Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Diplomityö

Pertti Föhr

TIETOKANTOJEN YHDISTÄMINEN JÄRJESTELMÄN RAJAPINTOJA HYÖDYNTÄEN

Työn tarkastaja(t): Professori Kari Smolander
Vastaava informaatikko Markku Laitinen

Työn ohjaaja(t): Vastaava informaatikko Markku Laitinen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Pertti Föhr

Tietokantojen yhdistäminen järjestelmän rajapintoja hyödyntäen

Diplomityö
2013
55 sivua, 20 kuvaa, 1 taulukko, 3 liitettä

Työn tarkastajat: Professori Kari Smolander
Vastaava informaatikko Markku Laitinen
Hakusanat: rajapinnat, ohjelmointi, Visual Studio, VBA, Oracle, Microsoft Access
Keywords: interfaces, programming, Visual Studio, VBA, Oracle, Microsoft Access

Tässä työssä on pyritty kartoittamaan mahdollisuudet omatoimiseen Voyager-kirjastojärjestelmän aineistotietokantojen ja asiakasrekisterien yhdistelyyn. Lähtökohtana on ollut oletus, että kohdejärjestelmän tietokantaan ei ole oikeuksia eikä sopimusteknistä mahdollisuuttakaan kirjoittaa tietoja suoraan kyselykielellä. Järjestelmän dokumentaatiota sekä verkostoa hyödyntämällä olen pyrkinyt kartoittamaan mahdollisuudet kaiken toiminnallisuuden vaatiman datan siirtoon. Hyödyntämällä järjestelmän rajapintoja, voidaan saavuttaa kustannussäästöjä sekä joustavuutta työn suorittamisen aikataulutukseen.

Bibliografisen datan siirtoon Voyager-kirjastojärjestelmässä on mahdollisuus hyödyntää palvelimella eräajona suoritettavaa ohjelmaa. Tässä eräajossa voidaan siirtää sekä bibliografiset tietueet että varastotietueet. Nidetietojen kirjoittamiseksi kohteena olevaan tietokantaan käytetään Visual Studio -sovellusta, joka hyödyntää luettelointirajapintaa. Asiakastietojen siirtoon on mahdollista hyödyntää palvelimella suoritettavaa eräajoa, jonka syötteeksi kirjoitetaan määrämittainen syötetiedosto. Asiakastietueisiin sidotut lainatiedot voidaan siirtää kohdetietokantaan asiakasohjelman offline-lainaustoiminnolla.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Pertti Föhr

Merging databases using systems interfaces

Master's Thesis
2013
55 pages, 20 figures, 1 tables, 3 appendices

Examiners: Professor Kari Smolander
Chief information specialist Markku Laitinen
Keywords: interfaces, programming, Visual Studio, VBA, Oracle, Microsoft Access

The aim of this thesis is to find possibilities to merge two Voyager databases using system's interfaces. There are no writing rights to use SQL-queries to write data to database. Also, from a risk management point of view, writing to database should only be done using system's own interfaces. Studying user manuals and previous similar works on other libraries, I've been trying to solve functionalities for database merge. By using system's interfaces, it is possible to reduce expenses and achieve more flexibility for scheduling database project.

For bibliographic data, it is possible to use server side batch processing applications. Using batch processing application, both bibliographic and holdings records can be processed. In this database project, Visual Basic -application was developed to process item records. This application calls system's cataloguing interface to function with library system. To manage patron data, server side batch processing application can be used. Patron data between databases is transferred in system's standard interface format. For check in information processing, offline circulation functionality is useful.

ALKUSANAT

Tämä diplomityö on tehty Päijät-Hämeen koulutuskonsernissa, osana Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen ja Lahden tiedekirjaston yhdistämisprojektia. Haluan kiittää tietopalvelujohtaja Sirkku Blinnikkaa saamastani tuesta työtä tehdessäni. Lisäksi haluan kiittää professori Kari Smolanderia ja sekä vastaava informaatikko Markku Laitista työni ohjauksesta. Kiitän myös perhettäni kannustuksesta opintojen edistyessä.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

ANSI	American national standards institute
API	Application programming interface
ASCII	American standard code for information interchange
BASIC	Beginner's all-purpose symbolic instruction code
BLOB	Binary large object
CSV	Comma-separated values
DLL	Dynamic link libraries
ER	Entity relationship
FUAS	Federation of Universities of Applied Sciences
ISBN	International standard book number
MARC	Machine readable cataloging
MS	Microsoft
ODBC	Open database connectivity
OPAC	Online public access catalog
PDF	Portable document format
RFID	Radio frequency identification
SIF	Standard interface file
SQL	Structured query language
UKJ	Uusi kirjastojärjestelmä
VBA	Visual basic for applications
WWW	World wide web

1 JOHDANTO

1.1 Tausta

Lahden keskustassa on otettu käyttöön nykyaikainen oppimiskeskus. Oppimiskeskuksessa tuotetaan korkeakoulutoimintaan kuuluvia ydinpalveluja keskitetysti Fellmannin kiinteistöön sijoittuvasta palvelu- ja kehittämiskeskuksesta käsin. Palvelukokonaisuuden muodostavat mm. korkeakoulujen yhteiskirjasto sekä kokous- ja kongressipalvelut. Lahdessa on aikaisemmin toiminut kaksi korkeakoulukirjastoa. Lahden ammattikorkeakoulun tieto- ja kirjastopalvelut on tuottanut yhdessä Lahden ammattikorkeakoulun ja Koulutuskeskus Salpauksen kanssa palveluja oppimisen, opetuksen sekä tutkimus- ja kehitystoiminnan tarpeisiin. Lahden tiedekirjasto on ollut kaupungin kirjastoverkostoon kuuluva kirjasto, jonka on tuottanut kirjasto- ja tietopalvelut Lahden yliopistokeskukseen kuuluville yliopistoyksiköille. Vuoden 2011 aikana muodostettu korkeakoulujen yhteiskirjasto on Lahden ammattikorkeakoulun, Lahden yliopistokeskukseen kuuluvien yliopistojen ja Lahden kaupungin yhteistoimintasopimukseen perustuva tieto- ja kirjastopalvelu (Lahden yliopistokeskus, 2013).

Opetusministeriön asettaman korkeakoulukirjastojen rakenteellisen kehittämisen hankkeen tavoitteena on hahmotella korkeakoulukirjastojen organisoituminen uudistuvassa korkeakoululaitoksessa siten, että korkeakoulujen yhdistymiset ja yhteistyösopimukset tulevat huomioon otetuiksi kirjastoja vähentäen, yhteistyötä lisäten ja toimintoja tehostaen. (Korkeakoulukirjastojen rakenteellisen kehittämisen hanke, 2010)

Lahdessa toimivien korkeakoulukirjastojen toiminnot on yhdistetty oppimiskeskuksen kiinteistöön. Lahden tiedekirjaston bibliografinen data ja asiakasrekisteri on yhdistetty Lahden ammattikorkeakoulun tietokantaan (https://masto.amkit.fi). Tietokantojen yhdistämisen voi teettää järjestelmätoimittajalla (Ex Libris) ostopalveluna. Lahden ammattikorkeakoulun Tieto- ja kirjastopalvelujen projektiryhmässä kuitenkin päätettiin tehdä siirtoon liittyvät toimenpiteet itse. Tähän päätökseen oli suurin vaikutus yhdistämisen aikataulutuksen suuremmalla valinnanvapaudella.

Työn suoritus on aloitettu aikaisemmin vastaavia operaatioita suorittaneiden korkeakoulujen kartoittamisella. Tällaisia on mm. Northwestern yliopisto (Chicago) ja Tukholman yliopisto. Batchcat-rajapinta on järjestelmätoimittajan sovellukseen sisällyttämä luettelointirajapinta, jonka määrittelyssä Northwestern yliopisto on ollut merkittävässä roolissa. Projektin suunnitteluvaiheessa on selvitetty myös Northwestern yliopiston kehittämän rajapinnan käytettävyyttä tietojen lukemiseen lähdejärjestelmästä. Työn edistyessä päädyttiin kuitenkin hyödyntämään Voyager-kirjastojärjestelmän omia bibliografisen datan siirtoon tarkoitettuja rajapintoja. Aineistona tutustumisvaiheessa käytettiin molempien edellä mainittujen rajapintojen dokumentointia sekä lisäksi Voyager-kirjastojärjestelmän manuaaleja. Tietojen muokkauksen yhteydessä käytettiin myös bibliografisen datan sisällönkuvauksen dokumentointia, jonka ylläpidosta vastaa Kansalliskirjasto. Tiedonhankinnassa olen ollut yhteydessä myös Tukholman yliopiston kirjaston järjestelmästä vastaavaan henkilöön, jonka kanssa olen vaihtanut kokemuksia Tukholman yliopiston vastaavista operaatioista. Hankitun pohjatiedon perusteella on toteutettu tarvittavien sovellusten koodaus Visual Studio -ympäristössä.

1.2 Tavoitteet ja rajaukset

Tutkimuksella haetaan vastausta kysymykseen, kuinka Voyager-kirjastojärjestelmän bibliografiset tiedot ja asiakastiedot saadaan siirretyksi kahden tietokannan välillä turvallisesti ja tietokannan eheyden kärsimättä?

Osakysymykset:

- Kuinka data (bibliografinen ja asiakkaiden tiedot) luetaan lähdejärjestelmän tietokannasta?
- Kuinka data (bibliografinen ja asiakkaiden tiedot) saadaan kirjoitettua kohdejärjestelmän tietokantaan rajapintoja hyödyntäen?
- Kuinka evaluoidaan siirrettävät tiedot tuplatietueiden välttämiseksi?
- Kuinka siirrettävä data muokataan kohdejärjestelmän rajapintojen vaatimaan muotoon?

Työssä on tarkoitus tutkia ja luoda menetelmiä, joilla tuetaan kahden

korkeakoulukirjaston toiminnallista ja organisaatiollista integraatiota kirjastojärjestelmän näkökulmasta. Bibliografisen datan siirtämiseen hyödynnettävien rajapintojen toiminnallisuuden selvittämisen jälkeen kahden viitetietokannan väliin pitää rakentaa muunto-ohjelma, jolla sijaintitietueen voi muokata siirrettävästä tiedostosta. Toinen vaihtoehto on luoda marc21-formaatin mukainen data ”lennosta”. Voyager tietokannoissa on olemassa kaksi eri rakennetta sijaintitietojen esittämiseen asiakasliittymässä. Vaihtoehtoina on 1) jokaiselle niteelle erillinen sijaintitietue tai 2) yksi sijaintitietue nimekkeellä oleville niteille yhdessä toimipisteessä. Mikäli rakenne on sama sekä lähteessä että kohteessa, marc21-tietueen konvertoinnin määrä riippuu siitä, minkä verran kohdejärjestelmän toimipisteisiin kohdistuu yhdistämistoimenpiteitä. Nimekkeeseen liittyvä bibliografinen tietue, jossa on teoksen sisällönkuvailu, voidaan siirtää sellaisenaan. Korkeakoulukirjastoissa on tehty vuoden 2009 aikana bibliografisen datan konvertointi marc21fin-formaatista marc21-formaattiin. Tuossa yhteydessä Lahdessa toimivien korkeakoulukirjastojen bibliografinen data on konvertoitu yhteensopivaksi yhdistymistä silmällä pitäen. Lainatietojen siirtämiseen voidaan hyödyntää niin sanottua offline-lainausta. Kyseinen toiminto on Voyager-kirjastojärjestelmässä helpottamassa verkkoliikenteestä johtuvia toimintahäiriöitä. Yhteyden ollessa poikki palvelimeen, asiakasohjelma voidaan käynnistää yhteydettömään tilaan. Tuossa tilassa asiakasohjelma kirjoittaa määrämuotoisen tekstitiedoston työhakemistoon lainauksista ja palautuksista. Palvelinyhteyden muodostuessa uudelleen siirtotiedosto voidaan ladata tietokantaan. Lainojen siirtämiseen joudutaan luomaan ulosluku-ohjelma, jolla kyseisen formaatin mukainen siirtotiedosto voidaan viedä kohdejärjestelmästä lähdejärjestelmään. Asiakkaiden siirtämiseen Voyager-järjestelmässä on rajapinta, jota kautta voidaan luoda uusia asiakastietueita tai päivittää olemassa olevia asiakastietoja. Asiakasrajapinta vaatii määrämuotoisen SIF-siirtotiedoston (Standard Inrerface File). Siirtotiedoston luomiseen ohjelmoidaan rajapinnan kuvauksen mukaisen tiedoston luova uloslukuohjelma.

Tarkoituksena on luoda prosessi, jolla korkeakoulukirjastot voivat yhdistää kahden viitetietokannan tiedot turvallisesti ja taloudellisesti. Yhdistämisessä hyödynnetään Ex Libriksen toimittamia rajapintoja ja näiden rajapintojen toiminnasta vastaa Ex Libris. Sisällön tuottamisen ja vastuu sen oikeellisuuden kuuluu kirjastoille. Tässä yhdistämisessä siirrettävät tiedot kuuluvat kirjaston sisältövastuun piiriin kuten kirjaston päivittäisissä

prosesseissakin. Tällaisia prosesseja ovat mm. uusien viitetietojen luominen sekä asiakasrekisterin ylläpitäminen.

Laadunvarmistus bibliografisen datan ja asiakastietojen siirron yhteydessä pyritään toteuttamaan informaatikoiden suorittamalla työn seurannalla. Varsinkin sijaintietueiden luomisessa tarvittavien tietueiden oikeellisuus tarkistutetaan luettelointivastaavilta. Varsinaisen siirron yhteydessä tehdään siirrettävistä tietueista seurantatiedosto, josta voidaan tarkistaa siirrettyjen tietueiden siirtyminen oikein lähdetietokannasta kohdetietokantaan. Seurantatiedostoa tarvitaan myös, kun nidetietoja siirretään kohdetietokantaan. Muodostuvien tietueiden pitää kiinnittyä oikein bibliografiseen dataan ja tässä yhteydessä ei voi hyödyntää lähdetietokannan avaintietoja vaan on tiedettävä kohdetietokannan avainkentän arvo.

Työn tulosten tavoitteena on yksi Lahdessa toimivien korkeakoulutoimijoiden toimintaa tukeva kirjaston viitetietokanta ja kirjastojärjestelmä. Tavoitteena on myös tuottaa muiden korkeakoulujen käyttöön jaettava ohjeistus tietokantojen yhdistämiseen tai yksittäisten kokoelmien siirtämiseen tietokantojen välillä. Korkeakoulukirjastojen rakenteellisen kehityksen mukaisesti useassa tietokannassa saattaa tulla kyseeseen ainakin yksittäisten toimipisteiden siirtäminen tietokannasta toiseen.

Tässä diplomityössä on tarkoitus keskittyä Voyager-kirjastojärjestelmän tietokantojen yhdistämiseen. Työhön kiinteästi liittyvät toiminnot, kuten asiakasliittymän ja asiakasohjelmien toiminnallisuuden määrittely jätetään tästä työstä pois. Vaikka kyseiset toiminnot liittyvät tietokantojen yhdistämiseen, ne ovat kuitenkin enemmän palvelurajapinnan määrityksiä ja luonteeltaan jatkuvia. Lisäksi aineiston evaluointiprosessi jätetään tässä työssä maininnan arvoiseksi muilta osin kuin mitä työn valmistuminen vaatii. Kesän 2010 yhdistämisprojektissa siirretään lähdejärjestelmästä kaikki bibliografinen data. Tässä työssä aineiston läpikäynnillä on tarkoitus estää tuplatietueiden muodostuminen kohdejärjestelmässä.

1.3 Työn rakenne

Luvussa 1 käsitellään työn asettamisen taustaa korkeakoulukirjastojen rakenteellisen muutoksen näkökulmasta. Siinä kerrotaan, miksi omatoimiseen tietokantaoperaatioon on ryhdytty ja mitä ongelmia työssä on pyritty ratkaisemaan. Luvussa 2 käsitellään tietokantaoperaatioita teoreettisesta näkökulmasta kirjallisuuteen viitaten. Lisäksi luvussa esitellään työssä käytettäviä tekniikoita. Luvussa 3 esitellään työn edistyessä tehtyjä ratkaisuja. Siinä kerrotaan vaiheittain sekä bibliografisen datan että asiakasrekisterin muokkaukseen tarvittavia toimenpiteitä. Luvussa 4 kerrataan tärkeimmät tulokset sekä pohditaan työssä saavutettuja ratkaisuja tulevaisuuden näkökulmasta. Rakenteellisen muutoksen myötä tietokantoja yhdistellään myös tulevaisuudessa ja jossain määrin työn tuloksia voidaan hyödyntää myös seuraavassa järjestelmävaihdossa. Luvussa 5 esitellään vielä työn tuloksia ja vaikutuksia jälkitiivistelmässä.

2 TIETOKANTAOPERAATIOIDEN TAUSTAA

2.1 Tietokantaprojektin laadulliset ja taloudelliset vaikuttimet

Tässä luvussa käsittelen tietokantojen omatoimiseen yhdistämiseen vaikuttavia teorioita lähdekirjallisuuteen ja artikkeleihin nojautuen. Verrattaessa oman työn osuutta ostopalveluun, on tärkeää huomioida laadulliset sekä taloudelliset tekijät.

Tietokantojen omatoimisen yhdistelyn kannustimena voidaan nähdä taloudellisia vaikuttimia. Ekman-Sarkki (2006) on luetellut pienyrityksen näkökulmasta www-sivuston (World Wide Web) ostamisen ja oman ylläpidon etuja. Samoja vaikuttimia voidaan havaita myös tietokantaprojektin osalta. Artikkelin mukaan palvelun ostamisen etuja on muun muassa oman ajan säästö, toteutuksen nopeus ja vaivattomuus sekä alan asiantuntemuksen saatavuus (Ekman-Sarkki, 2006). Järjestelmätoimittajalla on tarjottavana yksittäisistä henkilöistä riippumaton palvelu, jossa on vakioidut toimenpiteet myös operaatioon valmistautuessa. Ostamisen haittapuolina artikkeli mainitsee toteutuksen kalliimman hinnan sekä palvelun vakiomuotoisuuden (Ekman-Sarkki, 2006). Ostopalvelua harkittaessa onkin mietittävä, mikä on oman ylläpitohenkilöstön osaamisen taso operaatioon valmistautumisen näkökulmasta sekä datan siirtämisen osalta. Käytännössä suljettujen järjestelmien omatoimiset datasiirrot tapahtuvat järjestelmätoimittajan rajapintoja hyödyntäen. Rajapintoihin perehtymisen ja niiden testaamisen voidaan olettaa vaikuttavan myös kokonaiskuluihin siihen käytetyn työajan muodossa. Jos organisaatiolla on olemassa olevaa osaamista järjestelmään liittyen ja ennestään rajapintoja hyödyntäviä palveluja, rajapintoihin perehtymisen ja niiden testaamisen kulut jäävät vähäisiksi. Leen, Pipinon, Funkin ja Wangin (2006) mukaan datan laadun parantamiseen tähtäävät toimet voidaan arvioida taloudellisesti yksinkertaisella kaavalla Arvo = hyöty - kulut. Tätä voidaan hyödyntää myös laatutekijöitä sisältävän tietokantaoperaation arvioimiseen. Mikäli arviointia haluaa tehdä, pitää miettiä millä tasolla kuluja ja hyötyjä halutaan arvioida. Yksinkertaisimmillaan hyötyä voidaan arvioida niin, että kaikki mikä voidaan tietokantaprojektissa tehdä olemassa olevilla ja joka tapauksessa käytössä olevilla resursseilla, on käytettävissä mm. aineistojen hankintaan. Mikäli resursseja joudutaan

varaamaan normaalin toiminnan ulkopuolelta, pitää vähintään kiinteät kulut arvioida kuluihin. Tällaisia kuluja ovat Leen ym. (2006) mukaan palkkakulut sivukuluineen, laitteisto sekä ohjelmisto, jotka on tarvittu projektin läpiviemiseen. Arvioinnissa on kuitenkin huomioitava erinäisiä muuttuvia kustannuksiakin.

Suurempi huomio voidaankin kiinnittää datan laatuun ja oikeellisuuteen. Datan laatuongelmiin on Leen ym. (2006) mukaan kymmenen syytä:

1) Useita datan lähteitä, joista tulee eri arvot samalle informaatiolle.
2) Subjektiivinen arviointi datan luomisessa, josta muodostuu ennakkoasenteellista informaatiota.
3) Rajoitettu tietokonekapasiteetti voi rajoittaa pääsyä asiaankuuluvaan informaatioon.
4) Tietosuojan ja saavutettavuuden ristiriidat.
5) Koodatun datan ristiriitaisuuksien aiheuttamat ongelmat ymmärrettävyydelle.
6) Ei-numeerisen datan indeksoinnin hankaluus voi estää asiaankuuluvan informaation löytymisen.
7) Datan suuri määrä voi aiheuttaa ongelmia asiaankuuluvan informaation löytymiselle asiaankuuluvassa ajassa.
8) Tiedon tallentamisessa käytettävät säännöstöt voivat rajoittaa tai jopa estää tarvittavan tiedon tallentamisen. Joissain tapauksissa voi olla houkutus tallentaa "väärää" tietoa joihinkin kenttiin kuin pysäyttää tiedon tallentaminen.
9) Tallennetun datan käyttötarpeiden muuttuminen, esimerkiksi organisaatiomuutoksissa ja lakien muutoksissa.
10) Sovellusten puutteelliset integrointimekanismit voivat aiheuttaa asiaankuulumattomia päätelmiä, sääntöjä, arvoja, jne. Alkuperäinen datan tarkoitus voi kadota. (Lee ym. 2006)

Motro ja Rakov (1998) ovat artikkelissaan maininneet datan laadulle kriteereiksi eheyden, eli virheettömyyden, täsmällisyyden, tarkkuuden ja oikeellisuuden sekä koko datan saatavuuden (Motro ym. 1998). Datan saatavuudelle on olennaista, että sekä yksittäiset tietueet, että tietueen sisällä kaikki kentät ovat saatavilla sen mukaisesti kuin operatiivinen tarve sekä mm. raportointi niitä vaativat. Bibliografisen datan osalta datan siirrossa on huomioitava lisäksi bibliografiset marc21-formaatin (Machine readable cataloging) mukaiset tiedot. Marc21-formaatissa on määritetty aineiston kuvaamiseen tarvittavat

tietokentät. Esimerkiksi teoksen nimeke, tekijä, valmistumisvuosin sekä teokseen liittyvät asiasanat. Tietokannan taulussa bibliografinen marc21-tietue on tallennettu yhteen binäärimuotoiseen kenttään. Tuplatietueiden muodostumiseen olisi myös kiinnitettävä huomiota. Tuplatietueita voi bibliografisen datan osalta muodostua sekä sisällön kuvailua sisältäviin bibliografisiin tietueisiin että teoksen sijaintitietoja sisältäviin tietueisiin. Marc21-formaattia käsitellään tarkemmin luvussa 2.2.

Laadukkaan datan ja tiedon voidaan nähdä olevan merkittävä tuotantotekijä. Datan laatuongelmat näkyvät järjestelmän käytettävyyden heikkenemisenä ja kulujen kasvamisena (Lee ym. 2006). Redman (1998) on artikkelissaan luetellut huonolaatuisen datan vaikutuksia. Tyypillisiä vaikutuksia ovat:

- alentunut asiakastyytyväisyys,
- lisääntyneet kustannukset,
- alentunut työntekijöiden tyytyväisyys,
- dataan hyödyntämiseen perustuvan päätöksenteon ongelmallisuus,
- datan hyödyntämisen ongelmat tietovarastoissa ja tietokannan uudelleen suunnittelussa sekä
- strategioiden asettamisen ja jalkauttamisen ongelmat. (Redman, 1988)

Bibliografisten tuplatietueiden haittavaikutuksia on lukuisia. Tietokannan koko saattaa kasvaa tarpeettomasti. Pienissä tietokannoissa vaikutus tallennustilan osalta on melko vähäinen. Kuitenkin, usean tietokannan palvelimella Oracle -tietokannan tablespace-määritystä voidaan kuitenkin joutua tarpeettomasti kasvattamaan. Tuplatietueet vaikuttavat myös aineistohakujen toimivuuteen. Useampi samansisältöinen teos sekoittaa hakutuloksia, ja mm. aineistopyynnön kiinnittäminen teokseen hankaloituu ja aineiston kierrolle (tuottavuus) aiheutuu ongelmia. Samoin osakohteita sisältävien tietueiden näkyvyys asiakasliittymässä sekä henkilökunnan näkymässä voi mennä sekaisin. Lisäksi myös tietueiden määrään perustuvien yhteisten palveluiden rahoitusosuuksien laskennan kautta voi organisaatiolle kertyä tarpeettomia kuluja. Kaikilta osin tuplien muodostumista bibliografisessa datassa ei kuitenkaan voi välttää. Koska bibliografisen tietueen yksiselitteinen yksilöivä tunniste on teoksen ISBN-numero, sen tiedon sisältävä marc21-kentän pitäisi kuitenkin olla vastaava molemmissa yhdistettävissä tietueissa. Järjestelmässä

kyseinen kenttä voi kuitenkin sisältää useampia samanarvoisia tunnisteita. Kentässä voi olla sekä nidotusta teoksesta, sidotusta teoksesta tai pdf- muotoisesta (portable document format) teoksesta kertova tunniste. Lisäksi järjestelmään on mahdollista luoda sääntöjä, joiden perusteella tietueet jätetään yhdistämättä. Tällainen tieto voi sisältää mm. bibliografisen tietueen eroavaisuuksien tarkistuksen, jonka rajan ylittyessä tietuetta ei tulkita samaksi yhteneväisestä tunnisteesta huolimatta. Muodostuvien tuplatietueiden tarkistukseen luodaan sql-kyselyihin perustuva tarkistusprosessi, jossa tuplatietueet etsitään ja yhdistellään erillisellä suoritteella.

Kirjastojärjestelmän tietokantaa yhdisteltäessä myös asiakastiedot sekä niihin liittyvät lainatiedot pitää siirtää. Koska asiakkaat on tarkoitus siirtää aktiivisiksi asiakkaiksi heti yhdistämisestä alkaen, siirrettäviin tietoihin on kiinnitettävä myös lisähuomioita. Asiakastietojen osalta on olennaista kerätä lähdetietokannan asiakasrekisteristä kaikki olennainen data, jolla täytetään kohdejärjestelmän käyttösääntöjen mukaiset vaatimukset ja joita voidaan rajapinnan kautta järjestelmään siirtää. Kahden aktiivisen kirjastotietokannan asiakasrekisteriä yhdistettäessä siirron voi tehdä kahdessa osassa. Ensin siirretään sellaiset asiakkaat, joiden havaitaan olevan ennestään sekä kohdejärjestelmässä että lähdejärjestelmässä. Näiden tietojen siirrossa voidaan huomioida mm. osoitepäivitysten oikeellisuus sen mukaisesti, kummassa on ajantasaisempaa dataa esimerkiksi automaattisten päivitysprosessien johdosta. Sellaiset asiakkaat, joiden tiedot ovat vain lähdejärjestelmässä, voidaan siirtää kohdejärjestelmään uusien asiakkaiden prosessia hyödyntäen.

2.2 Käytettäviä tekniikoita

2.2.1 Visual Studio

Microsoft Visual Studio on useamman ohjelmointikielen sisältävä Microsoftin sovelluskehitysympäristö. Visual Studio -työkalua voidaan käyttää niin verkkosovellusten, mobiilisovellusten kuin perinteisempien Windows-sovellusten luomiseen. Randolphin, Gardnerin ja Andersonin (2010) mukaan Microsoft on työskennellyt kauan

sovelluskehittimien parissa ja yhtiön ensimmäinen tuotekin oli BASIC (Beginner's all-purpose symbolic instruction code). Nykyisissä sovelluskehittimien versioissa kehittäjillä on mahdollisuus tehdä ohjelmia omien vahvuuksien mukaisilla ohjelmointikielillä ja haluamilleen alustoille. Vaihtoehtoisia ohjelmointikieliä ovat esimerkiksi Visual Basic ja C++. Tässä työssä on kehitetty Visual Basicilla sovellus kirjoittamaan nidetietoja tietokantaan hyödyntäen Voyager-kirjastojärjestelmän Batchcat-rajapintaa. Evjenin, Hollisin, Sheldonin ja Sharkeyn (2008) mukaan ensimmäinen Visual Basicin versio on julkaistu vuonna 1991. Vuonna 1995 julkaistun Visual Basic 4.0 -version myötä tuote alkoi muuttua proseduuripohjaisesta ohjelmointikielestä olio-ohjelmoinnin suuntaan. Visual Studiossa on mahdollista luoda käyttöliittymälomake valmiista visuaalisista olioista. Lisäksi olioille kuuluville attribuuteille on mahdollista asettaa arvoja, joilla määritellään olion ulkoasua tai toiminnallisuutta. Tämä toiminnallisuus antaa mahdollisuuden niin kaupallisille toimijoille kuin omaan tarpeeseen sovelluksia tekeville ohjelmoijille tuottaa erilaajuisia sovelluksia nopeasti vastaamaan käyttötarpeita. Evjen ym. (2008) mukaan on suuri etu, jos pystytään tuottamaan sovelluksia kirjoittamalla vähemmän varsinaista ohjelmakoodia.

Visual Studio hyödyntää .NET Framework -komponenttikirjastoa. Krantin, Sandhun ja Jeet Chakrottin (2002) mukaan Microsoftin .NET lähestymistavan tarkoituksena on taata kehittäjille ketteryyttä tuottaa skaalautuvia sovelluksia. He jakavat .NET Frameworkin kolmeen tasoon: esitystapa-, toimintalogiikka-, sekä tiedon saanti ja tietokantataso. Esitystapatasolla on yleensä asiakasohjelmisto ja tämä taso on luonnollisesti ylin taso joka käyttää alemman tason palveluita hyväkseen. Toimintalogiikkataso mahdollistaa mm. COM+ (Component Object Model) palvelujen hyödyntämisen, hakemistopalvelut sekä tietoturvaratkaisut. Hierarkian alimmalla tasolla on tiedon saanti ja tietokantataso. (Kranti ym. 2002)

2.2.2 VBA- ohjelmointikieli

VBA (Visual basic for applications) on Microsoftin työkaluohjelmissa käytettävä sovelluskehityskieli. VBA muistuttaa Visual Basicin kielioppia. VBA:lla voidaan

automatisoida toimintoja tai luoda itsenäisiä toimintoja, kuten tietojen kirjoitus siirto- tai seurantatiedostoon. VBA voi hyödyntää käyttöjärjestelmän API-rajapintoja (application programming interface) ja DLL-sovelluslaajennuksia (dynamic link libraries). Vinen (2005) teoksessa mainitaan, että VBA-sovellus ei kuitenkaan yleensä pysty käynnistymään ilman isäntäohjelmaa. Samoin vaikka VBA kuuluu kaikkiin yleisimpiin Microsoft Officen työkaluohjelmiin, ne hyödyntävät jokaisen isäntäohjelman omia olioluokkia. Siten VBA for Access ymmärtää vain Microsoft Accessin olioita, vaikka Visual Basiciin perustuva ohjelmakoodi onkin ymmärrettävissä peruskielirakenteen puolesta kaikissa ympäristöissä.

Tässä työssä VBA-sovelluksia on käytetty datan kirjoittamiseen ulkoiseen siirtotiedostoon. Näissä toiminnoissa on hyödynnetty Voyager-kirjastojärjestelmän rajapintojen ymmärtämiä tiedostoformaatteja. Voyager-kirjastojärjestelmän siirtotiedot noudattavat yleensä ennalta määritettyä määrämittaista tiedostomuotoa, jossa tietueen kenttien pituudet on määritelty tarkasti. Esimerkiksi asiakastiedon SIF -tiedosto (standard interface file) noudattaa tarkoin määritettyä formaattia, jossa on määritetty mitkä kentät ovat siirrossa pakollisia ja monennestako merkistä mikäkin kenttä alkaa tietueessa. Tiedonsiirron formaattia kuvataan tarkemmin luvussa 3.

2.2.3 SQL-kyselykieli

SQL-kyselykieli (standard query language) on relaatiotietokantaan tehtävien kyselyiden tekemiseen kehitetty standardoitu kyselykieli. Sen avulla voidaan lähes mihin tahansa relaatiotietokantaan tehdä tiedonhakuja, lisäyksiä sekä päivityksiä. SQL on ANSI (American national standards institute) standardisoitu. Kuitenkin SQL-kyselykielestä on olemassa myös sovelluskohtaisia versioita. Mm. Oraclen ja Microsoft Access -tietokannan kyselykielet poikkeavat jonkin verran toisistaan ja tämä on otettava huomioon tehtäessä ristiin kyselyitä kyseisten tietokantojen osalta. Muun muassa Microsoft Accessin LEFT- ja RIGHT-funktiot on Oraclessa toteutettavissa INSTR- ja SUBSTR-funktioiden avulla. Toinen esimerkki on Microsoft Accessin TOP-funktio, joka Oraclessa on toteutettu ROWNUM-funktiolla. ODBC-rajapinta (open database connectivity) huolehtii tietotyyppien oikeellisuudesta tiettyyn rajaan saakka. Kuitenkin on pidettävä huolta, että

numeerisen ja alfanumeerisen kentän muuntaminen saman laatuiseksi on suoritettava tarvittaessa funktioilla, jotta kyselyn palauttamat vertailutulokset olisivat luotettavat.

2.2.4 Batchcat -luettelointirajapinta

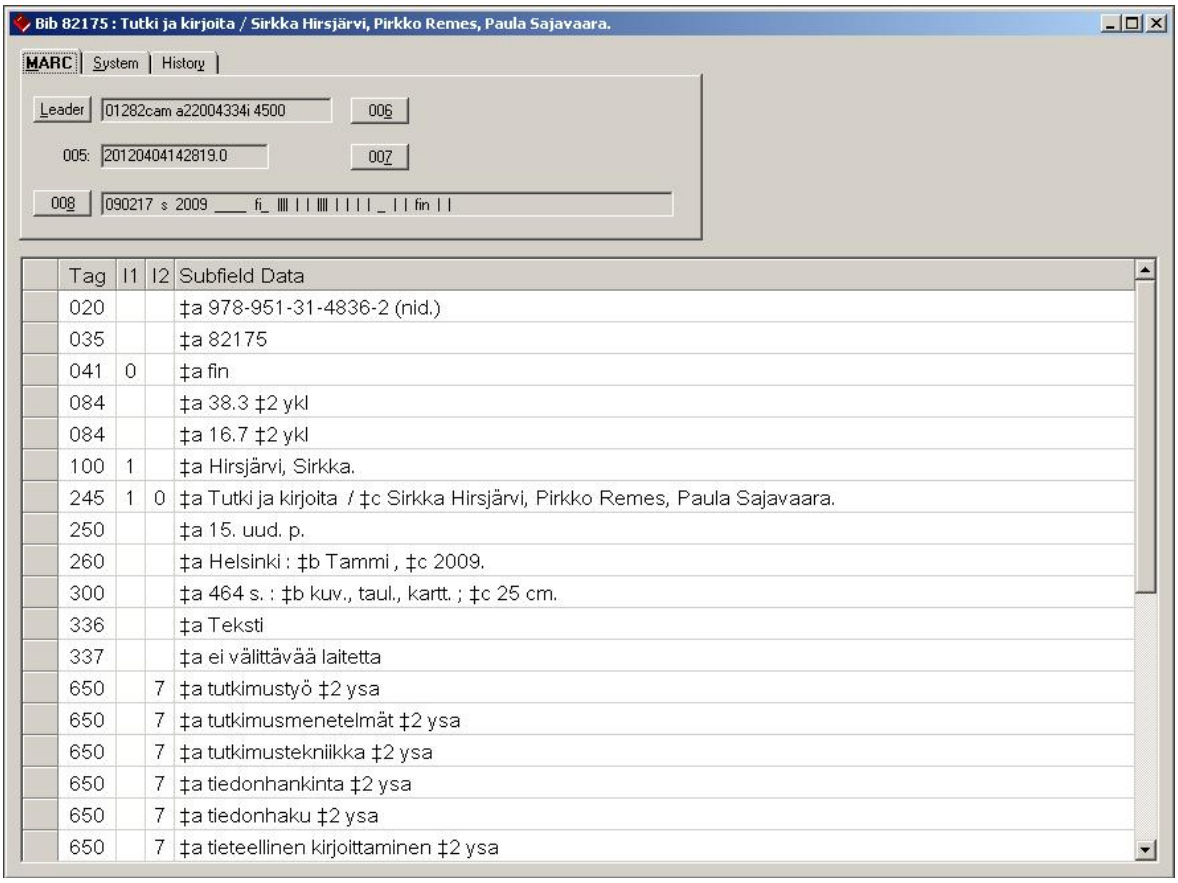
Batchcat-luettelointirajapinta on Ex Libriksen kehittämä sovelluslaajennus Voyager-kirjastojärjestelmään. Se tarjoaa mahdollisuuden lisätä, muokata ja poistaa bibliografisia tietueita ja nidetietoja Voyager-kirjastojärjestelmässä. Paras hyöty luettelointirajapinnan hyödyntämisessä saavutetaan, kun samanaikaisesti käsiteltävien tietueiden lukumäärä on suuri. Käytettäessä Batchcat-rajapintaa, tietueet voidaan ottaa käsiteltäväksi Voyager-kirjastojärjestelmän ulkopuolelle ja muokata jokainen tietue joko yksilöllisesti tai yhtenäisen säännöstön mukaisesti. Lopuksi voidaan kirjoittaa tietueet takaisin järjestelmään oman sisäisen tunnisteen perusteella. Tunnisteen avulla tunnistetaan, päivitetäänkö olemassa oleva tietue vai kirjoitetaanko uusi tietue.

Voyager 7.2. Batchcat.dll Technical User's Guide (2009) -käyttöohjeessa kerrotaan, että sovelluslaajennuksen saa Voyagerin ulkopuoliseen käyttöön lataamalla se esimerkiksi Visual Studion viitetiedostolistaan ja esittelemällä se Visual Basic -koodin alussa. Tämän jälkeen sovelluslaajennuksen funktiot ovat käytettävissä omassa sovelluksessa.

2.2.5 Marc21 -formaatti

Marc21-formaatti Library of Congressin ylläpitämä yhtenäisformaatti bibliografisen aineiston kuvailuun. Yhtenäisformaatin mukaiset tietueet ovat siirrettävissä helposti järjestelmästä toiseen. Marc21-tietueet koostuvat nimiöstä, hakemistosta ja kentistä ja ne ovat binäärimuotoisia tiedostoja. Marc21-formaatin kentät voivat olla joko kiinteitä kenttiä tai vaihtuvamittaisia kenttiä. Bibliografisen kuvailutiedon formaatti on tarkoitettu muun muassa painettujen tekstiaineistojen, e-aineistojen, nuottien ja äänitallenteiden kuvailuun. Kuvassa 1 on esitetty esimerkkitietue marc21-formaatissa. Yleensä kuvailutiedoissa on kerrottu omissa kentissään esimerkiksi nimeketiedot, tekijän tai tekijöiden tiedot ja

ulkoasutietoja. Lisäksi kuvailutiedoissa on yleensä myös sisällönkuvailutietoja, esimerkiksi teoksen sisältöön liittyviä asiasanoja. Näitä tietoja voidaan hyödyntää esimerkiksi järjestelmän asiakasliittymiä ja -indeksejä määriteltäessä, jolloin voidaan saada tarkempia hakutuloksia asiakkaan haluamasta aihealueesta. (Marc21-formaatti: Bibliografiset tiedot, 2013)



Tag	I1	I2	Subfield Data
020			‡a 978-951-31-4836-2 (nid.)
035			‡a 82175
041	0		‡a fin
084			‡a 38.3 ‡2 ykl
084			‡a 16.7 ‡2 ykl
100	1		‡a Hirsjärvi, Sirkka.
245	1	0	‡a Tutki ja kirjoita / ‡c Sirkka Hirsjärvi, Pirkko Remes, Paula Sajavaara.
250			‡a 15. uud. p.
260			‡a Helsinki : ‡b Tammi , ‡c 2009.
300			‡a 464 s. : ‡b kuv., taul., kartt. ; ‡c 25 cm.
336			‡a Teksti
337			‡a ei välittävää laitetta
650		7	‡a tutkimustyö ‡2 ysa
650		7	‡a tutkimusmenetelmät ‡2 ysa
650		7	‡a tutkimustekniikka ‡2 ysa
650		7	‡a tiedonhankinta ‡2 ysa
650		7	‡a tiedonhaku ‡2 ysa
650		7	‡a tieteellinen kirjoittaminen ‡2 ysa

Kuva 1: Marc21-formaatissa oleva esimerkkitietue.

Järjestelmästä ulos luettuja marc21-tietueita voidaan muokata kolmannen osapuolen ohjelmilla. Esimerkiksi MarcEdit-ohjelmalla marc21-formaatin mukaisia tietueita voidaan muokata normaalin tekstinkäsittelyn kaltaisesti. Tietueita voidaan muokata tekstieditorissa yksittäin tai kohdistaen muutokset tiettyyn marc -kenttään kaikkiin tietueisiin. (Reese, 2013)

2.2.6 Microsoft Access

Microsoft Access on Microsoft Office -työkaluohjelmiston mukana asennettava tietokanta-ohjelma. Access on tarkoitettu yleensä pienten tai keskisuurten tietokantojen hallintaan. Sen käyttöliittymä noudattelee Microsoft Office -työkaluohjelmiston käyttöliittymää, ja on siten helposti myös satunnaisemman käyttäjän omaksuttavissa.

Käyttökelpoiseksi usean tietolähteen käsittelyyn Microsoft Accessin tekee mahdollisuus tuoda tai linkittää tietoja ja tauluja ulkoisista lähteistä. Parhaimmillaan Microsoft Access tietokantaan voidaan tuoda ja linkittää eri tietokantojen tauluja, kuten esimerkiksi Oraclen tai Microsoft SQL Server -tietokannan tauluja. Tämä vaatii ODBC (open database connectivity) ajureiden asennuksen ja määrittelyn. Tuotuja ja linkitettyjä tauluja voidaan hyödyntää tietokantakyselyissä kuten Microsoft Accessin tietokantaankin luotuja tauluja. Tämä mahdollistaa helpon tietojen vertailun eri lähteiden välillä. Tuotujen taulujen avulla on mahdollista nopeuttaa kyselyiden suoritusaikoja verkon yli tapahtuvien kyselyiden jäädessä pois.

Tässä työssä on pyritty hyödyntämään Microsoft Accessin ulkoisten taulujen linkitys-ominaisuutta Oraclen tietokantoihin. Työssä on ollut tarpeellista vertailla eri tietokannoissa olevan datan ominaisuuksia. Tätä projektia varten luotiin oma Microsoft Access tietokanta, johon linkitettiin tauluja sekä lähdetietokannasta että kohdetietokannasta. Näin molempien tietokantojen tietoja on voitu vertailla luomalla SQL-kysely, jossa on käytetty tauluja molemmista linkitetyistä tietokannosta. Lisäksi projektissa on hyödynnetty aputaulujen luontimahdollisuutta Microsoft Access -ympäristössä. Aputauluja voidaan muodostaa muun muassa pilkkomaan pitkiä ja raskaita SQL-kyselyitä osiin. Lisäksi suorituskyvyssä on suuri ero, kun kysely kohdistetaan paikallisessa tietokannassa olevaan tauluun.

2.2.7 Oracle

Oracle tietokannan hallintajärjestelmän kehityksen voidaan Brylan (2004) mukaan katsoa alkaneeksi vuonna 1979, kun Oracle julkaisi ensimmäisen kelvollisen relaatiotietokannan

hallintajärjestelmän. Oraclen tietokannan hallintajärjestelmä pystyy hallitsemaan todella suuria tietokantoja sekä useita eri instansseja. Lisäksi Oracle on kehittänyt omat versiot eri käyttöjärjestelmille. Tässä työssä on osa tietokantakyselyistä toteutettu SQL*Plussalla Oraclen tietokantaan. Vaikka kaikki kyselyt olisi toteutettavissa linkitettyjen taulujen avulla Microsoft Accessissa, Oracleen tehtyjen kyselyiden voi käytännössä nähdä suurilla massoilla valmistuvan huomattavasti nopeammin. Binääristen bibliografisten tietueiden purkamiseen Voyager järjestelmässä toteutettujen funktioiden suoritusaikojen ero on laskettavissa jopa tunneissa Oraclessa tehtyjen kyselyiden ja Microsoft Accessissa tehtyjen kyselyiden välillä.

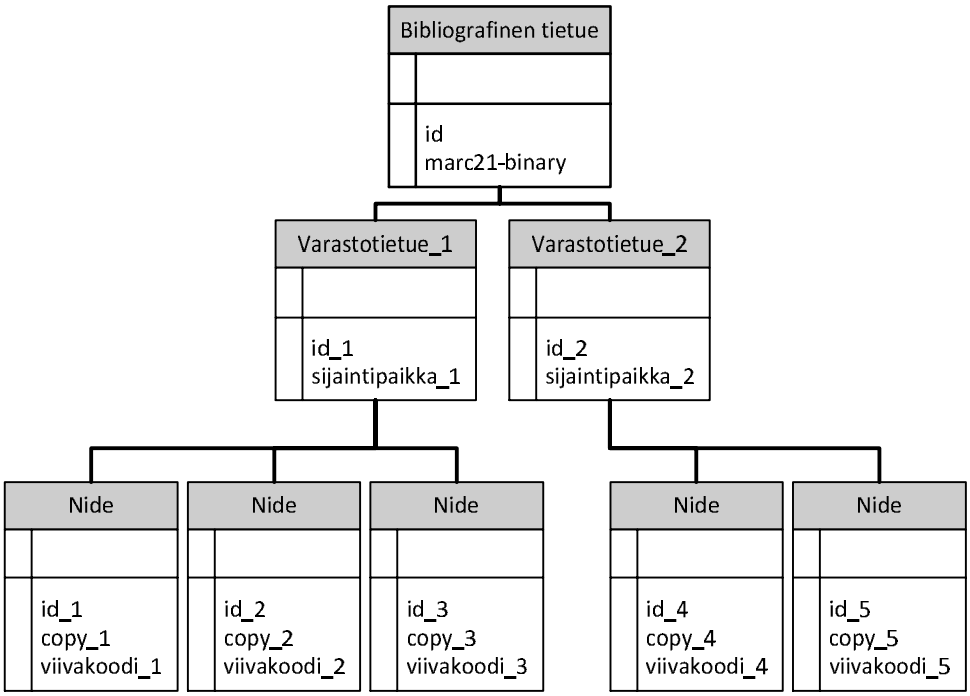
SQL*Plus on ollut Brylan (2004) mukaan toteutettuna Oracleen yhtä kauan kuin itse relaatiotietokannan hallintajärjestelmäkin on ollut olemassa. Sillä voidaan kytkeytyä tietokantaan ja suorittaa SQL-kyselyitä tauluihin tietokannassa. SQL*Plussan eduksi voidaan Brylan (2004) mukaan laskea sen toiminnallisuus myös silloin, kun tietokanta on jostain syystä alhaalla ja käytettävissä on vain merkkipohjainen istunto palvelimeen. Lisäksi sen eduksi voidaan laskea myös se, että sen toiminta on samanlainen riippumatta Oraclen taustalla toimivasta käyttöjärjestelmästä.

2.2.8 Voyager

Voyager -kirjastojärjestelmä on Ex Libris -yhtiön tuottama integroitu kirjastojärjestelmä (ILS, integrated library system). Ex Libris on johtava järjestelmätoimittaja akateemisille kirjastoille (Voyager Prochure, 2013). Suomessa Voyager -kirjastojärjestelmää käyttävät yliopistokirjastot ja korkeakoulukirjastot. Voyager on client/server -arkkitehtuuria hyödyntävä kirjastojärjestelmä. Järjestelmän verkkokirjasto (OPAC Online Public Access Catalog) on korvautumassa uudella asiakasliittymällä ja sen vaihto toteutetaan Päijät-Hämeen koulutuskonsernissa omana projektinaan. Voyager-kirjastojärjestelmässä on omina asiakasohjelminaan toteutettu aineiston luettelointi, hankinta ja kausijulkaisujen saapumisvalvonta, lainaus, raportointi sekä järjestelmän hallintaliittymä. Tässä diplomityössä käytetään lähinnä luettelointiin ja lainaukseen liittyviä rajapintoja.

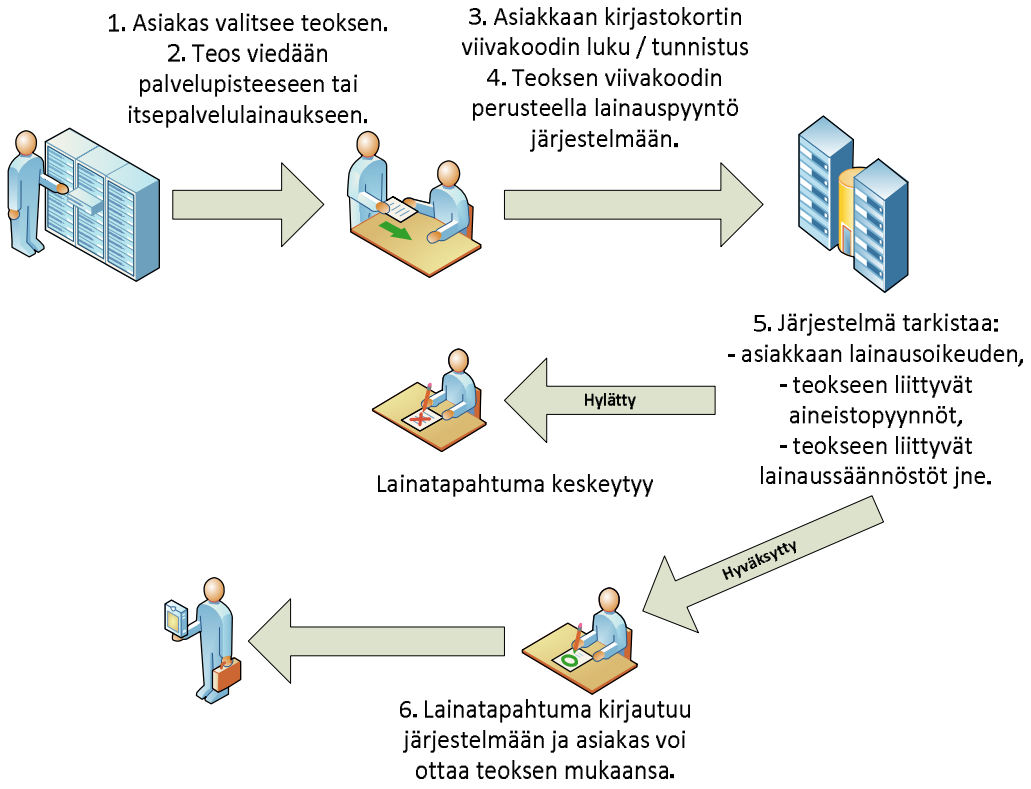
Kuvassa 2 on esitelty Voyager-kirjastojärjestelmässä luetteloitujen teoksiin liittyvien

tietueiden kaavio. Teoksen kuvaustiedot, kuten teoksen nimi, tekijä, ulkoasutiedot ja niin edelleen, on lueteltu bibliografisessa tietueessa. Varasto- tai sijaintitiedot on lueteltu varastotietueessa. Nämä tietueet on tallennettu binäärimuotoiseen kenttään omissa tauluissaan marc21-formaatissa. Hierarkian alimmaisella tasolla on nidetietueet. Ne sisältävät yksittäisen niteen tietoja, kuten viivakoodin tai RFID-tunnisteen (radio frequency identification), niteen järjestysnumeron, lehden numerotiedot ja niin edelleen. Näennäisestä yksinkertaisuudestaan huolimatta tietueisiin liittyvä tietokantarakenne on melko monimutkainen. Eri tasot linkittyvät toisiinsa erillisten taulujen kautta ja kaikkea tietueiden näkyvyyteen liittyviä ER-diagrammeja (entity relationship) ei ole julkistettu asiakasorganisaatioille. Siksi bibliografisen datan ja nidetietojen suoraan kantaan kirjoittaminen SQL-lauseilla olisi riskialtis ja jopa mahdotonkin tehtävä. Lähtökohtaisesti kaiken omatoimisen datan kirjoittaminen tietokantaan tulisi tapahtua rajapintojen tai asiakasohjelman kautta. Tällöin vastuukysymyksissäkään ei tule ristiriitoja. Järjestelmän toimittaja vastaa järjestelmän ja siihen liittyvien rajapintojen teknisestä toimivuudesta ja asiakkaalle jää vastuu datan oikeellisuudesta ja eheydestä. Toinen painava syy käyttää rajapintoja on se, että järjestelmän asiakasorganisaatioille on jaettu vain Oraclen read only -tasoiset tunnukset. Näitä tunnuksia tarvitaan järjestelmään liittyvän raportoinnin tarpeisiin ja tietojen lukemiseen tietokannasta.



Kuva 2: Voyager tietueiden kaavio.

Lainaus-asiakasohjelmassa käsitellään asiakastietoja, lainatietoja, asiakkaan maksuja, aineistopyyntöjä jne. Kuvassa 3 on kuvattu lainaustapahtuman prosessikaavio. Fyysisen teoksen lainaustapahtuma alkaa siitä, kun asiakas noutaa teoksen varastointipaikasta ja vie sen asiakaspalveluun tai itsepalveluun. Seuraavaksi asiakas tunnistetaan ja suoritetaan varsinainen lainaustapahtuma. Järjestelmä tarkistaa lainaukseen liittyvät mahdolliset esteet (esimerkiksi aineistopyyntö samaan teokseen tai asiakkaan lainakielto) sekä asiakkaan asiakasryhmään liittyvän järjestelmään määritellyn säännöstön. Mikäli säännöstö sallii aineistotyypin lainauksen eikä mitään ennalta määritellyn säännöstön mukaista estettä havaita, teos siirtyy asiakkaan lainatietoihin. Ennen tätä tapahtumaa asiakastiedot on täytynyt syöttää järjestelmän tietoihin sekä lainaukseen liittyvä säännöstö on täytynyt määrittää. Luonnollisesti varastosijainti sekä teokseen liittyvät tunnisteet on oltava järjestelmässä. Kun kahden kirjaston tietokantoja yhdistetään, täytyy lainaukseen liittyvät säännöstöt yhtenäistää ja määrittää myös uusien toimipisteiden osalta kohdejärjestelmään. Näitä määrityksiä tarvitaan yleensä myös rajapintojen kautta tapahtuvien tiedonsiirtojen mahdollistamiseksi. Jotta teoksen varastotietue voidaan ajaa Batchcat-rajapinnan kautta järjestelmään, uuden tietueen sisältämä sijaintitieto on löydyttävä myös järjestelmän systeemimäärityksistä.



Kuva 3: Lainaustapahtuman prosessikaavio.

3 TIETOKANTAOPERAATIOT

3.1 Bibliografinen data

Tässä luvussa esitellään Lahden korkeakoulukirjastojen tietokantaoperaation toiminnallisuus tietoteknisestä näkökulmasta. Koska tietokantakyselyt ja tietojen kirjoittaminen kohteena olevaan tietokantaan ovat olennainen osa tätä diplomityötä, tämä luku sisältää runsaasti lähdekoodia ja SQL-kyselyitä. Jo ennen kuin varsinainen tietokantaprojekti on voinut alkaa, tietojen siirron kohteena olevaan kirjastojärjestelmään on luotu uudet toimipistetiedot. Tässä projektissa aineisto siirtyi ensivaiheessa järjestelmässä tietokannasta toiseen, varsinainen tiedekirjaston fyysinen muutto uuteen oppimiskeskukseen tapahtui myöhemmässä vaiheessa. Uusina toimipisteinä luotiin Tiedekirjasto ja Ympäristökirjasto. Näille luotiin sijaintipaikat sekä lainaussäännöstöt vastaamaan Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen aikaisemmin määriteltyjä käytänteitä.

3.1.1 Varmistustoimenpiteet

Ennen mitään toimenpiteitä kohteena oleva tietokanta on hyvä varmistaa. Ammattikorkeakoulujen kirjastojärjestelmän palvelimella (armas.amkit.fi) on yhdeksäntoista kirjaston aineistotietokannat ja asiakasrekisterit (Amkit-konsortio, 2013). Vaikka tietosuojamääritysten myötä on käytännössä mahdotonta käsitellä muiden kuin oman organisaation tietokantaa, on kuitenkin syytä varautua myös mahdottomiin uhkiin etukäteen. Armas-palvelimella ajetaan koko palvelimen varmistukset joka ilta. Oracle tietokanta on varmistettavissa Export/Import -toiminnolla erilliseen tekstitiedostoon. Oraclen tietokanta on myös mahdollista varmistaa niin, että kantaan pääsy estetään varmistuksen ajaksi, jolloin voidaan varmistaa kaikki data, seurantatiedostot sekä kontrollitiedostot. Palautus saattaa olla hankalammin toteutettavissa. Koska kaikkia palvelimella toimivia tietokantoja ei voi sulkea tietokantojen yhdistämisen ajaksi, tietokannoissa tapahtuu muutoksia koko ajan. Käytännössä koko Oraclen tietokantaa on

mahdotonta palauttaa päivän tai useamman päivän päästä, koska tänä aikana tehdyt muutokset menetetään. Mikäli omaan tietokantaan joudutaan jotain varmistettuja tietoja palauttamaan, se tapahtuisi käytännössä vertaamalla muutoksia varmistetun ja muutoksia sisältävän taulun välillä ja palauttamalla tietoja manuaalisesti tietokantaan. Varsinaisten palvelimen ja Oraclen tietokannan varmistusten lisäksi on syytä ottaa ennen muutosten ajoa talteen yksittäisiä tietoja omasta kannasta. Esimerkiksi Voyagerin eräajona suoritettava marcexport-eräajo kirjoittaa bibliografiset tietueet sekä varastointitietueet tekstitiedostoon. Nidetiedot kannattaa ottaa talteen esimerkiksi Microsoft Accessilla tehdyllä SQL-kyselyllä. Asiakastiedot on myös mahdollista ottaa talteen Patron Extract -eräajolla. Tämä ohjelma kirjoittaa Voyagerin Patron SIF -muotoisen tekstitiedoston ja se on palautettavissa suoraan takaisin tietokantaan tarvittaessa.

3.1.2 Bibliografisen datan ulosluku

Varsinaisia varmistustoimenpiteitä täydentämään kannattaa jokaisen tietokantaan tehtävän suuren tietojoukon kirjoituksen aluksi varmistaa muokkaamisen kohteena olevat tiedot. Tietokannassa ennestään olevat bibliografiset tietueet ja varastotietueet on syytä ottaa talteen tekstitiedostoon. Näin menetellen voidaan palata aina työvaihetta edeltävään tilanteeseen näiden tietueiden osalta. Bibliografiset tietueet ja varastotietueet voidaan talteen Voyagerin marcexport-eräajolla (Voyager 7.0 Technical User’s Guide, 2008).
Samalla marcexport-eräajolla voidaan siirrettävät tietueet lukea ulos lähteenä olevasta tietokannasta. Voyager 7.0 Technical User’s Guide (2008) -käyttöohjeen mukaan ulosluku voidaan suorittaa Kuvassa 4 esitetyllä komennolla ja parametreilla. Esimerkin parametreilla siirrettäväksi otetaan kaikki bibliografiset tietueet sekä varastotietueet -r -parametrilla. ”G” arvo tarkoittaa molempien tietuetyyppien ryhmää. Bibliografisten tietueiden yksilöivän tunnisteen mukainen rajaus valitaan -m -parametrilla ja -t -parametrilla yksilöivän tunnisteen rajaukseksi asetetaan kaikki tietueet.

```
./Pmarcexport -rG -mR -tAll
```

Kuva 4: Siirrettävien bibliografisten tietueiden sekä varastotietueiden ulosluku lähdetietokannasta.

3.1.3 Bibliografisen datan muokkaus

Kirjastojärjestelmän ulkopuolella bibliografista dataa ja sijaintitietoja voi muokata MarcEdit-ohjelmalla. Tässä tietokantojen yhdistämisprojektissa ei nähty tarpeelliseksi muokata tietueita luettelointisääntöjen mukaisen oikeellisuuden tai laadukkuuden näkökulmasta. Tavoitteena tietueiden muokkauksessa oli tunnisteiden luominen marc21-formaatin kenttien joukkoon sekä sijaintipaikasta kertovien koodien vaihto vastaamaan kohdetietokantaan luotuja sijaintipaikkakoodeja. MarcEdit mahdollistaa normaalien tekstinkäsittelystä tuttujen toimintojen hyväksikäyttöä. Esimerkiksi etsi/korvaa, lisää kenttä ja poista kenttä -toimintoja voi hyödyntää. Prosessin aluksi MarcEdit ohjelmaan kuuluvalla MarcBreakerilla puretaan marc21-tietueet muokattavaan tekstimuotoon ja avataan muokattavaksi. Lahden tietokantaprojektissa sekä bibliografisiin tietueisiin että varastotietueisiin, lisättiin tunniste helpottamaan sen löytämistä myöhemmässä vaiheessa. Marc21-formaatin 650-kenttään lisättiin merkintä ”LAKKI”. Tässä on tärkeää huomata, että 650 on asiasanakenttä, jonka sisältö noudattaa asiasanastoja. Omat tunnisteet eivät kuulu asiasanastoihin ja lisäksi varastotietueen formaatti ei tunne 650-kenttää. Projektissa lisätyt omat tunnisteet on siivottu tietueista operaation jälkeen, jotta luettelointidatan laadukkuus on voitu säilyttää. Kun Voyager-kirjastojärjestelmään tuodaan tietueita bulkimport-eräajolla, järjestelmä lisää marc21-tietueeseen kentän 035. Tähän kenttään järjestelmä siirtää sisään tuodussa tiedostossa kentässä 001 olleen bibliografisen tietueen numerotunnisteen. Sisään luettavasta tiedostosta on siksi hyvä poistaa vanha 035-kenttä, jotta uudelleen sisään luettavassa tiedostossa oleva vanha tieto ei sekoittuisi uudelleen luotavaan tietoon. Tätä tietoa voidaan myöhemmin käyttää varastotietueita käsiteltäessä. Siirrettävistä varastotietueista muutetaan kentässä 852 oleva tietokeskuksen sijaintipaikkaa identifioiva koodi vastaamaan uuden kohdesijainnin koodistoa. Sijainnit ja niiden koodit on oltava luotuna järjestelmässä ennen tiedostojen latausta. Tässä yhteydessä voi myös tarkistaa muut mahdolliset muutokset tai korjaukset joita sisään luettaviin tietueisiin on tarkoitus tehdä. Muokkauksen lopuksi MarcEditissä tallennetaan tekstitiedosto ja viedään tiedot järjestelmän sisään luvun ymmärtämään muotoon.

3.1.4 Bibliografisen datan sisäänluku

Muokattujen tietueiden sisäänluku kohdejärjestelmän tietokantaan alkaa tietueiden käsittelyyn liittyvän säännöstön luomisella systeemiin. Järjestelmään on luotava tuplatietueiden muodostumisen estävä säännöstö. Jos käytetään yhden tietokannan tietueita järjestelmän ulkopuolella muokattavana, tietueiden tunnistus tapahtuu marc21-formaatin 001-kentän perusteella (tietueen yksilöivä tunniste järjestelmässä). Kun tietueita tuodaan toisesta järjestelmästä tai saman järjestelmän toisesta tietokannasta, tietueen yksilöintiä ei voi hyödyntää. Marc21-formaatissa kenttään 020 kirjoitetaan teoksen ISBN-tunniste (international standard book number). Tätä tunnistetta voi käyttää tunnistamaan saman nimekkeen bibliografinen tietue. Aivan täysin varmasti kyseisellä tunnisteella ei pystytä tunnistamaan bibliografista tietuetta samaa teosta tarkoittavasti, koska 020-kenttä on formaatissa toistettavissa. Tietue voi olla sidottu tai nidottu tai ISBN voi olla myös virheellinen. Lisäksi ISBN-tunniste voi olla joko 10- tai 13-merkkinen. Marc21-formaatin kenttä 022 taas on kausijulkaisuiden vastaava tunniste, ISSN (international standard serial number). Tässä projektissa näiden kenttien katsottiin riittävällä tarkkuudella yksilöivän yhdistettävät bibliografiset tietueet. Niiden tietueiden osalta, joiden katsottiin jäävän yhdistymättä tietokantaan latauksen yhteydessä, luotiin uusi bibliografinen tietue. Tässä menettelyssä muodostuneita tuplatietueita siivottiin ja yhdistettiin latauksen jälkisiivouksessa.

Kuvassa 5 on esitelty Lahden tietokantaprojektissa käytetty tuplatarkistuksen profiili (dublication detection profile). Käytettäessä bi-directional Merge -tuplakäsittelyä, järjestelmä tarkistaa latauksen yhteydessä tietueiden laadun. Mikäli järjestelmässä oleva tietue on luettelointisääntöjen mukaan laadukkaampi tietue, järjestelmä säilyttää sen pohjatietueena. Ladattavasta tietueesta lisätään kentät pohjaksi jäävään tietueeseen. Mikäli ladattava tietue on laadukkaampi, järjestelmässä olevan tietueen kentät siirretään ladattavaan tietueeseen ja se ladataan järjestelmään vanhan tietueen tilalle. Mikäli järjestelmä havaitsee useamman tietueen, johon ladattavan tietueen voisi yhdistää, se jättää tietueen kokonaan käsittelemättä. Tieto hylätystä tietueesta kirjautuu seurantatiedostoon. Lahden projektissa päädyttiin Bi-Directional Merge -tuplakäsittelyn menetelmään, koska nähtiin tarpeelliseksi säilyttää tarpeellisia kuvailutietoja molemmissa tietokannoissa. Vaikka etukäteen oli tiedossa, mitkä kentät haluttiin säilyttää siirrettävistä tietueista,

järjestelmän muut tuplatietueiden käsittelymuotojen ei riittävästi nähty tukevan kenttäkohtaisten tietojen siirtämistä olemassa oleviin tietueisiin.

Nimi: LAKKI
• Dublicate handling: Bi-directional Merge (luo ns. super-bibin)
• Field Definition: 020A, 020B, 020N, 020R, 022A

Kuva 5: Bibliografisen tietueen tuplatarkistuksen profiili.

Voyager-kirjastojärjestelmän Bulk import rule määrittää tietokantaan ladattavien tietueiden osalta, mitä niille tehdään ja miten. Kuvassa 6 on esitelty Lahden tietokantaprojektissa käytetty Bulk Import Rule. Latauksessa käytettävässä säännöstössä kerrotaan, mitä tuplatarkistuksen profiilia käytetään latauksen yhteydessä. Ladattavan tiedoston merkistö kerrotaan profiilissa, tietokantaa tietueet kirjoitetaan Unicode UTF-8 -muotoisena. Latauksen säännöstössä voidaan valita, ladataanko vain bibliografiset tietueet vai ladataanko myös varastotietueet samassa yhteydessä. Tässä yhteydessä on tarkoituksen mukaista ladata molemmat tietueet samassa yhteydessä. Batchcat-rajapinnan kautta olisi mahdollista kuitenkin kirjoittaa kaikki tietuetyypit kohdetietokantaan, mutta se olisi vaatinut lisäohjelmointia ja huomattavasti pidemmän ajan testaukselle.

Nimi: LAKKI
• BibDubProfile: LAKKI
• Merkistö: MARC21 UTF-8
• Ladataan bibliografiset tietueet ja varastotietueet: Bibs, MFHDs
• Leave OPAC suppress unchanged…

Kuva 6: Bulk import rule.

Varsinainen datan sisään luku palvelimella tapahtuu eräajona bulkimportn -komennolla. Ohjelmalle annetaan parametrina -i muokattu marc21-formaatin mukaisia tietueita sisältävän tiedoston nimi sekä sisäänluvussa käytettävä säännöstö. Lisäksi kerrotaan järjestelmälle vielä parametrilla -m, että myös varastotietueen halutaan muodostuvan samalla ajolla. Kuvassa 7 on esitelty Lahden tietokantaprojektissa käytetty sisäänluvussa käytetty komento. Järjestelmä kirjaa tiedostoon raportin latauksen onnistumisesta ja

hylätyistä tietueista. Tiedostossa on kerrottu käsiteltyjen tietueiden lukumäärä. Eritteleystä selviää, montako tietuetta on lisätty, hylätty, korvattu, yhdistetty tai poistettu. Lisäksi virheiden lukumäärä on kyseisessä tiedostossa. Edellä mainitut tietueen lataukseen liittyvät statukset on listattu tietueittain kunkin statuksen omaan seurantatiedostoon. Ennen marc21-tietueiden sisäänlukua tietueiden marc21-formaatin 001-kenttä sisältää lähdetietokannan yksilöivän tunnisteen. Kun sisäänluku on suoritettu, vanha 001-kenttä on siirtynyt 035-kenttään ja kohdetietokannan uusi yksilöivä tunniste on kirjoitettu 001-kenttään.

```
./Pbulkimport -muokatut_marcit.mrc -iLAKKI -m
```

Kuva 7: Bibliografisen datan sisään luku palvelimella

3.1.5 Niteiden latauslistan muodostus

Kun bibliografiset tietueet ja varastotietueet on saatu siirrettyä kohdetietokantaan, voidaan aloittaa nidetietojen siirron valmistelu. Kuvassa 8 on esitelty SQL-kysely, jolla voidaan muodostaa listaus ladatuista ja muokatuista tietueista kohdetietokannassa latauksen jälkeen. SQL-kysely on toteutettu Oraclen ympäristössä. Kyselyssä hyödynnetään Voyager-kirjastojärjestelmään kuuluvia BLOB-funktioita (binary lagre object). Vastaavat funktiot on kehitetty raportointi tarkoitukseen myös Microsoft Access -ympäristöön. Käytettäessä Oracleen toteutettuja funktioita SQL*Plus -liittymässä, vältetään Microsoft Access -ympäristöä vaivaavat suorituskyvyn rajoitteet. Lisäksi vältetään tietoliikenteen vaikutus työaseman ja palvelimen välillä. Kyselyssä haetaan sekä bibliografisesta tietueesta että varastotietueesta marc21-formaatin 035-kentästä lähdetietokannan yksilöivä tunniste. Vastaavat kohdetietokannan yksilöivät tunnisteet haetaan suoraan tietokannan taulusta. Marc21-formaatin 650-kentästä tarkistetaan, että ennen latausta muodostettu väliaikainen tunniste ”LAKKI” löytyy tietueista. Tämän listaus kertoo, mikä lähdetietokannan tietue on siirtynyt tai yhdistynyt kohdetietokannan tietueeksi. Samalla haetaan varastotietueen sijaintikoodi marc21-formaatin 852-kentästä. Tietokantojen yhdistämisprosessissa SQL*Plus -kyselyn tulokset siirrettiin Microsoft Access ympäristöön omaan tauluunsa jatkojalostettavaksi. Listan tietoja hyödynnetään, kun kirjoitetaan nidetietojen siirtolista.

```
/* TUOTANTO uusi_bib, lakki_bib, uusi_mfhd, lakki_mfhd, sijaintitieto holding 852*/
SELECT DISTINCT
lahtidb.bib_data.BIB_ID,
lahtidb.GETBIBTAG(lahtidb.bib_data.bib_id,'035') AS bib_035,
lahtidb.mfhd_data.mfhd_id,
lahtidb.GETMFHDTAG(lahtidb.mfhd_data.mfhd_id, '035') AS mfhd_035,
lahtidb.GETMFHDTAG(lahtidb.mfhd_data.mfhd_id, '852') AS mfhd_852
FROM lahtidb.bib_data, lahtidb.mfhd_data, lahtidb.bib_mfhd
WHERE (lahtidb.bib_data.bib_id = lahtidb.bib_mfhd.bib_id
AND lahtidb.mfhd_data.mfhd_id = lahtidb.bib_mfhd.mfhd_id)
AND (lahtidb.GETBIBTAG(lahtidb.bib_data.bib_id, '650') Like '%LAKKI%')
AND (lahtidb.GETMFHDTAG(lahtidb.mfhd_data.mfhd_id, '650') Like '%LAKKI%')
AND (lahtidb.GETMFHDTAG(lahtidb.mfhd_data.mfhd_id, '035') <> lahtidb.bib_mfhd.bib_id
AND lahtidb.GETBIBTAG(lahtidb.bib_data.bib_id, '035') <> lahtidb.bib_data.bib_id)
ORDER BY lahtidb.bib_data.BIB_ID
```

Kuva 8: SQL-kysely, joka listaa ladatut ja päivitetyt tietueet.

Siirrettävien nidetietojen listauksen muodostava SQL-kysely on esitelty kuvassa 9. Kysely on toteutettu Microsoft Access -tietokannassa. Kyselyssä haetaan lähdetietokannan bibliografisen tietueen sekä varastotietueen yksilöivät tunnisteet. Niteen tiedoista haetaan yksilöivä tietokannan tunniste, niteen viivakoodi sekä viivakoodin status, sijaintitieto sekä niteen tyyppitieto. Viivakoodin statuksen avulla siirrettävät niteet on rajattu vain käytössä oleviin niteisiin eli aktiivisena oleviin viivakoodeihin (status=1). Projektissa pyrittiin rajoittamaan sellaisen aineiston siirtoja, joiden käyttöaste on ollut alhainen. Tällaiset niteet on siivottu Tiedekirjaston varasto-sijaintiin ennen tietokantojen yhdistämistä. Tämänkin kysely tiedot tallentuu Microsoft Accessiin omaan aputauluunsa jatkokyselyitä varten. Tiedekirjaston varasto-sijainnissa olevat niteet jätettiin siirtämättä aktiiviseen tietokantaan.

```
SELECT LAKKIDB_BIB_MFHD.BIB_ID,
LAKKIDB_MFHD_ITEM.MFHD_ID,
LAKKIDB_ITEM.ITEM_ID,
LAKKIDB_ITEM.ITEM_TYPE_ID,
LAKKIDB_ITEM_BARCODE.ITEM_BARCODE,
LAKKIDB_ITEM_BARCODE.BARCODE_STATUS,
LAKKIDB_ITEM.PERM_LOCATION
FROM LAKKIDB_BIB_MFHD INNER JOIN ((LAKKIDB_MFHD_ITEM INNER
JOIN LAKKIDB_ITEM ON LAKKIDB_MFHD_ITEM.ITEM_ID =
LAKKIDB_ITEM.ITEM_ID) INNER JOIN LAKKIDB_ITEM_BARCODE ON
LAKKIDB_ITEM.ITEM_ID = LAKKIDB_ITEM_BARCODE.ITEM_ID) ON
LAKKIDB_BIB_MFHD.MFHD_ID = LAKKIDB_MFHD_ITEM.MFHD_ID
WHERE (((LAKKIDB_ITEM_BARCODE.BARCODE_STATUS)="1")) AND
LAKKIDB_ITEM.PERM_LOCATION not in ("4", "5")
ORDER BY LAKKIDB_BIB_MFHD.BIB_ID, LAKKIDB_MFHD_ITEM.MFHD_ID,
LAKKIDB_ITEM.ITEM_ID;
```

Kuva 9: SQL-kysely, joka listaa lähdetietokannan tietueiden relaatiot.

Edellisten listojen perusteella voidaan yhdistää niteen viivakoodi uuteen varastotietueeseen kohdetietokannassa. Latauksen jälkeinen tilanne siirrettyjen tietueiden osalta on viety tauluun ”Lakin aineisto mastoon siirrettynä bib_mfhd” ja tilanne lähdetietokannassa on viety tauluun ”lakki_nidelista20101207”. Nyt voidaan tunnistaa lähdetietokannan yksilöivä tietueen tunniste kohdetietokannan marc21-formaatin 035-kentästä poimittuun tietoon. Tuloksena syntyvä listaus sisältää kohdetietokannan bibliografisen tietueen ja varastotietueen tunnisteet ja niihin yhdistetyt niteen tunnisteet, kuten viivakoodi, viivakoodin status sekä sijaintitieto. Niteen sijaintitietoa hyödynnetään myöhemmässä vaiheessa, kun kirjoitetaan niteen tiedot siirtolistaukseen. Sijainnin perusteella kirjoitetaan niteen tietoihin tulevaan tekstikenttään vanhan sijainnin koodi. Tätä koodia voidaan hyödyntää, kun poimitaan tietokannasta eri korkeakoulutoimijoiden omaisuudeksi luokiteltavaa aineistoa. Tämänkin kyselyn tuloslista kannattaa viedä tauluun. Aputaulussa tietojen oikeellisuutta voidaan tarkastella ja lopulta siitä voidaan kirjoittaa nidesiirtolista.

```
SELECT [Lakin aineisto mastoon siirrettynä bib_mfhd].Masto_bib,
[Lakin aineisto mastoon siirrettynä bib_mfhd].Masto_mfhd,
lakki_nidelista20101207.ITEM_ID,
lakki_nidelista20101207.ITEM_TYPE_ID,
lakki_nidelista20101207.ITEM_BARCODE,
lakki_nidelista20101207.BARCODE_STATUS,
lakki_nidelista20101207.PERM_LOCATION
FROM lakki_nidelista20101207 INNER JOIN [Lakin aineisto mastoon siirrettynä bib_mfhd]
ON (lakki_nidelista20101207.MFHD_ID = [Lakin aineisto mastoon siirrettynä
bib_mfhd].lakki_mfhd) AND (lakki_nidelista20101207.BIB_ID = [Lakin aineisto mastoon
siirrettynä bib_mfhd].Lakki_bib)
```

Kuva 10: Lähdetietokannan ja kohdetietokannan tietueiden vastaavuuden listaava SQL-kysely.

Voyager-kirjastojärjestelmä sallii varastotietueiden ja niteiden välille kaksi vaihtoehtoista rakennetta. Kirjaston valinnan mukaisesti jokaiselle niteelle voi olla oma varastotietueensa. Vaihtoehtoisesti kaikille tietyssä sijainnissa olevilla niteillä voi olla yksi varastotietue, tämä on ns. merged-malli. Mikäli tietokantaprojektissa on tarpeellista muuttaa lähdetietueiden rakennetta 1 varastotietue/1 nide -mallista merged-malliseksi, siihen voidaan tässä vaiheessa hyödyntää mfhd_id-kentässä Microsoft Accesin min-funktiota. Kirjoitettaessa siirtotiedostoa, niteen tietoihin kirjautuu pienin saman sijainnin mukainen varastotietueen yksilöivä tunniste. Koska varastotietueet on kirjoitettu aikaisemmassa vaiheessa, merged-malliin siirtyminen aiheuttaa tässä prosessissa niteettömiä varastotietueita. Tieto näistä tietueista voidaan hakea SQL-kyselyllä ja poistaa tietojen siirtoprosessin lopuksi kohdetietokannasta.

Yhdistelykyselyn tulokset on viety Microsoft Access -tietokannan aputauluun ”nidelatauslista”. Tämä taulu asetetaan VBA-ohjelman alussa RecordSet-olioksi, jonka tauluihin viitataan myöhemmässä vaiheessa ohjelmaa. Siirtolista avataan ”Open” -komennolla kirjoitusmuotoon. RecordSet-oliosta poimitut tiedot kirjoitetaan VBA-sovelluksella csv-muotoiseen (comma-separated values) tekstitiedostoon. Tiedostossa

jokainen rivi muodostaa yhden niteen tiedot sisältävän tietueen. Jokaiseen tietueeseen kirjoitetaan luettelointipaikan numerokoodi, varastotietueen yksilöivä tunniste, niteen tyyppi sekä niteen kohdetietokannan uusi sijaintipaikka sekä niteen tietoihin tallennettava vapaatekstikenttään vietävä vanhan sijainnin mukainen tunniste. Niteiden siirtolistauksen kirjoittava VBA-sovellus on esitelty LIITE 1:ssä. ”nidelatauslista” -taulun tiedot luetaan ohjelmassa ensin muuttujiin, josta ne jokaisen ohjelmasilmukan jälkeen kirjoitetaan Print-komennolla siirtotiedostoon. Ohjelmassa asetetaan kiinteästi luettelointipaikka catloc-muuttujaan, sama tieto kirjoitetaan kaikkiin siirrettäviin niteisiin. Luettelointipaikka on järjestelmän systeemimäärityksissä oleva tapahtumasijaintipaikka, johon on määritetty luettelointisäännöstöihin liittyviä ominaisuuksia. Varastotietueen yksilöivä tunniste poimitaan taulusta mfhd-muuttujaan. Niteen tyyppimääritys muutetaan case-rakenteella ennen kuin se liitetään itemtype-muuttujaan. Projektissa haluttiin yhtenäistää niteen tyypit vastaamaan kohdetietokannan tyypitystä. Niteen tyypillä on vaikutusta muun muassa teoksen laina-aikaan ym. lainauskäytäntöihin liittyviin systeemin määrityksiin. Tässä projektissa haluttiin yhtenäistää normaalin lainattavan materiaalin (28 vuorokauden laina-aika) ja kurssimateriaaliksi luokitellun aineiston (14 vuorokauden laina-aika) luokituksia. Yhtenä tavoitteena oli pyrkimys tehostaa aineiston kiertoa asiakkailla. Niteen sijaintitiedon osalta muutos on toteutettu myös case-rakenteella. Lähdetietokannan sijaintitiedon perusteella muutetaan permloc-muuttujaan kirjoitettava tieto vastaamaan kohdetietokannan sijaintipaikkojen määritystä. Muodostuvan niteen vapaatekstikenttään haluttiin kirjoittaa saman lähtötiedon perusteella niteen omistajuudesta kertova tunniste. Tässä hyödynnetään freetext-muuttujaa. Esimerkiksi YEK-tunniste tarkoittaa ympäristöekologian laitoksen aineistoa. Näillä tunnisteilla voidaan aineistoa tilastoida ja mahdollisesti myöhempien aineistosiirtojen yhteydessä poimia tietyn organisaation aineisto. Kuvassa 11 on esitelty VBA-sovelluksen tuottamaa siirtotiedostoa.

```
95;156856;3;129;1370444539;2011 TKK - K
95;156865;3;131;1370432495;2011 YEK
95;156870;2;128;1370407834;2011 TKL
95;156886;3;131;1370431967;2011 YEK
95;156893;2;128;1370408832;2011 TKL
…
```

Kuva 11: Microsoft Accessin VBA -sovelluksen tuottama siirtotiedosto nidetiedoille.

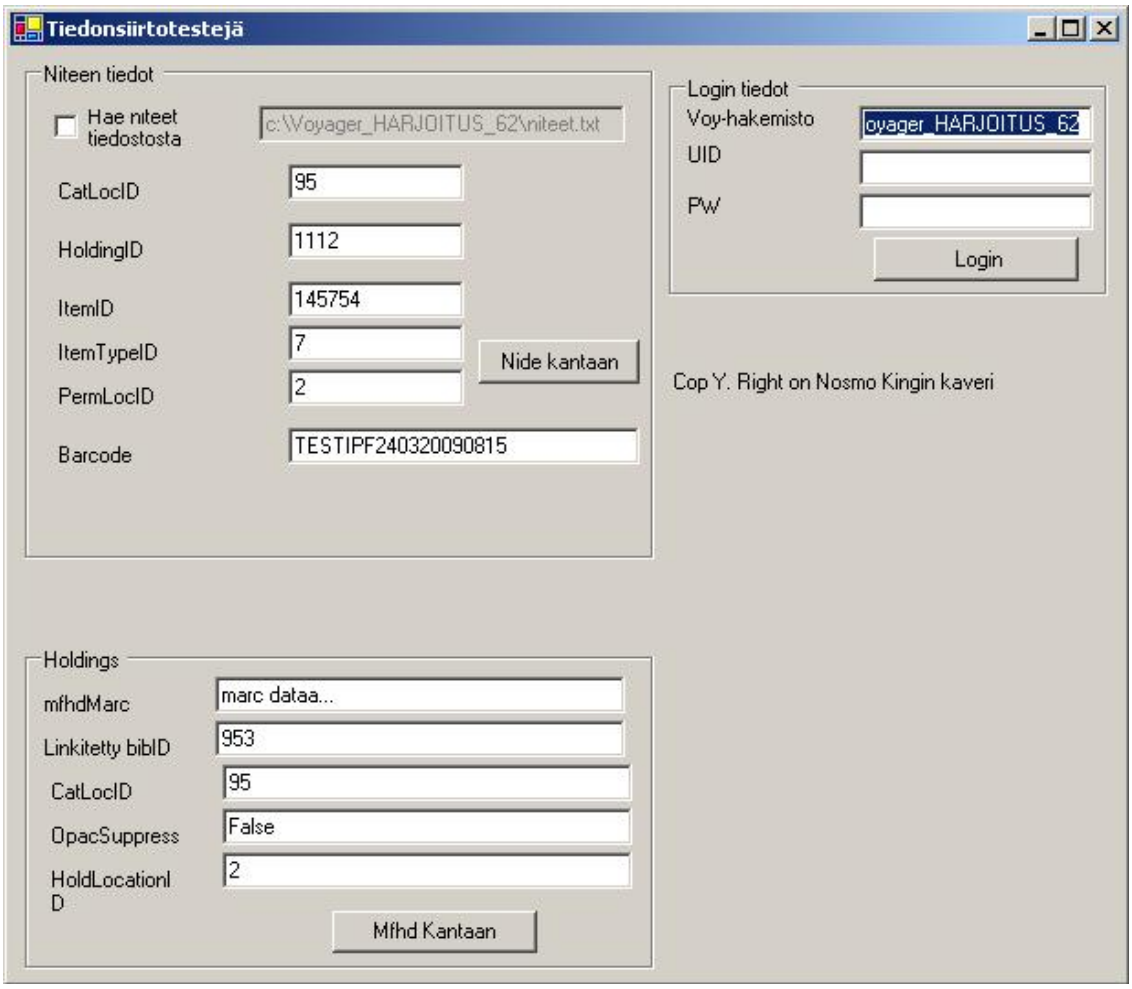
3.1.6 Niteiden lataus kohdekantaan

Nidetietojen lataus tietokantaan tapahtuu Visual Basic -sovelluksella. Sovellus on dokumentoitu kokonaisuudessaan LIITE2:ssa. Batchcat-ohjelmointirajapinta on Ex Libris -yhtiön toimittama sovelluslaajennus luettelointitietojen siirtämiseksi tietokantaan. Fyysisesti batchcat.dll -tiedosto löytyy työasemalta asiakasohjelman asennushakemistosta käyttöohjeineen. Ohjelmointirajapinta mahdollistaa luettelointidatan käsittelyn kaikkien kolmen tason tietueiden osalta. Omat funktiot ovat olemassa sekä bibliografiselle datalle että varastotietueille. Koska tässä projektissa päädyttiin hyödyntämään järjestelmän omia eräajo-ohjelmia, näitä funktioita ei ole tarpeellista käsitellä tämän projektin yhteydessä. Nidetietojen lataamisessa tietokantaan on tässä sovelluksessa hyödynnetty rajapinnan sisältämiä funktioita AddItemData ja AddItemBarcode. Lisäksi kirjautuminen järjestelmään luettelointioikeuksilla varustetulla tunnuksella on hyödynnetty rajapinnan funktioista.

Kuvassa 12 on esitetty Visual Basic -sovelluksen käyttöliittymä. Koska kyseessä on kertaluonteinen toiminto, käyttöliittymän suunnitteluun ei ole tässä projektissa panostettu. Ohjelman toiminta on suoraviivaista. Ohjelman toiminta alkaa ”Login tiedot” osasta. ”Voy-hakemisto” -kenttään kirjoitetaan työasemaan asennetun Voyager -asiakasohjelman asennushakemisto. UID-kenttään kirjoitetaan luetteloivan virkailijan tunnus. Tunnus on määritelty Voyager-kirjastojärjestelmään niin, että sillä on riittävät oikeudet luetteloida bibliografisia tietueita sekä nidetietoja tietokantaan niissä sijaintipaikoissa joihin tietojen lisääminen kohdistuu. ”Niteen tiedot” -osassa suoritetaan varsinainen niteiden kirjoittaminen tietokantaan. Käyttöliittymässä on tekstikenttiä yksittäisten nidetietojen käsittelemiseksi. Näitä kenttiä on käytetty testausvaiheessa. Varsinaista tietokantaprojektia varten sovellukseen on tehty valinta, jolla voidaan valita hakemisto, jossa niteiden siirtotiedosto sijaitsee. Ohjelma käy siirtotiedoston läpi tietue kerrallaan ja kirjoittaa tiedon tietokantaan. Niteiden ja niteiden viivakoodien lisäyksen onnistumisesta ja epäonnistumisesta sovellus kirjaa tiedon seurantatiedostoon. Seurantatiedostossa on tallennettuna niteen tunnisteiden lisäksi mahdollinen syykoodi ja sen selkokielinen merkitys mahdollisesta virheestä luettelointitapahtuman yhteydessä. Ohjelmassa on myös varastotietueen kirjoittamisen mahdollistava osa, mutta se on ollut käytössä vain projektin

alkuvaiheen testauksissa.

Visual Basicissa Batchcat-rajapinnan hyödyntäminen alkaa sovelluslaajennuksen lataamisella Visual Studion laajennuksiin. Kun laajennus on esitelty ohjelmakoodissa, luetteloinnin funktiot ovat hyödynnettävissä ohjelmakoodissa.



Kuva 12: Nidetiedot tietokantaan kirjoittavan ohjelman käyttöliittymä.

Kirjautuminen Voyager-kirjastojärjestelmään toteutetaan Batchcat-luettelointirajapinnan Connect-funktiolla. Funktion parametreiksi syötetään Voyager-asiakasohjelman asennushakemisto, Voyager-kirjastojärjestelmään määritetty luetteloinnin käyttäjätunnus ja salasana. Tähän projektiin kehitetyssä sovelluksessa kyseiset tiedot kysytään käyttäjältä käyttöliittymän tekstikentissä. Toiminnon aliohjelma on esitelty kuvassa 13. Rajapinta palauttaa arvon kirjautumisen onnistumisesta. Mahdollisesta epäonnistumisesta ohjelma

palauttaa virhekoodin ja ohjelmaan on mahdollista ohjelmoida selkokielinen selvitys virheen syystä. Tällainen syy voi johtua tietokantayhteyden puuttumisesta tai vääristä käyttäjätunnisteista. Tässä projektissa nähtiin tarpeelliseksi kertoa käyttäjälle vain kirjautumisen onnistuminen. Epäonnistumisen palauttavat koodit kuitataan vain ilmoituksella ”Kirjautuminen ei onnistunut” ja samalla suljetaan ohjelma.

```
Private Sub btn_login_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_login.Click
  Dim AppPath = txtVoyHak.Text
  Dim UserName = txtUID.Text
  Dim Password = txtPW.Text
  Dim login = MyBatchCat.Connect(AppPath, UserName, Password)
        If login <> iBatchCatSuccess Then
              MsgBox("Kirjautuminen ei onnistunut")
              Exit Sub
        Else
              MsgBox("Kirjautuminen toimii")
              btn_login.Enabled = False
              btn_login.Text = "Yhteydessä..."
        End If
End Sub
```

Kuva 13: Kirjautuminen Voyager-kirjastojärjestelmään ohjelmallisesti.

Nidetietojen kirjoittamisen onnistumista tietokantaan voidaan seurata jokaisen tietueen kirjoittamisen yhteydessä. Projektin sovelluksessa tieto kirjoitetaan seurantatiedostoon. Seurannan osalta lähdekoodin on esitelty kuvassa 14. Ohjelman logiikka kirjoittaa tietokantaan ensin niteen tiedot ja sen jälkeen viivakoodin tunnisteen. Molemmista toiminnoista saadaan erikseen myös paluukoodi onnistumisesta. Kuten kirjautumisenkin onnistumisesta, myös niteen tietojen kirjoittamisessa on tyydytty seuraamaan vain tietokantaan kirjoituksen onnistumisen. Epäonnistuneesta nidetietojen ja viivakoodin tietokantaan kirjoittamisesta kertovat vikakoodit kuitataan yhdellä ilmoituksella ja kirjoitetaan seurantatiedostoon. Mahdolliset ongelmatapaukset pitäisi kuitenkin tarkastaa tapauskohtaisesti. Mahdollisia syitä tietojen viennin epäonnistumiselle voisi olla esimerkiksi varastotietueen väärä tunniste, väärä niteen tyyppi tai se että niteen sijaintipaikkaa ei löydy.

```
…
FileOpen(2, "niteenlatauslogi20101220.txt", OpenMode.Append, OpenAccess.ReadWrite)
…
…
If vie = 0 Then
        'MsgBox("niteen vienti onnistui")
        Print(2, MyBatchCat.RecordIDAdded & " niteen vienti onnistui " & vie)
Else
        'MsgBox("niteen vienti epäonnistui")
        Print(2, MyBatchCat.RecordIDAdded & " niteen vienti epäonnistui - virhekoodi " & vie)
End If
        lisatty_nide = MyBatchCat.RecordIDAdded
        vie = MyBatchCat.AddItemBarCode(lisatty_nide, Nidearray(4))
If vie = 0 Then
        'MsgBox("viivakoodin vienti onnistui")
        Print(2, lisatty_nide & " viivakoodin vienti onnistui -" & vie)
Else
        'MsgBox("viivakoodin vienti epäonnistui")
        Print(2, lisatty_nide & " viivakoodin vienti epäonnistui - virhekoodi " & vie)
End If
…
…
```

Kuva 14: Nidetietojen tietokantaan kirjoittamisen seurantatoiminnot sovelluksessa.

Nidetietojen kirjoittaminen tietokantaan onnistuu muutamalla rivillä ohjelmakoodia. Nidetiedot sisältävä csv-muotoinen (comma separated value) siirtotiedosto avataan lukutilassa sovelluksen käyttöön. Jokaisen tietueen osalta tietueen kentät puretaan Split-komennolla tietojoukoksi, jonka jälkeen tietojoukon osat kirjoitetaan cItem-muuttujiin. Kun nide on luotu tietokantaan annetuilla tiedoilla, rajapinta palauttaa kirjoitetun niteen yksilöivän tunnisteen tietokannan taulusta. Tätä niteen tunnistetta käytetään niteen viivakoodin kohdistamiseen oikeaan niteeseen tietokannassa. Niteellä voi tietokannassa olla useampia viivakoodeja. Tietokanta antaa myös mahdollisuuden sille, että sama viivakooditunniste on useamman niteen niteen tiedoissa, eli tietokannassa kyseinen kenttä ei ole yksilöivä avainkenttä. Laadunvarmistuksen kannalta muodostuvan seurantatiedoston tarkistuksella ja toisaalta SQL-kyselyllä suoritettavalla vertailulla kummankin Oracle-tietokannan nidetauluihin varmennetaan, että kirjoitettujen niteiden tiedot ovat kirjautuneet oikeaan nimekkeeseen ja varastotietueeseen.

```
…
FileOpen(1, txtNidelista.Text, OpenMode.Input)
   Do Until EOF(1)
         Dim Text As String = LineInput(1)
         Dim Nidearray() As String = Split(Text, ";")
         MyBatchCat.cItem.HoldingID = Nidearray(1)
         MyBatchCat.cItem.ItemTypeID = Nidearray(2)
         MyBatchCat.cItem.PermLocationID = Nidearray(3)
         MyBatchCat.cItem.FreeText = Nidearray(5)
         vie = MyBatchCat.AddItemData(Nidearray(0))
…
         lisatty_nide = MyBatchCat.RecordIDAdded
…
         vie = MyBatchCat.AddItemBarCode(lisatty_nide, Nidearray(4))
…
   Loop
FileClose(1)
…
```

Kuva 15: Nidetiedot tietokantaan kirjoittavat toiminnot sovelluksessa.

Kun niteet on saatu ladattua tietokantaan, voidaan tehdä vertailua lähdetietokannan ja kohdetietokannan välillä. Tämä voidaan tehdä esimerkiksi Microsoft Access -tietokannassa, jossa on linkitettynä molempien kirjastojärjestelmien Oracle-tietokantojen taulut. Lahden projektissa havaittiin jonkin verran puuttuvia niteitä ensimmäisen ajon jälkeen. Puuttuvien niteiden osalta syyksi osoittautui bibliografisen tietueen hylkäys ensimmäisten latausten yhteydessä. Tällaisten tietueiden osalta prosessia muutettiin niin, että bibliografiset tietueet lisättiin tietokantaan ilman tuplatarkistusta uusina tietueina, jotta aineiston data saatiin tietokantaan eheänä. Bibliografisten tietueiden osalta muodostuneet tuplatietueet on käsitelty jälkisiivouksessa myöhemmin. Bibliografisten tietueiden siivous voidaan toteuttaa esimerkiksi automatisoimalla asiakasohjelman toimintoja makrojen avulla.

3.2 Asiakastiedot

Kirjastojen asiakasrekisteri on henkilörekisteri. Kun henkilörekisterin tietoja käsitellään, on huomioitava mitä laki sanoo henkilörekisterin käsittelystä. Henkilörekisterin tietojen käsittelyä ohjaa Henkilötietolaki 22.4.1999/523. Laki määrittelee tietojen käsittelyä koskevat yleiset periaatteet. Lain mukaan käsittelyssä on noudatettava huolellisuusperiaatetta. Lisäksi tietoja tulee käsitellä niin, ettei rekisteröidyn henkilön yksityisyyden suojaa loukata. Henkilötietojen käsittelyn tulee olla asiallisesti perusteltua toiminnan kannalta. Henkilötietoja saa käyttää vain rekisterin alkuperäisen käsittelyn tarkoituksiin, josta poikkeuksena hyödyntäminen tutkimuksellisiin tarpeisiin. Henkilötietojen käsittelyn tulee olla määritellyn tarkoituksen kannalta tarpeellista. Vanhentuneita tai virheellisiä tietoja ei käsitellä. (Finlex, henkilötietolaki, 1999)

Yhdistettäessä kaksi samaan tarkoitukseen perustettua henkilörekisteriä, rekisterin käyttötarkoitus ei muutu. Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen henkilörekisteristä pyritään poistamaan tarpeettomat tiedot säännöllisin väliajoin. Käytännön mukaisesti kolme vuotta vanhat asiakastiedot siivotaan rekisteristä, mikäli asiakkaalla ei ole tapahtumia kyseisenä aikana. Samaa periaatetta haluttiin noudattaa myös Lahden Tiedekirjaston asiakastietojen osalta. Lähdetietokannasta tehdyssä kyselyssä asetettiin kolmen vuoden aikainen tapahtumaehto tietojen siirtämiselle kohdetietokantaan. Samoin pyrittiin välttämään vajaita tietueita.

Tietojen ulosluku lähdetietokannasta tehdään Microsoft Access -tietokannassa, jossa voidaan kirjoittaa myös siirtotiedosto VBA-sovelluksella. Asiakastietojen sisään luku Voyager-kirjastojärjestelmään tapahtuu ptrnupdt-rajapinnan (patron update) kautta. Samaa prosessia hyödynnetään kohdetietokannassa päivittäin Lahden ammattikorkeakoulun ja Koulutuskeskus Salpauksen opiskelija-asiakkaiden osoitetietojen päivityksessä opintohallintorekisteristä. Tässä projektissa toteutetussa siirrossa haettiin kyselyillä erikseen uudet asiakkaat sekä kohdekannassa ja lähdekannassa olevat samat asiakkaat. Koska Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalveluilla sekä Lahden tiedekirjastolla on ollut osittain samat asiakassegmentit, kohdetietokannassa oleville asiakkaille siirrettiin kirjastokortin tiedot päivityksenä olemassa oleviin tietoihin, mikäli

tiedot jo kohdetietokannasta löytyivät. Muiden asiakkaiden osalta tiedot lisättiin tietokantaan uusina asiakkaina.

Kuvassa 16 on esitelty SQL-kysely asiakkaiden poimimiseksi lähdetietokannasta. Asiakkaiden osoitteet on haettu aputauluihin (lakki_emailosoitteet, lakki_postiosoitteet) kyselyn nopeuttamiseksi. Voyagerissa sekä postiosoite että sähköpostiosoite on tallennettu yhteen tauluun. Toinen syy eriyttää nämä tiedot erilliseen tauluun on mahdollisuus muodostaa relaatio asiakkaan perustiedoista molempiin osoitetyyppeihin yksinkertaisemmalla SQL-kyselyllä. Asiakkuuden aktiivisuutta testataan hakemalla viimeinen lainauspäivämäärä sekä tutkimalla asiakastietueen viimeistä muokkauspäivää.

```
SELECT DISTINCT "LAKKI" AS Lauseke1,
LAKKIDB_PATRON_BARCODE.PATRON_BARCODE,
LAKKIDB_PATRON_BARCODE.BARCODE_STATUS,
LAKKIDB_PATRON_BARCODE.PATRON_ID, LAKKIDB_PATRON.PATRON_ID,
LAKKIDB_PATRON.LAST_NAME, LAKKIDB_PATRON.FIRST_NAME,
LAKKIDB_PATRON.INSTITUTION_ID, LAKKIDB_PATRON.CREATE_DATE,
LAKKIDB_PATRON.EXPIRE_DATE, LAKKIDB_PATRON.CURRENT_CHARGES,
lakki_postiosoitteet.ADDRESS_TYPE, lakki_postiosoitteet.ADDRESS_LINE1,
lakki_postiosoitteet.ADDRESS_LINE2, lakki_postiosoitteet.CITY,
lakki_postiosoitteet.ZIP_POSTAL, lakki_postiosoitteet.COUNTRY,
lakki_emailosoitteet.ADDRESS_TYPE, lakki_emailosoitteet.ADDRESS_LINE1 INTO
lakki_aktiiviset_asiakkaat20101208
FROM (((LAKKIDB_PATRON INNER JOIN LAKKIDB_CIRC_TRANS_ARCHIVE ON
LAKKIDB_PATRON.PATRON_ID = LAKKIDB_CIRC_TRANS_ARCHIVE.PATRON_ID)
INNER JOIN LAKKIDB_PATRON_BARCODE ON LAKKIDB_PATRON.PATRON_ID =
LAKKIDB_PATRON_BARCODE.PATRON_ID) LEFT JOIN lakki_emailosoitteet ON
LAKKIDB_PATRON.PATRON_ID = lakki_emailosoitteet.PATRON_ID) LEFT JOIN
lakki_postiosoitteet ON LAKKIDB_PATRON.PATRON_ID =
lakki_postiosoitteet.PATRON_ID
GROUP BY "LAKKI", LAKKIDB_PATRON_BARCODE.PATRON_BARCODE,
LAKKIDB_PATRON_BARCODE.BARCODE_STATUS,
LAKKIDB_PATRON_BARCODE.PATRON_ID, LAKKIDB_PATRON.PATRON_ID,
LAKKIDB_PATRON.LAST_NAME, LAKKIDB_PATRON.FIRST_NAME,
LAKKIDB_PATRON.INSTITUTION_ID, LAKKIDB_PATRON.CREATE_DATE,
LAKKIDB_PATRON.EXPIRE_DATE, LAKKIDB_PATRON.CURRENT_CHARGES,
LAKKIDB_PATRON.CURRENT_CHARGES, lakki_postiosoitteet.ADDRESS_TYPE,
lakki_postiosoitteet.ADDRESS_LINE1, lakki_postiosoitteet.ADDRESS_LINE2,
lakki_postiosoitteet.CITY, lakki_postiosoitteet.ZIP_POSTAL,
lakki_postiosoitteet.COUNTRY, lakki_emailosoitteet.ADDRESS_TYPE,
lakki_emailosoitteet.ADDRESS_LINE1, LAKKIDB_PATRON.MODIFY_DATE,
LAKKIDB_PATRON.CREATE_DATE, LAKKIDB_PATRON.CURRENT_CHARGES
HAVING Max(LAKKIDB_CIRC_TRANS_ARCHIVE.DISCHARGE_DATE)>Date()-([raja
vuosina:]*365) OR (LAKKIDB_PATRON.MODIFY_DATE)>Date()-([raja vuosina:]*365) OR
(LAKKIDB_PATRON.CURRENT_CHARGES)>"0" OR
(LAKKIDB_PATRON.CREATE_DATE)>Date()-([raja vuosina:]*365);
```

Kuva 16: SQL-kysely asiakkaiden poimimiseksi lähdetietokannasta.

Liite 3:ssa on esitelty VBA-sovellus, jolla asiakastiedot voidaan kirjoittaa SIF-muotoiseksi siirtotiedostoksi. Standard interface file, SIF, on siirtotiedoston formaatti, jota voidaan hyödyntää Voyager-kirjastojärjestelmään siirryttäessä, olemassa olevan asiakastiedon päivittämisen sekä asiakastiedon uloslukemiseen Voyagerista (Voyager 7.0 Technical User's Guide, 2009). Taulukossa 1 on poimintoja Voyager-kirjastojärjestelmän asiakastietojen siirtoformaatista. Tietueen kentät ovat kiinteämittaisia ja jokaisesta kentästä on kerrottu, monennestako merkistä tietue alkaa ja montako merkkiä sen pituus on. Lisäksi jokaisen kentän osalta on kerrottu sen pakollisuus siirtotiedostossa. Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen ja Lahden tiedekirjaston kirjastojärjestelmän tietokannoissa asiakkaan yksilöinti tapahtuu ”institution ID” -kentän perusteella. ”Institution ID” -kenttään on molemmissa tietokannoissa tallennettu asiakkaan henkilötunnus. Tämän kentän perusteella tässä tietokantaprojektissa haetaan molemmissa tietokannoissa olevat samat asiakkaat ja päivitetään asiakkuuteen liittyvät tiedot kohdejärjestelmässä.

Taulukko 1: Esimerkki kenttämäärityksistä Voyager-kirjastojärjestelmän asiakastietojen siirtoformaatissa (Voyager 7.0 Technical User's Guide, 2009).

Item #	Item Name	Offset	Format	Required	Length
1	patron id	1	n		10
2	patron barcode id 1	11	n*		10
3	patron barcode 1	21	s		25
4	patron group 1	46	s		10
...					
23	institution ID	239	s		30
…					
53	address id	457	n		10
54	address type	467	n	y	1
55	address status code	468	s	y	1
...					
58	address line 1	489	s	y	50
...					

Projektissa siirrettävät asiakkaat tallennettiin SQL-kyselyn määrityksen mukaisesti tauluun ”lisattavat_patronit27012011”. Siirtotiedoston muodostavan VBA-koodin aluksi kyseinen taulu asetetaan RecordSet-olioksi komennolla ”Set rs =

CurrentDb.OpenRecordset("lisattavat_patronit27012011")”. Kun RecordSet-olio on asetettu, tietueen kentissä olevat tiedot voidaan lukea muuttujiin. Sovelluksen lopussa muuttujista koostetaan siirtotietoon kirjoitettava rivi ja viedään se siirtotiedostoon. Tämän jälkeen ohjelmassa ajetaan Do While -ohjelmasilmukka, kunnes viimeinen RecordSetin tietue on käsitelty. Kuvassa 17 kuvataan osa Liitteessä 3 esitellystä ohjelmakoodista. Siirtotiedoston määrityksissä mainittu tarvittava kentän pituus asetetaan VBA-funktiolla String. Parametreina kerrotaan toistettavien merkkien lukumäärä sekä merkki joka halutaan toistaa. Esimerkiksi ptrID-muuttujaan kirjoitetaan koko kentän pituudelta ASCII-merkistön (American Standard Code for Information Interchange) arvo 32, eli välilyönti. Niissä kentissä, joihin kirjoitetaan päivitettävää tai siirrettävää tietoa, kentän pituutta testataan sovelluksessa. Mikäli kirjoitettava arvo ylittää sallitun kentän pituuden, se katkaistaan kenttämäärityksen mukaiseen pituuteen. Mikäli kirjoitettavan arvon merkkimäärä on vähemmän kuin sallittu merkkimäärä, kirjoitetaan kirjoitettavan arvon perään riittävästi välilyönti-merkkiä täyttämään kentän pituuden. Sellaiset kentät, joiden merkkimäärän tiedetään jäävän alle maksimimäärän, kirjoitetaan vain arvo ja tarpeellinen määrä välilyöntejä täyttämään kentän vaatimukset. Uutena siirtyville asiakkaille asetettiin asiakkuuden vanhentumispäivä niin, että asiakkaan yhteystiedot tulee tarkastettua asiakkuuden siirron alkuvaiheessa. Myös asiakkaan tilastointiryhmien tarkistukset nähtiin tärkeäksi osana kirjastojen toimintojen yhtenäistämistä. Lahden tiedekirjaston asiakkaille määritettiin tietojen siirron yhteydessä tilastointikoodiksi tiedekirjaston asiakkuudesta kertova tilastoryhmä. Tilastoryhmiä on tarkennettu myöhemmässä vaiheessa vastaamaan Lahden yliopistokeskuksen toiminnassa olevia yliopistotoimijoita.

Tilastoryhmiä määriteltäessä on huomioitava vuosittaiset kirjastosektoreiden yhteiset tilastointitarpeet. Määrittelyn peritaatteita ei ole tarvetta käsitellä tämän työn yhteydessä, koska kyseessä on erillinen ja jatkuvaluonteinen prosessi. Kuitenkin kirjastojen tietokantoja yhdistettäessä on pohdittava tilastoissa jaoteltavien asiakasryhmien määrittelyä. Vuositilastoinnissa erotellaan korkeakoulujen oma henkilökunta ja omat opiskelijat muiden korkeakoulujen vastaavista. Tietokantaprojektin yhteydessä määritys olisi hyvä olla olemassa kohdetietokantaan kirjoitettaessa.

```
Open "lakki_uudet_asiakkaat27012011.sif" For Output As #1
Do While Not rs.EOF
        ptrID = String(10, " ")      ‘ järjestelmä käsittelee sisäisesti
…
        ptrbarcode1 = rs.Fields("patron_barcode") & String((25 -
Len(rs.Fields("patron_barcode"))), " ") ‘kirjastokortin viivakoodi
        ptrgrp1 = "AS" & String(8, " ") ' asiakasryhmä
        barcodestat1 = "1" ' viivakoodin statukseksi "active"
        ptrexprdate = "2012.01.31"  ’asiakkuuden vanheneminen
        instID = rs.Fields("institution_id") & String((30 - Len(rs.Fields("institution_id"))), " ")
        ssan = String(11, " ") ’ päivityksessä käytettävä yksilöivä tunniste
…
        statgat1 = "ZZ "                ‘tilastointiryhmä
…
        If Len(rs.Fields("last_name")) < 30 Then ‘osoite ja nimitietojen käsittelyesimerkki
        surname = rs.Fields("last_name") & String((30 - Len(rs.Fields("last_name"))), " ")
        Else
        surname = Left(rs.Fields("last_name"), 30)
        End If
        If Len(rs.Fields("first_name")) < 20 Then
…
        ' kasataan merkkijono tiedostoon kirjoitettavaksi
        conv_data = ptrID & ptrbarcodeID1 & ptrbarcode1 & _
        ptrgrp1 & barcodestat1 & … & addrstate_2 & zipcode_2 & country_2 & _
        primphone_2 & otherphones_2 & dateadded_2
Print #1, conv_data
rs.MoveNext
Loop
rs.Close
Set rs = Nothing
Close #1
```

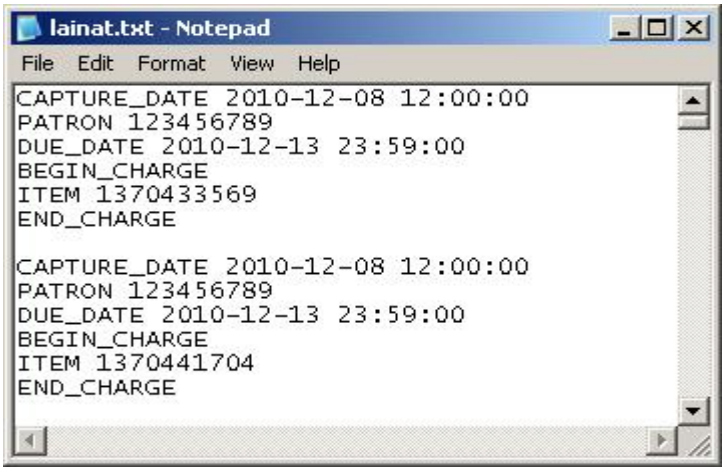
Kuva 17: Poimintoja asiakastietojen siirtotiedoston muodostavasta ohjelmasta.

Asiakastietojen käsittelyssä on pieniä eroavaisuuksia kun käsitellään uutena siirrettäviä tietoja ja kohdejärjestelmässä jo olevia tietoja. Uudet tietueet voidaan kirjoittaa tietokantaan tarkistusten ja testien jälkeen sellaisenaan ja niistä muodostuu uusi asiakas ja

uudelle asiakkaalle uusin kirjastokortin viivakoodi. Kohdejärjestelmässä jo olevien asiakastietojen kohdalla on huomioitava, kummassa tietokannassa on tuoreempi osoitetieto. Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen kirjastojärjestelmään päivitetään Lahden ammattikorkeakoulun ja Koulutuskeskus Salpauksen opiskelijoiden osoitetiedot päivittäin eräajona opintohallintojärjestelmistä. Lisäksi muiden asiakkaiden osoitetiedot on pyritty tarkistamaan vuosittain. Siksi projektissa päätettiin pyrkiä säilyttämään olemassa olevat asiakkaan yhteystiedot, mikäli sellaiset on järjestelmään kirjattu. Tällaisten asiakkaiden osalta vain tilastoryhmä päivitettiin. Mikäli kirjastokortin viivakoodissa löytyi eroavaisuutta, Lahden Tiedekirjaston tietokannassa oleva kirjastokortin viivakoodi lisättiin asiakkaan tietoihin. Ongelmallisiksi tietueiksi tässä tietokantaprojektissa osoittautuivat instituutioasiakkaiden, kuten esimerkiksi muiden kirjastojen kaukolainapalveluiden tietueet. Koska tällaisilla tapauksilla ei ole ”institution ID” -kentässä mitään yksilöivää tunnistetta, tällaiset tietueet käsiteltiin erillisellä siirrolla ja täydennettiin manuaalisessa prosessissa.

3.3 Lainatiedot

Asiakkaiden lainatietoja siirrettäessä on huomioitava tietojen oikeellisuus sekä asiakkaan oikeusturvan että kirjaston omaisuuden hallinnan kannalta. Asiakkaiden lainatiedot voidaan lukea kohdetietokantaan Voyager -kirjastojärjestelmän offline-lainausta hyväksi käyttäen. Kirjastot voivat käyttää offline-toimintoa lainaustapahtumiin ja lainausten uusintatoimintoihin, kun verkkoyhteys järjestelmän tietokantaan ei jostain syystä ole käytettävissä (Voyager 7.0 Circulation User’s Guide, 2009). Käytettäessä offline-lainausta, järjestelmä kirjaa lainatapahtuman työasemalle asiakasohjelman asennushakemistoon. Kuvassa 18 on esitelty esimerkki tällaisesta tiedostosta. Tiedostoon listataan lainauspäivä, asiakkaan kirjastokortin viivakoodi, asetettu eräpäivä sekä lainattujen teosten viivakoodi-tunniste. Kun verkkoyhteys tietokantaan palautuu käytettäväksi, offline-tiedosto luetaan lainauksen asiakasohjelman kautta tietokantaan.



Kuva 18: Offline-lainauksen muodostama tiedosto.

Lainatietojen poimiminen lähdetietokannasta toteutettiin Microsoft Access -tietokannassa SQL-kyselyllä. Tämänkin kyselyn tulostietueet vietiin aputauluun, josta ne voitiin VBA-sovelluksella kirjoittaa offline-muotoiseen tiedostoon. SQL-kysely on esitelty kuvassa 19. Tuloksissa poimitaan tarvittavat kentät, kuten asiakkaan kirjastokortin viivakoodi, lainan eräpäivä sekä niteen viivakoodi. Koska lainatieto kiinnittyy asiakastietueeseen eikä viivakoodiin, ehdoksi asetettiin sekä asiakkaan kirjastokortin viivakoodille että niteen viivakoodille aktiivinen status. Kyselyn tulosjoukkoon on muussa tapauksessa riskinä tulla myös mm. kadonneeksi merkityt kirjastokortit. Käytännössä laina kiinnittyisi oikeaan asiakkaaseen myös muilla kuin aktiivisen statuksen viivakoodilla, mutta tällä menettelyllä vältetään virheet lainatietojen sisäänluku-vaiheessa.

```
SELECT LAKKIDB_PATRON_BARCODE.PATRON_BARCODE,
LAKKIDB_PATRON_BARCODE.BARCODE_STATUS,
LAKKIDB_CIRC_TRANSACTIONS.CURRENT_DUE_DATE,
LAKKIDB_ITEM_BARCODE.ITEM_BARCODE,
LAKKIDB_ITEM_BARCODE.BARCODE_STATUS INTO lakki_lainat20101208
FROM LAKKIDB_ITEM_BARCODE INNER JOIN ((LAKKIDB_CIRC_TRANSACTIONS
INNER JOIN LAKKIDB_PATRON ON LAKKIDB_CIRC_TRANSACTIONS.PATRON_ID =
LAKKIDB_PATRON.PATRON_ID) INNER JOIN LAKKIDB_PATRON_BARCODE ON
LAKKIDB_PATRON.PATRON_ID = LAKKIDB_PATRON_BARCODE.PATRON_ID) ON
LAKKIDB_ITEM_BARCODE.ITEM_ID = LAKKIDB_CIRC_TRANSACTIONS.ITEM_ID
WHERE (((LAKKIDB_PATRON_BARCODE.BARCODE_STATUS)="1") AND
((LAKKIDB_ITEM_BARCODE.BARCODE_STATUS)="1"))
```

Kuva 19: SQL-kysely lainatietojen poimimiseksi lähdetietokannasta.

Kuvassa 20 esitellään offline-tiedoston kirjoittavan VBA-sovelluksen toiminta. Toiminnallisuus noudattaa samaa kaavaa, kuin aikaisemmissakin VBA-sovelluksissa tässä projektissa. Ensin avataan RecordSet-olioon tietokannan taulu. Lainatietoja poimittaessa tulokset vietiin tauluun ”lakki_lainat20101208”. Seuraavaksi avataan tekstitiedosto kirjoitusmuodossa. Tietueet käydään ohjelmasilmukassa läpi kunnes päästään tietueiden loppuun, jonka jälkeen RecordSet-olio ja tekstitiedosto suljetaan.

```
Sub lainat()
    Set rs = CurrentDb.OpenRecordset("lakki_lainat20101208")
    Open " lainat.txt" For Output As #1
    'luettelointipaikka, holdingid, nidetyyppi, permlocation, barcode, freetext
    Do While Not rs.EOF
        Print #1, "CAPTURE_DATE 2010-12-08 12:00:00"
        Print #1, "PATRON " & rs.Fields("patron_barcode").Value
        Print #1, "DUE_DATE " & Format(rs.Fields("current_due_date").Value,
        "YYYY-MM-DD") & " 23:59:00"
        Print #1, "BEGIN_CHARGE"
        Print #1, "ITEM " & rs.Fields("item_barcode").Value
        Print #1, "END_CHARGE"
        Print #1, ""
    rs.MoveNext
    Loop
    Close #1
    rs.Close
End Sub
```

Kuva 20: Offline-lainaustoimintoa jäljittelevän lainaustiedoton muodostava VBA-sovellus.

3.4 Asiakkaiden maksutiedot sekä aineistopyynnöt

Asiakastietoihin ja lainatietoihin kiinteästi liittyvät asiakkaiden maksutiedot sekä aineiston varaustiedot on Lahden kirjastojärjestelmien yhdistämisprojektissa hoidettu projektin

viimeisessä vaiheessa manuaalisesti asiakasohjelman kautta syöttäen. Lahden Tiedekirjastossa käsiteltiin asiakkaiden maksuja tehostetusti ennen tietokantojen yhdistämisprosessia, tästä johtuen merkittävien maksutietojen määrä lähdetietokannassa oli melko vähäinen. Samoin mahdollisten perintämenettelyn piirissä olevien aineistojen käsittely tapahtui ennen tietokantaprojektia. Tällainen aineisto huomioitiin projektissa myös niteiden siirron yhteydessä. Aineistopyyntöjen määrä katsottiin myös pienessä tietokannassa melko vähäiseksi, joten ohjelmallisen prosessin luominen ei välttämättä olisi tuonut mitään lisäarvoa näiden tietojen siirtämiseen. Aineistopyyntöjen tiedot haettiin projektin aikana SQL-kyselyllä lähdetietokannasta. Tietojen syöttö tietokantaan tapahtuu normaalisti aineistopyynnön toiminnoilla asiakasohjelman kautta.

4 POHDINTA JA TULEVAISUUS

Korkeakoulukirjastojen verkostoa muokkaava rakenteellinen kehitys luo tarpeita miettiä myös kirjastojärjestelmien toimintojen yhdistämistä. Viitetietokantojen ylläpitäminen suurempien kokonaisuuksien tarpeisiin vähentää järjestelmien ylläpitoon tarvittavan henkilökunnan tarvetta sekä järjestelmiin liittyvien kustannusten pienenemistä. Vapautuvat resurssit voidaan kohdistaa ydintoiminnan kehittämiseen. Myös omatoimisen tietokantojen yhdistämisen yhtenä motiivina voidaan nähdä kustannussäästöjä. Teetettäessä järjestelmien yhdistäminen järjestelmätoimittajalla hinnoittelu perustuu järjestelmätoimittajan ennalta määrittämiin perusteisiin. Hinnan muodostus riippuu nimekkeiden lukumäärästä, varastotietueiden käsittelytarpeesta, asiakastietojen käsittelystä jne. Kuitenkin asiakasorganisaatiolle kuuluu tässäkin vaihtoehdossa etukäteissuunnittelu, kuten aineistojen ja asiakastietojen kartoitus sekä systeemiin tehtävät määritykset. Lisäksi mm. asiakasliittymän määrittely ja muokkaaminen vastaamaan uutta palvelutilannetta jää asiakasorganisaatiolle. Koska omaa osaamista joudutaan joka tapauksessa kiinnittämään tietokantaprojektiin, on hyvä miettiä myös varsinaisen tiedonsiirron tekemistä omaan osaamiseen luottaen. Von Hippel (2001) on artikkelissaan pohtinut käyttäjälähtöisen innovaatiotoiminnan hyötyjä. Kun tarvepohjaista suunnittelua siirretään järjestelmätoimittajalta asiakasorganisaatiolle, hyödytään sekä ajallisesti että taloudellisesti. Lisäksi käyttäjäorganisaatioon liittyvä osaaminen saadaan huomaamattomasti hyödynnettyä järjestelmään liittyvässä kehitystyössä. Batchcat-luettelointirajapinnan voidaan nähdä kehittyneen käyttäjäinnovaation perusteella. Voyager-kirjastojärjestelmään on kehitetty sovelluslaajennus mahdollistamaan massakorjaukset tietokannan bibliografisessa datassa. Rajapinnan toiminnallisuutta on kehitetty yhteistyössä asiakasorganisaatioiden kanssa. Lisäksi laajassa korkeakoulukirjastojen käyttäjäyhteisössä jaetaan osaamista. Von Hippelin (2001) mukaan kerran kehitetty kehitystyökalu asiakasyhteisölle voidaan hyödyntää laajemminkin. Kerran kehitettynä rajapintaa voidaan hyödyntää useiden käyttäjätahojen yksilöllisiä tarpeita palvelevassa kehitystoiminnassa. Aikataulujen osalta järjestelmätoimittajan kanssa tehtävä projekti sitoo vahvasti oman organisaation projektin aikatauluja. Vaikutuksia ja rajoituksia laajasta käyttäjäyhteisöstä voi tulla, koska järjestelmätoimittajallakin on oma asiantuntijatiiminsä, joka tällaista työtä tekee. Vapaita aikoja voi olla hankala löytää juuri sellaiseen aikaan, kun esimerkiksi

koulutusorganisaatiossa olisi optimaalisin aika sulkea palveluita operaation ajaksi. Omaan osaamiseen luottamalla voidaan saavuttaa suurempi vapaus aikataulujen suunnitteluun, esimerkiksi silloin kun asiakkaita on tilastollisesti vähiten käyttämässä järjestelmiä. Omatoiminen tietokantaprojekti voidaan nähdä myös oppimisprojektiksi. Perehtymällä rajapintojen toimintaan, osaamista voidaan hyödyntää myöhemminkin muun muassa tietokantojen datan siivouksissa.

Lahdessa toteutetussa projektissa haluttiin hyödyntää olemassa olevaa tietoteknistä osaamista. Projektin alkuvaiheessa kartoitettiin vaihtoehtoja omatoimisen ja järjestelmätoimittajan tekemän prosessin näkökulmista. Tämän perusteella päädyttiin kartoittamaan muita Voyager-kirjastojärjestelmän rajapintoja hyödyntäviä korkeakoulukirjastoja. Muun muassa Northwesternin yliopisto on hyödyntänyt Batchcat-luettelointirajapintaa omassa toiminnassaan. Myös Tukholman yliopisto on hyödyntänyt rajapintoja samanlaiseen toimintaan kuin Lahden Voyager-kirjastojen tietokantaprojektissakin on ollut tarkoitus. Näiden esimerkin pohjalta lähdettiin kehittämään sovelluksia rajapintoja hyödyntämään. Visual Basic -sovellus hyödyntää Batchcat-luettelointirajapinnan funktioita tunnistautumiseen järjestelmään sekä nidetietojen ja niteen viivakoodien kirjoittamiseksi tietokantaan. Rajapinta mahdollistaisi laajemmankin käytettävyyden. Muilta osin omaa ohjelmointia käytettiin siirtotiedostojen muodostamiseen. Datan kerääminen Microsoft Accessiin linkitetystä lähdetietokannasta ja sen kirjoittaminen VBA-sovelluksilla Voyagerin rajapintojen kuvausta vastaavaan muotoon tarjoaa turvallisen tavan kirjoittaa dataa kohteena olevaan tietokantaan. Asiakasorganisaatioilla ei virallisesti ole kirjoitusoikeuksia järjestelmän taustalla olevaan Oracle-tietokantaan. Myöskään kaikkien taulujen relaatioista ei ole olemassa kattavaa kuvausta. Rajapintoja ja järjestelmän eräajoja hyödyntämällä järjestelmätoimittaja vastaa järjestelmän toiminnallisuudesta ja asiakasorganisaatiolla on normaaliin toimintaankin kuuluva vastuu sisällöistä.

Asiakastietojen osalta projektissa hyödynnetty osa on ollut jo aikaisemminkin käytössä. Päijät-Hämeen koulutuskonsernissa on kehitetty jo kirjastojärjestelmää käyttöönotettaessa prosessi, jossa konsernin tulosyksiköiden opiskelijoiden osoitetiedot päivitetään päivittäin opintohallintojärjestelmästä. Tälläkin on pyritty henkilöresurssien kohdentamiseen

enemmän tietointensiivisempään osaamiseen. Tässä projektissa oli havaittavissa, että ainakin asiakastietojen osalta oli havaittavissa päällekkäisyyksiä. Koska kaupungin korkeakoulukirjastojen palvelut on nyt yhdistetty, asiakassegmenttien palvelutarpeiden tunnistaminen on mahdollista entistä paremmin.

Ollila (2009) on tehnyt aikaisemmin selvityksen kirjastojärjestelmien yhdistämiseksi. Selvityksen lähtökohtana oli järjestelmätoimittajan suorittama tietokantojen yhdistäminen. Vaihtoehtoina oli toisen tietokannan yhdistäminen olemassa olevaan toiseen tietokantaan tai kokonaan uuden tietokannan luominen. Jälkimmäisessä vaihtoehdossa molemmat yhdisteltävät tietokannat siirretään uuteen, tyhjään tietokantaan. Ollilan (2009) mukaan järjestelmätoimittajan operaatioista puuttuu lainatietojen sekä maksutietojen siirto. Bibliografisten tietueiden osalta jälkisiivousta tulee tuplatietueiden osalta.

Jatkoselvityksissä olisikin kehitettävä bibliografisten tietueiden tuplatarkistuksia. Yleisemmällä tasolla tietokantojen yhdistämiseen liittyviä ongelmia on tutkittu melko pitkään. Hernandez & Stolfo (1995) on kiinnittänyt artikkelissaan huomiota tunnisteiden erimuotoisuuteen. Osioimalla ja pienempien osien yksilöivän avainkentän tarkennuksilla, lajittelulla ja lopulta tietojen yhdistelyllä voidaan saavuttaa varsinkin korruptoituneen tiedon osalta tuloksia. Tulosten saavuttaminen edellyttää, että tunnetaan tietokannan sisältöä riittävästi. Esimerkiksi ISBN-tunniste voi olla 10-merkkinen tai 13-merkkinen. Lisäksi se voi olla osioituna väliviivoilla, mutta se voi olla myös ilman väliviivoja. Suomessa toimivien korkeakoulukirjastojen osalta tähän tuo ratkaisun Kansalliskirjaston ylläpitämä bibliografisten tietueiden metatietovaranto, Melinda. Tonterin (2009) yhteisluetteloista tekemän selvityksen mukaan yhteinen metatietovaranto tehostaa tietueiden ylläpitoon käytettäviä resursseja. Tietokantojen yhdistämisprojekteissa yhteisluettelon metatietovarannosta on hyötyä, koska sen avulla voidaan havaita samaa teosta kuvaavat tietueet paremmin. Tonterin (2009) tekemässä selvityksessä todetaan, että luetteloidun metatiedon laadukkuus paranee ja yhdenmukaistuu yhteisluetteloinnissa.

Jatkossa tämän tietokantaprojektin tuottamia tuloksia on mahdollista hyödyntää vastaavissa tietokantaprojekteissa korkeakoulukirjastoissa. Yksi jatkotutkimuksen tarve voi olla tässä työssä esitetyn tietokantojen yhdistämiseen tähtäävän prosessin automatisoinnin

mahdollistava määrittelytyö. Tarkemmasta prosessikuvauksesta voisi olla hyötyä sekä tietokantojen yhdistämiseen valmistautumisessa että sen jälkisiivoukseen liittyvissä toimenpiteissä.

Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen osalta saattaa tulevina vuosina tulla järjestelmiin muutoksia. Lahden ammattikorkeakoulu, Laurea ammattikorkeakoulu sekä Hämeen ammattikorkeakoulu ovat muodostaneet liittouman. Tavoitteena on palvella paremmin Helsingin laajan metropolialueen opiskelijoita, elinkeinoelämää sekä julkishallintoa (FUAS-strategia 2011 - 2015, 2013). FUAS (Federation of Universities of Applied Sciences) järjestää tällä hetkellä muun muassa yhteisiä kesäopintoja ja jossain määrin tietoja siirretään myös opintohallintojärjestelmien välillä. Koska kaikki kolme toimijaa jatkavat kuitenkin itsenäisinä toimijoina, ei voida vielä tietää mitä vaikutuksia liittoumalla on järjestelmiin. Mikäli jotain muutoksia tapahtuu, tämän projektin myötä hankittua osaamista on mahdollista hyödyntää. Toinen tulevaisuuden kirjastojärjestelmään liittyvä muutos liittyy Voyager-kirjastojärjestelmän ikääntymiseen. Kansalliskirjasto on käynnistänyt UKJ-määrittelyprosessin (uusi kirjastojärjestelmä), jossa on tavoitteena määrittää ja toteuttaa kirjastojärjestelmä palvelemaan eri kirjastosektoreita. Vuoden 2013 aikana on tavoitteena toteuttaa projektisuunnitelma, vaatimusmäärittely aikatauluineen sekä vaiheistuksineen (UKJ-hanke suunnitellaan vuonna 2013, 2013). Tämän tietokantaprojektin tuloksia on mahdollista hyödyntää varsinkin tietojen muokkauksen osalta. Koska mahdollisesti toteutettava uusi kirjastojärjestelmä on vielä määrittelyvaiheessa, on kuitenkin mahdotonta vielä tietää mahdollisten rajapintojen kuvauksia.

Tässä tietokantaprojektissa asetettiin tavoitteeksi selvittää, kuinka Voyager-kirjastojärjestelmän tietokantojen yhdistäminen voidaan toteuttaa järjestelmän rajapintoja hyödyntäen. Tähän liittyen rajapintojen tutkiminen sekä tietojen uloslukemisen mahdollistavien kyselyiden luominen kuului tämän projektin tavoitteisiin. Näiltä osin työn tavoitteiden voidaan katsoa toteutuneen. Projekti on ollut mielenkiintoinen sekä opettavainen.

5 YHTEENVETO

Tässä työssä on ollut tavoitteena selvittää, kuinka Voyager-kirjastojärjestelmän rajapintoja voidaan hyödyntää tietokantojen yhdistämisprosessissa. Tutkimuksessa on selvitetty aikaisemmin vastaavia operaatioita omatoimisesti tehneitä Voyager-kirjastojärjestelmää käyttäviä organisaatioita. Northwesternin yliopisto on hyödyntänyt tässä tietokantaprojektissa hyödynnettävää Batchcat-rajapintaa omissa tietueiden hallinnan prosesseissaan. Tukholman yliopistossa rajapintaa on hyödynnetty Lahdessa toteutettua tietokantaprojektia vastaavalla tavalla.

Ennen tiedonsiirtoja tietokantojen välillä, kohdetietokannan systeemimäärityksiin pitää luoda uutta toimipisteverkostoa vastaava rakenne. Systeemimäärityksiin pitää luoda myös kohdetietokannan säännöstöä vastaavat lainaussäännöstöt, jotta kirjastoverkoston toiminta voi alkaa mahdollisimman pian tietokantojen yhdistämisen jälkeen.

Päijät-Hämeen koulutuskonsernin Tieto- ja kirjastopalvelujen toiminnoissa on aikaisemmin hyödynnetty asiakastietojen siirtämiseen tarkoitettua palvelimen ptrnupdt-eräajoa. Automatisoitua prosessia on hyödynnetty opiskelija-asiakkaiden osoitetietojen päivityksessä opintohallintojärjestelmästä kirjastojärjestelmään. Rajapinta edellyttää määrämittaisen syötetiedoston luomista. Tässä projektissa lähdejärjestelmän tiedot on kerätty SQL-kyselyllä. VBA-sovelluksella tiedot on muokattu rajapinnan vaatimaan muotoon. Prosessissa huomioitiin molemmissa kannoissa olevien asiakkaiden käsittely päivitysmenettelyn kautta. Tällöin vain tuoreemmat osoitetiedot kirjattiin tietokantaan ja kirjastokortin viivakooditunniste lisättiin asiakkaan tietoihin. Mikäli asiakkaan tietoja ei kohdetietokannasta löytynyt, ne kirjoitettiin tietokantaan uusina asiakkaina.

Bibliografisten tietueiden sekä varastotietueiden osalta päädyttiin hyödyntämään palvelimella suoritettavia eräajoja. Tietueet voidaan kirjoittaa lähdetietokannasta siirtotiedostoon marcexport-eräajolla. Koska molemmissa järjestelmissä bibliografiset tietueet ovat marc21-formaatin mukaisia, tietueiden voisi siirtää ilman käsittelyä kohdejärjestelmään. Tässä tietokantaprojektissa haluttiin lisätä projektin aikaiset tunnisteet siirrettäviin tietueisiin sekä muuttaa varastotietueiden toimipisteistä kertovat tunnisteet

vastaamaan kohdejärjestelmän tunnisteita. Lisättyjen tunnisteiden ja kirjastojärjestelmän käsittelemien järjestelmätunnisteiden avulla luotiin taulukko, jonka avulla saatiin kirjoitettua lähdetietokannan nidetiedot oikeisiin varastotietueisiin lähdetietokannassa. Nidetietojen kirjoittamiseen kirjoitettiin Visual Basic -sovellus, joka hyödyntää Voyager-järjestelmän Batchcat-luettelointirajapintaa.

Valmistelevissa selvityksissä havaittiin, että tietokantojen yhdistämistä edeltävät valmistelevat työvaiheet sekä yhdistämisen jälkeiset toiminnot ovat Voyager-kirjastojärjestelmän asiakasorganisaatiolle lähes samat riippumatta siitä, tekeekö varsinaisen tietokantojen yhdistämisen järjestelmätoimittaja vai tehdäänkö se omatoimisesti rajapintoja hyödyntäen. Oman osaamisen hyödyntämisellä saavutetaan taloudellista hyötyä, mutta myös projektin aikataulujen laadintaan enemmän vapauksia vastaamaan organisaation vuosisuunnitelmaa. Rajapintoja hyödyntämällä säilytetään normaalin kirjastotoiminnan vastuun määritykset. Kirjastojärjestelmän toimittaja vastaa järjestelmän toiminnasta ja siihen lisätyistä toiminnallisuuksista. Kirjastojärjestelmän asiakasorganisaatio vastaa tuotetun tiedon sisällöstä, oikeellisuudesta ja eheydestä.

LÄHTEET

1. Amkit-konsortio, 2013, tietokannat, [viitattu 04.04.2013], saatavissa: http://www.amkit.fi/index.php?tietokannat
2. Bryla, B. 2004, Oracle Database Foundations, Sybex, ISBN: 9780782151237
3. Ekman-Sarkki, M. Mikroyrityksen www-sivut, 1.11.2006, TIEKE Tietoyhteiskunnan kehittämiskeskus, [viitattu 27.11.2011], saatavissa: http://www.tieke.fi/mp/db/file_library/x/IMG/19936/file/Mikroyrityksen_www-sivut.pdf
4. Evjen, B. & Hollis, B. & Sheldon, B. & Sharkey, K. 2008, Professional Visual Basic 2008, Wrox, ISBN: 9780470378670
5. Finlex, Henkilötietolaki, 1999, [viitattu 06.04.2013], saatavissa: http://www.finlex.fi/fi/laki/ajantasa/1999/19990523
6. FUAS-strategia 2011 - 2015, 2013, FUAS, [viitattu 06.04.2013], saatavissa: http://www.fuas.fi/fuas/Materiaalipankki/fuas_strategia_2011_2015.pdf
7. Föhr, P. 2011, DIY Voyager-tietokantojen yhdistäminen, Lahden Ammattikorkeakoulu, [viitattu 18.03.2013], saatavissa: https://wiki.hamk.fi/download/attachments/18290424/DIY++-kantayhdistely.pdf?version=1&modificationDate=1298989948000
8. Hernandez, M. & Stolfo, S. 1995, The merge/purge problem for large databases, SIGMOD '95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data, ACM, New York, pp. 127-138
9. Korkeakoulukirjastojen rakenteellisen kehittämisen hanke -verkkosivut. [viitattu 08.04.2010]. saatavissa: http://www.kansalliskirjasto.fi/kirjastoala/rake.html
10. Kranti, U. & Sandhu, R. & Jeet Chakrotti, A. 2002, .NET Framework Professional Projects, Course Technology/Cengage Learning, ISBN: 9781931841245
11. Lahden ammattikorkeakoulun tieto- ja kirjastopalvelut -verkkosivut. [viitattu 08.04.2010]. saatavissa: http://www.phkk.fi/tietokeskus/esittely/
12. Lahden yliopistokeskus, [viitattu 16.04.2013], saatavissa: http://www.lahdenyliopistokeskus.fi/fi/esittely/korkeakoulujen_yhteiskirjasto
13. Lee, Y. W. & Pipino, L. & Funk, J. & Wang, R. Y. 2006, Journey to Data Quality, MIT Press, ISBN:9780262122870

14. Marc21-formaatti: Bibliografiset tiedot, Kansalliskirjasto, [viitattu 03.04.2013], saatavissa: http://www.kansalliskirjasto.fi/extra/marc21/bib/index.htm
15. Motro, A. & Rakov, I. 1998, Estimating the quality of databases, Flexible Query Answering Systems, Lecture Notes in Computer Science, 1998, Vol. 1495, pp. 298-307, Springer Berlin / Heidelberg, ISBN: 9783540650829
16. Ollila, M. 2009, Selvitys kirjastojärjestelmien yhdistämisestä, Rovaniemen ammattikorkeakoulu.
17. Randolph, N. & Gardner, D. & Anderson, C. 2010, Professional Visual Studio 2010, Wrox, ISBN:9 780470873120
18. Redman, T. C. The impact of poor data quality on the typical enterprice, Communications of the ACM, February 1988, vol 41, No. 2.
19. Reese, T. MarcEdit - About MarcEdit, Origon State University, [viitattu 05.04.2013], saatavissa http://people.oregonstate.edu/~reeset/marcedit/html/about.html
20. UKJ-hanke suunnitellaan vuonna 2013, Kansalliskirjasto, [viitattu 06.04.2013], saatavissa: http://www.kansalliskirjasto.fi/kirjastoala/uutiskirje/1_2013/ukj.html
21. Uuden ajan oppimiskeskus - Oppimiskeskustyöryhmän loppuraportti. 08.05.2009. [viitattu 08.04.2010], saatavissa: http://www.phkk.fi/material/oppimiskeskuslr.pdf
22. Vine, M. 2005, Microsoft Access VBA Programming for the Absolute Beginner, Second Edition, Course Technology/Cengage Learning, ISBN: 9781592007233
23. Von Hippel, E. 2001, User Toolkits for Innovation, MIT Sloan School of Management, Journal of Product Innovation Management, July, 2001
24. Voyager 7.0 Catalging User’s Guide, 2008, Ex Libris
25. Voyager 7.0 Circulation User’s Guide, 2008, Ex Libris
26. Voyager 7.0 System Administration Guide, 2008, Ex Libris
27. Voyager 7.2 Batchcat.dll Technical User's Guide, 2010, Ex Libris
28. Voyager Prochure, 2013, Ex Libris, [viitattu 04.04.2013], saatavissa http://www.exlibrisgroup.com/files/Products/Aleph,Voyager/Voyager/Voyager_brochure_A4_low.pdf

LIITE 1. Niteiden latauslistan muodostus

```
Sub lakkilistat()
Set rs = CurrentDb.OpenRecordset("nidelatauslista")
Open "nidelista_20101220.txt" For Output As #1
'tiedoston rakenne: luettelointipaikka, holdingid, nidetyyppi, permlocation, barcode,freetext
'taulun rakenne bib_id, mfhd_id, item_type, item_barcode, barcode_status, perm_location
Do While Not rs.EOF
catloc = "95"
mfhd = rs.Fields("mfhd").Value
'Muutetaan niteen tyyppi vastaamaan kohdekannan item_typeä
Select Case rs.Fields("item_type_id").Value
Case "1"
itemtype = "3"
Case "3"
itemtype = "3"
Case "2"
itemtype = "2"
End Select
'Muutetaan vanhan kannan location koodi vastaamaan uuden kannan sijaintikoodia.
'Kirjoitetaan niteen freetext-kenttään nimekkeen seurantatunniste.
Select Case rs.Fields("perm_location").Value
Case "16"
permloc = "126"
freetext = "2011 VER"
Case "12"
permloc = "136"
freetext = "2011 LEH"
Case "2"
permloc = "127"
```

(jatkuu)

LIITE 1. (jatkoa)

```
freetext = "2011 TKK"
Case "7"
permloc = "127"
freetext = "2011 TKKU"
Case "19"
permloc = "129"
freetext = "2011 TYK"
Case "18"
permloc = "130"
freetext = "2011 TYL"
Case "14"
permloc = "131"
freetext = "2011 YEK"
Case "22"
permloc = "129"
freetext = "2011 TKK - K"
Case "21"
permloc = "130"
freetext = "2011 TKK - L"
Case "23"
permloc = "137"
freetext = "2011 YLEH"
Case "3"
permloc = "128"
freetext = "2011 TKL"
End Select
vkoodi = rs.Fields("item_barcode").Value
Print #1, catloc & ";" & mfhd & ";" & itemtype & ";" & permloc & ";" & vkoodi & ";" &
freetext
```

(jatkuu)

LIITE 1. (jatkoa)

```
rs.MoveNext
Loop
Close #1
rs.Close
End Sub
```

LIITE 2. Ohjelmakoodi niteiden kirjoittamiseksi tietokantaan

```
Public Class Form1
Inherits System.Windows.Forms.Form
Public MyBatchCat As New BatchCat.ClassBatchCat
Public StatusLine
#Region " Windows Form Designer generated code "
Public Sub New()
MyBase.New()
'This call is required by the Windows Form Designer.
InitializeComponent()
'Add any initialization after the InitializeComponent() call
End Sub
'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub
'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents grpLoginInfo As System.Windows.Forms.GroupBox
Friend WithEvents txtVoyHak As System.Windows.Forms.TextBox
Friend WithEvents txtUID As System.Windows.Forms.TextBox
```

(jatkuu)

LIITE 2. (jatkoa)

```
Friend WithEvents txtPW As System.Windows.Forms.TextBox
Friend WithEvents lblVoyHak As System.Windows.Forms.Label
Friend WithEvents lblUID As System.Windows.Forms.Label
Friend WithEvents lblPW As System.Windows.Forms.Label
Friend WithEvents btn_login As System.Windows.Forms.Button
Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
Friend WithEvents txtCatLocID As System.Windows.Forms.TextBox
Friend WithEvents lblCatLocID As System.Windows.Forms.Label
Friend WithEvents lblHoldingID As System.Windows.Forms.Label
Friend WithEvents txtHoldingID As System.Windows.Forms.TextBox
Friend WithEvents lblItemID As System.Windows.Forms.Label
Friend WithEvents txtItemID As System.Windows.Forms.TextBox
Friend WithEvents lblItemTypeID As System.Windows.Forms.Label
Friend WithEvents txtItemTypeID As System.Windows.Forms.TextBox
Friend WithEvents lblPermLocID As System.Windows.Forms.Label
Friend WithEvents txtPermLocID As System.Windows.Forms.TextBox
Friend WithEvents lblBarcode As System.Windows.Forms.Label
Friend WithEvents txtBarcode As System.Windows.Forms.TextBox
Friend WithEvents btnKantaan As System.Windows.Forms.Button
Friend WithEvents grpHoldings As System.Windows.Forms.GroupBox
Friend WithEvents lblmfhdMarcData As System.Windows.Forms.Label
Friend WithEvents txtMfhdMarcData As System.Windows.Forms.TextBox
Friend WithEvents txtLinkitettyBibID As System.Windows.Forms.TextBox
Friend WithEvents lbllinkitettyBibId As System.Windows.Forms.Label
Friend WithEvents txtmfhdCatLocID As System.Windows.Forms.TextBox
Friend WithEvents lblmfhdCatLocID As System.Windows.Forms.Label
Friend WithEvents txtOpacSuppress As System.Windows.Forms.TextBox
Friend WithEvents lblOpacSuppress As System.Windows.Forms.Label
Friend WithEvents txtHoldLocID As System.Windows.Forms.TextBox
```

(jatkuu)

LIITE 2. (jatkoa)

```
Friend WithEvents lblHoldLocID As System.Windows.Forms.Label
Friend WithEvents btnHoldKantaan As System.Windows.Forms.Button
Friend WithEvents chkKaytaSyotetta As System.Windows.Forms.CheckBox
Friend WithEvents txtNidelista As System.Windows.Forms.TextBox
Friend WithEvents lblAbout As System.Windows.Forms.Label
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
Me.grpLoginInfo = New System.Windows.Forms.GroupBox
Me.btn_login = New System.Windows.Forms.Button
Me.lblPW = New System.Windows.Forms.Label
Me.lblUID = New System.Windows.Forms.Label
Me.lblVoyHak = New System.Windows.Forms.Label
Me.txtPW = New System.Windows.Forms.TextBox
Me.txtUID = New System.Windows.Forms.TextBox
Me.txtVoyHak = New System.Windows.Forms.TextBox
Me.GroupBox1 = New System.Windows.Forms.GroupBox
Me.txtNidelista = New System.Windows.Forms.TextBox
Me.chkKaytaSyotetta = New System.Windows.Forms.CheckBox
Me.btnKantaan = New System.Windows.Forms.Button
Me.lblBarcode = New System.Windows.Forms.Label
Me.txtBarcode = New System.Windows.Forms.TextBox
Me.lblPermLocID = New System.Windows.Forms.Label
Me.txtPermLocID = New System.Windows.Forms.TextBox
Me.lblItemTypeID = New System.Windows.Forms.Label
Me.txtItemTypeID = New System.Windows.Forms.TextBox
Me.lblItemID = New System.Windows.Forms.Label
Me.txtItemID = New System.Windows.Forms.TextBox
Me.lblHoldingID = New System.Windows.Forms.Label
Me.txtHoldingID = New System.Windows.Forms.TextBox
Me.lblCatLocID = New System.Windows.Forms.Label
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.txtCatLocID = New System.Windows.Forms.TextBox
Me.grpHoldings = New System.Windows.Forms.GroupBox
Me.btnHoldKantaan = New System.Windows.Forms.Button
Me.txtHoldLocID = New System.Windows.Forms.TextBox
Me.lblHoldLocID = New System.Windows.Forms.Label
Me.txtOpacSuppress = New System.Windows.Forms.TextBox
Me.lblOpacSuppress = New System.Windows.Forms.Label
Me.txtmfhdCatLocID = New System.Windows.Forms.TextBox
Me.lblmfhdCatLocID = New System.Windows.Forms.Label
Me.txtLinkitettyBibID = New System.Windows.Forms.TextBox
Me.lbllinkitettyBibId = New System.Windows.Forms.Label
Me.txtMfhdMarcData = New System.Windows.Forms.TextBox
Me.lblmfhdMarcData = New System.Windows.Forms.Label
Me.lblAbout = New System.Windows.Forms.Label
Me.grpLoginInfo.SuspendLayout()
Me.GroupBox1.SuspendLayout()
Me.grpHoldings.SuspendLayout()
Me.SuspendLayout()
'
'grpLoginInfo
'
Me.grpLoginInfo.Controls.Add(Me.btn_login)
Me.grpLoginInfo.Controls.Add(Me.lblPW)
Me.grpLoginInfo.Controls.Add(Me.lblUID)
Me.grpLoginInfo.Controls.Add(Me.lblVoyHak)
Me.grpLoginInfo.Controls.Add(Me.txtPW)
Me.grpLoginInfo.Controls.Add(Me.txtUID)
Me.grpLoginInfo.Controls.Add(Me.txtVoyHak)
Me.grpLoginInfo.Location = New System.Drawing.Point(360, 16)
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.grpLoginInfo.Name = "grpLoginInfo"
Me.grpLoginInfo.Size = New System.Drawing.Size(240, 120)
Me.grpLoginInfo.TabIndex = 0
Me.grpLoginInfo.TabStop = False
Me.grpLoginInfo.Text = "Login tiedot"
'
'btn_login
'
Me.btn_login.Location = New System.Drawing.Point(112, 88)
Me.btn_login.Name = "btn_login"
Me.btn_login.Size = New System.Drawing.Size(112, 24)
Me.btn_login.TabIndex = 6
Me.btn_login.Text = "Login"
'
'lblPW
'
Me.lblPW.Location = New System.Drawing.Point(8, 64)
Me.lblPW.Name = "lblPW"
Me.lblPW.Size = New System.Drawing.Size(88, 24)
Me.lblPW.TabIndex = 5
Me.lblPW.Text = "PW"
'
'lblUID
'
Me.lblUID.Location = New System.Drawing.Point(8, 36)
Me.lblUID.Name = "lblUID"
Me.lblUID.Size = New System.Drawing.Size(88, 24)
Me.lblUID.TabIndex = 4
Me.lblUID.Text = "UID"
```

(jatkuu)

LIITE 2. (jatkoa)

```
'lblVoyHak
'
Me.lblVoyHak.Location = New System.Drawing.Point(8, 16)
Me.lblVoyHak.Name = "lblVoyHak"
Me.lblVoyHak.Size = New System.Drawing.Size(88, 24)
Me.lblVoyHak.TabIndex = 3
Me.lblVoyHak.Text = "Voy-hakemisto"
'
'txtPW
'
Me.txtPW.Location = New System.Drawing.Point(104, 64)
Me.txtPW.Name = "txtPW"
Me.txtPW.PasswordChar = Microsoft.VisualBasic.ChrW(42)
Me.txtPW.Size = New System.Drawing.Size(128, 20)
Me.txtPW.TabIndex = 2
Me.txtPW.Text = ""
'
'txtUID
'
Me.txtUID.Location = New System.Drawing.Point(104, 40)
Me.txtUID.Name = "txtUID"
Me.txtUID.Size = New System.Drawing.Size(128, 20)
Me.txtUID.TabIndex = 1
Me.txtUID.Text = ""
'
'txtVoyHak
'
Me.txtVoyHak.Location = New System.Drawing.Point(104, 16)
Me.txtVoyHak.Name = "txtVoyHak"
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.txtVoyHak.Size = New System.Drawing.Size(128, 20)
Me.txtVoyHak.TabIndex = 0
Me.txtVoyHak.Text = "c:\Voyager_HARJOITUS_62"
'
'GroupBox1
'
Me.GroupBox1.Controls.Add(Me.txtNidelista)
Me.GroupBox1.Controls.Add(Me.chkKaytaSyotetta)
Me.GroupBox1.Controls.Add(Me.btnKantaan)
Me.GroupBox1.Controls.Add(Me.lblBarcode)
Me.GroupBox1.Controls.Add(Me.txtBarcode)
Me.GroupBox1.Controls.Add(Me.lblPermLocID)
Me.GroupBox1.Controls.Add(Me.txtPermLocID)
Me.GroupBox1.Controls.Add(Me.lblItemTypeID)
Me.GroupBox1.Controls.Add(Me.txtItemTypeID)
Me.GroupBox1.Controls.Add(Me.lblItemID)
Me.GroupBox1.Controls.Add(Me.txtItemID)
Me.GroupBox1.Controls.Add(Me.lblHoldingID)
Me.GroupBox1.Controls.Add(Me.txtHoldingID)
Me.GroupBox1.Controls.Add(Me.lblCatLocID)
Me.GroupBox1.Controls.Add(Me.txtCatLocID)
Me.GroupBox1.Location = New System.Drawing.Point(8, 8)
Me.GroupBox1.Name = "GroupBox1"
Me.GroupBox1.Size = New System.Drawing.Size(344, 272)
Me.GroupBox1.TabIndex = 1
Me.GroupBox1.TabStop = False
Me.GroupBox1.Text = "Niteen tiedot"
'
'txtNidelista
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.txtNidelista.Enabled = False
Me.txtNidelista.Location = New System.Drawing.Point(128, 24)
Me.txtNidelista.Name = "txtNidelista"
Me.txtNidelista.Size = New System.Drawing.Size(200, 20)
Me.txtNidelista.TabIndex = 24
Me.txtNidelista.Text = "c:\Voyager_HARJOITUS_62\niteet.txt"
'
'chkKaytaSyotetta
'
Me.chkKaytaSyotetta.Location = New System.Drawing.Point(16, 24)
Me.chkKaytaSyotetta.Name = "chkKaytaSyotetta"
Me.chkKaytaSyotetta.Size = New System.Drawing.Size(96, 24)
Me.chkKaytaSyotetta.TabIndex = 23
Me.chkKaytaSyotetta.Text = "Hae niteet tiedostosta"
'
'btnKantaan
'
Me.btnKantaan.Location = New System.Drawing.Point(248, 152)
Me.btnKantaan.Name = "btnKantaan"
Me.btnKantaan.Size = New System.Drawing.Size(88, 24)
Me.btnKantaan.TabIndex = 22
Me.btnKantaan.Text = "Nide kantaan"
'
'lblBarcode
'
Me.lblBarcode.Location = New System.Drawing.Point(16, 208)
Me.lblBarcode.Name = "lblBarcode"
Me.lblBarcode.Size = New System.Drawing.Size(120, 16)
Me.lblBarcode.TabIndex = 11
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.lblBarcode.Text = "Barcode"
'
'txtBarcode
'
Me.txtBarcode.Location = New System.Drawing.Point(144, 200)
Me.txtBarcode.Name = "txtBarcode"
Me.txtBarcode.Size = New System.Drawing.Size(192, 20)
Me.txtBarcode.TabIndex = 10
Me.txtBarcode.Text = "TESTIPF240320090815"
'
'lblPermLocID
'
Me.lblPermLocID.Location = New System.Drawing.Point(16, 176)
Me.lblPermLocID.Name = "lblPermLocID"
Me.lblPermLocID.Size = New System.Drawing.Size(120, 16)
Me.lblPermLocID.TabIndex = 9
Me.lblPermLocID.Text = "PermLocID"
'
'txtPermLocID
'
Me.txtPermLocID.Location = New System.Drawing.Point(144, 168)
Me.txtPermLocID.Name = "txtPermLocID"
Me.txtPermLocID.Size = New System.Drawing.Size(96, 20)
Me.txtPermLocID.TabIndex = 8
Me.txtPermLocID.Text = "2"
'
'lblItemTypeID
'
Me.lblItemTypeID.Location = New System.Drawing.Point(16, 152)
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.lblItemTypeID.Name = "lblItemTypeID"
Me.lblItemTypeID.Size = New System.Drawing.Size(120, 16)
Me.lblItemTypeID.TabIndex = 7
Me.lblItemTypeID.Text = "ItemTypeID"
'
'txtItemTypeID
'
Me.txtItemTypeID.Location = New System.Drawing.Point(144, 144)
Me.txtItemTypeID.Name = "txtItemTypeID"
Me.txtItemTypeID.Size = New System.Drawing.Size(96, 20)
Me.txtItemTypeID.TabIndex = 6
Me.txtItemTypeID.Text = "7"
'
'lblItemID
'
Me.lblItemID.Location = New System.Drawing.Point(16, 128)
Me.lblItemID.Name = "lblItemID"
Me.lblItemID.Size = New System.Drawing.Size(120, 16)
Me.lblItemID.TabIndex = 5
Me.lblItemID.Text = "ItemID"
'
'txtItemID
'
Me.txtItemID.Location = New System.Drawing.Point(144, 120)
Me.txtItemID.Name = "txtItemID"
Me.txtItemID.Size = New System.Drawing.Size(96, 20)
Me.txtItemID.TabIndex = 4
Me.txtItemID.Text = "145754"
'
```

(jatkuu)

LIITE 2. (jatkoa)

```
'lblHoldingID
'
Me.lblHoldingID.Location = New System.Drawing.Point(16, 96)
Me.lblHoldingID.Name = "lblHoldingID"
Me.lblHoldingID.Size = New System.Drawing.Size(120, 16)
Me.lblHoldingID.TabIndex = 3
Me.lblHoldingID.Text = "HoldingID"
'
'txtHoldingID
'
Me.txtHoldingID.Location = New System.Drawing.Point(144, 88)
Me.txtHoldingID.Name = "txtHoldingID"
Me.txtHoldingID.Size = New System.Drawing.Size(96, 20)
Me.txtHoldingID.TabIndex = 2
Me.txtHoldingID.Text = "1112"
'
'lblCatLocID
'
Me.lblCatLocID.Location = New System.Drawing.Point(16, 64)
Me.lblCatLocID.Name = "lblCatLocID"
Me.lblCatLocID.Size = New System.Drawing.Size(120, 16)
Me.lblCatLocID.TabIndex = 1
Me.lblCatLocID.Text = "CatLocID"
'
'txtCatLocID
'
Me.txtCatLocID.Location = New System.Drawing.Point(144, 56)
Me.txtCatLocID.Name = "txtCatLocID"
Me.txtCatLocID.Size = New System.Drawing.Size(96, 20)
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.txtCatLocID.TabIndex = 0
Me.txtCatLocID.Text = "95"
'
'grpHoldings
'
Me.grpHoldings.Controls.Add(Me.btnHoldKantaan)
Me.grpHoldings.Controls.Add(Me.txtHoldLocID)
Me.grpHoldings.Controls.Add(Me.lblHoldLocID)
Me.grpHoldings.Controls.Add(Me.txtOpacSuppress)
Me.grpHoldings.Controls.Add(Me.lblOpacSuppress)
Me.grpHoldings.Controls.Add(Me.txtmfhdCatLocID)
Me.grpHoldings.Controls.Add(Me.lblmfhdCatLocID)
Me.grpHoldings.Controls.Add(Me.txtLinkitettyBibID)
Me.grpHoldings.Controls.Add(Me.lbllinkitettyBibId)
Me.grpHoldings.Controls.Add(Me.txtMfhdMarcData)
Me.grpHoldings.Controls.Add(Me.lblmfhdMarcData)
Me.grpHoldings.Location = New System.Drawing.Point(8, 328)
Me.grpHoldings.Name = "grpHoldings"
Me.grpHoldings.Size = New System.Drawing.Size(344, 176)
Me.grpHoldings.TabIndex = 2
Me.grpHoldings.TabStop = False
Me.grpHoldings.Text = "Holdings"
'
'btnHoldKantaan
'
Me.btnHoldKantaan.Location = New System.Drawing.Point(168, 144)
Me.btnHoldKantaan.Name = "btnHoldKantaan"
Me.btnHoldKantaan.Size = New System.Drawing.Size(112, 23)
Me.btnHoldKantaan.TabIndex = 10
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.btnHoldKantaan.Text = "Mfhd Kantaan"
'
'txtHoldLocID
'
Me.txtHoldLocID.Location = New System.Drawing.Point(108, 112)
Me.txtHoldLocID.Name = "txtHoldLocID"
Me.txtHoldLocID.Size = New System.Drawing.Size(224, 20)
Me.txtHoldLocID.TabIndex = 9
Me.txtHoldLocID.Text = "2"
'
'lblHoldLocID
'
Me.lblHoldLocID.Location = New System.Drawing.Point(12, 120)
Me.lblHoldLocID.Name = "lblHoldLocID"
Me.lblHoldLocID.Size = New System.Drawing.Size(80, 24)
Me.lblHoldLocID.TabIndex = 8
Me.lblHoldLocID.Text = "HoldLocationID"
'
'txtOpacSuppress
'
Me.txtOpacSuppress.Location = New System.Drawing.Point(108, 88)
Me.txtOpacSuppress.Name = "txtOpacSuppress"
Me.txtOpacSuppress.Size = New System.Drawing.Size(224, 20)
Me.txtOpacSuppress.TabIndex = 7
Me.txtOpacSuppress.Text = "False"
'
'lblOpacSuppress
'
Me.lblOpacSuppress.Location = New System.Drawing.Point(12, 96)
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.lblOpacSuppress.Name = "lblOpacSuppress"
Me.lblOpacSuppress.Size = New System.Drawing.Size(80, 24)
Me.lblOpacSuppress.TabIndex = 6
Me.lblOpacSuppress.Text = "OpacSuppress"
'
'txtmfhdCatLocID
'
Me.txtmfhdCatLocID.Location = New System.Drawing.Point(108, 64)
Me.txtmfhdCatLocID.Name = "txtmfhdCatLocID"
Me.txtmfhdCatLocID.Size = New System.Drawing.Size(224, 20)
Me.txtmfhdCatLocID.TabIndex = 5
Me.txtmfhdCatLocID.Text = "95"
'
'lblmfhdCatLocID
'
Me.lblmfhdCatLocID.Location = New System.Drawing.Point(12, 72)
Me.lblmfhdCatLocID.Name = "lblmfhdCatLocID"
Me.lblmfhdCatLocID.Size = New System.Drawing.Size(80, 24)
Me.lblmfhdCatLocID.TabIndex = 4
Me.lblmfhdCatLocID.Text = "CatLocID"
'
'txtLinkitettyBibID
'
Me.txtLinkitettyBibID.Location = New System.Drawing.Point(104, 40)
Me.txtLinkitettyBibID.Name = "txtLinkitettyBibID"
Me.txtLinkitettyBibID.Size = New System.Drawing.Size(224, 20)
Me.txtLinkitettyBibID.TabIndex = 3
Me.txtLinkitettyBibID.Text = "953"
'
```

(jatkuu)

LIITE 2. (jatkoa)

```
'lbllinkitettyBibId
'
Me.lbllinkitettyBibId.Location = New System.Drawing.Point(8, 48)
Me.lbllinkitettyBibId.Name = "lbllinkitettyBibId"
Me.lbllinkitettyBibId.Size = New System.Drawing.Size(80, 24)
Me.lbllinkitettyBibId.TabIndex = 2
Me.lbllinkitettyBibId.Text = "Linkitetty bibID"
'
'txtMfhdMarcData
'
Me.txtMfhdMarcData.Location = New System.Drawing.Point(104, 16)
Me.txtMfhdMarcData.Name = "txtMfhdMarcData"
Me.txtMfhdMarcData.Size = New System.Drawing.Size(224, 20)
Me.txtMfhdMarcData.TabIndex = 1
Me.txtMfhdMarcData.Text = "marc dataa..."
'
'lblmfhdMarcData
'
Me.lblmfhdMarcData.Location = New System.Drawing.Point(8, 24)
Me.lblmfhdMarcData.Name = "lblmfhdMarcData"
Me.lblmfhdMarcData.Size = New System.Drawing.Size(80, 24)
Me.lblmfhdMarcData.TabIndex = 0
Me.lblmfhdMarcData.Text = "mfhdMarc"
'
'lblAbout
'
Me.lblAbout.Location = New System.Drawing.Point(360, 176)
Me.lblAbout.Name = "lblAbout"
Me.lblAbout.Size = New System.Drawing.Size(240, 320)
```

(jatkuu)

LIITE 2. (jatkoa)

```
Me.lblAbout.TabIndex = 3
'
'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(608, 509)
Me.Controls.Add(Me.lblAbout)
Me.Controls.Add(Me.grpHoldings)
Me.Controls.Add(Me.GroupBox1)
Me.Controls.Add(Me.grpLoginInfo)
Me.Name = "Form1"
Me.Text = "Tiedonsiirtotestejä"
Me.grpLoginInfo.ResumeLayout(False)
Me.GroupBox1.ResumeLayout(False)
Me.grpHoldings.ResumeLayout(False)
Me.ResumeLayout(False)
End Sub
#End Region
Private Sub btn_login_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_login.Click
Dim AppPath = txtVoyHak.Text
Dim UserName = txtUID.Text
Dim Password = txtPW.Text
Dim login = MyBatchCat.Connect(AppPath, UserName, Password)
If login <> iBatchCatSuccess Then
MsgBox("Kirjautuminen ei onnistunut")
Exit Sub
Else
MsgBox("Kirjautuminen toimii")
```

(jatkuu)

LIITE 2. (jatkoa)

```
btn_login.Enabled = False
btn_login.Text = "Yhteydessä..."
End If
'mybatchcat
End Sub
Private Sub btnKantaan_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnKantaan.Click
If chkKaytaSyotetta.Checked = True Then
'95;1112;7;2;TestiBC0001
'95;1112;7;2;TestiBC0002
'95;1112;7;2;TestiBC0003
FileOpen(1, txtNidelista.Text, OpenMode.Input)
Do Until EOF(1)
Dim Text As String = LineInput(1)
Dim Nidearray() As String = Split(Text, ";")
MyBatchCat.cItem.HoldingID = Nidearray(1)
MyBatchCat.cItem.ItemTypeID = Nidearray(2)
MyBatchCat.cItem.PermLocationID = Nidearray(3)
vie = MyBatchCat.AddItemData(Nidearray(0))
If vie = 0 Then
MsgBox("niteen vienti onnistui")
Else
MsgBox("niteen vienti epäonnistui")
End If
lisatty_nide = MyBatchCat.RecordIDAdded
vie = MyBatchCat.AddItemBarCode(lisatty_nide, Nidearray(4))
If vie = 0 Then
MsgBox("viivakoodin vienti onnistui")
Else
```

(jatkuu)

LIITE 2. (jatkoa)

```
MsgBox("viivakoodin vienti epäonnistui")
End If
Loop
FileClose(1)
Else
MyBatchCat.cItem.HoldingID = txtHoldingID.Text
'MyBatchCat.cItem.ItemID = txtItemID.Text
MyBatchCat.cItem.ItemTypeID = txtItemTypeID.Text
MyBatchCat.cItem.PermLocationID = txtPermLocID.Text
vie = MyBatchCat.AddItemData(txtCatLocID.Text)
If vie = 0 Then
MsgBox("niteen vienti onnistui")
Else
MsgBox("niteen vienti epäonnistui")
End If
lisatty_nide = MyBatchCat.RecordIDAdded
vie = MyBatchCat.AddItemBarCode(lisatty_nide, txtBarcode.Text)
If vie = 0 Then
MsgBox("viivakoodin vienti onnistui")
Else
MsgBox("viivakoodin vienti epäonnistui")
End If
End If
End Sub
Private Sub btnHoldKantaan_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnHoldKantaan.Click
vie = MyBatchCat.AddHoldingRecord(txtMfhdMarcData.Text,
txtLinkitettyBibID.Text, txtmfhdCatLocID.Text, txtOpacSuppress.Text,
txtHoldLocID.Text)
```

(jatkuu)

LIITE 2. (jatkoa)

```
If vie = 0 Then
lisatty_mfhd = MyBatchCat.RecordIDAdded
MsgBox("0 - Holdings " & lisatty_mfhd & "luotu")
ElseIf vie = 1 Then
MsgBox("1 - Tuntematon virhe")
ElseIf vie = 2 Then
MsgBox("2 - Not a valid record")
ElseIf vie = 3 Then
MsgBox("3 - Invalid record type")
ElseIf vie = 5 Then
MsgBox("5 - could no insert 001")
ElseIf vie = 10 Then
MsgBox("10 - could not add record")
ElseIf vie = 15 Then
MsgBox("15 - invalid leader length")
ElseIf vie = 16 Then
MsgBox("16 - no leader field")
ElseIf vie = 20 Then
MsgBox("20 - could not add related id")
ElseIf vie = 21 Then
MsgBox("21 - record edit error")
ElseIf vie = 23 Then
MsgBox("23 - invalid location code")
ElseIf vie = 24 Then
MsgBox("24 - invalid cataloging location ID")
ElseIf vie = 31 Then
MsgBox("31 - could not assemble record")
ElseIf vie = 32 Then
MsgBox("32 - could not create new ID")
```

(jatkuu)

LIITE 2. (jatkoa)

```
ElseIf vie = 33 Then
MsgBox("33 - invalid 008 length")
ElseIf vie = 34 Then
MsgBox("34 - record has no 008 data")
ElseIf vie = 34 Then
MsgBox("35 - linked ID error")
ElseIf vie = 36 Then
MsgBox("36 - record structure is invalid")
ElseIf vie = 37 Then
MsgBox("37 - record has no 852 field")
ElseIf vie = 38 Then
MsgBox("38 - 852 field field has no subfield")
ElseIf vie = 39 Then
MsgBox("39 - 852 subfield has no data")
ElseIf vie = 203 Then
MsgBox("203 - read only - server error")
End If
End Sub
Private Sub chkKaytaSyotetta_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles chkKaytaSyotetta.CheckedChanged
If chkKaytaSyotetta.Checked = True Then
txtNidelista.Enabled = True
txtHoldingID.Enabled = False


txtItemID.Enabled = False
txtItemTypeID.Enabled = False
txtPermLocID.Enabled = False
txtCatLocID.Enabled = False
```

(jatkuu)

LIITE 2. (jatkoa)

```
txtBarcode.Enabled = False
Else
txtNidelista.Enabled = False
txtHoldingID.Enabled = True
txtItemID.Enabled = True
txtItemTypeID.Enabled = True
txtPermLocID.Enabled = True
txtCatLocID.Enabled = True
txtBarcode.Enabled = True
End If
End Sub
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
about = "Cop Y. Right on Nosmo Kingin kaveri"
lblAbout.Text = about
End Sub
End Class
```

LIITE 3. VBA-sovellus asiakastietojen kirjoittamiseksi siirtotiedostoon

```
Function kirjoitauudet()

Set rs = CurrentDb.OpenRecordset("lisattavat_patronit27012011_b")
Open "lakki_uudet_asiakkaat27012011.sif" For Output As #1
Do While Not rs.EOF
   'ptrbarcodeID1 = rs.Fields("hlotun")
      'String(xx, " ") -> kirjoitetaan tiedostoon tarvittava määrä "tyhjää", _
      tarpeettomia voi yhdistellä
      ptrID = String(10, " ")
      ptrbarcodeID1 = String(10, " ")
      'rs.Fields("patron_barcode") & String((25 - Len(rs.Fields("snimi"))), " ")
      ptrbarcode1 = rs.Fields("patron_barcode") & String((25 -
Len(rs.Fields("patron_barcode"))), " ")
      ptrgrp1 = "AS" & String(8, " ") ' asiakasryhmä
      barcodestat1 = "1" ' viivakoodin statukseksi "active"
      barcodemoddate1 = String(10, " ")
      barcode2string = String(56, " ")
      barcode3string = String(56, " ")
      regstdate = String(10, " ")
      'ptrexprdate = "2004.12.31"
      'expire date, winhasta siirretyt asetetaan vanhentuneeksi
      ptrexprdate = "2012.01.31"

      ' purge date, winhasta oletettu valmistumisajankohta
ptrpurgedate = "2012.01.31"
      voydate = String(10, " ")
      voyupd = String(10, " ")
      lbrloccode = String(10, " ")
      ' institution ID -kenttään oppilaitoskoodi + opiskelijanro
```

(jatkuu)

LIITE 3. (jatkoa)

```
        instID = rs.Fields("institution_id") & String((30 - Len(rs.Fields("institution_id"))), "
")
        ssan = String(11, " ")
        'statgat1 = String(3, " ") 'stat(ActiveCell.Offset(0, 2).Value) ' muokataan funktiossa
tilastoryhmä1
        statgat1 = "ZZ "
        statgat2 = String(3, " ") 'stat2(ActiveCell.Offset(0, 3).Value) ' muokataan funktiossa
tilastoryhmä2
        statgatx = String(24, " ")
        nametyp = "1"
        If Len(rs.Fields("last_name")) < 30 Then
        ' sukunimi
        surname = rs.Fields("last_name") & String((30 - Len(rs.Fields("last_name"))), " ")
        Else
        surname = Left(rs.Fields("last_name"), 30)
        End If
        If Len(rs.Fields("first_name")) < 20 Then
        ' etunimet
        firstname = rs.Fields("first_name") & String((20 - Len(rs.Fields("first_name"))), " ")
        Else
        firstname = Left(rs.Fields("first_name"), 20)
        End If
        middlename = String(20, " ")
        ptrtitle = String(10, " ")
        transactcountall = String(65, " ")
        addrscount = "2"
        addrID = String(10, " ")
        addrtype = "1"
        addrstatus = "N" ' asetaan siirrettävien osoitteet holdiin, alla osoite- ja puhelinkentät
```

(jatkuu)

LIITE 3. (jatkoa)

```
  addrbegindate = "2003.08.11"

        addrenddate = "2030.12.31"
        addrline1 = rs.Fields("lisattavat_patronit27012011_ADDRESS_LINE1") &
String((50 - Len(rs.Fields("lisattavat_patronit27012011_ADDRESS_LINE1"))), " ")
        addrline2to5 = String(160, " ")
        city = rs.Fields("lisattavat_patronit27012011_city") & String((40 -
Len(rs.Fields("lisattavat_patronit27012011_city"))), " ")
        addrstate = String(7, " ")
        zipcode = rs.Fields("lisattavat_patronit27012011_zip_postal") & String((10 -
Len(rs.Fields("lisattavat_patronit27012011_zip_postal"))), " ")
        country = String(20, " ")

        primphone = String(25, " ")
        'primphone = rs.Fields("aspuh") & String((25 - Len(rs.Fields("aspuh"))), " ")
        'apunro = rs.Fields("aspuh")
        'purettu = Split(apunro, " ") ' puretaan osiin välilyönnein
        'apunro = Join(purettu, "") ' liitetään osat takaisin
        'purettu = Split(apunro, "-") ' puretaan osiin väliviivoin
        'apunro = Join(purettu, "") ' liitetään osat takaisin
        'apunro = Replace(apunro, "+358", "0")
        'numeronpituus = Len(apunro)
        "tehdään vielä muotoon xxx xxx xxxx
 'apunro = Left(apunro, 3) & " " & Mid(apunro, 4, 3) & " " & Mid(apunro, 7,
numeronpituus)
        'primphone = apunro
        "MsgBox primphone
        'primphone = primphone & String((25 - Len(primphone)), " ")
        "MsgBox rs.Fields("aspuh") & String((25 - Len(rs.Fields("aspuh"))), "-")
```

(jatkuu)

LIITE 3. (jatkoa)

```
"MsgBox primphone

otherphones = String(75, " ")
dateadded = String(10, " ")
'sähköposti
addrID_2 = String(10, " ")
addrtype_2 = "3"
addrstatus_2 = "N"
addrbegindate_2 = "2003.03.31"
addrenddate_2 = "2012.01.31"

If Len(rs.Fields("LAKKIDB_PATRON_ADDRESS_ADDRESS_LINE1")) < 50
Then
addrline1_2 = rs.Fields("LAKKIDB_PATRON_ADDRESS_ADDRESS_LINE1") &
String((50 - Len(rs.Fields("LAKKIDB_PATRON_ADDRESS_ADDRESS_LINE1"))), "
")
Else
addrline1_2 =
Left(rs.Fields("LAKKIDB_PATRON_ADDRESS_ADDRESS_LINE1"), 50)
End If
addrline2to5_2 = String(160, " ")
city_2 = String(40, " ")
addrstate_2 = String(7, " ")
zipcode_2 = String(10, " ")
country_2 = String(20, " ")
primphone_2 = String(25, " ")
otherphones_2 = String(75, " ")
dateadded_2 = String(10, " ")
'notesit mukaan jos tarvitaan
```

(jatkuu)

LIITE 3. (jatkoa)

```
'notes_apu = ActiveCell.Offset(0, 11).Value & ", " & ActiveCell.Offset(0, 12).Value
    'notes = notes_apu & String((1000 - Len(notes_apu)), " ")
    ' kasataan merkkijono tiedostoon kirjoitettavaksi
    conv_data = ptrID & ptrbarcodeID1 & ptrbarcode1 & _
    ptrgrp1 & barcodestat1 & barcodemoddate1 & barcode2string & _
    barcode3string & regstdate & ptrexprdate & ptrpurgedate & _
    voydate & voyupd & lbrloccode & instID & ssan & statgat1 & _
    statgat2 & statgatx & nametyp & surname & firstname & middlename & _
    ptrtitle & transactcountall & addrscount & addrID & addrtype & _
    addrstatus & addrbegindate & addrenddate & addrline1 & _
    addrline2to5 & city & addrstate & zipcode & country & _
    primphone & otherphones & dateadded & addrID_2 & addrtype_2 & _
    addrstatus_2 & addrbegindate_2 & addrenddate_2 & addrline1_2 & _
    addrline2to5_2 & city_2 & addrstate_2 & zipcode_2 & country_2 & _
    primphone_2 & otherphones_2 & dateadded_2

    Print #1, conv_data



rs.MoveNext
Loop
rs.Close
Set rs = Nothing
Close #1

End Function
```