Saimaan ammattikorkeakoulu
Tekniikka, Lappeenranta
Tietotekniikka
Organisaation IT-palvelut

Simone Andreani

# Saldonlaskentajärjestelmä Steeri Oy:lle

Opinnäytetyö 2014

## Tiivistelmä

Simone Andreani
Saldolaskentajärjestelmä Steeri Oy:lle, 44 sivua
Saimaan ammattikorkeakoulu
Tekniikka, Lappeenranta
Tietotekniikka
Organisaation IT-palvelut
Opinnäytetyö 2014
Ohjaajat: lehtori Mikko Huhtanen, Saimaan ammattikorkeakoulu, kehityspäällikkö Sami Helin, Steeri Oy

Opinnäytetyössä tehtiin Steeri Oy:lle työtuntien saldolaskentajärjestelmä ja raportointijärjestelmä, jonka avulla työntekijä pystyy seuraamaan työtuntiensa ja saldonsa kehitystä. Saldolaskentajärjestelmän tekoon käytettiin Pentaho Data Integration -työkalua, jonka avulla rakennettiin dataintegraatiot tuntikirjausjärjestelmän ja saldolaskentajärjestelmän välille. Pentaho BI Server asennettiin näyttämään mittaristoa työtekijöille. Mittaristo kehitettiin käyttäen Community Dashboard Editoria, joka oli Pentaho BI Serverin lisäosa.

Tuntikirjausjärjestelmä oli Aalto yliopiston kehittämä AgileFant, josta integraatiot hakivat työntekijöiden kirjaamat työtunnit ja laskivat yli- tai alijäämän näiden perusteella. Yli- tai alijäämä lisättiin tämän jälkeen työntekijän vanhaan saldoon. Kokonaissaldo lähetettiin kerran viikossa työntekijöille ja esimiehille sähköpostitse, jos työntekijän saldo ylitti tai alitti 30 tuntia. Mittaristo kehitettiin työntekijöitä varten.

Opinnäytetyön valmistuttua Steeri Oy:llä oli käytössä valmis järjestelmä, joka laskee työntekijän saldoa automaattisesti tuntikirjausjärjestelmän kautta. Määritysvaiheessa sovitut tarpeet täyttyivät ja esimiehet olivat tyytyväisiä. Järjestelmään jäi vielä kehitysmahdollisuus saldokorjauksiin.

Asiasanat: BI, saldolaskenta, dataintegraatio, mittaristo

## Abstract

Simone Andreani
Balance counting system for Steeri Oy, 44 pages
Saimaa University of Applied Sciences
Technology, Lappeenranta
Degree Program in Information Technology
Organization's IT services
Bachelor´s Thesis 2014
Instructors: Mr. Mikko Huhtanen, senior lecturer at Saimaa University of Applied Sciences, Mr. Sami Helin, development manager at Steeri Oy

This thesis created a system for Steeri Oy, which counted the balance of the employees' logged working hours and a reporting environment. Pentaho Data Integration -tool was used to create the data integrations between the effort system, which was already used by Steeri Oy, and the new balance system. Pentaho BI Server was installed to show the new dashboard for the employees. The dashboard was developed using Community Dashboard Editor, which was a plug-in for the Pentaho BI Server.

The efforts system was AgileFant, a creation of Aalto University. The data integrations extracted the efforts from the efforts system and counted the surplus or deficit and added this to the old balance. The balance was sent once a week to the employees and to the manager, if the employees' balance varied by 30 hours. The dashboard was developed for the employees.

After completing the thesis, Steeri Oy had a complete system, which counted their employees balance automatically using the effort system. The needs that were agreed-on during the definition phase were fulfilled and the managers were satisfied. The balance system could still be improved, because the balance corrections have to be made by hand in the database.

Keywords: BI, balance counting, data integration, dashboard

## Sisältö

## Termistö

| | |
|---|---|
| BI | Business Intelligence. Käsittää liiketoimintatiedon hallinnan, tallennuksen ja analysoinnin. |
| BO | SAP BusinessObjects. SAP:n Business Intelligence ympäristö. |
| CDE | Community Dashboard Editor. Pentaho BI Serverin lisäosa, jolla voidaan kehittää mittaristoja. |
| CRM | Asiakkuudenhallintajärjestelmä. Käytetään pääasiassa asiakastietojen tallennukseen, oman toiminnan ohjaamiseen ja analysointiin sekä kehittämiseen. |
| Dataintegraatio | Tällä tarkoitetaan tiedon siirtämistä tietojärjestelmästä toiseen. |
| DOD | Definition of done. Kertoo, millä perustein projekti tai vastaava on valmis. |
| ETL | Extract-Transform-Load. Tarkoittaa tiedon hakemista, muuntamista ja lataamista johonkin, yleensä tietovarastoon. |
| JDBC | Java Database Connectivity. Java-rajapinta, jolla sovellus pystyy käyttämään tietokantaa. |
| JNDI | Java Naming and Directory Interface. Java-rajapinta, jonka avulla voidaan luoda yhteys tietokantaan. |
| Jobi | PDI:n valmiiksi ohjelmoitu objekti. Hierarkian korkein, jonka alla on Transformaatio ja Step. Ohjaa missä järjestyksessä Transformaatiot käynnistyvät. |
| OBI | Oracle Business Intelligence. Oraclen Business Intelligence -ympäristö. |
| PDI | Pentaho Data Integration. Pentahon dataintegraatio / ETL-työkalu. |
| SaaS | Software-as-a-Service. Ohjelmisto, joka toimii pilvessä. |
| Saldo | Työntekijän yli-/alijäämän kertymä tekemistään työtunneistaan. Saldo on 0, jos työntekijä on tehnyt 37,5 tuntia töitä. |
| SF | Salesforce.com. SF on pilvessä toimiva asiakkuudenhallintajärjestelmä. |

| | |
|---|---|
| Step | PDI:n valmiiksi ohjelmoitu objekti, joka hakee tai käsittelee dataa eri tavoin. Hierarkiassa alimpana. |
| Transformaatio | PDI:n valmiiksi ohjelmoitu objekti. Hierarkiassa keskimmäinen, Jobin alla ja Stepin yllä. Määrittelee, missä järjestyksessä Stepit ajetaan integraatiossa. |

# 1 Johdanto

Steeri Oy on 54 hengen it-yritys, joka toimii asiakastoiminnanohjausjärjestelmien kanssa ja liikkeenjohdon konsultoinnissa. Liikevaihto vuoden 2013 kolmanteen neljännekseen mennessä oli noin 5,3 milj. euroa, josta liikevoittoa 236k euroa (Steeri Oy:n julkiset taloustiedot, Taloussanomat, 2013). Steeri Oy:ssä työntekijöillä on käytössä Aalto yliopiston suunnittelema tuntikirjausjärjestelmä, nimeltään AgileFant. Vuoden 2014 alusta työnantajalla on ollut velvoite valvoa työntekijöidensä liukumatuntien tilannetta, jos työntekijöillä on liukumamahdollisuus.

Tässä opinnäytetyössä tehdään integraatio- ja raportointijärjestelmä, josta työntekijät ja työntekijöiden esimiehet saavat tarvittaessa tarkistettua kirjattujen tuntien sekä liukumasaldon tilanteen. Tämä järjestelmä toteutetaan käyttäen Pentaho Community Edition tuoteperheen tuotteita, jotka ovat maksuttomia. Tavoitteena on saada tuntikirjaukset pitämään paikkansa, sillä monet työntekijät eivät saata muistaa tai eivät yksinkertaisesti kirjaa tunteja tuntikirjausjärjestelmään. Lisäksi tavoitteena on saada esimiehille mahdollisuus tarkistaa, onko hänen tiiminsä ylityöllistetty ja tarvitseeko hänen reagoida jotenkin tilanteeseen.

Opinnäytetyössä työ alkoi määrittelyvaiheella, jossa henkilöstötiimin esimies oli suunnittelussa mukana, antaen näkemystä siitä, mitä järjestelmältä vaadittaisiin. Steeri Oy:ssä kaikki työt tehdään korostaen ketteryyttä ja pyrkien vähentämään niin sanottua turhaa työtä, joten suunnitteluvaiheessa suunniteltiin eri vaiheet vain pääpiirteittäin jättäen varaa vielä muutoksiin. Opinnäytetyössä käydään läpi käytetyt Pentahon tuotteet, näytetään, mitä näillä tuotteilla saadaan aikaiseksi sekä kuvataan lopullinen järjestelmä. Lopuksi pohditaan, mitä saldolaskentajärjestelmällä saavutettiin ja miten koko työprosessi onnistui.

Kokonaisuutena tämä on hyvä opinnäytetyö Business Intelligence (BI) maailmaan perehtymiseen, sillä tässä siirretään ja muunnetaan dataa järjestelmästä toiseen (dataintegraatio) sekä käytetään BI-palvelinta raportointiin ja mittaristojen esillepanoon. Käytettävät työkalut ja työprosessi eivät eroa merkittävästi asiakastöistä, mikä tarkoittaa sitä, että tämä on hyvää harjoitusta myös tuleviin asiakastöihin. Lisäksi tämä järjestelmä tulee työpaikalle aktiiviseen käyttöön, mikä itsessään jo antaa merkityksen tälle opinnäytetyölle.

# 2 Dataintegraation ja ETL-prosessin erot

Tässä opinnäytetyössä käytetään paljon termiä dataintegraatio. Monet sekoittavat dataintegraation ETL-prosessiin (Extract-Transform-Load), sillä molemmissa luetaan dataa, mahdollisesti käsitellään sitä jollakin tavalla ja viedään jonnekin, missä dataa tullaan mahdollisesti käyttämään. Jos ajatellaan asiaa ainoastaan näin, niin eroa ei juurikaan ole, mutta erot ovat siinä, mitä dataa haetaan ja minne sitä viedään.

Dataintegroinnissa dataa haetaan esimerkiksi useasta eri ohjelmistosta, jotka sisältävät dataa samasta asiasta, esimerkiksi tuotetietoja. Nämä tuotetiedot voidaan ottaa esimerkiksi kahdesta eri lähdejärjestelmästä ja viedä kolmanteen järjestelmään, jolloin kolmannessa järjestelmässä on kaikkien kolmen järjestelmän data. Operatiivinen data kopioidaan paikasta A paikkaan B, dataa ei poisteta siirron yhteydessä (Lenzerini, M. 2002). Esimerkkinä olkoon tilanne, jossa yritys haluaa dataa toiminnanohjausjärjestelmästään verkkokauppaansa. Yrityksellä on x määrä tuotteita y määrästä tuotteita, jotka pitäisi saada verkkokaupan tietokantaan. Tähän voidaan rakentaa dataintegraatio, joka hakee yrityksen toiminnanohjausjärjestelmästä verkkokauppaan lisättävät tuotteet sekä tarvittavat tiedot niille ja siirtää ne verkkokaupan tietokantaan. Tässä dataa kopioidaan yhdestä operatiivisesta järjestelmästä, toiminnanohjausjärjestelmästä, toiseen, joka oli yrityksen verkkokauppa.

ETL-prosessissa dataa otetaan lähdejärjestelmästä, esimerkiksi yrityksen toiminnanohjausjärjestelmästä ja käsitellään tarvittavalla tavalla, jotta data saadaan vietyä tietovarastoon. Tietovaraston data ei ole operatiivisessa käytössä, eli se ei ole minkään ohjelmiston käyttämä tietokanta, vaan tietokanta, jonka rakenne on erityisesti tehty raportointia ajatellen. Joskus yritys ei halua pitää kaikkea dataa ohjelmistonsa tietokannassa, vaan haluaa siirtää yleensä operatiiviselta kannalta irrelevantin datan tietovarastoon. Käytetään esimerkkinä yrityksiä, jotka käyttävät Salesforce.com-asiakkuudenhallintajärjestelmää (SF). SF tarjoaa rajatun määrän tallennustilaa tietokannoistaan perushintaan, mutta hinnat lähtevät nousemaan erittäin nopeasti, jos rajat ylittyvät. Tällöin yrityksen kannattaa harkita irrelevantin datan siirtämistä tietovarastoon ja ainoastaan operatiivisen datan säilyttämistä SF:ssa. Tässä tarvitaan ETL-prosessia, joka hakee tarvittavat tiedot SF:n

tietokannoista, muuntaa datan tietovarastoon sopivaksi ja syöttää datan tietovarastoon. Tämän jälkeen irrelevantin datan voi poistaa SF:n tietokannoista, sillä tämä data löytyy nyt tietovarastosta, mistä sitä voi myöhemmin hyödyntää mm. raportoinnissa tai muussa tarpeellisessa toiminnassa (af Hällström, K. 2014.)

# 3 Määrittely ja suunnittelu

Kaikissa projekteissa tarvitaan selkeä päämäärä ja tieto siitä, kuinka kyseiseen päämäärään päästään. Lisäksi on hyvä suunnitella aikataulu eri projektin vaiheille, jos tämän on tarkoitus valmistua ennalta määriteltyyn aikaan. Tässä tapauksessa kyse oli sisäisestä projektista eikä meillä ollut tarvetta laittaa tätä työtä asiakastöiden eteen, joten aikataulu tässä projektissa oli hyvin avoin. Tässä on hyvät ja huonot puolensa. Jos aikataulu on liian vapaa ja lisäksi kyseessä on sisäisestä työstä, voi syntyä herkästi tilanteita, joissa työaika painotetaan asiakastyöhön. Tällöin projektin valmistumisaikaa on hyvin vaikea ennakoida ja myöskään saada edistettyä. Parhaimmassa tapauksessa taas aikataulussa on liukumavaraa, mutta projektille on suunnattu työaikaresursseja, jolloin projektia voi edistää silloin, kun on ylimääräistä aikaa tai sille voi jopa tehdä aikaa. Kokonaisuudessaan opinnäytetyöhön meni aikaa yli 200 tuntia, joka alitti alkuperäisen 300 tunnin työaika-arvion.

Kun sain työtehtävän hoitaakseni, se oli määritelty pääpiirteittäin, mutta sen pohjalta ei vielä voinut sanoa, missä vaiheessa projektin voisi sanoa olevan valmis. Suunnitteluun minulle annettiin pitkälti vapaat kädet, kunhan järjestelmä tekisi lopulta kaiken, mitä siltä vaadittiin ja se olisi valmis 2014 vuodenvaihteeseen mennessä.

## 3.1 Järjestelmän määrittely

Määrittelyssä oli mukana Jussi Lukkarila, joka on toisen tiimin tiiminvetäjä, sekä oma esimieheni, Pasi Helasuo. Päätarpeet tulivat Lukkarilalta sähköpostitse, joihin minä ja Helasuo vastailimme tarkentavilla kysymyksillä. Lopulta pitkän sähköpostiketjun lopputuloksena saimme listan tarpeista, jotka järjestelmän täytyisi täyttää.

Ensimmäinen ja tärkein tarve oli tietenkin, että työntekijän saldo laskettaisiin AgileFant-tuntikirjausjärjestelmää käyttäen. Tämä tarkoitti sitä, että AgileFantin ja saldolaskentajärjestelmän väliin täytyisi tehdä dataintegraatio, joka hakee AgileFantin tietokannasta työntekijöiden tuntikirjaukset ja laskee erikseen jokaisen työntekijän saldon jokaiselle päivälle. Esimerkiksi, jos kyseessä olisivat minun tuntini, niin järjestelmän tulisi hakea tuntikirjaukseni ja laskea saldoni sen mukaan, paljonko olisin tehnyt tunteja yli tai alle 7,5 h, joka on normaalin työpäivän pituus.

Toinen tarve oli, että järjestelmä laskee ainoastaan nykyisen ja edellisen viikon saldoja eikä muuta edellisten viikkojen saldoja. Näin työntekijä ei pysty muuttamaan saldoaan itse, vaan muutokset menevät työntekijän esimiehen kautta meidän tiimille. Tällä tavalla saadaan työntekijät oikeasti merkitsemään tuntinsa ajoissa tuntikirjausjärjestelmään, sillä projektipäälliköt käyttävät tuntikirjausjärjestelmään merkittyjä tunteja asiakkaille lähtevien laskujen laskemiseen. Jos työntekijä ei ole pitänyt tunteja ajan tasalla, laskun summa ei pidä paikkaansa. Saldon lukematta jättäminen on tarpeellista myös siksi, että työntekijä ei voi mennä lisäämään tehtyjä työtunteja edellisille viikoille ja täten muuttamaan saldoaan luvatta.

Kolmas tarve oli, että järjestelmä pystyisi lähettämään saldot työntekijöille sähköpostitse sekä lähettämään esimiehille sähköpostia, jos työntekijän saldo on ylittänyt tai alittanut 30 tuntia. Sähköpostituksen haluttiin tapahtuvan joka tiistaiaamu, koska monet työntekijät merkitsevät edellisen viikon tunteja vielä maanantaina. Työntekijän sähköpostin tuli sisältää työntekijän edellisen viikon tekemät työtunnit ja saldo viikon alkuun. Työnantajan sähköpostin taas tuli sisältää työntekijöiden nimet ja heidän saldot, jos 30 tunnin rajat olisivat ylittyneet. Sähköpostitus oli itseasiassa alkuperäisissä määrityksissä ainut osio järjestelmässä, joka näkyi työntekijöille.

Edelliset kolme tarvetta tulivat Lukkarilalta ja hioimme niitä yhdessä sellaisiksi, että ne voisi toteuttaa. Neljännen päätin lisätä itse, koska näin tarpeen työntekijöiden puolelta, jota ei ollut huomioitu aikaisemmin. Tämä päätös tuli muita määrittelyitä hieman myöhemmin ja itseasiassa saldolaskenta oli jo käynnissä siinä vaiheessa, kun aloitin tekemään tätä lisäystä. Kehitykseen varattiin 300 tuntia testauksineen ja muutostöineen.

Päätin tehdä BI-mittariston, josta työntekijät voisivat tarkistaa työtuntinsa ja saldonsa lähes reaaliaikaisesti. Kolme edellistä tarvetta antoivat raamit mittariston suunnitteluun, mutta näihin täytyi lisätä vielä aikadimensio, jotta mittaristossa olisi mitään järkeä. Joten päätin, että järkevät aikavälit voisivat olla 3, 6 ja 12 kuukautta. Mittariston oli tarve päivittyä mahdollisimman nopeasti, sillä työtunteihin tehdyt muutokset haluttiin näkyvän mittaristossa pikimmiten. Lisäksi tämän tulisi integroitua Steeri Oy:n muokkaamaan Customer Relationship Management (CRM) -järjestelmään, Force.comiin. Force.com on käytännössä Salesforce.com, jota voi muokata haluamakseen. SF on Software-as-a-Service (SaaS) järjestelmä, johon päästään käsiksi Internet-selaimen kautta. (Boucher Ferguson, R. 2008.)

Kaikki tarpeet oli keskusteltu sähköpostin välityksellä, mutta halusin vielä varmistaa, että annetut tarpeet eivät muuttuisi tai niitä tulisi lisää projektin edetessä. Tämän vuoksi järjestin pikaisen palaverin minun ja Lukkarilan kesken, jossa kävimme tehdyt määritykset läpi ja sovimme, että kun tarpeet on täytetty, niin projekti olisi valmis, eli ”Definition of Done” (DOD) täyttyisi. Koko määritysvaihe kesti noin kolme työviikkoa kovan asiakastyöpaineen vuoksi.

### 3.2 Järjestelmän suunnittelu

Suunnitteluvaiheen halusin pitää mahdollisimman lyhyenä, sillä minä itse toteutin koko tuntikirjausjärjestelmän, joten en halunnut lisätä turhan byrokratian määrää, vaan pitää koko toteutusvaiheen mahdollisimman vapaana, koska olin saanut vapaat kädet toteuttaa projektista omanlaiseni. Lisäksi tarkoituksena oli saada työ tehtyä muun asiakastyön ohessa, eli käytettyjen tuntien määrää haluttiin pitää mahdollisimman pienenä. Steeri Oy:ssä on huomattu, että dataintegraatioiden kehitysvaiheessa ei kannata liikaa suunnitella dataintegraatioiden toimintaa, vaan lähinnä suunnitella, mitä tietoa tarvitaan, mistä ja mitä tietoa haetaan sekä mitä datalle tehdään ennen kuin se syötetään tietokantaan. Dataintegraatiot kehitetään ja muokataan matkan varrella, mikä on ehdottomasti omaankin työtyyliin hyvin sopiva, varsinkin jos töitä tekee ainoastaan yksi henkilö.

### 3.2.1 Dataintegraatioiden suunnitteluvaihe

Oma kokemus Pentahon tuotteista oli alussa vielä olematon, mutta pystyin katsomaan asiakastöiden integraatioista ja mittaristoista hieman esimerkkiä. Ohjelmia on hyvin helppo käyttää, sillä opettelemalla muutamat perusobjektit, on mahdollista tehdä alkeellisia dataintegraatioita pienen harjoittelun jälkeen. Järjestelmän suunnitteluun käytin noin kolme henkilötyöpäivää dataintegraatioiden osalta, sillä opiskelin ohjelman toimintaa työn ohessa ja tein järjestelmään muutoksia sitä mukaa, kun sain lisää tietoa ohjelman toiminnasta. Tarkemmin itse dataintegraatioista kerron myöhemmissä luvuissa.

### 3.2.2 Mittariston suunnitteluvaihe

Mittariston suunnitteluun käytin noin viisi henkilötyöpäivää kokonaisuudessaan. Kollegani Kristian af Hällström auttoi minua paljon mittariston suunnittelun kanssa, sillä hän oli tehnyt lähes kaikki Community Dashboard Editor (CDE)-mittaristot meidän yrityksessä. Ensin täytyi suunnitella, mitä työntekijä pääasiassa haluaa saada selville saldostaan ja työtunneistaan, kun hän aukaisee mittariston. Tämän avulla pystyisin suunnittelemaan, miten esitän tarvittavat tiedot ruudulla siten, että työntekijän tarvitsisi lähinnä vain vilkaista ruutua ja saisi kaiken tarvittavan tiedon siitä (af Hällström, 2014).

Koska tästä tuli ensimmäinen suunnittelemani mittaristo, en oikein osannut suunnitella sitä paperille etukäteen, mutta af Hällström sanoi, että se ei ole ongelma, kunhan tietäisin mitä aion ruudulla esittää. Seuraava vaihe oli saada ideoista toteutettua CDE-mittaristo.

## 4 Pentaho-tuoteperhe

Steeri Oy:ssä käytetään asiakastöissä paljon erilaisia BI-työkaluja, joista osa on ilmaisia ohjelmia, mm. Pentahon Community Edition (CE)-tuoteperheen tuotteita, sekä maksullisia, kuten SAP:n BusinessObjects (BO) tai Oracle Business Intelligence (OBI). Normaalisti Steeri Oy:n tarjoamissa ratkaisuissa pyritään välttämään maksullisia ohjelmistoja, sillä niiden lisenssit ovat hyvin kalliita sekä vaati-

vat melko paljon palvelimilta, joilla nämä ohjelmistot ovat käynnissä. Oppimiskäyrä ohjelmistojen kanssa on myös melko jyrkkä ja ne vaativat melko paljon omistautumista, jotta ohjelmiston kanssa pääsisi sinuiksi. Toisaalta niissä on hyviäkin puolia, varsinkin siinä vaiheessa, kun yritys on iso ja dataa on erittäin paljon. Itse olen tehnyt töitä BO:n kanssa ja olen huomannut melko huomattavia eroja dataintegraatioiden nopeuksissa, mutta tämäkin tietty riippuu monesta tekijästä. Toinen hyvä asia maksullisissa ohjelmistoissa on, että sen asennukseen, käyttöön ja testaukseen saa hyvin kattavaa apua tukisivuilta tai jopa puhelimitse. (SAP BusinessObjects BI Suite Master Guide, 2014.)

Pentahon tuotteet ovat helppokäyttöisiä ja nopeasti opittavia, mutta sillä kustannuksella, että ongelmatilanteissa täytyy turvautua Googlen tehokkaaseen käyttöön ja foorumeiden selailuun. Pentahon tuoteperheestä on olemassa maksullinen versio, Enterprise-versio, mutta me käytämme ainoastaan Pentaho Community Edition (CE)-versiota, koska asiakkaille on melko vaikea perustella maksullisen version paremmuutta, jos erona ovat vain tukipalvelut ja samalla meidän kuuluisi tarjota heille käyttötuki. Pentaho CE-tuotteet ovat pääasiassa kehitetty Pentahon toimesta, mutta ne sisältävät myös korjauksia ja uusia ominaisuuksia yhteisön kehittäjiltä, jotka auttavat tuotteiden kehityksessä. Pentaho-yhteisö on melko laaja ja hyvin aktiivinen. Ohjelmistovirheet ilmoitetaan Pentahon sivuille hyvin nopeasti heti uuden päivityksen tultua, ja ongelmien tilanteita voi seurata ja kommentoida Pentahon sivuilta (Pentaho Community, 2014). Lisäksi Pentahon yhteisöfoorumille voi laittaa kysymyksiä, joihin useasti saa vastauksen saman päivän aikana.

Pentahon tuoteperheeseen kuuluu hyvin monta tuotetta, mutta minä käytin tässä projektissa kolmea työkalua, jotka Steeri Oy:ssä on käytössä hyvin monessa asiakasprojektissa. nämä olivat Pentaho Data Integration (PDI), Pentaho BI Server ja Community Dashboard Editor (CDE). Alakappaleissa syvennytään hieman tarkemmin kyseisiin työkaluihin.

### 4.1 Pentaho Data Integration 4.8

PDI on ilmainen visuaalisen ohjelmoinnin dataintegraatiotyökalu, jota pääsääntöisesti käytämme Steeri Oy:ssä suunnittelemien projektien dataintegraatioihin.

PDI, kuten muutkin Pentaho CE tuoteperheen tuotteet, on ilmainen. Lisäksi työkalun kehityksessä on vahvasti mukana alan asiantuntijoita ja PDI:n tehokäyttäjiä, jotka ilmoittavat ohjelmistossa löytyvistä virheistä ja antavat tärkeää palautetta sekä kehitysehdotuksia PDI-kehittäjille. Tämän avulla PDI:n kehitys tapahtuu paljon nopeammin kuin jos sitä kehittäisivät ja korjaisivat ainoataan Pentahon omat kehittäjät. Näin PDI:stä on tullut erittäin vakaa sisältäen uusimpia ominaisuuksia, kuten Hadoop-Stepit (Stepeistä enemmän alakappaleessa) Big Dataa varten ja paljon muitakin. Suurta yhteisöä kannattaa hyödyntää, varsinkin kun se on aktiivinen ja valmiina auttamaan tuotteen kehityksessä, ja Pentaho on onnistunut hyödyntämään tämän tilanteen hyvin.

Työkalun opiskelua varten on kirjoitettu myös paljon materiaalia, joista muutamat ovat Matt Castersin, yksi PDI:n alkuperäisiä luojia, kirjoittamia. Hyviä kirjoja ovat mm. ”Pentaho Data Integration 4 Cookbook” ja ”Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration”.

### 4.1.1 Jobit ja Transformaatiot

PDI:ssä käytetään eri nimityksiä eri objekteista, jotka kuuluvat työkaluun. Helpoiten nämä on hahmottaa, jos aloitetaan kertomalla hierarkia. Ensimmäisenä PDI:ssä luodaan Job-tiedosto, joka on päätason objekti. Jobin sisään voi halutessaan luoda toisen Jobin tai luoda Transformaation. Jobit sekä Transformaatiot tallennetaan tiedostoiksi ja voidaan myöhemmin avata uudestaan. Jobin sisällä voidaan käyttää muutamia Steppejä, mutta reippaasti vähemmän kuin Transformaatioissa. Eli voidaan ajatella niin, että Jobit järjestelevät, missä järjestyksessä Transformaatiot ajetaan, kun taas Transformaatioiden sisällä suoritetaan datan haku, tarvittaessa datan muuntelu ja lopulta datan syöttö kohteeseen. Huomioitavaa on, että toistaiseksi Transformaation sisällä ei voida käyttää Jobeja, sillä tätä PDI ei tue. tähän on tosin mahdollisesti tulossa muutos lähitulevaisuudessa. (Pulvirenti, A.S. & Roldán M.C., 2011.)

Jos Jobin käynnistää, niin ennen kuin data alkaa virrata Transformaatioiden sisällä, PDI tarkistaa koko integraation syntaksin. Näin integraatioissa olevat suurimmat virheet, kuten väärin käytetyt Stepit, väärät tietokantayhteydet tai vastaavat ongelmat huomataan ajoissa ja ne voidaan korjata ilman, että vääränlaista

dataa menee minne sitä ei pitäisi mennä. Jobien sisällä voidaan myös määritellä muuttujia, jotka voidaan hakea alaobjekteihin, kuten Transformaatioihin tai toisiin Jobeihin. Myös tietokantayhteydet voidaan määrittää Jobi-tasolla ja päättää jakaa alemmille objekteille, jolloin kyseisille objekteille ei tarvitse käydä luomassa itse tietokantayhteyksiä, vaan ne periytyvät ylemmältä tasolta.

Koska kaikki tapahtuu sarjassa, niin data ei liiku Transformaatiosta toiseen ennen kuin kaikki data on käynyt edellisen Transformaation läpi. Jos data liikkuisi toiseen Transformaatioon samalla, kun toinen on vielä käynnissä, se olisi rinnakkain. Steppejä voi ajaa rinnakkain, mutta ei Jobeja eikä Transformaatioita. Dataa voi kopioida edellisestä Transformaatiosta kahteen seuraavaan, jolloin yhdestä Transformaatiosta voi mennä yhteys kahteen eri paikkaan, mutta normaali tilanne on se, että yksi lähde tulee Transformaatioon ja yksi lähtee pois, jos kyseessä ei ole integraation päätepiste (Pulvirenti, A.S. & Roldán M.C., 2011.)

### 4.1.2 Stepit ja niiden käyttö

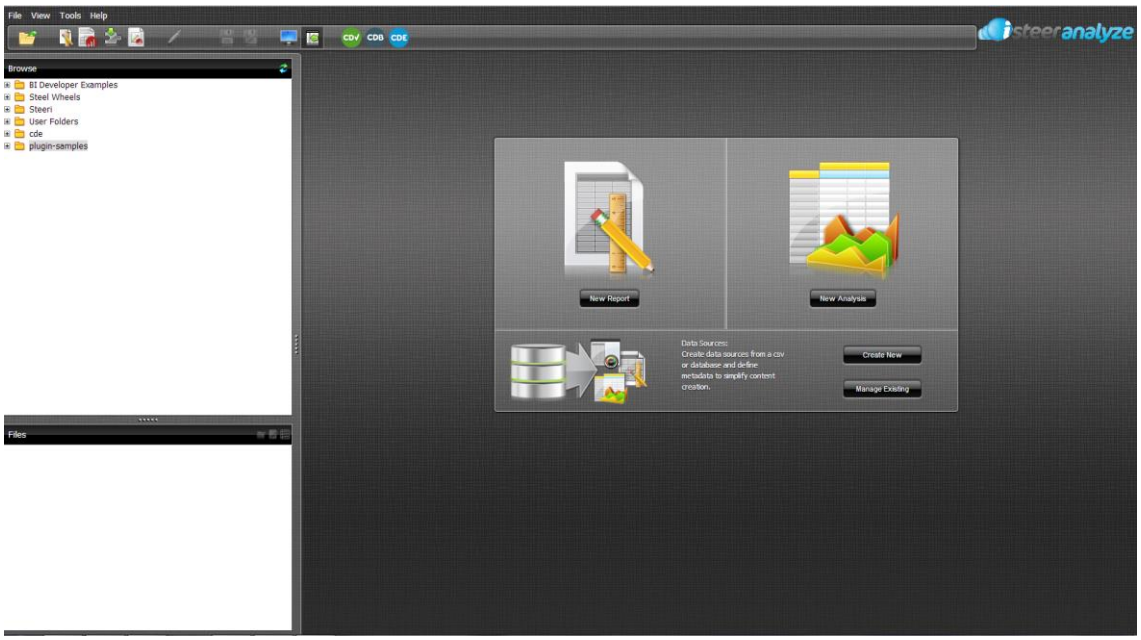
PDI käyttää ohjelmoinnissa valmiiksi koodattuja objekteja nimeltään Step ja paremman sanan puutteessa käytän tässä opinnäytetyössä PDI:n määritelmää omasta työkalun osasta. Stepit tekevät erinäisiä asioita, riippuen Stepistä, esimerkki yhdestä voidaan pitää sellaista kuin ”Table Input” -Step, joka ottaa yhteyden haluttuun tietokannan tauluun ja hakee dataa taulusta sen mukaan, millaista SQL:ää on kirjoitettu kyseisen Stepin sisään. Tässä ei tarvitse käyttää Javaa itse ollenkaan, vaan kyseisen Stepin käyttö generoi tarvittavan Java-koodin itsestään taustalla. Tämä tarkoittaa, että vaikka integraatioiden kehittäjä ei osaisi koodata Javaa, hän voisi silti kehittää integraatioita ilman esteitä.

Valmiita Steppejä on PDI:ssä noin 100, mutta kaikkia Steppejä voi tarvittaessa muokata ja tallentaa ominaan tai jopa tehdä omia Steppejä Javalla ja käyttää niitä kuten muiden tapaan dataintegraatioissa. Steppejä voi käyttää dataintegraatioissa Jobien ja Transformaatioiden sisällä (Casters, M., Bouman R. & van Dongen, J., 2010). Minun käyttämiäni Steppejä käyn tarkemmin läpi luvuissa, joissa rakennan itse dataintegraatiot.

## 4.2 Pentaho BI Server 4.8

PDI ei tarvitse BI-palvelinta toimiakseen. Se hoitaa datayhteyksien luonnit itsenäisesti ja ohjelmisto pyörii omana prosessinaan. Mittaristot eivät toimi ilman, että jokin palvelin pystyy jakamaan niitä. Tähän tarvitaan Pentaho BI Server. Tätä palvelinta tarvitaan raporttien, mittaristojen ja analyysien näyttämiseen ja tekemiseen. Olennaisin osa tätä palvelinta on sisäänrakennettu Apache Tomcat -ohjelmistopalvelin. Tämä Java-ohjelmistopalvelin mahdollistaa kaiken tämän ja samalla hallitsee verkkokäyttöliittymää, josta mittaristot ym. rakennetaan ja josta niitä pääsee katsomaan. Kuvassa 1 on BI palvelimen sisäänkirjautumisikkuna ja kuvassa 2 on BI palvelimen näkymä, kun siihen ollaan kirjautumassa sisään. Kuvat on otettu Internet-selaimen ikkunasta, mutta kuvista on rajattu osoiterivit tietoturvan vuoksi pois.


*Kuva 1. BI palvelimen sisäänkirjautumisikkuna (Kuvakaappaus BI-palvelimesta).*




*Kuva 2. BI palvelimen aloitussivu (Kuvakaappaus BI-palvelimesta)*

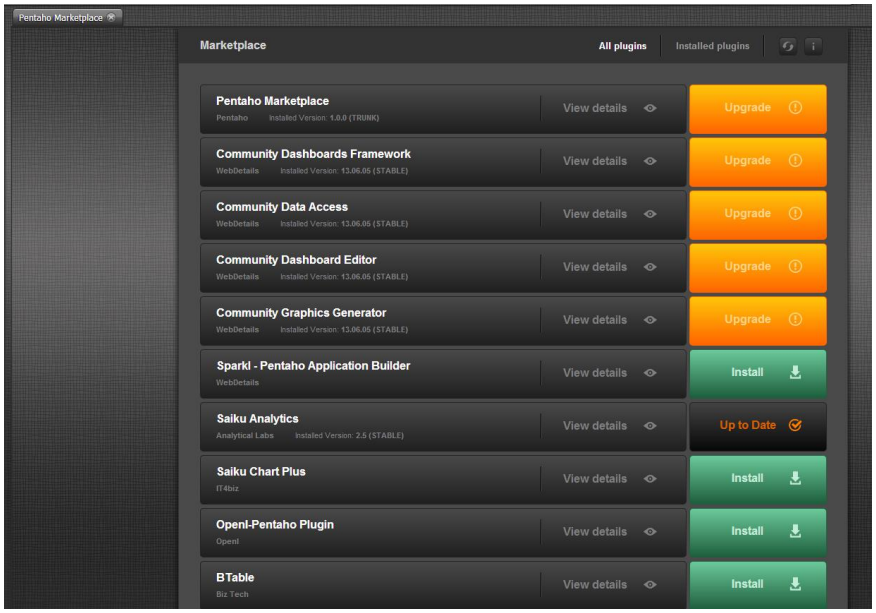
Kuvassa 2 näkyy, miten hyvin palvelimen lisäosat on integroitu siihen. Yläpalkissa näkyy monta erilaista kuvaketta. Osa niistä on palvelimeen asennettua lisäosaa. Kiinnostavin kuvakkeista on lähinnä ensimmäinen oikealta, joka on CDE Dashboards -lisäosa. Vasemmalla näkyvä puurakenne on täysin muokattavissa oman tarpeen mukaan ja ulkoasun voi muuttaa muokkaamalla palvelimen konfiguraatiotiedostoja. (Pochampalli, A. 2013.)

BI-palvelinta ei varsinaisesti tarvitse erikseen asentaa, vaan sen voi hakea Pentahon kotisivun kautta ilmaiseksi ja purkaa .zip-tiedosto haluttuun hakemistoon. Tämän jälkeen palvelimen voi teoriassa käynnistää suoraan, mutta palvelin kannattaa konfiguroida, varsinkin jos palvelin on tulossa tuotantokäyttöön (Pochampalli, A. 2013). Palvelimen konfigurointi käydään läpi omassa luvussaan.

### 4.3 Marketplace ja CDE Dashboard

Community Dashboard Editor on Pentaho BI Serverin lisäosa. Se on ilmainen ja täysin Pentaho-yhteisön kehittämä, kuten nimikin jo kertoo. Tämä liitännäinen tuli asentaa ennen BI-palvelimeen lisätyökalulla nimeltä CTools, joka haki tarvittavat tiedostot Internetistä ja asensi ne BI-palvelimen sisäisiin kansioihin. Nykyään Pentaho BI Serverissä on sisäinen lisäosa, nimeltään ”Marketplace”, mistä voi valita halutut lisäosat ja asentaa ne vain hiiren klikkauksella. Ne voi myös poistaa

samasta paikasta yhtä helposti. Kuvassa 3 näkyy Marketplacesta löytyvät tämänhetkiset lisäosat.

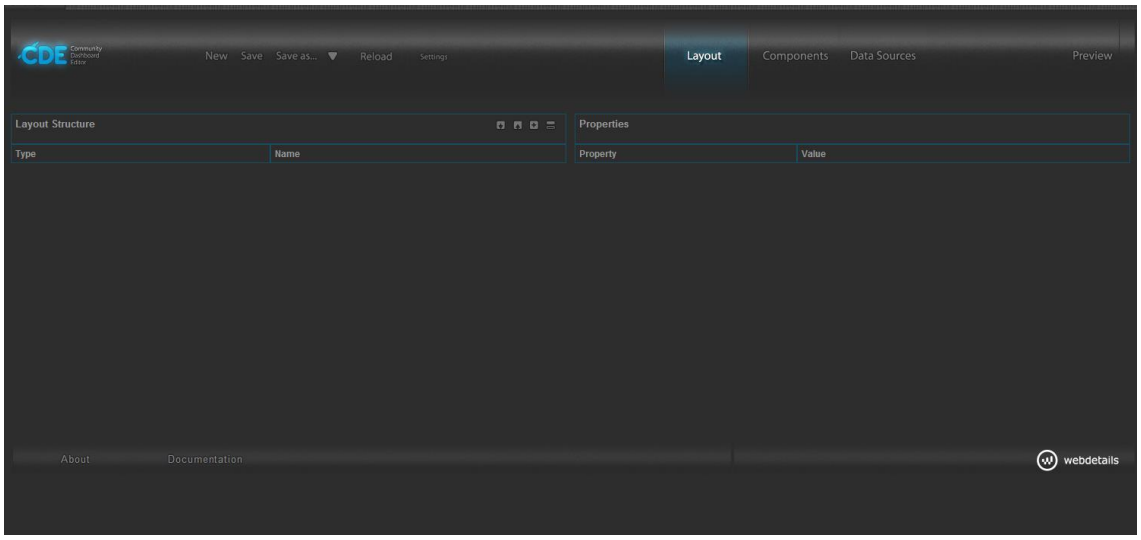



*Kuva 3. BI palvelimen "Marketplace" (Kuvakaappaus)*

Lisäosan asennuksen jälkeen BI-palvelin täytyy käynnistää uudelleen käsin, eli muutoksia ei kannata tehdä, jos ei ole varma, että palvelimen voi käynnistää uudelleen heti tarvittaessa. Yleensä lisäosista on olemassa STABLE- ja TRUNK-versiot, joista STABLE on tarkoitettu tuotantokäyttöön ja TRUNK lähinnä testauskäyttöön.

Sen jälkeen, kun CDE Dashboardin on asentanut Marketplacen kautta ja käynnistänyt BI-palvelimen uudelleen, CDE-kuvake ilmestyy BI palvelimen työkalupalkkiin, kuten kuvasta 2 nähdään. Kuvassa 4 on CDE Dashboardsin kehitysikkuna, jossa mittaristo rakennetaan. Oikeassa yläreunassa näkyy kolme osiota, jotka ryhmittävät mittariston kehityksen kolmeen osaan. Nämä osat ovat Layout, Components ja Data Sources. Otetaan esimerkiksi tilanne, jossa mittaristossa halutaan näyttää kuluvan vuoden bruttotulot viivakaaviossa. Components-osiossa määritellään haluttu viivakaavio ja tarvittavat parametrit, Data Source -osiossa haetaan tarvittava data annetuilla parametreilla ja Layout-osiossa määritellään, minne päin ruutua viivakaavio sijoitetaan. Mittaristo on paljon helpompi toteuttaa, jos pitää mittaristoa kolmena eri osana, sillä jos ajatuksia ei saa pidettyä selkeinä, kehitysvaiheessa voi mennä helposti sekaisin mittariston rakenteesta.

Lisäksi jos kehitysvaiheessa tekee virheitä, CDE ei osaa itsessään etsiä siitä virheitä, eli kehityksessä täytyy olla tarkkana ja tehdä yksittäisiä muutoksia sekä lisäyksiä.



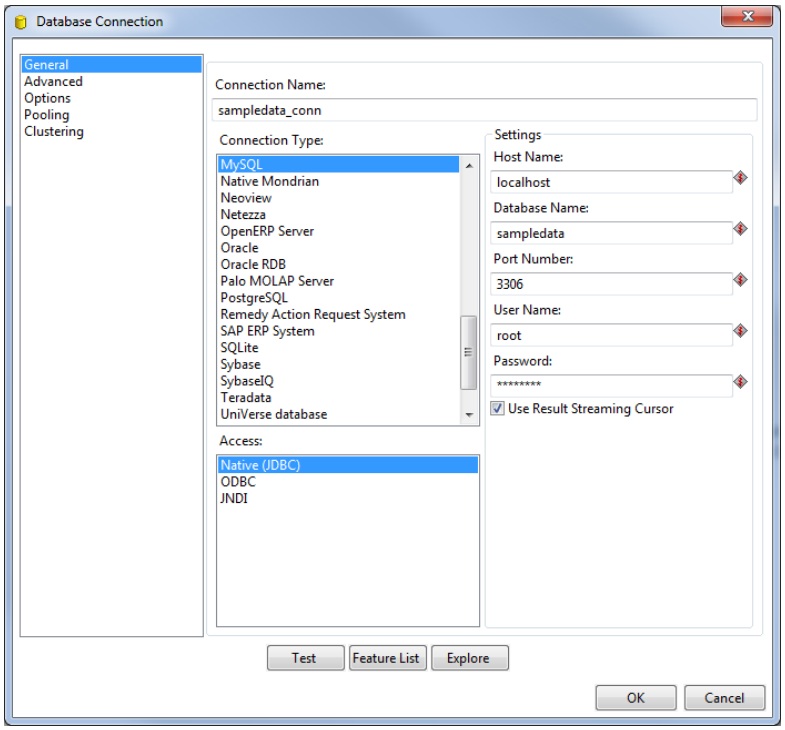

*Kuva 4. CDE Dashboardsin kehitysikkuna (Kuvakaappaus)*

# 5 Dataintegraatioiden toteutus

Integraatio jakaantui lopulta kahteen osaan, saldolaskentaan ja saldon sähköpostitukseen. Saldolaskenta täytyi erottaa sähköpostituksesta, koska mittariston dataa haluttiin päivittyvän samanaikaisesti kuin tuntikirjausjärjestelmä päivittyi, jotta muutokset saldoissa näkyisivät mahdollisimman nopeasti. Tämä ratkaistiin liittämällä kehittämäni saldolaskentaintegraatio osaksi tuntikirjausjärjestelmän integraatioita, ja sähköpostitusintegraatiosta tehtiin oma Pentaho Data Integration (PDI) projekti, joka ajastettiin itsenäisesti. Näin saldo päivittyisi useasti, mutta sähköpostit lähtisivät ainoastaan kerran viikossa. Seuraavissa alaluvuissa syvennytään tarkemmin integraatioiden kehitysprosessiin.

## 5.1 Datayhteyksien luonti

Datayhteyksien luonti on tehty erittäin helpoksi ja luodut yhteydet on helppo testata. Kuvassa 5 on PDI:n uuden yhteyden luontivalikko, jonka avulla voidaan luoda uusi yhteys melkeinpä mihin tahansa tietokantaan.




*Kuva 5. PDI:n tietokantayhteyden luonti (Kuvakaappaus PDI:stä)*

Kuvassa 5 oleva yhteys on malliyhteys, eikä tässä opinnäytetyössä tulla näyttämään virallisia datayhteysasetuksia tietoturvan vuoksi. Opinnäytetyössä on käytetty Microsoft SQL Server -tietokantaa ja lisäksi määritetty PDI:n konfiguraatiotiedostoihin tietokannalle ”Java Naming and Directory Interface” (JNDI) -yhteys, jolloin kuvan 5 valikossa riittää, että valitsee oikean yhteystyypin, Microsoft SQL Server, valitsee ”Access”-valikosta JNDI, jolloin oikeaan sarakkeeseen ilmestyy ”JNDI Name”. Kuvassa 6 on PDI:n JNDI-konfiguraatiotiedosto, josta näkyy, kuinka JNDI-yhteys luodaan tiedostossa.

jdbc.properties

```
1  SampleData/type=javax.sql.DataSource
2  SampleData/driver=org.hsqldb.jdbcDriver
3  SampleData/url=jdbc:hsqldb:hsql://localhost/sampledata
4  SampleData/user=pentaho_user
5  SampleData/password=password
6  Quartz/type=javax.sql.DataSource
7  Quartz/driver=org.hsqldb.jdbcDriver
8  Quartz/url=jdbc:hsqldb:hsql://localhost/quartz
9  Quartz/user=pentaho_user
10 Quartz/password=password
11 Hibernate/type=javax.sql.DataSource
12 Hibernate/driver=org.hsqldb.jdbcDriver
13 Hibernate/url=jdbc:hsqldb:hsql://localhost/hibernate
14 Hibernate/user=hibuser
15 Hibernate/password=password
16 Shark/type=javax.sql.DataSource
17 Shark/driver=org.hsqldb.jdbcDriver
18 Shark/url=jdbc:hsqldb:hsql://localhost/shark
19 Shark/user=sa
20 Shark/password=
21
```

*Kuva 6. PDI:n JNDI-konfiguraatiotiedoston data*

Tarkastellaan esimerkiksi kuvan 6 SampleData-yhteyttä, joka määritellään tiedoston ensimmäisellä viidellä rivillä. Siinä määritetään datatyypiksi SQL-datayhteys, jonka Access-tyyppi on ”Java Database Connectivity” (JDBC). Tämän jälkeen tiedostossa luodaan kantayhteys paikalliseen tietokantaan, jonka nimi on ”sampledata” käyttäjätunnuksella ”pentaho_user” salasanalla ”password”. Samalla tavalla luotiin yhteys Steeri Oy:n Microsoft SQL-palvelimeen, jolloin tietokantayhteyden luonnissa voitiin käyttää JNDI-nimeä. Tällä tavalla käyttäjätunnuksen ja salasanan voi salata myös dataintegraatiokehittäjiltä, sillä heidän ei tarvitse luoda yhteyksiä käyttäen käyttäjätunnuksia. Lisäksi tämä helpottaa tietokantayhteyksien hallinnointia, sillä tietokantayhteydet on määritelty tällöin yhteen paikkaan eikä jokaisessa dataintegraatiossa erikseen.

## 5.2 Integraatioiden Stepit

PDI:ssä on reippaasti yli sata erilaista valmiiksi ohjelmoitua Steppiä (Casters, M., Bouman R. & van Dongen, J., 2010), joten valinnanvaraa löytyy. Normaalissa käytössä riittää tosin vain osa kaikista objekteista. Samaan lopputulokseen voi päästä käyttämällä eri Steppejä, eli eri dataintegraation kehittäjillä voi olla eri tyyli, jolla he käsittelevät dataa päästääkseen samaan lopputulokseen. Tässä on hyviä

ja huonoja puolia. Jos jonkun toisen kehittäjän pitää käydä selvittämään toisen tekemiä integraatioita, selvittelyyn voi mennä helposti paljonkin aikaa. Hyvää on se, että toisen valitsema tyyli on nopeampi, jolloin integraatiot kehittyvät paremmiksi ajan myötä.

Näissä dataintegraatioissa käytettiin lopulta noin viittätoista eri Steppiä. Käytetyt Stepit olivat samoja, joita on käytetty useimmin asiakasprojekteissakin, joten nämä olivat ennestään tuttuja. Täytyy muistaa, että tässä vaiheessa olin nähnyt vain pikaisesti, millaisilta PDI-integraatiot näyttävät. Käytetyt Stepit voisi jakaa kolmeen eri kategoriaan, datan syöttöön, datan käsittelyyn ja PDI:n sisäisiin Steppeihin.

Datan syötöllä tarkoitan PDI:lle dataa hakevia ja yhdistäviä Steppejä sekä PDI:ltä syöttäviä Steppejä. Esimerkki dataa hakevasta Stepistä on ”Table Input” -Step, joka hakee jonkin luodun datayhteyden alla olevan tietokannan taulusta dataa sen mukaan, mitä SQL:ää on kirjoitettu Stepin sisään. Tämä Steppi siis hakee dataa, josta se jatkaa eteenpäin. Dataa yhdistäviä Steppejä on esimerkiksi ”Join Table” -Step, joka yhdistää jonkin datayhteyden päässä olevan taulun avaimilla, jotka tulevat edellisestä Stepistä. Esimerkiksi ”Table Input” -Stepistä tulee henkilön ID-numero, joka linkitetään toiseen tauluun, josta haetaan tämän avaimen avulla henkilön etu- ja sukunimi. Tämän jälkeen integraatiossa voidaan käyttää henkilön id-numeroa, etu- sekä sukunimeä. Nämä ovat siis Steppejä, jotka eivät varsinaisesti muokkaa integraation dataa, vaan syöttävät haluttua dataa integraatiolle.

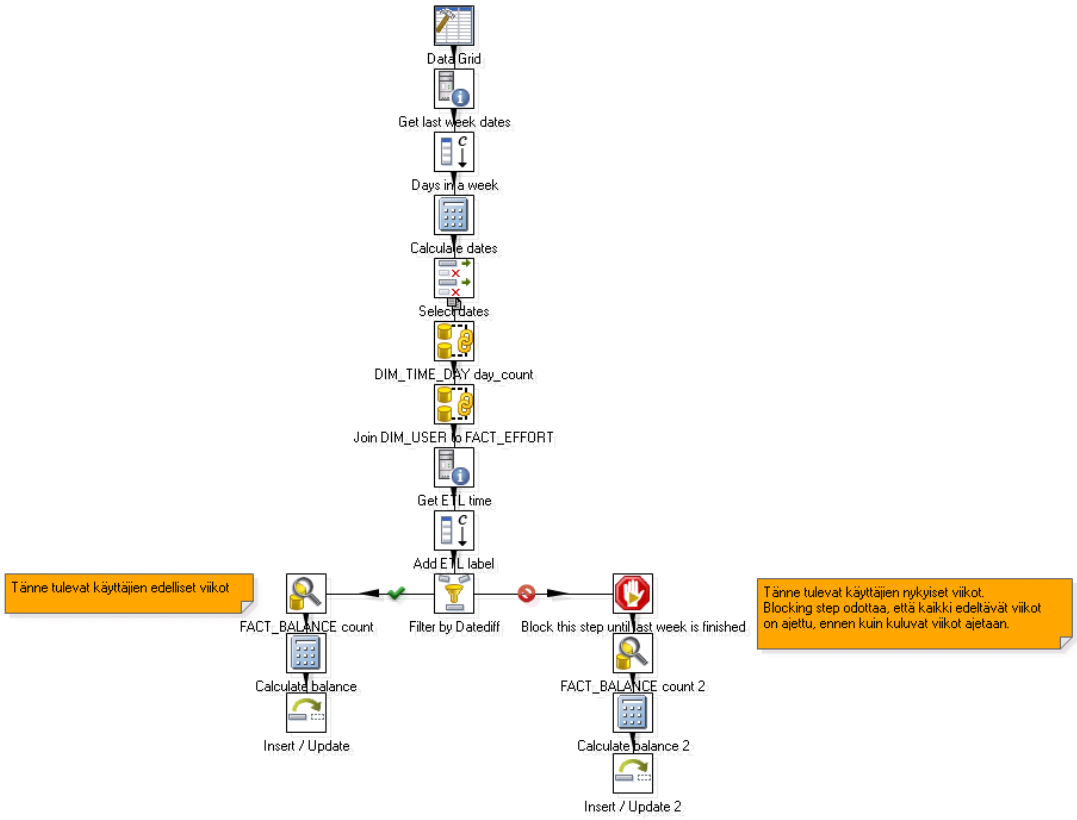
Seuraavana on datan käsittelyyn käytettävät Stepit. Nämä ovat Steppejä, jotka käsittelevät edellisistä Stepeistä tulevaa dataa halutulla tavalla. Näillä Stepeillä data saadaan haluttuun formaattiin tai otettua ainoastaan sellaista tietoa datasta, jota tarvitaan. Esimerkkinä tämän tyyppisistä Stepeistä on Calculator-Step, joka laskee eri kenttien arvoja sen mukaan, mitä laitetaan kaavaksi. Esimerkiksi, datassa on kaksi kenttää, jotka ovat kokonaislukuja, esimerkiksi kenttä x ja kenttä y. Kentän x arvo on 1 ja kentän y arvo 2. Calculator-Stepissä voidaan määrittää kaavaksi tällöin A + B ja määrittää tälle kentälle jokin nimi, vaikkapa ”Sum”, jonka jälkeen integraatiossa on käytettävissä ”Sum”-tietue, jonka arvo on 3. Datan käsittelyyn käytettäviä Steppejä on myös ”Filter Rows”-Step, joka jakaa edellisestä

Stepistä tulevan datan määritettyjen sääntöjen mukaan. Esimerkiksi filtteriä voisi käyttää Calculator-Stepin jälkeen jakamaan eri Steppeihin rivejä sen mukaan, mikä on ”Sum”-tietueen arvo.

PDI:n sisäiset Stepit ovat sellaisia, jotka tuottavat rivejä ilman, että sitä haetaan mistään datalähteestä. Esimerkkinä System Variable -Step, josta voi valita eri systeemin muuttujia, vaikka koneen kellonajan.

## 5.3 Saldolaskentaintegraation kehitys

Saldolaskennan integraatio tehtiin omaksi Transformaatioksi, sillä se tuli yhdistää osaksi tuntikirjausjärjestelmän dataintegraatiota, joka päivitti dataa meidän tietovarastoon. Hyvä puoli tässä oli se, että koska Transformaation voi laittaa Jobin alle suoraan, niin minun ei tarvinnut tehdä tälle omaa Jobia ja ajastaa sitä. Suunnittelin paljon sitä, miten saisin tehtyä integraation määritysten mukaisesti; sen piti pystyä päivittämään työntekijöiden saldoja kuluvalta viikolta ja edelliseltä viikolta tiistaihin asti. Lopputulos näkyy kuvassa 7.




*Kuva 7. Lopullinen saldolaskentaintegraatio (Kuvakaappaus)*

Valikoin siis dataa sen mukaan, mikä viikonpäivä oli kyseessä. Jos integraatio käynnistyisi tiistaina, sen pitäisi vielä päivittää työntekijöiden edellisenkin viikon saldot, mutta jos viikonpäivä olisi keskiviikko, Transformaation tulisi päivittää ainoastaan kuluvan viikon saldot. Tämän seurauksena edeltävä viikko ”lukittuisi” tietokannassa, eikä siihen voisi tehdä muutoksia. Kun sain selvitettyä tämän ajatuksentasolla, suunnittelin, kuinka saisin kehitettyä tällaisen validoinnin integraatiotasolla. Ensimmäiseksi tein integraatiolleni taulun tietokantaan, jonne saisin vietyä työntekijöiden saldot.

### 5.3.1 Saldotaulun luonti

Tein taulun, FACT_BALANCE, jossa oli rivi id, työntekijän id, lasketun viikon ensimmäinen päivä, lasketun viikon viimeinen päivä, kyseisen viikon syötetyt työtunnit tunteina, kyseisen viikon syötetyt työtunnit minuutteina, viikon saldo tunteina, viikon saldo minuutteina, juokseva saldo minuutteina, saldon korjauskenttä, saldon korjauskentän selitekenttä, rivin syöttö-/päivitysaikakenttä ja lähdekenttä.

Rivi-id oli kokonaislukukenttä, jossa oli juokseva lukujono. Tämä kenttä toimi ainoastaan taulun pääavaimena, jotta saisin helpon indeksin tauluun. Kenttää ei käsitellä integraation puolella ollenkaan, vaan uuden rivin tullessa tauluun, se lisääntyy yhdellä. Toinen id-kenttä, oli henkilön id, joka yksilöi saldotaulun rivin viikon ensimmäisen päivän kanssa. Esimerkiksi henkilöllä 1 on ainoastaan yksi rivi, jossa viikon ensimmäinen päivä on 20131230 (tämän vuoden ensimmäinen maanantai). Tämän rivin laskettu viimeinen päivä olisi 20140105. Tämän viikon syötetyt tunnit riippuisivat siitä, mitä työntekijä 1 olisi syöttänyt tuntikirjausjärjestelmään kyseisten päivien ajanjaksolle. Lasketut saldot tälle viikolle riippuisi tietenkin siitä, miten paljon hän olisi työskennellyt tällä viikolla. Tämä lasketaan integraation puolella.

Saldokorjauskenttä on lisätty rivien korjauksia varten. Integraatio ottaa tämän kentän huomioon, mutta ei syötä tähän kenttään mitään, eli korjaukset on tehtävä itse kenttään. Selitekenttä tälle kentälle on tehty sitä varten, kun riviä korjataan, niin tähän kirjoitetaan, miksi riviä on korjattu ja päivämäärä. Tämäkin on tehtävä

käsin. syöttö-/päivitysaika- ja lähdekentät tulevat integraatiossa. Syöttö-/päivitysaika tarkoittaa sitä, milloin integraatio on luonut tai päivittänyt kenttää. Lähdekenttään syötetään, mikä integraatio teki muutoksen. Viimeiset kaksi kenttää eivät ole pakollisia, mutta kuuluvat integraatioiden ”hyvä olla” osioon, sillä nämä helpottavat ongelmatilanteiden selvittämistä.

### 5.3.2 Viikkojen rajaus

Integraatiossa suunniteltiin ensin, kuinka siihen saisi rajattua nykyinen viikko ja edeltä viikko, sillä nämä olisivat ainoat viikot, joita integraation tulisi muokata. Kuvan 7 ensimmäiset viisi Steppiä ratkaisivat tämän. Ensimmäisessä Stepissä luodaan kaksi riviä, joissa on arvot 0 ja 7. Seuraavassa Stepissä näille riveille haetaan palvelimen päivämäärä, vähennetään edellisestä Stepistä tulleiden arvojen verran päiviä tästä päivämäärästä ja haetaan tämän viikon ensimmäinen päivämäärä. Eli jos integraatio ajettaisiin 2.1.2014, joka on torstai, niin riville 0 palautuisi 30.12.2013 ja riville 7 palautuisi 23.12.2013. Nyt integraatiolla olisi edellisen viikon ensimmäinen päivä ja nykyisen viikon ensimmäinen päivä.

Seuraavissa kahdessa Stepissä haetaan lasketuille viikoille edellisen viikon ensimmäinen päivä, jotta integraatio saisi haettua edellisen saldon. Neljäs Steppi lisää kiinteän arvon, -7, ja Calculator-Step vähentää muuttujasta 7 päivää ja palauttaa riville tuloksen. Lopputuloksena esimerkkiriveillä olisi:

Rivi 1 - 23.12.2013, 29.12.2013, 16.12.2013

Rivi 2 – 30.12.2013, 5.1.2014, 23.12.2013

### 5.3.3 Työtuntien haku ja viikkojen jakaminen

”DIM_TIME_DAY day_count”-Step yhdistää Steeri Oy:n tietovaraston taulut DIM_TIME_DAY ja työtuntitaulun haettujen päivämäärien avulla ja laskee mones arkipäivä on menossa kyseisellä viikolla ja palauttaa arvon day_count-muuttujaan. Stepissä ”Join DIM_USER to FACT_EFFORT” yhdistetään työntekijöiden taulu työtuntitauluun. Tulojoukossa on tässä vaiheessa työntekijän id-numero, hänen tekemät työtunnit tunteina ja minuutteina ja lasketut saldot. Saldon laskentaan käytetään hyväksi day_count-muuttujaa. Tämän muuttujan avulla työnteki-

jän työtunneista vähennetään day_count-arvo kertaa 7,5 tuntia, joka on normaalin työpäivän verran. Tästä seuraa, että työntekijä, joka on tehnyt maanantaina normaalin työpäivän, saa viikon saldoksi tasan 0, joka on oikein. Koska päivämäärärivejä oli haettu kaksi kappaletta, jokaiselle työntekijälle tulee kaksi riviä, jossa on kuluvan viikon työtunnit ja kyseisen viikon saldo sekä edeltävän viikon työtunnit ja kyseisen viikon saldo.

Seuraavat kolme Steppiä valmistavat integraation rivit tauluunsyöttöä varten. ”Get ETL time”-Step hakee palvelimen kellonajan ja ”Add ETL label”-Step luo riveille tekstin, joka kertoo mistä integraatiosta on kyse. ”Filter by Datediff”-Step jakaa rivit sen mukaan, onko kyseessä kuluva viikko vai edeltävä viikko. Tämä onnistuu riveille alussa määritellystä datediff-arvosta, joka oli joko 0 tai 7. Työntekijän rivi 7 on edeltävä viikko (Kuvassa 7 menee suodattimesta vasemmalle) ja rivi 0 on kuluva viikko.

### 5.3.4 Kokonaissaldon laskeminen ja tietokannan päivitys

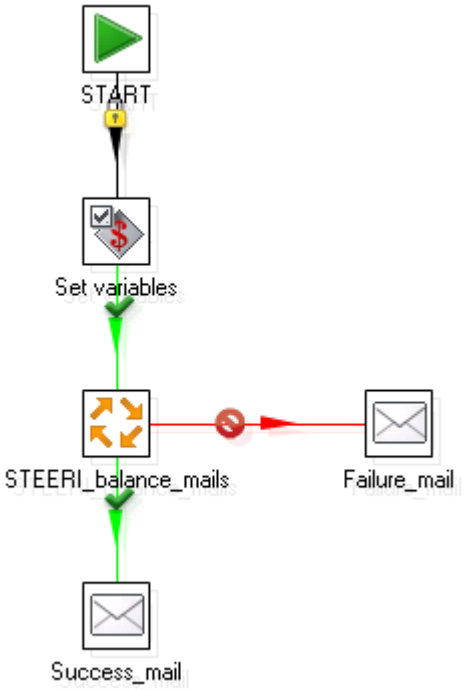
Suodatuksen jälkeen riveillä on siis työntekijän id-numero, viikon ensimmäisen ja viimeisen päivän päivämäärä, kyseistä viikkoa edeltävän viikon ensimmäisen päivän päivämäärä, kyseisen viikon työtunnit tunteina ja minuutteina, viikon saldo sekä integraation aika- ja lähdeleimat. Datasta puuttuu tässä vaiheessa vielä kokonaissaldo. Kuluvan viikon puolella (kuvassa 7 suodattimen oikea puoli) ensimmäinen Steppi odottaa ainoastaan sitä, että suodattimen vasen puoli valmistuu ensin. Tämä tehtiin sitä varten, koska ”FACT_BALANCE count”-Step käyttää edellisen viikon dataa. Jos tuota odotus-Steppiä ei olisi, integraatio voisi kaatua satunnaisesti, jos se sattuisi käsittelemään ensin työntekijän nykyisen viikon ja sitten vasta edellisen viikon.

”FACT_BALANCE”-Step hakee saldotaulusta viikkoriville edeltävän viikon kokonaissaldon ja saldokorjauksen saldotaulusta. Näitä tietoja taas hyödynnetään ”Calculate balance”-Stepissä. Tämä Steppi laskee viikon saldon, kokonaissaldon ja saldokorjauksen yhteen, josta tulee tämän viikon kokonaissaldo. Tämän prosessin jälkeen integraatio on laskenut työntekijälle kaikki tarvittavat työtunti- ja saldoarvot, jotka se päivittää ”Insert / Update”-Stepissä saldotauluun käyttäen

avaimina työntekijän id-numeroa ja viikon alkamispäivämäärää, sillä nämä kentät yksilöivät saldotaulun rivin.

### 5.4 Sähköpostitusintegraatioiden kehitys

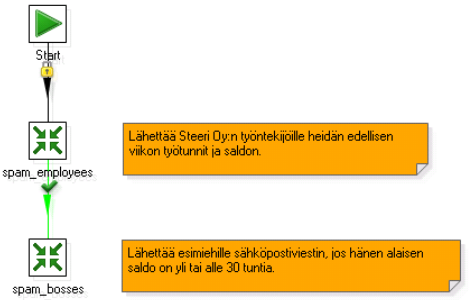
Toinen osa saldojärjestelmän integraatioita oli sähköpostitusintegraatio, joka sisältäisi työntekijöiden työtuntien ja saldojen sähköpostituksen sekä esimiesten sähköpostin, joka sisältäisi niiden työntekijöiden saldot, jotka ovat menneet yli tai alle 30 tuntia. Sähköpostitusintegraatio ajastettiin meidän integraatioseurantajärjestelmään, sillä tästä tehtiin oma projekti. Kuvassa 8 on sähköpostitusintegraation pääprojekti, jossa määritetään sähköpostitukseen tarvittavat muuttujat (kohdassa Set variables), käynnistetään saldosähköposti-integraatio ja lähetään lopuksi sähköposti integraatioseurantajärjestelmään, jos sähköpostitusintegraatio on mennyt läpi tai jos se on kaatunut ajon aikana. Jos integraatio on kaatunut, integraatioseurantajärjestelmä lähettää sähköpostia BI-tiimin jäsenille, jotka taas reagoivat tähän asianmukaisella tavalla.




*Kuva 8. Sähköpostitusintegraation ylin taso (Kuvakaappaus)*

”Steeri_balance_mails” on sähköpostitusintegraation varsinainen Jobi. Kuvasta 9 huomaa, että tämä integraatio sisältää kaksi Transformaatiota, jossa toisessa

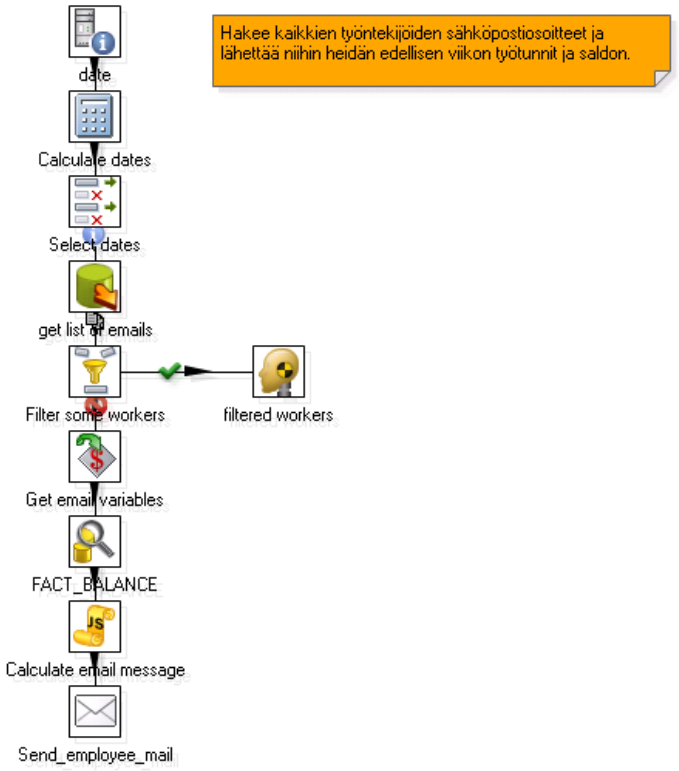
suoritetaan sähköpostin lähetys työntekijöille ja toisessa sähköpostin lähetys esimiehille.




*Kuva 9. Sähköpostitusintegraation toiseksi ylin taso (Kuvakaappaus)*

### 5.4.1 Työntekijöiden sähköpostitusintegraatio

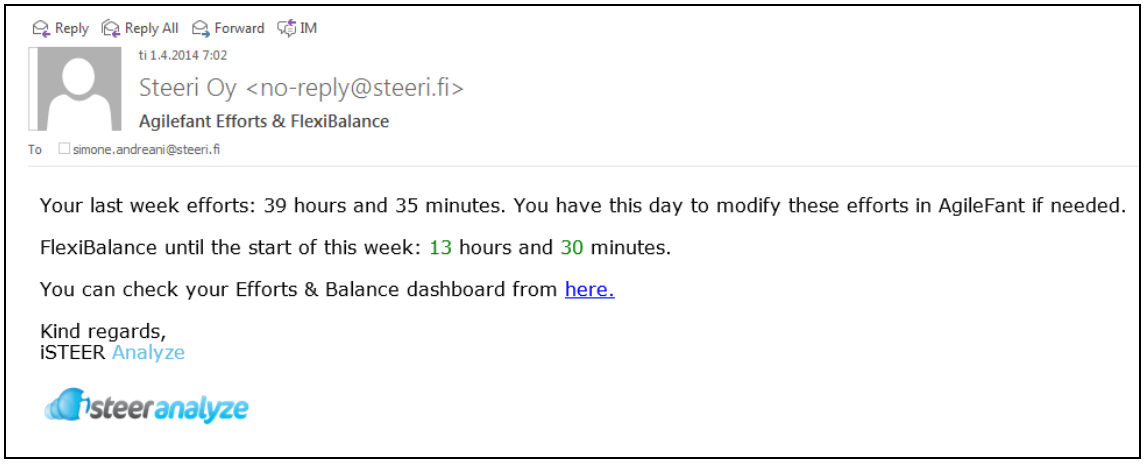
Ensimmäinen Transformaatio, jonka tein, oli työntekijöiden sähköpostitus. Tämä integraatio näkyy Kuvassa 10.




*Kuva 10. Työntekijöiden sähköpostitusintegraatio (Kuvakaappaus)*

Tämä integraatio oli sähköpostitusintegraatioista suoraviivaisempi. Mitä tässä integraatiossa tapahtuu, on melko selkeää. Ensimmäisissä kolmessa Stepissä haetaan käynnistyshetken ensimmäinen viikonpäivä ja viimeinen viikonpäivä, ei muuta. Seuraavaksi haetaan lista Steerin kaikista aktiivisista työntekijöistä (en voi valitettavasti näyttää SQL:ää) ja suodatetaan sähköpostituslistalta tuntityöntekijät. Tämä suodatus tapahtuu kohdassa ”Filter some workers” ja nämä työntekijät suodatetaan tässä Stepissä käsin, sillä tietovarastossa ei ollut tietoa tuntityöntekijöistä.

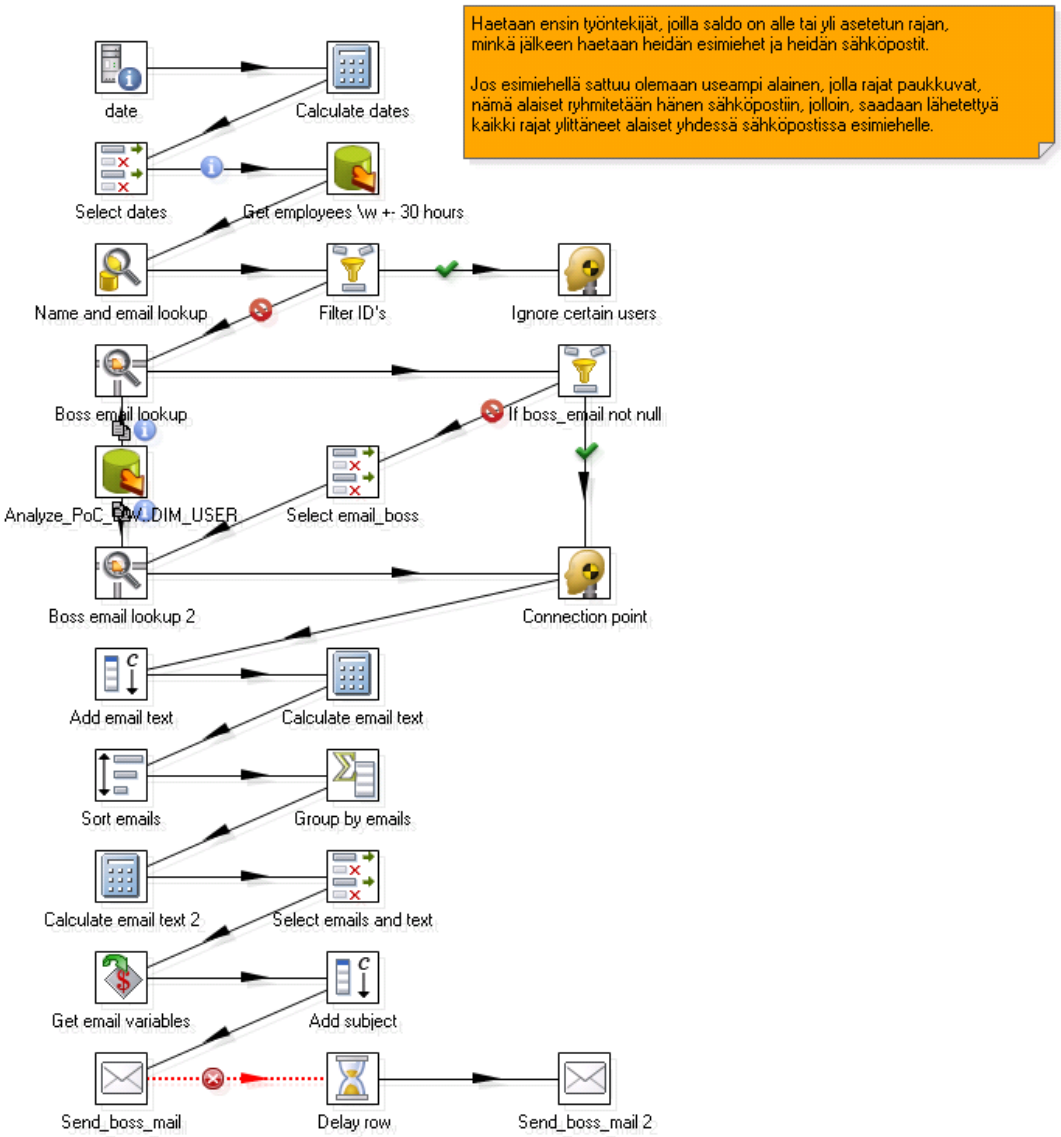
”Get email variables”-Step hakee integraation ylimmältä tasolta sähköpostitukseen tarvittavat muuttujat. ”FACT_BALANCE”-Step hakee saldotaulusta työntekijöille saldon ensimmäisten kolmen Stepin antaman päivämäärän perusteella. Tämän jälkeen tulee Steppi nimeltä ”Calculate email message”, joka on minun kirjoittamaa ”Custom Javascript”-Step. Tämä Steppi muokkaa datan haluttuun sähköpostimuotoon. Tästä esimerkki näkyy Kuvassa 11, jossa on minun oma saldoni maaliskuun viimeiseltä viikolta, 2014.




*Kuva 11. Sähköpostitusintegraation työntekijälle lähtevä sähköposti*

### 5.4.2 Esimiesten sähköpostitusintegraatio

Esimiesten sähköpostitusintegraatio oli hieman haasteellisempi verrattuna työntekijöiden vastaavaan. Kuvassa 12 on esimiesten sähköpostitusintegraatio. Tässä integraatiossa oli haasteena saada esimiehelle, jolla oli useampi ylittävä työntekijä, lähtemään ainoastaan yksi sähköposti. Tässä integraatiossa tuli käytettyä hieman enemmän Steppejä, jotta saisin toivotun lopputuloksen.




*Kuva 12. Esimiesten sähköpostitusintegraatio (Kuvakaappaus)*

Alku tässä integraatiossa on hyvin samanlainen kuin työntekijöiden sähköpostituksessa, sillä tässäkin määritellään päivämäärä, jolta saldo haetaan. Tästä eteenpäin integraatio eroaa kuitenkin huomattavasti. Saldotaulusta haetaan kaikki työntekijät, joiden juokseva saldo on yli tai alle 30 tuntia. Tämä tehdään ”Table Input”-Stepillä, johon on kirjoitettu SQL:ää. Where-ehdossa on määritelty tuo 30 tunnin raja. Tämän jälkeen integraatio hakee näiden työntekijöiden nimet ja sähköpostit Steerin tietovarastosta ja suodattaa tuntityöntekijät pois. Tämän jälkeen on olemassa lista kaikista työntekijöistä, jotka ylittävät tai alittavat 30 tunnin saldorajan, heidän nimistään ja sähköposteistaan.

Näihin työntekijöihin yhdistetään heidän esimies kahdella eri tavalla. Ensin esimies yritetään liittää riviin käyttämällä ”Stream Lookup”-Steppiä, jossa haetaan Steerin henkilötaulusta heidän esimiehen sähköpostiosoite. Jos tämä Steppi ei palauta esimiehen sähköpostia, toinen ”Stream Lookup”-Step hakee sen tietovaraston toisesta kentästä varmuuden vuoksi. Seuraavat stepit lisäävät riveille sähköpostiviestin, ryhmittävät saman esimiehen alle sen alaiset ja lähettää sähköpostit eteenpäin.

# 6 Pentaho BI-palvelimen asennus

Pentaho BI-palvelimen asennus ja konfigurointi oli selkeää. BI-palvelimen saa ladattua .zip-tiedostona Pentahon sivujen kautta ilmaiseksi, jonka jälkeen sen voi purkaa haluamaansa kansioon. Steerissä nämä palvelimet asennetaan ennalta sovitulla tavalla, jotta niitä olisi helpompi ylläpitää kenen tahansa toimesta. Huomioitavaa on, että Steerin palvelimet, joihin BI-palvelimet asennetaan, ovat suurimmaksi osaksi Linux-palvelimia, kuten tässä asennuskerrassa.

## 6.1 BI-palvelimen asennus

Linux-palvelimeen oli tehtävä ensin BI-palvelimelle kansio ”/opt/”-kansioon. Steerissä oli päätetty, että nämä nimetään ”/pentaho_[versio_numero]_[projekti]/”. Näin tiedetään suoraan palvelimen versio ja kenen käytössä se on. Tässä tapauksessa tein kansion, jonka nimi oli ”/pentaho_4.8_analyze/”. Tämän jälkeen kopioin BI-palvelimen kansioihin ”/biserver-ce/tomcat/webapps/pentaho/WEB-INF/lib/” ja ”/administration-console/jdbc” tiedostot ”sqljdbc4.jar” ja ”mysql-connector-java-5.1.17.jar”, jotta BI-palvelin tukisi Microsoft SQL Server sekä MySQL datayhteyksiä. ”/biserver-ce/”-kansio on varsinaisen BI-palvelimen alainen kansio, ”/administration-console/” on BI-palvelimen käyttäjienhallintapalvelin. Jos näitä .jar-tiedostoja ei kopioi näihin kansioihin, BI-palvelin ei saa yhteyttä kyseisiin tietokantoihin. Huomioitavaa on, että tässä opinnäytetyössä tiedosto ”mysql-connector-java-5.1.17.jar” oli käytännössä turha kopioida, mutta kyseinen BI-palvelin tuli muuhunkin käyttöön, jolloin tämä tiedosto oli pakollinen.

Tämän jälkeen kansiolle on annettava oikeanlaiset käyttöoikeudet eri käyttäjille, jotta palvelinta ei voisi käynnistää kuka tahansa. Tähän oli luotu käyttäjä ”pentaho”, jolle annettiin omistusoikeudet BI-palvelimen kansioon. Tämä tapahtui komennolla ”chown -R pentaho:pentaho /opt/pentaho_4.8_analyze”. Tämän komennon jälkeen pentaho-käyttäjä omisti kansion, mutta muilta käyttäjiltä poistettiin oikeudet. Tämä tapahtui komennolla ”chmod -R 755 /opt/pentaho_4.8_analyze/”. BI-palvelimen kansioissa on palvelimen käynnistystiedosto, nimeltään start-pentaho.sh. Tätä tiedostoa ei saanut käynnistää kukaan muu kuin pentaho-käyttäjä. Komennolla ”chmod 700 start-pentaho.sh” asetettiin käynnistysoikeudet ainoastaan omistajalle, eli pentaho-käyttäjälle.

### 6.2 BI-palvelimen konfigurointi

Tässä luvussa jätetään joitakin konfiguraatioita kertomatta, sillä BI palvelin on kirjoitushetkellä vielä tuotantokäytössä. BI-palvelin on muunneltavissa alusta loppuun, jos tietämystä riittää. Tässä tapauksessa riitti, että saisin luotua tietoturvallisen BI-palvelimen. Tämä tarkoitti, että eräitä muutoksia oli tehtävä Pentahon BI-palvelimen konfiguraatiotiedostoihin. Ensimmäiseksi vaihdettiin publisher-salasana publisher-config.xml-tiedostosta, joka löytyi ”/biserver-ce/pentaho-solutions/system/”-polusta. Salasana löytyi ”<publisher-password>”-tagien sisältä.

BI-palvelin haluttiin toimivan HTTPS:n kautta, joten BI palvelin täytyi konfiguroida käyttämään SSL-salausta. Ensimmäiseksi kopioin Steerin keystore-tiedosto Linux-palvelimelle, jotta BI palvelin voisi käyttää tätä SSL-salauksessa. Tämän jälkeen BI palvelimen SSL-asetuksia muutettiin. Nämä asetukset löytyivät kansiosta ”biserver-ce\tomcat\conf\”, tiedostosta server.xml. Oletusprotokolla on HTTP. Tiedostosta kommentoitiin seuraavat rivit:

*<Connector URIEncoding="UTF-8" port="8080" protocol="HTTP/1.1"*
*connectionTimeout="20000"*
*redirectPort="8443" />*

ja poistettiin kommentit seuraavilta riveiltä:

*<Connector URIEncoding="UTF-8" port="8443" protocol="HTTP/1.1"*
*SSLEnabled="true"*

*maxThreads="150" scheme="https" secure="true"*
*clientAuth="false" sslProtocol="TLS" />*

Lisäksi SSL-yhteyden sisään lisättiin keystore-tiedoston polku ja salasana. Yhteyksiä ei tarvinnut tällä kertaa enempää muunnella, mutta jos samalla palvelimella olisi ollut käynnissä useampi BI-palvelin, olisin joutunut muuttamaan porttia, jossa palvelin olisi käynnissä, esimerkiksi portista 8443 porttiin 8444. Tämä on yleensä tarpeellista palvelimilla, joissa pyörii useamman asiakkaan omat BI-palvelimet. Protokollamuunnoksen jälkeen Linux-palvelimen palomuuriin tuli lisätä kyseinen portti hyväksyttyyn liikenteeseen. Palomuurin asetukset pystyi tarkistamaan komennolla ”ufw status” ja lisäämään tarvittava portti palomuurin listaan komennolla ”ufw allow 8443/tcp”.

BI-palvelin itsessään tukee aluksi monta tiedostotyyppiä, mutta lisäosien asennuksen jälkeen muutamat tiedostotyypit tulee lisätä käsin sallittuihin tiedostotyyppeihin. Tämä onnistui lisäämällä kansiossa ”/biserver-ce/pentaho-solutions/system/” olevaan tiedostoon pentaho.xml halutut tiedostotyypit ”<acl-files>”-tagien sisään. Oletusarvoisesti sieltä löytyi xaction, url, prpt, prpti, xdash ja xcdf-tiedostotyypit, mutta mittaristot vaativat, että tuohon listaukseen lisättiin cdfde-tiedostotyyppi. Jos tätä lisäystä ei käy tekemässä, BI-palvelin ei suostu avaamaan näitä tiedostoja.

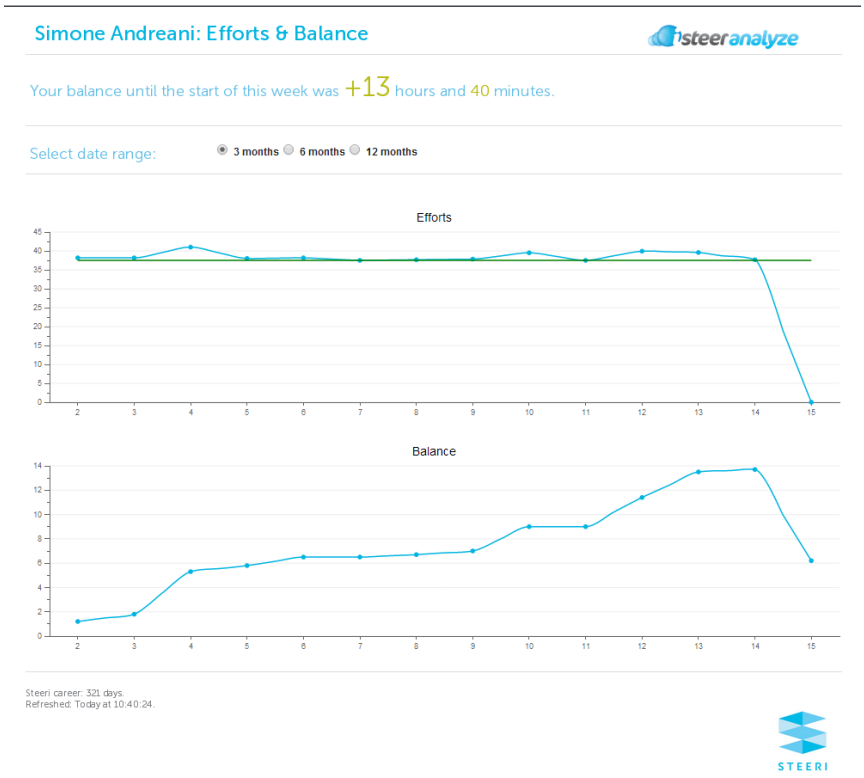
Näiden asetusten jälkeen Pentaho BI palvelimen pystyi käynnistämään start-pentaho.sh-tiedoston kautta ja asentamaan tarvittavat lisäosat, mm. CDE Dashboardsin (Kuva 3).

# 7 CDE-mittariston kehitys

BI-palvelimen asennus ja konfigurointi ei näy käytännössä loppukäyttäjälle lainkaan. Palvelin asennettiin ainoastaan sen takia, että työntekijöille saataisiin tehtyä mittaristo, jonka he voisivat aukaista Steerin asiakkuudenhallintajärjestelmän kautta. Steerin CRM-järjestelmä oli opinnäytetyötä tehdessä Steerin muokkaama Force.com-järjestelmä, joka on Salesforce.com-järjestelmän muokattavampi versio. Salesforce.com on täysin selainpohjainen SaaS-järjestelmä. Työntekijän ei

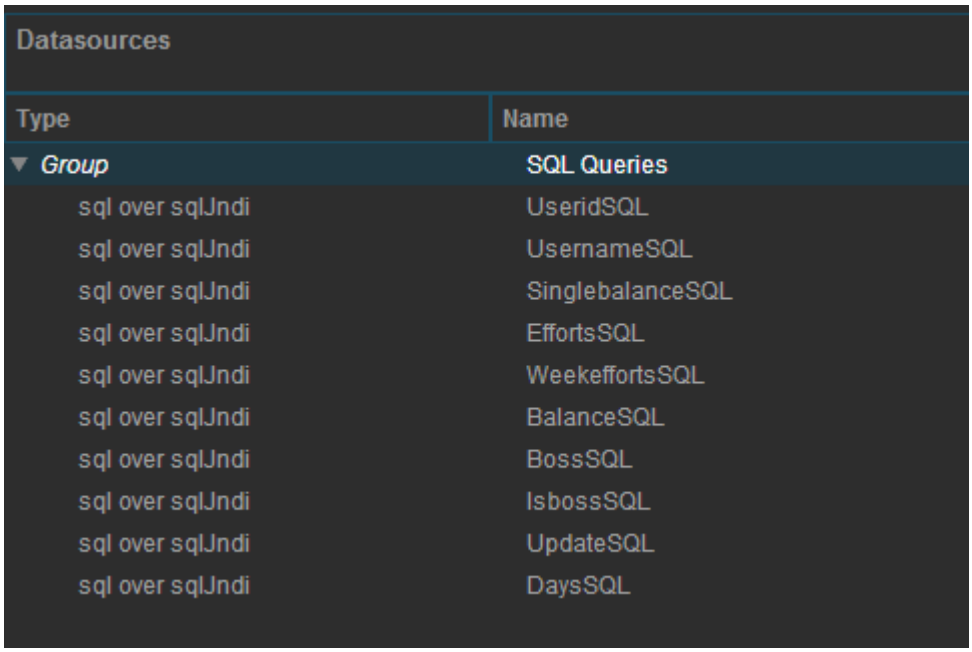
tarvitse kirjautua erikseen BI-palvelimeen, sillä käyttäjätunnukset saadaan siirrettyä osoiterivin kautta linkin mukana. Linkin parametrinsiirtoon tarvitsin hieman Kristianin, tiimikaverini, apua, sillä en ollut ennen siirtänyt SF:n dataa parametrissa osoiterivin kautta.

Kuvassa 13 näkyy lopullinen mittaristo, jonka kautta työntekijät näkevät suoraan saldonsa ylimmältä riviltä. Jos saldossa olisi jotakin hämminkiä, viivakaavioista näkyisi suoraan, minkä päivän kohdalla tämä virhe olisi tapahtunut. Yleensä saldon heittely selittyy sillä, että työntekijä ei ole merkinnyt tunteja tai on merkinnyt ne sen verran myöhään, että saldolaskentaintegraatio on ehtinyt lukita tämän viikon tauluun. Työtuntien merkitsemättä jättäminen näkyy notkahduksena molemmissa viivakaavioissa samalla viikolla. Työtuntien myöhässä merkitseminen on vaikeampi ymmärtää, sillä ylempi kaavio näyttää normaalilta, mutta saldokaavio notkahtaa alaspäin. Valintapainikkeet antavat mahdollisuuden valita kaavioiden aikajanan pituuden kolmen, kuuden ja kahdentoista kuukauden väliltä.



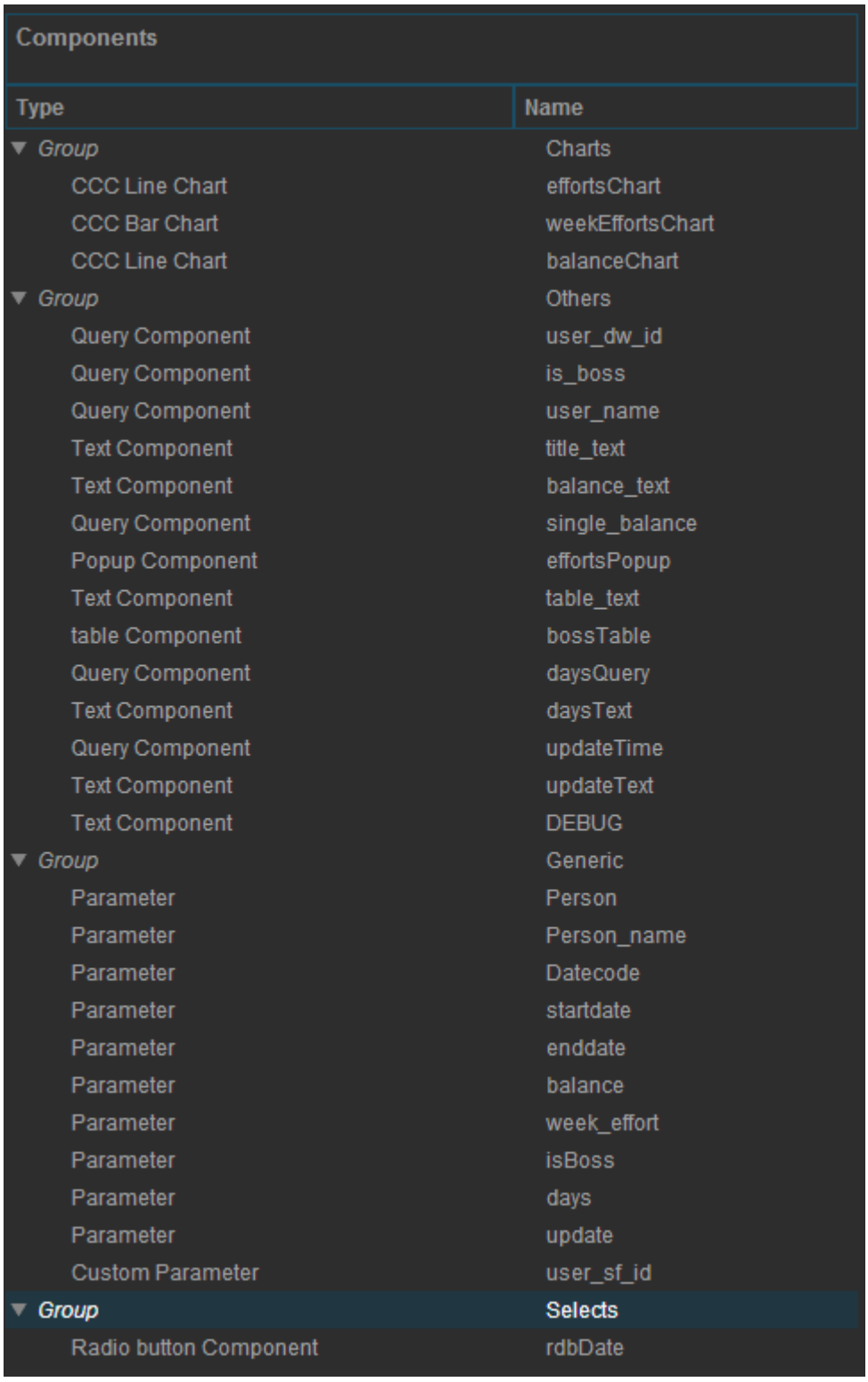

*Kuva 13. Lopullinen mittaristo (Kuvakaappaus)*

Kokonaisuudessaan mittaristo ei ole niin yksinkertainen kuin ulospäin näyttäisi olevan. Mittariston tekoon on käytetty kymmentä eri SQL-kyselyä, jotta mittaristoon saisi kaiken tarvittavan datan. Osa kyselyistä palauttaa arvoja parametreille, eivätkä ne näy mittaristossa. Kyselyt näkyvät kuvassa 14.




*Kuva 14. Mittariston SQL-kyselyt (Kuvakaappaus, CDE)*

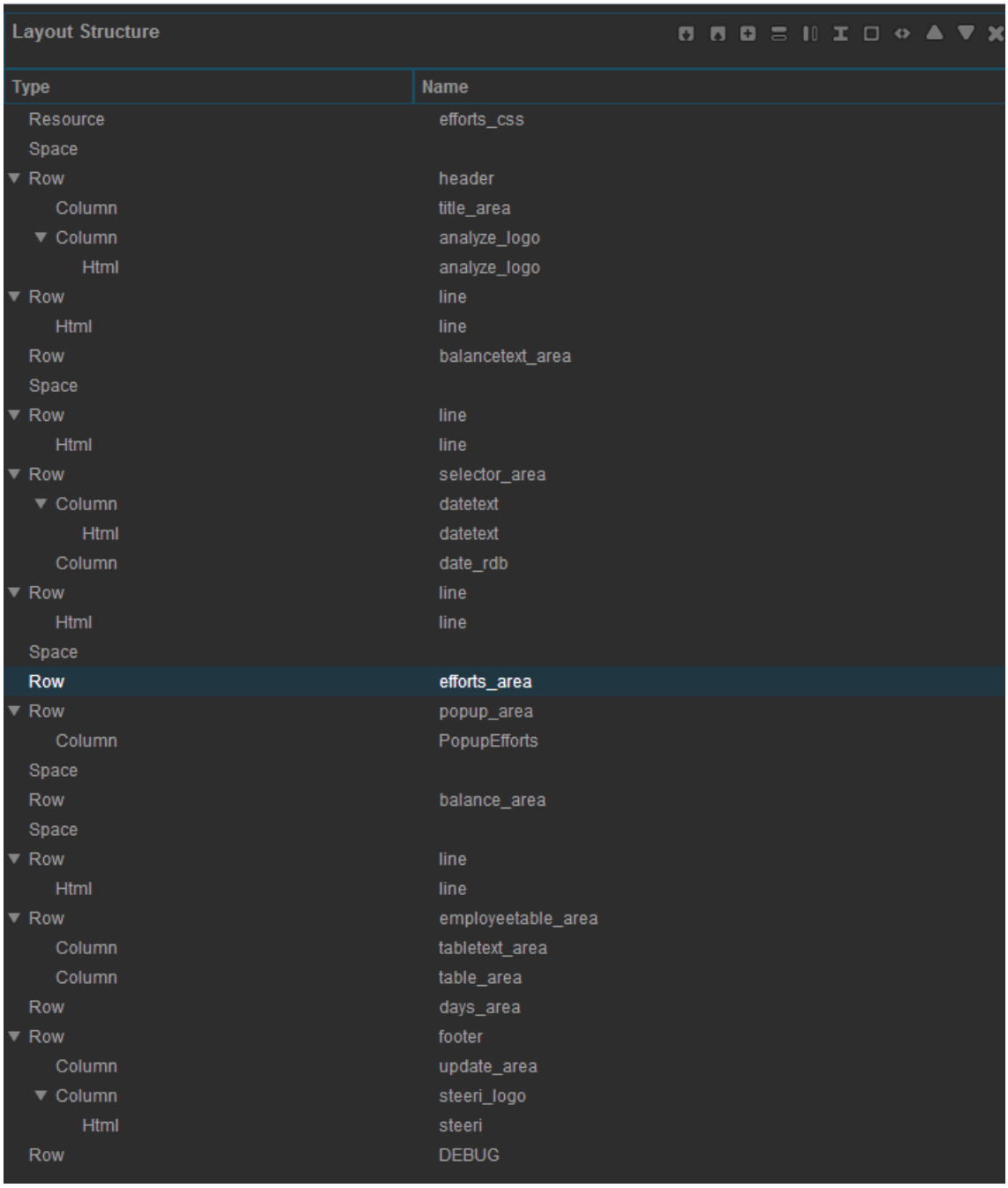
Kyselyiden tuottama data täytyy liittää joihinkin komponentteihin, jotta niitä voidaan käyttää mittaristossa. Kuvassa 15 näkyy kaikki käyttämäni komponentit tässä mittaristossa. Näistä komponenteista vajaa puolet näkyvät mittaristolla, sillä osaa käytetään siirtämään dataa SQL-kyselyille parametrien muodossa tai osaa siirtämään dataa komponentilta toiselle. Mittaristolla näkyviä komponentteja ovat CCC Line ja Bar Chartit, eli kolme ensimmäistä komponenttia, kaikki tekstikomponentit sekä alin komponentti, valintapainikekomponentti. Ei riittänyt vielä, että olin tehnyt kaikki komponentit valmiiksi ja tehnyt niille SQL-kyselyt, joiden avulla niihin saisi dataa, vaan nämä komponentit sijoitetaan vielä mittaristolle. Tätä varten mittariston kehityksessä on oma osio, Layout.




*Kuva 15. Mittariston komponentit (Kuvakaappaus, CDE)*

Komponentteja voi olla niin monta kuin halutaan, mutta nämä eivät tule mittaristoon, ennen kuin ne on lisättynä Layout-osioon. Kuvassa 16 on mittariston Layout-näkymä. Ensimmäisellä rivillä voidaan määrittää mittaristo käyttämään jo-

takin tiettyä CSS-tiedostoa, joka muotoilee mittariston ennalta määritellyillä tyyleillä. Tässä tapauksessa Steeri Oy:llä oli oma CSS-tiedosto, jonka kopioin BI-palvelimelle ja nimesin uudelleen tätä projektia varten. Mittaristolle komponentit tulevat näkyviin siinä järjestyksessä, kuin ne on lisätty Layout-näkymään. Layout-näkymä toimii siten, että siihen luodaan rivi ja kolumni, jotka nimetään halutulla tavalla. Kuvan 16 ensimmäinen kolumni on nimetty title_area:ksi. Tämä tulee käydä määrittelemässä tähän kolumniin tulevaan komponenttiin.




*Kuva 16. Mittariston Layout-näkymä (Kuvakaappaus, CDE)*

### 7.1 Otsikon ja saldorivin teko

Ensimmäisenä mittaristosta tein valmiiksi yläotsikon ja saldorivin (ks. Kuva 13). Otsikossa näkyy kenen mittaristosta on kyse ja tämä tulee automaattisesti SF:sta. Käytin tässä osoiterivin kautta siirrettyä parametria, joka sisälsi työntekijän id-arvon. Tämä arvo löytyi Steeri Oy:n tietovarastosta, joten pystyin hakemaan tämän avulla SQL-kyselyllä työntekijän nimen. Tähän nimen hakuun käytin kahta parametria (ks. Kuva 15), user_sf_id ja Person_name. Näitä parametreja käytin kahdessa komponentissa saadakseni henkilön nimen näkymään, user_name- ja title_text-komponenteissa. user_sf_id-parametri poimii selaimen osoiterivistä työntekijän SF id:n, jonka jälkeen tämä yksilöivä id on käytössä tämän parametrin välityksellä koko mittaristossa. Tämän avulla user_name-komponentti pystyy hakemaan Steerin tietovarastosta työntekijän nimen Person_Name-parametriin käyttäen UsernameSQL-kyselyä (kts. kuva 14). title_text-tekstikomponentti luo tekstin, jossa se käyttää Person_Name-parametria lisätäkseen tähän tekstiin työntekijän nimen. Tekstikomponentti on lisätty Layout-näkymässä kolumniin title_area, joka on header-rivin ensimmäinen kolumni. Toiseen kolumniin on haettuna Steerin logo BI-palvelimen kansioista käyttäen HTML-koodia.

Mittariston otsikkorivin alla on saldorivi, josta työntekijä näkee suoraan, paljonko hänellä on plussaa tai miinusta edellisen viikon loppuun mennessä (ks. Kuva 13). Tämä on luotu käyttäen SF:n id:tä, mutta id:n avulla haetaan työntekijän tietovaraston id-arvo, sillä saldotaulussa on käytetty tietovaraston id-arvoa, ei SF:n id-arvoa. Tämä on haettu käyttäen user_dw_id-komponenttia, joka hakee Person-parametriin työntekijän tietovaraston id:n. Tämä tieto haetaan UseridSQL-kyselyllä. Single_balance-komponentti hakee tämän id:n avulla saldolaskentaintegraation taulusta työntekijän viimeisimmän kokonaisen viikon saldorivin juoksevan saldon ja sijoittaa tämän saldon balance-parametriin. Balance_text-komponentti luo tekstin, jossa käyttää balance-parametrin arvoa. Värien lisäys on tehty käyttäen if-lauseita, jotka tarkastavat, onko balance-parametri negatiivinen vai positiivinen; positiivinen on kirjoitettu vihreällä värillä, negatiivinen punaisella. Jos saldo sattuu olemaan nolla, saldo näkyy sinisellä värillä. balance_text-komponentti löytyy kolumnista balancetext_area Layout-näkymässä.

## 7.2 Valintapainikkeiden luonti

Halusin, että mittariston kaavioiden aikajanoja pystyisi muuttamaan, joten tein valintapainikkeet mittaristoon. Päädyin kolmeen, kuuteen ja 12 kuukauteen, sillä nämä aikavälit näyttivät sopivan parhaiten kaavioihin. rdbDate-komponentti on valintapainikekomponentti ja se vie parametreihin startdate ja enddate aloitus- ja loppupäivämäärät. Päivämäärät eivät tule tietovaraston kautta, vaan ne lasketaan mittaristossa sen mukaan, mitä painiketta on painettu. Layout-näkymässä valintapainikekomponentti on sijoitettu kolumniin date_rdb, joka on selector_area-rivillä. Ennen valintapainikkeita löytyy datetext-kolumni, jossa on määritetty kiinteä teksti, ”Select date range: ”.

## 7.3 Työtuntikaavio

Työtuntikaavion komponentti on effortsChart, joka on viivakaavio. Tämä komponentti käyttää EffortsSQL-kyselyä, joka hakee dataa tuntikirjausjärjestelmän tietovaraston taulusta. Tämä taulu on eri taulu, kuin mistä saldot löytyvät. EffortsSQL-kysely käyttää kolmea parametria, user_dw_id, startdate ja enddate. Viivakaavio tunnistaa, jos jokin kolmesta parametrista muuttuu, sillä ne on määritetty komponentin Listener-valikossa. Toisin sanoen, jos valintapainikkeista valitaan kuusi kuukautta, niin startdate- ja enddate -parametrit saavat uudet arvot. effortsChart tunnistaa, että parametrit ovat muuttuneet ja viivakaavio hakee SQL-kyselyn kautta uuden datan. Komponentti on sijoitettuna efforts_area-riville.

Ulkoasua on hieman muokattu. Pystyakselin suuntaiset apuviivat on poistettu, koska ruudulle tuli liikaa viivoja ja ne sekoittivat käyttäjiä. Kaavioon on luotu kiinteä vihreä viiva, joka kulkee 37,5 tunnin kohdalla. Tämä kuvastaa normaalia tuntiarvoa yhdelle viikolle. Varsinaisen viivan väri on muutettu Steeri Oy:n viralliseen siniseen, jota käytetään mm. Steeri Oy:n CSS-tiedostossa. Viivan pisteiden suuruutta on pienennetty hieman, sillä oletuskoko korosti niitä turhan paljon. Kaavion ympäriltä on poistettu reunukset, sillä reunukset tulivat liikaa esille. Vaaka-akselin apuviivoja on hieman häivytetty, jotta ne eivät korostuisi niin paljoa. Pääviivan kuuluu näkyä ensimmäisenä, kun kaaviota katsoo, ja näin se onnistui.

Pääviivan pisteitä voi painaa hiirellä, jolloin ruudulle ilmestyy pop-up-kaavio, joka on weekEffortsChart-komponentti. Tässä kaaviossa näkyy valitun viikon työtunnit

päivätasolla. Virheellisen kirjauksen huomatessaan työntekijä voi siis painaa effortsChart-komponentin pistettä ja tarkistaa millä päivällä on väärä kirjaus. Tätä tietoa ei kuitenkaan haluta normaalitilanteessa, joten siksi päädyin pop-up-tyyliseen ratkaisuun, jossa kaavio tulee esiin ainoastaan, kun sitä oikeasti tarvitaan. Tätä varten komponenttivalikossa näkyy effortsPopup-komponentti, joka aktivoituu effortsChart-komponentin pistettä painettaessa ja vie week_effort-parametriin painetun viikon viikkonumeron. Tämän jälkeen effortsPopup-komponentti kutsuu weekEffortsChart-komponenttia pop-up-ikkunaan. weekEffortsChart-komponentti taas käyttää WeekeffortsSQL-kyselyä, joka palauttaa painetun viikon työtunnit eriteltynä viikonpäiville, week_effort-parametrin avulla.

### 7.4 Saldokaavio

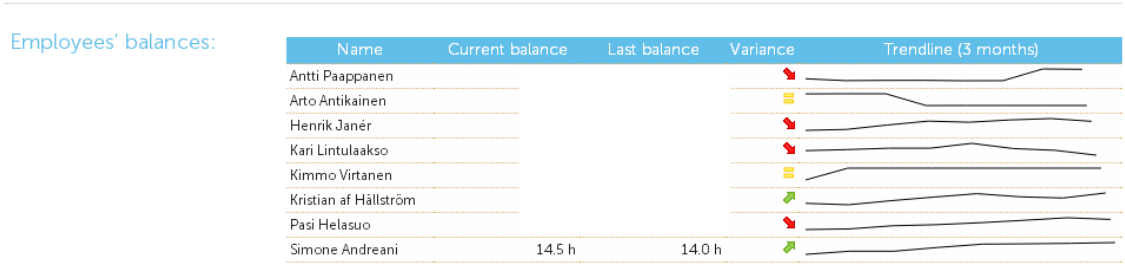
Saldot näkyvät balanceChart-komponentissa, joka on myös viivakaavio. Kyseinen komponentti hakee dataa BalanceSQL-kyselyn kautta, jossa parametreina ovat user_dw_id, startdate ja enddate. BalanceSQL-kysely käyttää siis samoja parametreja kuin EffortsSQL-kysely. Kysely palauttaa työntekijän viikkosaldon sen mukaan, mitä valintapainikkeista on valittuna. Jos valitaan eri ajanjakso, balanceChart-komponentti päivittyy samalla tavalla kuin effortsChart-komponentti. Koska molemmat komponentit käyttävät samoja parametreja, ajanjaksot ovat samat, jolloin kaavioissa pysyvät samat ajanjaksot, vaikka sitä muutettaisiin. balanceChart-komponentti on sijoitettuna suoraan effortsChart-komponentin alle mittaristossa, riville balance_area.

### 7.5 Esimiesnäkymän luonti

Esimiesnäkymän lisäsin melko loppuvaiheessa projektia. Tajusin, että voisin tehdä taulukon mittaristoon, jossa olisi alaisten nimet ja heidän saldonsa sekä saldon kehitys jonkinlaisena viivakaaviona. Ensimmäinen haaste oli saada mittaristo tarkistamaan, onko mittaristoa käyttävä työntekijä joidenkin esimies vai ei. Tämä tarkistus onnistui lopulta melko helposti, sillä Steeri Oy:n tietovarastossa on työntekijätaulu, jossa on ilmoitettu heidän esimiehen SF id.

Tein IsbossSQL-kyselyn, joka tarkistaa heti mittaristoon tullessa, onko kyseisen työntekijän SF id:tä tässä sarakkeessa kenenkään työntekijän kohdalla. Jos on,

niin IsbossSQL-kysely palauttaa alaisten lukumäärän count()-funktiolla parametriin isBoss. bossTable-komponentti kutsuu BossSQL-kyselyä, joka hakee kaikki hänen alaisensa ja heidän viimeisimmän saldonsa ja tekee tästä datasta taulukon. Aiemmin tehty tarkistus siitä, onko työntekijä esimies vai ei mahdollistaa sen, että esimiestaulukkoa ei piirretä mittaristolle ollenkaan, jos isBoss-parametri on yhtä suuri kuin nolla. Tämä tarkistus on tehty bossTable-komponentin alustusosioon. Kuvassa 16 näkyy mittariston lopullinen esimiesnäkymä, josta olen poistanut tiimikavereideni saldot.




*Kuva 17. CDE esimiesmittariston taulukko (Kuvakaappaus)*

## 7.6 Päivälaskurin ja footer-osion teko

Mittariston alin rivi on ns. footer-osio, jossa näkyy Steerin logo, milloin data on päivittynyt ja työntekijän ura Steerissä ensimmäisestä syötetystä tuntikirjauksesta lähtien. Lisäsin päivämittarin, koska ajattelin, että se voisi olla kiinnostava lisäys, jota ei välttämättä heti huomaisi. Viimeisessä versiossa mittarista Kuvassa 13 ei näy DEBUG-osiota, joka näkyy Layout-osiossa, sillä DEBUG-osiota käytettiin ainoastaan kehitysvaiheessa tarkastamaan, että parametrit käyttäytyvät oikein. Käytännössä DEBUG-osio ei ole kuin listaus parametrien arvoista. Steeri Oy:n logo on linkitetty HTML-koodilla kuten toinen kuva ylimmällä rivillä. Layout-näkymässä kyseinen logo löytyy footer-rivin steeri_logo-kolumnista. Kuva haetaan tähän kolumniin <ahref>-komennolla BI palvelimen kansiosta.

Datan viimeinen päivityshetki on tehty käyttämällä ensin updateTime-komponenttia, joka hakee UpdateSQL-kyselyllä tietokannasta aikaleiman rivin syöttö-/päivitysajasta. Tulos viedään update-parametriin, jota vastaavasti käytetään updateText-komponentissa osana tekstiä. ”Refreshed: ” osa tekstikomponentista on kiinteä, loppuosa tarkastetaan if-lauseella. Jos päivämäärä on samalta päivältä,

kuin mittariston ajopäivä, tekstiin kirjoitetaan ensin ”Today ” ja tämän jälkeen otetaan aikaleimasta pelkkä kellonaika ja liitetään osaksi tekstikomponenttia. Jos rivi on eri päivältä kuin ajohetken päivä, ”Refreshed: ” -tekstin perään tulee haettu päivämäärä kokonaisuudessaan. Tekstikomponentti on lisätty footer-osion ”update_area” -kolumniin.

Steerin työpäivälaskuri taas hakee tietokannasta työntekijän DaysSQL-kyselyn kautta erotuksen ajopäivän ja ensimmäisen syötetyn työtunnin välillä ja laittaa tuloksen parametriin ”days”. Tämän jälkeen tekstikomponentissa ”daysText” tehdään kiinteä teksti ”Steeri career: ”, lisätään days-parametrin arvo ja loppuun lisätään vielä kiinteä teksti ” days.”. Tekstikomponentti on liitetty Layout-näkymän riville ”days_area”.

# 8 Järjestelmän testaus

Ennen kuin otin järjestelmän käyttöön koko yritykselle, päätin ottaa muutaman työntekijän testiryhmään, jotta saisin testattua järjestelmän toimivuuden. Dataintegraatioiden osalta testiryhmän rajaus oli tosi helppoa, sillä minun ei tarvinnut muuta kuin rajata SQL-kyselyitä haluttujen työntekijöiden id-numeroilla. Ennen testiryhmän valitsemista, ajoin rutiinit kertaalleen vain omalla id-numerolla testatakseni, että rutiinit toimivat kuten niiden pitikin. Kaikki toimi hyvin, joten päätin, että testiryhmänä saisi toimia oma tiimini. Saldolaskentaintegraatio toimi alusta asti hyvin tuntikirjausjärjestelmän integraatioissa, eikä siihen tarvinnut tehdä muutoksia. Sähköpostitusrutiinit toimivat kanssa moitteetta.

Ainut ongelma integraatioiden kanssa syntyi, kun siirryimme testaamaan rutiineja koko yrityksen työntekijöillä. Käyttämämme SMTP-palvelin ei tukenut yli 50 sähköpostin lähetystä muutamassa sekunnissa ja SMTP-palvelin päätti katkaista integraation yhteydet siihen. Keskustelin työpaikan infra-tiimin kanssa hetken ja päätimme luoda gmail-tunnukset integraatiolle, jotta tätä ongelmaa ei syntyisi, jonka jälkeen ongelma korjaantui.

Järjestelmää ei tarvinnut jälkeenpäin testata, sillä integraatiotyökalu ei anna tehdä virheitä. Jos integraatioissa on jotakin vialla, se herjaa näistä eikä suostu käynnistymään. Työkalulle taas on aivan sama, onko työntekijöitä yksi vai onko

niitä sata, sillä jokainen rivi käy saman prosessin integraatiossa. Virheitä ei siis synny enempää, mitä enemmän rivejä menee integraatioiden läpi.

# 9 Yhteenveto ja pohdinta

Kokonaisuudessaan opiskelin laajan alueen, jotta pystyin toteuttamaan kaiken tarvittavan. Dataa haettiin paikasta A, käsiteltiin sopivaksi ja vietiin paikkaan B. Rakensin validin dataintegraation alusta loppuun itsenäisesti. Tämän lisäksi asensin ja konfiguroin toimivan BI-palvelimen Linux-ympäristöön, johon saa yhteyden ulkomaailmasta. Lopuksi kehitin täysin toimivan mittariston, josta työntekijät pystyvät seuraamaan syöttämiään työtunteja ja saldoansa. Olen käynyt läpi ison osan perus-BI-työskentelystä tämän opinnäytetyön aikana ja mikä parasta, sain luotua jotakin, joka on operatiivisessa käytössä.

Aikaa tämä on vienyt paljon, noin 200 tuntia (ilman opinnäytetyöraporttia) 2013 vuoden syyskuusta 2014 vuoden huhtikuuhun, mutta saamani palaute työntekijöiltä ja esimieheltäni on ollut pelkkää positiivista, joten voin olla ylpeä aikaansaannoksestani. Projekti oli onnistunut, sillä järjestelmä täyttää kaikki ennalta sovitut tarpeet ja sain jopa lisättyä ominaisuuksia mukaan projektiin. Alkuperäinen suunnitelma ei pitänyt mittariston kehitystä ollenkaan mukana, mutta se on ollut ahkerimmassa käytössä.

Koko järjestelmään on tähän päivään asti ainoastaan yksi kehitysehdotus ja sekin oikeastaan minulta. Järjestelmä ei tue tällä hetkellä saldojen korjausta oikein hyvin. Tämä oli osittain harkittua, sillä saldojen haluttiin lukittuvan ja että niitä ei enää muutettaisi siinä vaiheessa. Käytännössä tämä ei ole toiminut hyvin, vaan saldokorjauksia tulee viikoittain, johtuen ylityötunneista tai merkitsemättömistä työtunneista. Tämä ei ole ongelma, sillä saldolaskentaintegraatio osaa laskea korjaukset mukaan tietokannasta, mutta saldokorjausten tekeminen suoraan tietokantaan ei ole paras mahdollinen ratkaisu. Ratkaisu tähän haasteeseen on vielä suunnitteilla, mutta tulossa varmasti.

## Lähteet

af Hällström, K. 2014, Steeri Oy, Helsinki. Haastattelu aiheesta dataintegraatiot ja mittaristot 12.2.2014.

Boucher Ferguson, R. 2008, Salesforce.com Unveils Force.com Cloud Computing Architecture. http://www.eweek.com/c/a/Enterprise-Applications/Salesforce-com-Unveils-Forcecom-Cloud-Computing-Architecture/. Luettu 10.3.2014.

Casters, M., Bouman R. & van Dongen, J., 2010. Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration, 10-230.

Lenzerini, M. 2002. "Data Integration: A Theoretical Perspective", http://www.dis.uniroma1.it/~lenzerin/homepagine/talks/TutorialPODS02.pdf, 233–246. Luettu 29.3.2014.

Pentaho Community, 2014. http://community.pentaho.com/, käyty 4.5.2014.

Pochampalli, A. 2013. Installation of Pentaho BI Server Community Edition 4.8.0 in Windows 7 x.64, http://pentaho-bi-suite.blogspot.fi/2013/04/installation-of-pentaho-bi-server.html. Luettu 21.8.2013.

Pulvirenti, A.S. & Roldán M.C., 2011. Pentaho Data Integration 4 Cookbook, 30-57.

SAP BusinessObjects BI Suite Master Guide, 2014. http://service.sap.com/~sapidb/011000358700000488252013E/sbo41_master_en.pdf, luettu 15.4.2014.

Steeri Oy:n julkiset taloustiedot. Taloussanomat, 2013. http://yritys.taloussanomat.fi/y/steeri-oy/helsinki/1574488-9/. Luettu 30.3.2014.