jamk.fi

Tietokantojen muutosten hallinta

Case: Landis+Gyr

Sami Salovaara

Opinnäytetyö
Joulukuu 2016
Luonnontieteiden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Salovaara, Sami	Opinnäytetyö, AMK	Joulukuu 2016
	Sivumäärä 42	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Tietokantojen muutosten hallinta
Case: Landis+Gyr Oy

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
Jarkko Immonen

Toimeksiantaja(t)
Landis+Gyr Oy

Tiivistelmä
Tehtävänä oli tutkia ja valita paras tietokannan muutosten hallinnan ratkaisu Landis+Gyrin luomaan valvontapalveluun. Tietokantojen muutokset tehtiin ennen käsin, mutta kolmannen tietokannan lisäyksen myötä muutosten hallintaan tarvittiin kestävämpi ratkaisu, joka mahdollistaa tietokantojen versioinnin ja rakenteiden vertailun.

Tutkimus toteutettiin kehittämistutkimuksena, jossa vertailtiin viittä esitutkimuksen aikana valittua tietokantojen versiointiratkaisua. Vertailtavista sovelluksista valittiin ulkoisten ominaisuuksien ja resurssien perusteella kaksi valvontapalvelulle soveltuvinta tarkempaan tutkimukseen, jossa valitut sovellukset asennettiin omiin virtuaaliympäristöihinsä käytännön vertailua varten. Vertailtavista kategorioista annettiin pisteet valvontapalvelun näkökulmasta. Enemmän yhteispisteitä saanut sovellus asennettiin valvontapalveluun.

Valvontapalvelulle sopivimmaksi vaihtoehdoksi valikoitui Liquibase. Se asennettiin ja konfiguroitiin onnistuneesti valvontapalvelun kehitys-, testaus- ja tuotantokantaan. Muutosten hallintaa Liquibasen ja vanhan manuaalisen muutosten hallinnan välillä vertailtiin kolmella yleisimpiä käyttötapauksia kuvaavalla testillä, joiden avulla selvitettiin Liquibasen nopeutta käsin tehtyihin muutoksiin verrattuna. Tuloksista huomattiin, että Liquibase oli ylivertaisen nopea kahdessa kolmesta testitapauksesta. Ainoastaan taulujen luonti oli nopeampaa SQL-kyselyillä sekä SQL Server Management Studion avulla.

Lopputuloksena tuotettiin toimiva tietokantojen muutosten hallinta valvontapalvelulle, mikä tukee laajentumista automatisaation ja jatkuvan kehityksen suuntaan. Tietokantojen muutosten hallinnan ansiosta muutosten tekoon kuluva aika sekä manuaalisesta työstä johtuvat inhimilliset virheet ovat vähentyneet. Myös tietokantojen rakenteiden vertailu on nopeampaa ja selkeämpää kuin ennen.

Avainsanat (asiasanat)
Relaatiotietokanta, kehittämistutkimus, versionhallinta, muutosten hallinta, tietokantaskeema

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Salovaara, Sami	Bachelor's thesis	December 2016
		Language of publication: Finnish
	Number of pages 42	Permission for web publication: x

Title of publication
Database change management
Case: Landis+Gyr

Degree programme
Business Information Systems

Supervisor(s)
Immonen, Jarkko

Assigned by
Landis+Gyr

Abstract

The objective of this thesis was to examine and compare multiple database version control products and install one for Landis+Gyr's Active Monitoring Service. Previously the schema changes to databases have been done manually; however, with the introduction of a third database a more robust solution was needed.

The study was carried out using design research methods to compare five database version control systems that were chosen during a feasibility study. Two out of the five were chosen for a more detailed inspection on the basis of their documentation, size of their developer community and by the activity of product's main developer or developers. Those chosen for next phase of the research were installed to their own virtual environments where more in-depth features and functionalities were tested.

The most suitable candidate for Active Monitoring System was Liquibase. It was installed and configured successfully for Active Monitoring System's development, testing and production databases. The database change control using Liquibase was compared to a manually done change management through three different tests. The goal of these tests was to find out how much quicker the most common use cases are carried out by using Liquibase than by using SQL scripts or SQL Server Management Studio manually.

Liquibase excelled in every test and was superior to manually done change management in almost every measured way. Only the table creation was done faster using SQL queries or SQL Server Management Studio. Implementing a database version control to Active Monitoring Service has helped the project's development effort with less time consuming database structure management and by reducing human errors when creating schema changes.

Keywords/tags (subjects)
Relational database, design research, version control, change management, database schema

Miscellaneous

Sisältö

Kuviot

Taulukot

1 Johdanto

Tietokannat ovat merkittävässä roolissa yhteiskunnan sekä yritysten toiminnan takaamisessa. Ilman pysyvän tiedon tehokasta käsittelyä ja tallennusmahdollisuutta useiden maailman suosituimpien sovellusten ja sivustojen toiminta ei olisi mahdollista. Myös nyky-yhteiskunta nojaa vahvasti tiedon jatkuvaan saatavuuteen. Potilastiedot, poliisien tietojärjestelmä, sääpalvelut, pörssi sekä pankkien tietokannat ovat vain muutamia esimerkkejä jokapäiväisestä tarpeesta tiedon hallintaan. Edellä mainittujen järjestelmien toiminta on täysin riippuvaista hyvin suunnitelluista tietokannoista. Yritysten näkökulmasta asiakas- sekä laskutustietojen turvallinen tallentaminen sekä varmuuskopiointi on välttämätöntä sujuvan liiketoiminnan kannalta.

Ohjelmistotuotannossa sovellusten lähdekoodin versionhallinta on arkipäivää. Valitettavasti tietokantojen versiointi ei ole yhtä yleistä. Tietokantojen muutosten hallintaa laiminlyödään usein projekteissa. Versiointia aletaan suunnitella vasta sitten, kun vahinko on jo ehtinyt tapahtua. Oikein käytettynä tietokantojen versiointi tehostaa kehitystyötä ja vähentää tietokantojen muutoksista johtuvia palvelukatkoja sekä muita ongelmia. Isoissa kehitystiimeissä yksittäisten kehittäjien muutokset tietokantoihin menevät helposti ristiin, jolloin seurauksena voi olla buginen ja toimimaton tietokanta. Varsinkin relaatiotietokantojen muutosten hallintaan on luotu runsaasti erilaisia ratkaisuja valmiista skripteistä laajoihin kaupallisiin ohjelmistoihin.

Relaatiotietokantojen muutosten hallinta on aiheena ajankohtainen, sillä NoSQL-tietokantojen suosion noususta huolimatta kolme selvästi suurinta tietokannan hallintajärjestelmää ovat relaatiotietokantapohjaisia ratkaisuja (DB-Engines Ranking 2016). Useat eri tietokantaversiot ja -ympäristöt ovat arkipäivää nykyaikaisessa sovelluskehityksessä, jolloin muutosten hallinta on ensiarvoisen tärkeää kriittisten virheiden välttämiseksi sekä palveluiden käytettävyysajan maksimoimiseksi.

Landis+Gyrin valvontapalvelun laajentumisen vuoksi manuaalinen tietokantojen hallinta ei ole enää järkevää. Markkinoilla on useita ilmaisen yrityslisenssin ratkaisuja, joilla tietokantojen hallintaa saadaan suoraviivaistettua, mikä vähentää samalla kehittäjän työmäärää ja inhimillisiä virheitä muutosten hallinnassa. Tutkimuksessa vertaillaan kahta ratkaisua projektin vaatimusten näkökulmasta. Tutkituista ratkaisuista projektille sopivin asennetaan ja konfiguroidaan projektin yhteyteen.

2 Tutkimusasetelma

2.1 Toimeksiantaja

Landis+Gyr Oy on sähkön etämittaukseen etäluentalaitteita ja -sovelluksia valmistava monikansallinen yritys, jonka yksi tutkimus ja kehityskeskuksista sijaitsee Jyväskylän Jyskässä. Jyväskylässä on valmistettu etäluettavia sähkömittareita ja niitä tukevia järjestelmiä jo 1980-luvulta lähtien. Jyväskylän toimipiste työllistää noin 200 henkilöä. (Luotola 2016.)

Valittu ratkaisu toteutetaan etäluentainfrastruktuurin tilan jatkuvaan seurantaan luodun valvontapalvelun yhteyteen. Valvontapalvelun sovellus asennetaan asiakkaan järjestelmään, josta se lähettää tilannetietoja ympärivuorokautisesti keskuspalvelimelle, joka käsittelee ne ja ilmoittaa asiakkaalle, jos arvot poikkeavat normaaleista. Palvelun avulla mahdolliset häiriöt ja ongelmat voidaan ennaltaehkäistä vaivattomasti. Palvelu tarjoaa asiantuntijatukea järjestelmän päivittäiseen seurantaan. (Järjestelmän valvontapalvelu Gridstream-ratkaisulle 2015.)

2.2 Opinnäytetyön asetelma ja rajaus

Idea tietokantojen muutosten hallinnan parantamiseksi kehittyi, kun projektiin lisättiin testikanta kehitys- ja tuotantokantojen rinnalle. Tähän asti muutokset kehityksestä tuotantoon on tehty käsityönä, mikä on hidasta ja virhealtista toimintaa. Kolmannen tietokannan eli laadunvalvontakannan luonnin jälkeen manuaalisen työn määrä kasvoi huomattavasti, sillä muutokset kehityksestä testikantaan ja testikannasta joko takaisin kehityskantaan tai tuotantoon pitäisi tehdä edelleen käsin. Tavoitteena on siis kehittää ratkaisu, jonka avulla tietokantojen muutoksia on helpompi ja nopeampi hallita, ilman käsityöllä tehtävien muutosten aiheuttamia inhimillisiä virheitä. Tätä varten on luotu useita sovelluksia, joita vertaillaan tässä opinnäytetyössä ja joista valitaan valvontapalvelun kannalta paras. Valittu sovellus asennetaan ja konfiguroidaan projektin yhteyteen.

Opinnäytetyön empiirinen osuus rajataan ainoastaan tietokantojen muutosten hallintaan. Ajatuksena oli myös automatisoida palvelinten välinen sekä asiakasohjelmien

välinen konfiguraatio, mutta tiukka rajaus tietokantapuoleen palvelee opinnäytetyötä paremmin, mikä mahdollistaa tiiviimmän tutkimuksen sekä syvällisemmät tulokset ja johtopäätökset.

Tutkimusta varten luotavien testiympäristöjen luominen täytyy suoraviivaistaa, sillä kaikki kolme ympäristöä pyörivät eri palvelimilla, joissa kaikissa on eri käyttöjärjestelmä. Kehitysympäristön käyttöjärjestelmä on Windows 7 Enterprise, testausympäristön Windows Server 2012 ja tuotantoympäristön Windows Server 2008 R2. Täten opinnäytetyössä vertailtavat ratkaisut rajataan koskemaan vain sovelluksia, joissa on tuki kaikille edellä mainituille käyttöjärjestelmille.

2.3 Tutkimusmenetelmä

Opinnäytetyö toteutetaan kehittämistutkimuksena. Kanasen (2015, 11) mukaan kehittämistutkimuksen tarkoituksena on pyrkiä muutokseen, joka tarkoittaa käytännön ongelman poistamista tai parannettua olotilaa (Kananen 2015). Kehittämistutkimukseen on aina liitettävä myös tutkimus kehittämistyön rinnalle (mts. 33). Nämä seikat pätevät myös tähän tutkimukseen, sillä lopputulokseksi pyritään saamaan nopeampi ja helpompi tietokantojen muutosprosessi tutkimuksessa valitun ratkaisun avulla. Tutkimusmetodina käytetään määrällistä menetelmää, sillä tutkimuskohteina ovat tietokantojen muutosten hallintaan kehitetyt sovellukset, joita voidaan vertailla keskenään ja mitata tutkimusta varten laaditun kriteeristön mukaan. Kriteeristö esitellään omassa alaluvussaan.

Tutkimus etenee vaiheittain. Aluksi kartoitetaan tutkimuksessa vertailtavat tietokannan muutosten hallintaratkaisut, jotka täyttävät alla listatut vaatimukset.

- SQL Server 2008 -tuki
- Windows Server 2012 -tuki, mikäli ratkaisu on keskitetty
- Windows 7 -tuki, mikäli ratkaisu on hajautettu
- Ilmainen lisenssi yrityskäyttöön.

Vaatimukset on määritelty tietokantojen versionhallintaratkaisujen ja valvontapalvelun yhteensopivuuden takaamiseksi. Vaatimusten perusteella tutkimukseen otetaan

mukaan viisi tietokannan versionhallintaratkaisua. Ensimmäisessä vaiheessa kaikki valitut sovellukset käydään pääpiirteittäin läpi, samalla kerrotaan niiden toimintaperiaatteesta ja listataan niiden merkittävimmät edut ja epäkohdat. Viidestä sovelluksesta valitaan kaksi valvontapalveluun soveltuvinta dokumentaation määrän ja laadun, käytettävyyden ja kehitysyhteisön koon sekä aktiivisuuden perusteella. Näihin valittuihin sovelluksiin perehdytään syvällisemmin, ne asennetaan omille virtuaalipalvelimilleen ja vertaillaan toisen vaiheen kriteerien mukaan. Tutkimuksen perusteella vertailussa enemmän pisteitä saanut sovellus asennetaan valvontapalveluun.

Kahta tarkemmin tutkittavaa sovellusta varten pystytetään kummallekin omat virtuaalipalvelimet. Käyttöjärjestelmät valitaan vertailtavien sovellusten mukaan. Mikäli kyseessä on keskitetty ratkaisu, asennetaan sovellus Windows Server 2012 -ympäristöön. Hajautettu ratkaisu asennetaan puolestaan Windows 7 Professional -ympäristöön. Palvelimille asetetaan kaksi eri versiota tätä tutkimusta varten luodusta SQL Server -testauskannasta, jotta muutosten hallintaa voidaan vertailla ja mitata tarkasti.

2.4 Tutkimusongelma ja -kysymykset

Tutkimusongelmana on Landis+Gyrin valvontapalvelun tietokantojen muutosten hallinnan tehostaminen. Se on johdettu valvontapalvelun kehittämisen tarpeesta hallita tehokkaammin ja varmemmin muutoksia kehitys-, testaus- sekä tuotantotietokannassa. Tutkimusongelman ratkaisemista varten on laadittu kolme tutkimuskysymystä, joihin vastaamalla valvontapalvelun muutosten hallintaa koitetaan parantaa. Kysymykset on listattu alla.

- Kuinka relaatiotietokantojen muutosten hallintaa voidaan parantaa?
- Mikä on paras ratkaisu valvontapalvelun tietokantojen muutosten hallintaan?
- Miten valittu ratkaisu toteutetaan projektiin?

Ensimmäiseen kysymykseen vastataan tietoperustalla, jossa kerrotaan relaatiotietokannoista ja versionhallinnasta yleisesti sekä relaatiotietokantojen muutosten hallinnasta. Toisen kysymyksen mittaus hoidetaan pisteytyksellä, jossa jokaisesta kriteeristä annetaan pisteet yhdestä kolmeen. Eniten pisteitä saanut ratkaisu valitaan ja to-

teutetaan projektiin. Kriteeristö on listattu omassa luvussaan. Kolmas kysymys vastaa ratkaisun täytäntöönpanosta ja valitun ratkaisun vertailusta vanhaa manuaalista muutosten hallintaa vastaan. On otettava huomioon mahdolliset asennuksen yhteydessä esiintyvät komplikaatiot ja asennuksesta kirjoitettava dokumentointi. Lopputuloksena on toimiva sovellus valvontapalvelun tietokantojen muutosten hallintaan.

2.5 Kriteeristö

Tutkimuskysymysten mittaus hoidetaan pisteytyksellä, jossa yhteenlasketuista pisteistä paras valitaan. Ensimmäisen vaiheen kriteereiksi on valittu sovelluksen ulkopuolisia tekijöitä ja resursseja, jotka eivät vaadi sovelluksen asentamista. Kriteerit keskittyvät mahdollisten sovellusten käytössä ilmaantuvien ongelmatapausten ratkaisemiseen ja kehittäjäyhteisöltä saatavilla olevaan tukeen näissä tapauksissa.

Ensimmäisessä vaiheessa tutkitaan kaikkien viiden sovelluksen:

- dokumentoinnin määrää ja laatua
- kehittäjäyhteisön kokoa
- kehittäjän aktiivisuutta ja päivitysten julkaisutiheyttä.

Ensimmäisessä vaiheessa kaksi eniten pisteitä saanutta sovellusta siirtyy toiseen vaiheeseen tarkempaan tutkimukseen. Toisessa vaiheessa vertailtavat seikat on valittu kattamaan valvontapalvelun kannalta tärkeimmät kriteerit, kuten asennukseen, muutosten hallintaan, käytettävyyteen, konfiguraatioon ja ylläpitoon keskittyvät seikat. Painotus on muutosten hallinnan nopeudessa sekä usean tietokannan synkronoinnissa. Näistä kahdesta kategoriasta saa kaksinkertaiset pisteet.

Toisessa vaiheessa tutkitaan:

- asennusprosessin nopeutta
- konfiguroitavuutta
- muutosten hallinnan nopeutta ja vaivattomuutta
- usean tietokannan synkronointia
- ylläpitoa asennuksen jälkeen.

3 Relaatiotietokannat ja versionhallinta nykyaikaisessa ohjelmistokehityksessä

3.1 Relaatiotietokannat

Tietokannat voidaan määritellä karkeasti suuriksi ja keskitetyiksi datasäilöiksi. Täysiveristen tietokantojen tulee säilyttää tietonsa eheänä sekä sisältää metatietoa omasta toiminnastaan. Metatieto tarkoittaa kuvauksia tietokannan omasta rakenteesta sekä tietokannan sisältämästä datasta. Tietokannat eivät kuitenkaan kykene toimimaan itsenäisesti, vaan ne tarvitsevat tietokannan hallintajärjestelmän eli DBMS:n apua käsittelemään dataa. Tietokantoja käyttävät sovellukset eivät voi itsestään hakea tietoa tai tehdä muutoksia tietokantojen rakenteisiin, vaan kaikki tietokantaan liittyvä toiminta kulkee DBMS:n kautta. (Ritchie 2005, 5–6.) DBMS hoitaa tiedon tallennuksen, ylläpidon sekä noutamisen. DBMS:n tärkeimpinä tehtävinä on datan saatavuuden, eheyden, turvallisuuden ja itsenäisyyden takaaminen. Relaatiotietokantojen hallintajärjestelmiä kutsutaan lyhenteellä RDBMS. (Sumathi & Esakkirajan 2007, 3–4.)

Markkinoilla on useita RDBMS -vaihtoehtoja. Suosituin kaikista on Oracle Database, jota seuraavat MySQL sekä SQL Server. (DB-Engines Ranking 2016.) Relaatiotietokannoissa kaikki data tallennetaan tauluihin, jotka koostuvat sarakkeista. Jokaisella sarakkeella on tietty tietotyyppi, kuten kokonaisluku tai merkkijono. Tiedot tallennetaan riveinä, jossa jokainen rivi on oma instanssinsa. RDBMS:lle ominaista on avainten käyttö rivien identifiointiin sekä suhteiden määrittelyyn. Tauluille voidaan tehdä indeksejä nopeuttamaan hakuja sarakkeiden perusteella. (Ambler n.d.) Indeksit ovat hyödyllisiä varsinkin tauluissa, jotka sisältävät satoja tuhansia rivejä dataa.

Ennen tietokantajärjestelmien yleistymistä suurin tietojen tallentaminen hoidettiin käyttöjärjestelmien mukana tulleiden tiedostojärjestelmien avulla. Tiedostojärjestelmät koostuvat erillisistä ohjelmista, jotka suorittavat käyttäjän määrittelemiä tehtäviä. Tiedostojärjestelmät kuitenkin luovat runsaasti saman tiedon duplikaatteja, jolloin kuluu paljon ylimääräistä levytilaa. Ne myös ylläpitävät erinäisiä riippuvuuksia, joissa ohjelma tarvitsee tiettyjä tiedostoja toimiakseen. Mikäli tiedostot olivat muut-

tuneet, saattoi ohjelma lakata toimimasta. Nykyään tietokannat ovat korvanneet tehottomat tiedostojärjestelmät. (Sumathi & Esakkirajan 2007, 6–8.)

Relaatiotietokantojen käyttöä varten on luotu kyselykieliä, joista selvästi suosituin on SQL (engl. Structured Query Language). Eri tietokantajärjestelmillä on omia versioita kielestä, mutta peruskomennot ovat samat kaikkien RDBMS:ien kesken. Komennot voidaan jakaa kategorioihin, joista kaksi tärkeintä ovat Data Manipulation Language (DML) sekä Data Definition Language (DDL). DML käsittelee dataa ja DDL tietokannan rakennetta. (A Relational Database Overview n.d.)

Relaatiotietokannat ohjelmistokehityksessä

Nykyaikaisessa oliopohjaisessa ohjelmistokehityksessä tiedon käsittely toteutetaan usein ORM-ratkaisulla (engl. Object Relational Mapping). ORM abstraktoi tietokantojen taulut ja suhteet, jolloin niitä voidaan kutsua suoraan ohjelmakoodista. (Ambler 2013.) Tämä yksinkertaistaa ja suoraviivaistaa ohjelmalogiikkaa lataamalla tietokantaa imitoivan tietorakenteen muistista, samalla abstraktoiden logiikan oliopohjaiseksi, jolloin poistuu myös tarve suorille SQL-kyselyille. Poikkeuksetta kaikki suosituimmat olio-ohjelmointikielet, kuten Java, C#, Ruby ja PHP, tukevat ORM-tekniikkaa joko kielen kehittäjän tarjoamina rajapintoina tai kolmannen osapuolen luomina viitekehyksinä ja kirjastoina. Jotkut ORM-kirjastot hoitavat tietokantojen migraatiota ja osittain ajavat tietokantojen versionhallintaa jossain määrin. (Nimkar 2016b.)

Abstraktoidessa tietokannan tauluista tehdään omat luokkansa, ja luokan attribuuteiksi asetetaan taulun sarakkeet. Luokkaan voidaan luoda myös attribuutteja, jotka eivät ole yhteydessä tietokantaan, vaan ovat väliaikaisia, vaikkapa sovelluksen käyttämiä laskumuuttujia. Abstraktointi vähentää manuaalisesti kirjoitettavien kyselyiden määrää ja piilottaa tietokantakohtaisia eroavaisuuksia tasosta riippuen. Se mahdollistaa myös eri tietokantajärjestelmien käyttämisen samalla koodilla, sillä ORM-ratkaisut kääntävät kyselyt tietokantaspesifiseen muotoon. (Why Use an ORM and an Abstraction Layer? n.d.)

3.2 Versionhallinta

Versionhallintajärjestelmän tarkoitus on seurata ja hallita useita versioita samoista tiedostoista, mikä merkitsee ohjelmistokehityksessä usein sovelluksen lähdekoodia.

Versionhallinnan avulla muutosten seuranta on mahdollista, sillä kaikki tehdyt muutokset tallennetaan erilliseen säilöön (engl. repository). Seuratut tiedostot voi palauttaa mihin tahansa edelliseen versioon käyttäjän niin halutessa. (Boag 2008.)

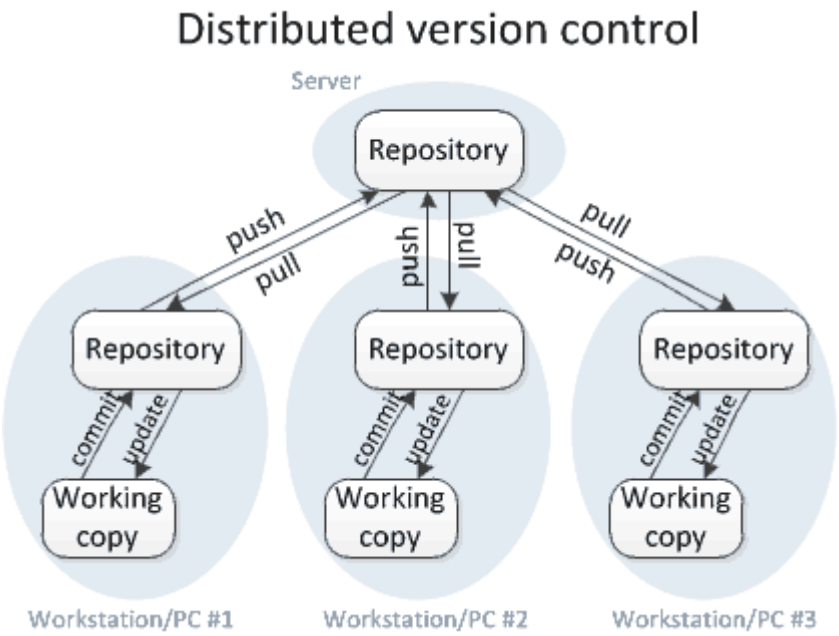
Suurinta hyötyä versionhallinnasta on projekteissa, joissa useat kehittäjät muokkaavat samoja tiedostoja. Versionhallinnan avulla luodaan kaikille tiedoston muokkaajille oma haara (engl. branch), jossa kehittäjät voivat muokata omaa versiotaan tiedostosta. Lopuksi muutokset yhdistetään yhteiseen haaraan. (Boag 2008.) Versionhallinnasta on hyötyä myös ongelmanratkaisussa. Viat voi yksinkertaisesti paikantaa vertailemalla tiedoston viallista versiota viimeisimpään toimivaan versioon. Uusien kehittäjien on helppo liittyä projektiin, sillä uusin versio voidaan ladata suoraan versionhallinnasta. (Azarian 2013.)

Tietokantojen versionhallintaankin tarvitaan perinteisiä ohjelmiston versionhallintajärjestelmiä. Kaikki tässä tutkimuksessa vertailtavat sovellukset hyödyntävät versionhallintaa, jolloin isoissakin kehittäjätiimeissä pystytään järjestelmällisesti koordinoimaan muutoksia.

Versionhallintasovellukset jaetaan usein kahteen pääryhmään: hajautettuihin ja keskitettyihin versionhallintasovelluksiin (Lionetti 2012). Näitä termejä ei tule sekoittaa itse tietokannan keskittämiseen ja hajauttamiseen, sillä niillä ei ole lainkaan tekemistä ohjelmiston versionhallinnan kanssa.

3.2.1 Hajautettu versionhallinta

Hajautetut ratkaisut ovat kehittäjäkohtaisia. Ne eivät välttämättä nojaa keskuspalvelimeen, joka tallentaa kaikki projektitiedostojen versiot. Mikään ei kuitenkaan estä keskuspalvelimen käyttämistä projektin pääversion säilytyspaikkana. Sen sijaan jokainen kehittäjä kloonaa kopion koko säilöstä (engl. repository) omalle kovalevylleen. Tämä kloonattu kopio pitää sisällään täyden kehityshistorian kloonaushetkeen saakka, sekä alkuperäisen projektin metadatan. Hajautettuja sovelluksia ovat mm. Git, Mercurial sekä Bazaar. (Lionetti 2012.) Kuviossa yksi on kuvattu perinteinen hajautettu versionhallintajärjestelmä.

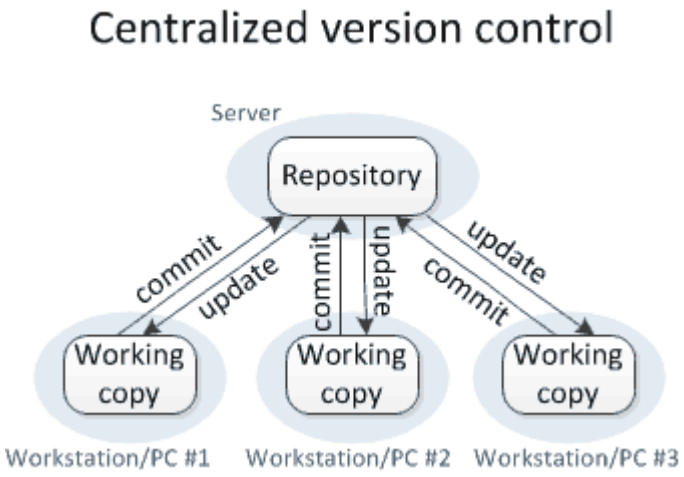


Kuvio 1. Hajautettu versionhallintajärjestelmä. (Ernst 2012.)

Preiße ja Stachmann (2014) listaavat lukuisia hajautetun järjestelmän etuja keskitettyyn järjestelmään verrattuna. Hajautetut järjestelmät ovat tehokkaampia, sillä lähes kaikki operaatiot toteutetaan paikallisesti ilman tarvetta verkon yli toimivalle keskuspalvelimelle. Kehittäjät voivat myös käyttää paikallisia kehityshaaroja, jolloin vaihto tehtävästä ja versiosta toiseen sujuu nopeasti. Tiedostojen varmuuskopiointi on turhaa, sillä jokaisella kehittäjällä on täysi kehityshistoria kovalevyillään, jolloin kokonaisen projektin katoaminen usean hengen kehitystiimissä on epätodennäköistä. (Preiße & Stachmann 2014.) Etuna hajautetussa tietokantojen muutosten hallinnassa on kehittäjien mahdollisuus tehdä muutoksia omissa ympäristöissään ja testata ne läpikohtaisin, ennen kuin muutokset lisätään tuotantoon tai testaukseen kaikkien saataville. (Sheldon, Richardson & Davis 2014, 49.)

3.2.2 Keskitetty versionhallinta

Keskitetyn versionhallinnan keskiössä on projektin pääkopio, joka sijaitsee usein palvelimella, johon kehittäjät tekevät muutoksensa suoraan. Muut kehittäjät pystyvät näkemään tehdyt muutokset, ja voivat ladata omat muutoksensa keskuspalvelimelle suoraan, jolloin ne päivittävät kaikkiin tiedostoihin tehdyt muutokset. (Lionetti 2012.) Kuviossa kaksi on kuvattu keskitetty versionhallintajärjestelmä.



Kuvio 2. Keskitetty versionhallintajärjestelmä. (Ernst 2012.)

Useimmat nykyaikaiset versionhallinnat käsittelevät muutosryhmiä (engl. changeset), jotka ovat monesta muutoksesta koostuvia kokoelmia ja joita tulee käsitellä yhtenä kokonaisuutena. Tyypillinen asiankäsittely (engl. workflow) keskitetyssä versionhallinnassa on ladata muiden tekemät muutokset keskuspalvelimelta, tehdä omat muutokset ja varmistaa niiden toimivuus, ja vasta lopuksi siirtää omat versiot keskuspalvelimelle muiden kehittäjien nähtäville. (Lionetti 2012.)

Etuina keskitetyssä ratkaisussa hajautettuun versionhallintaan verrattuna on suurten tiedostojen ja muutoshistorian käsittely. Suurten binääritiedostojen lukuisia versioita ei tarvitse ladata jokaisen kehittäjän omalle kovalevylle, vaan ne tallennetaan keskuspalvelimelle. Jos projektiin on tehty kymmeniä tuhansia muutoksia, on kokonaisen muutoshistorian lataaminen hajautetusta järjestelmästä aikaa- ja levytilaa vievää. (Lionetti 2012.)

3.3 Muutosten hallinta relaatiotietokannoissa

Tietokantojen muutosten hallinnan tärkeyttä ei pidä aliarvioida. Ketterien kehitysmallien myötä päivityksiä julkaistaan useammin, jolloin myös tietokantoihin tehtävät muutokset tulevat voimaan pienempinä kokonaisuuksina. Nopea julkaisutahti vaatii erityistä tarkkuutta tietokantojen ylläpitäjiltä, sillä mitä useammin muutoksia tehdään, sitä suurempi mahdollisuus on tietokantavirheiden esiintymiselle. Muutosten

hallintaan on kuitenkin lukuisia erilaisia vaihtoehtoja kokonaan käsin kirjoitettavista muutoksista täysin automatisoituihin prosesseihin. (Rees 2014.)

Muutokset tietokantoihin eivät ole täysin ongelmattomia. Allen (n.d) väittää relaatiotietokantojen olevan suunnittelunsa takia muutosten vastainen teknologia. Tietokantojen normalisointi, tietojen eheys sekä nopeat kyselyt ovat relaatiokantojen suuria etuja, mutta joustamaton malli hidastaa väistämättä muutosten tekemistä. Valtavien relaatiotietokantojen monimutkaisia skeemoja on hankala muuttaa jälkikäteen. Jos tietokannan varaan on rakennettu sovellus, pitää tietokantamuutosten yhteydessä myös sovelluksen lähdekoodia muuttaa vastaamaan uutta rakennetta. Miljardeja dollareita käytetään vuosittain tiedon ”täydelliseen” mallintamiseen. Näin mallinnettujen relaatiotietokantojen uskotaan kestävän sovelluksen koko elinkaaren ja pysyvän rakenteeltaan muuttumattomana. Sovelluksen elinkaaren aikana esiintyvien bugien korjauksen sekä lisättävien ominaisuuksien takia muutos on kuitenkin väistämätöntä suunnitteluvaiheen valmisteluista huolimatta. (Allen n.d.)

Schumacher (2005) listaa valvontapalvelussakin tietokannan hallintajärjestelmänä käytettävän SQL Serverin muutosten hallinnan perusedellytykset neljään aktiviteettiin. Tietokannan skeeman arkistoinnissa otetaan tilannevedokset (engl. snapshot) kannan asetuksista, sisällöstä ja rakenteesta, eli säilytetään tietokanta siinä muodossa, miltä se juuri sillä hetkellä näytti. Toinen tehtävä on tietokantojen vertailu keskenään. Juuri tätä tietokantojen vertailua varten on luotu tässä opinnäytetyössäkin tutkittavia työkaluja. Kolmas kohta on tietokannan kopiointi tai kloonaus. Se onnistuu Microsoftin omilla apuvälineillä. Neljäs ja viimeinen muutosten hallinnan perusedellytys on tietokannan synkronointi. Se suoritetaan yleensä kahdesta syystä: kun ei-toivottu muutos tietokannan tietoturvassa tai asetuksissa tapahtuu, voi tietokannan ylläpitäjä palauttaa tietokannan aikaisempaan versioon tai ajaa muutokset yhteen tietokantaan ja synkronisoida muutokset sen kautta muihin tietokantoihin. (Schumacher 2005, 77–78.)

Skriptit

Ennen ketterän kehityksen esiinmarssia suurin osa tietokantojen hallinnasta tehtiin sitä varten luoduilla skripteillä. Nämä skriptit luotiin hoitamaan automaattisesti tietokannan varmuuskopiointi, indeksointi, tilastointi sekä lokien hallinta. (Fritchey

2015.) Eri tarkoituksia varten luotuja tietokantaskriptejä on verkossa paljon tarjolla useaan eri tarpeeseen, mutta ne ovat kuitenkin vaivalloisesti ylläpidettäviä, sillä aina kun tietokannan rakenteeseen tehdään muutoksia, pitää skripti päivittää vastaamaan uutta rakennetta. (Mullins 2002.)

Tietokantaskriptit ovat vieläkin varteenotettava vaihtoehto niiden yksinkertaisuuden takia, sillä niitä voidaan versioida samassa versionhallintaohjelmassa muun koodin kanssa. Skriptejä käytettäessä muutosten dokumentointi on erittäin tärkeää, jotta varmistutaan tietokannan versionhallinnan luotettavuudesta ja oikeellisuudesta. (Verveer 2011.)

Tietokantojen vertailusovellukset

Tietokantojen versionhallinta on oleellinen osa sovelluskehitystä. Käyttäjän löytämän virheen korjaaminen vaatii mahdollisuutta palauttaa sovellus sekä tietokanta samaan versioon, jossa virhe havaittiin. (Allen 2008.) Versionhallinnan teoria on, että jokainen tietokantaan tehty muutos asettaa tietokannan sen hetkiselle tilalle oman versionumeronsa, jolloin tietokannan taakse- ja eteenkierto onnistuu turvallisesti. Versionhallinnan avulla muutokset ovat myös helpommin hallittavissa, jäljennettävissä ja siirrettävissä. (Nimkar 2016a.)

Tietokannan ylläpitäjän työtä helpottamaan on luotu useita työkaluja, jotka vertailevat tietokantaympäristöjä keskenään. Ne tutkivat järjestelmäkatalogeja, tietorakenteita sekä DDL -skriptejä. (Mullins 2002.) DDL (engl. Data Definition Structure) on syntaksi, jota käytetään tietorakenteiden ja tietokannan skeemojen määrittämiseen (Generating DDL scripts n.d). Työkalu on lähestulkoon vaatimus monimutkaisissa tietokantaratkaisuissa, sillä muutosten jäljittäminen yhdestä ympäristöstä toiseen on hyvin hankalaa. Mitä useampi ympäristö on olemassa, sitä vaikeammaksi muutosten hallinta käy. (Mullins 2002.)

Baseline

Mikäli käytössä on tietokantojen versiointisovellus tai tietokantaskriptejä, niiden versiointia hoidetaan yleensä perinteisellä versionhallintasovelluksella, kuten Gitillä tai Subversionilla. Ennen kuin tietokannan versiointia voi aloittaa, on luotava ns. baseline, joka on tyhjä versio tietokannasta, jonka päälle kaikki muutokset tehdään. Baseli-

nen voi tehdä kopioimalla olemassa olevan tietokannan skeeman tai luomalla runkorakenteen (engl. skeleton copy), josta typistetään kaikki rivit kaikista tauluista. Baselinen luominen runkorakenteista on kannattavampaa kuin skeemoista, sillä skeemoista ei välttämättä pystytä luomaan kokonaista tietokantaa onnistuneesti. (Nimkar 2016b.)

4 Esikatsaus vertailtaviin versionhallintasovelluksiin

Liquibase

Liquibase on Daticalin kehittämä avoimen lähdekoodin tietokantojen versionhallintaratkaisu. Se on kehittäjäkohtainen tietokantojen versionhallintasovellus, jonka toiminta perustuu muutoslokeihin. Muutoslokit sijaitsevat jokaisen kehittäjän omissa kehitysympäristöissä. Muutosloki voi olla natiivisti XML-, JSON-, SQL- tai YAML-formaatissa. Myös mikä tahansa muu formaatti on mahdollista luoda Liquibasen laajennuksen avulla. (Source control for your database n.d.)

Flyway

Flyway on Boxfusen kehittämä avoimen lähdekoodin migraatiotyökalu, jonka kehitysajatuksena on ollut helppous ja yksinkertaisuus konfiguraation kustannuksella. Migraatiot kirjoitetaan joko SQL-komentoina tai Javalla. Flywayn yksinkertaisuus kulminoituu käytettävissä olevien komentojen määrään; sovellus tukee vain kuutta eri komentoa. Sovellus on luotu hajautetun versionhallinnan jatkuva integraatio mielessä, jonka takia se on hyvin yleisesti käytetty ratkaisu ketterissä ohjelmistoprojekteissa. (Documentation n.d.)

Schema Zen

Schema Zen on pieni SQL Serverin versiointityökalu, joka on nopea ja kevyt. Vertailtavista sovelluksista se on ainoa, joka voidaan ajaa ainoastaan komentoriviltä. Schema Zen on aktiivisen kehityksen alla, ja sen suurin valtti on kyky takaisinmallintaa (engl. reverse engineer) tietokanta sen luontiskriptiin. Sovellus kykenee vertailemaan kahta tietokantaa pelkästään niiden luontiskirptejä vertailemalla. Se ei sisällä yhtä kattavia ominaisuuksia kuin muut vertailtavat ratkaisut, mutta sen nopeus on erin-

omainen ominaisuus automaatiota ajatellen. (Schema Zen - Script and create SQL Server objects quickly n.d.)

Sqitch

Sqitch on itsenäinen sovellus, joka ei tukeudu minkään yksittäisen viitekehyksen tai alustan varaan. Se kykenee luomaan riippuvuuksia muutosten välillä, mikä mahdollistaa tarkan ajojärjestyksen, kun tiedostoja ladataan versionhallintaan. Sqitch ei tue natiivisti SQL Serveriä. Aktiivinen kehittäjäyhteisö on kuitenkin luonut version, joka tarjoaa SQL Serverille täyden tuen. (Sqitch 2016.)

dbv

dbv on tietokantojen versionhallintaan luotu web-sovellus, joka kykenee hallitsemaan skeemoja sekä tarkastamaan skriptejä. Koska se on web-sovellus, sen mukana tulee php-pohjainen graafinen hallintajärjestelmä. Web-pohjaisuutensa ansioista se on sopiva vaihtoehto isoille kehitystiimeille, sillä kaikki jäsenet näkevät yhdellä silmäyksellä muiden kehittäjien tietokantamuutokset graafisesta käyttöliittymästä. (Database version control, made easy! 2016.)

5 Vertailtavien sovellusten valinta

5.1 Dokumentoinnin määrä ja laatu

Liquibase

Liquibasen dokumentointi on jaettu viiteen osioon. Ensimmäiseksi etusivulla on kerrottu lyhyesti Liquibasen pääkonseptit, jotka tarjoavat pikakatsauksen sovelluksen tärkeimpiin toiminnallisuuksiin. Seuraavassa osassa kerrotaan tarkasti Liquibasen ytimenä toimivan muutoslokitiedoston toiminnasta. Muun muassa muutoslokin toteuttaminen eri formaateilla kerrotaan kattavasti ja selkeästi. Kolmannessa osassa kerrotaan komennoista. Komentojen dokumentaatio on jaettu omiksi luvuikseen. Käytännön esimerkit ovat hyvä lisä, sillä ne helpottavat komentojen hahmottamista. Neljäs kohta käy läpi sovelluksen ajamista itse ajettavana ja automatisoituna prosessina. Lisäksi esitellään esimerkkien kera kolmannen osapuolen sovelluskehysten yhteensovittaminen Liquibasen kanssa esimerkkien kera. Laatu on tässä osiossa vaihte-

levaa, sillä esimerkiksi valvontapalvelun käyttämistä Spring-viitekehyksen attribuuteista on kerrottu todella niukasti, mutta esimerkiksi Mavenista on huomattavasti enemmän tietoa ja esimerkkejä. Viimeinen osio keskittyy Liquibasen luomiin tietokantatauluihin, jotka pitävät kirjaa ajetuista muutoslokeista ja rajoittavat Liquidbasen instanssien määrän yhteen. Liquibase luo tietokantataulut automaattisesti ensimmäisten migraatioiden yhteydessä. (Liquibase documentation 2016.)

Dokumentointi on hyvin ryhmitelty, ja haluttu tieto löytyy helposti. Tietoa on paljon tarjolla, mutta suurin puute on asennusohjeiden summittaisuus, eikä pika-aloitusohje tarjoa juurikaan ongelmanratkaisuehdotuksia, mikäli asennuksen yhteydessä ilmenee häiriöitä.

Flyway

Ensisilmäyksellä dokumentaatiota on runsaasti. Dokumentaatio on jaettu helposti ymmärrettäviin osioihin. Asennusohjeet eivät sisälly dokumentointiin, vaan ovat erillään omassa osiossaan.

Komennoista kerrotaan ensimmäisessä osassa. Jokaiselle kuudelle komennolle on oma, suppeahko sivunsa. Komentojen selventämisessä auttavat havainnollistavat kuvat. Käytännön esimerkkejä komentojen käytöstä ei ole lainkaan. Toisessa osassa keskitytään Flywayn konsepteihin. Migraatioista kertovaan lukuun on panostettu huomattavasti enemmän kuin aiempiin komentoihin. Alussa on yleiskatsaus migraatioihin, jonka jälkeen muut aiheeseen liittyvät asiat ovat omissa aliluvuissaan. Myös Java- sekä SQL-pohjaiset migraatiot ovat omissa alaluvuissaan käytännön esimerkkien ja yleisimpien käyttötapausten kera. Callbackeista on vain yksi luku, jossa on listattu Flywayn tukemat hookit ja kerrottu lyhyesti niiden käyttömahdollisuuksista Javalla sekä SQL:llä. Kolmas osio sisältää dokumentaatiota Flywayn ajosta ja suorittamisesta. Jokaisesta vaihtoehdosta on latauslinkkejä sekä ohjeet Flywayn konfigurointiin valitulla tekniikalla. Jäljellä olevissa osioissa käsitellään eri tietokantojen vaatimuksia Flywayn toiminnan kannalta, listataan Flywayta koskevia artikkeleita ja vedeoita sekä mahdollisuuksista tukea tai osallistua sovelluksen kehittämiseen. (Evolve your Database Schema easily and reliably across all your instances n.d.)

Kokonaisuudessaan Flywayn dokumentaatio on hyvin kahtiajakautunut. Joissain osioissa, kuten migraatioista sekä Flywayn suorittamisesta, on laajasti materiaalia, mutta komentojen syvällisen informaation puutteellisuus on silmiinpistävää.

Schema Zen

Schema Zen on pieni sovellus, joten dokumentaatio on sen mukaista. GitHubissa sovelluksen mukana tuleva readme-tiedosto on käytännössä ainoa tietolähde. Se sisältää ainoastaan kolmen pääkomennon esimerkit sekä lyhyet selvitykset niiden toiminnallisuudesta. Tekstiä ei ole paljoa, mutta se on hyvin tiivistetty ja sisältää tärkeimmät tiedot täsmällisiä asennusohjeita lukuun ottamatta. Määrällisesti Schema Zenissä on vähiten dokumentaatiota kaikista vertailtavista sovelluksista. (Schema Zen - Script and create SQL Server objects quickly n.d.)

Sqitch

Dokumentaatio koostuu Sqitchin pääkonseptien esittelystä, lukuisista erilaisista asennusohjeista, tietokantakohtaisista käyttöohjeista (eng. tutorial) sekä presentaatioista. Sovelluksen avainkonseptit ja eroavuudet muihin tyypillisiin migraatiopohjaisiin ratkaisuihin perustellaan aluksi pääpiirteittäin, jonka jälkeen esitellään itse sovelluksen asennusohjeet. Lopuksi kerrotaan tietokantakohtaisesti seurattavan kannan luonnista, kehityksestä ja ylläpidosta. Kuten sovelluksen esikatselussa todettiin, Sqitch ei tue SQL Serveriä natiivisti, vaan ainoastaan käyttäjäyhteisön luomana laajennuksena. Tästä syystä SQL Serverille ei ole lainkaan käyttöopasta Sqitchin omassa projektissa, mutta kattava käyttöohje kuitenkin löytyy laajennuksen GitHub -forkista (Sqitch tutorial for MS SQL 2014.)

dbv

dbv:n dokumentointi alkaa lyhyehköillä asennusohjeilla, jonka jälkeen luetellaan neljä yleisintä käyttötapausta ja opastukset niihin. Käyttötapausten havainnollistamiseen käytetään kuvia. Myöskään dbv ei tue SQL Serveriä suoraan. Dokumentoinnissa kuitenkin opastetaan muiden kuin MySQL-tietokannan käyttäjiä luomaan oma php-luokka, jonka avulla muidenkin tietokantajärjestelmien käyttäminen on mahdollista. dbv:n dokumentointi on lyhyt, mutta kuvien avulla havainnollistaminen antaa sille selkeyttä. Käyttäjätapaukset antavat käytännönläheisiä esimerkkejä, miten asiankä-

sittely kaikissa neljässä tapauksessa tällä sovelluksella etenee. (Database version control, made easy! 2016.)

5.2 Käytettävyys ja asennus valmiina olevaan tietokantaan

Liquibase

Liquibase tukee SQL Server 2008 -tietokantajärjestelmää natiivisti. Jo olemassa olevaan tietokantaan liittäminen onnistuu kahdella tapaa. Ensimmäinen tapa on monimutkaisempi, jolloin muutoslokia muokkaamalla saadaan näyttämään siltä, että Liquibase olisi ollut käytössä alusta alkaen. Toinen tapa on nopeampi, jolloin määritellään Liquibasen versiohistorian alkamisaika nykyiseen versioon. Nopeampi tapa ei mahdollista versionhallintaa nykyisestä versiosta aiempaan versioon, mutta on helpompi asentaa. (Liquibase documentation 2016.)

Flyway

Flywayn asentamiseen valvontapalvelulle on käytännössä kaksi tapaa. Ensimmäinen tapa on asentaa sovellus komentoriviltä. Komentorivityökalu toimii ilman muita vaatimuksia. Asetukset määritellään omaan konfiguraatiotiedostoonsa. Nimensä mukaisesti kaikki toiminnot suoritetaan komentoriviltä. Toinen tapa on käyttää kokoonpanotyökalu Antia. Asennus Java API:n avulla ei toimi, sillä valvontapalvelu ei käytä Flywayn vaatimaa projektinhallintatyökalu Mavenia. (Evolve your Database Schema easily and reliably across all your instances n.d.)

Schema Zen

Schema Zenin asennusohjeet ovat hyvin selkeät. Komentoriviltä ajetaan komento, joka luo tiedostorakenteen. Yhdellä komennolla voidaan vertailla kahta tietokantaa ja luoda SQL-skripti, jonka avulla tietokannat voidaan luoda rakenteeltaan identtisiksi. Alkukonfiguraation jälkeen Schema Zenin käyttäminen on yksinkertaista, sillä komentoja tarvitaan vain yksi. Haittapuolena on luotujen SQL-skriptien manuaalinen ajaminen, sillä Schema Zen ei suoraan tue automatisointia. Ulkoisella skriptillä on kuitenkin mahdollista toteuttaa automatisointi. (Schema Zen - Script and create SQL Server objects quickly n.d.)

Sqitch

Sqitch ei tue SQL Serveriä natiivisti, mutta laajennuksen avulla sen käyttö onnistuu. Sqitch toimii samalla periaatteella myös yhteisön luoman laajennuksen avulla, tarjoten täyden tuen SQL Serverille. Laajennuksen kattava asennusohjeistus kertoo itse asennusprosessista, sovelluksen yleisimmät käyttötapaukset ja ratkaisuja useisiin ongelmatilanteisiin (Sqitch MSSQL Tutorial 2016).

dbv

Koska dbv on php-pohjainen web-sovellus, sen ajamiseen tarvitaan Apachen web-palvelinta, jota valvontapalvelun ympäristössä ei ole. Myöskään natiivia tukea SQL Serverille ei ole, joten dbv ei ole heti käyttövalmis valvontapalvelun yhteyteen. Sovellukselle ei ole tehty myöskään laajennusta SQL Serverille yhteisön toimesta kuten Sqitchin tapauksessa. (Database version control, made easy! 2016.)

5.3 Kehitysyhteisön koko ja aktiivisuus

Liquibase

Liquibasen sivuilla on oma osionsa yhteisölle, joka ohjaa sovelluksen käyttäjiä ja kehittäjiä muiden resurssien pariin. Esimerkiksi kysymykset koskien sovelluksen käyttöä ohjataan Stack Overflowhun Liquibasen omalla tagilla, bugit kirjataan Liquibasen Jiraan, keskustelut foorumeille ja niin edelleen. Stack Overflown Liquibase -tagin kysymysmäärän sekä Liquibasen omalle foorumille rekisteröityneiden käyttäjien lukumäärän perusteella Liquibasella on muihin sovelluksiin verrattuna suuri ja aktiivinen kehittäjäyhteisö. Liquibase on suunniteltu laajentamisen ehdoilla. Kehittäjäyhteisön luomia laajennuksia onkin tarjolla jo lähes sata.

Flyway

Toisin kuin Liquibasella, Flywaylla ei ole yhtä tiettyä osiota yhteisöä varten, vaan tiedot on ripoteltu ympäri sivustoa. Myös Flywaylla on oma tagi Stack Overflowssa, jossa voi esittää kysymyksiä sovelluksen käyttöön liittyen muille käyttäjille.

Flywayn kehittäjä Datical tarjoaa bugin korjausta tai uusien ominaisuuksien kehittämistä maksua vastaan. Flyway on avoimen lähdekoodin sovellus, joten edellä mainit-

tujen toimintojen tekeminen on mahdollista myös itse tai liittyä johonkin jo toimivista kehitystiimeistä.

Schema Zen

Schema Zenin yhteisö on keskittynyt viestipalvelu Gitteriin. Keskustelu ei ole kovin aktiivista, sillä viestien välillä on pahimmillaan yli vuoden tauko. Yhteisön kehittämät muutokset hoidetaan GitHubin pull requesteilla. Edellä mainittujen seikkojen perusteella vaikuttaa kehittäjäyhteisö olevan hyvin pieni eikä kovinkaan aktiivinen.

Sqitch

Myös Sqitchin yhteisö on kerääntynyt GitHubiin, jossa keskustelu on Schema Zeniä aktiivisempaa. Sovelluksen pääkehittäjä vastaa henkilökohtaisesti suureen osaan käyttäjien kirjaamista ongelmista. Kehittäjäyhteisön aktiivisuudesta kertoo paljon myös SQL Serveriä varten luotu oma mittava laajennus, jonka kehittäminen on kokonaan ulkoisten kehittäjien vastuulla. Laajennus sisältää kaikki samat toiminnallisuudet kuin Sqitchin pääversio.

dbv

dbv on vertailtavista sovelluksista kaikista pienin, ainakin kun vertailtavana on kehittäjäyhteisö. Sovelluksen kehittäjä ei ole vastannut yli vuoteen käyttäjien ongelmiin tai pull requesteihin sovelluksen GitHubissa. Myöskään mitään muuta alustaa keskustelulle dbv:stä ei GitHubin lisäksi ole.

5.4 Testattavien sovellusten valinta

Pisteytyksessä sovellukset asetetaan paremmuusjärjestykseen edellä käsiteltyjen kohtien mukaan. Paras sovellus saa viisi pistettä, toiseksi paras neljä pistettä, kolmanneksi paras kolme pistettä, neljänneksi paras kaksi pistettä ja viimeinen yhden pisteen. Pisteet lasketaan yhteen, ja kahta eniten pisteitä saanutta sovellusta vertaillaan tarkemmin virtuaaliympäristössä. Näistä kahdesta sovelluksesta valitaan valvontapalvelun vaatimuksiin nähden parempi vaihtoehto. Taulukossa yksi on kuvattu sovellusten pisteytys.

Taulukko 1. Sovellusten vertailutaulukko.

	Dokumentaatio	Käytettävyys	Kehitysyhteisö	Yhteensä
Liquibase	5	3	5	13
Flyway	4	4	4	12
Schema Zen	1	5	2	8
Sqitch	3	2	3	8
dbv	2	1	1	4

Yhteispisteiden perusteella virtuaaliympäristöihin asennetaan Liquibase ja Flyway. Kaksikko erottui selvästi edukseen dokumentaation laadun ja kehittäjäyhteisön koon perusteella. Käytettävyydessä Schema Zen oli edukseen, mutta dokumentaation keveyden ja pienehkön kehittäjäyhteisön takia sitä ei valittu toiseen vaiheeseen.

6 Valittujen sovellusten vertailu

6.1 Virtuaaliympäristöjen valmistelu

Ennen vertailtavien sovellusten asentamista valmistellaan molemmille testattaville ratkaisuille omat virtuaaliympäristöt. Koska kumpikin vertailtavista ratkaisuista on hajautetun versionhallinnan ratkaisu, ovat ympäristöt täysin identtiset. Virtuaaliympäristöjen käyttöliittymät ovat samat kuin valvontapalvelun tuotantoympäristössä, eli Windows Server 2008. SQL Server 2008 -tietokantajärjestelmä sekä Microsoft SQL Server Management Studio asennetaan molempiin virtuaalipalvelimiin. Testitietokantana käytetään Northwind varmuuskopiosta palautettua mallikantaa. Testikannan rakenne on kuvattu liitteessä yksi. Testitietokannasta luodaan kaksi versiota, DB_1 ja DB_2, molempiin virtuaaliympäristöihin simuloimaan usean tietokannan muutosten hallintaa.

Virtuaaliympäristöjen hallintaan käytetään VMware Workstation Pro -ohjelmistoa. Asennuksen jälkeen luodaan virtuaalipalvelimesta klooni, jolloin lopputuloksena on

kaksi täysin identtistä virtuaaliympäristöä. Taulukossa kaksi listataan virtuaaliympäristöjen spesifikaatio.

Taulukko 2. Virtuaalipalvelinten spesifikaatio.

Käyttöjärjestelmä	Windows Server 2008
Tietokantajärjestelmä	SQL Server 2008 R2
Tietokannanhallintajärjestelmä	Microsoft SQL Server Management Studio
Testitietokanta	Northwind -mallitietokanta
Muut	Java Runtime Environment 8 Update 11
	SQL Server JDBC Driver

6.2 Liquibase

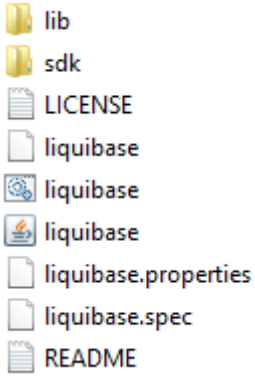
Asennus

Testausta varten Liquibase asennetaan komentoriviltä, sillä testaamista varten luoduissa virtuaaliympäristössä ei ole asennettuna valvontapalvelua tietoturvasyistä.

Komentorivin asennus suoritetaan lataamalla Liquibasen verkkosivuilta uusin versio, purkamalla se haluttuun hakemistoon ja lopuksi ajamalla liquidbase.bat -skripti, joka luo liquibase.jar -tiedoston. Valvontapalveluun sovellus voidaan asentaa vaihtoehtoisesti asettamalla liquibase.jar-tiedosto luokkapolulle, josta kokoonpanotyökalu Ant pääsee siihen käsiksi.

Hakemistorakenne

Alla on kuvattuna Liquibasen hakemistorakenne.



Kuvio 3. Liquibasen hakemistorakenne.

Hakemisto lib sisältää laajennusten kirjastot. Vakiona siellä on vain muutoslokille yaml -tuen tarjoava snakeyaml -kirjasto. Liquibasen sdk shell ja batch skriptit ovat sdk kansiossa. Hakemisto on käytännössä Liquibasen kehittäjiä varten luotu, sillä se sisältää ainoastaan kehityksessä käytettäviä kirjastoja ja dokumentaatiota. Hakemiston juuressa sijaitsee itse sovelluksen toiminnallisuudesta vastaava Liquibasen jar -tiedosto, sekä konfiguraatiotiedosto liquibase.properties.

Muutosten hallinta

Onnistuneen asennuksen jälkeen luodaan muutosloki, joka toimii Liquibasen muutosten hallinnan ytimenä. Muutosloki on Liquibasen tärkein yksittäinen resurssi. Siihen kirjataan kaikki tietokannan muutokset siinä järjestyksessä, missä ne ajetaan tietokantaan. Jokaiselle tietokannalle pitää luoda oma muutosloki. Olemassa olevaan tietokantaan voidaan tehdä muutosloki komennolla liquibase generateChangeLog. Tällöin tuloksena on muutosloki, joka sisältää koko kannan sen hetkisen rakenteen. Mukana ovat myös näkymät ja avaimet. Komento ei kuitenkaan pysty generoimaan liipaisimia, proseduureja tai funktioita muutoslokiin.

Kuviossa kolme kuvatussa muutoslokissa on yksi muutos, joka lisää tauluun order details sarakkeen notes. Komennolla liquibase migrate muutoslokin muutokset viedään tietokantaan. Samalla tietokannasta luodaan baseline, jota Liquibase käyttää tietokantojen muutosten lähtöpisteenä.

```
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

        <changeSet id="1" author="sami">
            <addColumn
                schemaName="dbo"
                tableName="order details">
                <column name="notes" type="varchar(500)"/>
            </addColumn>
    </changeSet>
</databaseChangeLog>
```

Kuvio 4. Esimerkki Liquibasen muutoslokista.

Kun Liquibasen komento ajetaan ensimmäistä kertaa, Liquibase luo samalla tietokantaan kaksi uutta taulua: DATABASECHANGELOG sekä DATABASECHANGELOGLOCK. DATABASECHANGELOG pitää kirjaa tietokantaan ajetuista muutoksista. DATABASECHANGELOGLOCK varmistaa, että ainoastaan yksi instanssi Liquibasesta on käynnissä kerrallaan. (Source control for your database n.d.)

Mikäli on tarve peruuttaa tietokantaan tehtyjä muutoksia, suoritetaan takaisinvienti muutoslokin kautta. Rollback -tagien sisään voidaan laittaa mitä tahansa Liquibasen perustageista. Vaihtoehtoisesti takaisinvienti voidaan suorittamisen sijaan kirjoittaa tiedostoon, jolloin pystytään tarkastelemaan kyselyä ilman sen toteuttamista.

Usean kannan muutokset

Liquibase voi hoitaa tietokantojen välisten eroavaisuuksien tarkastuksen itsenäisesti diff -komennolla. Tällöin Liquibase vertailee kahden tietokannan tauluja, sarakkeita, näkymiä, avaimia ja indeksejä keskenään, luoden tarvittaessa muutoslokin, jolla voidaan synkronoida tietokannat keskenään.

Muutoslokiin voi asettaa kontekstin, joka määrittelee ympäristössä ajettavat migraatiot. Esimerkiksi kontekstilla dev merkatut muutokset voidaan määritellä suoritettaviksi vain kehitysympäristössä.

Konfiguroitavuus

Mikäli käytössä on Ant tai Maven, voidaan komentojen nopeuttamista varten luoda konfiguraatioasetukset useille kannoille. Edellä mainittujen työkalujen konfiguraatiotiedostoihin lisätään Liquibasen vaatimat tiedot, kuten ajurin osoite ja tyyppi, tietokannan osoite, käyttäjätunnus, salasana ja muutoslokin sijainti. Konfiguraatioasetusten ansiosta komentoihin ei tarvitse kirjoittaa edellä mainittuja kohtia ajettaessa, sillä Liquibase hakee tiedot suoraan asetuksista.

Komentoriviä käytettäessä voidaan käyttää vain yhtä liquibase.properties -konfiguraatiotiedostoa kerrallaan**.** Näin ollen useamman kuin yhden tietokannan hallinta hankaloituu. On kuitenkin mahdollista luoda omat konfiguraatiotiedostot jokaiselle tietokannalle ja vaihdella Liquibasen käyttämää tiedostoa tarpeen mukaan.

```
classpath=C:\\Program Files (x86)\\Microsoft JDBC Driver 4.0 for SQL Server\\sqljdbc_4.0\\enu\\sqljdbc4.jar
driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
url=jdbc:sqlserver://localhost:1433;databaseName=DB_1
username=Sami
password=Passw0rd
changeLogFile=C:\\Temp\\changelog.xml
```

Kuvio 5. Esimerkki liquibase.properties -tiedostosta

Ylläpito asennuksen jälkeen

Liquibase ei tarvi erityisiä huoltotoimenpiteitä asennuksen jälkeen tai käytön yhteydessä. Uusi versio Liquibasesta voidaan asentaa ajamalla uuden version liquibase.bat vanhan version työhakemistossa.

6.3 Flyway

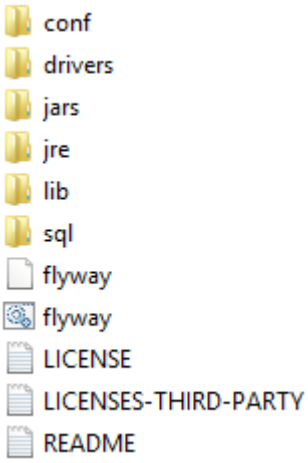
Asennus

Flywayta ei tarvitse erikseen asentaa, vaan riittää, että Flywayn verkkosivuilta ladataan uusin versio sovelluksesta ja puretaan se haluttuun työhakemistoon. Lisätään MS SQL JDBC -ajuri hakemistoon drivers. Aluksi asetetaan flyway.conf -konfiguraatiotiedostoon tietokannan url, ajuri, käyttäjätunnus ja salasana.

Valmiina olevasta tietokannasta luodaan ensin baseline. Komennolla flyway baseline kantaan luodaan taulu schema_version, joka pitää lokia tehdyistä muutoksista.

Hakemistorakenne

Alla on kuvattuna Flywayn hakemistorakenne.



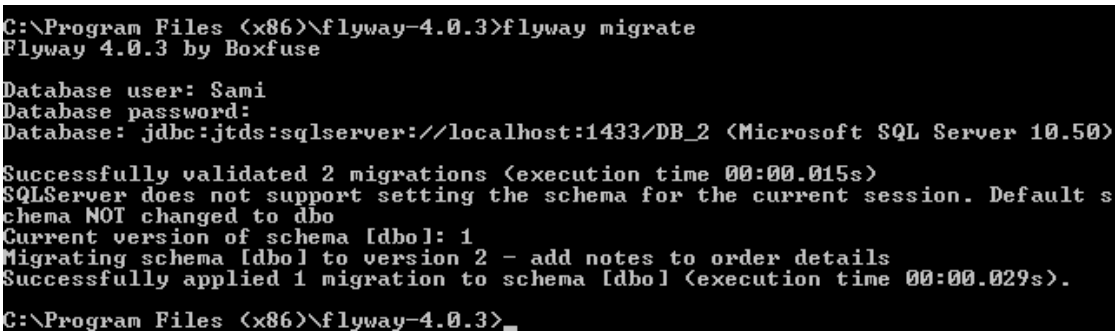
Kuvio 6. Flywayn hakemistorakenne.

Conf sisältää Flywayn konfiguraatiotiedoston. Tiedostossa määritellään Flywayn asetukset, muun muassa versioitavien tietokantojen osoitteet, käyttäjätunnukset, salasanat, ajurit ja skeemojen hallintaan liittyviä valintoja. Konfiguraatiotiedoston ansiosta komentoja ajettaessa asetuksia ei tarvitse määritellä jokaiseen komentoon erikseen. Hakemistoon drivers sisällytetään tietokantojen JDBC -ajurit. Mikäli migraatiot kirjoitetaan Javalla, ne säilötään hakemistoon jars. Jre sisältää Java Runtime Environmentin, jota käytetään Javalla tehtyjen migraatioiden suorittamiseen. Hakemistossa lib sijaitsevat Flywayn omat kirjastot. Komentoriviversiossa on kirjastot Flywayn peruskirjastolle (engl. core) ja komentoriville. Mikäli halutaan käyttää Antia tai Mavenia komentorivin sijaan tai rinnalla, pitää niiden omat kirjastot ladata ja sijoittaa ne lib -hakemistoon. SQL -kielellä kirjoitetut migraatiot sijoitetaan hakemistoon sql.

Muutosten hallinta

Flyway ei käytä yhtä muutoslokia kuten Liquibase, vaan jokaiselle muutokselle luodaan oma sql -tiedostonsa numero- tai aikajärjestyksessä. Luodaan sama kysely kuin Liquibasessa, eli order details tauluun luodaan sarake notes. Kuviossa seitsemän on

kuvattu onnistunut migraatio. Jokainen onnistunut migraatio kirjataan riviksi Flywayn tauluun schema_version. Kuviossa seitsemän esitellään migraation vieminen kantaan käyttäen komentoriviä.



Kuvio 7. Esimerkki onnistuneesta migraatiosta.

Kyselyt voidaan toteuttaa joko Javalla tai SQL -kielellä. Migraatiot voi kirjoittaa Javalla ilman erillistä Java-projektia Flywayn mukana tulevan JRE:n ansiosta. Javalla luodut migraatiot ovat tavallisia Java-luokkia, jotka toteuttavat (engl. implement) JdbcMigration -luokan. Java yhdessä tehokkaan ORM-ratkaisu Hibernaten kanssa voi helpottaa monimutkaisten SQL-kyselyiden tekemistä varsinkin valvontapalvelun kaltaisessa JavaEE-sovelluksessa. Flyway tukee molempien ratkaisujen sekoitusta, jossa osa migraatioista voidaan kirjoittaa Javalla ja osa SQL:lla.

Flyway ei tue manuaalista tai tarvittaessa aktivoituvaa takaisinkiertoa suoraan. Jokainen migraatio ajetaan omana transaktionaan, ja mikäli transaktio epäonnistuu, se palautetaan tilaan ennen transaktiota. Takaisinkierron puuttumisen voi siis kiertää suorittamalla ainoastaan yhden rakennemuutoksen per migraatio, mutta tämä lähestymistapa on raskas ja epäkäytännöllinen, eikä tue manuaalista takaisinkiertoa. Flywayn puutteellinen takaisinkierto onkin sovelluksen suurin heikkous.

Konfiguroitavuus

Kuten asennuksen yhteydessä todettiin, konfiguraatiolle on oma tiedostonsa flyway.conf. Jokaiselle kannalle luodaan oma konfiguraatiotiedostonsa. Komentorivillä komennolla -configFile voidaan määritellä käytettävä konfiguraatiotiedosto.

Ylläpito asennuksen jälkeen

Flywayta ei tarvitse ylläpitää asennuksen tai käytön jälkeen. Uuden sovellusversion voi asentaa käytännössä korvaamalla vanhat tiedostot uuden version mukana tulevilla tiedostoilla.

6.4 Asennettavan sovelluksen valinta

Sovelluksille annetaan jokaiselle kategorialle pisteitä yhdestä kolmeen, kolmen pisteen ollessa paras mahdollinen tulos. Muutosten hallintaa ja usean kannan muutoksia painotetaan, tuplaten niistä saadut pisteet, eli niissä pisteitä annetaan kolmesta kuuteen. Enemmän yhteispisteitä saanut sovellus valitaan asennettavaksi valvontapalveluun. Pisteet on koottu luvun loppuun taulukkoon kolme.

Asennus

Asennus oli molemmissa sovelluksissa hyvin tasavertaista. Molemmat sovellukset asennettiin saman kaavan mukaisesti purkamalla latauspaketti hakemistoon ja määrittelemällä tietokannan asetukset konfiguraatiotiedostoon. Liquibasessa piti ajaa skripti, joka asensi koko sovelluksen kerralla. Flyway vaati ajurin omaan drivers -hakemistoon. Myös baseline piti luoda Flywaylla erikseen, toisin kuin Liquibasella, joka luo baselinen automaattisesti ensimmäisen migraation yhteydessä.

Muutosten hallinta

Sovellukset käsittelevät migraatioita omilla tavoillaan. Liquibase käyttää muutoslokeja, Flyway SQL-tiedostoja tai Java-luokkia. Muutoslokeihin saa useampia muutoksia, jolloin hakemisto ei täyty niin monesta tiedostosta kuin Flywayta käytettäessä. Toisaalta Flywayn mahdollisuus useampaan formaattiin ja mahdollisuus eri formaattien yhteiskäyttöön on hyödyllinen ominaisuus. Flywayn etuna on yksinkertaisuus, sillä Liquibasen muutoslokin syntaksi lukuisine tageineen on hankalaselkoista Flywayn perinteisiin SQL-kyselyihin tai Java-luokkiin verrattuna.

Toinen suuri eroavaisuus Liquibasen ja Flywayn välillä on takaisinkierron toiminnallisuus. Flywayn tuki takaisinkierrolle on puutteellinen verrattuna Liquibaseen. Yksi valitun sovelluksen vaatimuksista on mahdollisuus palauttaa tietokanta takaisin edel-

liseen versioonsa, mikäli bugeja havaitaan. Täten Liquibase on parempi vaihtoehto valvontapalvelun näkökulmasta.

Usean kannan muutokset

Molemmat ratkaisut kykenevät tekemään muutoksia useaan kantaan. Flywayssa ei ole sisäänrakennettua toiminnallisuutta usean tietokannan vertailuun. Liquibase pystyy luomaan muutoslokin kahden tietokannan rakennemuutoksista, jonka voi suorittaa manuaalisesti komentoriviltä tai automatisoida Mavenilla tai Antilla.

Konfiguroitavuus

Sovellusten konfigurointi oli helppoa, sillä molemmat ratkaisut hyödynsivät omia konfiguraatiotiedostojaan. Flywayn konfigurointi on helpompaa, sillä konfiguraatiotiedostoja ei tarvitse siirrellä manuaalisesti, vaan komentoriviltä voidaan määritellä halutun ympäristön asetukset komentoja suoritettaessa.

Vertailun tulokset

Liquibase sai huomattavasti enemmän pisteitä, joten se tulee toimimaan valvontapalvelun tietokantojen muutosten hallinnan ratkaisuna. Se voitti kaikki kategoria lukuun ottamatta konfiguroitavuutta. Liquibasen valtteina on mahdollisuus takaisinkiertoon sekä tuki usean kannan vertailuun. Flywayn etuina on yksinkertaisuus ja nopea konfigurointi, mutta se häviää kaikissa muissa kategorioissa. Taulukossa kolme on kuvattu sovellusten pisteytys.

Taulukko 3. Liquibasen ja Flywayn vertailu.

	Liquibase	Flyway
Asennus	3	2
Muutosten hallinta	6	4
Usean kannan muutokset	6	2
Konfiguroitavuus	2	3
Yhteensä	17	11

7 Valitun sovelluksen asennus valvontapalveluun

Liquibase asennetaan ja konfiguroidaan aluksi valvontapalvelun kehitysympäristöön. Kun sovelluksen toimivuus on varmistettu kehitysympäristössä, tehdään tarvittavat asetukset myös testaus- ja tuotantoympäristöihin. Ennen asennusta otetaan varmuuskopiot jo olemassa olevista tietokannoista, jotta virhetilanteessa tietokannan rakenne ei vioitu tai data häviä pysyvästi.

Liquibasen asennus aloitetaan lataamalla sovelluksen uusin versio Liquibasen nettisivuilta. Kirjoitushetkellä Liquibasen uusin versio on 13.10.2016 julkaistu v3.5.3. Ladatun tiedoston sisältö puretaan haluttuun hakemistoon. Onnistuneen purkamisen jälkeen luodaan liquibase.jar -tiedosto ajamalla skripti liquibase.bat. Juuri luotu liquibase.jar -tiedosto kopioidaan Antin kirjastohakemistoon. Ennen muutoslokin generointia kopioidaan JDBC-ajuri Liquibasen /lib -hakemistoon ja luodaan kehitysympäristölle konfiguraatiotiedosto liquibase.properties.

Muutoslokin luonti tuottaa xml-tiedoston, joka pitää sisällään tietokannan rakenteen, mutta ei rivejä. Tästä versiosta luodaan ensimmäinen tietokannan versiotagi muutoslokiin, jonka avulla voidaan takaisinkiertää alkuperäiseen tietokantaversioon, jos tarve niin vaatii. Liquibase Bestpractices (n.d) kehoittaa luomaan useita muutoslokeja julkaisuversioiden mukaan, jolloin muutosten hallinta selkeytyy (Liquibase Best Practices n.d). Muutoslokit sijoitetaan projektihakemistoon, josta ne lisätään valvontapalvelun säilöön Gitiin.

Asennuksen onnistumisen varmistaminen suoritetaan tekemällä ensimmäinen migraatio kehityskantaan. Ensimmäinen migraatio sisältää vain yhden uuden taulun. Onnistuneen migraation jälkeen testataan takaisinkierron toimivuus palauttamalla tietokanta alkuperäiseen versioon käyttämällä alussa luotua tietokannan versiotagia. Ensimmäisen onnistuneen migraation yhteydessä Liquibase luo tarvitsemansa taulut tietokantaan, viimeistellen asennus- ja konfiguraatiovaiheen. Kuviossa kahdeksan on kuvattu Liquibasen muutosten hallintataulu ennen ja jälkeen takaisinkierron.

ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECU...	EXECTYPE	MD5SUM	DESCRIPTION	COMMENTS	TAG
original	salovaas	[redacted]	2016-11-29 10:5...	1	EXECUTED	7:96d32d5f6007...	tagDatabase		original
1	salovaas		2016-11-29 10:5...	2	EXECUTED	7:bf61d047a103...	createTable tab...		NULL
0.1	salovaas		2016-11-29 10:5...	3	EXECUTED	7:3259e72857ad...	tagDatabase		0.1

ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECU...	EXECTYPE	MD5SUM	DESCRIPTION	COMMENTS	TAG
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Kuvio 8. Liquibasen muutosten hallintataulu ennen ja jälkeen takaisinkierron alkuperäiseen tietokantaan.

Kun Liquibasen toimivuus on testattu, luodaan konfiguraatiotiedostot myös testaus- sekä tuotantotietokannoille. Näiden tiedostojen sisältö on rakenteeltaan sama, ainoastaan parametrien arvot eroavat toisistaan. Käytettäessä muita kuin kehityskannan konfiguraatiotiedostoja täytyy komentoa ajettaessa erikseen määritellä haluttu konfiguraatiotiedosto. Jos konfiguraatiotiedostoa ei määritellä, Liquibase olettaa, että muutokset ajetaan kehityskantaan.

Ant -asennus

Asennus valvontapalveluun eroaa hieman aiemmin asennetusta virtuaalipalvelimesta, sillä komentorivin lisäksi Liquibase asennetaan toimimaan myös Ant -kokoonpanotyökalulla. Liquibasen jar-tiedosto pitää sijoittaa Antin kirjastojen kanssa samaan hakemistoon tai vaihtoehtoisesti valvontapalvelun luokkapolulle. Valvontapalvelun tapauksessa jar-tiedosto sijoitetaan valvontapalvelun luokkapolulle, jolloin polku pysyy relatiivisena, ja .jar -tiedosto siirtyy samalla Gitin versionhallinnan piiriin.

Liquibase määritellään Antin build.xml –konfiguraatiotiedostoon omana tehtävänään. Kuviossa yhdeksän on kuvattu Liquibasen määrittely build.xml –tiedostossa.

```
<project name="[redacted]" basedir="." default="dist" xmlns:liquibase="antlib:liquibase.integration.ant">

    <!-- Liquibase task-->
    <taskdef resource="liquibase/integration/ant/antlib.xml" uri="antlib:liquibase.integration.ant">
        <classpath path="lib/liquibase.jar"/>
    </taskdef>
```

Kuvio 9. Liquibasen määrittely build.xml tiedostoon.

8 Liquibasen ja manuaalisen muutosten hallinnan vertailu

Tässä luvussa vertaillaan Liquibasen avulla toteutettua muutosten hallintaa ja manuaalista muutosten hallintaa käytännössä valvontapalvelun tavallisia käyttötapauksia mukaillen. Metodien vertailua varten toteutetaan kolme testiä. Tietoturvasyistä ei kerrota tarkkoja SQL-lauseita eikä havainnollisteta valvontapalvelun tietokantojen rakennetta lainkaan. Ensimmäisessä testissä tehdään uusi taulu sarakkeineen kehityskantaan, jonka jälkeen sama taulu siirretään myös testauskantaan. Toisessa testissä vertaillaan kehitys- ja testausympäristöjen rakenteiden eroja. Kolmannessa testissä palautetaan kehityskanta ja testauskanta eri versioista aiempaan versioon.

Testeissä kaikki tarvittavat sovellukset ovat valmiiksi auki, joten ajanottoon ei lasketa mukaan sovellusten käynnistämiseen tai kirjautumiseen kuluvaa aikaa. Liquibase on käyttövalmiina komentorivillä ja SQL Server Management Studiossa on kirjauduttu sisään sekä kehitys- että testauskantaan.

Taulun luonti

Ensimmäinen käyttötapaus on taulun luonti kehityskantaan, joka kopioidaan myös testauskantaan. Käyttötapaus on valvontapalvelun tapauksessa hyvin yleinen.

Liquibasen avulla luodaan muutoslokiin uusi muutossetti, johon lisätään uuden taulun luontilause. Kun muutokset lokiin ovat valmiit, se voidaan ajaa komennolla liquibase migrate, jolloin muutoslokin sisältö, joka on tässä tapauksessa vain yksi taulu, lisätään tietokantaan. Saman muutoslokin voi ajaa myös testaustietokantaan vaihtamalla konfiguraatiotiedostoa ja ajamalla saman komennon, jolla luodaan identtinen taulu testauskantaan. Aikaa kului yhteensä 3 minuuttia ja 4 sekuntia.

Manuaalinen taulun lisääminen testataan sekä SQL Server Management Studion kautta sekä SQL-lauseella. Manuaalisesti taulujen lisääminen on hyvin vikkelää SQL Server Management Studion avulla. Sitä käyttämällä aikaa kului vain 24 sekuntia. SQL-kyselyn kirjoittamiseen ja molempiin kantoihin ajamiseen aikaa kului 55 sekuntia. Yksinkertaisten taulujen luonti ilman indeksointia tai liipasimia on nopeampaa tehdä manuaalisesti kahdelle tietokannalle.

Kahden tietokannan vertailu

Kehitys- ja testauskannan eroavaisuuksia joudutaan usein tarkastelemaan, mikäli testiversiosta löytyy bugeja, jolloin se täytyy palauttaa takaisin kehitysversioon. Toisessa käyttötapauksessa kehityskannan versio on 1.2 ja testauskannan versio on 1.1. Tarkoituksena on vertailla kantoja keskenään, ja luoda muutosloki tai SQL –skripti, jolla testauskannan voi päivittää versioon 1.2.

Liquibaselle luodaan uusi konfiguraatiotiedosto, johon kirjataan verrattavana olevan tietokannan sekä vertailevan tietokannan tiedot. Valvontapalvelun tarpeisiin tälläiset yhdistetyt konfiguraatiotiedostot luodaan kehitys- ja testauskannalle, sekä testaus- ja tuotantokannalle.

Kun konfiguraatiotiedostot on tehty, komennolla liquibase diffChangeLog luodaan muutosloki, jonka ajaminen vertailtavaan kantaan synkronoi ensimmäisenä määritellyn kannan toisena määritellyn verrattavan kannan mukaan. Liquibasella aikaa meni kahden kannan yhdistetyn konfiguraatiotiedoston luominen mukaan luettuna 1 minuutti 45 sekuntia, ja konfiguraatiotiedoston jo olemassa ollessa aikaa kului kokonaisuudessaan 8 sekuntia.

Ilman Liquibasea käydään manuaalisesti läpi jokainen taulu, sarake sekä indeksi molemmista kannoista eroavaisuuksia etsien. Tämä on aikaavievää sekä virhealtista toimintaa, sillä joitain muutoksia voi jäädä huomaamatta. Aikaa manuaaliseen vertailuun kului 16 minuuttia. Mitä suurempi ja monimutkaisempi tietokanta on, sitä enemmän aikaa sen manuaaliseen läpikäyntiin kuluu suhteessa Liquibaseen. Eroavaisuudet täytyy luoda käsin tietokantakyselyiksi. Tähän aikaa käytettiin vielä 11 minuuttia lisää, jolloin manuaaliseen vertailuun kului yhteensä 27 minuuttia.

Tietokantojen palautus aiempaan versioon

Viimeisessä käyttötapauksessa testaus- ja kehittäjäkanta on päivitetty Liquibasen migraatioita käyttäen. Testauskannan versio on 1.1 ja kehityskannan versio on 1.2. Versiot 1.1 ja 1.2 ovat lähes identtisiä, mutta muutamat pienet eroavaisuudet erottavat ne toisistaan. Molemmissa versioissa on lisätty ja poistettu muutamia sarakkeita sekä indeksejä. Tarkoituksena on palauttaa molemmat kannat takaisin alkuperäiseen

versioon 1.0. Manuaalisesti toteutettavassa vertailussa käytetään hyväksi erillistä dokumenttia, jossa listataan tietokantoihin tehdyt muutokset versioittain.

Liquibasella takaisinkierto onnistuu versiotagilla varustettuihin migraatoihin. Komennolla liquibase rollback 1.0 voidaan muuttaa kehitystietokannan rakenne takaisin versioon 1.0. Testikannan palautus onnistuu vaihtamalla kehityksen konfiguraatiotiedoston testauksen vastaavaan ja ajamalla saman komennon. Aikaa Liquibasen avulla tähän tehtävään kului 36 sekuntia.

Ilman erillistä dokumentointia, varmuuskopiota tai SQL -skriptiä on käytännössä mahdoton tietää, mitä tietty versio sisälsi, sillä SQL Server 2008 ei sisällä muutoksia jäljittävää työkalua. Valvontapalvelun dokumentaatiossa on kuitenkin listattu eri versioiden muutoksia, joten manuaalisesti muutosten tekeminen on yksinkertainen prosessi. Molemmat kannat pitää palauttaa aiempaan versioon yksi kerrallaan, jolloin aikaa yhteensä kului 8 minuuttia.

9 Tutkimuksen johtopäätökset

Tietokantojen versiointiin ja muutosten hallintaan on runsaasti erilaisia ratkaisuja, jotka soveltuvat erilaisiin tarpeisiin. Pienille projekteille voi riittää kevyt skriptipohjainen ratkaisu, mutta isommat sovellukset hyötyvät laajoista ja kokonaisvaltaisista ratkaisuista.

Tutkimuksen ensimmäiseen vaiheeseen valittiin viisi hajautettuun versionhallintaan pohjautuvaa sovellusta. Otantaan valitut sovellukset olivat valvontapalvelun kannalta sopivia, sillä pelkkiin skripteihin pohjautuvan ratkaisun käyttö ei olisi ollut järkevää useita tietokantaympäristöjä sisältävän valvontapalvelun kannalta. Sovellukset olivat kuitenkin tarpeeksi erilaisia keskenään, mikä mahdollisti niiden järkevän vertailun ja pisteytyksen.

Tutkimuksen ensimmäisen vaiheen pohjalta valikoitui kaksi hyvää vaihtoehtoa valvontapalvelun muutosten hallintaratkaisuksi. Liquibase ja Flyway ovat molemmat käyttökelpoisia ratkaisuja, mutta Flywayn puutteet takaisinkierrossa sekä tietokantojen vertailussa ratkaisivat vertailun Liquibasen hyväksi. Liquibasen asennus sujui hyvin samankaltaisesti sekä virtuaalipalvelimella että valvontapalvelussa, sillä ympäris-

töt virtuaaliympäristöt oli luotu mahdollisimman samankaltaisiksi kuin valvontapalvelun tuotantoympäristö.

Asennuksen ja konfiguraation jälkeen suoritettiin kolme testiä, joiden avulla vertailtiin muutosten hallintaa Liquibasen sekä vanhan manuaalisen tavan välillä. Liquibase suoriutui useimmista testeistä nopeammin kuin manuaalisesti tehtynä. Ainoastaan ensimmäisen testin yksinkertaisen taulun lisääminen ilman indeksejä tai liipaisimia on nopeampaa tehdä käsin. Liquibasen tehokas käyttö voi nopeuttaa suoritusaikoja entisestään. Taulukkoon neljä on koottu testien tulokset.

Taulukko 4. Testauksen tulokset.

	Liquibase	Manuaalinen
Taulun luonti	3 minuuttia 4 sekuntia.	SQL Server Management Studio: 24 sekuntia. SQL-kysely: 55 sekuntia.
Kahden tietokannan vertailu	Konfiguraatiotiedoston luomisen kanssa: 1 minuutti 45 sekuntia. Konfiguraatiotiedoston jo olemassa ollessa: 8 sekuntia.	27 minuuttia
Tietokantojen palautus aiempaan versioon	36 sekuntia	8 minuuttia

Tuloksista voidaan päätellä, että Liquibasen käyttö parantaa tietokantojen hallinnan tehokkuutta ja vähentää muutoksiin käytettävää aikaa. Ennen Liquibasen asentamista muutosten hallintaan käytettiin ainoastaan SQL Server Management Studiota ja SQL-skriptejä. Liquibase ei kuitenkaan korvaa niitä kokonaan. SQL Server Management Studiolla voidaan tarvittaessa luoda SQL-skriptejä, jotka voidaan sellaisenaan kopioida Liquibasen muutoslokiin, jolloin saadaan hyödynnettyä molempien sovellusten vahvuudet.

Tulosten perusteella voidaan todeta, että Liquibase on oikea ratkaisu valvontapalvelun tietokantojen muutosten hallintaan. Valmiiden muutoslokien ja konfiguraatiotiedostojen ansioista uusien kehittäjienkin on helppo aloittaa Liquibasen käyttö lataamalla tarvittavat tiedostot suoraan valvontapalvelun Gitistä.

Ratkaisu tutkimusongelmaan

Tutkimusongelmaan haettiin ratkaisua etsimällä keinoja tehostaa valvontapalvelun muutosten hallintaa. Ongelma kiteytettiin kolmeen tutkimuskysymykseen, joihin tutkimuksessa pyrittiin vastaamaan. Ensimmäisessä tutkimuskysymyksessä tutkittiin keinoja parantaa tietokantojen muutosten hallintaa. Tulokseksi saatiin tietopaketti, jossa kerrotaan relaatiotietokannoista, versionhallinnasta sekä mahdollisuuksista relaatiotietokantojen muutosten hallintaan. Toisessa tutkimuskysymyksessä valittiin paras sovellus valvontapalvelun tarpeisiin. Vertailemalla viittä sovellusta päästiin lopputulokseen, jossa Liquibase valittiin asennettavaksi valvontapalvelulle. Kolmanteen kysymykseen vastattiin asentamalla Liquibase onnistuneesti valvontapalveluun ja vertailemalla sen toimintavarmuutta ja käyttönopeutta manuaaliseen muutosten hallintaan verrattuna. Tämä validointi vahvisti Liquibasen olleen oikea ratkaisu valvontapalvelulle, sillä se nopeutti muutosten ajamista useisiin eri tietokantaympäristöihin ja vähensi samalla inhimillisten virheiden määrää. Tutkimusongelmaan saatiin vastaus Liquibasen asennuksen, konfiguraation sekä validoinnin myötä.

10 Pohdinta

Kirjallisuutta ja verkkoartikkeleita löytyy runsaasti relaatiotietokannoista sekä versionhallinnasta. Relaatiotietokantojen versionhallinnasta sen sijaan ei löydy läheskään yhtä paljoa luotettavaa tietoa. Sen takia tietoperustan kirjoittamista piti lähestyä epäsuorasti sekä relaatiotietokantojen että versionhallinnan näkökulmista ja yhdistää materiaali lopulta mahdollisimman johdonmukaiseksi kokonaisuudeksi.

Rakenteen pääpiirteet muodostuivat lähestulkoon lopulliseen muotoonsa jo esitutkimuksen aikana. Työn toteuttaminen valmiiksi mietityn rakenteen pohjalta oli selkeää ja helposti aikataulutettavaa. Ajallisesti Liquibasen asentamista valvontapalveluun oli vaikea arvioida, sillä asennuksen yhteydessä ilmeneviä mahdollisia komplikaatioita

on hyvin vaikea ennustaa etukäteen. Lopulta mitään ongelmia ei asennuksen ja konfiguraation aikana ilmaantunut.

Työn tuottamisen kannalta tutkimusmenetelmäksi valikoitunut kehittämistutkimus oli paras vaihtoehto tämänkaltaisen tutkimuksen toteuttamiseen, sillä pääosassa oli tietokantojen muutosten hallinnan kehittäminen. Valvontapalvelulle soveltuvimman sovelluksen valinnassa käytettiin kvalitatiivista tutkimusotetta, joka ilmeni kriteeristön empiirisenä mittauksena virtuaalipalvelimilla, jotka mukailivat mahdollisimman tarkasti valvontapalvelun tuotantoympäristöä. Onnistuneen menetelmävalinnan ansiosta tutkimuksella oli vahva pohja, jonka myötä tutkimus saatiin valmiiksi aikataulussa.

Tutkimus toteutettiin Landis+Gyrin valvontapalvelun vaatimusten mukaan. Tuloksia voidaan kuitenkin hyödyntää myös muissa projekteissa, joissa tietokantojen muutosten hallinta on puutteellista, sillä tutkimustulokset ovat varsin yleisluontoisia. Asennetun ratkaisun kolme testattua käyttötapausta ovat hyvin perinteisiä tietokantaoperaatioita yhden tai useamman tietokannan sovelluksissa.

Valvontapalvelulle asennettu ja konfiguroitu Liquibase on kattava ja monipuolinen tietokantojen versiointisovellus, jota kehitetään ja päivitetään ahkerasti. Kuitenkaan Liquibase ei välttämättä ole paras vaihtoehto jokaiselle projektille. Sen takia tutkimuksessa pyrittiin tuomaan esiin myös muiden sovellusten vahvuuksia ja heikkouksia, jotta tutkimustulokset olisivat sovellettavissa myös muihin projekteihin Landis+Gyrin sisällä sekä ulkopuolella.

Jatkokehitysmahdollisuuksia on lukuisia, joihin tämä tutkimus antaa hyvän pojan. Esimerkiksi tietokantojen muutosten hallinnan automatisointia ja jatkuvan integraation hyödyntämistä ei tutkittu tässä työssä juuri lainkaan. Useat tässä tutkimuksessa käsitellyt sovellukset tukevat automatisointia, joten aihetta syvälliseenkin tutkimukseen on olemassa. Tutkimuksessa käytiin läpi vain viittä yleistä tietokantojen muutosten hallintaan kehitettyä ratkaisua, joista kaikki perustuivat hajautettuun versionhallintaan. Skriptipohjaisia sekä keskitetyn versionhallinnan ratkaisuja on myös tarjolla, joihin paneutumalla saisi uusia näkökulmia vaikkapa jonkun toisen tietokantajärjestelmän vaatimusten kautta.

Lähteet

Allen, M. N.d. Relational Databases Are Not Designed To Handle Change. Viitattu 10.12.2016. http://www.marklogic.com/blog/relational-databases-change/.

Allen, S. 2008. Three Rules for Database Work. Viitattu 1.11.2016. http://odetocode.com/blogs/scott/archive/2008/01/30/three-rules-for-database-work.aspx.

Ambler, S. 2006. Mapping Objects to Relational Databases: O/R Mapping In Detail. Viitattu 1.11.2016. http://www.agiledata.org/essays/mappingObjects.html.

Ambler, S. N.d. Relational Databases 101: Looking at the Whole Picture. Viitattu 11.12.2016. http://www.agiledata.org/essays/relationalDatabases.html

A Relational Database Overview. N.d. Oracle. Viitattu 8.12.2016. https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html.

Azarian, I. 2013. A Review of Software Version Control: Systems, Benefits, and Why it Matters. Viitattu 10.12.2016. http://www.seguetech.com/a-review-of-software-version-control-systems-benefits-and-why-it-matters/.

Boag, P. 2008. To Version Control or Not? Viitattu 10.12.2016. https://boagworld.com/dev/to-version-control-or-not/.

Database version control, made easy! N.d. Sovelluksen dbv esittely. Viitattu 21.11.2016. https://dbv.vizuina.com/.

DB-Engines Ranking. 2016. Kuukausiranking DB-engines.com sivulla. Viitattu 14.11.2016. http://db-engines.com/en/ranking.

Documentation. N.d. Sovelluksen Flyway dokumentaatio. Viitattu 19.11.2016. https://flywaydb.org/documentation/.

Ernst, M. 2012. Version control concepts and best practices. Viitattu 14.11.2016. https://homes.cs.washington.edu/~mernst/advice/version-control.html.

Evolve your Database Schema easily and reliably across all your instances. N.d. Sovelluksen Flyway esittely. Viitattu 19.11.2016. https://flywaydb.org/.

Fritchey, G. 2015. Top 10 Most Common Database Scripts. Viitattu 1.11.2016. https://www.simple-talk.com/sql/t-sql-programming/top-10-most-common-database-scripts/.

Generating DDL scripts. N.d. IBM. Viitattu 3.11.2016. http://www.ibm.com/support/knowledgecenter/SS9UM9_7.5.3/com.ibm.datatools.fe.ui.doc/topics/cddl.html.

Järjestelmän valvontapalvelu Gridstream-ratkaisulle. 2015. Tuote-esittely Landis+Gyr Oy:n www-sivuilla. Viitattu 14.10.2016. http://www.landisgyr.fi/webfoo/wp-content/uploads/2012/12/Active-Monitoring-Service-for-Gridstream_fi_final-9.11.2015.pdf.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Lionetti, G. 2012. What is Version Control: Centralized vs. DVCS. Viitattu 8.11.2016. http://blogs.atlassian.com/2012/02/version-control-centralized-dvcs/.

Liquibase Best Practices. N.d. Ohjeistus Liquibasen oikeaoppiseen käyttöön. Viitattu 28.11.2016. http://www.liquibase.org/bestpractices.html.

Liquibase Documentation. N.d. Sovelluksen Liquibase dokumentaatio. Viitattu 21.11.2016. http://www.liquibase.org/documentation/index.html.

Luotola, J. Eurooppalaiset uusivat 200 miljoonaa mittaria – Kymmeniä työpaikkoja Suomeen. 2016. Viitattu 28.11.2016. http://www.tekniikkatalous.fi/tekniikka/energia/eurooppalaiset-uusivat-200-miljoonaa-mittaria-kymmenia-tyopaikkoja-suomeen-6531495.

Mullins, C. 2012. Database Administration: The Complete Guide to DBA Practices and Procedures. Boston: Addison-Wesley.

Nimkar, N. 2016a. Database Version Control Tools. Viitattu 7.11.2016. http://www.releasemanagement.org/2016/02/database-version-control-tools/.

Nimkar, N. 2016b. Database Versioning: The ignored aspect of version control. Viitattu 7.11.2016. http://www.releasemanagement.org/2016/01/database-versioning-the-ignored-aspect-of-version-control/.

Northwind database. N.d. Kuvaus Northwind-tietokantaprojektista. Viitattu 14.11.2016. https://northwinddatabase.codeplex.com/.

Preiße, R & Stachmann, B. 2014. Git: Distributed Version Control--Fundamentals and Workflows. Quebec: Brainy Software Inc.

Rees, B. 2014. How Mature is Your Database Change Management Process? Viitattu 10.12.2016. https://www.simple-talk.com/blogs/how-mature-is-your-database-change-management-process/.

Ritchie, C. 2002. Relational Database Principles. Lontoo: Thomson Learning.

Schema Zen - Script and create SQL Server objects quickly. N.d. Sovelluksen Schema Zen esittely. Viitattu 20.11.2016. https://github.com/sethreno/schemazen.

Schumacher, R. 2005. High Performance SQL Server DBA: Tuning & Optimization Secrets. Kittrell: Rampant TechPress.

Sheldon, R., Richardson, R. & Davis, T. 2014. SQL Server Source Control Basics. Cambridge: Red gate books.

Source control for your database. N.d. Sovelluksen Liquibase esittely. Viitattu 19.11.2016. https://www.liquibase.org.

Sqitch. N.d. Sovelluksen Sqitch esittely. Viitattu 20.11.2016. http://sqitch.org/.

Sqitch tutorial for MS SQL. 2014. Sovelluksen Sqitch dokumentaatio. Viitattu 16.11.2016. https://github.com/drmuey/sqitch/blob/master/lib/sqitchtutorial-mssql.pod.

Sumathi, S. & Esakkirajan, S. 2007. Fundamentals of Relational Database Management Systems. Berliini: Springer Science & Business Media.

Verveer, H. 2011. Database Version Control. Viitattu 1.11.2016. http://techportal.inviqa.com/2011/01/11/database-version-control/.

Why Use an ORM and an Abstraction Layer? N.d. Artikkeli Librosweben www-sivuilla. Viitattu 8.12.2016. http://librosweb.es/libro/symfony_1_2_en/chapter_8/why_use_an_orm_and_an_abstraction_layer.html.

Liitteet

Liite 1. Testitietokannan rakenne. (Northwind database N.d.)