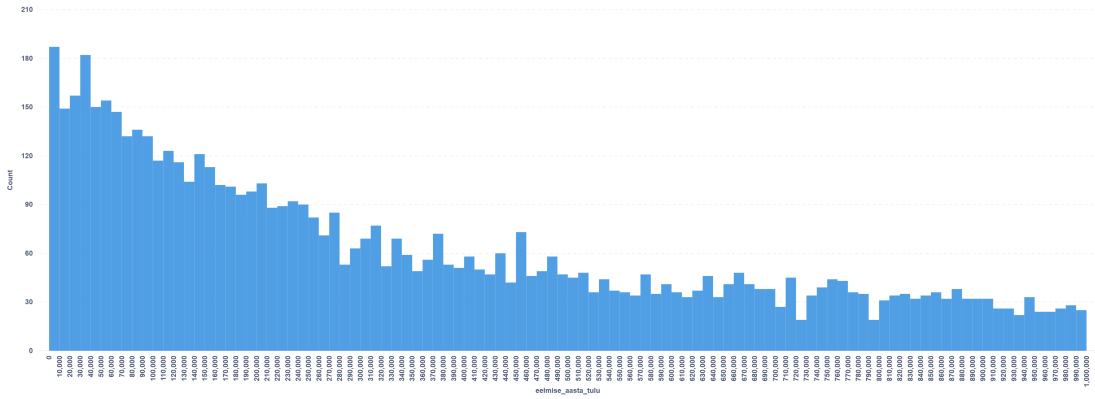
Teiseks tuleb kätte saada ettevõtete aastakasumid. See info on võimalik kätte saada Äriregistri majandusaasta aruannete tabelist järgneva päringuga:

```
select distinct
        registrikood,
        aasta,
        vaartus as tulu
from warehouse.ariregister.f_majandusaasta_aruanded
    where elemendi_nimetus = 'Revenue'
```

Kombineerides nende kahe päringu tulemused läbi ettevõtte koodi ja aasta ongi võimalik kätte saada ettevõtte eelmise ja käesoleva aasta tulu võrdluses riigihangetest saadud rahaga:

```
select
    h.aasta,
    h.toovotja_kood as registrikood,
    h.lepingute_summa,
    a.tulu as eelmise_aasta_tulu,
    b.tulu as selle_aasta_tulu
from hanke_lepingud h
left join aastakasumid a
  on h.toovotja_kood = a.registrikood and h.aasta = a.aasta - 1
left join aastakasumid b
  on h.toovotja_kood = b.registrikood and h.aasta = b.aasta
where h.lepingute_summa is not null
```

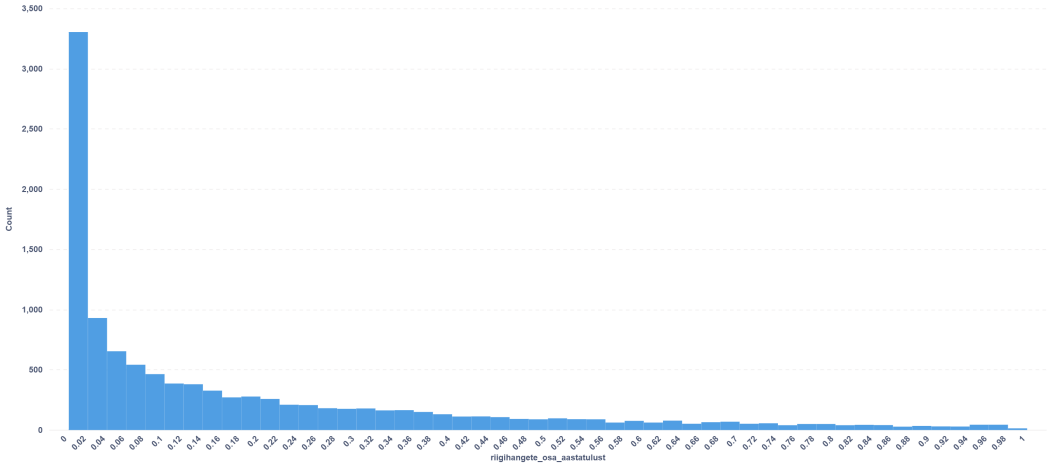
Moodustades Metabase'is selle päringu pealt eelmise aasta tulude histogrammi, saame tulemuseks joonise 13. Jooniselt võib välja lugeda, et enamiku hangetest teostavad



Joonis 13. Riigihangete võitjate eelmise aasta müügitulu histogramm.

mikroettevõtted, kelle tulu jääb alla 900 000 €. Tekitades uue tulba, kus on jagatud ettevõtte tulu riigihangetest saadud summaga, on võimalik välja uurida, palju moodustavad riigihangetest saadud rahad firmade tuluallikatest. Seda Metabase'is histogrammina

visualiseerides saame joonise 14, kust võib välja lugeda, et enamike firmade tuluallikad pärinevad siiski erasektorist.



Joonis 14. Riigihangete osakaal ettevõtete müügitulust.

7 Kokkuvõte

Käesoleva magistritöö eesmärk oli pakkuda alusraamistikku avalike andmete kasutamisega seotud probleemide lahendamiseks. Selle eesmärgi saavutamiseks töötati välja uus keskne avaandmete analüütiline andmesüsteem nimega Avaandmeait. See platvorm koondab seni hajali olnud Eesti avaliku sektori andmed ühte andmejärvemajja ning pakub nendele andmetele standardiseeritud juurdepääsuvõimalusi. Sellega paraneb avaandmete kättesaadavus: andmekasutajad ei pea enam andmeid käsitsi otsima mitmest erinevast allikast, vaid saavad need struktureeritult kätte ühest kesksest platvormist.
Pakutud lahendus tõstab oluliselt ka andmete kvaliteeti ja sidusust. Ühtlustatud tabelistruktuuride ja ühise päringukihi abil muutuvad erinevatest allikatest pärinevad andmestikud paremini võrreldavaks ning seostatavaks. Andmete kvaliteedi parandamiseks viidi esialgu heterogeensetes formaatides olnud andmed üle ühtsesse Icebergi tabeliformaati ning ühtlustati andmetüübid ja korrastati andmestikud. Seeläbi väheneb andmekasutajate töökoormus andmete puhastamisel ning nad saavad kiiremini alustada sisulist andmeanalüüsi.
Lisaks praktilise platvormi loomisele käsitleb magistritöö ka olulisi kitsaskohti, mis takistavad avaandmete tõhusat kasutust Eestis, ning pakub soovitusi nende lahendamiseks. Esmalt tõstetakse esile masinloetavuse probleem. Avaandmete portaalis olevatest andmestikest on vaid 29% kohe masinloetavad ning 46% masinloetavad vaid juhul, kui nende struktuur on korrektne. Seetõttu on oluline, et andmed avaldataks standardsetes, masinloetavates formaatides ning selge ja üheselt mõistetava struktuuriga. Teiseks probleemiks on andmete vormistuslik ebaühtlus. Sageli esinevad samad väärtused eri andmestikes erinevates formaatides – näiteks koordinaadid võivad olla ühes andmestikus EPSG:3301 formaadis ja teises GCS formaadis. Samuti on levinud halb praktika, kus Exceli faile struktureeritakse visuaalse vorminduse abil (nt mitmerealised päised, tühjad read), mis muudab masinloetavuse keerukaks. Kolmandaks probleemiks on andmete liigne agregeerimine. Analüütiliseks kasutuseks on oluline, et andmed oleksid võimalikult detailsed. Selle asemel, et andmeid liigselt agregeerida, peaksid andmevaldajad kasutama privaatsuskaitse meetodeid, nagu pseudonüümimine või anonüümimine. Need võimaldavad kaitsta isikuandmeid ilma, et kaoks andmestiku analüüsiväärtus.
Avaandmeaida arendustöö jätkub ka edaspidi. Võimalik on veel teha infrastruktuuri toimimist kiiremaks, turvalisemaks ja veakindlamaks. Lisaks on veel paljusid avaliku sektori andmestikke platvormile lisamata. Kokkuvõttes on magistritöö käigus loodud tugev alusraamistik, mis võimaldab seniseid avaandmete kasutamise kitsaskohti oluliselt vähendada ning toetab andmepõhise otsustusprotsessi edasist arengut Eestis.

Viidatud kirjandus

[1] J. Berends, W. Carrara, W. Engbers, and H. Vollers, "Recommendations for open data portals: From setup to sustainability," Study report OA-03-20-042-EN-N, European Data Portal, Luxembourg, 2020.

[2] E. Liit, "Open data impact." https://data.europa.eu/en/publications/open-data-impact, 2020. Loetud: 2025.03.24.

[3] Euroopa Liit, "Avaandmed ja avaliku sektori valduses oleva teabe taaskasutamine." https://eur-lex.europa.eu/ET/legal-content/summary/open-data-and-the-reuse-of-public-sector-information.html. Loetud: 2025.03.31.

[4] M. Page, E. Hajduk, E. N. L. Arriëns, G. Cecconi, and S. Brinkhuis, "2023 open data maturity report," tech. rep., European Commission (data.europa.eu), Luxembourg, 2023.

[5] Ott Velsberg, "Estonia achieves trendsetter status in open data benchmark study." https://e-estonia.com/estonia-trendsetter-open-data. Loetud: 2025.03.11.

[6] K. McBride, M.Õlesk, A. Kütt, and D.Šhysh, "Systemic change, open data ecosystem performance improvements, and empirical insights from estonia: A country-level action research study," *Information Polity*, vol. 25, no. 3, pp. 377–402, 2020.

[7] Euroopa Liit, "Open data maturity." https://data.europa.eu/en/publications/open-data-maturity. Loetud: 2025.03.31.

[8] Vabariigi Valitsus, "Eesti avaliku teabe masinloetav avalikustamise roheline raamat," Tech. Rep. Versioon 0.9, Majandus- ja Kommunikatsiooniministeerium, Tallinn, 2014.

[9] Eesti Avaandmed, "Dcat rakendusprofiil euroopa andmeportaalidele. versioon 2.0.1." https://avaandmed.eesti.ee/instructions/dcat-application-profile. Loetud: 2025.03.31.

[10] N.Ž. Government, "Formats for open data." https://data.govt.nz/catalogue-guide/releasing-data-on-data-govt-nz/formats-for-open-data-machine-readable-and-human-readable. Loetud: 2025.04.01.

[11] Riigikogu, "Isikuandmete kaitse seadus." https://www.riigiteataja.ee/akt/104012019011. Loetud: 2025.04.01.

[12] D. Bogdanov, E. Brito, P. Etti, L. Kamm, P. Laud, T. Mällo, A.Õstrak, K.Šein, R. Talviste, and M. Toomsalu, "Privaatsuskaitse tehnoloogiate eestis rakendamise teekaart: Aruanne," Report D-16-215, Majandus- ja Kommunikatsiooniministeerium, Tallinn, Estonia, Mar. 2023.

[13] E. avaandmed, "Avaandmete loomise juhend." https://avaandmed.eesti.ee/instructions/avaandmete-loomise-juhend. Loetud: 2025.04.20.

[14] J. Henninger, E.Šwanson, L. Noe, T. Wahabzada, and E. Baldi, "2024/25 open data inventory (odin): A decade of progress," tech. rep., Open Data Watch, Washington, DC, 2025. 7th edition, assessing data coverage and openness in 197 countries.

[15] T. Hirst, "When machine readable data still causes "issues" – wrangling dates..." https://blog.ouseful.info/2012/11/27/when-machine-readable-data-still-causes-issues-wrangling-dates/. Loetud: 2025.05.11.

[16] K. Porovarde, "Eesti avaandmete portaali probleemide kaardistamine ja andmeanalüüsi töölaua loomine," bakalaureusetöö, Tartu Ülikool, 2024.

[17] C. Gröger, "Industrial analytics–an overview," *it-Information Technology*, vol. 64, no. 1-2, pp. 55–65, 2022.

[18] T. Davenport, *Big data at work: dispelling the myths, uncovering the opportunities*. Harvard Business Review Press, 2014.

[19] D. Larson and V. Chang, "A review and future direction of agile, business intelligence, analytics and data science," *International Journal of Information Management*, vol. 36, no. 5, pp. 700–710, 2016.

[20] DataBricks, "Data lakehouse." https://www.databricks.com/glossary/data-lakehouse. Loetud: 2025.04.29.

[21] Oracle, "What is oltp?." https://www.oracle.com/database/what-is-oltp/. Loetud: 2025.05.17.

[22] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM computing surveys (CSUR)*, vol. 15, no. 4, pp. 287–317, 1983.

[23] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *ACM Sigmod record*, vol. 26, no. 1, pp. 65–74, 1997.

[24] R. Kimball and M. Ross, *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013.

[25] W. H. Inmon, *Building the Data Warehouse*. Indianapolis, IN: Wiley Publishing, 4 ed., 2005.

[26] S.Š. Ally and N. Khan, "Data warehouse and bi to catalize information use in health sector for decision making: A case study," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 92–97, IEEE, 2016.

[27] H. P. Putro, E.Šuciana, H. L. H.Š. Warnars, M. K. Muyeba, A. F. Sianipar, and D. Jonathan, "Data warehouse implementation for mixing process in tire manufacture," in *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*, pp. 147–155, IEEE, 2018.

[28] F. Carifio Jr and M. Jahnke, "Bank of america case study: the information currency advantage," 1998.

[29] M. U. Shaikh, Y. Al-Bastaki, A. Al-Alawi, and S. Hamad, "The implementation of datawarehouse in batelco: a case study evaluation and recommendation," in *Proceedings. 2004 International Conference on Information and Communication Technologies: From Theory to Applications, 2004.*, pp. 445–446, IEEE, 2004.

[30] James Dixon, "Pentaho, hadoop, and data lakes." `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`. Loetud: 2025.05.11.

[31] M. Armbrust, A. Ghodsi, R. Xin, and M.Žaharia, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," in *Proceedings of CIDR*, vol. 8, p. 28, 2021.

[32] SAP, "What is a data warehouse?." `https://www.sap.com/mena/products/data-cloud/datasphere/what-is-a-data-warehouse.html`. Loetud: 2025.05.11.

[33] A. A. Harby and F.Žulkernine, "From data warehouse to lakehouse: A comparative review," in *2022 IEEE international conference on big data (big data)*, pp. 389–395, IEEE, 2022.

[34] J.Šchneider, C. Gröger, A. Lutsch, H.Šchwarz, and B. Mitschang, "The lakehouse: State of the art on concepts and technologies," *SN Computer Science*, vol. 5, no. 5, p. 449, 2024.

[35] Iceberg, "Documentation." `https://iceberg.apache.org/docs/nightly/`. Loetud: 2025.05.05.

[36] D. Pálma, "Apache iceberg vs hudi: Key features, performance & use cases." https://estuary.dev/blog/apache-iceberg-vs-apache-hudi/, December 2024. Loetud: 2025.05.11.

[37] E.Šmith, "Apache iceberg vs delta lake: What are the differences?," Sept. 2024. Accessed 15 May 2025.

[38] R.Šethi, M. Traverso, D.Šundstrom, D. Phillips, W. Xie, Y.Šun, N. Yegitbasi, H. Jin, E. Hwang, N.Šhingte, *et al.*, "Presto: Sql on everything," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1802–1813, IEEE, 2019.

[39] Z. Wang and A. Aiken, "Efficient fault tolerance for pipelined query engines via write-ahead lineage," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 436–448, IEEE, 2024.

[40] Trino, "Documentation." https://trino.io/docs/current/admin/properties.html. Loetud: 2025.05.05.

[41] P. Jain, P. Kraft, C. Power, T. Das, I.Štoica, and M.Žaharia, "Analyzing and comparing lakehouse storage systems.," in *CIDR*, 2023.

[42] R. R. Vanga, "Etl vs elt: Evolving approaches to data integration," *International Journal for Multidisciplinary Research (IJFMR)*, vol. 6, pp. 1–10, Sep–Oct 2024.

[43] P.Šawadogo and J. Darmont, "On data lake architectures and metadata management," *Journal of Intelligent Information Systems*, vol. 56, no. 1, pp. 97–120, 2021.

[44] D. Naphade, "The evolution and modernization of data pipeline architectures," *European Journal of Computer Science and Information Technology*, vol. 13, pp. 42–53, 03 2025.

[45] A. Airflow, "Documentation." https://airflow.apache.org/docs/apache-airflow/stable/index.html#. Loetud: 2025.04.21.

[46] Astronomer, "Documentation." https://www.astronomer.io/docs/learn/. Loetud: 2025.04.21.

[47] dbt Labs, "What is dbt?." https://docs.getdbt.com/docs/introduction. Loetud: 2025.03.24.

[48] T. Ferreira, I. Pedrosa, and J. Bernardino, "Evaluating open source business intelligence tools using osspal methodology.," in *KDIR*, pp. 283–288, 2017.

[49] C. Lennerholt, J. Van Laere, and E.Šöderström, "User-related challenges of self-service business intelligence," *Information Systems Management*, vol. 38, no. 4, pp. 309–323, 2021.

[50] P.Štrengholt, *Building Medallion Architectures: Designing with Delta Lake and Spark*. O'Reilly Media, 2025.

[51] DuckDB, "Duckdb." `https://duckdb.org`. Loetud: 2025.05.10.

Lisad

I. Infrastruktuuri koodihoidla

Näidiskood jooksutamaks Avaandmeaida infrastruktuuri on leitav siit:
https://github.com/shanazar/data_lakehouse

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kristjan Lõhmus**,
(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Avaandmeait,

   mille juhendaja(d) on Kristo Raun ja Priit Kongo,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kristjan Lõhmus
15.05.2025