Joona Rahkonen

Tietokannan versionhallinta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

2.9.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Joona Rahkonen Tietokannan versionhallinta 42 sivua + 2 liitettä 2.9.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Lehtori Vesa Ollikainen Software Specialist Lauri Lehtinen

Insinöörityössä selvitettiin Digia Financial Solutions -organisaation tietokannan versionhallinnan käytänteitä ja toimintatapoja sekä etsittiin organisaation käyttöön sopiva tietokannan versionhallintaohjelmisto, jonka avulla tietokannan muutosten hallinta ja muutosten toimittaminen asiakkaalle onnistuu mahdollisimman vaivattomasti.

Teoriaosuudessa selvitettiin yleisesti versionhallinnan historiaa. Sen jälkeen siirryttiin kohti nykyaikaa, mikä käsitti paikallisen, keskitetyn ja jaetun versionhallinnan. Lisäksi käsiteltiin tietokannan versionhallinnan käytänteitä ja periaatteita.

Työosuudessa vertailtiin tietokannan versionhallintaohjelmistoja keskenään niiden ominaisuuksien ja tilaajan tarpeisiin soveltuvuuden mukaan. Alustavan versionhallintaohjelmistojen ominaisuuksien selvittämisen perusteella tarkempaan vertailuun päätyivät avoimen lähdekoodin ratkaisut Flyway ja Liquibase. Näistä kahdesta ohjelmistosta valittiin lopulta Liquibase asennettavaksi ja testattavaksi sen laajempien ominaisuuksien ansiosta. Liquibase asennettiin testiympäristöön ja integroitiin se toimimaan yhdessä lähdekoodin versionhallinnan, Subversionin ja jatkuvan integraation ympäristön, TeamCityn kanssa. Lisäksi tietokannan päivittämiseen sekä toimittamiseen kehitettiin batch-skriptit, joiden avulla toimenpiteet onnistuvat automaattisesti.

Insinöörityön tuloksena varmistui, että Liquibase soveltuu organisaation käyttöön, mikä helpottaa tietokannan päivittämistä ja toimittamista yksinkertaisessa tilanteessa. Tuloksena syntyi myös uusi malli tietokannan versionhallintaan, jota lähdetään kehittämään edelleen, jotta sen avulla voidaan ratkaista myös kompleksit reaalimaailman ongelmat.

Avainsanat	Tietokannat, versionhallinta, Liquibase, TeamCity, jatkuva toimittaminen, jatkuva integraatio

Author Title Number of Pages Date	Joona Rahkonen Database version control 42 pages + 2 appendices 2 September 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Vesa Ollikainen, Lecturer Lauri Lehtinen, Software Specialist

The purpose of this thesis was to examine Digia Financial Solutions organization's practices on database version control and to find the most suitable solution for database change management and database delivery for the organization.

In the theory part of the thesis the history of version control was researched from the beginning to the present, including local, centralised and distributed version control and the practices and principles of database version control.

In the work part of the thesis different database version control software were compared to each other by their features and how they fit to the organization's needs. Two open-source solutions, Flyway and Liquibase, were selected for more detailed comparison on basis of the preliminary research. Of these two Liquibase was selected to be installed and tested based on its more thorough features. Liquibase was installed and integrated with source control software Subversion and continuous integration environment TeamCity. The change management and database delivery SQL script creation were automated by batch scripts that run Liquibase.

Results of the thesis were that Liquibase is suitable for the usage of the organization at least in simple cases and that it helps to manage database changes and decreases the time spent on creating database delivery SQL scripts on delivery phase. Developing the database version control process to include complex cases will be continued.

Keywords	Databases, version control, Liquibase, TeamCity, continuous delivery, continuous integration

Sisällys

Lyhenteet

DiFS	Digia Financial Solutions. Voidaan tarkoittaa myös yleisesti Financial Solutions -tuoteperhettä.
CD	Continuous delivery. Jatkuvan toimittamisen malli.
SVN	Subversion. Lähdekoodin versionhallintasovellus.
SQL	Structured Query Language. Relaatiotietokantojen hallitsemiseen käytettävä kieli.
XML	Extensible Markup Language. Datan säilytykseen ja siirtämiseen tarkoitettu kuvauskieli.
TKHJ	Tietokannan hallintajärjestelmä.
ODBC	Open Database Connectivity. Standardoitu avoin rajapinta tietokannoille.
JDBC	Java Database Connectivity. Standardoitu Java-ohjelmointikielen rajapinta tietokannoille.
T-SQL	Transact SQL. Microsoft SQL Serverin käyttämä kyselykieli.
PL/SQL	Procedural Language/SQL. Oracle Databasen käyttämä kyselykieli.
STS	Securities Trading System. DiFSin tuote kaupankäyntiin.

1 Johdanto

Versionhallinta on ollut jo pitkään käytössä ohjelmistokehityksessä lähdekoodin osalta, eikä nykyään tulisi kyseeseen aloittaa minkään kokoista projektia ilman lähdekoodin versionhallintaa, mutta tietokannan versionhallinnan kohdalla näin ei välttämättä toimita. Useimmissa ohjelmistoprojekteissa tarvitaan tietokantaa tietojen tallentamiseen, ja se toimii koko ohjelmiston pohjana. Tietokannan muutosten hallinta on siitä riippumatta toteutettu lyhytkatseisesti huomioimatta tietokantaan tehtävien muutosten lisääntymistä projektin edetessä, jolloin tietokannan ja lähdekoodin pitäminen samassa versiossa manuaalisesti ei olekaan enää niin yksinkertaista. Usean tietokannan pitäminen samassa versiossa tuo lisää vaikeusastetta. Kun tähän yhtälöön lisätään vielä se, että monta kehittäjää tekee samanaikaisia muutoksia samaan tietokantaan, tulee tilanteesta erittäin haastava.

Tietokannan versionhallinta koetaan monimutkaiseksi prosessiksi, eikä siihen soveltuvia työkaluja tunneta kovin yleisesti, joten se joko jää tekemättä tai tehdään puutteellisesti, vaikka todellisuudessa tietokannan versionhallinnan kunnollinen toteutus ja kehityskulttuurin muuttaminen tietokannan huomioimiseksi säästää aikaa myöhemmissä työvaiheissa ja tuottaa siten lisäarvoa yritykselle. Erityisesti riittämätön tietokannan versionhallinta näkyy tuotteen toimitusvaiheessa, kun asiakkaalle toimitettava ohjelmisto ja tietokanta täytyy saattaa samaan, toisiinsa yhteensopivaan versioon. Tuossa vaiheessa tietokannan tilan selvittäminen on vaikeampaa kuin se voisi olla tarkemman versionhallinnan avulla.

Tämän työn aiheena on tutustua tietokannan versionhallintaan ja siihen soveltuviin ohjelmistoihin. Työn tavoitteena on löytää tilaajan tarpeisiin sopiva ratkaisu tietokannan versionhallintaan ja asentaa se toimimaan yhdessä sekä lähdekoodin versionhallinnan että jatkuvan koonnin ympäristön kanssa.

Työn tilaaja on Digia, joka on monialainen IT-alan palveluyritys Suomessa ja Ruotsissa. Digia työllistää yli 1000 osaajaa Helsingissä, Jyväskylässä, Oulussa, Raumalla, Tampereella, Turussa, Vaasassa, Lahdessa ja Tukholmassa. Digia on listattuna Nasdaq Helsingissä, ja sen liikevaihto vuonna 2017 oli 94,5 miljoonaa euroa. Digialta löytyy vahvaa toimialaosaamista erityisesti kaupan, logistiikan ja teollisuuden alalta, julkiselta sektorilta sekä pankki- ja vakuutusalalta. [1.]

Tämä työ toteutetaan Digian pankkialan osastolle nimeltä Financial Solutions (DiFS), jonka tietokannat ovat erittäin laajoja ja moninaisia, tieto on tärkeää ja toimituksia tehdään usein. Siten tietokannan versionhallinta on tärkeässä osassa ohjelmistojen kehityksessä ja sen tehostaminen on erittäin hyödyllistä. Suurimpia haasteita työssä tuleekin olemaan tuotteiden runsaan modulaarisuuden aiheuttama suuri määrä tietokantaskriptejä, jotka tulee korvata joko versionhallintaohjelmiston ominaisuuksien avulla tai valjastaa versionhallinta tukemaan näitä olemassa olevia skriptejä.

Työ on osa suurempaa kokonaisuutta, jonka tavoitteena on päästä tiukemmin jatkuvan toimittamisen malliin (continuous delivery, CD). Jatkuva toimittaminen on nykyaikainen tapa tehdä tiheitä asiakastoimituksia mahdollisimman ketterästi ja automatisoidusti, jolloin asiakas saa uusimmat kehitykset nopeasti käyttöönsä, ja manuaalisen työn tarve vähenee. Näin saadaan laskettua inhimillisten virheiden määrää, lisättyä toimitusvarmuutta ja nostettua asiakastyytyväisyyttä.

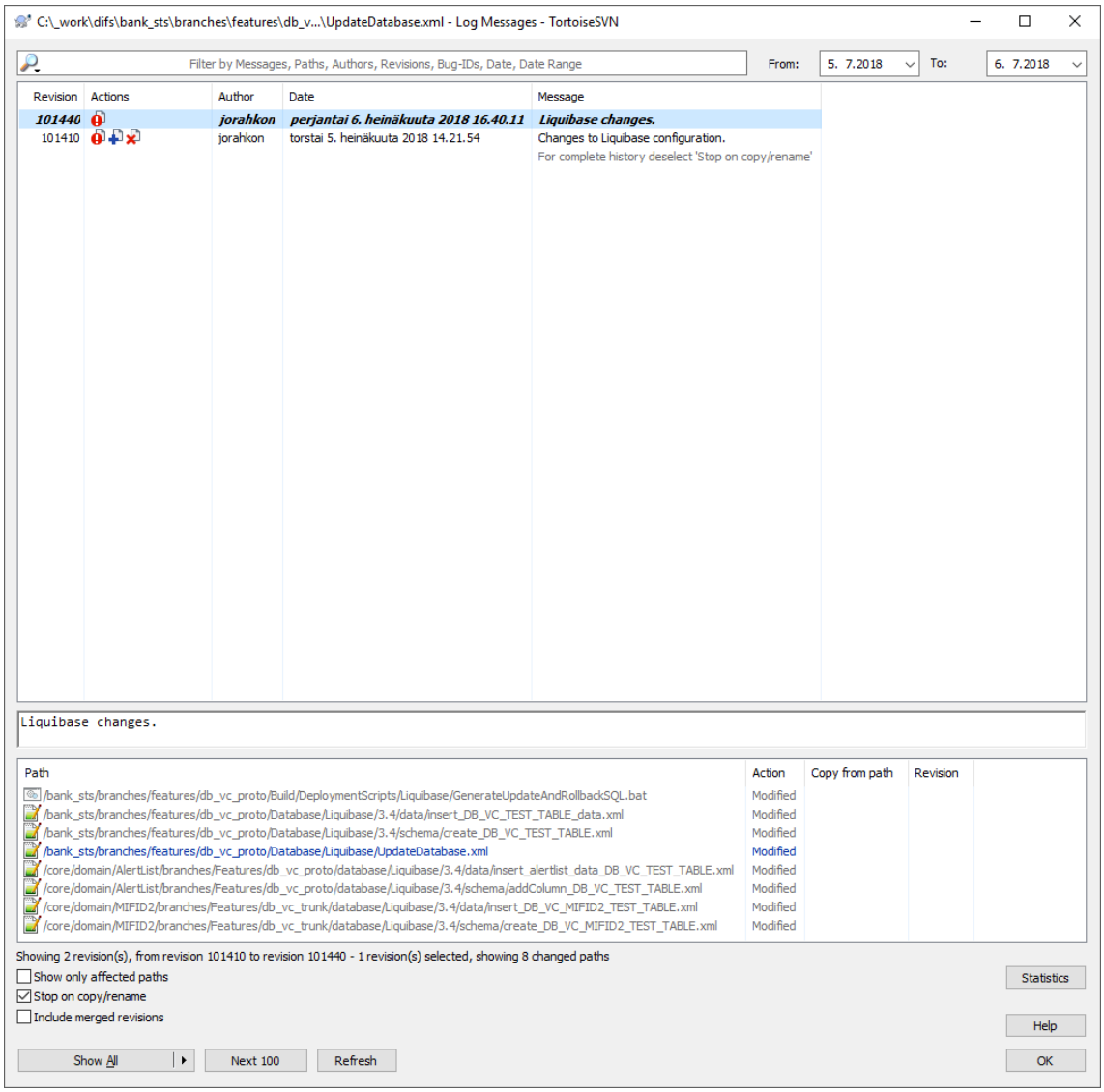
2 Versionhallinta ketterässä ohjelmistokehityksessä

2.1 Versionhallinta yleisesti

Versionhallinta tarkoittaa järjestelmää, joka muistaa tiedostoihin tehdyt muutokset ja mahdollistaa tiettyyn versioon palaamisen. Versionhallinta siis valvoo tiedostoja ja tekee jokaisesta muutoksesta erillisen merkinnän, jotta tuohon tilaan on mahdollista palata. Versionhallintasovellus luo paikallisen hakemiston samaan paikkaan, jossa tiedosto sijaitsee ja käyttää luotua hakemistoa tallentaakseen tiedoston tarkan historian. [2, s. 30.] Versionhallinnan avulla voidaan myös palauttaa poistettuja tiedostoja tai hakemistoja peruuttamalla revision muutokset. Yhdessä nämä ominaisuudet antavat kehittäjälle mielenrauhaa, sillä jos jokin menee vikaan ja omat muutokset rikkovat toimivia ominaisuuksia, voidaan edelliseen versioon palata yksinkertaisilla toimenpiteillä. Kehittäjät voivat keskittyä tekemään työtään ja kokeilla suuriakin muutoksia ilman pelkoa peruuttamattomista seurauksista. [3, s. 34-35.]

Pelkän historian tallentaminen ei vielä riitä, vaan historiaan täytyy myös päästä käsiksi. Versionhallintasovellukset tarjoavat historian tarkastelemiseen lokin, jonka kautta on kätevä tutkia, miten tiedostot ovat muuttuneet versiosta toiseen.

Kuvassa 1 nähdään Subversion (SVN) -versionhallintasovelluksen graafinen työkalu TortoiseSVN ja sen tarjoama näkymä hakemiston muutoshistoriaan. Graafinen käyttöliittymä helpottaa muutoshistorian hahmottamista ja luo muutoksille selkeän aikajanan. [3, s. 35-36.]

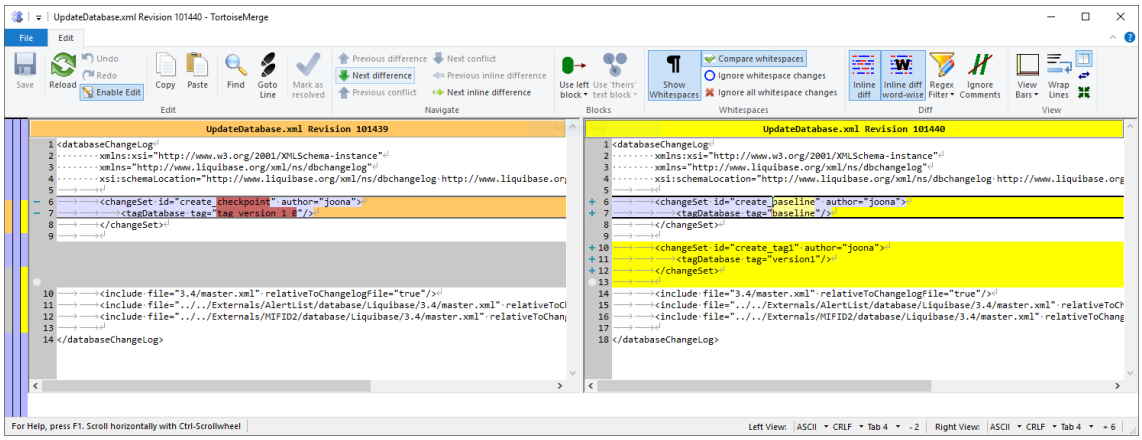


Kuva 1. TortoiseSVN-työkalun muutoshistoria.

Seuraava tärkeä ominaisuus versionhallinnassa on muutostyökalu, jolla nähdään kahden version muutokset toisiinsa nähden. Vertailtavana voi olla tietyn tiedoston, hakemiston tai koko projektin mitkä tahansa kaksi tilaa. Työkalu esittää tiedostot vierekkäin ja merkitsee koodivärein muuttuneet rivit.

Kuvassa 2 näkyy TortoiseSVN:n toteutus muutostyökalusta. Vasemmalla nähdään base eli versionhallinnassa oleva viimeisin versio, johon paikallisesti muutettua oikealla nähtävää versiota – nimeltään working copy – verrataan. Muuttuneet ja lisätyt rivit näkyvät oikealla puolella keltaisella ja poistetut rivit harmaalla. Vasemmalla muuttuneet rivit näkyvät punaisella, poistetut oranssilla ja lisätyt harmaalla. Työkalulla on helppo tarkastella tekemiään muutoksia. Vertailutyökalu on käytännössä hyödyllinen vain tekstipohjaisten

tiedostojen vertailussa, sillä binäärimuodossa olevien tiedostojen, kuten kuvien binääridata ei ole ihmiselle ymmärrettävää luettavaa. Tällaisten tiedostojen vertailu on parasta tehdä manuaalisesti avaamalla kaksi versiota vierekkäin. [3, s. 36.]



Kuva 2. TortoiseSVN näyttää tiedostossa muuttuneet, lisätyt ja poistetut rivit koodivärein.

Seuraava suuri ja tärkeä perusominaisuus versionhallinnassa on haaroittaminen (branching) ja haarojen yhdistäminen (merging). Haaroittamisella tarkoitetaan projektille uuden kehityshaaran perustamista, eli muutokset vaikuttavat jatkossa vain kehityshaarassa oleviin tiedostoihin. Kehityshaara on tavallaan kopio alkuperäisestä haarasta. Tämä ominaisuus tulee erityisesti tarpeeseen siinä vaiheessa, kun johonkin tuotteeseen halutaan tehdä suuria muutoksia tai uusia ominaisuuksia. Tuote voi olla epäkunnossa ennen kuin kaikki ominaisuudet on saatu valmiiksi, joten paras ratkaisu on tehdä kehityshaara, jonka ei tarvitse olla käyttökunnossa kehitysvaiheen ajan. [3, s. 38.]

Kuvassa 3 havainnollistetaan haaroituksen etuja. Tuote on päähaarassaan kuvan oikeassa laidassa versiossa 1.0. Tuotteeseen on suunniteltu uusia ominaisuuksia, joiden myötä sen versio nousee versioon 1.1, mutta päähaaran on pysyttävä jatkuvasti käyttökunnossa ja siihen täytyy säilyä mahdollisuus tehdä pieniä kriittisiä korjauksia. Luodaan siis kehityshaara versiolle 1.1, johon voi tehdä version nostoon vaadittavia kehityksiä ilman huolta päähaaran rikkoutumisesta. Lopuksi, kun ominaisuudet on tehty ja testattu, yhdistetään kehityshaara päähaaraan eli tehdään merge, jonka jälkeen päähaara on nostettu versioon 1.1 ja tuote on ollut toiminnassa alusta loppuun. [3, s. 38.]

Kuva 3. Tuotteen version nosto versionhallinnan haaroja hyödyntämällä [3, s. 38].

Merge on monimutkainen ja virhealtis toimenpide, joten on tärkeää, että versionhallintasovellus helpottaa sen tekemistä mahdollisimman paljon tunnistaen, miten tehdyt muutokset tuodaan sujuvimmin uuteen haaraan. Kaikkia tapauksia versionhallintakaan ei pysty ratkaisemaan ja näitä tapahtumia kutsutaan konflikteiksi. Konflikti voi esimerkiksi syntyä, jos yhdessä haarassa muokataan tiedostoa ja toisessa haarassa se on poistettu. Konfliktien ratkaiseminen jää kehittäjän tehtäväksi ja niitä ratkaistaessa täytyy olla tarkkana, jotta lopputulos on toimiva. [3, s. 39.]

Versionhallinta on elintärkeä konsepti kaikessa ohjelmistokehityksessä, sillä useat kehittäjät tekevät samanaikaisia muutoksia samoihin tiedostoihin. Versionhallinnalla yhdessä oikean toimintatavan kanssa varmistetaan, että muutokset toimivat yhdessä toistensa kanssa ja että kaikilla kehittäjillä on yhtenevät versiot tiedostoista. Mikäli jokin muutos aiheuttaa virheen ohjelmassa, on edelliseen ja toimivaan versioon helppo palata.

Jaetaan versionhallinta kahteen osaan tarkastelua varten: lähdekoodin versionhallintaan ja tietokannan versionhallintaan.

2.2 Lähdekoodin versionhallinta

Lähdekoodin versionhallinta on ohjelmistokehityksen perusasioita ja se tulisi olla käytössä kaikissa projekteissa, sillä se helpottaa kehitystä huomattavasti ja nykyiset versionhallintasovellukset ovat erittäin käyttäjäystävällisiä.

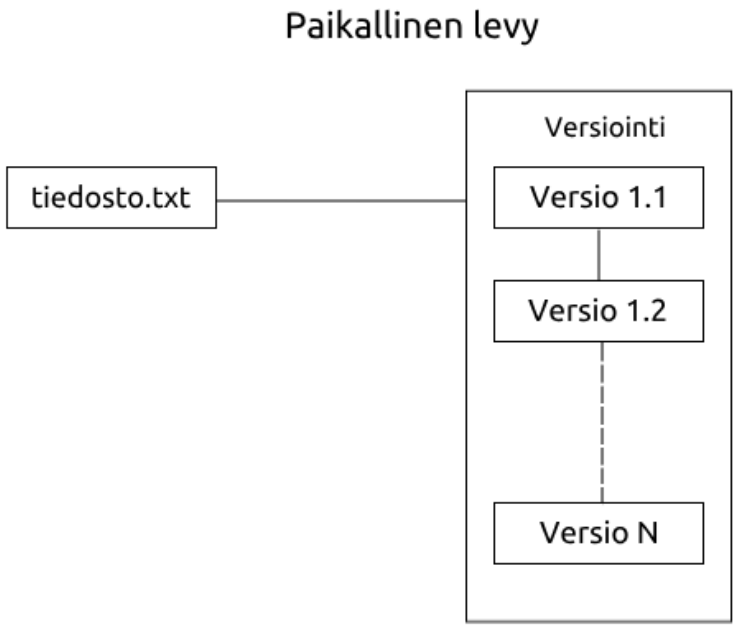
Lähdekoodin versionhallinta jaetaan kolmeen kategoriaan, joita ovat:

- paikallinen versionhallinta
- keskitetty versionhallinta
- hajautettu versionhallinta.

Paikallinen versionhallinta

Paikallinen versionhallinta oli ensimmäinen ratkaisu ongelmaan, jossa samasta tiedostosta täytyi olla samanaikaisesti useita versioita ilman virhealtista ja sekavaa nimeämiskäytäntöä. Revision control system (RCS) oli yksi suosituimmista paikallisista versionhallintasovelluksista. RCS tallentaa tiedoston sisällön muutokset eri vaiheissa käyttäen paikalliselle levylle tallennettua version seurantaa. Käytännössä RCS pilkkoo tiedoston sisällön pieniin osiin ja säilyttää ne omassa formaatissaan. Näitä osia yhdistämällä RCS voi palauttaa tiedoston mihin versioon tahansa. [2, s. 34.]

Kuvassa 4 on esitetty paikallisen versionhallinnan toimintaperiaate. Versiot tallennetaan käyttäjän levylle, joten jos levyyn tulee vikaa, menee myös tiedosto mukana. Paikallinen versionhallinta ei myöskään mahdollista sujuvan yhteistyön tekemistä, sillä jokaisella kehittäjällä on omat versionsa tiedostoista, eikä niitä yhdistetä missään vaiheessa toisten versioihin.

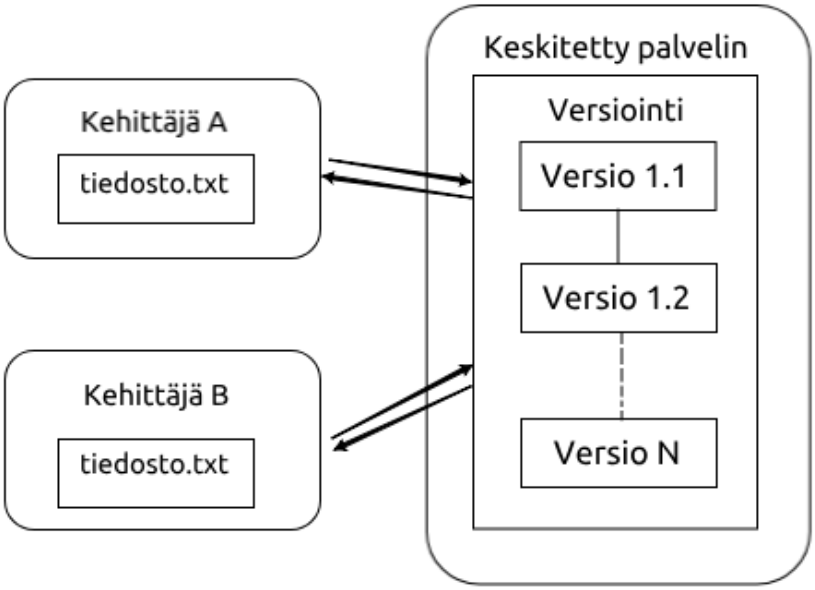


Kuva 4. Paikallisen versionhallinnan toimintaperiaate [2, s. 34].

Keskitetty versionhallinta

Keskitetty versionhallinta on paikallisesta versionhallinnasta seuraava vaihe. Sen avulla kehittäjät pystyvät työskentelemään yhteistyössä ilman paikallisen versionhallinnan rajoituksia. Sen toimintaperiaatteena on säilyttää versionhallinnassa olevat tiedostot palvelimella, johon kehittäjillä on pääsy ja tiedostoja noudettaessa palvelimelta haetaan aina tiedoston uusin versio. Näin kaikki kehittäjät pääsevät samoihin tiedostoihin käsiksi ja näkevät, mitä muutoksia toiset ovat tehneet. [2, s. 35-36.]

Kuvassa 5 nähdään, miten versiointi on tallennettuna keskitetylle palvelimelle paikallisen levyn sijaan ja kehittäjillä on sekä luku- että kirjoitusoikeudet tiedostoihin.



Kuva 5. Keskitetyn versionhallinnan toimintaperiaate [2, s.35].

Keskitetystä versionhallinnasta tuli nopeasti suosittua, ja se on ollut pitkään hallitseva tekniikka versionhallinnan saralla. Paikalliseen versionhallintaan verrattuna yhteistyön parantamisen lisäksi se toi ylläpitäjille paremmat mahdollisuudet hallita pääsyoikeuksia jaettuihin tiedostoihin. [4, s. 10-11.]

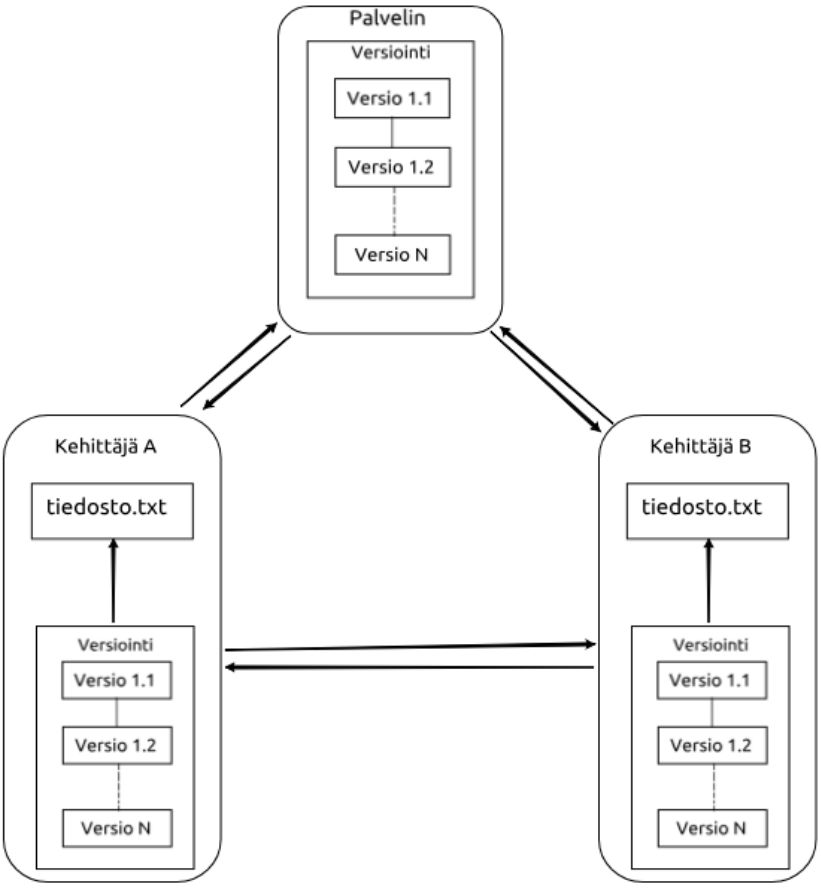
Tekniikassa on kuitenkin myös haittapuolensa, joista suurin on keskitetystä tiedostojen säilyttämisestä johtuva riski. Palvelimen ollessa tavoittamattomissa tiedostojen versiointi

ei ole mahdollista eikä siten myöskään kehittäjien välinen yhteistyö. Myös tiedon säilyminen vaarantuu, sillä jos palvelimen kiintolevy korruptoituu, on kaikki tieto menetetty. Kehittäjien tietokoneilla olevat versiot (snapshot) säilyvät, jos heillä niitä sattuu sillä hetkellä olemaan [4, s. 11.], joten tiedostoista säilyy vain viimeisin versio ja versiohistoria menetetään [2, s. 36]. Tätä riskiä voidaan laskea säännöllisillä varmuuskopioinneilla. Keskitetty versionhallinta kärsii siis tässä mielessä samoista ongelmista kuin paikallinenkin. Yksi tunnetuimmista keskitetyn versionhallinnan sovelluksista on Subversion, jota käytetään myös DiFS-kehityksissä. Muita vastaavia ovat CVS ja Perforce. [4, s. 11.]

Hajautettu versionhallinta

Viimeisin ja edistynein tekniikka versionhallinnassa on hajautettu versionhallinta, joka taklaa aikaisempien tekniikoiden ongelmia hyvällä menestyksellä. Hajautetussa versionhallinnassa käyttäjä saa palvelimelta tiedoston koko versiohistorian, ei vain viimeisintä versiota, joten palvelimen korruptoituessa ei menetetä mitään ja palvelimen tila saadaan palautettua kopioimalla kenen tahansa käyttäjän versiohistoria takaisin palvelimelle. Jokaisella käyttäjällä on siis käytännössä varmuuskopio tiedostojen versiohistoriasta. [4, s. 11.]

Kuvassa 6 on esitetty käyttäjien ja palvelimen väliset suhteet. Jokaisella käyttäjällä on versiohistoria levyllään ja tuota historiaa synkronoidaan sekä palvelimen että muiden käyttäjien kesken.



Kuva 6. Hajautetun versionhallinnan toimintaperiaate [2, s. 37].

Keskitetyssä versionhallinnassa historian säilyttäviä repositorioita on vain yksi ja hajautetussa niitä on useita. Hajautetussa versionhallinnassa tulee siis ylimääräinen vaihe tiedostoversion päivittämiseen, sillä ensin täytyy päivittää paikallinen versiohistoria, josta sen jälkeen päivitetään paikallisen tiedoston (working copy) versio. [5.]

Tiivistettynä hajautetun versionhallinnan edut verrattuna aiempiin tekniikoihin voidaan tiivistää kahteen ydinkohtaan: muutoksia tiedoston versiohistoriaan pystyy tekemään huolimatta siitä, onko versionhallintapalvelimelle jatkuva yhteys ja että versiohistoriaa säilytetään useissa eri paikoissa yhden sijaan.

Hajautettu versionhallinta sisältää näiden lisäksi keskitetyn versionhallinnan ja paikallisen versionhallinnan jo aiemmin tuomat edut, joten se on ikään kuin hybridijärjestelmä. [2, s. 36-37.] Hajautetun versionhallinnan puolesta puhuukin seuraava sanonta:

> You name anything that a centralized version control system can perform; a distributed version control system can handle the same thing and perform much better [2, s. 37].

Tällä hetkellä hajautettu versionhallinta on huomattavasti edistynein versionhallintajärjestelmä ja sitä tulisikin käyttää nykyajan ohjelmistokehitysprojekteissa. Suosittuja ja yleisesti käytettyjä sovelluksia ovat git, Mercurial, Bazaar ja Darcs [4, s. 11].

2.3 Tietokannan versionhallinta

Lähdekoodin versionhallinnasta poiketen tietokannan versionhallinnan käytännöt ja parhaat tavat eivät ole vielä löytäneet paikkaansa ohjelmistokehityksen kivijalassa. Tietokannan versionhallinta jää liian usein toissijaiseksi, sillä sen käyttöönotto koetaan vaikeaksi ja aikaa vieväksi, eikä sen käytänteitä tunneta kovin laajasti. Ongelmaksi muodostuu myös se, että tietokanta sisältää tietoa, jota ei saa hävittää. Muutoksia tehdessä tietokantaa ei voi siis korvata uudella kannalla, kuten lähdekoodin tapauksessa tiedostot voidaan korvata toisilla, sillä tuolloin menetettäisiin myös tietokannan sisältämä tieto. Tietokannan versionhallinnan vaikeuden takia asian lykkääminen ei kuitenkaan ole kestävä toimintatapa, sillä projektien edetessä ja tietokannan kasvaessa muutosten hallintaan ja tietokannan ylläpitoon kuluu runsaasti työtunteja, jolloin tietokannan versionhallintaan panostaminen olisi tullut halvemmaksi. Ongelmia tulee erityisesti toimitusvaiheessa, jos tietokannan tila ei ole selkeä eikä se toimi yhdessä lähdekoodin kanssa.

Tietokannan versionhallinnasta hyötyy muun muassa seuraavilla tavoilla:

- Koodimuutosten helppo jaettavuus tiimin kesken. Tietokannan skeeman ja referenssidatan ollessa versionhallinnassa, on kaikilla pääsy näihin muutoksiin.
- Parempi kokonaiskuva kehityksen etenemisestä.
- Mahdollisuus palata aiempaan versioon tietokannasta.
- Muutosten seurannan helpottuminen: muutoksista jää aina jälki.
- Mahdollistaa tehokkaamman automaation ja jatkuvan toimittamisen myös tietokannalle.
- Mahdollistaa version synkronoinnin tietokannan ja lähdekoodin välillä. [6.]

Tietokannan versionhallinta on nykypäivänä aiempaa helpompaa valmiiden työkalujen myötä, joten se yleistyy jatkuvasti [6].

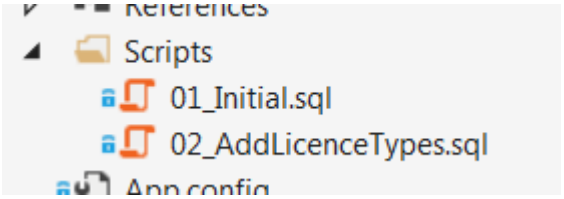
Scott [7] esittelee tietokantojen hallinnan kolme tärkeintä asiaa:

- kehitystyössä ei tule käyttää jaettuja tietokantoja
- tietokannan skeemalle on vain yksi lähde
- tietokanta tulee aina versioida.

Jaetun tietokannan käyttäminen saattaa kuulostaa yksinkertaisimmalta ja parhaalta lähestymistavalta: kaikki kehittäjät käyttävät palvelimelta samaa tietokantaa, jota vasten testataan kehityksiä ja jonka skeemaan tehdään muutoksia. Näin kaikilla on pääsy jatkuvasti uusimpaan dataan ja tietokannan viimeisimpään versioon. Se kuitenkin aiheuttaa väistämättä jossain vaiheessa ongelmia, sillä yhden kehittäjän muutokset rikkovat toisen tekemiä muutoksia, jolloin tekeminen hidastuu ja vaikeutuu sekä on virhealtista. Jaettujen tietokantojen sijaan jokaisella kehittäjällä tulisi olla kehitystietokanta lokaalisti omalla tietokoneellaan. [7.]

Skeemasta tulisi olla vain yksi virallinen versio ja paikka, josta sen löytää. Ideana on, että projektiin liittyvä kehittäjä voi hakea versionhallinnasta uusimman version sekä lähdekoodista että tietokannasta ja saada projektin pystyyn omalle koneelleen muutamalla toimenpiteellä. Tietokannan luonti täytyy siis olla mahdollista valmiiden skriptien avulla. [7.]

Suosituin tapa versioida tietokanta on säilyttää SQL-skriptit yhdessä lähdekoodin kanssa versionhallinnassa. Versionhallinnassa ei säilytetä pelkkää skeemaa, vaan myös ohjelman toiminnalle välttämätöntä dataa, niin sanottua referenssidataa. Jokaisesta muutoksesta kirjoitetaan oma, numeroitu skripti. [8.] Kuvassa 7 nähdään kaksi skriptiä, jotka on tallennettu yhdessä lähdekoodin kanssa omaan kansioonsa ja numeroitu oikein.



Kuva 7. Numeroidut esimerkkiskriptit [8].

Skriptit ovat tärkeä pitää niiden ajamisen jälkeen muuttumattomana. Tätä on ehdotonta noudattaa, sillä muutosten jakaminen yksittäisiin skripteihin tehdään juuri sen takia, että muutoksia voidaan seurata ja versioinnin hyödyt menetetään, jos jo ajettuja skriptejä

muutetaan. Mikäli jo toimitettuja muutoksia täytyy perua, tehdään tätä varten uusi skripti. Muutokset skeemaan ja referenssidataan tehdään vain ja ainoastaan skriptien kautta, ei koskaan manuaalisesti, sillä skriptien ohi toimiminen kumoaisi sekin versioinnin. [8.]

Tietokannan versionhallinta manuaalisesti skriptejä hallinnoimalla voi käydä raskaaksi ja tämän vuoksi on kehitetty useita erilaisia aputyökaluja sekä skriptien hallintaan että koko tietokannan ylläpitoon ja päivittämiseen. Näitä työkaluja on kahdenlaisia: tila- ja migraatiopohjaisia.

Tilapohjaisia työkaluja käyttäessä ei tarvitse kuin ylläpitää tilannevedosta tietokannasta ja versiota nostettaessa vertailutyökalu vertaa päivitettävää tietokantaa malliksi asetettua skeemaa (etalon) vasten ja generoi tietokannan päivitysskriptin automaattisesti. Työkalu tekee suurimman osan raskaasta työstä, ja kehittäjät voivat keskittyä koodin kirjoittamiseen. Redgate esitteli tilapohjaiset työkalut ensimmäisen kerran ja Microsoft toi ne suuren yleisön tietouteen. [9.] Tilapohjaisia versionhallintatyökaluja ovat esimerkiksi kaupalliset Redgate SQL Source Control ja DBmaestro Source Control [10].

Migraatiopohjaiset työkalut sen sijaan tekevät useita migraatioita nostaessaan tietokannan versiota toiseen eli jokainen muutos on oma migraationsa, siinä missä tilapohjaiset hoitavat asian kerralla riippumatta muutosten määrästä. Nämä työkalut tukevat useita eri esitystapoja migraatioskripteille, kuten XML, SQL, Java ja C#, jotka työkalu muuttaa tietokannan tukemalle kielelle, kun migraatio tehdään. Migraatioskriptit, eli muutokset versioiden välillä, kirjoitetaan käsin, ja työkalu pitää kirjaa, mitkä migraatiot mihinkin tietokantaan on ajettu. [9.] Migraatiopohjaisia versionhallintatyökaluja ovat esimerkiksi vapaan lähdekoodin Flyway ja Liquibase sekä kaupallinen Datical DB [10].

Tila- ja migraatiopohjaisilla työkaluilla on omat vahvuutensa, ja lähestymistapa tulisi valita projektin vaatimusten mukaan. Konfliktien hallitsemisessa on suuri ero näiden tekniikoiden välillä. Tilapohjaisissa työkaluissa se on suhteellisen yksinkertaista, sillä koko tietokanta esitetään SQL-skriptinä, jolloin konfliktit ovat yksiselitteisiä. Migraatiopohjaisissa työkaluissa ongelmia tulee erityisesti tallennettujen proseduurien migraatioissa, ja jos näihin tarvitsee jatkuvasti tehdä muutoksia, saattaa tilapohjainen ratkaisu olla tehokkaampi. [9.] Fowler & Sadalage [11] mainitsevat, että toimituksia pitää tehdä usein. Tämä puoltaa migraatiopohjaista lähestymistapaa, sillä ne mahdollistavat pienetkin muutokset nopeasti, mikä on myös agilen mukainen toimintatapa. Tekniikoista ei voi kuitenkaan

yksiselitteisesti sanoa parempaa, vaan niiden heikkouksia ja vahvuuksia täytyy punnita tarpeen mukaan ja valita tilanteeseen sopivin.

3 Tietokannat DiFS-organisaatiossa

3.1 Tietokannan hallintajärjestelmät

Tietokannan hallintajärjestelmä (TKHJ) on ohjelmisto, joka mahdollistaa tietokannan luomisen ja hallinnoimisen. Se tulkkaa käyttäjän antamat SQL-kyselyt, suorittaa tarvittavat toimenpiteet ja palauttaa vastauksen käyttäjälle.

TKHJ:n toimintoihin kuuluvat

- tiedon määrittely: relaatiot, riippuvuudet, näkymät jne.
- tiedon muuttaminen: lisääminen, päivittäminen, poistaminen, hakeminen, uudelleenjärjestäminen ja koonti
- tiedon eheyden varmistaminen
- tiedon ja kyselyjen optimointi
- tietohakemiston ylläpito
- tuki erilaisille tietokantaan liittymättömille toiminnoille, kuten kopioinnille
- tuki ohjelmointikielelle tai -kielille
- transaktioiden hallinta
- varmuuskopiointi ja sen palautus
- integraatiomahdollisuus tiedonvälitysohjelmistolle
- tuki tietokantojen rajapinnoille, kuten Open Database Connectivity (ODBC) ja Java Database Connectivity (JDBC). [12, s. 21.]

Näistä toiminnoista vastaavat TKHJ:n eri osat, joita ovat

- tietokantamoottori, joka välittää ja tulkkaa viestejä järjestelmien välillä
- tiedonmäärittelyjärjestelmä (data definition subsystem), joka vastaa tietohakemiston ylläpidosta
- suorakäyttöliittymä
- tietohakemistojärjestelmä, joka sisältää tietokannan määrittelyt
- ohjelmointirajapintajärjestelmä
- tiedonhallintajärjestelmä, joka vastaa tietokannan eheydestä, varmuuskopioinnista ja sen palauttamisesta sekä kyselyjen optimoinnista.

Tietokantamoottori on siis TKHJ:n runko, ja kaikki muut järjestelmät ovat kiinni siinä. [12, s. 23-24.]

DiFS-organisaatiossa on käytössä kaksi TKHJ:ää: Microsoft SQL Server ja Oracle Database. SQL Server on selkeästi yleisempi asiakkailla, mutta tietokantaskriptejä ylläpidetään jokaisessa tuotteessa ja jokaisessa haarassa myös Oraclelle.

Microsoft SQL Server on Microsoftin ratkaisu tietokantojen hallintaan. SQL Serverin avulla voi kehittää moderneja sovelluksia millä tahansa ohjelmointikielellä paikallisesti tai pilvessä. Se on saatavilla Windowsin lisäksi myös Linuxille. Microsoft [13] kertoo SQL Serverillä olevan hyvä suorituskyky ja skaalautuvuus, vähän haavoittuvuuksia, reaaliaikainen tiedon analytiikka ja sovellukset myös mobiilialustoille. SQL Serverin käyttämä kieli on Transact SQL (T-SQL), jonka kehitti alun perin Sybase. T-SQL mielletään yksinkertaiseksi ja helpoksi käyttää. [14.]

Oracle Database on Oraclen ratkaisu tietokantojen hallintaan ja maailman suosituin tietokantojen hallintajärjestelmä. Oraclen [15] mukaan Oracle Database tarjoaa kaikenkokoisille yrityksille nopean, skaalautuvan ja luotettavan tietokantateknologian sekä pilvessä että lokaalisti. Uusin versio Oracle Database 18c mahdollistaa asiakkaan lokaalisti kehitetyn ja paikallisilla palvelimilla ajetun ohjelmiston viemisen pilveen tekemättä muutoksia ohjelmistoon.

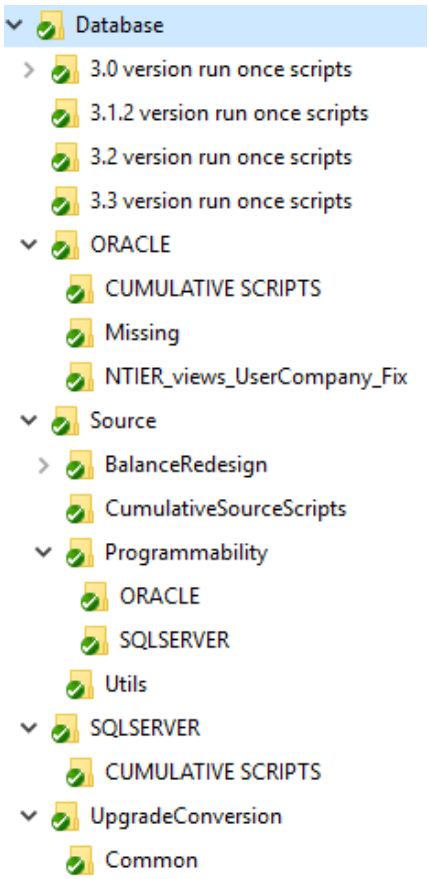
Oracle Databasen käyttämä kieli on Procedural Language/SQL (PL/SQL). Merkittävin ero Microsoftin T-SQL- ja PL/SQL-kielien välillä on muuttujien ja tallennettujen toimenpiteiden hallinta sekä sisäänrakennetut funktiot. PL/SQL mielletään monimutkaisemmaksi, mutta myös mahdollisesti tehokkaammaksi kieleksi. [14.] Syntaksi eroaa yllättäen jopa pienissä toimenpiteissä: SQL Serverissä NULL-arvon tarkistaminen tapahtuu funktiolla ISNULL() ja Oraclessa funktiolla NVL().

DiFSin käytössä on SQL Serverin hallintaan SQL Server Management Studio ja Oracle databasen hallintaan Oracle SQL Developer.

3.2 Tietokantojen ylläpito

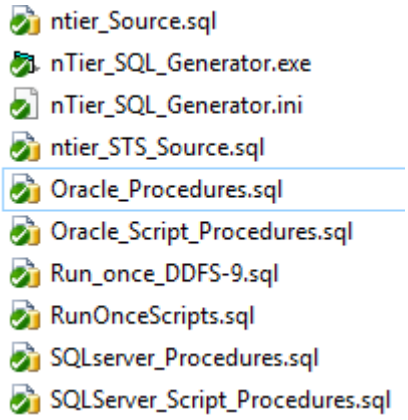
Tietokantojen ylläpito on toteutettu DiFSissä tietokantaskripteillä, jotka on lisätty SVN-versionhallintaan. Jokaisessa tuotteessa, haarassa ja moduulissa on skripteille oma kansionsa, jonka nimi on Database. Kuvassa 8 nähdään erään tuotteen ja haaran Database-kansion sisältö. Kansiorakenteesta löytyvät asiakaskohtaiset kansiot, joihin sijoitetaan

yleisistä skripteistä eroavat, vain tiettyä asiakasta koskevat muutokset. Asiakaskohtaiset kansiot on piilotettu kuvasta. Yleiset muutokset, jotka koskevat kaikkia asiakkaita ja muuttavat joko tietokannan skeemaa tai referenssidataa, säilötään Source-nimiseen kansioon. ORACLE- ja SQLSERVER-kansiot sisältävät nimensä mukaisesti SQL Server- ja Oracle-skriptit.



Kuva 8. Database-kansion sisältö.

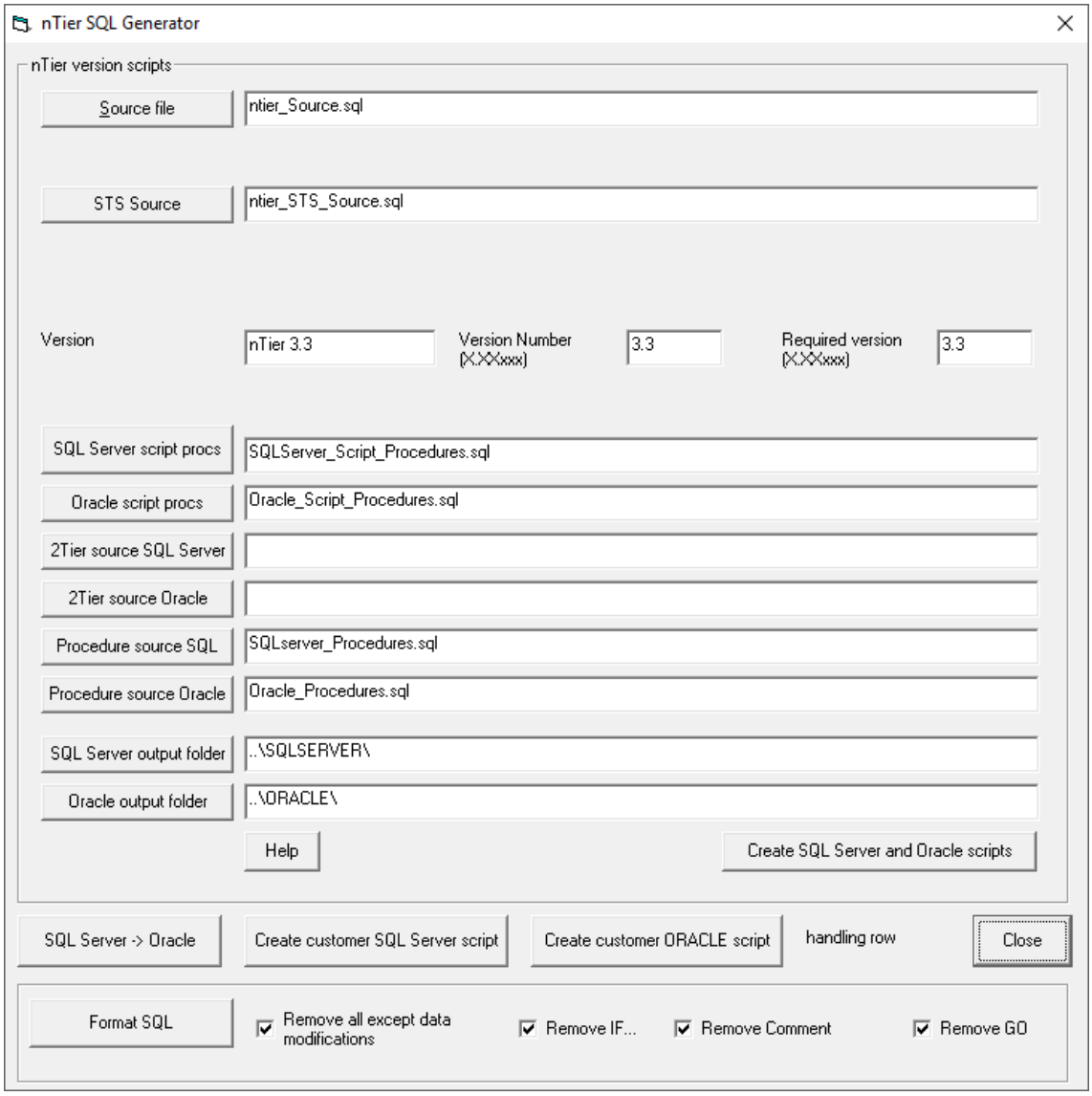
Keskitytään Source-kansion sisältöön, joka on tietokantojen ylläpidon kannalta kiinnostavin ja tärkein. Kuvassa 9 nähdään tärkeimmät Sourcen juuressa olevat skriptit. Skeeman muutokset ovat tiedostossa ntier_Source.sql ja referenssidatan muutokset tiedostossa ntier_STS_Source.sql.



Kuva 9. Source-kansion sisältö.

Muutokset kirjoitetaan tiedostoihin T-SQL:ää käyttäen, mutta DiFS tukee myös Oraclea, joten skriptit täytyy kääntää PL/SQL:lle. Ongelma on ratkaistu kehittämällä kuvassa 9 näkyvä ohjelma nTier_SQL_Generator.exe eli nTier SQL Generator. SQL Generator on kehitetty DiFSin sisällä kehittäjien toimesta ratkaisemaan kahden TKHJ:n eroavaisuuksista johtuvia ongelmia ja helpottamaan tietokannan versiointia. Sen avulla voidaan kääntää skriptit myös Oraclen ymmärtämään muotoon.

Kuvassa 10 nähdään SQL Generatorin käyttöliittymä. Skeemamuutokset sisältävä skripti annetaan kohtaan Source file ja referenssidatan muutokset sisältävä skripti kohtaan STS Source. Painamalla nappia Create SQL Server and Oracle scripts ohjelma generoi sekä SQL Serverille että Oraclelle erilliset kumuloituvat skriptit. Ohjelma generoi skripteihin myös logiikkaa, kuten IF NOT EXISTS -lauseita, jotka mahdollistavat sen, että samaa tiedostoa voidaan ajaa useita kertoja. Tämä onkin välttämätöntä kumuloituvissa skripteissä, jotka jatkuvan integraation ympäristö TeamCity on konfiguroitu ajamaan joka kerta, kun haara käännetään.



Kuva 10. nTier SQL Generator -ohjelman käyttöliittymä.

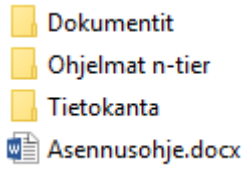
SQL Generator ratkaisee useita ongelmia, jotka syntyvät erillisen tietokannan version-hallintaohjelmiston puutteesta, ja se on tehtävässään kätevä.

Haaroilla on yleisesti ottaen kehitys- ja testitietokannat, mutta joissain tapauksissa käytetään samaa tietokantaa kehitykseen ja testaukseen. Toimitushaaroilla on lisäksi tietokannat, jotka pidetään samalla tasolla kuin toimitettava ohjelma. Nämä tietokannat sijaitsevat yhteisillä palvelimilla ja kaikki samaan ympäristöön muutoksia tekevät DiFS-kehittäjät käyttävät samaa kehitystietokantaa. Paikallisten tietokantojen käyttämättömyyteen on syynä tarvittavien tietokantojen suuri koko, sillä tietokantoja on kehitetty pitkään, ja

ohjelmistojen vaatiman referenssi- sekä testidatan määrä on suuri. Paikallisten tietokantojen puute hidastaa työntekoa esimerkiksi silloin, kun kehitys vaatii muutoksia tietokantaan: tietokantaskriptiä muutetaan ensimmäisenä, ajetaan se SQL Generatorin läpi, viedään muutokset versionhallintaan ja ajetaan TeamCityn käännös. TeamCityn käännös kestää noin 30 minuuttia, joten koko prosessiin kuluu tarpeettoman paljon aikaa. Tämän ongelman voi ratkaista tietokannan versionhallintaohjelmistolla, sillä tietokannan päivityksen voi tehdä omalta koneelta ja siten varmistaa, että tietokantamuutokset toimivat.

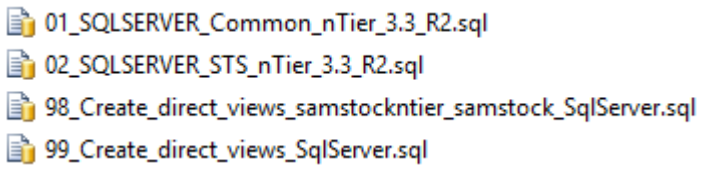
3.3 Tietokannat asiakastoimituksissa

DiFSin asiakastoimitukset muodostuvat dokumenteista, binääritiedostoista, asennusohjeesta ja tietokannasta. Dokumentteja ovat esimerkiksi toimitettavien uusien ominaisuuksien käyttöohjeet ja muutosloki, joka sisältää kaikkien toteutettujen korjauksien ja ominaisuuksien kuvaukset. Ohjelman binääritiedostot saadaan suoraan TeamCitystä. Kuvassa 11 on esitetty toimituksen rakenne.



Kuva 11. Toimituksen rakenne.

Tietokanta-kansio toimitetaan vain, jos tietokantaan on tehty muutoksia. Kansio sisältää skriptit, jotka voivat olla kerran ajettavia tai skeemaan ja referenssidataan muutoksia tekeviä kumuloituvia skriptejä. Mukana toimitetaan myös näkymät päivittävät skriptit. Kuvassa 12 näkyy esimerkki Tietokanta-kansion sisällöstä. Skriptit on numeroitu niiden ajojärjestyksen mukaisesti.



Kuva 12. Esimerkki Tietokanta-kansion sisällöstä.

Tietokanta-kansio täytyy koostaa käsin toimituksen mukaan. Paketin kokoojan täytyy käydä toimitettavat muutokset läpi ja tarkastaa tehtävänhallintaohjelmistosta, vaikuttavatko ne tietokantaan. Mikäli vaikuttavat, on kerättävä oikeat tietokantaskriptit pakettiin. Tämä on toimituksen tekemisen aikaa vievin osio tuotteiden suuren modulaarisuuden vuoksi. Iso osa DiFSin tuotteiden toiminnallisuuksista on kehitetty moduuleihin, jotta niitä voidaan käyttää eri tuotteissa vain lisäämällä viittaus moduuliin. Toiminnallisuuksia muutettaessa riittää muutos yhteen paikkaan.

Modulaarisuus aiheuttaa sivuvaikutuksenaan sen, että jokaisessa moduulissa täytyy olla omat tietokantaskriptinsä, jotka tekevät muutoksia skeemaan ja referenssidataan, kun moduuli otetaan käyttöön. Nämä muutokset eivät voi olla yleisissä skripteissä, sillä silloin vain moduulia koskevat muutokset tulisivat joka paikkaan. Tästä syystä tiedostosijainteja, joissa on mahdollista olla muutoksia toimituksen hetkellä, on runsaasti. Toimituspakettia koottaessa täytyy olla siis erittäin tarkkana ja tarkastaa, minne kaikkialle on tullut muutoksia ja mitkä skriptit pakettiin täytyy kerätä. Unohtunut skripti voi estää siihen liittyvän toiminnallisuuden käyttämisen, joten tähän ongelmaan on tärkeää löytää kestävä ratkaisu, joka minimoi inhimillisen virheen mahdollisuuden.

3.4 Ehdotuksia kehityskulttuurin kehittämiseksi

Tietokannan versionhallinta tehtynä tehokkaammin vaatii toimintatapojen ja kehityskulttuurin muutoksia DiFSin sisällä. Versionhallintaohjelmiston käyttöönotto on vain yksi osa koko kuvaa, sillä jos toimintatapoja sen ympärillä ei muuteta, ei tuosta muutoksesta pelkästään ole huomattavaa hyötyä. Versionhallintaohjelmisto mahdollistaa parempien kehitystapojen helpomman omaksumisen, joten tilaisuus tulee ehdottomasti hyödyntää.

Tietokantojen osalta tärkein muutos on siirtyä selkeämpään käytäntöön, miten tietokantoja käytetään kehityksessä. Scott [7] ja Khorikov [8] nostavat esille tärkeänä asiana, että jokaisella kehittäjällä tulisi olla paikallinen kopio tietokannasta, jotta kehitys olisi mahdollisimman tehokasta. Fowlerin ja Sadalagen [11] mukaan jokaisella kehittäjällä pitää olla vielä lisäksi oma versionsa tietokannasta palvelimella. DiFSin tietokannat ovat usein niin suuria sisältämänsä referenssi- ja testidatan vuoksi, ettei paikallisten tietokantojen käyttäminen ole mahdollista. Joissain tilanteissa ohjelmiston pohjana voidaan käyttää pienempää tai jopa tyhjää tietokantaa, jolloin paikallisten kehitystietokantojen käyttämistä olisi syytä harkita. Paikallisten tietokantojen hyödyntäminen aina kun mahdollista on

hyvä tavoite kehityskulttuurin kehittämiseksi. Versionhallintaohjelmisto tuo tähän suuren avun, sillä sen avulla jokainen pystyy yhdellä komennolla luomaan paikallisen tietokannan, joka vastaa versionhallinnassa olevaa skeemaa ja referenssidataa.

Tietokantaskriptien ylläpitäminen muuttuu tietokannan versionhallintaohjelmiston myötä, joten kehittäjien täytyy opetella uusi tapa tehdä tutut asiat. Valitusta ohjelmistosta riippuen saattaa skriptien ylläpitoon käytettävä kieli vaihtua SQL:sta toiseen, kuten XML:ään, ja SQL Generatorin tarve vähentyä, tai se voi poistua kokonaan. Toimitusten kokoamisessa päästään toivottavasti tilanteeseen, jossa skriptejä ei tarvitse enää erikseen kerätä, vaan tietokannan versionhallintaohjelmisto huolehtii tietokantaskriptien generoinnista yhteen paikkaan.

4 Tietokannan versionhallintaohjelmistot

4.1 Tietokannan versionhallintaohjelmiston valintakriteerit

Versionhallintaohjelmistoja on erilaisiin käyttötarkoituksiin ja erilaisilla ominaisuuksilla, joten valinta on tehtävä mahdollisimman hyvin, jotta tietokantojen versionhallinta ja päivittäminen onnistuu mahdollisimman tehokkaasti. Tavoitteena on valita DiFSille sopivin versionhallintaohjelmisto, joten kriteerit valinnalle ovat tuki

- Microsoft SQL Serverille ja Oracle Databaselle
- SVN:lle ja Gitille
- TeamCitylle
- Octopus Deploylle.

Ominaisuuksia, joita ohjelmistolla tulee mielellään olla ovat

- helpottaa tietokannan toimittamista ja päivittämistä
- samalla ohjelmistolla onnistuu sekä SQL Serverin että Oracle Databasen päivittäminen ja skriptien generointi
- tukee modulaarista ohjelmistoarkkitehtuuria
- laadukas dokumentaatio
- kohtuulliset kokonaiskustannukset.

Tärkeintä ohjelmistolle on tukea molempia TKHJ:ä ja muita DiFSissä käytettyjä työkaluja ja tekniikoita. Ominaisuuksista taas tietokannan toimittamisen ja päivittämisen helpottaminen, skriptien generointi molemmille TKHJ:lle ja modulaarisen ohjelmistoarkkitehtuurin tuki ovat vaatimuslistan kärjessä. Valitun versionhallintaohjelmiston on tuettava DiFSin tavoitteita. Dokumentaation laadukkuudella tarkoitetaan käyttöönottoa, käyttöä ja ongelmatilanteita varten tehtyjä oppaita ohjelmiston valmistajan puolesta. Samoilla skripteillä molempien tietokantatyyppien päivittäminen olisi käytännöllinen ominaisuus, jotta skriptejä ei tarvitsisi erotella Oraclelle ja SQL Serverille erikseen ja ohjelma hoitaisi tietokantojen kielien erot.

4.2 Vertailtavat versionhallintaohjelmistot

Erilaisia tietokannan versionhallintaohjelmistoja etsittiin internetistä Googlen avulla ja löydetyistä listauksista [10] valittiin kiinnostavimmat vaihtoehdot, haettiin niistä käyttäjäkokemuksia ja katsottiin aiheesta tehtyjä Youtube-videoita. Näistä sopivimmilta vaikuttaneista ohjelmistoista otettiin enemmän selvää.

Kaupalliset ratkaisut, kuten Redgate [16] ja DBmaestro [17], esiintyvät useissa listoissa. Niiden vahvuuksia ovat viimeistelty ulkonäkö ja toiminta sekä hienot graafiset käyttöliittymät. Nämä valitut kaupalliset ratkaisut ovat tilapohjaisia toiminnaltaan ja mahdollistavat vahvan automaation tietokantojen hallintaan. Redgaten suurin heikkous on, että sama työkalu ei tue SQL Serveriä ja Oraclea, vaan heiltä täytyisi ottaa kaksi tuotetta, mikä tuplaisi kustannukset. Tässä tilanteessa kustannukset olisivat noin 2500 euroa kehittäjää kohti. DBmaestron ratkaisu tukisi molempia TKHJ:itä, mutta heidän sivunsa ovat informaation suhteen erittäin puutteelliset. Sivuilta ei löydy hinnoittelua eikä ilmaista kokeiluversiota. Internetistä ei myöskään löydy käyttäjäkokemuksia heidän tuotteistaan, joten tuki jäisi kokonaan valmistajan varaan. Kaupalliset tuotteet vaikuttavat tässä vaiheessa varsin järeiltä, ja näistä syistä johtuen ne jätetään pois vertailusta.

Varteenotettavimmiksi vaihtoehdoiksi nousivat migraatiopohjaiset, avoimeen lähdekoodiin perustuvat Flyway [18] ja Liquibase [19]. Flyway on Boxfusen tietokannan versionhallintaohjelmisto, joka on kasvattanut suosiotaan huomattavasti viime vuosina [19]. Flywayn ideana on toteuttaa tietokannan migraatiot mahdollisimman yksinkertaisesti ja tehokkaasti. Tässä se myös onnistuu käyttäjäkokemusten mukaan.

Liquibase on Daticalin ohjelmisto, ja se on myös erittäin laajasti käytetty tietokannan versionhallinnassa. Liquibasea pidetään hieman monimutkaisempana, mutta myös laajempana ja monikäyttöisempänä ohjelmistona kuin Flywayta.

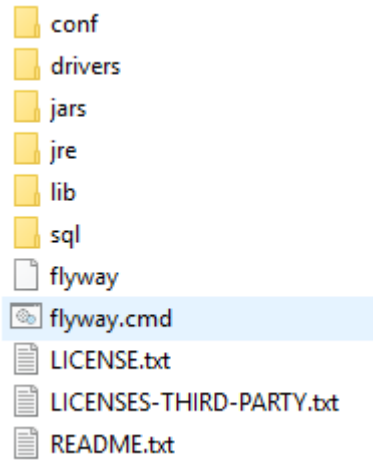
Liquibase täyttää kaikki DiFSin vaatimukset tietokannan versionhallintaohjelmistoksi, ja Flyway vaatii tarkempaa tarkastelua modulaarisuuden tuen osalta. Valitaan nämä kaksi ohjelmistoa vertailtavaksi.

4.3 Kandidaattien vertailu ja asennettavan ohjelmiston valinta

DiFSin vaatimusten perusteella tietokannan versionhallintaohjelmiston kandidaateiksi valittiin kaksi, Boxfusen Flyway ja Daticalin Liquibase. Ohjelmistojen toimintaperiaate on sama molemmissa eli migraatiopohjainen versionhallinta, ja molemmat täyttävät kaikki aiemmin luetellut ohjelmistojen yhteensopivuudet. Molempia ohjelmistoja käytetään komentoriviltä, ja kaikki esitetyt komennot ajetaan sieltä.

Flyway

Flywayn asentamiseen riittää, kun lataa uusimman paketin Flywayn sivuilta ja purkaa sen haluttuun hakemistoon. Kuvassa 13 on esitetty Flywayn hakemistorakenne. Flyway on Java-pohjainen, joten drivers-kansiosta löytyvät JDBC-ajurit eri tietokantatyypeille. Conf-kansiossa sijaitsee Flywayn konfiguraatio ja riippuen migraatioskriptien kirjoituskielestä ne sijoitetaan joko jars- (Java) tai sql-kansioon (SQL).



Kuva 13. Flywayn hakemistorakenne.

Käyttöönotto tapahtuu yksinkertaisimmillaan lisäämällä konfiguraatioon tietokannan tyypin, URL:n, käyttäjätunnuksen ja salasanan, kuten esimerkkikoodissa 1 on esitetty.

```
flyway.url=jdbc:h2:file:./foobardb
flyway.user=SA
flyway.password=
```

Esimerkkikoodi 1. Flywayn esimerkkikonfiguraatio [20].

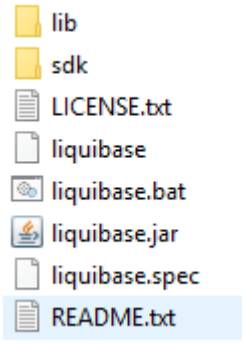
Tämän jälkeen voidaan luoda yhdellä komennolla baseline, eli pohjaversio, johon jatkossa migraatioita verrataan, jo olemassa olevasta tietokannasta. Tämä tapahtuu komennolla flyway baseline. Skeema ja referenssidata pohjaversiolle täytyy tuoda SQL-muodossa Flywayn luettavaksi. Flyway toimii lisäämällä tietokantaan taulun FLYWAY_SCHEMA_HISTORY, johon lisätään merkinnät ajetuista skripteistä. [21.]

Migraatioskriptit voidaan kirjoittaa joko Javalla tai SQL:lla, mutta Flyway ei osaa muuttaa kieltä toiseen, eli DiFS:n tapauksessa jouduttaisiin ylläpitämään T-SQL- ja PL/SQL-skriptejä. Kääntämisessä kielestä toiseen voitaisiin hyödyntää edelleen nTier SQL Generatoria. Migraatioskriptit nimetään aloittaen konfiguraatiossa määritellyllä versiointitunnisteella, joka on oletuksena VX__, jossa X on version numero [20]. Jokaisesta muutoksesta kirjoitetaan oma skriptinsä. Migraatio tapahtuu komennolla flyway migrate, jolloin Flyway käy läpi tietokannan historiataulun ja selvittää skriptit, jotka on lisätty edellisen migraation jälkeen ja ajaa ne numerojärjestyksessä. Ajetuista skripteistä lisätään merkintä historiatauluun. Flyway ei tue tietokannan automaattista palautusta edelliseen versioon (rollback), vaan palautus on tehtävä käsin. [21.]

Flywayn dokumentaatio on selkeä ja yksinkertainen varsinkin alkuun pääsemisessä, mutta monimutkaisempien toimenpiteiden tekemiseen on etsittävä vastaukset muualta. Esimerkiksi Flywayn valjastaminen tukemaan jatkuvan toimittamisen mallia täytyy selvitellä käyttäjien kirjoittamien oppaiden perusteella.

Liquibase

Liquibasen asentaminen on hieman monimutkaisempaa kuin Flywayn. Liquibasen sivuilta haetaan viimeisin versio ja puretaan se haluttuun hakemistoon. Kuvassa 14 näkyy Liquibasen hakemistorakenne. Liquibase.bat täytyy lisätä Windowsin PATH-ympäristömuuttujaan, jotta Liquibasea voi käyttää komentoriviltä. Uusimman version paketista on jäänyt valmistajalta puuttumaan logger-komponentti slf4j-api-1.7.25.jar, joka täytyy hakea itse ja viedä lib-kansioon. Vaihtoehtoisesti voi käyttää vanhempaa versiota Liquibasesta, jossa tuo komponentti on mukana. Asennuksen jälkeen Liquibasea kutsutaan komentoriviltä komennolla liquibase.



Kuva 14. Liquibasen hakemistorakenne.

Liquibasen konfigurointi tapahtuu lisäämällä liquibase.properties-tiedosto asennuskansion juureen. Liquibase tarjoaa mahdollisuuden tehdä erittäin monimutkaisia konfiguraatioita ja on siten monipuolisempi kuin Flyway, mutta yksinkertainen konfiguraatio on kuitenkin hyvin samannäköinen kuin Flywaylla, kuten esimerkkikoodissa 2 nähdään.

```
driver: oracle.jdbc.OracleDriver
classpath: ../ojdbc14.jar
url: jdbc:oracle:thin:@localhost:1521:XE
username: tbd
password: tbd
```

Esimerkkikoodi 2. Liquibasen esimerkkikonfiguraatio [22].

Liquibasen tuominen jo valmiina oleviin tietokantoihin on haastavampaa kuin Flywayn. Liquibase sisältää komennon generateChangelog, jolla se tekee kohteena olevasta tietokannasta changelog-tiedoston, jotta näyttäisi siltä kuin Liquibase olisi ollut käytössä projektin alusta asti. Se ei välttämättä kuitenkaan osaa tuoda kaikkia monimutkaisempia rakenteita, kuten tallennettuja proseduureja, joten kaikki changeSetit eli migraatiot täytyy käydä läpi käsin. DiFSin tilanteessa tämä on varsin suuri urakka tietokantojen laajuuden takia. Toinen mahdollisuus on ottaa Liquibase käyttöön ilman pohjaversiota, jolloin versiointi Liquibasen kautta alkaa seuraavassa tietokannan versiossa. Tässä ongelmana on, että uuden tietokannan luominen ei onnistu Liquibasen avulla ja se täytyy tehdä käyttäen esimerkiksi SQL Management Studion varmuuskopiointitoimintoa. [23.]

Liquibase lisää kaksi taulua tietokantaan, jotka ovat nimeltään databasechangelog ja databasechangeloglock. Databasechangelog-taulu sisältää kaikki SQL-lauseet, joita tietokantaan on ajettu, eli tietokannan versiohistorian ja databasechangeloglock-taulua

käytetään varmistamaan, että tietokantaan voi tehdä muutoksia vain yksi käyttäjä kerrallaan. [23.]

Migraatioskriptien asiaa ajaa Liquibasen kohdalla changelog-tiedosto. Liquibase on erittäin joustava changelogin formaatin suhteen, sillä ne voidaan kirjoittaa XML-, YAML-, JSON- tai SQL-formaatissa [19]. Tämä onkin Liquibasen suurin etu verrattuna Flywayhin, sillä DiFSin tilanteessa riittää, kun changelogin kirjoittaa XML-muodossa ja konfiguroi Liquibasen toimimaan sekä SQL Serverille että Oracle Databaselle, jolloin Liquibase hoitaa muutoksen XML-formaatista oikealle SQL-variaatiolle. Tällöin ei tarvitse ylläpitää molemmille TKHJ:lle omia skriptejä, vaan yksi riittää. Näin myös tarve nTier SQL Generatorille mahdollisesti häviää. Toinen suuri etu on tuki edelliseen versioon palaamiselle automaattisesti Liquibasen komennoilla. Liquibasen dokumentaatio on myös erittäin laaja ja ulottuu monimutkaisiinkin ominaisuuksiin [24].

Ohjelmiston valinta

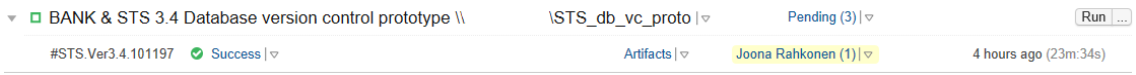
Suurimmat erot Flywayn ja Liquibasen välillä ovat migraatioskriptien käsittelyssä ja formaatissa sekä toimintojen laajuudessa. Flyway on yksinkertaisempi työkalu ja kenties pelkässä versionhallinnassa yhdelle tietokantatyypille parempi. Pohjaversion luonti on sillä mahdollisesti sujuvampaa, mutta Liquibasen tuki usealle tietokantatyypille samaan aikaan XML-muotoisten migraatioskriptien myötä on suurempi etu. Liquibase vaikuttaa myös laajennettavammalta ja sopivammalta työkalulta jatkuvan toimittamisen tavoitteeseen. Liquibasen valmistajalla Daticalilla on myös tuote nimeltä Datical DB, joka jatkaa siitä, mihin Liquibase jää, ja lisää ominaisuuksia laajempaan käyttöön. Liquibase on siis mahdollista päivittää Daticaliin tulevaisuudessa, jos ominaisuuksia tarvitaan lisää.

Näiden kahden ohjelmiston ominaisuuksien vertailun jälkeen Liquibase nousi voittajaksi monipuolisuutensa vuoksi. Flyway on tehokas skriptien ylläpitoon, mutta sen ominaisuudet jäävät liian laihoiksi muun muassa puuttuvan rollback-ominaisuuden vuoksi. Valitaan siis testattavaksi tietokannan versionhallintaohjelmistoksi Liquibase.

5 Tietokannan versionhallintaohjelmiston käyttöönotto

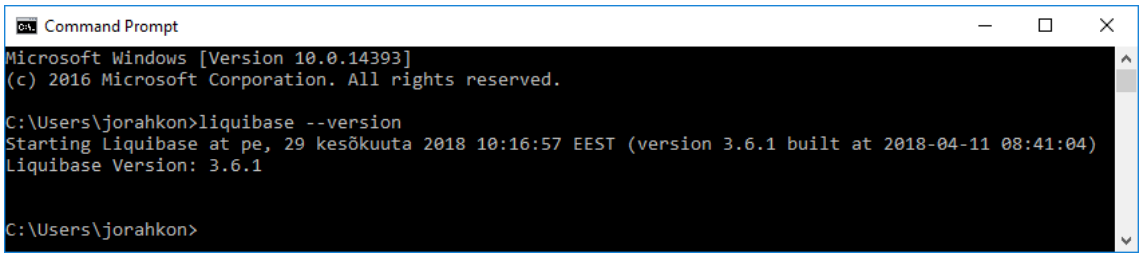
5.1 Asennus

Asennuksen valmistelu aloitettiin tekemällä DiFSin Securities Trading System (STS) -tuotteesta uusi haara SVN:ään, jotta muiden työskentely ei häiriintyisi versionhallinnan testauksesta. Haaran nimeksi valikoitiin db_vc_proto. Tuotteessa on 27 moduulia, joista kahdesta tehtiin myös uudet haarat modulaarisuuden testausta varten. Testiympäristöä varten tarvittiin lisäksi TeamCityn agenttipalvelin, jolle asennettiin Liquibase, ja tietokanta, jota käytettiin testeissä. Jatkuvan koonnin palvelimeksi otettiin yksi aktiivisessa käytössä olevista agenttipalvelimista ja tietokannaksi kehitystietokanta, joka on tuotteen nykyisen version kanssa samalla tasolla. Jatkuvan koonnin palvelimelle asennettiin Liquibasen esivaatimuksiin kuuluva Java 8 ja TeamCityyn tehtiin uusi koonti, joka asetettiin viittaamaan uuteen haaraan. Kuvassa 15 nähdään, että koonti on luotu ja käännös on mennyt läpi.



Kuva 15. Testiympäristön koonti TeamCityssä.

Liquibasen asentaminen palvelimelle aloitettiin lataamalla uusin 3.6.1-versio ja purkamalla paketti osoitteeseen C:\Liquibase. Samalla vietiin aiemmin mainittu logger-komponentti lib-kansioon. Tämän jälkeen ladattiin SQL Serverin JDBC-ajuri versiota 6.4 ja purettiin se hakemistoon C:\Program Files (x86)\ Microsoft JDBC Driver 6.4 for SQL Server sekä kopioitiin x64-arkkitehtuurin sqljdbc_auth.dll-tiedosto samaan kansioon ajurin kanssa, mikä mahdollistaa Windows-autentikoinnin käyttämisen palvelimeen yhteyttä otettaessa. Lopuksi lisättiin Liquibasen asennuskansion ja autentikoinnin tiedoston polut PATH-ympäristömuuttujaan, jonka jälkeen Liquibase oli käytettävissä komentoriviltä, kuten kuvasta 16 nähdään.

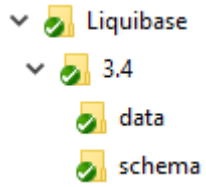


Kuva 16. Liquibase on käytettävissä palvelimen komentoriviltä.

5.2 Yhdistäminen lähdekoodin versionhallintaan

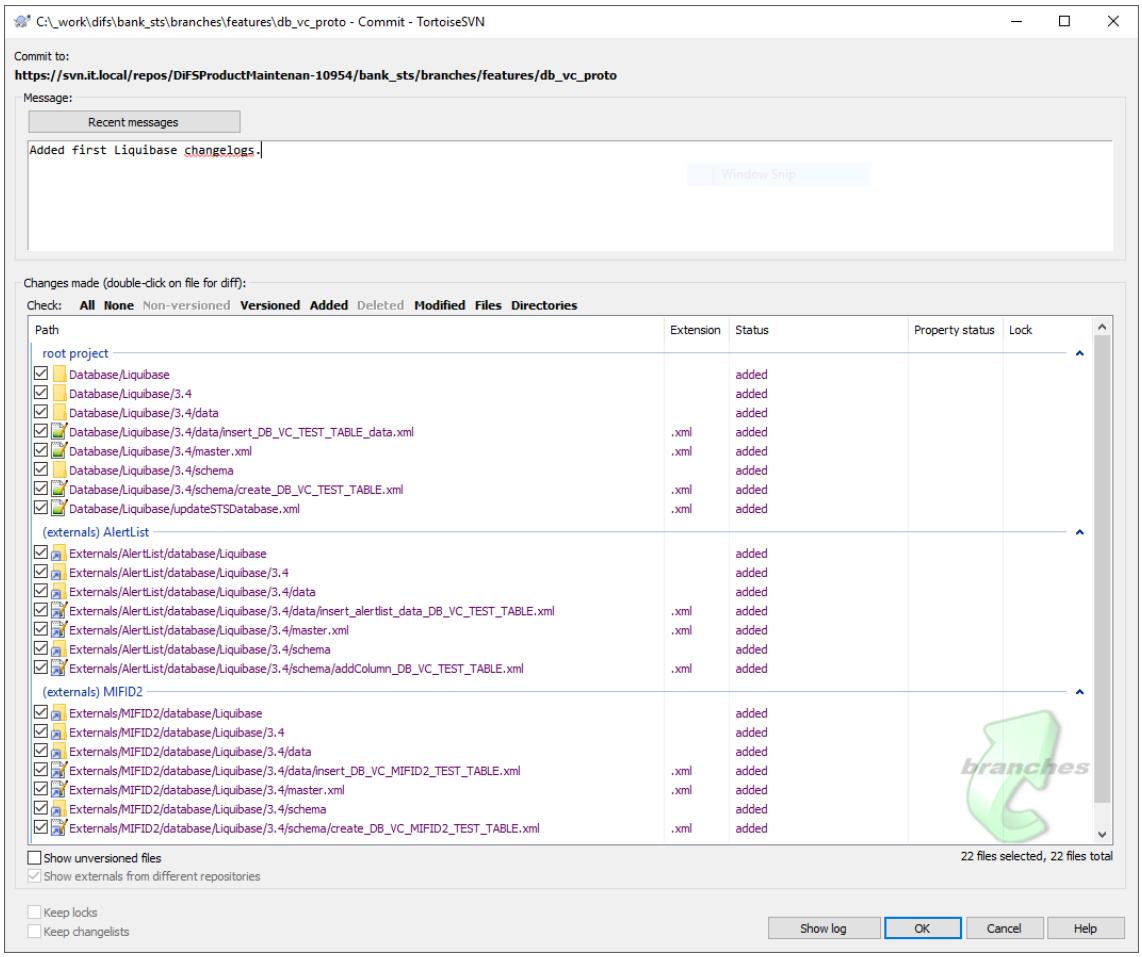
Liquibasen yhdistäminen lähdekoodin versionhallintaan tarkoittaa changelog-tiedostoista koostuvan ketjun lisäämistä versionhallinnan alle. Tavoitteena oli, että Liquibasen update-komentoa voi käyttää antamalla sille polku yhteen tiedostoon, joka sisältää polut kaikkiin haaran käyttämiin changelogeihin.

Testiympäristöön hahmoteltiin hakemistorakennetta, jossa Liquibase-kansio sisältää kaikki siihen liittyvät tiedostot. Liquibase-kansion juuressa on tiedosto, joka sisältää polut kaikkiin changelogeihin ja versioiden changelogit on niputettu omiin kansioihinsa riippuen siitä, koskeeko muutos skeemaa vai referenssidataa. Jokaisen versiokansion juuressa on master.xml, joka sisältää sen version tiedostojen polut, jotta pääskriptiin ei tarvitse lisätä kuin jokaisen version master.xml-tiedosto. Kuvassa 17 on esitetty hakemistorakenne, joka tulee jokaisen moduulin Database-kansioon.



Kuva 17. Liquibasen changelog-hakemistorakenne.

Testiympäristössä on tehty haarat testikäyttöön MIFID2- ja AlertList-moduuleille, joihin tehtiin sama hakemistorakenne. Moduulien muutoksille tehdään omat tiedostot omiin kansioihinsa, ja jos moduuli otetaan käyttöön johonkin tuotteeseen, lisätään koko tietokannan päivittävään tiedostoon polku moduulin master.xml-tiedostoon. Kuvassa 18 nähdään koko hakemistorakenne tiedostoineen valmiina testikäyttöön. Kuva on otettu TortoiseSVN:n commit-näkymästä.



Kuva 18. Hakemistorakenne tiedostoineen lisättiin versionhallintaan.

UpdateSTSDatabase.xml on siis ylin skripti, joka sisältää viittaukset kaikkiin haaran master.xml-tiedostoihin, ja sen polku annetaan Liquibaselle tietokannan päivityksen yhteydessä. Esimerkkikoodissa 3 on esitetty yksinkertainen sisältö, jolta updateSTSDatabase.xml voisi näyttää. Esimerkki sisältää viittaukset yhteiseen sekä kahden moduulin master.xml-tiedostoihin.

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

            <include file="3.4/master.xml" relativeToChangelogFile="true"/>
            <include file="../../Externals/AlertList/database/Liquibase/3.4/mas-
            ter.xml" relativeToChangelogFile="true"/>
            <include file="../../Externals/MIFID2/database/Liquibase/3.4/mas-
            ter.xml" relativeToChangelogFile="true"/>

</databaseChangeLog>
```

Esimerkkikoodi 3. UpdateSTSDatabase.xml-skriptin sisältö.

Tässä ylimmässä skriptissä vain osoitetaan Liquibaselle, mistä muutokset todellisuudessa löytyvät. Polusta 3.4 löytyy ensimmäinen päivityksiä ajava master.xml-tiedosto. Esimerkkikoodissa 4 on esitetty sen sisältö.

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

           <include file="schema/create_DB_VC_TEST_TABLE.xml" relativeToChan
           gelogFile="true"/>
           <include file="data/insert_DB_VC_TEST_TABLE_data.xml" relativeTo-
           ChangelogFile="true"/>

</databaseChangeLog>
```

Esimerkkikoodi 4. Polun 3.4/master.xml-tiedoston sisältö.

Master-tiedostot sisältävät taas polut varsinaisiin muutoksia tekeviin skripteihin eli changelogeihin. Esimerkkikoodissa 5 on esitetty DB_VC_TEST_TABLE-taulun luonti XML-muodossa.

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

    <changeSet id="1" author="joona">
        <createTable tableName="DB_VC_TEST_TABLE">
            <column name="message" type="varchar(255)"/>
        </createTable>
    </changeSet>

</databaseChangeLog>
```

Esimerkkikoodi 5. Taulun luonti XML-muodossa.

Moduulien tietokannan muutosten tekeminen menee täysin vastaavalla tavalla. Master-skriptit ovat mukana siksi, ettei moduulia lisätessä tarvitse viitata jokaiseen skriptiin erikseen, vaan viittaus master-skriptiin riittää. Master-skriptiin on toki lisättävä viittaus lisättyihin muutoksiin.

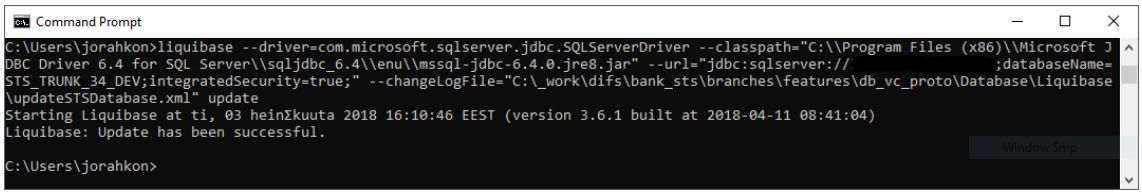
5.3 Muutosten hallinta

Muutosten hallinta tapahtuu Liquibasella changelog-tiedostojen kautta. Niihin kirjoitetaan muutosryhmiä, eli changesetejä XML-muodossa. Liquibase kääntää ne valitulle tietokantamoottorille sopivaan SQL-muotoon. Testiympäristössä skeemaan muutoksia tekevät tiedostot menevät kansioon schema ja referenssidataan muutoksia tekevät tiedostot kansioon data. Liquibasea kutsutaan esimerkkikoodissa 6 esitellyllä komennolla. Sille annetaan tietokanta-ajurin tiedostopolku, tietokantapalvelimen osoite, tietokannan nimi ja asetetaan Windows-autentikaatio päälle, jotta palvelimelle ei tarvitse erikseen kirjautua, polku tietokannan päivittävään skriptiin ja päivityskomento.

```
liquibase
      --driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
      --classpath="C:\\Program Files (x86)\\Microsoft JDBC Driver 6.4 for SQL
      Server\\sqljdbc_6.4\\enu\\mssql-jdbc-6.4.0.jre8.jar"
      --url="jdbc:sqlserver://*****;database
      Name=STS_TRUNK_34_DEV;integratedSecurity=true;"
      --changeLogFile="C:\_work\difs\bank_sts\branches\features\db_vc_proto\Da
      tabase\Liquibase\updateSTSDatabase.xml"
      update
```

Esimerkkikoodi 6. Liquibasella muutosten vieminen kantaan.

Päivityskomennon antamisen jälkeen Liquibase tarkastaa tietokannan DATABASECHANGELOG-taulun ja toteuttaa päivitykset, joita ei ole vielä ajettu kantaan. Kuvassa 19 nähdään, miltä komentorivi näyttää onnistuneen päivityksen jälkeen.



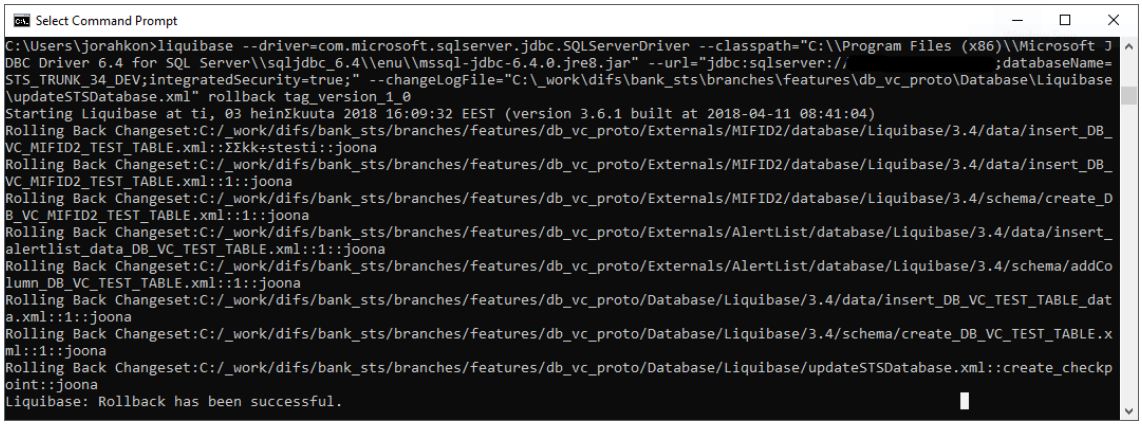
Kuva 19. Tietokannan päivitys onnistui.

Halutut muutokset päivittyivät tietokantaan ja tieto niiden ajamisesta tallentuu DATABASECHANGELOG-tauluun, kuten kuvasta 20 nähdään.

Kuva 20. Kohdetietokannan DATABASECHANGELOG-taulu päivityksen jälkeen.

Tietokannan päivittäminen ja muutosten vienti kantaan onnistui yhdellä komennolla turvallisesti skriptien kautta suoraan kehittäjän tietokoneelta ilman 30 minuuttia kestävän jatkuvan koonnin käännöksen valmistumisen odottamista. Onnistuneen päivityksen jälkeen skriptit voidaan viedä versionhallintaan muiden kehittäjien saataville.

Rollback-ominaisuus toimii myös erittäin näppärästi. Skriptien alussa luotiin tietokantatagi tag_version_1_0, johon palaaminen tapahtuu korvaamalla update-komento rollback-komennolla ja lisäämällä tagi sen perään. Kuvassa 21 havainnollistetaan rollback-ominaisuutta.



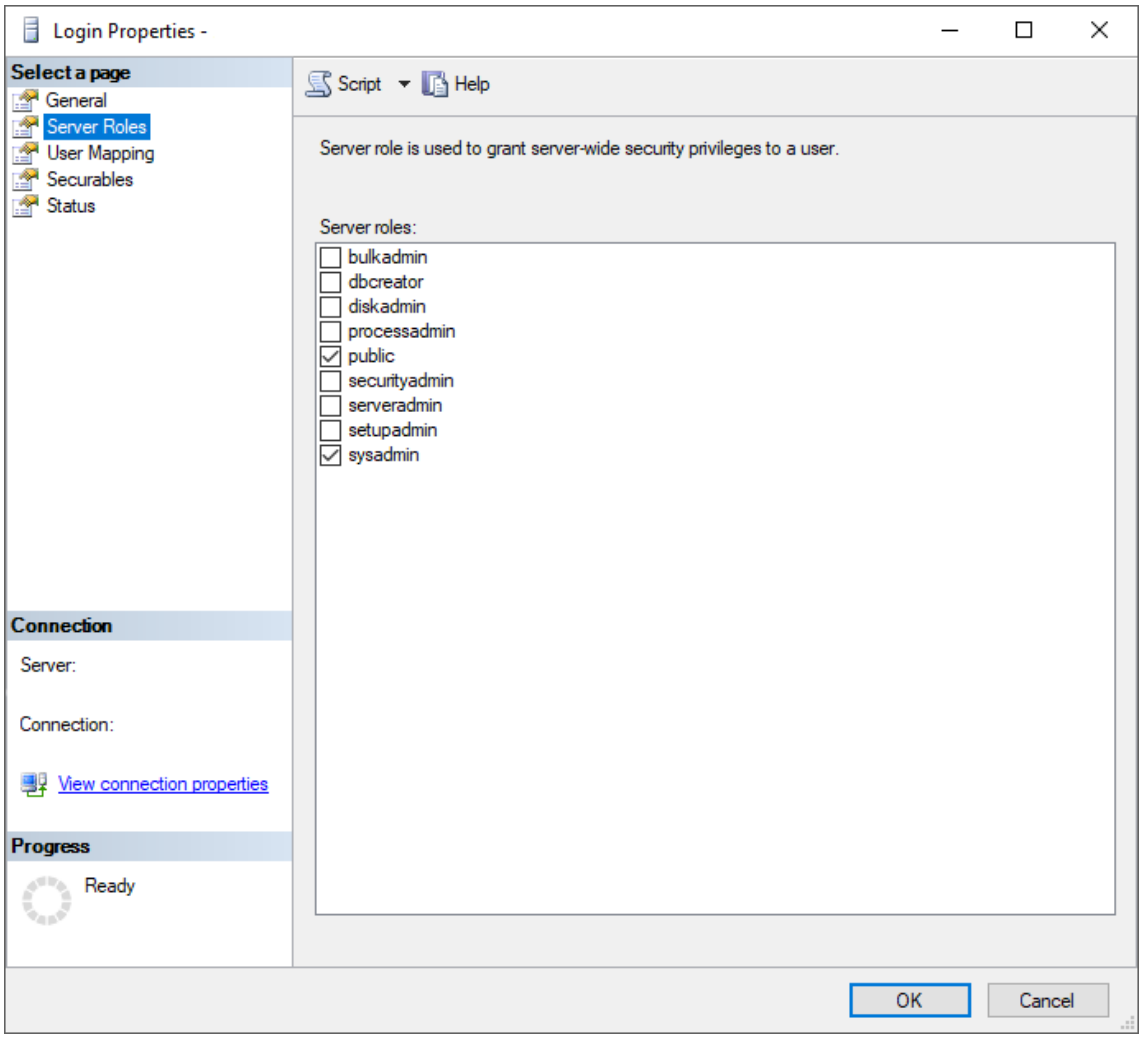
Kuva 21. Liquibasen rollback-ominaisuuden esittelyä.

Tietokanta palaa samaan tilaan kuin ennen päivityksen tekemistä. Skriptien testailuun on myös tarjolla kätevä komento updateTestingRollback, joka päivittää tietokannan, palauttaa sen takaisin tilaan ennen päivitystä ja lopuksi vielä päivittää tietokannan uudelleen [25]. Näin varmistetaan, että version palautus todella toimii ennen skriptien viemistä versionhallintaan.

5.4 Integroiminen jatkuvan koonnin ympäristöön

Liquibasen integroiminen jatkuvan koonnin ympäristöön, TeamCityyn, vaatii useita toimenpiteitä, ja valmisteluja vaaditaan sekä TeamCityn että Liquibasen päässä. TeamCity suorittaa Liquibasen sen agenttipalvelimelta käyttäen Windows-autentikaatiota, joten tie-

tokantapalvelimelle on lisättävä sysadmin-rooli TeamCityn käyttäjälle. Näin agentti pystyy tekemään tietokantaan muutoksia samaan tapaan kuin kehittäjä. Kuvassa 22 nähdään roolin asettaminen SQL Server Management Studiolla.



Kuva 22. Sysadmin-roolin asettaminen SQL Server Management Studiolla.

Liquibase käyttää yhtenä tunnistetietona ajetuille skripteille niiden polkua ja tiedostonimeä, joten polku toisiin skripteihin ja changelogiin täytyy antaa relatiivisesti, ei absoluuttisesti. Muuten esimerkiksi kehittäjän koneelta ajettu päivitys tunnistautuu kokonaan uudeksi päivitykseksi, ja DATABASECHANGELOG-taulu menee sekaisin. TeamCity ottaa uusimmat tiedostot Subversionista automaattisesti agenttikoneelle samaan tapaan kuin kehittäjä ottaa omalle koneelleen, joten relatiiviset polut tiedostosta toiseen toimivat, kunhan Liquibase suoritetaan samasta sijainnista joka kerta.

Tätä silmällä pitäen Liquibasen update-komennolle kirjoitettiin batch-skripti, joka ajetaan ympäristön kokoavan MSBuild-tiedoston kautta. Näin päivitys tapahtuu aina samasta kansiosta eikä tiedostopolkujen kanssa tule ongelmia. Batch-skriptin sisältö näkyy esimerkkikoodissa 7.

```
@echo off
cmd /k liquibase --driver=com.microsoft.sqlserver.jdbc.SQLServerDriver --
classpath="C:\Program Files (x86)\Microsoft JDBC Driver 6.4 for SQL
Server\sqljdbc_6.4\enu\mssql-jdbc-6.4.0.jre8.jar" --
url="jdbc:sqlserver://*****;databaseName=STS_TRUNK_34_DEV;integratedSecu-
rity=true;"
--changeLogFile="..\..\..\Database\Liquibase\UpdateDatabase.xml" update
```

Esimerkkikoodi 7. Tietokannan päivittävän skriptin sisältö.

Tietokannan päivittäminen kehittäjän koneelta onnistuu samalla tiedostolla, joten jatkuvan koonnin ympäristöä ei tarvitse kääntää jokaisen tietokantamuutoksen takia, jos ei niin halua. MSBuild-tiedostoa muokattiin kutsumaan tätä skriptiä jokaisen koonnin yhteydessä, ja jos tietokanta on päivitetty etukäteen, ei Liquibase tee mitään. Esimerkkikoodissa 8 nähdään MSBuild-tiedostoon lisätty kutsu.

```
<Target Name="UpdateDatabaseWithLiquibase">
     <Exec Command="call $(RootFolder)\Build\DeploymentScripts\Li-
     quibase\UpdateDatabase.bat"/>
</Target>
```

Esimerkkikoodi 8. MSBuild-tiedostoon lisätty kutsu.

Kehityshaarat saadaan näillä toimenpiteillä päivitettyä kätevästi, mutta toimitushaaroihin täytyi kehittää ratkaisu, jolla tietokantaskriptit saadaan mukaan toimituspaketteihin automaattisesti. Tätä varten kirjoitettiin toinen batch-skripti. Tietokannan tila tarkastetaan Liquibasella ja jos tietokantaan on tullut muutoksia, Liquibase generoi muutoksista tietokannan päivittävän SQL-skriptin sekä samat muutokset peruuttavan SQL-skriptin. Tämän jälkeen molemmat tietokantaskriptit lisätään versionhallintaan TortoiseSVN:llä. MSBuild-tiedostoon lisättiin komento, joka kopioi kaikki siihen mennessä generoidut SQL-skriptit TeamCityn kokoamaan pakettiin omaan kansioonsa. Ideana on, että edellisen toimituksen jälkeen generoidut SQL-skriptit pidetään versionhallinnassa omassa kansiossaan tallessa, jotta toimitusta tehdessä tietokannan oikeaan versioon päivittävät tiedostot saadaan helposti mukaan. Tämän toteuttavan skriptin sisältö on nähtävissä liitteessä 1. Päivitysskripti generoidaan komennolla updateSQL ja peruutusskripti komennolla futureRollbackSQL. Generoivaa skriptiä on kutsuttava ennen päivittävää, jotta Liquibase voi generoida SQL-skriptit.

Kuvassa 23 nähdään TeamCityn käännöksen loki, josta ilmenee, että SQL-skriptit on generoitu ja lisätty versionhallintaan sekä tietokanta päivitetty.

```
[GenerateAndCommitScripts] Exec  (13s)
    [Exec] call E:\BuildAgent\work\STS_db_vc_proto\Externals\Build\STS\..\..\..\Build\DeploymentScripts\Liquibase\GenerateAndCommitScripts.bat
    [Exec] Update and rollback SQL script generator
    [Exec] ----------------------------------------
    [Exec] Database is not up to date, generating scripts.
    [Exec] Starting Liquibase at ti, 10 heinäkuuta 2018 15:58:41 EEST (version 3.6.1 built at 2018-04-11 08:41:04)
    [Exec] Update script generated.
    [Exec] Starting Liquibase at ti, 10 heinäkuuta 2018 15:58:45 EEST (version 3.6.1 built at 2018-04-11 08:41:04)
    [Exec] Rolling Back Changeset:../../../Database/Liquibase/../../Externals/AlertList/database/Liquibase/3.4/data/insert_alertlist_data_DB_VC
    [Exec] Rollback script generated.
    [Exec] A         E:\BuildAgent\work\STS_db_vc_proto\Database\Liquibase\DatabaseIncrementalScripts\STS.Ver3.4.101509_update.sql
    [Exec] A         E:\BuildAgent\work\STS_db_vc_proto\Database\Liquibase\DatabaseIncrementalScripts\STS.Ver3.4.101509_rollback.sql
    [Exec] Adding          Database\Liquibase\DatabaseIncrementalScripts\STS.Ver3.4.101509_rollback.sql
    [Exec] Adding          Database\Liquibase\DatabaseIncrementalScripts\STS.Ver3.4.101509_update.sql
    [Exec] Transmitting file data ..done
    [Exec] Committing transaction...
    [Exec] Committed revision 101512.
    [Exec] Scripts added to version control.
    [Exec] Press any key to continue . . .
ploy] CallTarget  (3s)
[CallTarget] UpdateDatabaseWithLiquibase  (3s)
[UpdateDatabaseWithLiquibase] Exec  (3s)
    [Exec] call E:\BuildAgent\work\STS_db_vc_proto\Externals\Build\STS\..\..\..\Build\DeploymentScripts\Liquibase\UpdateDatabase.bat
    [Exec] Starting Liquibase at ti, 10 heinäkuuta 2018 15:58:49 EEST (version 3.6.1 built at 2018-04-11 08:41:04)
    [Exec] Liquibase: Update has been successful.
```

Kuva 23. TeamCityn loki tietokannan päivittämisestä.

SQL-skriptit kopioituvat TeamCityn kokoamaan pakettiin eli artefakteihin yhdessä ohjelmiston binääritiedostojen kanssa. Kuvassa 24 nähdään esimerkkitapauksen artefaktit, joissa SQL-skriptit ovat mukana.

Kuva 24. TeamCityn kokoamat artefaktit.

Liitteessä 2 nähdään tiedoston STS.Ver3.4.101505_update.sql sisältö esimerkkinä, miltä ulostulevat SQL-skriptit näyttävät. Asiakkaan tietokantaan on luotava taulut DATA-

BASECHANGELOG ja DATABASECHANGELOGLOCK, jotta skriptit toimivat. Liquibasen komentojen perässä on annettava Javan systeemiparametri -DoutputFileEncoding=UTF-8, jotta uloskirjoitettavien tiedostojen merkistökoodaus on muotoa UTF-8, jolloin skandinaaviset erikoismerkit toimivat generoiduissa tiedostoissa.

TeamCityn agenttipalvelimelle täytyi asentaa TortoiseSVN, jotta SVN-komennot olivat käytettävissä komentoriviltä ja generoitujen tiedostojen lisääminen versionhallintaan onnistui, sillä TeamCity käyttää tiedostojen hakuun palvelimelta sisäänrakennettua SVNKit-ohjelmistoa, jota ei voi käyttää TeamCityn ulkopuolelta. Lisäksi TeamCityn jatkuvan koonnin konfiguraatioon täytyi lisätä versionhallinnan käyttäjätunnus ja salasana ympäristömuuttujiin, kuten kuvasta 25 nähdään, sillä SVN ei tue Windowsin tunnistautumista.

Kuva 25. Versionhallinnan käyttäjätunnus ja salasana on asetettava TeamCityn ympäristömuuttujiin.

5.5 Tulokset

Työn tuloksena varmistui, että Liquibase toimii yhdessä DiFSin muiden työkalujen kanssa. Yksinkertaisessa tilanteessa se soveltuu tietokannan päivittämiseen sekä toimituspaketin päivitysskriptien generointiin. Tietokannan päivittäminen onnistuu helpommin kuin pelkkien SQL-skriptien kanssa toimiessa, ja seuraavan toimituksen SQL-skriptit saadaan kerättyä kätevästi yhdestä paikasta, joten työn määrä vähenee toimituspakettia koottaessa huomattavasti. Liquibase mahdollistaa myös tietokannan palauttamisen edelliseen versioon, joka on toivottu ominaisuus. Haasteena on kuitenkin edelleen tietokantojen erittäin suuri koko, joten paikalliset tietokannat eivät ole mahdollisia kehityksessä. Esimerkiksi työssä käytetyn tietokannan koko on 155 GB. Ongelmaa voi kiertää perustamalla itselleen palvelimelle pienen muutaman taulun sisältävän tietokannan ja

kokeilla omia päivityksiä sekä edelliseen versioon palaamista tuossa tietokannassa ennen yhteisen kehitystietokannan päivittämistä. Näin vältetään yhteisen tietokannan menemistä epäkuntoon.

6 Yhteenveto

Insinöörityössä tutustuttiin teorian kautta versionhallintaan yleisesti ja syvennyttiin erityisesti tietokannan versionhallintaan. Tavoitteena oli selvittää Digia Financial Solutions -organisaation tapaa hallita ja toimittaa muutoksia tietokantoihin ja löytää sopiva tietokannan versionhallintaohjelmisto organisaation käyttöön sekä havainnollistaa käytännön avulla sen toimintaa. Tietokantojen versionhallinnassa on ollut kehitystarpeita, jotka ilmenevät erityisesti, kun tietokantamuutoksia toimitetaan asiakkaille. Sopivia tietokannan versionhallintaohjelmistoja vertailtiin niiden ominaisuuksien perusteella ja päädyttiin valitsemaan kokeiluun potentiaalisin vaihtoehto: Liquibase. Liquibase asennettiin sekä palvelimelle että paikallisesti ja se otettiin käyttöön testiympäristössä jatkuvan integraation ympäristö mukaan lukien.

Liquibasen avulla muutosten tekeminen tietokantaan helpottui, ja toimitushaaroja varten kehitetty skripti, joka tulostaa SQL-käskyt tiedostoihin, auttaa huomattavasti toimituksia tehtäessä. Tästä johtuen toimituspaketin tekeminen on nopeampaa ja käyttäjäystävällisempää kuin ennen. Työn tavoitteet saavutettiin siis varsin hyvin. Parannettavaa olisi paikallisten kehitystietokantojen käyttöönotto, mutta se vaatisi kevyempiä versioita nykyisistä tietokannoista, sillä käytössä olevat tietokannat ovat jopa 155 GB:n kokoisia niiden sisältämän referenssi- ja testidatan vuoksi. Liquibase ei myöskään pysty tekemään näin laajoista tietokannoista suoraan changelogeja, eli se on otettava käyttöön tästä hetkestä lähtien, ei takautuvasti.

Liquibasea kokeiltiin yksinkertaisessa tilanteessa ja siinä se toimi hyvin. Seuraava vaihe on jatkokehittää tätä prosessia ja varmistaa, että se soveltuu myös reaalimaailman poikkeustilanteisiin. Tämän jälkeen Liquibase voidaan ottaa käyttöön varsinaisessa pilottiprojektissa, jossa kokonainen tiimi opettelee sen käyttämisen ja työkalu pääsee monipuolisempaan testaukseen tositoimissa. Samalla tässä työssä muutostiedostojen säilytykseen käytettyä hakemistorakennetta voidaan kehittää parempaan suuntaan sekä hioa toimitushaarojen muutosskriptien tallentamista versionhallintaan.

Lähteet

1 Digia yrityksenä. 2018. Verkkoaineisto. Digia Oyj. <http://digia.fi/yritys/>. Luettu 4.6.2018.

2 Somasundaram, Ravishankar. 2013. Git: Version Control for Everyone. Birmingham: Packt Publishing.

3 Gyerik, Jason. 2013. Bazaar Version Control. Birmingham: Packt Publishing.

4 Chacon, Scott & Straub, Ben. 2014. Pro Git. 2nd Edition. New York: Apress.

5 Ernst, Michael. 2012. Version control concepts and best practices. Verkkoaineisto. University of Washington. <https://homes.cs.washington.edu/~mernst/advice/version-control.html>. Päivitetty 3.3.2018. Luettu 7.6.2018.

6 Duggan, Kate. 2017. Six reasons to version control your database. Verkkoaineisto. Redgate. <https://www.red-gate.com/blog/database-devops/database-version-control-3>. Luettu 8.6.2018.

7 Allen, K. Scott. 2008. Three rules for Database Work. Verkkoaineisto. Ode to Code. <https://enterprisecraftsmanship.com/2015/08/10/database-versioning-best-practices>. Luettu 11.6.2018.

8 Khorikov, Vladimir. 2015. Database versioning best practices. Verkkoaineisto. Enterprise Craftmanship. <https://enterprisecraftsmanship.com/2015/08/10/database-versioning-best-practices>. Luettu 11.6.2018.

9 Khorikov, Vladimir. 2015. State vs migration-driven database delivery. Verkkoaineisto. Enterprise Craftmanship. <https://enterprisecraftsmanship.com/2015/08/18/state-vs-migration-driven-database-delivery/>. Luettu 11.6.2018.

10 16 Database Version Control tools. Verkkoaineisto. <https://dbmstools.com/version-control-tools>. Luettu 11.6.2018.

11 Fowler, Martin & Sadalage, Pramod. 2016. Evolutionary Database Design. Verkkoaineisto. ThoughtWorks. <https://martinfowler.com/articles/evodb.html>. Luettu 12.6.2018.

12 Foster, C. Elvis & Godbole, Shripad. 2016. Database Systems: A Pragmatic Approach. Second Edition. Berkeley: Apress.

13 SQL Server 2017. 2018. Verkkoaineisto. Microsoft. <https://www.microsoft.com/fi-fi/sql-server/sql-server-2017>. Luettu 13.6.2018.

14 Stansfield, Josh. 2014. Microsoft SQL Server vs. Oracle: The Same, But Different? Segue Technologies. <https://www.seguetech.com/microsoft-sql-server-vs-oracle-same-different>. Luettu 13.6.2018.

15 Introducing Oracle Database 18c: Oracle white paper. 2018. Verkkoaineisto. Oracle. <http://www.oracle.com/technetwork/database/oracledatabase18c-wp-4392576.pdf>. Luettu 13.6.2018.

16 Redgate. 2018. Verkkoaineisto. <https://www.red-gate.com>. Luettu 18.6.2018.

17 DBmaestro. 2018. Verkkoaineisto. <https://www.dbmaestro.com>. Luettu 18.6.2018.

18 Flyway. 2018. Verkkoaineisto. Boxfuse. <https://flywaydb.org>. Luettu 18.6.2018.

19 Liquibase. 2018. Verkkoaineisto. Datical. <https://www.liquibase.org>. Luettu 18.6.2018.

20 First Steps: Command-line. 2018. Verkkoaineisto. Boxfuse. <https://flywaydb.org/getstarted/firststeps/commandline>. Luettu 19.6.2018.

21 How Flyway works. 2018. Verkkoaineisto. Boxfuse. <https://flywaydb.org/getstarted/how>. Luettu 19.6.2018.

22 The liquibase.properties configuration file. 2018. Verkkoaineisto. Datical. <http://www.liquibase.org/documentation/liquibase.properties.html>. Luettu 19.6.2018.

23 Liquibase: Quick Start. 2018. Verkkoaineisto. Datical. <http://www.liquibase.org/quickstart.html>. Luettu 19.6.2018.

24 Liquibase: User and Developer Community. 2018. Datical. <https://www.liquibase.org/community/index.html>. Luettu 19.6.2018.

25 Liquibase Command Line. 2018. Datical. <https://www.liquibase.org/documentation/command_line.html>. Luettu 3.7.2018.

Toimitettavien SQL-skriptien generointi

SQL-skriptit generoidaan seuraavanlaisella batch-skriptillä.

```
@echo off
echo Update and rollback SQL script generator
echo ----------------------------------------

liquibase --driver=com.microsoft.sqlserver.jdbc.SQLServerDriver --
classpath="C:\Program Files (x86)\Microsoft JDBC Driver 6.4 for SQL
Server\sqljdbc_6.4\enu\mssql-jdbc-6.4.0.jre8.jar" --
url="jdbc:sqlserver://*****;databaseName=STS_TRUNK_34_DEV;integratedSecu-
rity=true;" --changeLogFile="..\..\..\Database\Liquibase\UpdateDatabase.xml"
status | findstr /C:"is up to date"

if %errorlevel% == 0 (

     echo Database is up to date, no scripts are generated.
     pause

) else (

     echo Database is not up to date, generating scripts.

     liquibase --driver=com.microsoft.sqlserver.jdbc.SQLServerDriver --
     classpath="C:\Program Files (x86)\Microsoft JDBC Driver 6.4 for SQL
     Server\sqljdbc_6.4\enu\mssql-jdbc-6.4.0.jre8.jar" --
     url="jdbc:sqlserver://*****;databaseName=STS_TRUNK_34_DEV;integratedSecu-
     rity=true;" --outputFile=..\..\..\Database\Liquibase\DatabaseIncremen-
     talScripts\STS.Ver3.4.%build_vcs_number%_update.sql --changeLog-
     File="..\..\..\Database\Liquibase\UpdateDatabase.xml"
     updateSQL -Dliquibase.outputFileEncoding=UTF-8

     echo Update script generated.

     liquibase --driver=com.microsoft.sqlserver.jdbc.SQLServerDriver --
     classpath="C:\Program Files (x86)\Microsoft JDBC Driver 6.4 for SQL
     Server\sqljdbc_6.4\enu\mssql-jdbc-6.4.0.jre8.jar" --
     url="jdbc:sqlserver://*****;databaseName=STS_TRUNK_34_DEV;integratedSecu-
     rity=true;" --outputFile=..\..\..\Database\Liquibase\DatabaseIncremen-
     talScripts\STS.Ver3.4.%build_vcs_number%_rollback.sql --changeLog-
     File="..\..\..\Database\Liquibase\UpdateDatabase.xml"
     futureRollbackSQL -Dliquibase.outputFileEncoding=UTF-8

     echo Rollback script generated.

     svn add ..\..\..\Database\Liquibase\DatabaseIncrementalScripts\ --force
     svn commit ..\..\..\Database\Liquibase\DatabaseIncrementalScripts\ -m
     "STS.Ver3.4.%build_vcs_number% SQL scripts" --username=%vcs_username% --
     password=%vcs_password%

     echo Scripts added to version control.
     pause
)
```

Tietokannan päivittävä SQL-skripti

Esimerkki generoidusta tietokannan päivittävästä SQL-skriptistä.

```
-- ********************************************************************
-- Update Database Script
-- ********************************************************************
-- Change Log: ../../../Database/Liquibase/UpdateDatabase.xml
-- Ran at: 10.7.2018 14:54
-- Against: *****@jdbc:sqlserver://*****;authentication=NotSpecified;authenti-
cationScheme=nativeAuthentication;xopenStates=false;send-
TimeAsDatetime=true;trustServerCertificate=false;TransparentNetworkIPResolu-
tion=true;serverNameAsACE=false;sendStringParametersAsUnicode=true;select-
Method=direct;responseBuffering=adaptive;packetSize=8000;multiSubnetFailo-
ver=false;loginTimeout=15;lockTimeout=-1;lastUpdateCount=true;en-
crypt=false;disableStatementPooling=true;databaseName=STS_TRUNK_34_DEV;col-
umnEncryptionSetting=Disabled;applicationName=Microsoft JDBC Driver for SQL
Server;applicationIntent=readwrite;
-- Liquibase version: 3.6.1
-- ********************************************************************

USE STS_TRUNK_34_DEV;
GO

-- Lock Database
UPDATE DATABASECHANGELOGLOCK SET LOCKED = 1, LOCKEDBY = '*****(**.*.*.***)',
LOCKGRANTED = '2018-07-10T14:54:55.854' WHERE ID = 1 AND LOCKED = 0
GO

-- Changeset ../../../Database/Liquibase/UpdateDatabase.xml::cre-
ate_tag1::joona
INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID, TAG) VALUES ('create_tag1', 'joona', '../../../Database/Liqui-
base/UpdateDatabase.xml', GETDATE(), 2, '8:b093da6e0bd66f95a8d21b547bfa40d3',
'tagDatabase', '', 'EXECUTED', NULL, NULL, '3.6.1', '1223697924', 'version1')
GO

-- Changeset ../../../Database/Liquibase/3.4/schema/create_DB_VC_TEST_TA-
BLE.xml::create DB_VC_TEST_TABLE::joona
CREATE TABLE DB_VC_TEST_TABLE (message varchar(255))
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('create DB_VC_TEST_TABLE', 'joona', '../../../Data-
base/Liquibase/3.4/schema/create_DB_VC_TEST_TABLE.xml', GETDATE(), 3,
'8:588557c9376ded5fdec898e6f2a504b4', 'createTable table-
Name=DB_VC_TEST_TABLE', '', 'EXECUTED', NULL, NULL, '3.6.1', '1223697924')
GO

-- Changeset ../../../Database/Liquibase/3.4/data/insert_DB_VC_TEST_TA-
BLE_data.xml::insert DB_VC_TEST_TABLE::joona
INSERT INTO DB_VC_TEST_TABLE (message) VALUES ('testataan')
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('insert DB_VC_TEST_TABLE', 'joona', '../../../Data-
base/Liquibase/3.4/data/insert_DB_VC_TEST_TABLE_data.xml', GETDATE(), 4,
```

```
'8:8bd02f284498a7ebaa49baa3942c89a5', 'insert tableName=DB_VC_TEST_TABLE', '',
'EXECUTED', NULL, NULL, '3.6.1', '1223697924')
GO

-- Changeset ../../../Database/Liquibase/../../Externals/AlertList/data-
base/Liquibase/3.4/schema/addColumn_DB_VC_TEST_TABLE.xml::add column
DB_VC_TEST_TABLE::joona
ALTER TABLE DB_VC_TEST_TABLE ADD AlertListMessage varchar(255)
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('add column DB_VC_TEST_TABLE', 'joona', '../../../Data-
base/Liquibase/../../Externals/AlertList/database/Liquibase/3.4/schema/addCol-
umn_DB_VC_TEST_TABLE.xml', GETDATE(), 5, '8:6426466fe846b7d887f645018e6f2da5',
'addColumn tableName=DB_VC_TEST_TABLE', '', 'EXECUTED', NULL, NULL, '3.6.1',
'1223697924')
GO

-- Changeset ../../../Database/Liquibase/../../Externals/AlertList/data-
base/Liquibase/3.4/data/insert_alertlist_data_DB_VC_TEST_TABLE.xml::insert
DB_VC_TEST_TABLE::joona
INSERT INTO DB_VC_TEST_TABLE (message, AlertListMessage) VALUES ('AlertList
insert', 'Message from AlertList module')
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('insert DB_VC_TEST_TABLE', 'joona', '../../../Data-
base/Liquibase/../../Externals/AlertList/database/Liquibase/3.4/data/in-
sert_alertlist_data_DB_VC_TEST_TABLE.xml', GETDATE(), 6,
'8:5e0e24471fc5b85382e90d4090952af7', 'insert tableName=DB_VC_TEST_TABLE', '',
'EXECUTED', NULL, NULL, '3.6.1', '1223697924')
GO

-- Changeset ../../../Database/Liquibase/../../Externals/MIFID2/data-
base/Liquibase/3.4/schema/create_DB_VC_MIFID2_TEST_TABLE.xml::create DB_VC_MI-
FID2_TEST_TABLE::joona
CREATE TABLE DB_VC_MIFID2_TEST_TABLE (id int IDENTITY (1, 1) NOT NULL, message
varchar(255), CONSTRAINT PK_DB_VC_MIFID2_TEST_TABLE PRIMARY KEY (id))
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('create DB_VC_MIFID2_TEST_TABLE', 'joona',
'../../../Database/Liquibase/../../Externals/MIFID2/database/Liqui-
base/3.4/schema/create_DB_VC_MIFID2_TEST_TABLE.xml', GETDATE(), 7,
'8:47517f54a2a215225e9eab6f34e23821', 'createTable tableName=DB_VC_MI-
FID2_TEST_TABLE', '', 'EXECUTED', NULL, NULL, '3.6.1', '1223697924')
GO

-- Changeset ../../../Database/Liquibase/../../Externals/MIFID2/data-
base/Liquibase/3.4/data/insert_DB_VC_MIFID2_TEST_TABLE.xml::MIFID2 in-
sert::joona
INSERT INTO DB_VC_MIFID2_TEST_TABLE (message) VALUES ('MIFID2 insert')
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES ('MIFID2 insert', 'joona', '../../../Database/Liqui-
base/../../Externals/MIFID2/database/Liquibase/3.4/data/insert_DB_VC_MI-
FID2_TEST_TABLE.xml', GETDATE(), 8, '8:a0ae62257ceac62a1798860f438eaaae', 'in-
sert tableName=DB_VC_MIFID2_TEST_TABLE', '', 'EXECUTED', NULL, NULL, '3.6.1',
'1223697924')
GO
```

```
-- Changeset ../../../Database/Liquibase/../../Externals/MIFID2/data-
base/Liquibase/3.4/data/insert_DB_VC_MIFID2_TEST_TABLE.xml::ääkköstesti::joona
INSERT INTO DB_VC_MIFID2_TEST_TABLE (message) VALUES (N'ääkkösiä')
GO

INSERT INTO DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXE-
CUTED, MD5SUM, DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE,
DEPLOYMENT_ID) VALUES (N'ääkköstesti', 'joona', '../../../Database/Liqui-
base/../../Externals/MIFID2/database/Liquibase/3.4/data/insert_DB_VC_MI-
FID2_TEST_TABLE.xml', GETDATE(), 9, '8:1abffc176354eb87789421f04f38ce2f', 'in-
sert tableName=DB_VC_MIFID2_TEST_TABLE', '', 'EXECUTED', NULL, NULL, '3.6.1',
'1223697924')
GO

-- Release Database Lock
UPDATE DATABASECHANGELOGLOCK SET LOCKED = 0, LOCKEDBY = NULL, LOCKGRANTED =
NULL WHERE ID = 1
GO
```