Daniel Finnerman

Tietokantataulujen arkistointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

18.8.2022

Tiivistelmä

Tekijä:	Daniel Finnerman
Otsikko:	Tietokantataulujen arkistointi
Sivumäärä:	32 sivua
Aika:	18.8.2022

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaaja:	Osaamispäällikkö Janne Salonen

Insinöörityössä kehitettiin tietokantataulujen arkistoinnin suorittava sovelluskomponentti eräälle yritykselle. Työ toteutettiin työntekijänä yrityksessä.

Työn tavoitteena oli arkistoida yrityksen sisäisiä tietokantataulukoita, ja täten parantaa tulevien tietokantahakujen suorituskykyä. Tulevien tietokantahakujen suorituskyky on parempi, kun ajankohtaista tietoa on vähemmän, koska vanhemmat tietueet on arkistoitu.

Insinöörityö tehtiin yrityksen potilastietojärjestelmään OutSystems -low-code-ohjelmistokehitysalustaa käyttäen. Arkistointilogiikasta luotiin täysin automatisoitu komponentti. Työssä tutustuttiin myös low-code-kehitykseen, relaatiotietokantoihin ja tiedon arkistoinnin käytäntöihin.

Arkistointilogiikka paransi tietokantatauluihin kohdistettujen hakujen nopeutta merkittävästi. Menetelmää kehitettiin aluksi vain yhdelle tietokantataululle, mutta se toteutettiin niin, että sitä voidaan käyttää tulevaisuudessa pohjana myös laajemmin muissa tauluissa yrityksen tietokannassa.

Avainsanat: tiedon arkistointi, low-code-ohjelmointi

Abstract

Author:	Daniel Finnerman
Title:	Archiving of database tables
Number of Pages:	32 pages
Date:	18 August 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Mobile Solutions
Supervisor:	Janne Salonen, Head of School (ICT)

The purpose of this thesis was to develop an application component that archives database tables for a company. The thesis was carried out as an employee in the company.

The goal of the work was to archive company's internal database tables, and thereby improve the performance of future database searches. The performance of future database searches will be better when there is less data to be dealt with because older records have been archived.

The thesis project was implemented on the company's medical records system using the OutSystems low-code software development platform. A fully automated component of the archiving logic was created. The thesis also introduces low-code-development, relational databases and data archiving practices.

The archiving logic significantly improved the speed of searches on the company database tables. The method was initially developed for a single database table, but was refined so that it can be used in the future as a basis for a wider range of other tables in the company's database.

Keywords: data archiving, low-code-programming

Sisällys

1 Johdanto

Insinöörityössä perehdytään tietokantataulujen arkistointilogiikan kehittämiseen OutSystems-low-code-ohjelmistokehitysalustaa hyödyntäen ja luodaan sovelluskomponentti eräälle yritykselle.

Yrityksen tuote on potilastietojärjestelmä, jonka tietokannat ovat suuria ja niihin tallennetaan tietoa jatkuvasti. Tallennettu tieto on terveydenhuoltoalan velvoitteiden mukaan myös arkistoitava ja saatava tarvittaessa käyttöön myöhemmin. Kuitenkin tietokantojen tietomäärän kasvaessa niihin kohdistuvien tietokantakyselyjen vastausajat pitenevät jatkuvasti.

Yritys on jo aikaisemmin käyttänyt tietokantojen puhdistuksessa erilaisia siihen tarkoitettuja valmiita komponentteja, mutta arkistointia ei ole vielä tehty. Lopulta joidenkin tietokantataulujen kasvettua suuriksi niihin kohdistuvat kyselyt ovat niin hitaita, että käytettävissä olevien ohjelmistojen suorituskyky heikkenee merkittävästi. Vaikka hakuja voidaan nopeuttaa tietokantakyselyiden parametrien tarkennuksilla ja tietokantataulujen indeksoinnilla, on yrityksen ohjelmistokehitysalusta OutSystems antanut dokumentaatiossaan vaihtoehtoja tiedon arkistointiin liittyvistä metodeista.

Opinnäytetyössä tutkitaan, kuinka paljon tietokantahakujen suorituskykyä voidaan parantaa arkistoinnin jälkeen vertailemalla eri tietokantatauluihin kohdistettavia hakuja. Työssä tutustutaan myös low-code-ohjelmointiin, relaatiotietokantoihin ja tiedon arkistoinnin käytäntöihin OutSystemsin dokumentaation pohjalta.

2 Low-code

Low-code-ohjelmistokehitys on menetelmä, joka mahdollistaa yritystason sovelluksien kehityksen minimoimalla tai jopa poistamalla perinteisen käsin ohjelmoimisen tarpeen. Tämä nopeuttaa sovelluksen kehittämistä ja tuotantoon viemistä. [The Low-Code Development Guide 2022.]

Low-code-ohjelmointi on maailmanlaajuisesti kasvava ilmiö. Sen markkinoiden koko vuonna 2020 oli 13,2 miljardia dollaria ja sen ennustettiin kasvavan 45,5 miljardiin dollariin vuoteen 2025 mennessä. [Low-Code Development Platform Market 2020.] COVID-19-pandemia on luonut lisääntyneen tarpeen automatisoida eri prosesseja ja lisätä digitaalisen muutoksen aloitteita. Low-code-alustojen kasvu on vastannut tarpeisiin nopeuttamalla ja helpottamalla kehitystyön kulkua. [Low-Code 2021.] Kasvua vauhdittaa myös yritysten tarve saada nopeasti päivityksiä sovelluksiinsa ja vähentää IT-taitojen puuttumisesta johtuvia kehitykseen liittyviä ongelmia. [Low-Code Development Platform Market 2020.] Koska low-code-ohjelmointi ei vaadi perusteellisia perinteisen ohjelmoinnin taitoja, voivat esimerkiksi sovelluksen markkinoinnin parissa työskentelevät osallistua sovellusten kehitykseen ja testaukseen. [Pratt 2021.]

Tästä huolimatta low-code-kehitysalustoja voi hyödyntää myös kokeneemmat sovelluskehittäjät. Vähäinen ohjelmointikokemuksen vaatimus mahdollistaa joustavuuden kehittäjän ohjelmointitaustassa. Monet liiketoimintasovellukset vaativat jonkun tietyn ohjelmointikielen osaamista, jolloin käytettävissä olevien ja osaavien kehittäjien määrä saattaa olla niukka. Low-code mahdollistaa kehittämisen työkalut myös useammalle sovelluskehittäjälle. [Low-Code 2021.]

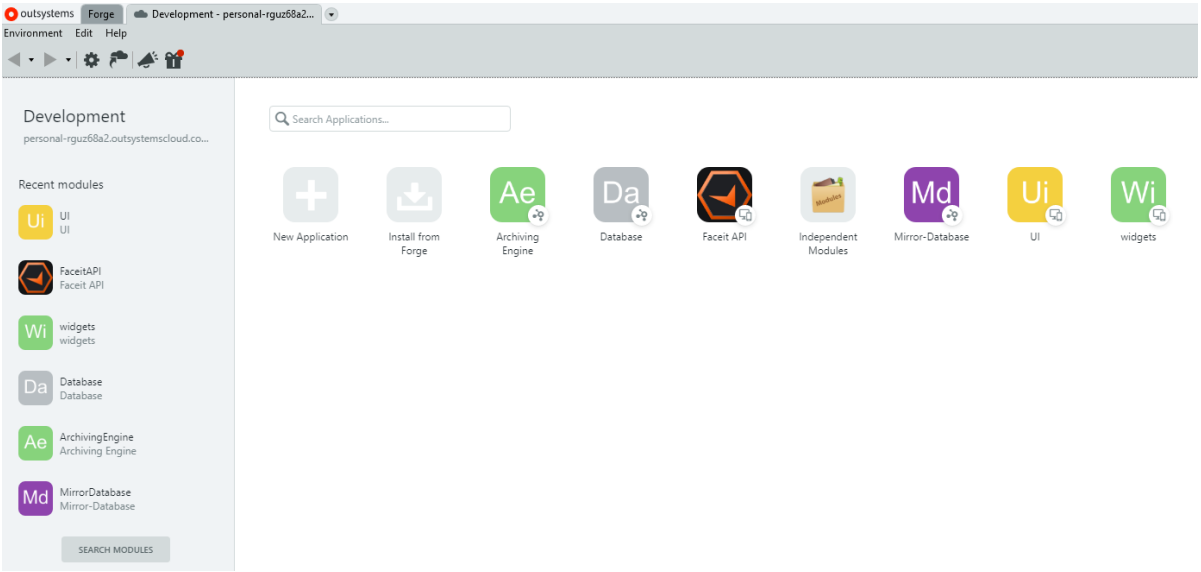
Esimerkkejä yleisimmistä low-code-ohjelmointiympäristöistä ovat

- Appian
- Mendix
- OutSystems
- Salesforce
- Microsoft PowerApps. [Pratt 2021.]

2.1 Low-code-kehitys

Low-code-ohjelmointiympäristössä on graafinen käyttöliittymä, jonka avulla käyttäjä voi luoda ja yhdistellä komponentteja sekä käyttää ohjelmointirajapintoja. Yksittäinen sovellus on moduuli, johon sovelluksen logiikka luodaan. Useimmat yritysten low-code-sovellukset rakentuvat useimmista moduuleista, joilla kaikilla on oma tehtävänsä. Yksinkertaisessa sovelluksessa yksi moduuli voi vastata toiminnallisista backend-ominaisuuksista ja toinen visuaalisista frontend-ominaisuuksista. [Pratt 2021.]

Suuremmissa liiketoimintasovelluksissa yksi moduuli saattaa olla useamman sovelluksen käytössä. Eri moduulit voivat kommunikoida ja jakaa tietoja keskenään, mikäli ne asetetaan riippuvuussuhteeseen toisiinsa. Moduulit jaotellaan lähtökohtaisesti palvelu- ja käyttöliittymämoduuleihin. Palvelumoduulit vastaavat backend-logiikasta, kuten integraatioista, prosesseista, ajastimista ja tietokannoista. Käyttöliittymämoduulit vastaavat käyttöliittymästä ja sen logiikasta, paikallisista tietokannoista ja istuntomuuttujista. [Use Services to Expose Functionality 2022.]



Kuva 1. OutSystems Service Studio 11 -ohjelmistokehitysympäristö, jossa useita moduuleita.

Käyttäjä rakentaa funktiot visuaalisesti vuokaavioiden avulla. Funktioiden ohjelmakoodi rakentuu ohjelmointiympäristössä taustalla sitä mukaa, miten käyttäjä luo kaavionsa. [Pratt 2021.]

Kuva 2. Yksinkertainen funktio OutSystems Service Studiossa, jossa suoritetaan API-kutsu ja asetetaan palautunut arvo muuttujan arvoksi.

```
const APIFetch = await fetch('https://open.faceit.com/data', {
     headers: {
          'accept': application/json',
          'Authorization': 'Bearer ' + apiKey
     }
});

const FaceitVal = await APIFetch.json();
```

Esimerkkikoodi 1. JavaScript -kielen funktio, jonka OutSystems suorittaa selaimessa taustalla, kun käyttäjä luo ja ajaa OutSystems Service Studiossa kuvan 2 kaltaisen funktion.

Low-codeen on mahdollista sisällyttää myös muita ohjelmointikieliä tai muokata itse valmiita funktioita, elementtejä tai toiminnallisuuksia. Tämä kuitenkin vaatii perinteisen ohjelmoinnin osaamista. [Kotilainen 2018.]

2.2 Low-coden hyödyt

Low-coden suurimpana hyötyinä verrattuna perinteiseen ohjelmointiin ovat kehityksen ajankäytön vähentyminen ja kustannustehokkuuden parantuminen. [Airas 2021.] Tällöin projektit, joita ei aikaisemmin ajanvuoksi ehditty tai kustannusten takia voitu tehdä, voidaan nyt mahdollistaa. [Low code – mistä kyse? 2020.] Noin 80–95 % kaikista sovelluksista on mahdollista luoda low-code-alustalla. [Kotilainen 2018.]

Monet yritykset ovat huomanneet kehityksessään osaamisvajetta. Korkean osaamistason omaavien sovelluskehittäjien palkkaaminen on yrityksille kallista etenkin, jos heidän on työskenneltävä sellaisten sovellusten ja järjestelmien parissa, jotka ovat toiminnallisuuksiltaan vanhentuneita. Low-code mahdollistaa osaamisen lisäämisen olevassa olevaan työvoimaan. Esimerkiksi erään yrityksen johto käytti low-code-ohjelmointiympäristöä muuttaakseen COBOL-kehittäjät eli kaupallishallinnollisten järjestelmien korkean tason ohjelmointikielen osaavat sovelluskehittäjät web-kehittäjiksi. Low-code ohjelmistokehitys mahdollistaa myös sovelluksen kehityksen kerralla monenlaiselle laitteelle. Ristikkäisalustatoiminnallisuuden avulla päivitykset luodaan vain kerran, ja tuloksena on sujuva käyttökokemus työpöytä- ja mobiililaitteilla. [Pros and Cons of Low Code Development.]

2.3 Low-coden haasteet

Erittäin monimutkaiset prosessien, kuten rahoitusalan liiketoimintaprosessien luominen low-code-alustoilla saattaa olla haastavaa. Sovelluskehitys avaimet käteen -periaatteella tehokkaasti lyhyemmässä ajassa heikentää mahdollisuutta kustomoida sovellusta toiminnallisesti ja ulkonäöllisesti. Näissä tapauksissa käytetään kuitenkin lähtökohtaisesti perinteistä ohjelmointia sovelluskehityksessä.

Monilla teknologia-alan yrityksillä on myös huolta siitä, että he joutuvat liian riippuvaisiksi low-code-ohjelmistoympäristöjen toimittajista ja niiden asiantuntijapalveluista ja lisensioiduista teknologioista. Yritykset haluavat omaa vapautta sovelluskehityksessä riippumatta siitä, mitä ohjelmointiympäristöä käytetään.

Koska low-code-lähestymistavan tavoitteena on tuoda sovelluskehitys useamman loppukäyttäjän ulottuville, on huolenaiheena myös ns. varjo-IT:n lisääntyminen. [Arh 2021.] Varjo-IT:tä tapahtuu silloin, kun IT-osaston ulkopuoliset henkilöt tuottavat ohjelmistoja tai tekevät muutoksia ohjelmistoon IT-osaston tietämättä. Tällöin perustiedot eri liiketoimintayksiköiden välillä saattaa eriytyä, tietojen omistajuuden alkuperä jäädä epäselväksi ja tietoturvallisuus heikentyä.

Myös nimeämiseen, konfigurointiin ja dokumentointiin liittyvät käytännöt saattavat jäädä toteutumatta. [Mattila 2019.] Näitä ongelmia voidaan kuitenkin vähentää tai jopa estää teknisten valvontatoimien ja roolipohjaisten oikeuksien avulla. Eri rooleilla voi tällöin toteuttaa eri toimenpiteitä ohjelmistokehitysalustalla. [Arh 2021.]

2.4 OutSystems

OutSystems on Portugalissa perustettu yritys, joka tarjoaa low-code-ohjelmistoympäristöjä. OutSystems on low-code-ohjelmistokehitysalustojen markkinajohtaja, jolla on yritysasiakkaita 52 maassa. OutSystemsin päätuote, OutSystems Service Studio 11 on sovelluskehitystyökalu, jota käytetään web- ja mobiilisovelluskehityksessä, vanhentuneiden toiminnanohjausjärjestelmien ja asiakkuushallintatyökalujen modernisoinnissa ja esimerkiksi verkkopankkien kehityksessä. [Solita ja OutSystems low-code-yhteistyöhön Suomessa ja Ruotsissa 2019.]

Web- ja mobiilisovelluskehityksessä OutSystems Service Studio 11:ssa luoduista komponenteista muodostuu ReactJS -ohjelmistokehystä käyttävää JavaScript -ohjelmakoodia, joka ajetaan selaimessa. [Modern Web Apps with ReactJS and OutSystems.]

Tässä opinnäytetyössä low-code-ohjelmistokehitysalustana toimii OutSystems, jossa luodaan web-sovellus ja ajetaan SQL-kyselykieltä OutSystemsin kautta yrityksen relaatiotietokannoissa.

3 Relaatiotietokannat

Relaatiotietokannat ovat relaatiomalliin perustuvia tietokantoja. Relaatiotietokanta on suosituin tietokantatyyppi, koska käyttäjän on helppo ymmärtää niitä, tietokantataulujen välille voi luoda yhteyksiä ja ne tarjoavat operaattoreita, kuten SQL-kyselykieliä niiden tietojen käsittelyyn. Ero ei-relaatiotietokantoihin eli NoSQL-tietokantoihin perustuu siihen, miten tiedot tallennetaan ja järjestetään.

NoSQL-tietokannat eivät tallenna tietoa taulukkomuotoisella tavalla, kuten relaatiotietokannat, vaan yksittäisinä tiedostoina, jotka eivät liity toisiinsa. [What is a relational database? 2022.]

3.1 Relaatiotietokantojen rakenne

Projektissa arkistoidaan yrityksen relaatiotietokantoja. Relaatiotietokanta perustuu tiedon jakamiseen eri tietokantatauluihin ja niiden välisiin yhteyksiin. Tietokannassa tauluja voi olla yksi tai useammassa tapauksessa useita. [Luukkainen & Vihavainen 2017.] Relaatiotietokanta mahdollistaa sen, että mikä tahansa taulu voidaan liittää toiseen, mikäli niille on määritelty jokin yhteinen ominaisuus. [What is a relational database? 2022.] Taululla on eri ominaisuuksia, joita kutsutaan attribuuteiksi. Jokaiselle attribuutille on taulussa oma sarakkeensa. Tallennettava tieto rakentuu riveistä eli tietueista, jotka ovat tietokantataulun yksittäisiä ilmentymiä. Rivejä yksilöidään toisistaan lähtökohtaisesti avaimilla. Avain voi olla pääavain, joka viittaa taulun yksilöivään attribuuttiin tai viiteavaimeen, joka viittaa toisessa taulussa olevaan käsitteeseen. [Luukkainen & Vihavainen 2017.]

Pelaaja Id	Pelaaja Pelitunnus	Pelaaja Taso	Pelaaja Luotu	Pelaaja ProfiiliVahvistettu
3	talent	8	2019-11-01	true
2	peluri	5	2022-08-23	false
1	aimgod	7	2021-03-05	true

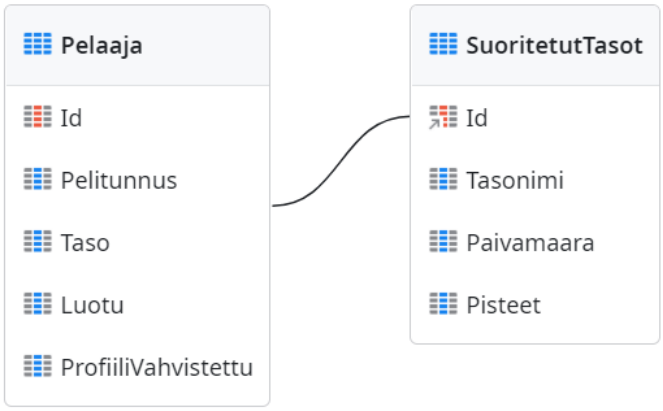
Kuva 3. Esimerkkitaulu "Pelaaja", johon on tallennettu kolme riviä eli tietuetta. Taulu esittää kuvitteellisen videopelin alustalle rekisteröityneitä käyttäjiä.

Kuvan 3 tietokantataulussa on kolme riviä eli tietuetta. Rivit ovat yksilöity Id-attribuutilla, joka on tietokantataulun pääavain. Tällöin esimerkiksi Pelitunnus-attribuutti voi olla useamman eri tietueen kohdalla sama, mutta käyttäjät voidaan silti erottaa toisistaan yksilöllisellä attribuutilla. Kuva 3 esittää myös sarakkeiden eri formaatteja eli datamuotoja. Taso-attribuutti on numeerinen tieto, Luotu-attribuutti kertoo tietueen luontipäivämäärän ja ProfiiliVahvistettu on totuusarvomuuttuja, joka kertoo, onko käyttäjä vahvistanut profiilinsa vai ei.

SuoritetutTasot Id	SuoritetutTasot Tasonimi	SuoritetutTasot Paivamaara	SuoritetutTasot Pisteet
1 aimgod	Level 2	2022-03-11	8
2 peluri	Level 1	2022-06-19	12
3 talent	Level 1	2022-07-01	15
1 aimgod	Level 1	2022-03-10	11

Kuva 4. Tietokantataulu "SuoritetutTasot", joka on yhteydessä "Pelaaja"-tietokantatauluun.

Kuvan 4 tietokantataulu esittää, miten yhdellä käyttäjällä voi olla useampi suoritettu taso, mutta jokaiseen tason suoritukseen voi liittyä vain yksi käyttäjä. Tässä tietokantataulussa Id-attribuutti on viiteavain, joka viittaa Pelaajat-tietokantatauluun.



Kuva 5. Kahden relaatiotietokantataulun välinen yhteys tietokannassa.

3.2 Relaatiotietokantamallin hyödyt

Relaatiotietokantamallin hyödyt perustuvat sen tapoihin esittää tietoa selkeästi ja mahdollistaa helpon pääsyn eri taulujen välisiin tietoihin. Siksi relaatiotietokantoja käyttävät yritykset ja organisaatiot, jotka hallinnoivat suuria määriä tietoa. [What is a relational database? 2022.] Näitä yrityksiä voi olla esimerkiksi suuret verkkokaupat, joilla on paljon asiakkaita. Relaatiotietokantaa voi

hyödyntää missä tahansa tarpeessa, jossa halutaan luoda tietojen välille yhteksiä ja hallinnoida tätä prosessia turvallisesti ja johdonmukaisesti. [What is a Relational Database (RDBMS)? 2022.]

Hyötyjä ovat myös taulukoiden sisällön ja tietojen välisten suhteiden helppo lisäys, päivitys tai poisto ilman, että tietokantarakennetta tarvitsee muuttaa. Näiden toimintojen suorittamista voidaan myös rajoittaa käyttäjien käyttöoikeuksilla. Relaatiotietokannat käyttävät myös normalisointitekniikkaa, joka vähentää tietojen redundanssia eli toistoa ja parantaa tietojen eheyttä. [What is a relational database? 2022.]

3.3 OutSystems-aggregaatit

Aggregaatit ovat visuaalisia elementtejä, joilla OutSystemsissä voidaan hakea relaatiotietokantatauluista tietoa optimoiduilla ja täsmennetyillä kyselyillä. Ne tukevat useiden tietokantataulujen suodatusta ja tietueiden lajittelua sekä tuovat tietokantapalvelimelta vain ne taulujen attribuutit, mitä kehitettävä sovellus käyttää toiminnoissaan tai näyttää käyttöliittymässään. [Aggregate 2022.]

Opinnäytetyössä tehtävän arkistointilogiikan luonnissa käytetään aggregaatteja tietokantataulujen suodatuksessa.

4 Tiedon arkistointi

Prosessi, jossa tietoa siirretään sen ensisijaisesta tallennustilasta toissijaiseen tallennustilaan, kutsutaan arkistoinniksi. Toissijaisessa tallennustilassa data koostuu tiedoista, joihin ei enää tarvitse suorittaa useita tietokantahakuja. Näitä tietoja saatetaan kuitenkin tarvita myöhempää käyttöä varten tai niiden säilytyksen voi velvoittaa eri toimialojen säädökset ja määräykset tai valtion määrittelemät lait. [Data Archiving 2022.] Niiden rikkominen voi johtaa seuraamuksiin, kuten vahingonkorvauksiin, sakkoihin tai erilaisten sopimusten mitätöintiin. Tietojen arkistointi auttaa yrityksiä välttämään edellä mainittuja tilanteita säilyttämällä tietoja pitkällä aikavälillä ja tarvittaessa yhdistämään tiedot esimerkiksi

auditointien yhteydessä. Säädökset siitä, miten kauan tietoja on säilytettävä, vaihtelevat toimialoittain ja sen mukaan, millaista tietoa käsitellään. [Posey & Yu 2018.]

Tämän opinnäytetyön ohjelmointiprojektissa arkistoidaan potilastietojärjestelmän tietoja. Niiden arkistoinnista vastaa se terveydenhuollon yksikkö, jossa ne on alun perin laadittu. [Potilasasiakirjojen säilyttäminen 2020.]

Tässä osiossa keskitytään tiedon arkisointiin OutSystems -low-code-alustalla.

4.1 Ero varmuuskopiointiin

Varmuuskopioitu tieto on kopio alkuperäisestä tiedosta. Arkistoitu ja varmuuskopioitu tieto ovat tallennettu toissijaiseen muistiin. Lähtökohtaisesti niiden tallennusväline, kuten palvelimen kiintolevy, on kapasiteetiltaan suurempi, mutta suorituskyvyltään heikompi. Suorituskykyyn vaikuttaa myös tietoarkistoille ja varmuuskopioille tyypillinen suurempi tallennetun tiedon määrä. Näillä kahdella tallennusmuodoilla on kuitenkin eri tarkoitus. Arkistoja käytetään harvoin käytettävien, mutta helposti saatavilla olevien tietojen säilytysvarastona. Tavallisesti arkistoitu tieto poistetaan sen alkuperäisestä sijainnista, kun se on siirtynyt toissijaiseen muistiin. Varmuuskopioita taas käytetään tietojen suojaamisessa mahdollisten katastrofien varalta. Tiedot säilyvät identtisinä ja niitä ei ole tarkoitus poistaa. Katastrofitilanteissa vahingoittuneet tai tuhoutuneet tiedot voidaan palauttaa varmuuskopioina nopeasti. [Posey & Yu 2018.] Jotkut yritykset käyttävät varmuuskopioita arkistoina. Tämän haittana on se, että varmuuskopiot ovat monesti kopioita koko tietokannasta tai järjestelmästä, jolloin yksittäisten tietueitten hakeminen tulevaisuudessa voi olla haastavaa. [Data Archives and Why You Need Them.]

4.2 Arkistoinnin hyödyt

Tietojen arkistointi on tärkeää, kun käsitellään suurta määrää tietoa. Arkistointi auttaa vähentämään suorituskykyyn liittyviä ongelmia, jotka ovat tietokantojen

koon paisuessa väistämättömiä. Vanha tieto hidastaa kaikkia tietokantatauluihin liittyviä operaatioita, ja useimmiten vanhoja tietoja käytetään vain noin 1 % ajasta. [Performance Best Practices - Data model 2022.]

Arkistoinnin suurin hyöty loppukäyttäjän näkökulmasta on suorituskyvyn paraneminen. Kun päätietokannan tietomäärä vähenee, kyselyjen suorituskyky paranee ja sovellus reagoi nopeammin. Tämä ilmenee yksinkertaisimmillaan web-pohjaisen sovelluksen ikkunan nopeammalla lataamisella. [Data Archiving 2022.]

Tietotekniikan näkökulmasta suurimmat hyödyt ovat sekä suorituskyvyn parantaminen että kustannussäästöt. [Data Archiving 2022.] Kustannussäästöjä syntyy siitä, että arkistoitu tieto voidaan tallentaa nopeiden levyasemien sijasta pienitehoisille ja suurkapasiteettisille kiintolevyasemille tai optisille tallennusvälineille. [Posey & Yu 2018.] Arkistointi voi lisätä myös tietoturvallisuutta. Arkistointi eristää ja poistaa tietoja järjestelmien ulottuvilta, jolloin mahdollisten tietoturvahyökkäysten aiheuttamat haitat vähenevät. [Data Archives and Why You Need Them.]

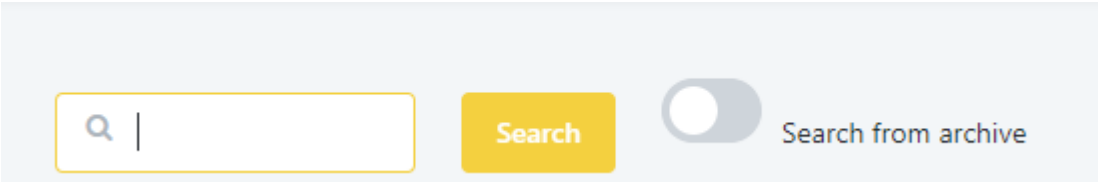
Päätietokannan tietomäärän vähentyessä varmuuskopiointi- ja palautustoiminnot toimivat tehokkaammin ja mahdollisista ääritilanteista, kuten palvelimien tuhoutumisista palautuminen on edullisempaa ja nopeampaa. Tällöin järjestelmän käyttökatkos on myös tavallista lyhyempi. [Data Archiving 2022.]

4.3 Kevyt arkistointi

Kevyt arkistointi on tiedon säilyttämisen muoto, jossa arkistoitua tietoa voidaan hakea ja näyttää yhdessä päätietokannan tietojen kanssa. Tieto voidaan myös palauttaa arkistoidusta tietokannasta päätietokantaan. Kohdistamalla tietokantahaun ensin päätietokantaan nopeuttaa hakua merkittävästi. Lähtökohtaisesti kullekin arkistoitavalle tietokantataululle tehdään sitä vastaava arkistotietokantataulu, johon tieto siirretään arkistoitavaksi. Tällöin pää- ja arkistotietokantataulu

ovat identtisiä attribuuteiltaan, muttei kooltaan, koska tietueiden määrä päätietokantataulussa on pienempi. [Data Archiving 2022.]

Tässä opinnäytetyössä sovelletaan kevyttä arkistointia sovellusprojektissa, jossa rakennetaan yrityksen seulontatietokantataululle arkistotietokantataulu ja kehitetään arkistoinnin toteuttava arkistointilogiikka.



Kuva 6. Kuvankaappaus testiympäristön tietokantahakukentästä. Hakukentän oikealla puolella olevan kytkimen ollessa päällä tietokantahaku tehdään myös arkistotietokantaan.

4.4 Historiallinen arkistointi

Tietokanta voidaan arkistoida myös historiallisesti, mikäli tieto halutaan arkistoida lopullisesti. Tällöin tietokantaan tehdään hakuja harvoin ja sitä säilytetään lähinnä lakisääteisten vaatimusten vuoksi. Historiallisessa arkistoinnissa tietokannalle ei myöskään luoda tietokantahakulogiikkaa eli siihen ei pystytä tekemään SQL-kyselyitä OutSystems-ohjelmien kautta, vaan ainoastaan tietokantojen hallintajärjestelmistä. [Data Archiving 2022.]

Historiallisessa arkistoinnissa tulee huomioida, että tietokantatauluja ei voi arkistoida useampaan eri arkistotauluun, vaan kaikki tallennetaan JSON-muodossa samaan yleiskäyttöiseen keskusarkistoon. Huomioitavaa on myös, että keskusarkistoon mahdollisesti tallennettavan tiedon määrästä voi kasvaa suorituskyvyn pullonkaula jopa silloin, kun siihen kohdistetaan hakuja vain tietokannan hallintajärjestelmästä. [Data Archiving 2022.]

4.5 Arkistointiin liittyviä huomioita

Tietokannan ylläpidossa on huomioitava tietokannan kasvu ja välittömästi saatavilla olevan tiedon määrän tarpeellisuus. Tietokannan yleisen kunnon kannalta on tärkeää, että huomioidaan myös tietokantojen indeksointi, levytilan vapauttaminen ja tietokannan arkistointitarve. [Data Archiving 2022.]

4.5.1 Arkistointistrategian puuttuminen

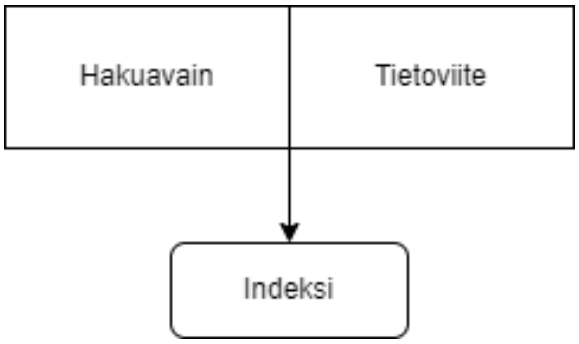
Virheellinen arkistointistrategia tai sen puuttuminen on yleistä. Tämä voi johtaa suorituskyvyn heikkenemiseen. Tietomäärän kasvua tulee ennakoida olemassa olevan tiedon perusteella. Tietokantahakujen suorituskyvyn heikkenemisen estämiseksi tietokannan kasvua tulee seurata säännöllisesti ja harkita arkistointilogiikan suunnittelua. [Data Archiving 2022.]

Arkistoinnin tulisi myös olla täysin automatisoitu prosessi. Kun arkistointilogiikka ja arkistoitujen tietojen poisto päätietokantatauluista automatisoidaan täysin, ei sitä tarvitse erikseen tehdä manuaalisesti esimerkiksi kuukausittain, ja tällöin aikaa vapautuu muihin tehtäviin. [Performance Best Practices - Data model 2022.] Manuaalinen arkistointi on epäkäytännöllistä myös muista syistä. Yhdysvalloissa terveydenhuollon tietojen kasvu on nykyään lähes 40 % vuodessa, jolloin tiedot saattavat kasvaa eksponentiaalisesti ajan myötä. Tällöin manuaalinen arkistointi muuttuu mahdottomaksi prosessiksi. Tulee myös ottaa huomioon, että manuaalinen arkistointi altistaa inhimillisille virheille, kuten väärään arkistoon siirrolle, ennenaikaiselle tyhjennykselle tai arkistoinnin suorittamisen unohtumiselle. Etenkin potilastietojärjestelmissä arkistointiin liittyvillä virheillä voi olla vakavia seurauksia. [6 DOs And DON'Ts: For Data Archiving.]

4.5.2 Arkistotietokannan indeksoinnin puuttuminen

Arkistotietokanta sisältää päätietokantaa enemmän tietoa ja sen tietoja tarvitaan harvemmin. Loppukäyttäjät ovat lähtökohtaisesti tietoisia siitä, että arkistotietokantaan tehtävien kyselyjen tekeminen kestää kauemmin, koska monesti

tietokantaa ei ole optimoitu parhaalla tavalla. Indeksointi on kuitenkin tärkeä arkistotietokantaan tehtävä optimointi. [Data Archiving 2022.] Indeksointi on tapa, jolla tietokannan suorituskykyä voidaan optimoida vähentämällä tietokantahaun aiheuttamien levykäyntien määrä minimiin. [Kaur 2021.]



Kuva 7. Indeksin rakenne tietokannassa.

Indeksi koostuu hakuavaimesta ja tietoviitteestä. Hakuavain sisältää kopion tietokantataulun ensisijaisesta avaimesta. Nämä arvot tallennetaan lajiteltuun järjestykseen, jotta niitä vastaaviin tietoihin saadaan pääsy mahdollisimman nopeasti. Tietoviite sisältää useita osoittimia, joissa on se levylohkon osoite, josta kyseinen avainarvo löytyy. Näin tietokantahaku keskittyy vain tietokantariveille, joihin osoitin näyttää. [Kaur 2021.] Indeksoinnin puute voi hidastaa tietokantaa niin paljon, että tietokantaan ei voi enää tehdä kyselyitä mahdollisten aikakatkosten vuoksi. Tämän takia arkistotietokanta tulisi olla indeksoitu, jotta tiedot voidaan tarvittaessa palauttaa mahdollisimman nopeasti. [Data Archiving 2022.]

4.5.3 Arkistointi ilman tyhjennystä

Tietojen tyhjennys tai puhdistus on arkistoinnin kannalta merkittävä prosessi. Tyhjennysprosessiin kuuluu tiedon poistaminen pysyvästi ensisijaisesta tallennuspaikasta, kuten päätietokannasta. Toissijaista tallennuspaikkaa, johon tieto on siirretty, kutsutaan arkistoksi. Tyhjennys eroaa poistamisesta siten, että tieto on mahdollista palauttaa tarvittaessa arkistotietokannasta. Yritysmaailmassa arkistoinnilla ja tyhjennyksellä tarkoitetaan myös suurien tietomäärien poistamista ja tallennustilan vapauttamista muuhun käyttöön, kun taas pelkkä poistaminen

tarkoittaa lähtökohtaisesti pienien tietomäärien pysyvää poistamista. [Data Purging 2021.]

Mikäli päätietokannasta ei poisteta arkistoituja tietoja, arkistoinnin hyöty on mitätön. Arkistoidut tiedot tulisi poistaa, jotta suorituskyky pysyisi mahdollisimman hyvänä. Sama pätee myös arkistotietokantaan. Tiedot, joita ei tarvita edes arkistotietokannassa, voidaan poistaa esimerkiksi ajastimilla. Tällöin vanhat tiedot poistetaan niiden päivämäärän tai aktiivisuustilan mukaan. [Data Archiving 2022.]

Vanha ja tarpeeton tieto vie palvelimien levytilaa, hidastaa tietokantahakuja ja varmuuskopiointia sekä vaikeuttaa ylläpitoa. Kuten arkistointi, myös tietojen tyhjennys parantaa suorituskykyä ja vähentää kustannuksia. Tietokantojen tyhjennys tulee kuitenkin suunnitella niin, että tarvittavat tiedot säilyvät liiketoiminnan ja lakien velvoittamalla tavalla. [Data Purging 2022.]

5 Sovelluskomponentin kehitys

Arkistointilogiikka tehtiin alustavasti yrityksen potilastietojärjestelmässä olevien seulontatietokannan historiatiedoille. Tämän tietokannan koko on kasvanut niin suureksi, että yritys on kokenut tarpeelliseksi, että siihen suoritettavien tietokantahakujen suorituskykyä optimoidaan. Seulontatietokannalle tehtyä arkistointilogiikkaa voidaan tietyillä muutoksilla käyttää myös muissa yrityksen tietokannoissa ja siten parantaa koko potilastieto- ja toiminnanohjausjärjestelmän (ERP) suorituskykyä ja käyttökokemusta.

5.1 Monivaiheinen ohjelmistokehitys

Yrityksen potilastietojärjestelmän ohjelmistokehitysalustalla on käytössä kolme eri ohjelmointiympäristöä:

- kehittämisympäristö
- laadunvarmistusympäristö

- tuotantoympäristö

Tämä monivaiheinen ohjelmistokehityskehys on monen ohjelmistokehitysyrityksen käytössä, ja se takaa, että ohjelmaan tehdyt muutokset voidaan perusteellisesti testata, ennen kuin ne päivitetään tuotantoympäristöön. [McVey 2017.]

Arkistointilogiikan kehitys ja testaus tapahtuu kehittämisympäristössä, jossa sitä arvioivat ja testaavat myös muut yrityksen sovelluskehityksen henkilökuntaan kuuluvat henkilöt. Kehitysympäristön tietokanta on mallinnettu tuotantoympäristön tietokannasta, mutta se sisältää vain testitietokantataulukoita. Logiikan ollessa valmis se voidaan päivittää laadunvarmistusympäristöön, jossa tietokannat sekä muut sovelluksen toiminnot muistuttavat enemmän tuotantoympäristöä, joka on yrityksen ja sen asiakkaiden todellisessa, jokapäiväisessä käytössä.

5.2 Sovelluskomponentin suunnittelu

Arkistointilogiikan varsinainen suunnittelu aloitettiin keväällä 2022. Sovelluksen suunnittelu alkoi yhdessä sovelluskehitystiimin johtajan kanssa. Suunnitelma perustui sovelluskomponenttiin sisällettävien ominaisuuksien määrittelyyn. Tärkeimmiksi ominaisuuksiksi määriteltiin arkistotietokantatauluun siirtofunktio ja päätietokantataulun tietomäärän vähenemisen toteuttava tyhjennysfunktio. Funktioiden suorituksien tuli olla myös automaattisia. Sovelluksen varsinainen suunnitelma oli aikajana, jossa eri sovelluskomponentin kehitettävät osat olivat järjestyksessä perustuen siihen, missä aikataulussa ne tulisi saada tehtyä.

Yritys on harkinnut arkistointilogiikan luontia jo aikaisemmin, mutta toteutus on jäänyt vasta idean tasolle yrityksen järjestelmän muiden kehityskohtien takia. Tämä mahdollisti sen, että suunnitteluun ja toteutukseen annettiin yrityksen puolesta vapaus luoda omannäköinen sovelluskomponentti huomioiden kuitenkin OutSystems Best Practices-dokumentaation soveltaminen kehityksessä. Suunnittelussa päätettiin, että arkistointilogiikka perustuu OutSystemsin kevyen arkistoinnin dokumentaatioon.

Sovelluksesta ei luotu käyttöliittymän suunnitelmaa, koska tässä arkistointilogiikassa käyttöliittymä on lähtökohtaisesti tarkoitettu vain kehittäjän testaukseen sekä tietokantataulujen visuaaliseen esittämiseen näytöllä. Käyttöliittymä ei myöskään tule loppukäyttäjän näkyville lainkaan.

5.3 Arkistointimoduulin luonti

Yrityksen OutSystemsin kehittämisympäristöön luotiin ensin uusi moduuli arkistointilogiikan kehittämiselle. Vaikka arkistointilogiikka ei tarvitse varsinaista käyttöliittymää muutoksien tapahtuessa vain tietokannassa ja tietotasolla, valittiin silti moduulin pohjaksi reaktiivinen web-sovellus. Reaktiivisen web-sovelluksen pohja sisältää backend-ominaisuuksien lisäksi frontend-käyttöliittymän. Valinta frontendin osalta tehtiin, jotta tietokantamuutoksia voitaisiin havainnoida ja testata käyttäjäystävällisesti käyttöliittymän kautta. Käyttöliittymä on web-sovellus, jolla tehdään tietokantahakuja tietokantapalvelimelle, jossa pää- ja arkistotietokantataulut sijaitsevat. Samalla kyettiin luomaan moduuliin arkistointi- ja tyhjennyslogiikan backend-ominaisuudet. Luomalla riippuvuussuhteita muista moduuleista arkistointimoduuliin arkistointilogiikkaa pystytään hyödyntämään myöhemmin kaikissa potilastietojärjestelmän osissa, missä sitä tarvitaan.

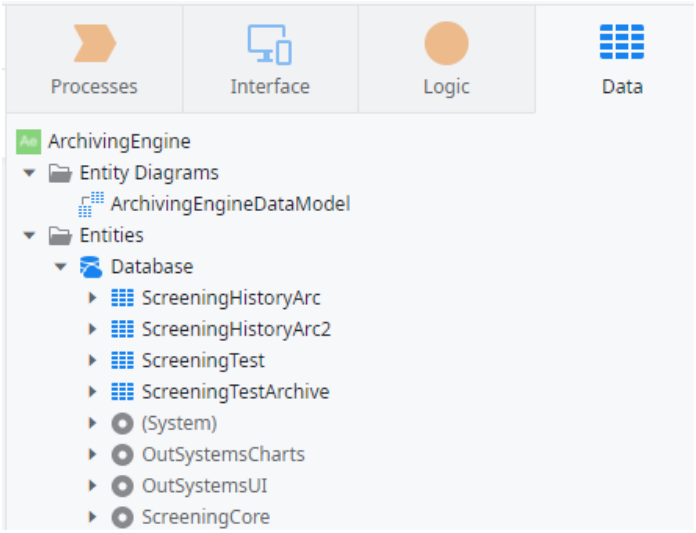
5.4 Tietokannan tuonti arkistointimoduuliin

Yrityksen potilastietojärjestelmä sisältää kymmeniä moduuleita, jotka erillisinä palasina luovat potilastietojärjestelmän kokonaisuuden.

Työssä luotuun arkistointimoduuliin luotiin ensin kaksi tietokantapohjaa. Ensimmäinen päätietokannalle ja toinen arkistotietokannalle. Moduuliin tuotiin järjestelmän kehittämisympäristön seulontamoduulista seulontahistoriatietokanta lisäämällä seulontamoduuli riippuvuussuhteeseen arkistointimoduulin kanssa. Kehittämisympäristön seulontatietokanta ei sisällä todellisia potilastietoja, mutta se on mallinnettu samankaltaiseksi. Seulontatietokanta on relaatiotietokanta, jota hallinnoidaan Microsoft SQL Server Management Studio -hallintajärjestelmällä, joka on liitetty potilastietojärjestelmän palvelimeen. Yhteys

hallintajärjestelmään saadaan luomalla etäyhteys palvelimeen. Luomalla OutSystems-moduuliin uuden tietokannan latautuu se suoraan tietokantapalvelimelle. Kyselyitä siihen voi tehdä etäyhteydellä, OutSystemsin sovelluskehitysympäristöllä tai kehitetyllä ohjelmalla, kuten web-sovelluksella. Tässä opinnäytetyössä käytetään kaikkia edellä mainittuja menetelmiä.

Kehityksessä kehittämisympäristön seulontatietokanta kopioitiin arkisointimoduuliin päätietokannaksi. Tällöin seulontatietokantaan ei tule muutoksia mahdollisten kehityksessä tapahtuvien virheiden takia, jolloin sitä voidaan edelleen käyttää normaalisti potilastietojärjestelmän muiden osien kehityksessä.



Kuva 8. Kuvankaappaus arkistointimoduulin dataosiosta, jossa ScreeningHistoryArc on päätietokantataulu ja ScreeningHistoryArc2 on arkistointitietokantataulu. Muista osioista voi hallinnoida ja kehittää käyttöliittymää, funktioita ja prosesseja, kuten ajastimia.

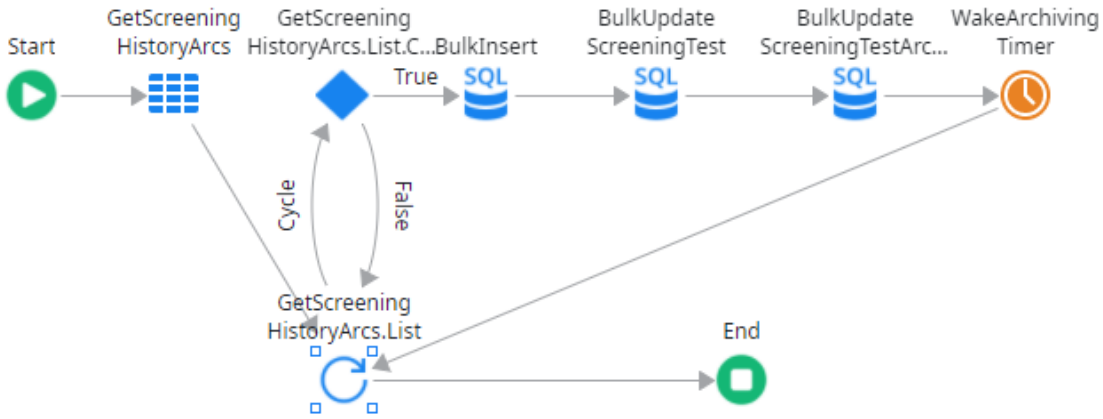
5.5 Arkistointilogiikka

Arkistointilogiikan luominen aloitettiin lisäämällä kehitysympäristöstä kopioituun seulontahistoriatietoja sisältävään päätietokantatauluun ja siitä myöhemmin arkistoitavaan tyhjään arkistotietokantatauluun IsArchived-ohjaussarake, joka sisältää totuusarvomuuttujan perustuen siihen, onko yksittäinen tietue arkistoitu. Kaikki tietokannan uudet ja arkistoimattomat tietueet saavat oletusarvoisesti

sarakkeen arvoksi epätosi. IsArchived-ohjaussarake on merkittävässä roolissa arkistoinnin kannalta yhdessä tietueen aikaleiman kanssa. Sen avulla tunnistetaan arkistoidut tietueet, koska sen arvoa voidaan muuttaa, kun tietue siirtyy taulusta toiseen. Se toimii myös varmistavana tekijänä tilanteiden varalta, jossa vanhaa tietoa poistetaan tietokantataulusta. Mikäli sen arvo on epätosi eli tietuetta ei ole arkistoitu, tietueen poisto voidaan estää.

Pää- ja arkistotietokannassa yksittäiset tietueet sisältävät yksilöllisen numeerisen pääavaimen, jotta yksittäiset rivit voidaan aina erottaa toisistaan, vaikka niiden muu sisältö olisi samankaltaista. Jotta pääavain pysyisi yksittäisen tietueen kohdalla samana tietueita siirtäessä, arkistotietokannan pääavaimelta poistettiin OutSystemsin sille antama IsAutoNumber-ominaisuus. Ominaisuuden tarkoituksena on tavallisesti antaa uudelle tietueelle aina uusi yksilöivä tunnus. Arkistoinnissa on kuitenkin tärkeää, että arkistoitavan tietueen pääavain pysyy samana, vaikka se siirtyisi tietokantataulusta toiseen. Tällöin myös tulevat tietokantahaut palauttavat oikean tietueen, kun pääavain on valmiiksi hakijan tiedossa.

Tämän jälkeen arkistointimoduulin logiikkaosioon luotiin monivaiheinen kuvan 9 mukainen palvelinpuolella suoritettava funktio.



Kuva 9. Palvelinpuolella suoritettava OutSystems-funktio, joka toteuttaa päätietokannan arkistoinnin.

5.5.1 Arkistoinnin siirtovaihe

Funktion ensimmäisen eli aloitusvaiheen jälkeen tulee aggregaatti GetScreeningHistoryArcs, joka suodattaa päätietokannasta ne tietueet, joiden luonnin aikaleiman päivämäärä ylittää ennalta määritetyn aikamääreen, joka tässä tapauksessa on 730 vuorokautta eli kaksi vuotta. Tämän jälkeen silmukka käy läpi kaikki suodatetut tietueet, ja mikäli ne sopivat jos-lauseen määrittämään 730 vuorokauden aikamääreeseen, siirrytään BulkInsert-siirtovaiheeseen, jossa valitaan SQL-kielen kyselyillä kaikki päätietokannasta arkistotietokantaan siirrettävät sarakkeet ja niiden tietueet, jotka sisältävät totuusarvomuuttujan IsArchived arvolla epätosi. WakeArchivingTimer herättää ajastimen vielä kertaalleen tarkastaakseen, onko arkistoitavia tietueita jäljellä.

```
INSERT INTO {ScreeningHistoryArc2} ([Id], [Description], [Timestamp],
[IsArchived])
SELECT
    {ScreeningHistoryArc}.[Id],
    {ScreeningHistoryArc}.[Timestamp],
    {ScreeningHistoryArc}.[Description],
    {ScreeningHistoryArc}.[IsArchived]
FROM
    {ScreeningHistoryArc}
WHERE
{ScreeningHistoryArc}.[IsArchived] = 0
```

Esimerkkikoodi 2. Potilastietoturvasyistä yksinkertaistettu BulkInsert-siirtovaiheen SQL-lausuma. INSERT INTO sisältää käskyn arkistotietokantaan siirrettävät sarakkeista, SELECT valitsee päätietokannan sarakkeet kyselyyn, FROM määrittelee päätietokannan haettavaksi tauluksi ja WHERE-lauseen ehtona on IsArchived-totuusarvomuuttujan epätosi arvo.

5.5.2 Arkistoinnin päivitysvaiheet

Arkistointifunktion (kuva 9) kolmas ja neljäs vaihe sisältävät SQL-tietokannan käsittelyyn sisältyvän UPDATE-kyselyt, joissa päätietokannasta jo kopioidut ja arkistotietokantaan lisätyt taulut saavat IsArchived-sarakkeisiin arvon tosi. Tällöin molemmissa tietokantatauluissa voi olla arkistoitua tietoa, mutta jo arkistoituja tietueita ei arkistoida enää uudelleen. Päätietokannassa olevat arkistoidut tietueet poistetaan, kun niiden aikaleimaan perustuva päivämäärä on ennalta

määritetyn aikamääreen päässä nykyhetkestä. Päätietokannan tyhjennystä käsitellään seuraavassa osioissa.

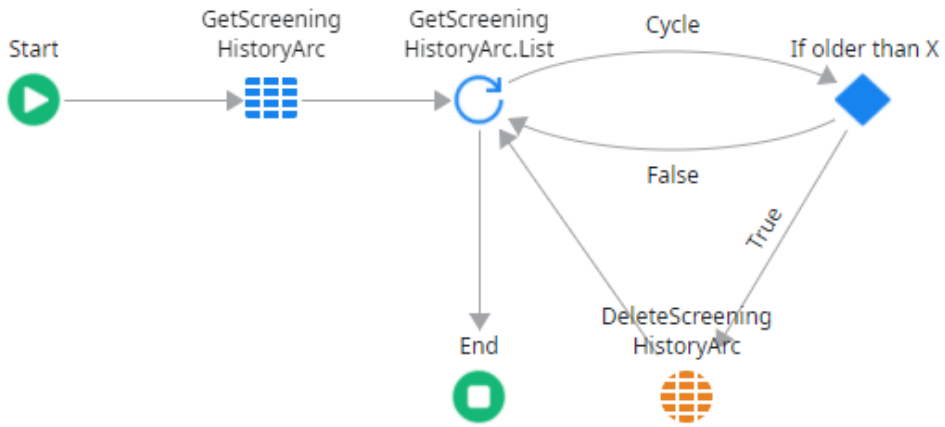
```
UPDATE
    {ScreeningHistoryArc}
SET
    {ScreeningHistoryArc}.[IsArchived] = 1
WHERE
    {ScreeningHistoryArc}.[IsArchived] = 0
```

Esimerkkikoodi 3. Päätietokannan arkistoitujen tietueiden IsArchived-sarakkeen epätosien totuusarvomuuttujien päivitys tosiksi arkistoinnin jälkeen

5.6 Tietojen tyhjennys

Päätietokantatauluun jää arkistoinnin jälkeen tietueita, jotka ovat jo kopioina arkistossa, jolloin ne kuormittavat turhaan päätietokantaa turhaan. Tästä syystä arkisointimoduuliin luotiin poistofunktio, joka tyhjentää arkistoidut tietueet tietokannasta.

Tietue poistetaan sen pääavaimen perusteella, jolloin koko tietueen tiedot häviävät. Kuvan 10 GetScreeningHistoryArc-aggregaatti suodattaa päätietokantataulusta arkistoidut ja 730 vuorokautta vanhat tietueet ja varmistaa, että niiden IsArchived-arvo on tosi eli ne on arkistoitu. Sitä seuraava silmukka käy suodatetun listan läpi ja poistaa ne kokonaan tietokantataulusta käyttäen SQL-kielen DELETE-lausumaa.



Kuva 10. Päätietokannan arkistoitujen tietojen poistofunktio.

5.7 Logiikan suoritus ajastimilla

5.7.1 Ajastin

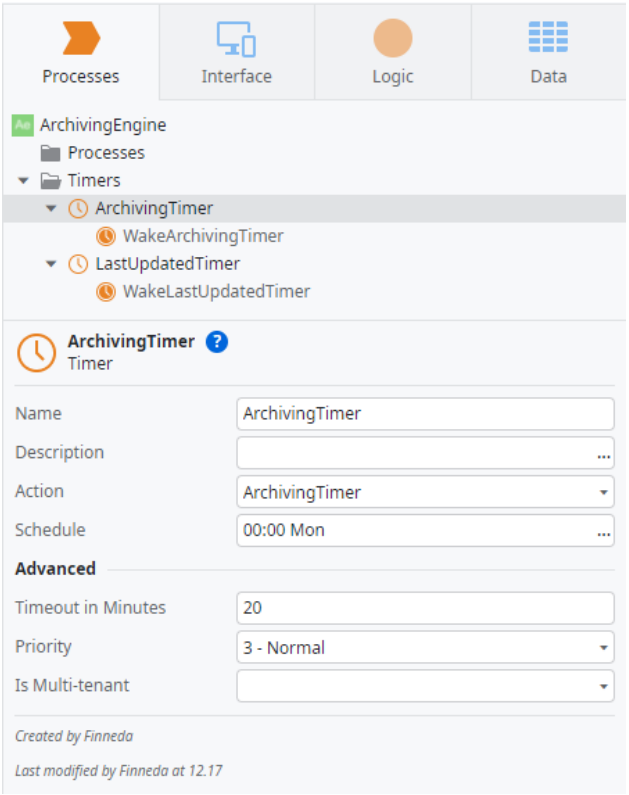
Ajastin on OutSystemsin työkalu, jolla sovelluksen funktioita voidaan suorittaa ajastetusti haluttuna aikana. Yhdessä sovelluksessa voi olla useita ajastimia. OutSystemsin ajastinpalvelu on monisäikeinen, jolloin useampia ajastimia voidaan suorittaa samanaikaisesti, mutta yksi ajastin ei voi suorittaa useaa funktiota samanaikaisesti. Ajastimeen voidaan lisätä myös aikakatkaisu, jolloin ajastimen suorittama funktio pysäytetään. Aikakatkaisu voi tulla tarpeeseen, kun suoritetaan runsaasti resursseja vaativia toimenpiteitä, kuten tietokantojen arkistointia. Arkistointia voidaan ajastaa tapahtuvan esimerkiksi kahden tunnin ajan joka yö, kun sovellusta käytetään vähemmän. Tällöin sovelluksen suorituskyky ei heikenny päiväsaikaan. [Timer 2022.]

5.7.2 Ajastimien luonti

Arkistointilogiikka luotiin ajastimilla yrityksen toiveesta täysin automatisoiduksi komponentiksi. Tällöin arkistointia ja tietojen poistoa ei tarvitse erikseen suorittaa manuaalisesti käyttäjän tai sovelluskehitystiimin toimesta. Logiikan suoritus

voidaan ajoittaa aktiivisten aikojen ulkopuolelle, jolloin arkistointi ei kuormita järjestelmää, kun sitä käytetään.

Arkisointimoduuliin luotiin kaksi ajastinta. Ensimmäinen ajastin suorittaa funktion, jossa päätietokannan sisältö arkistoidaan arkistotietokantaan. Toinen ajastin suorittaa päätietokannan arkistoitujen tietojen tyhjennysfunktion, jossa sen tietueet poistetaan niiden aikaleimaan perustavan päivämäärän sekä IsArchived-totuusarvomuuttujan arvon perusteella.



Kuva 11. Moduulin prosessiosiossa sijaitsevat ajastimet ArchivingTimer ja LastUpdatedTimer. ArchivingTimer -ajastin on ajastettu suorittamaan ArchivingTimer -funktio maanantaisin kello 00:00. Aikakatkaisuksi on määritelty 20 minuuttia.

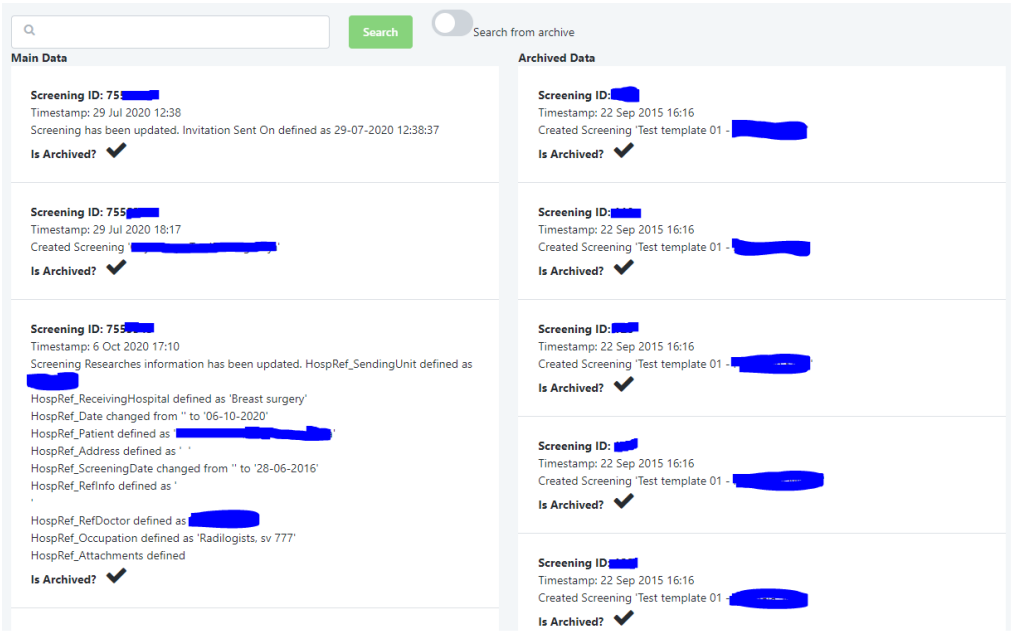
5.8 Testikäyttöliittymä

Arkistointimoduuli ei varsinaisesti tarvitse käyttöliittymää, mutta se rakennettiin, jotta pää- ja arkistotietokanta voidaan esittää selkeyttävästi vierekkäin. Kuten

yrityksen potilastietojärjestelmä, on projektissa luotu testikäyttöliittymä tavallinen web-sivusto.

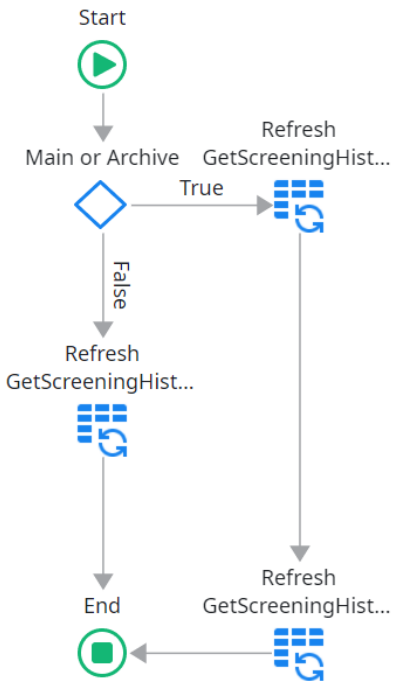
Käyttöliittymä luotiin low-code-ohjelmoinnille tyypilliseen ”vedä ja pudota” -komponenteista rakennettuun näyttöön. Komponenteille voidaan antaa lausekkeita, jotka rakentavat niille sisällön. Esimerkiksi listakomponentille voidaan antaa lausekkeeksi aggregaatti, joka tuo listan sisällöksi tietokantataulun tietueet.

Sivusto tekee kutsun SQL-kyselystä tietokantapalvelimelle käyttäen ennalta määritettyjä aggregaatteja, jotka tässä tapauksessa palauttavat käyttöliittymälle pää- ja arkistointitietokantataulut kokonaisuudessaan. Niiden rivit näkyvät käyttöliittymällä listana. Tämän avulla voitiin alustavasti testata, miten tietokantahaut toimivat tietokantapalvelimen ulkopuolelta tehdyissä SQL-kyselyissä. Kyselyiden kestoja voitiin myöhemmin tarkastella selaimen kehittäjätyökalun verkkopaneelista.



Kuva 12. Web-pohjainen käyttöliittymä, jossa vasemmalla päätietokantataulu ja oikealla arkistotietokantataulu sekä hakukenttä ja haun lähdettä koskeva vaihdin. Mahdolliset yksityisyyttä koskevat tiedot on yliviivattu.

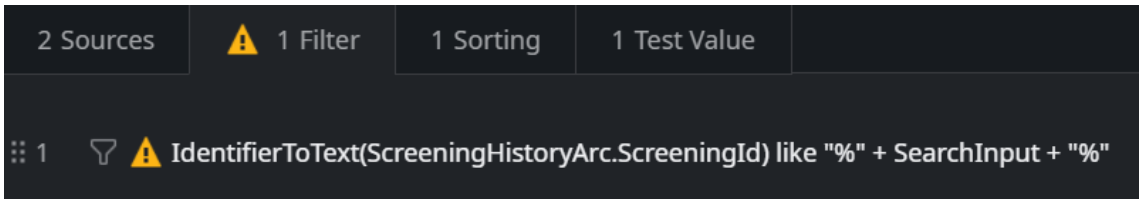
5.8.1 Hakutoiminto



Kuva 13. Käyttöliittymän hakutoiminnon logiikka.

Käyttöliittymän hakutoiminnan logiikka perustuu kuvan 12 hakukentän vieressä olevaan vaihtimen asentoon. Hakukentän Search-painike on yhdistetty käyttöliittymän SearchOnClick-funktioon. Vaihtimen asento määrittää sen, kohdistetaanko tietokantahaku pää- vai arkistotauluun.

SearchOnClick-funktio toteuttaa kuvan 13 mukaiset tehtävät. Funktion toinen vaihe on Main or Archive-muuttujaan perustuva jos-lause. Kun vaihtimen asento on false eli pois päältä, tehdään hakukenttään syötön mukainen haku päätietokantataulun sisältöön. Kun vaihtimen asento on true eli päällä, tehdään haku sen lisäksi myös arkistotietokantatauluun. Haut ovat aggregaattien päivittämiseen perustuvia toimintoja. Aggregaatit tekevät tietokantahaun palvelimelle hakukentässä tallennettavan muuttujan mukaan. Käyttöliittymän esimerkissä hakuja voidaan tehdä tietueen yksilöivän id:n perusteella. Haku on mahdollista jatkokehittää hakemaan tietueen muidenkin attribuuttien perusteella.



Kuva 14. Hakukentän sisällön perusteella suoritettava tietokantahaku.

5.9 Tulokset

5.9.1 Havainnot tietokantapalvelimella

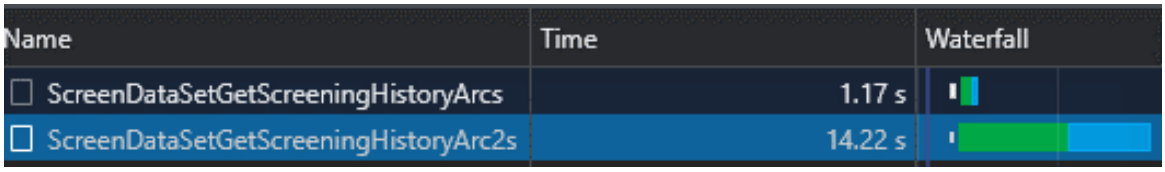
Projektisovelluksen ensimmäisessä versiossa käytettiin päätietokantatauluna kehittämisympäristön seulontahistoriatietokannasta mallinnettua kopiota. Se ei vastaa kooltaan yrityksen tuotantoympäristön potilastietojärjestelmän todellista tietokantataula, mutta sen attribuutit ovat samat. Ennen arkistointia kopioidun päätietokantataulun koko oli noin 210 000 riviä ja suoraan tietokantapalvelimelta tehdyn SELECT * FROM-kyselyn kesto oli noin 5 sekuntia. Edellä mainittu SQL-kysely palauttaa kaikki tietokantataulun sarakkeet tietueineen.

Arkistoinnin jälkeen päätietokantataulun koko supistui noin 9000 riviin. Suoraan tietokantapalvelimelta tehdyn SELECT * FROM-kyselyn kesto lyheni alle yhteen sekuntiin. Tällöin arkistotietokantataulun koko kasvoi noin 200 000 riviin. Siihen tehdyn kyselyn kesto oli noin 5 sekuntia.

5.9.2 Havainnot testikäyttöliittymällä

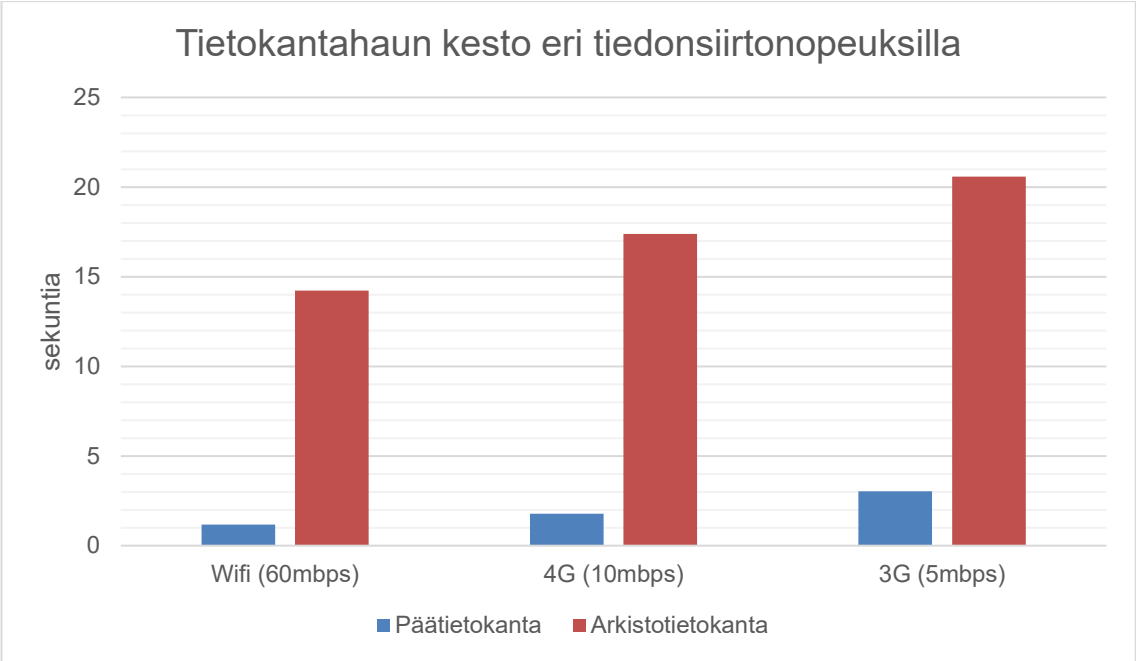
Jotta pää- ja arkistotietokantataulut näkyvät listoina käyttöliittymällä, tekee selain tietokantapalvelimelle kutsun SQL-kyselyistä. Testissä oli tarkoituksena palauttaa käyttöliittymään kahtena listana molempien tietokantataulujen sisällöt. Pelkästään silmin oli havaittavissa, että arkistotietokantalistan latautumisessa meni pidempään kuin päätietokantalistan. Testatessa pyyntöjen kestoa selaimen kehittäjätyökalun verkkopaneelista erot olivat vielä huomattavammat kuin suoraan tietokantapalvelimelta tehdyt haut. Päätietokantatauluun tehdyn

kyselyn vasteaika oli noin 0,9 sekuntia ja tiedon lataamisen kesto noin 0,1 sekuntia, jolloin koko haun kesto oli yhteensä noin yksi sekunti. Arkistotietokantatauluun tehdyn kyselyn vasteaika oli noin 8,2 sekuntia ja tiedon lataamisen kesto noin 5,9 sekuntia, jolloin koko haun kesto oli peräti 14 sekuntia. Käytössä olevan verkon tiedonsiirron latausnopeudeksi mitattiin testaushetkenä 60,1 megabittiä sekunnissa. Arkistoinnin kannalta tämä on merkittävä tiedonsiirron nopeuden parannus. Mikäli arkistotaulua ei olisi, tehtäisiin vain yksi tietokantahaku, joka kohdistuisi päätietokantatauluun. Sen tietueiden määrän ollessa sama, kuin tämän testin arkistotaulussa, voidaan olettaa, että sen latausnopeus olisi myös sama. Tämän testin perusteella arkistoimalla päätietokantataulun tietueita voidaan nopeuttaa siihen kohdistettavien tietokantakyselyiden nopeutta selvästi.



Kuva 15. Kuvankaappaus selaimen kehittäjätyökalusta, joka kertoo tietokantapalvelinhakuihin kuluneen ajan. Päätietokantataulu kuvassa ylempänä, arkistotietokantataulu alempana.

Selaimen kehittäjätyökalulla tiedonsiirtonopeutta voidaan rajoittaa, jolloin kyetään testaamaan web-sivuston latausaikoja simuloiden eri tiedonsiirtonopeuksia (kuva 16). Kaikilla testinopeuksilla tuli ilmi arkistohaun moninkertainen kesto verrattuna suhteellisen nopeaan päähakuun.



Kuva 16. Tietokantahaun kesto eri tiedonsiirtonopeuksilla. Päätietokantataulun suorituskyky on selvästi parempi arkistoinnin jälkeen.

5.10 Jatkokehitysmahdollisuudet

Sovelluskomponentista tullaan jatkokehittämään yrityksen sovelluskehitystiimissä muitakin potilastietojärjestelmän osia arkistoiva moduuli. Insinöörityön raportin kirjoitushetkellä komponentti arkistoi vain yhtä tietokantataulua, mutta luomalla riippuvuussuhteita muihin moduuleihin voidaan niiden tietokannat yhdistää arkisointimoduuliin ja siten saada pääsy niiden tietokantatauluihin. Arkistointimoduulin arkistointi- ja tyhjennysfunktiota voi monistaa useille tietokantatauluille muuttamalla vain niiden suorittamien SQL-kyselyiden sisältöä sen mukaan, millaisia relaatiotietokantatauluja halutaan arkistoida.

Arkistointimoduuliin voi luoda useita ajastimia ja funktioita suorittamaan arkistointia ja tyhjennystä. Eri tietokantatauluihin tehtäviä muutoksia voidaan ajastaa eri ajankohtiin ja niiden prioriteettia ja aikakatkaisua voi muuttaa tarpeen mukaan.

6 Yhteenveto

Insinöörityön tarkoituksena oli luoda eräälle yritykselle arkistointilogiikan toteuttava sovelluskomponentti low-code-sovelluskehitystyökalulla. Arkistointikomponentin tarkoituksena oli nopeuttaa tietokantahakujen suorituskykyä vähentämällä käytössä olevien relaatiotietokantataulujen tiedon määrää arkistoimalla tietueita, joita ei enää tarvita.

Arkistointikomponentti luotiin OutSystems-low-code-sovelluskehitystyökalulla, jolloin sovelluskehitys voitiin toteuttaa nopeammin verrattuna perinteiseen ohjelmointiin. Sovelluskomponentin liittäminen yrityksen potilastietojärjestelmän kehitysalustalle pystyttiin myös tehdä perinteistä helpommaksi. Kevyen arkistoinnin logiikan luomisessa pystyttiin hyödyntämään OutSystemsin tarjoamia Best Practices-käytäntöjä, jolloin monilta odottamattomilta ongelmilta pystyttiin välttymään.

Lopullinen sovellus oli tehokas ja täysin automatisoitu arkistointikomponentti. Arkistoinnin jälkeen päätietokantatauluun suoritettujen kyselyiden suorituskyky parani merkittävästi.

Komponentin arkistointifunktioita voidaan monistaa toteuttamaan arkistointia usealle eri tietokantataululle. Sitä voidaan siis tulevaisuudessa hyödyntää järjestelmän muissa osissa, jolloin koko järjestelmän käyttökokemus ja suorituskyky oletettavasti kasvavat merkittävästi.

Lähteet

6 DOs And DON'Ts: For Data Archiving. Verkkoaineisto. Iron Mountain. <https://www.ironmountain.com/resources/whitepapers/d/6-dos-and-donts-for-data-archiving>. Luettu 4.8.2022.

Aggregate. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/11/Reference/OutSystems_Language/Data/Handling_Data/Queries/Aggregate>. Päivitetty 29.6.2022. Luettu 22.7.2022.

Airas, Matti. 2021. Miten valita toimialaan ja toimintamalliin sopiva low-code-alusta? Verkkoaineisto. Tietoevry. <https://www.tietoevry.com/fi/blogi/2021/12/miten-valita-toimialaan-ja-toimintamalliin-sopiva-low-code-alusta/>. 3.12.2021. Luettu 21.6.2022.

Arh, Mia. 2021. Four Challenges Low-Code Platforms Face. Verkkoaineisto. Planet Crust. <https://www.planetcrust.com/overcoming-the-limitations-and-challenges?utm_campaign=blog>. 6.11.2021. Luettu 29.6.2022.

Data Archives and Why You Need Them. Verkkoaineisto. Cloudian. <https://cloudian.com/guides/data-backup/data-archive/>. Luettu 4.8.2022.

Data Archiving. 2022. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/Best_Practices/Architecture/Data_Archiving>. Päivitetty 6.5.2022. Luettu 29.6.2022.

Data Purging. 2021. Verkkoaineisto. Techopedia. <https://www.techopedia.com/definition/28042/data-purging>. Päivitetty 8.11.2021. Luettu 28.7.2022.

Data Purging. 2022. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/Best_Practices/Architecture/Data_Purging>. Päivitetty 6.5.2022. Luettu 29.6.2022.

Kaur, Avneet. 2021. Indexing in Databases. Verkkoaineisto. GeeksForGeeks. <https://www.geeksforgeeks.org/indexing-in-databases-set-1/>. Päivitetty 15.9.2021. Luettu 29.6.2022.

Kotilainen, Samuli. 2018. Nyt loppuu ohjelmointi? Verkkoaineisto. Tivi. <https://www.tivi.fi/uutiset/nyt-loppuu-ohjelmointi/c5548184-d96b-3272-9fa5-b246d6b83b58>. Päivitetty 8.6.2020. Luettu 21.6.2022.

Low-Code. 2022. Verkkoaineisto. IBM. <https://www.ibm.com/cloud/learn/low-code>. 22.4.2021. Luettu 4.8.2022.

Low code – mistä kyse? 2020. Verkkoaineisto. Festum Software. <https://software.festum.fi/blogi/low-code-mista-kyse/>. 5.6.2020. Luettu 21.6.2022.

Low-Code Development Platform Market. 2020. Verkkoaineisto. MarketsandMarkets Research. <https://www.marketsandmarkets.com/Market-Reports/low-code-development-platforms-market-103455110.html>. Luettu 14.6.2022.

Luukkainen, Matti & Vihavainen, Arto. 2017. Tietokantojen perusteet. Verkkoaineisto. <https://tietokantojen-perusteet.github.io/>. Luettu 22.7.2022.

Mattila, Merja. 2019. Varjo-IT on todellinen tietoturvariski – tiedätkö, mitä kaikkia järjestelmiä teidän asiantuntijanne käyttävät työssään? Verkkoaineisto. Sulava. <https://sulava.com/tietoturva/varjo-it-on-todellinen-tietoturvariski-tiedatko-mita-kaikkia-jarjestelmia-teidan-asiantuntijanne-kayttavat-tyossaan/>. 26.11.2019. Luettu 29.6.2022.

McVey, Cathy. 2017. Dev, Test, Prod: Oh, My! Verkkoaineisto. Miami University. <https://miamioh.edu/it-services/news/2017/08/dev-test-prod.html>. 10.8.2017. Luettu 20.7.2022.

Modern Web Apps with ReactJS and OutSystems. Verkkoaineisto. OutSystems. <https://www.outsystems.com/webinars/modern-web-apps-reactjs/>. Luettu 29.6.2022.

Performance Best Practices - Data model. 2022. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/Best_Practices/Performance_and_monitoring/Performance_Best_Practices_-_Data_model/#archive-old-data-in-separate-entities>. Päivitetty 6.5.2022. Luettu 14.6.2022.

Potilasasiakirjojen säilyttäminen. Verkkoaineisto. Valvira. <https://www.valvira.fi/terveydenhuolto/hyva-ammatinharjoittaminen/potilasasiakirjat/potilasasiakirjojen_sailyttaminen>. Päivitetty 26.5.2020. Luettu 21.6.2022.

Posey, Brian & Yu, Johnny. 2018. Data archiving. Verkkoaineisto. TechTarget. <https://www.techtarget.com/searchdatabackup/definition/data-archiving>. Päivitetty 1.11.2018. Luettu 27.7.2022.

Pratt, Mary. 2021. Low-code and no-code development platforms. Verkkoaineisto. TechTarget. <https://www.techtarget.com/searchsoftwarequality/definition/low-code-no-code-development-platform>. Päivitetty 1.3.2021. Luettu 21.6.2022.

Pros and Cons of Low Code Development. Verkkoaineisto. Alpha Software. <https://www.alphasoftware.com/pros-and-cons-of-low-code-development>. Luettu 29.6.2022.

Solita ja OutSystems low-code-yhteistyöhön Suomessa ja Ruotsissa. Verkkoaineisto. Solita. <https://www.solita.fi/solita-ja-outsystems-low-code-yhteistyoehoen-suomessa-ja-ruotsissa/>. 18.9.2019. Luettu 29.6.2022.

The Low-Code Development Guide. Verkkoaineisto. OutSystems. <https://www.outsystems.com/guide/low-code/>. Luettu 14.6.2022.

Timer. 2022. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/11/Reference/OutSystems_Language/Processes/Timer>. Päivitetty 29.6.2022. Luettu 20.7.2022.

Use Services to Expose Functionality. 2022. Verkkoaineisto. OutSystems. <https://success.outsystems.com/Documentation/11/Developing_an_Application/Reuse_and_Refactor/Use_Services_to_Expose_Functionality>. Päivitetty 29.1.2022. Luettu 4.8.2022.

What is a relational database? Verkkoaineisto. Google Cloud. <https://cloud.google.com/learn/what-is-a-relational-database>. Luettu 3.8.2022.

What is a Relational Database (RDBMS)? Verkkoaineisto. Oracle. <https://www.oracle.com/database/what-is-a-relational-database/>. Luettu 3.8.2022.