LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

TIETOVARASTON MUODOSTAMINEN AUTOMAATTISESTI LÄHDETIETOKANNOISTA

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2012
Jaakko Reinman

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

REINMAN, JAAKKO: Tietovaraston muodostaminen automaattisesti lähdetietokannoista

Ohjelmistotekniikan opinnäytetyö, 45 sivua, 1 liitesivu

Kevät 2012

TIIVISTELMÄ

Tässä opinnäytetyössä tutkittiin data vault -tietovarastomallia ja sen mukaisen tietovaraston muodostamista automaattisesti useista eri lähdetietokannoista yhden sovelluksen avulla. Automatisoimalla toimitusprosessista rutiininomaisen osuuden voidaan pienentää tietovaraston pystyttämiseen käytettyä aikaa ja vähentää käsityössä syntyvien virheiden määrää.

Opinnäytetyön kohdeyrityksellä oli sovelluksesta valmiina proof of concept -versio, jolla data vault -ideaa oli kokeiltu käytännössä ja alettu kehittää eteenpäin. Työn päämääränä oli syventyä data vault -tietovarastointimalliin ja kirjoittaa PoC-toteutuksen arkkitehtuuri uusiksi siten, että tuloksena syntyy toimiva data vault -järjestelmä. Työn testaamista varten annettiin osakopio oikeasta tuotantokannasta, jolloin testidata oli realistista ja kehittäessä osattiin heti vastata oikealla tavalla annettuihin vaatimuksiin.

Työn kohdeyritys oli Logica Suomi Oy ja työpaikkana Lahden toimisto. Logica on yksi suurimpia IT-alan yrityksiä Pohjoismaissa ja myös maailmanlaajuisesti. Yrityksen toimialaa on IT-alan konsultointi, infrastruktuuriratkaisut, tietojärjestelmien integraatiot sekä IT- ja liiketoimintaprosessien ulkoistamispalvelut. Viime vuonna sen liikevaihto oli noin 4,5 miljardia euroa.

Opinnäytetyönä tehty data vaultin generointisovellus onnistui hyvin kaikki tavoitteet täyttäen ja peräti hieman etuajassa. Ohjelmiston toteuttamiseksi vaadittu teoriaan tutustuminen veikin yllättäen kolme viikkoa, koska työtä oli aikaa tehdä vain kolmesti viikossa opiskelujen takia. Teorian valjettua siirryttiin arkkitehtuurin suunnitteluun ja sitä kautta itse toteutukseen. Loppuaika kuluneesta ajasta käytettiin pääasiassa testaamiseen.

Sovelluksen valmistuttua oli pian jo uusi lista seuraavista tarvittavista ominaisuuksista kasassa jatkokehitystä varten. Uusia parannusehdotuksia tulee todennäköisesti ajan kuluessa lisää, kun sovellusta aletaan käyttää tietovarastojen toteutuksessa.

Asiasanat: tietovarasto, data vault, tietokannat

Lahti University of Applied Sciences
Degree Programme in Information Technology

REINMAN, JAAKKO:

Creating a data warehouse automatically from multiple source databases

Bachelor's Thesis in Software Engineering, 45 pages, 1 page of appendix

Spring 2012

ABSTRACT

The objective of this thesis was to study a data warehousing model called data vault and develop an application to form data vault storage automatically from multiple source databases. By making routine steps of the data warehouse delivery process automatic, less time can be spent building it and the number of mistakes can be lowered and human errors reduced.

The work was commissioned by Logica Suomi Oy and it was based on an already developed proof of concept. Preliminary tests of the idea had been conducted in practice and the decision about continuing the development of software was made recently. The work plan was to rewrite the proof of concept as fully designed software with modular architecture and to be able to build a data vault warehouse as the end product. For testing functionality of the software, sample of real production environment database was provided, which helped developing the proposed features greatly. All ideas developed were transformed into a generic form to work in any data warehouse.

The data vault generator software developed here succeeded well, filling all the placed objectives, and it was completed almost a month ahead of time. The studying and understanding of the concept took three weeks, which was longer than expected. Reasons for this might have been complicated theory and that the work was done in three days in a week. After grasping the theory architecture design and implementation was started. The latter half of the time was used almost completely for testing.

After completing the thesis, there was soon a list of new features desired from the software. Especially new user interface features or changes to them can be expected because extensive user testing was not conducted yet and there might be room for improvement.

Key words: data vault, data warehousing, data storage, databases

SISÄLLYS

1 JOHDANTO 1

2 TIETOVARASTOTEKNIIKKA 3

2.1 Yleistä 3

2.2 Historiasta 4

2.3 Tietovarastot nykyisin 7

2.4 Tietovaraston muodostaminen 8

2.5 ETL-prosessi 11

2.6 SSIS-lataajat 14

2.7 Metatiedon kuvaus 15

2.8 Data vaultin rakenne 16

3 TIETOVARASTON GENEROINTISOVELLUKSEN TOTEUTUS 22

3.1 Yleistä toteutuksesta 22

3.2 Vaatimukset DW-generointisovelluksen hyväksymiselle 23

3.3 Data warehouse -generointisovellus 24

3.4 Generointisovelluksen tekninen toiminta 29

4 YHTEENVETO 37

LÄHTEET 40

LIITTEET 41

1 JOHDANTO

Tietovarastointi (Data Warehousing, DW) ja liiketoimintatiedon hallinta (Business Intelligence, BI) ovat yksi nopeimmin kehittyviä tietotekniikan alueita. Yritysten toiminnan kautta muodostuneen tiedon määrän jyrkästi kasvettua ja nykyisten analysointi- ja raportointitarpeiden vaativuuden noustua ovat nykyiset operatiiviset perusjärjestelmät käyneet lähes riittämättömiksi. Suuria tietomassoja täytyy kuitenkin pystyä hallitsemaan, jotta niistä saadaan uutta informaatiota liiketoiminnan tilan selvittämiseksi ja toiminnan kehittämiseksi.

Tietovarastoinnissa tiedot eri järjestelmistä kopioidaan omaan, erilliseen tietokantaansa, josta jalostettua tietoa voidaan sitten helposti analysoida ja raportoida monilla eri työkaluilla. Tietovaraston rakentamiseen kuuluu menetelmiä ja tekniikkaa sekä projektin suunnittelua ja hallintaa. Data vault -mallinnusmenetelmä on tietovarastoinnin uusin teoria ja kokoelma tapoja, joilla tietovaraston tietoa tallennetaan ja käsitellään.

Työn kohdeyritys on Logica, joka on eurooppalainen IT-palveluyritys, ja se toimii yli 41 maassa. Yritys on suuri, sillä sen liikevaihto oli vuonna 2011 noin 4,5 miljardia euroa. Logican asiakkaita ovat kaiken kokoiset yritykset, eri valtioiden julkishallinnon organisaatiot ja muutamat yhdistykset. Yrityksen osaamista ovat IT-alan konsultointi, infrastruktuuriratkaisut, tietojärjestelmien integraatiot sekä IT- ja liiketoimintaprosessien ulkoistamispalvelut.

Logicalla on jo käytössä alun perin käsin muodostettu, toimiva tietovarasto, johon on tallennettu Logican ja asiakasyritysten liiketoimintatietoja. Sitä hyödynnetään liiketoiminnan päätöksenteossa, kun on voitu analysoida eri lähdejärjestelmien tietoja ristiin ja saatu kerätyn historian perusteella luotua ennusteita esimerkiksi myynnistä, tilauskannan koosta ja kuluista. Tähän asti uudet tietoa lataavat ajoketjut on tehty aikaa vievästi käsin, mutta koska ajoketjun rakenne on aina sama, on pohdittu tuon rutiininomaisen vaiheen automatisointia ohjelmallisesti. Ohjelmallinen latausketjujen luominen mahdollistaa myös tietovaraston nopean käyttöönoton ja siten tällaisen ratkaisun myymisen asiakkaille BI-ratkaisujen pohjaksi.

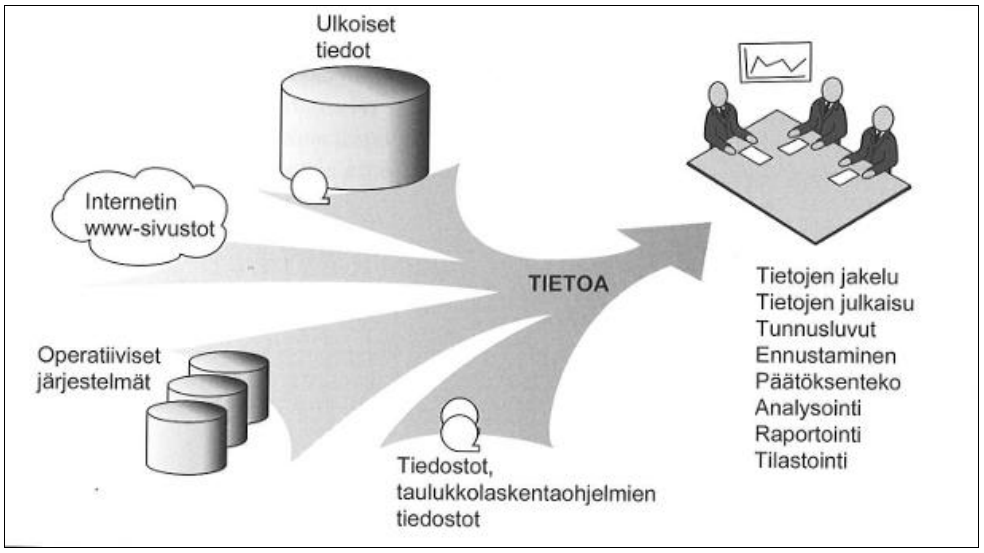
Logicalla oli tietovaraston luontiin Proof of Concept -toteutus (jäljempänä PoC), mutta sen luonteen takia ohjelman arkkitehtuuri oli suunniteltava ja toteutettava uudelleen. Työssä siis suunniteltiin uuden ohjelman arkkitehtuuri ja toteutettiin se hyödyntäen PoC-toteutuksen olemassa olevaa business-logiikan ohjelmakoodia. Lopputuloksena tulevan ohjelmakoodin haluttiin olevan helposti laajennettavissa ja ylläpidettävissä mahdollisen jatkokehityksen varalta, joten sovelluksen suunnittelussa käytettiin mahdollisimman modulaarista arkkitehtuuria. Aikatauluksi Logican puolelta oli sanottu, että valmista olisi oltava toukokuuhun 2011 mennessä. Sovelluksen suunnittelu ja toteutus aloitettiin tammikuussa 2011.

Asiakasvaatimuksissa edellytettiin myös sitä, että tietovaraston muodostamissovellus toteutettiin C#-kielellä ja käyttöliittymässä käytettiin Windows Presentation Foundation -rajapintaa. Sovellukseen toivottiin Microsoftin SQL-palvelintuen lisäksi tuki ODBC- eli Open Database Connect -tuelle, joka mahdollistaa kaikkien sitä rajapintaa tukevien tietokantapalvelimien käytön sovelluksessa.

2 TIETOVARASTOTEKNIIKKA

2.1 Yleistä

Yritysten ja julkishallinnon päättäjät sekä viranomaiset tarvitsevat monenlaista tietoa päätöksenteon tueksi. Myynnin kehitystä halutaan usein seurata monesta näkökulmasta, tarvitaan muun muassa erilaisia asiakasanalyysejä. Tilauskannan kokoa tarvitsee seurata, jotta osataan tarvittaessa palkata lisää tai vähentää henkilöstöä suurien vaihteluiden myötä tulojen tasaamiseksi. Henkilöstön suunnittelu edellyttää myös tarkkoja tietoja esimerkiksi henkilöiden lukumääristä, koulutuksesta ja työajoista pitkiltä aikaväleiltä. (Hovi, Koistinen & Hervonen 2009, XI.)



KUVIO 1. Tiedonkulku organisaatioissa (Hovi ym. 2009, 4)

Tarvittavista tiedoista on valtaosa olemassa organisaation omissa perusjärjestelmissä eli niiden tietokannoissa. Monissa yrityksissä on hankittu yksittäisiä valmisohjelmistoja tai laajoja ERP-järjestelmiä eli yritystason resurssienhallintajärjestelmiä (Enterprise Resource Planning). Joissakin toisissa yrityksissä puolestaan kerätään tietoa www-sivustoilta tai yrityksen ajoneuvojen sijainneista ja aika usein tallennetaan tietoa taulukkolaskentaohjelmilla. Osa

sovelluksista on saatettu tehdä itse tai teetetty räätälöitynä ohjelmistopalveluja tarjoavassa yrityksessä. Kuviossa 1 on havainnollistettu tätä tietojen keräämistä yhteen ja yleisimpiä tietojen käyttötarkoituksia. Tietolähteiden lukumäärä ja tyyppi sekä tietojen käyttötarkoitus riippuvat paljon organisaation koosta ja toimialasta. Ajan saatossa tietokantoihin on varastoitunut suuria määriä tietoa asiakkaista, tuotteista, henkilöstöstä ja taloudesta. Tämä tieto muodostaa arvokkaan pääoman, jota hyödyntämällä ja analysoimalla voidaan saavuttaa kilpailuetua samalla alalla oleviin kilpailijoihin nähden. Julkishallinnon puolella tiedot on lisäksi lain vaatimuksesta pidettävä tallessa ennalta määrätyn ajan. Laki edellyttää myös tiettyjen yritysten, kuten vakuutusyhtiöiden, raportointia viranomaisille ja tällaisenkin raportoinnin määrä on kasvussa. (Hovi ym. 2009, XII, 4.)

Tietojen hajanaisuus johtaa tarpeeseen yhdistää eli integroida tietoja ja jalostaa uusi informaatio päätöksenteon tueksi. Tätä kutsutaan Business Intelligenceksi. Rakentamalla tietovarasto (Data warehouse) voidaan useasta eri lähteestä tuleva tieto koota ja yhdenmukaistaa hallitusti yhteen paikkaan ja BI-ratkaisujen ja -välineiden avulla esittää helppokäyttöisesti käyttäjälle. (Hovi ym. 2009, XI, XIII.)

2.2 Historiasta

Jo ensimmäisten, suurien tietokoneiden aikakaudelta lähtien on tietokoneita käytetty tieteelliseen laskentaan sekä raporttien muodostamiseen. Eräs todiste tästä hyvin aikaisesta raporttien tarpeesta ja tekemisestä on Sven Hedin kirjassa 60-luvulta. Jo tuolloin on alettu korostaa sitä, miten atk:n avulla saadaan johdolle informatiivisia raportteja. (Hovi ym. 2009, 10.)

Tietotekniikan kehittyessä on kehitetty erilaisia johdon MIS-järjestelmiä (Management Information Systems), DSS-järjestelmiä (Decision Support Systems) eli päätöksenteon tukijärjestelmiä sekä EIS-järjestelmiä (Executive Information Systems). Kaikki nämä järjestelmät ovat samakaltaisia ja palvelivat käyttötarkoitustaan vaihtelevin menestyksin. Ongelmana oli usein ollut päättää,

mitä tietoja johdolle tarjoillaan, kuinka paljon ja millä välineillä. (Hovi ym. 2009, 10.)

Näihin johdon järjestelmiin tiedot tallennettiin usein summatasolla. Monesti hetken raporttien tarkastelun jälkeen johdolle tuli tarve nähdä joitakin lukuja tarkemmalla tasolla, he halusivat tarkastelemaan yksityiskohtia, joista luvut olivat summattu. Esimerkiksi Itä-Suomen alueen työkalutuoteryhmän myynti oli pienentynyt edelliseen kuukauteen verrattuna (summataso), jolloin johdolle tuli tarve päästä katsomaan myymälöittäin, missä on mennyt huonosti. Näissä varhaisissa johdon järjestelmissä ei kuitenkaan ollut mahdollisuutta porautua tarkemmalle tasolle, jolloin moni johto turhautui ja vähitellen lakkasi käyttämästä koko järjestelmää. (Hovi ym. 2009, 10.)

1980-luvulla syntyivät informaatiokannat eli lyhyesti sanottuna infokannat operatiivisten kantojen rinnalle. Isoissa yrityksissä lanseerattiin käyttöön raportointiin erikoistuneita palveluyksiköitä, joita kutsuttiin nimellä Infocenter. Näitä infokantoja varten luotiin omia tietokantoja, joihin kopioitiin tarpeen mukaan erilaisia tietoja operatiivisista järjestelmistä. Tämä muistuttaa hieman nykyisiä tietovarastoja, mutta tuon aikaisten järjestelmien suunnittelu oli vähäistä ja tietoa kopioitiin järjestelmiin kysynnän mukaan eikä johdonmukaisesti kokonaisuutena. (Hovi ym. 2009, 10.)

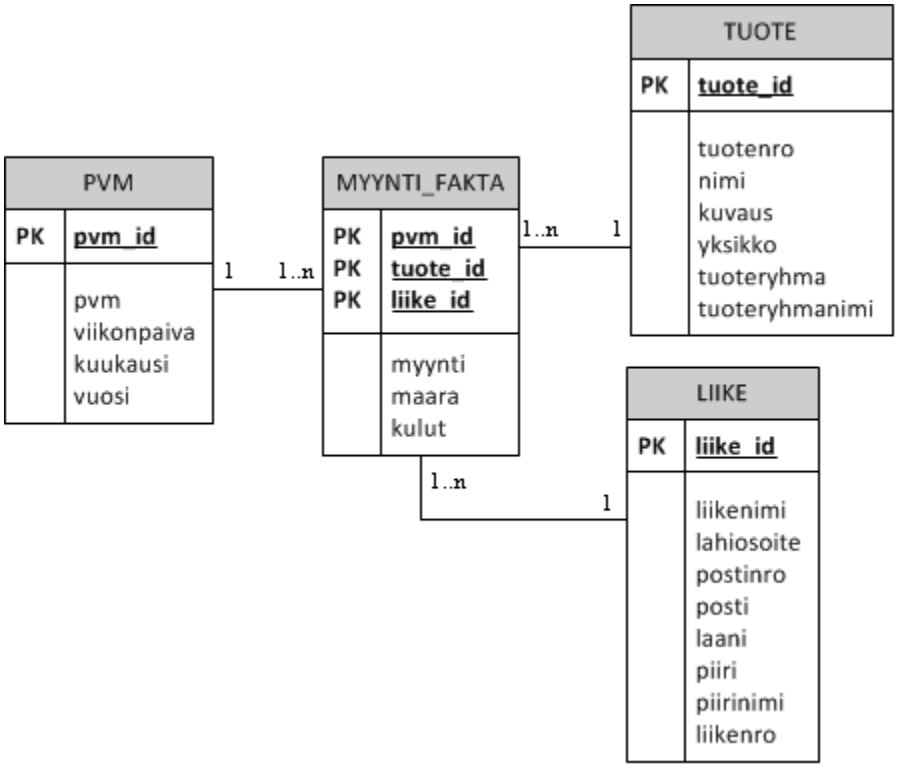
Tietotekniikan kehittyessä eteenpäin henkilökohtaiset tietokoneet yleistyivät ja mahdollistivat periaatteessa kenen tahansa analysoida tietoja koneellaan. Taulukkolaskentaohjelmilla varustettuja koneita käytettiinkin tiedon tutkimiseen ja analysointiin, mutta käytännössä kuitenkin huomattiin, että tiedot olivat usein väärin. Sen lisäksi tiedot olivat puutteellisia ja vanhentuneita, joten ne saattoivat johtaa myös vääriin johtopäätöksiin. Eri henkilöillä saattoi olla samoista tiedoista eri versio eikä tietojen määrityksiä ja kuvauksia ollut. (Hovi ym. 2009, 10.)

Henkilökohtaisten tietokoneiden aikakauden kanssa samaan aikaan kehiteltiin jälleen uudenlaista tietovarastoa nimeltä Data Warehouse. Termi esiintyi ensimmäisen kerran jo vuonna 1988 Devlinen ja Murphyn artikkelissa: ”An architecture for business and information systems” IBM Systems Journal -lehdessä. W. H. Inmonia pidetään USA:ssa yleisesti Data Warehouse -konseptin

eli tietovarastoinnin isänä. Data Warehouse suomennetaan yleisesti tietovarastoksi. (Hovi ym. 2009, 11.)

Tietovarastoinnin isänä pidetyn henkilön jälkeen teoriaa kehitti edelleen Ralph Kimball, joka jalosti tietovarastojen moniulotteisen suunnittelun menetelmän, niin sanotun tähtimallin. Kimballia pidetään dimensionaalisen suunnittelun ykkösnimenä ja hänen mallinsa on yleinen datamarttien suunnittelussa. Konsultit Inmon ja Kimball ovat tietovaraston rakentamisen arkkitehtuurista hieman eri mieltä, mutta tietovaraston käyttötarkoituksesta riippuu, kumpaa arkkitehtuuriratkaisua kannattaa käyttää. (Hovi ym. 2009, 11, 36.)

Tähtimallin (Start schema) mukainen suunnittelumenetelmä tukee hyvin moniulotteista ja numeerista tietoa. Nimi tähtimalli tulee sen rakenteesta, joka muistuttaa tähteä sakaroineen. Sen avulla simuloidaan moniulotteisuutta käyttäen kuitenkin perinteistä relaatiokantaa. Mallissa taulurakenne jaetaan dimensio- ja faktatauluihin, joissa faktataulut on kytketty niiden ympärillä oleviin dimensioihin keinotekoisin avaimin eli surrogaattiavaimilla. Faktatauluissa tietoja ei toisteta eli toisin sanoen taulut ovat normalisoitu, mutta dimensioissa sen sijaan tietoja toistetaan. Jos toisteisuus poistettaisi myös dimensioista, muuttuisi malli niin sanotuksi lumihiutalemalliksi. Seuraava kuvio on esimerkki tähtimallista, jossa on yksi faktataulu ja kolme dimensiota. Perusavaimet on alleviivattu. (Hovi ym. 2009, 36–37.)



KUVIO 2. Esimerkki tähtimallista (Hovi ym. 2009, 37)

2.3 Tietovarastot nykyisin

Uusin Data Warehousing -tekniikoiden alle kehitetty teoria on suunniteltu vasta 2000-luvulla. Data vault -malli on Dan Lindstedtin kehittämä melko uusi keskitetyn yritystason tietovaraston malli. Se on hybridimalli, joka hyödyntää kolmannen normitetun mallin ja Kimballin tähtimallin parhaimpia puolia. Tällainen tietovarasto säilyttää kaikki siihen tehdyt muutokset, ja sieltä ei koskaan poisteta tai päivitetä mitään dataa, ainoastaan lisätään. Uutta tietoa lisätessä päivitetään vain viittaus dataan, joka on hakuhetkellä voimassa. Vanhat viittaukset jäävät kertomaan datan historiallisen tilanteen kyseisellä aikavälillä. Tietokannan päälle toteutettava looginen malli on todettu olevan hyvin skaalautuva ja joustava, joten siihen on helppo lisätä uusia taulurakenteita ja dataa ilman suuria muutoksia olemassa oleviin tauluihin tietokannassa. Menetelmän muita etuja on mahdollisuus hallita isoja tietomassoja, tiedon integrointi monista eri lähteistä, historiointi ja jäljitettävyys tiedon alkulähteille sekä auditointi. (Korhonen 2010, 11–12.)

Nykyään tietovarastoinnin yhteydessä käytetään myös termiä Business Intelligence. Laajemmista kokonaisuuksista puhuttaessa BI:llä tarkoitetaan käyttäjien työkaluosuutta ja tietovarastoinnilla puolestaan tarkoitetaan latausprosessien ja itse tietovarastotietokannan suunnittelua ja toteutusta. Siinä missä tietovarasto on IT-ammattilaisten aluetta, ovat BI-työkalut liiketoiminnan käyttäjien aluetta. (Hovi ym. 2009, 11.)

Syitä siihen, miksi tietovarastot juuri nyt herättävät suurta kiinnostusta ja investointihaluja sekä yrityksissä että julkishallinnon yksiköissä, on pääasiassa kahdenlaisia. Nämä kaksi kategoriaa ovat liiketoimintasyyt ja tekniset syyt. Liiketoimintaan liittyvä syy on esimerkiksi kilpailutilanne, joka vaatii sisäisen ja ulkoisen tietoresurssin tarkempaa analysointia erilaisten vaihtoehtojen vertailemiseksi ja tutkimiseksi. Asiakastietojen tehokkaampi hyödyntäminen on noussut tärkeäksi asiaksi. Asiakkaansa hyvin tuntevat ja heidän kanssaan yhteistyötä tekevät yritykset ovat tulevaisuuden menestyjiä. Myös informaatiota yrityksen toiminnasta halutaan saada ja käyttää entistä tuoreempana eri organisaatiotasoissa. Yritysrakenteiden muutokset kuten yritysostot, fuusiot ja allianssit on helppo merkitä tietovarastoon. (Hovi ym. 2009, 11.)

Teknisiä syitä sille, miksi tietovarastotekniikkaa puolletaan, on muun muassa halu sovittaa erilliset ja yhteensopimattomat tietojärjestelmät yhteen, mahdollisuus kasvattaa operatiivisten järjestelmien elinikää ja tietovarastojen kyky säilyttää tietojen historia. Kaikkien eri lähdejärjestelmien tietojen ollessa samassa tietovarastossa voidaan tehdä ristiinanalysointia sekä raportoida kaikkia liiketoiminnan osa-alueita. Koska tietovarasto sisältää lähdejärjestelmiensä tiedoista kopion, voidaan analysointi- ja raportointityökaluilla yhdistää operatiivisten järjestelmien sijaan tietovarastoon ja näin pienentää lähdejärjestelmien kuormitusta päivisin. (Hovi ym. 2009, 12.)

2.4 Tietovaraston muodostaminen

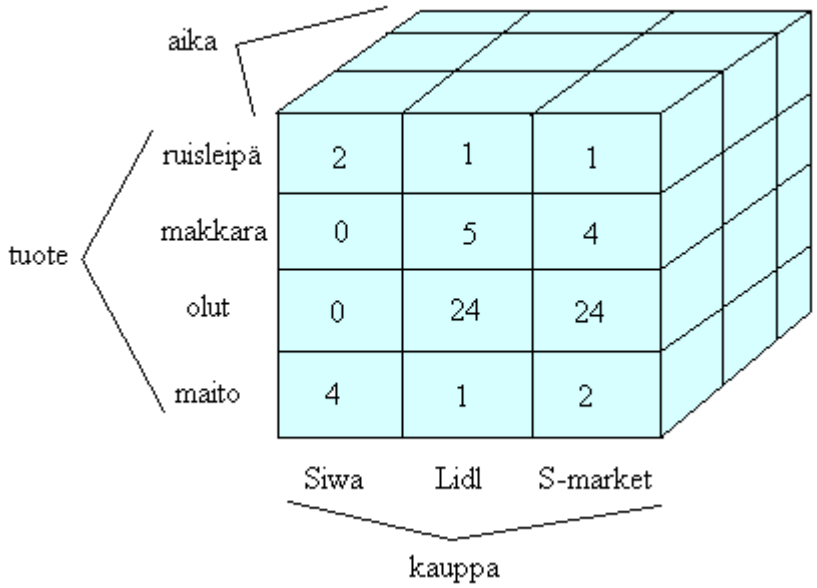
Tietovarastoon täytyy aluksi suunnitella varastoinnin mukaan tulevat lähdejärjestelmät ja tarvittava tietojen laajuus. Tietovarastoon voidaan tuoda kaikki mahdollinen data lähteistä, mutta tavallisesti vain oleellisesti analysointiin

ja raportointiin liittyvät tiedot tuodaan eikä esimerkiksi kuva-, ääni- tai XML-tiedostoja tarvita. Seuraavaksi täytyy selvittää mukaan tulevien tietokantataulujen kaikki sarakkeet, jotta tiedetään, mitä dataa järjestelmään oikeasti siirretään. Lähdejärjestelmien kanta- ja tietokuvaukset ovat melko usein auttamattoman vanhentuneita tai lähes olemattomia, kun järjestelmää on laajennettu jälkeenpäin toiveiden mukaan eikä dokumentointia ole muistettu päivittää. Myös tekniset rajapinnat lähteisiin ja tulevaan tietovarastoon täytyy selvittää.

Nykymuotoiseen Lindstedtin data vault -mallinnusta hyödyntävään tietovarastoon tarvitaan vähintään kaksi tietokantaa: työalueen- (Staging Area) ja data vaultin tietokannat. Työalueelle kopioidaan haluttu tieto lähteistä valituista tauluista ja halutulla aikavälillä ETL-latausprosessin (Extract, Transform, Load) kautta. Data vault on itse tiedon säilytyksen ydin. Sinne luodaan Lindstedtin periaatteiden mukaisesti taulurakenteet, jotka ovat helposti tarvittaessa laajennettavissa ilman koko tietovaraston tai edes tietokannan uudelleensuunnittelua (Lindstedt 2002). Kaikki tietokannoille muutoinkin oleellinen tieto, kuten primääriavaimet ja ulkopuolelle viittaavat taulujen avaimet (Primary Key, PK ja Foreign Key, FK), voiko tieto olla NULL-tilassa ja tiedon tietotyyppi mukaan lukien, täytyy myös tallettaa metadataan ja tietovarastoon.

Vähimmillään kaksi tietokantaa riittää, jotta tietovarastoa voidaan hyödyntää raportoinnissa suoralla tietoyhteydellä, mutta ajan myötä tiedon muutokset ja uusi tieto kasvattavat data vaultin kokoa niin, että työkaluille tarjotaan yhä enemmän tietoa kerrallaan. Tämä on ongelma vain silloin, kun tarvitaan raportoinnissa nykytilanteen lisäksi historiallinen tilanne. Jos vain nykytilannetta vastaavat rivit haetaan DV:stä, tietokuorma kasvaa, mutta hillitymmin. Yleensä loogisen rakenteen päälle liitetään vielä yksi tai useampi tietokanta, joita kutsutaan datamarteiksi. Datamartin voi suomentaa paikallis- tai alitietovarastoksi. Se on suppeampi otos data vaultin datasta, esimerkiksi tiedot viimeiseltä 12 kuukaudelta, ja on usein aihealuekohtainen tai organisaatioyksikkökohtainen. Taulujen rakenne voi olla tähti- tai summataulumuotoinen. Datamartteja käytettäessä voidaan valita, kytkeydytäänkö analysointi- ja raportointivälineillä suoraan data vaultiin vai datamarttiin, jonka hakuajat ovat pienemmät kuin data vaultin sen pienemmän tietomäärän takia. (Hovi ym. 2009, 188.)

Datamartteja voidaan myös hyödyntää tiedon esittämisessä OLAP-kuutioiden (OnLine Analytical Processing) avulla, koska tieto muutetaan muutenkin kuutioiden vaatimaan muotoon datamarttiin. OLAP-tekniikka tarkoittaa tietojen koostamista moniulotteiseksi tietomalliksi, joka voidaan ajatella kolmiulotteisena kuutiona. Esimerkkinä kuutiosta voidaan ajatella henkilön viikon ruokaostoksia. Tällöin kuution kolmeen dimensioon, periaatteessa x-, y- ja z-akseleille, tulee ostetun tuotteen nimi, kauppa sekä ostopäivämäärä ja kuution soluihin ostettu määrä. Kuviossa 3 on havainnollistettu esimerkin henkilön ostostiedot kolmiulotteisessa kuutiossa, jossa jokainen kuution lohko kuvaa yhtä viikkoa, eli aikatasossa yksi vaakarivi tarkoittaa samaa viikkoa. Käytännössä tietoja selataan kaksiulotteisena taulukkona analysointiohjelmilla.



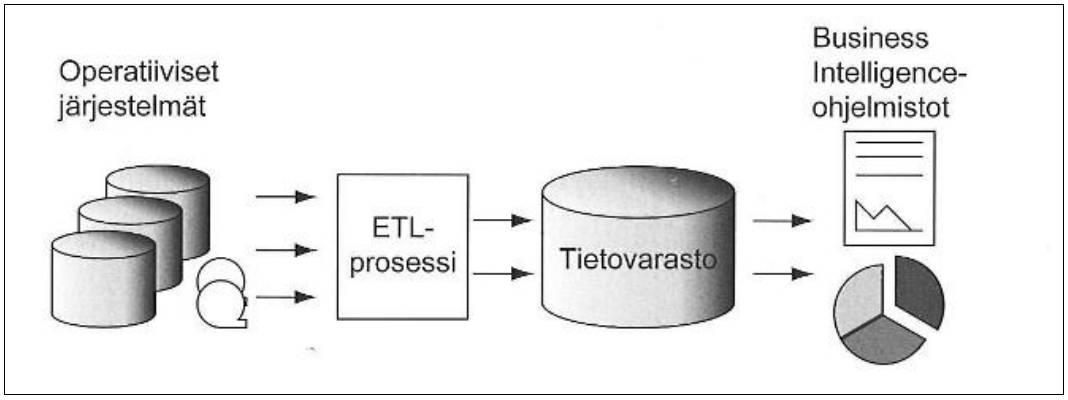
KUVIO 3. Esimerkkihenkilön ruokaostokset OLAP-kuutiossa

Kun tarvittavat tiedot on valittu ja tieto kuvattu hyvin eli lähdedatasta on muodostettu metadata, voidaan arvioida tarvittava ohjelmisto ja laitteisto toteutettavalle tietovarastolle. Laitteiston tehoon liittyy vielä tietojen päivitystarve: jos tiedot tarvitsee pystyä lataamaan usein ja nopeasti, tarvitaan tehokas palvelin. Tavallinen latausväli järjestelmissä on yksi päivä. Työalueen tietokannan lataamista varten tarvitaan nopeat tietoliikenneyhteydet ja tarpeeksi levytilaa, jonka nopeudella ei ole suurta merkitystä. Data vault -puoli sen sijaan

vaatii prosessointitehoa ETL-käsittelyn ajoon ja tietokannan käskyjen ajoon eli yhden tai useamman moniytimisen prosessorin, paljon keskusmuistia sekä paljon kiintolevytilaa.

2.5 ETL-prosessi

ETL (Extract, Transform, Load) on prosessi, joka tarkoittaa tiedon purkamista eri lähteistä, sen muuntamista yhtenäiseen muotoon ja lataamista eteenpäin tietovarastoon (Korhonen 2010, 17). ETL-prosessiin on olemassa valmiita työkaluja, mutta tarvittavan ohjelmiston voi toteuttaa myös itse halutuilla ominaisuuksilla ja tiedon muunnosten tarkkuudella. Kuvio 4 näyttää ETL-vaiheen sijainnin tiedonkulun etenemiskaaviossa. Prosessi siirtää tiedot tietovarastoon, josta ne ovat edelleen BI-työkalujen käytettävissä



KUVIO 4. Tiedon kulku tietovarastoa käyttäessä (Hovi ym. 2009)

ETL:ssä ensimmäinen vaihe on lähdedatan hakeminen lähdejärjestelmistä työalueen tietokantaan. Tietoa voidaan ladata tietokantojen lisäksi myös teksti-, taulukkolaskenta- tai XML-tiedostoista riippuen ETL-lataajasovelluksen tuesta. Yleensä tiedot ladataan prosessoitavaksi iltaisin työajan jälkeen tai öisin, sillä suuren tietomäärän takia tietovarastopalvelimen käyttö voi olla hidasta tai epäonnistua suuren kuormituksen vuoksi (Korhonen 2010, 20). Lähdekantoja ladatessa tietoa kopioidaan kymmenistä megatavuista kymmeniin gigatavuihin asti tietovarastoon riippuen ladattavasta aikavälistä. Tietojen lataaminen

työalueelle voidaan suorittaa myös eri aikaan eri lähteistä tai vaikka tietyn aikajakson sisällä, jonka jälkeen tieto prosessoidaan yhtenäiseksi ja ladataan tietovarastoon. Koska lähdejärjestelmien taulujen tietotyyppien pituus vaihtelee, muunnetaan tietotyyppi ja sarakkeen pituus yhtenäiseksi, jotta tietoa on helpompi käsitellä tietokannassa. (Korhonen 2010, 17–19.)

Seuraava vaihe on tiedon muuntaminen yhtenäiseksi. Käytännössä tämä tarkoittaa taulujen tiettyjen tietojen muuntamista työalueen ja tietovaraston tietokantojen tukemaan muotoon. Esimerkiksi totuusarvoja voidaan merkitä jossain tietokannassa kirjaimilla F ja T (F = False, T = True) ja toisessa tietokannassa arvoilla 1 ja 0. Myös päivämäärätietojen muuntaminen ja pilkkujen vaihtaminen pisteiden tilalle desimaaliluvuissa ovat mahdollisia muunto-operaatioita. Lisäksi prosessin aikana poistetaan mahdolliset duplikaattirivit työalueen tietokannasta. Tiedolle voidaan tehdä myös muita muunnoksia, mutta varsinainen tiedon jalostus tapahtuu tietovarastosta kyselyillä saadulle datalle. (Korhonen 2010, 19–20.)

Viimeinen vaihe ETL:ssä on tiedon lataaminen tietovarastoon. Päivityksiä voidaan tehdä joko tietue kerrallaan tai kaikki tieto kerrallaan sovelluksen avulla, joka on nopeampi tapa. Reaaliaikavaatimukset täyttävissä tietovarastoissa tiedot siirretään yleensä tietue kerrallaan, jotta uusin tieto on mahdollisimman nopeasti saatavilla tietovarastossa kyselyjä ja raportointia varten. Tiedonsiirto voidaan myös jakaa useisiin rinnakkaisiin operaatioihin siirron nopeuttamiseksi. (Korhonen 2010, 20.)

Customer					
custno	custkey	name	addr	city	country
123512	186125-8	Leipäpojat Oy	Jauhokatu 5	Lahti	Finland
123512	135322-8	Matin Kone Oy	Hiomokatu 17	Helsinki	Finland
123512	198457-8	Siivouskulma Ky	Kulmakatu 2	Tampere	Finland
...	...	...	...	..	..

Asiakas				
astun	y-tunnus	nimi	osoite	kunta
73423	186125-8	Leipäpojat Oy	Jauhokatu 5	Lahti
69234	194383-8	Teknodata Oy	Tietotie 3	Oulu
...	...	...	...	..

KUVIO 5. Esimerkki eri lähdejärjestelmien asiakastauluista

Kuviossa 5 on kahden eri lähdejärjestelmän asiakastaulut. Tauluissa on hieman eri määrä sarakkeita ja niiden nimetkin eroavat, ja eri tauluissa samalla yrityksellä, Leipäpojat Oy:llä, on eri asiakasnumero. Metadataa määritettäessä täytyy jokaisesta taulusta poimia yksilölliset avaimet joka taululle ja tässä asiakastaulussa yksilöllinen tieto on yritystunnus. Kuviossa 6 näkyy, kuinka data vaultiin on tuotu kaikki erilaiset y-tunnukset ja niitä vastaavat tiedot. Koska Teknodata Oy:lle ei ollut määritelty maata, jää solun tieto NULL-tilaan eli tietokannan ei arvoa -tilaan. Kyseisen yrityksen maatieto korjaantuu, jos jostakin muusta lähteestä tulee Teknodatan maatieto, lähdejärjestelmään lisätään maa-kenttä ja tieto tai haetaan tieto käsin tehdystä poikkeustaulusta.

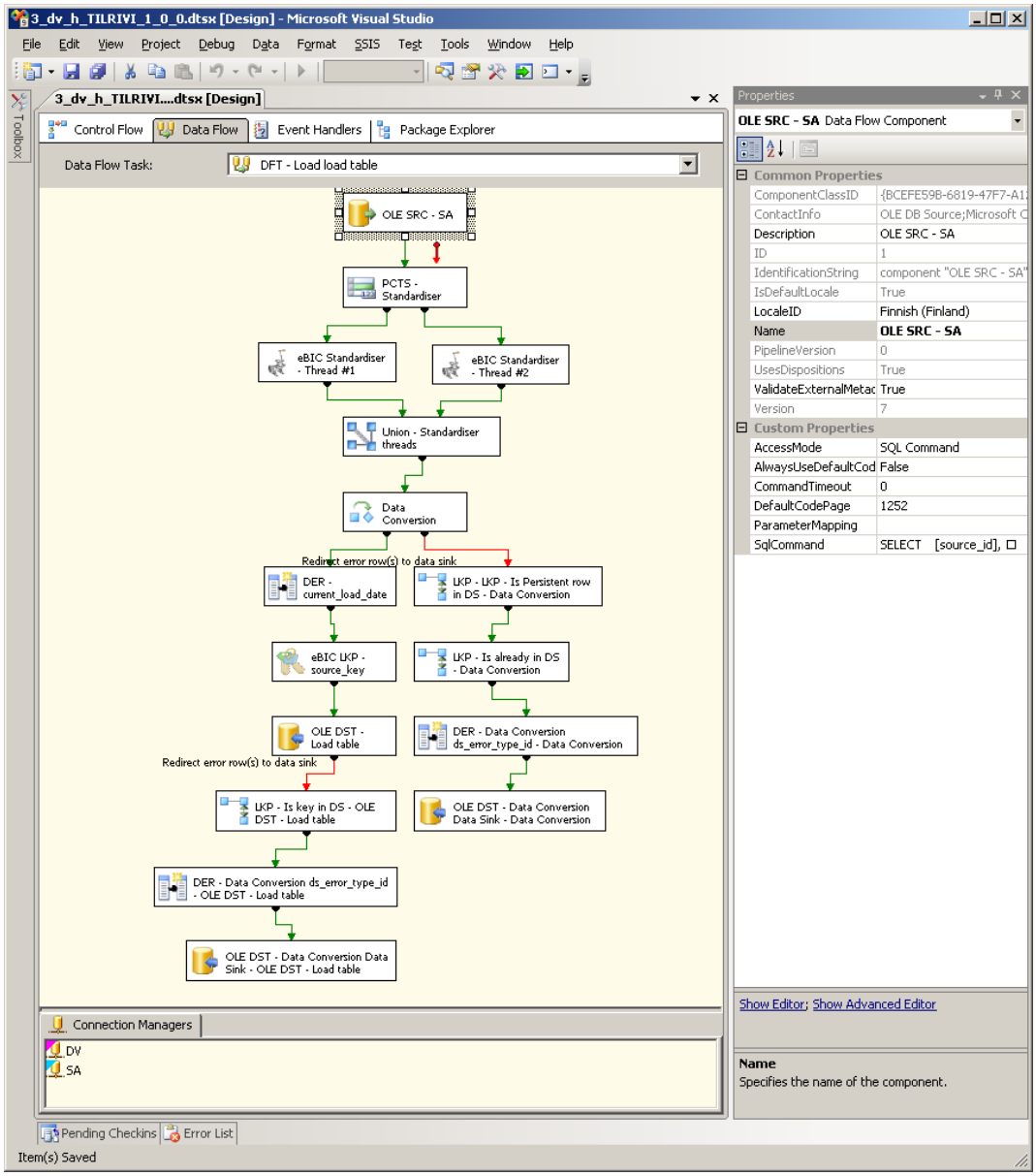
Customer						
customer_key	source_key	vat_num	name	address	city	country
1	1	186125-8	Leipäpojat Oy	Jauhokatu 5	Lahti	Finland
2	1	135322-8	Matin Kone Oy	Hiomokatu 17	Helsinki	Finland
3	1	198457-8	Siivouskulma Ky	Kulmakatu 2	Tampere	Finland
4	2	194383-8	Teknodata Oy	Tietotie 3	Oulu	NULL
...	...	...	...	...	..	..

KUVIO 6. Tietovaraston asiakastaulu

2.6 SSIS-lataajat

SSIS- eli SQL Server Integration Services -komponentit ovat Microsoftin SQL-palvelimien yhteydessä käytettävä tekniikka, jonka avulla voidaan ladata, muuntaa ja siirtää tietoa toiseen tietokantaan (Wikipedia 2012). Nämä lataajat ovat XML-muotoisia tiedostoja, jotka kokoavat yhteen joukon SQL-käskyjä ja tiedon muuntamiseen tarkoitettuja toimintoja. Tiedostot voidaan ajaa tiedostoa tuplaklikkaamalla tai konsolin kautta tai MS SQL -palvelimen kautta ajastetusti millä tahansa koneella, jossa on käytettävissä lataajassa määritetyt tietolähteet ja kohteet sekä SQL-palvelimen SSIS service asennettuna. Sekä lähteenä että kohteena voi olla XML-tiedostot, taulukkolaskentaohjelmien tiedostot, Microsoftin tai muiden valmistajien tietokannat tai pelkät tekstitiedostot.

Kuviossa 7 on hubilataajan h_TILRIVI Data Flow –tietovuovälilehti auki, jossa näkyy ensimmäisenä työalueen (SA, Staging Area) tietokanta tietolähteenä ja sieltä saatavalle tiedolle tehdään erilaisia käsittelyjä. Muunnosten jälkeen prosessoidut tiedot ladataan data vault -tietokantaan.



KUVIO 7. SSIS-paketti avattuna Visual Studiossa

2.7 Metatiedon kuvaus

Metatieto on tietoa tiedosta. Metatiedot kuvaavat tietojen sisällön ja merkityksen. Metatieto voidaan jakaa käyttäjäorientoituneeseen ja tekniseen tietoon. Ensin mainittu kuvaa tiedot käyttäjän näkökulmasta sisältäen kunkin tiedon ja taulun määritelmän sekä kuvauksen. Tekninen metatieto kuvaa tiedon lataukseen liittyviä asioita, kuten käsittelysääntöjä ja teknisiä kuvauksia. Metatiedon keräämiseen, kokoamiseen ja tallennukseen on hyvin vähän olemassa valmiita työkaluohjelmia. Yksi vaihtoehto on kehittää metatiedon hallintaan ohjelma itse, jolloin se ainakin

täyttäisi halutut vaatimukset ja toimisi niissä ympäristöissä, joissa sitä tarvitaan. Tämä tietenkin sillä edellytyksellä, että tietovaraston toteuttajalla on aikaa ja halua käyttää resursseja ohjelmistokehitykseen. (Hovi ym. 2009, 188.)

2.8 Data vaultin rakenne

Data vault on uusin tiedon integroinnin-, varastoinnin- ja historioinnin toteuttava konsepti. Määritelmäksi idealle on antanut sen kehittäjä Dan Lindstedt seuraavan: ”se on yksityiskohtaorientoitunut, historian säilyttävä ja uniikisti linkitetty kokoelma normalisoituja tietokantatauluja, jotka tukevat yhtä tai useampaa funktionaalista liiketoiminta-aluetta” (suomennettu). Se voi käsitellä suuria tietomääriä ja jäljitellä tosielämän monimutkaisia rakenteita tarvittaessa, sillä se on Enterprise Data Warehouse -tekniikkaan eli laajimpaan ja monimutkaisimpaan teoriaan pohjautuva. (Lindstedt 2002.)

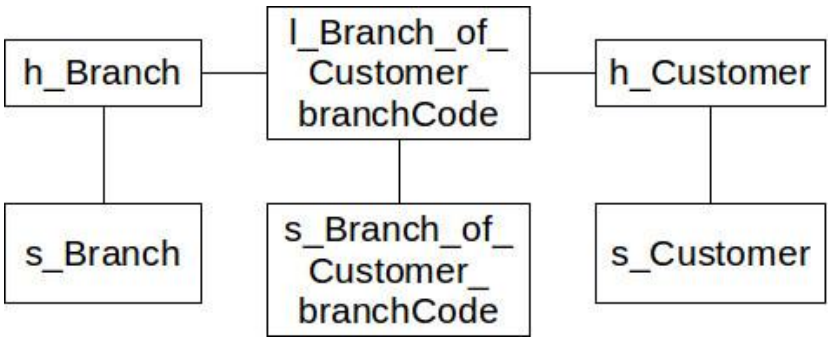
Sen voi lisäksi sanoa olevan tekniikkapainottunut, sillä sinne tuodaan kaikki valittu tieto riippumatta siitä, onko se oikein vai väärin, sillä kaiken tiedon oletetaan olevan relevanttia. Jos tieto on väärin, se on vain liiketaloudellisessa mielessä väärin, mutta teknisesti oikein. Tämä lähtökohta poikkeaa edellisistä päätöksenteko- ja tietovarastointiohjelmista selkeästi ja virheiden korjaamiseksi tietovarastossa täytyy virheet korjata lähdejärjestelmistä ja ladata uudelleen järjestelmään.

Data vault yhdistää kolmannen normitetun muodon eli 3NF:n ja tähtiskeeman parhaimmat puolet omiin ominaisuuksiinsa. Sen tukemia muita ominaisuuksia ovat monen suhde moneen -yhteydet, viite-eheys, minimaalinen tiedon toistuvuus ja liiketoiminnan avaimiin perustuvat taulujen avaimet. Lisäksi siinä on tuki dynaamisille tietojen riippuvuuksien merkitsemiselle, jota eivät muut mallit tue. Mallin kasvamisnopeutta ja laajenemismalleja on kehittäjän mukaan tutkittu matemaattisesti joustavan mallin löytämiseksi. Se siis rakenteensa puolesta sopisi laajoihin, eri alueita kuvaaviin malleihin kuten vakuutus, pankki- ja tuotantoalueisiin. (Lindstedt 2002.)

Lyhyesti sanottuna data vault -malli koostuu satelliitti-, hub- ja linkkitauluista. Data vault -teoria on keskittynyt liiketoiminnan toiminnallisen alueen ympärille

säilyttäen tietojen yksiköllisiä, primääriavaimia hub-tauluissa. Linkit muodostavat vuorovaikutuksen eri hubien välille. Satelliitit puolestaan tarjoavat loput hubiin liittyvät tiedot, hubin kontekstin.

Kuviossa 8 on esimerkki kahdesta hub-taulusta ja niitä yhdistävästä linkkitaulusta sekä näiden taulujen satelliittitauluista. Branch- eli toimialataulussa on yksilöllisenä avaimena kenttä Code, joka on yrityksessä päätetty toimialan koodi. Asiakastaulussa puolestaan on branchCode-kenttä, jossa kerrotaan asiakkaan toimiala koodinumerona. Linkkitaulussa on viittaus molempien hub-taulujen surrogaattiavaimiin, joka yhdistää toimialan nimen asiakkaaseen.



KUVIO 8. Esimerkki data vaultin taulurakenteesta

Hub on yksittäinen tietokantataulu, joka listaa vähimmäismäärän business-avaimia muodostaakseen yksiköllisen avaimen. Jokaista lähdejärjestelmän taulua vastaa yksi hub-taulu. Nämä avaimet ovat samoja tietoja, jota liike-elämässäkin hyödynnetään. Esimerkiksi malli tällaisesta avaimesta voisi olla yritys, asiakasnumero, sopimusnumero ja sopimusrivi. Näiden lisäksi jokaisella hubilla on surrogaattiavain eli virtuaalinen avain, joka on yleensä juokseva numero; latauspäivämäärän aikaleima, jolloin tieto on ladattu tietovarastoon ensimmäisen kerran sekä lähteen tunnus, jotta tiedetään, mistä tieto on tullut ja voidaan säilyttää tiedon jäljitettävyys. Lähteiden nimistä ja vastaavista avaimista muodostetaan myös oma taulunsa, sillä toimivassa data vaultissa voidaan selvittää tiedon lähde mistä taulusta tahansa. (Lindstedt 2002.)

Kuviossa 9 on aiemman esimerkin toimialataulun hubin ja sen satelliitin rakenne kuvattuna. Code-kenttä muodostaa yhdessä source_keyn kanssa uniikin indeksin tauluun. Tämä tarkoittaa sitä, että jos sama yhdistelmä tulee myöhemmin vastaan, sitä ei enää lisätä tauluun eikä olemassa olevaa tietoa päivitetä. Ainoastaan satelliittitauluun luodaan uusia rivejä, jos sitä vastaavan hubin tiedot muuttuvat. Esimerkiksi nimen kirjoitusasun muututtua, päivitetään edellisen arvon end_date-leimaksi kuluvan hetken aika satelliittiin ja lisätään uusi rivi uusin tiedoin.

h_Branch				
Branch_key	source_key	code	loaded	loaded_by
1	1	1	2012-02-21 03:00:01	reinjaak
2	1	2	2012-02-21 03:00:01	reinjaak

s_Branch					
Branch_key	loaded	loaded_by	end_date	name	nameEn
1	2012-02-21 03:00:01	reinjaak	NULL	Vaatetus	Clothing
2	2012-02-21 03:00:01	reinjaak	NULL	Elintarvikkeet	Food supplies

KUVIO 9. Esimerkki hubista ja sen satelliitista

Linkkitaulut ovat fyysinen ilmentymä monen suhde moneen 3NF-muotoisista riippuvuussuhteista. Linkit esittävät tietoa kahden tai useamman liiketoiminnan komponentin välillä eli käytännössä kahden tai useamman business-avaimen välillä. Linkeillä on myös surrogaattiavain kuten hubeilla, mutta sitä ei ole pakko käyttää kuin siinä tapauksessa, että linkki yhdistää useamman kuin kaksi hubia yhteen. Sitten merkitään hubin 1. avaimesta viittaus hubin avaimeen N, jossa N kasvaa niin suureksi, kuin tarkasteltavassa hubissa on loogisia yhteyksiä. Linkeissä pitää myös olla latauspäivämäärä, milloin se on luotu ensimmäisen kerran ja viittaus lähdekantaan. (Lindstedt 2002.)

Aiemmin mainitun asiakas- ja toimialataulun yhteys kuvataan linkkitaululla kuvion 10 mukaisesti. Link_key on taulujen surrogaattiavain, PK- ja FK-avaimet osoittavat asiakas- ja toimialataulujen surrogaattiavaimiin. Kuten kuviosta nähdään, on toinen linkki päivittynyt osoittamaan toiseen toimialakoodiin ja ensimmäiseen linkkiin on lisätty end_date-aikaleima, jolloin linkki on vanhentunut.

l_Branch_of_Customer_branchCode					
link_key	source_key	pk_Branch_key	fk_Customer_key	loaded	loaded_by
1	1	4	6	2012-04-29 03:01:23	reinjaak
2	1	35	17	2012-02-21 03:00:01	reinjaak
2	1	51	17	2012-04-29 03:01:23	reinjaak

s_Branch_of_Customer_branchCode			
link_key	loaded	loaded_by	end_date
1	2012-04-29 03:01:23	reinjaak	NULL
2	2012-02-21 03:00:01	reinjaak	2012-04-01 04:03:12
2	2012-04-29 03:01:23	reinjaak	NULL

KUVIO 10. Esimerkki linkkitaulusta ja sen satelliitista

Lindstedtin tapa on sovellutus monen suhde moneen suhteista 3NF-tyyliin, mutta skaalautuvuus- ja joustavuusongelmat ratkaistuna. Tämä tapa soveltuu tietovarastointikäyttöön, muttei OLTP-järjestelmille eli tosiaikaiseen tapahtumankäsittelyyn (OnLine Transaction Processing). Jos mallia kuitenkin välttämättä halutaan käyttää OLTP-järjestelmissä, se onnistuu pakottamalla varastointiohjelma käyttämään linkeissä ainoastaan yhden suhde moneen -linkkejä. Tällaisessa käytössä täytyy välttää liiallisia välineen käytöstä tulevia tarpeita muokata tietovarastoa, sillä data vaultin toimintaan on olemassa sääntölista. Jos vähimmäissääntöjä ei noudateta, ei järjestelmän jotkin osat, esimerkiksi skaalautuvuus tai suorituskyky, suuren kuorman alla välttämättä toimi suunnitellusti. (Lindstedt 2002.)

Pelkät hubit ja linkit sinällänsä kuvaavat vasta liiketoiminnan tapahtumien muutosta, muutoksien virtaa (business flow) (Lindstedt 2002). Hubien ja linkkien avulla voidaan esimerkiksi kertoa, että autovuokraamon omistajalla on auto, jonka rekisterinumero on ABC-123 ja asiakas 01234 ja hän haluaa selvittää auton sekä asiakkaan nimen. Hubit listaavat yksikölliset asiakkaat ja linkitys kertoo, että kyseinen asiakas 01234 on vuokrannut auton ABC-123 illaksi. Viimeinen oleellinen komponentti tallettamaan jäljelle jäävän tiedon on hubien ja linkkien satelliittitaulut.

Satelliittien päätehtävänä on kertoa, milloin tieto lisättiin, miksi se lisättiin (muutoksen takia, uusi rivi vai uusi linkitys), mitä lisättiin ja minne. Jokaista satelliittiriviä vastaa tietty hub-taulun rivi, ja ne tallettavat kaikki loput muuttuvat tiedot, kuten sopimuksen loput tiedot business-avainkenttiä lukuun ottamatta, satelliittitauluun. Koska esimerkiksi sopimuksissa voi olla mahdollista muuttaa melkein mitä tahansa tietoa, paitsi sopimusnumeroa ja muita avainkenttiä, on satelliittien rakenne suunniteltu toimimaan tiedon muutoksista huolimatta. Muutokset eivät myöskään eskaloidu yhdestä taulusta muihin riippuviin tauluihin, jos satelliitin tiedot tai jopa kenttien määrä vaihtelee, toisin kuin puhtaasti 3NF-muotoisessa tietovarastossa. (Lindstedt 2002.)

Satelliiteissa vaadittavat kentät ovat seuraavat: kaksiosainen satelliitin primääriavain, johon tulee hubin tai linkin primääriavaimen numero ja lataamispäivämäärä; valinnainen primääriavain, joka on juokseva numero sekä viite lähteeseen. Satelliittien toimintaperiaate muistuttaa läheisesti Kimballin tyypin 2 dimensiota. Se tallettaa muutokset pienellä tasolla ja viittaa aina hubiin. Itse satelliitin toiminta on mahdollista toteuttaa kahdella tavalla: Kimballin tyyliin, jolloin muutoksen tullessa tallennetaan kaikki satelliitin arvot uusiksi tai sitten eriytetään usein muuttuvat tiedot omaan pieneen satelliittiinsa ja loput toiseen satelliittiin. (Lindstedt 2002.)

Esimerkki muutosten eriyttämisestä voisi olla vaikka seuraava: tietovarastoon on tallennettu Auto-dimensio, joka kuvaa auton ominaisuuksia ja sisältää 160 erilaista attribuuttia. Jos toiminta noudattaa Kimballin tapaa, täytyy kaikki satelliitin 159 muuta tietoa kirjoittaa myös uusiksi, kun autoon vaihdetaan erilaiset renkaat. Jos taas rengasattribuutti erotetaan omaksi satelliitikseen, ei renkaiden vaihtaminen aiheuta muissa dimension tiedoissa muutoksia. (Lindstedt 2002.)

Vaikka data vaultin toteuttamisessa hyödynnettäisi vain Kimballin tapaa ja tietokantaan monistuisi samat arvot useaan kertaan, ei käytännössä tilantarve kasva suoraan rivien lukumäärän mukaisesti. Esimerkiksi Microsoftin SQL Server tukee tiedon pakkaustekniikkaa, joka pakkaa samanlaiset ja lähes samanlaiset rivit hyvin pieneen tilaan merkitsemällä yhtä riviä tai tietueiden ryhmää vastaavat lohkot vain muutamalla merkillä joka rivillä ja ainoastaan muutokset vievät enemmän tilaa. (Mishra 2009.)

Data vault siis taltioi kaikesta tiedon, mistä tieto on peräisin, kuka sen latasi järjestelmään, milloin sitä muutettiin eli suorittaa jo rakenteellisesti auditointia. Kaikki tämä on arvokasta tietoa, kun halutaan muun muassa etsiä syy, miksi jokin yrityksen luku muuttui tai kuka päivitti uudet tuloslaskelmat ja milloin. Auditointi ei ole vain yrityksille sisäisesti kätevä muutosten jäljitykseen, mutta myös joiltakin yrityksiltä odotetaan tietokannalta tarkkoja käyttöoikeuksia ja valvontaa. Muun muassa viranomaisten, kuten poliisin, järjestelmissä taltioituu jokainen kysely, joka tietokantaan tehdään. Tämä on yksi tiukempia kriteereitä, joita malli täyttää. (Lindstedt 2002.)

Lindstedtin muodostama koko lista data vault -tietovarastoinnin säännöistä ja riippuvuuksista on listattu liitteessä 1. Listan ja teorian avulla on mahdollista muodostaa oma toteutus hänen konseptistaan. Lindstedt itse myy koulutusta ja omaa ratkaisuaan korkeaan hintaan, mutta tarjoaa kuitenkin konseptista kiinnostuneille mahdollisuuden tutustua nettisivuillaan kattavaan pakettiin materiaalia kehittämästään teoriasta.

3 TIETOVARASTON GENEROINTISOVELLUKSEN TOTEUTUS

3.1 Yleistä toteutuksesta

Työ sovittiin tehtäväksi Logican Lahden toimiston tiloissa ja sitä varten annettiin käyttöön kannettava tietokone tarvittavine ohjelmineen. Testitietokannoista annettiin kopiot ja niitä käytettiin paikallisesti annetulla tietokoneella. Tietokantojen testaaminen tällä tavalla vastasi täsmälleen tuotantokantojen yhteysrajapintoja, mutta oli täysin irrallinen tuotantokantakoneista, jotta mitään vahinkoa ei testattaessa pääsisi käymään. Myös kuormituksen takia testaaminen tehtiin omalla koneella, sillä tietovaraston lataaminen SSIS-lataajien avulla aiheuttaa huomattavan kuorman lähdetietokannoille.

Tutorina opinnäytetyötä tehdessä toimi Logican Lahden toimistossa työskentelevä vanhempi ohjelmistokehittäjä, joka oli toteuttanut PoC-toteutuksen generointisovelluksesta. Annetusta dokumentaatiosta muodostettiin arkkitehtuurisuunnitelma, ja kun se oli hyväksytetty ohjelmistokehittäjällä, sovelluksen toteutus aloitettiin.

Data vault -tietokannan generointisovelluksen oli toteutettava työalueen ja data vault -tietokannan tietokantataulujen generoiminen lähdejärjestelmän tietokantataulujen perusteella ja ETL-latauspakettien luominen kumpaakin tietokantaa varten. Lopputuloksena tuli syntyä tietokantataulujen SQL-luontilauseet ja SSIS-lataajat niille.

Visiona sovelluksen toteutukselle oli toimitusprosessin nopeuttaminen automatisoimalla osa toimitusprosessista. Data vault -tietokannan taulujen luonti käsin lähdejärjestelmän tietokannan tauluista on aikaa vievää ja virhealtista työtä. Työtä ei kuitenkaan tarvitse tehdä käsin, sillä tietokantataulut luodaan ennalta määrätyillä säännöillä jokaisessa tapauksessa. Generointisovelluksella Logica vähentäisi huomattavasti tuotekehitysprosessien kokonaisläpimenoaikaa. (Rämö 2010a.)

3.2 Vaatimukset DW-generointisovelluksen hyväksymiselle

Logica asetti sovelluksen toteutuksen hyväksymiselle vaatimuksia, joiden täytyttyä projekti voitiin katsoa valmiiksi. Lista vaatimuksista oli seuraavanlainen:

1. Toteutettua PoC-toteutusta ei saanut käyttää suoraan DW-generointisovelluksen toteutuksen pohjana. Käytännössä tämä tarkoitti sitä, että mallia sai ottaa, mutta yleisesti ottaen koodi oli kirjoitettava uudelleen.
2. DW-generointisovelluksen arkkitehtuuri oli suunniteltava etukäteen ja arkkitehtuuridokumentaatio hyväksytettävä Logican nimeämällä henkilöllä, PoC-toteutuksen tehneellä ohjelmistokehittäjällä.
3. DW-generointisovelluksen tuli pystyä lukemaan tietotovaraston generoimiseen tarvittavat metatiedot SQL Server –tietokannasta. PoC-toteutuksessa oli esimerkki metatiedon lukemisesta SQL Server -tietokannasta.
4. DW-generointisovelluksen täytyi pystyä lukemaan tietovaraston generoimiseen tarvittavat metatiedot myös ODBC-tietolähteestä olettaen, että ODBC-tietolähde tarjosi metatietojen lukemiseen tarvittavat järjestelmätaulut.
5. DW-generointisovelluksen täytyi pystyä generoimaan Logican vaatimuksien mukainen työalueen tietokanta ja tietovarasto. PoC-toteutus generoi jo tällaiset Logican vaatimuksien mukaiset tietokannat.
6. DW-generointisovelluksen täytyi pystyä generoimaan SSIS 2008 -paketteja, joilla voidaan kytkeytyä seuraaviin tietolähteisiin:
   a) OLE DB Provider for SQL Server
   b) ODBC

   PoC-toteutuksessa oli esimerkki kohdan a) tietolähteen käyttämiselle.
7. Kohtiin kolme (3) ja neljä (4) liittyen: DW-generointisovelluksen oli toimittava Logican tarjoamia testikantoja vasten. Käytännössä tämä tarkoitti sitä, että Logica tarjosi testikannat, joista DW-generointisovelluksen oli pystyttävä generoimaan tietovarasto ja SSIS 2008 -lataajat.
8. DW-generointisovelluksen tuli toteuttaa käyttämällä C#-kieltä business-logiikan osalta ja WPF-kuvauskieltä käyttöliittymän osalta.

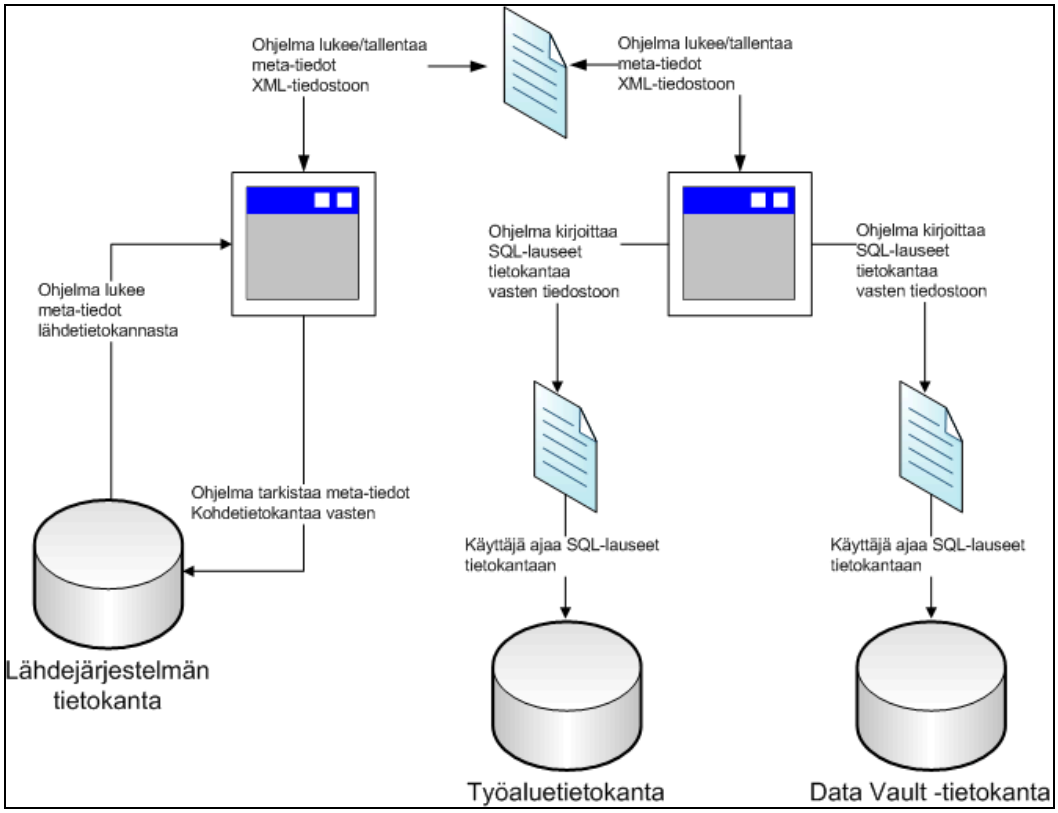
> Toteutuksessa oli käytettävä .NET Framework 3.5 _versiota ja ohjelmointiympäristönä Visual Studio 2010 Professional -versiota.

(Rämö 2010b.)

SSIS-komponenttien ohjelmointi .NET-kielillä vaati MS SQL Server API:n eli ohjelmointirajapinnan. SQL Serverin API asennettiin työkoneelle samalla SQL-palvelimen asentamisen yhteydessä.

3.3 Data warehouse -generointisovellus

DW-generointisovelluksen arkkitehtuurin valitsemiseksi ja suunnittelemiseksi tarvittiin PoC-toteutuksen vaatimusmäärittelydokumentti, itse PoC-toteutuksen lähdekoodit sekä muutama suunnittelupalaveri vanhemman ohjelmistokehittäjän kanssa. Arkkitehtuurina päätettiin pitää jo olemassa oleva tasoarkkitehtuuri, sillä sellaisen suunnittelu ja toteutus on suoraviivaista eikä sovellus vaatinut monimutkaisempaa arkkitehtuuria. Ohjelman toiminnasta oli jo olemassa korkealla tasolla oleva kaavio sen tuottamasta metadatasta ja metadatan hyödyntämisestä. Mainittu kaavio on esitetty kuviossa 11.

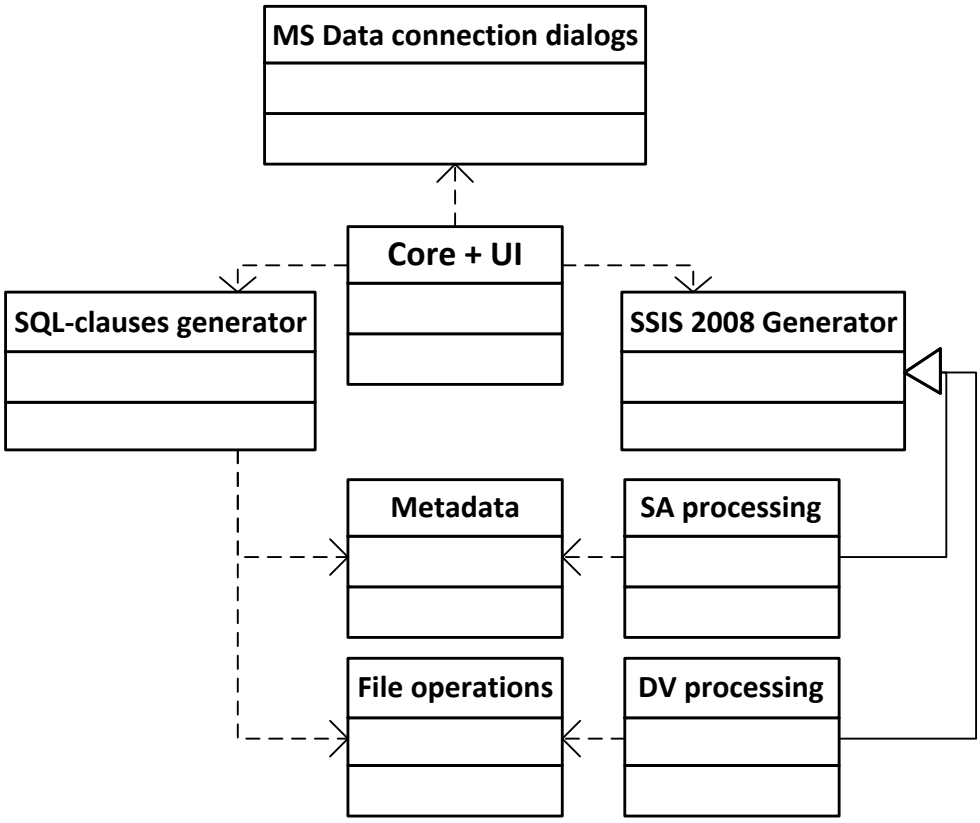


KUVIO 11. Metatiedon lukija- ja DV-generointisovelluksen looginen arkkitehtuuri

Ohjelma on suunniteltu lukemaan tietokantojen rajapintojen kautta kantakohtaiset tiedot tiedoista eli metadatan. Tämä on havainnollistettu kuvion 11 vasemmassa reunassa. Tällä metadatalla kuvataan jokaisen taulun nimi, siihen kuuluvat kentät, avaintiedot, viittaukset muihin tauluihin, indeksit ja mahdolliset taulukuvaukset. Tämä lähdekannasta muodostettu metadata voidaan sen jälkeen tallentaa XML-tiedostona, jolloin voidaan metadatan tallennushetken aikaisia lähdekannan rakenteita tarkastella myöhemminkin. Itse asiassa metadatakoodia hyödynnetään kahdessa erillisessä ohjelmassa: on tehty pieni ohjelma, jolla tarvittaessa vaikka asiakas kahdella napin painalluksella lukee tietokantansa rakenteiden tiedot ja kirjoittaa sen tiedostoon, jonka voi toimittaa Logicalle data vaultin suunnittelua ja luomista varten. Kuvion vasemmanpuoleinen ohjelma on kyseinen metadatan lukijaohjelma. Oikeanpuoleinen ohjelma on projektin aiheena ollut DW-generaattori.

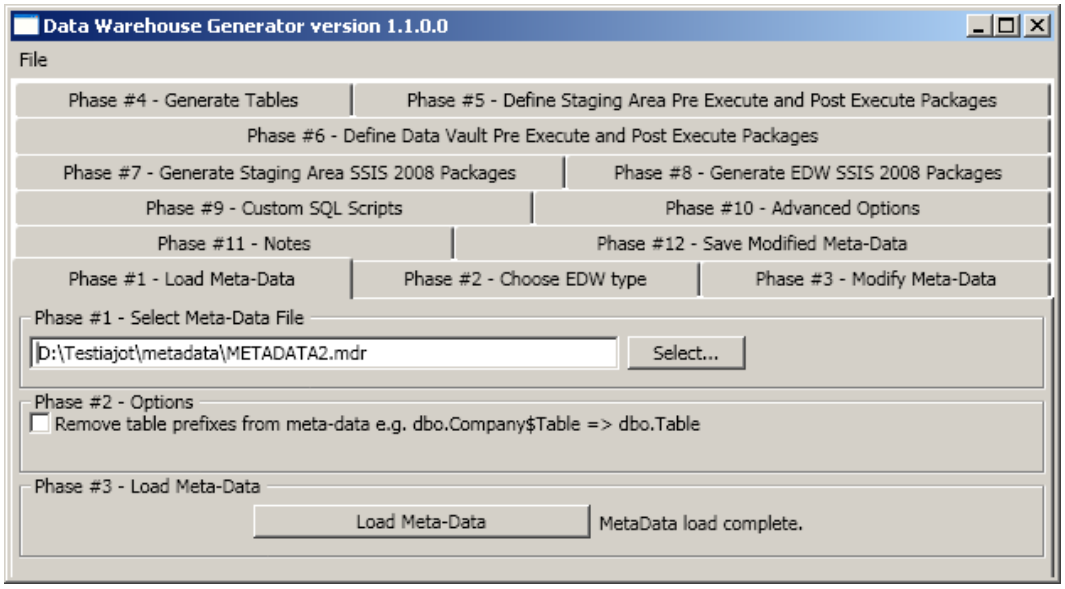
Generointisovelluksesta tehtiin myös kaavio tarvittavista moduuleista yleisellä tasolla, joka on esitetty kaaviossa 12. Sovelluksen ydin, jossa luodaan muiden luokkien olioita ja kutsutaan niiden metodeja sekä määritetään graafinen käyttöliittymä, on komponentissa ”Core + UI” (UI, User Interface). Kun ohjelmassa pyydetään käyttäjää syöttämään lähde-, työalueen- tai DV-tietokannan yhteystietoja, kutsutaan valmiiksi ikkunointi- ja yhteysrajapintoja kapseloivasta Microsoft data connection dialog -luokasta oliota. Tämä luokka ladattiin Microsoftin palvelimelta, jolla oli valmiita esimerkkejä SQL Server -rajapinnasta kehittäjille ilmaiseksi töiden pohjaksi tai osaksi. Koska tämä luokka helpotti yhteysrajapinnan (ODBC, OLE DB) käyttöä, otettiin se mukaan osaksi ohjelmaa.

Pääohjelma kutsuu käyttäjän syötteen perusteella SQL-luontilauseiden generointiluokkaa, joka luo SQL-lauseet SA- ja DV-alueille ja SSIS 2008 generator –luokkaa, joka puolestaan luo kyseisille alueille SSIS-lataajat. SQL-luontilausegeneraattori- ja SSIS 2008 generaattoriluokka hyödyntävät molemmat tiedostotallennusta tarjoavaa File operations -luokkaa ja Metadata-luokkaa, jossa on ohjelman tuotoksia varten tarvittu tieto luokkarakenteessa. Metadataluokassa on myös metadatan serialisointimetodi XML-muotoon.



KUVIO 12. DW-generaattorin luokkakaavio (vain merkittävimmät luokat)

Kuviossa 13 on kuvankaappaus ohjelman aloitusruudusta, jossa on jo ladattu esimerkin vuoksi metadatatiedosto nimeltä METADATA2.mdr. Ohjelman käyttöliittymän välilehtien määrä on hieman paisunut alkuperäisestä kehityksen aikana tulleiden toiveiden myötä, mutta koska ohjelma on tarkoitettu yrityksen sisäiseen käyttöön, ei sillä ole niin suurta merkitystä.



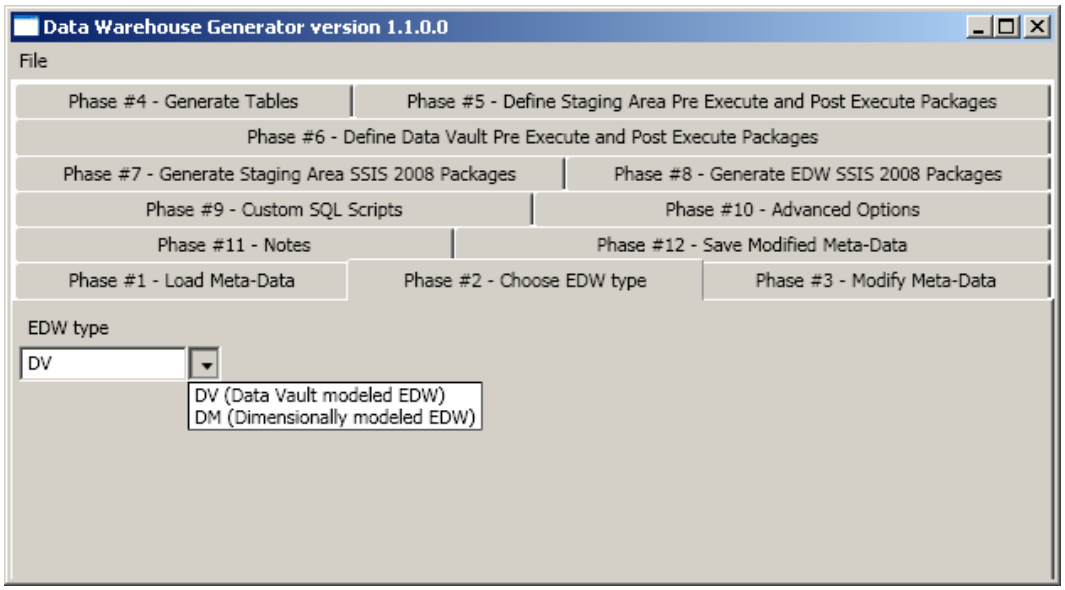
KUVIO 13. DW-generaattorin aloitusnäkymä

Seuraavaksi tallennettu metadata luetaan taulukkoon, joka hyödyntää samaa metadataluokkaa kuin pieni erillinen metadatan lukijaohjelmakin. Metadatan perusteella esitetään lista lähteestä skannatuista tauluista ja voidaan valita data vaultin generointiin mukaan tulevat taulut. Kun halutut taulut on valittu ja tarvittavat lisäasetukset tehty, voidaan tiedon perusteella generoida SQL-luontilauseet ja Microsoftin SSIS 2008 -lataajakomponentit, jotka ovat latausketjuja datan siirtämiseen ensin lähteistä työalueelle ja työalueelta data vaultiin. Ohjelman päätarkoitus on generoida SQL-lauseet ja SSIS-lataajat ohjelmallisesti, jotta päästään eroon suuresta osasta rutiininomaista käsityötä.

Näiden tuotosten generointi ohjelmallisesti vähentää työaikaa, pienentää komponenttien ja koko järjestelmien rakentamisessa syntyviä virheitä sekä nopeuttaa tietovaraston toimitusprosessia kokonaisuudessaan aiempaan verrattuna. Sen lisäksi muodostuvat tuotokset ovat toistettavia, joten esimerkiksi vahingossa poistettujen pakettien uudelleengenerointi on mahdollista metadatasta. Proof of concecpt -toteutus täyttää myös nämä annetut parannukset, mutta konseptiluonteensa takia se täytyi toteuttaa suunnitellusti uudelleen. Sovellus on vasta pienessä testikäytössä, joten edellä mainittuja etuja ei olla vielä saavutettu.

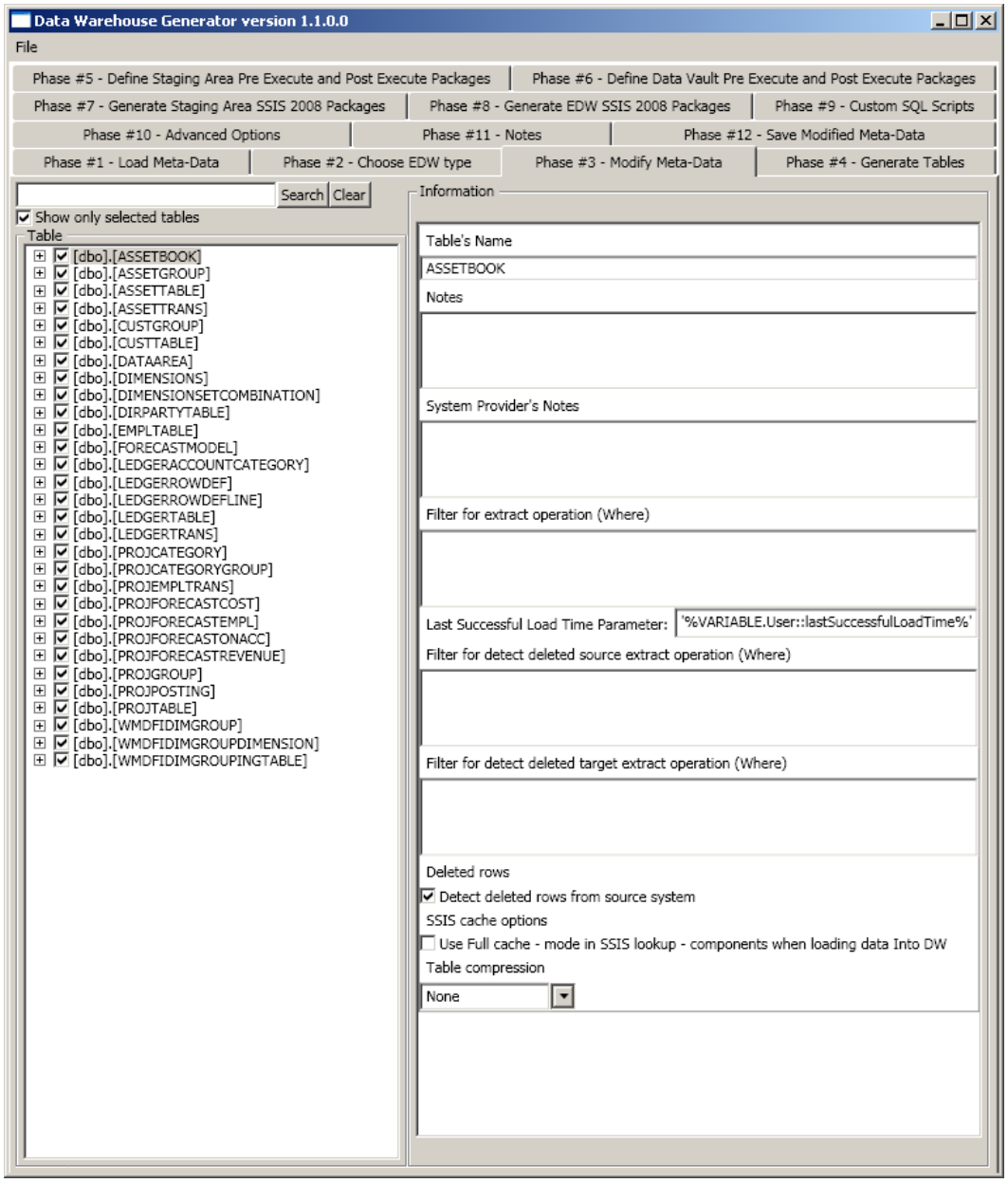
3.4 Generointisovelluksen tekninen toiminta

Kun sovellus on käynnistetty ja metadata ladattu, täytyy valita generoitavan tietovaraston tyyppi. Kuviossa 14 on valittu EDW-tyypiksi eli Enterprise Data Warehouse -tason tietovaraston tyypiksi ”data vault modeled EDW”. EDW-tyypin tietovarastoilla tarkoitetaan kaikkein monimutkaisimpia rakenteita tukevia ja historioinnilla varustettuja tietovarastotekniikoita, kuten Kimballin tähtimallia tai data vaultia. DV-valinta tuottaa siis Lindstedtin mukaisen data vault -tietovaraston ja DM-valinta Kimballin mukaisen dimensionaalisen tietovaraston esimerkiksi datamartteja varten.



KUVIO 14. Tietovaraston tyypin valinta

Sovelluksessa on mahdollisuus selata tarkemmin taulujen tietoja sovelluksen kolmannella välilehdellä, joka on esitetty kuviossa 15. Tauluille voi kirjoittaa ja tallentaa muistiinpanoja ja lähdejärjestelmän ylläpitäjän kommentteja kustakin taulusta, kuten kuviostakin näkyy. Kuviossa on valittu osa Microsoft Dynamics AX -toiminnanohjausjärjestelmän oletuksena luomista tauluista mukaan tietovarastoon.



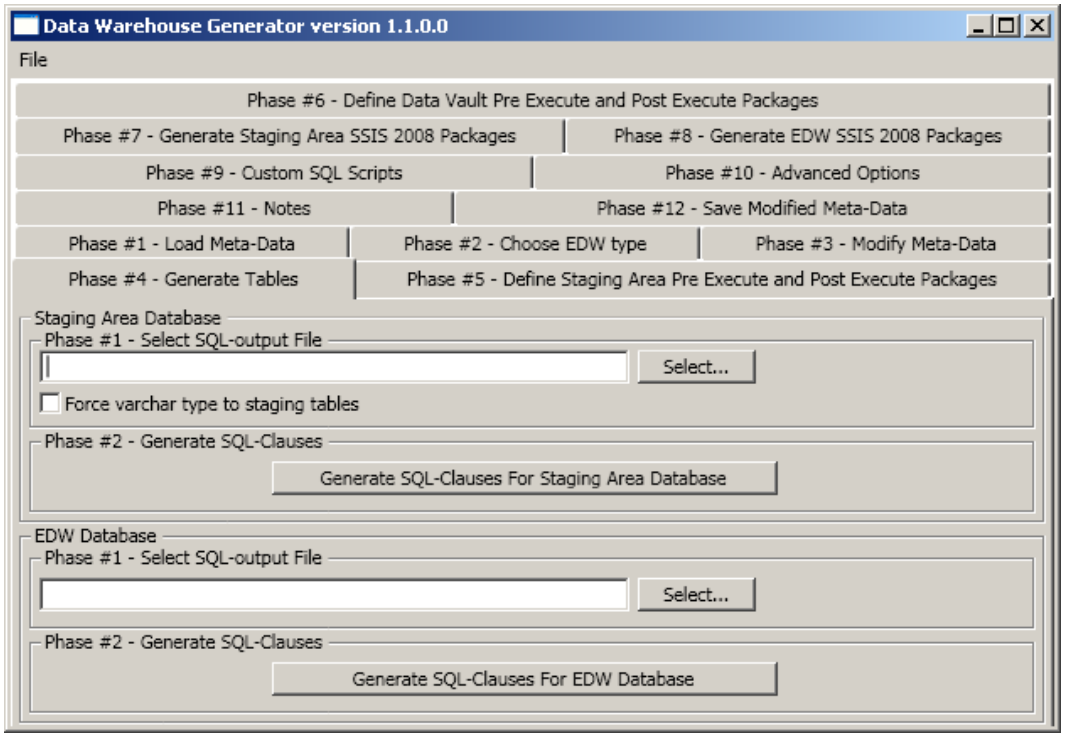
KUVIO 15. Tietovarastoon tulevien taulujen valinta

Jos ohjelmalle tulee vastaan jokin aiemmin tuntematon tietokantatyyppi, täytyy myös PK- ja FK-liitokset asettaa käsin jokaiseen tauluun. Ohjelman metadataluokkaan on kuitenkin toteutettu kohta taulujen PK- ja FK-tietojen automaattiselle tunnistukselle sekä Sybase SQL -tietokantaan versiolle 10, että yleisempi tietokannan järjestelmätauluja lukeva algoritmi muille kannoille. Tämä yleisempi algoritmi tukee Microsoftin SQL Serveriä ja muita samanlaisia tauluja käyttäviä kantoja. Modulaarinen rakenne mahdollistaa helposti uusienkin tietokantojen tunnistusten lisäämisen. Automaattisen tunnistuksen toteutuksen

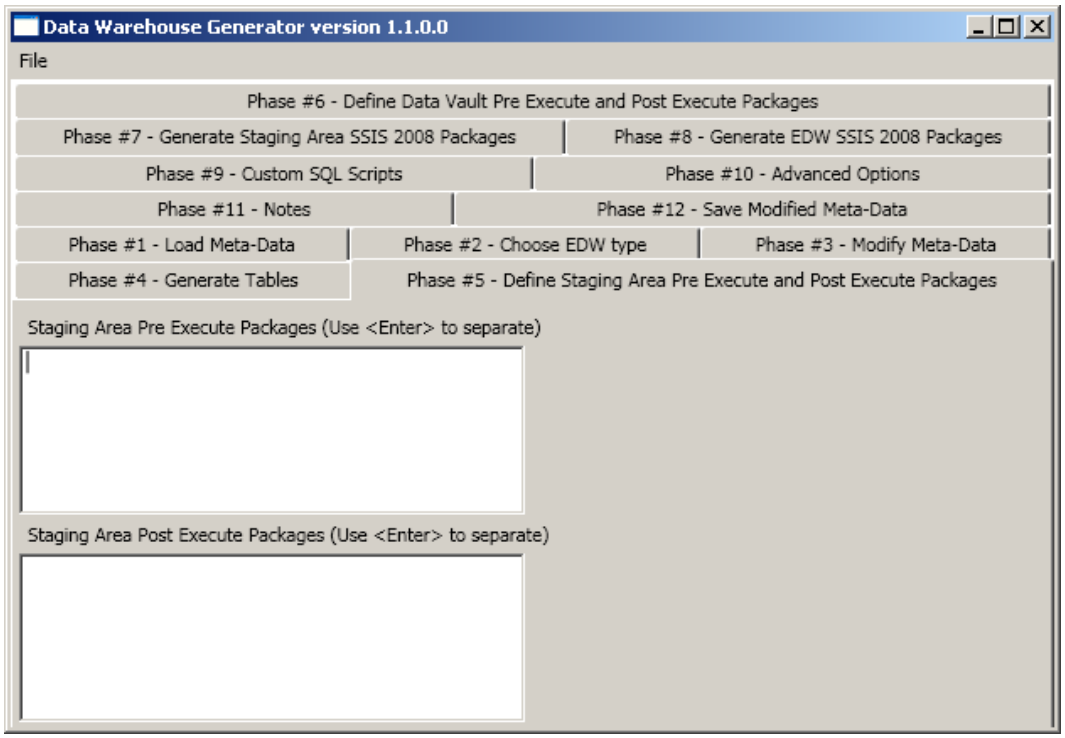
lisääminen sovellukseen vähensi entisestään käsityön määrää, kun tietoja ei tarvitse enää laittaa käsin muissa kuin vanhemmissa tietokantatuotteissa. Jos algoritmi tunnistaa kannan rakenteet, se myös tallentaa tiedot metadataan.

Seuraavaksi sovelluksessa voidaan luoda seuraavalla välilehdellä SQL-tiedostot työ- ja DV-alueelle, jotka sisältävät luontilauseet tietokannan tauluille kyseisille alueille kaikkine avain- ja NULL-tietoineen. Tehdyt SQL-lauseet tallennetaan annetun käyttöliittymän kautta erikseen työ- ja DV-alueille. Tiedostot on tarkoitettu käyttäjän tai järjestelmän toimittajan ajettavaksi haluttuun tietokantaan. Tiedostojen hyviä puolia on myös mahdollisuus ajaa skriptit tarvittaessa useasti tai useaan koneeseen.

Kuviossa 16 on näytetty välilehti, jossa voi generoida taulujen SQL-luontilauseet. Tekstikenttiin annetaan tiedostopolku, johon SQL-komentojono tallennetaan. Kuviossa 17 puolestaan on välilehti, jossa voidaan määrittää ajoketjuun mukaan käsintehtyjä SSIS-lataajia. Nämä lataajat voidaan määrittää ajettavaksi joko ennen generoitua ajoketjua (pre execute) tai ajoketjun jälkeen viimeiseksi (post execute). Kyseisellä välilehdellä määritetään esi- ja jälkiajoon lataajat työalueen ajoketjuun ja ohjelmassa on toinen välilehti, Phase #6, data vaultin ajoketjun vastaaviin määrityksiin.



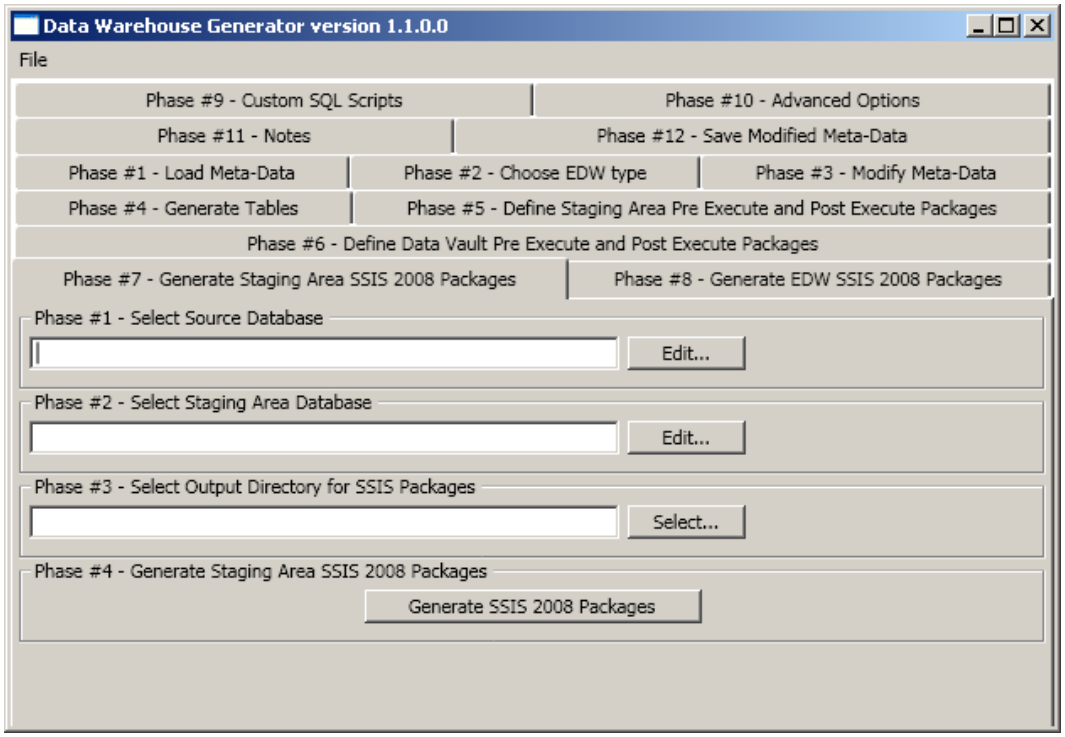
KUVIO 16. SQL-lauseiden generointi



KUVIO 17. Esi- ja jälkiajolataajien määritys

Sovelluksessa päälogiikka on työ- ja data vault -alueiden Microsoft SSIS 2008 -lataajakomponentteja tuottava sovelluslogiikka, joiden ohjelmakoodi vie suurimman osan koko koodista. Koska data vault -tietokannat tuli Logicalla toteuttaa sopimusten ja päätösten vuoksi Microsoftin ohjelmistoilla ja välineillä, oli SSIS-komponenttien luonnollinen valinta tietojen siirtotavan toteuttamiseksi.

Kuviossa 18 on esitetty työalueen SSIS-lataajien generointivälilehti. Kyseiselle välilehdelle syötetään lähdejärjestelmän yhteysmerkkijono (connection string), joka voi osoittaa tietokantaan tai tiedostoon. Sen voi asettaa joko kirjoittamalla tai helpommin edit-napista avautuvasta ikkunasta. Seuraavaksi syötetään SA-kannan yhteystiedot sekä kolmanneksi annetaan tiedostopolku, jonne SSIS-lataajat tallennetaan. Viimeisestä napista painamalla käynnistyy lataajien generointi metadatan ja valintojen perusteella. Sovelluksen välilehdeltä 10 voidaan asettaa generointi tapahtumaan sarja-ajona, jolloin luodaan yksi lataaja kerrallaan tai rinnakkaisajona, jolloin lataajat luodaan omissa säikeissään tietokoneen koko prosessointiteho hyödyntäen. Rinnakkaisajo on huomattavasti nopeampi tapa luoda paketit. Välilehdellä 8 on vastaavat asetukset DV-alueen SSIS-lataajien luomiseen. Annettavat tiedot ovat SA- ja DV-alueen yhteysmerkkijonot ja DV-lataajien tiedostopolku.



KUVIO 18. Työalueen SSIS-lataajien generointi

Lopputuloksena SSIS-pakettien generoinnista tulee kymmeniä lataajapaketteja, dtsx-päätteisiä tiedostoja. Vaikka valittaisiin vain 5 taulua mukaan, voi lopputuloksena syntyä jopa 20 tiedostoa. Jokaiselle valitulle taululle luodaan yksi hub-lataaja, hubin satelliitin, detect deleted -osuuden ja data sink -osuuden lataajat sekä linkitysten määrästä riippuen yksi, useampi tai ei yhtään linkkilataajaa. Paketit luodaan hierarkiaan, sillä pakettien ajojärjestyksellä on merkitystä lopputuloksen toimivuuden kannalta. Paketit on nimetty numerolla aloittaen niin, että numerot 1-3 kuvaavat kolmea eri ajoketjun tasoa. Ylimmällä tasolla eli ”1_”-alkuinen paketti on tasollaan yksin ja se sisältää logiikan ajaa kaikki seuraavan tason lataajat, vaikka yksittäinen tai useampi lataajan suoritus epäonnistuisi välissä. Seuraava numerolla 2 alkavat lataajat vastaavat kukin yhdestä tärkeästä data vaultin tekijästä: kantaan täytyy ajaa uusien ja muuttuneiden hub-tietojen rivit ja niiden satelliittien tiedot sekä tietoa yhdistävien linkkien ja niiden satelliittien muuttuneet tai uudet tiedot. Hub-tietojen jälkeen on itse asiassa vielä malliin muualta lisätty tapa yrittää ladata data sink -tauluihin menneet rivit uudestaan DV:hen.

Data sink on ikään kuin työskentelyallas datalle sen data vaultiin siirron aikana ja koska INSERT-lauseet ajetaan suorituskyvyn takia kantaan 10 000 rivin erissä, on nopeampaa pudottaa altaaseen eli kirjoittaa data sink -tauluun yksi kokonainen erä. Tallentaminen erissä on nopeampaa, sillä MS SQL:n erätallennustekniikka mahdollistaa suurille rivimäärille INSERT-operaation vain muutamalla SQL-käskyllä. Esimerkiksi 10 000 rivin erän tallennus vaatii vain kymmeniä kyselyjä, kun rivikohtainen tallennus vaatisi 10 000 SQL-lauseen ajoa.

Näistä data sinkissä olevista eristä voi virheellisiä rivejä, jotka eivät uudelleen yrittämälläkään mene kantaan, olla 1-10 000 kappaletta. Jo yksikin rivi pudottaa koko erän data sinkkiin, mutta sen läpikäyminen ja kelvollisten rivien lataus altaasta DV:hen on silti nopeaa. Testien mukaan tämä on nopeampaa, kuin jos rivit lisättäisi data vaultiin rivi kerrallaan ja vain yksittäiset virherivit lisättäisi data sinkkiin. Data-altaan tyhjennysnopeus riippuu virheellisten rivien määrästä, sillä rivit yritetään lisätä DV:hen rivi kerrallaan ja jokainen virhe hidastaa suoritusta hieman. Kun näin on käyty kaikki rivit hylätyistä eristä läpi, on tietotyyppien, virheellisten merkkien kannalta tai muun teknisen asian kannalta virheettömät rivit tietovarastossa. Edelleen altaaseen jääneet, jollakin tavalla virheen INSERT-lauseessa aiheuttavat rivit, merkitään läpikäydyiksi odottamaan tarkastelua. Jos virheelliset rivit korjataan lähteessä, eivät ne enää joudu data sink -käsittelyyn, vaan menevät suoraan tietovarastoon.

Näiden jälkeen on olennaista ajaa viimeinen osa latausketjusta, jotta lähteestä poistuneet tiedot heijastuisivat data vaultiin ja se pysyisi ajan tasalla. Nämä ”detect deleted -rows”-logiikaksi nimetyt ketjut merkitsevät kustakin taulusta lähteestä poistuneet rivit vanhentuneiksi eli niille annetaan kuluvan hetken aikaleima. Kun tämän jälkeen haetaan rivit, joissa päättymisaikaleima on NULL, ovat ne voimassa olevia rivejä, joita voidaan käyttää tiedon jalostuksessa analyyseiksi ja raporteiksi.

Microsoftin SSIS-lataajat mahdollistavat kaikkien suoritettujen operaatioiden laajan lokituksen, jolloin voidaan ajojen päätyttyä helposti yhden taulun sisältöä kyselemällä saada selville latausketjun ajoaika, mahdolliset virheet, virhelataajien nimet ja ajon aikana seurattuna se kertoo etenemisasteen. Data-altaaseen jääneet rivit täytyy myös käydä manuaalisesti läpi ja mahdollisesti raportoida

lähdejärjestelmistä vastaaville henkilöille, jotta virheelliset tiedot osataan korjata. Data vaultin sisältöä kun ei saa muokata suoraan.

4 YHTEENVETO

Projektina oli toteuttaa proof of concept –toteutus data vault –tietovaraston generoijasta uusiksi huolellisesti suunniteltuna sovelluksena. Tuloksena syntyneen DW-generaattorin vaadittiin asiakkaan määrittelystä tuottavan SQL-luontilauseet ja SSIS-lataajat data vault –mallin mukaiselle tietovarastolle. Valitun toteutustavan täytyi olla modulaarinen ja helposti ymmärrettävä, jotta sitä voidaan jatkossa kehittää eteenpäin helposti ilman toista koko sovelluksen uudelleenohjelmoimista.

Lopputuloksena syntynyt, rakenteeltaan uusiksi kirjoitettu, generointisovellus täytti kaikki Logicalta asetetut vaatimukset ja valmistui kuukautta annettua aikaa etuajassa eli noin kolmessa kuukaudessa kalenteriajassa mitattuna. Uusiksi suunniteltu arkkitehtuuri toi paljon modulaarisuutta lisää ja helpottaa jatkossa kehitystä, kun eri toiminnot ovat hajautettuina omiin luokkiinsa ja tiedostoihinsa. Luokkiin upotetut SQL-kyselyt kasvattivat luokkatiedostojen kokoa reilusti, mutta olivat omissa funktioissaan ja hyödynsivät mahdollisimman paljon muuttujia. Koska sovelluksen tarkoitus on luoda SQL-tietokantataulujen luontilauseet, ei seassa oleva SQL-koodi oleellisesti häiritse koodin luettavuutta. Myös ohjelman muistinkulutus oli PoC-totetusta pienempi, kun projektin loppupuolella jäi aikaa optimoida ohjelmakoodia. Kommentoinnin määrä oli paljon suurempi kuin PoC:ssa, mutta enimmäkseen koodi on itsensä dokumentoivaa, sillä .NET-rajapinnan metodit ovat nimeltään pitkiä ja käyttötarkoitusta kuvaavia.

Tällä hetkellä ohjelmaa käytetään Logican sisäisen tietovaraston uudistamis- ja kehittämisprojektissa ja samalla listataan käytön yhteydessä tulevat uudet tarpeet jatkokehitystä varten. Sovelluksen käyttö on käytännössäkin vähentänyt tietovaraston toteutukseen kuluvaa aikaa ja helpottanut huomattavasti muutoksiin sopeutumista data vaultin ja sovelluksen luonteen ansiosta. Sen käyttö kuitenkin edellyttää data vault –teorian hyvää tuntemusta, sillä ohjelman avulla on tehty yksi epäonnistunutkin testiympäristön toteutus, kun sen teki henkilö, joka ei tuntenut kyseistä tietovarastointimallia eikä sovelluksen käyttöä. Käyttöä varten tarvitseekin kouluttaa erikseen ihmisiä ja tämän vaatimuksen vuoksi sovellusta ei osaa käyttää tällä hetkellä muut kuin sen kaksi tekijää.

Huolimatta siitä, että sovellus valmistui annettua aikataulua nopeammin ja täytti aluksi määritetyt vaatimukset sekä annetut testit, ei sovelluksen lopputuloksesta ole saatu juurikaan palautetta. Syynä tähän saattoi olla tutorin ja esimiehen muut kiireet ja se, että vasta varsinaisessa käytössä tulevat puutteet paremmin ilmi tai jatkokehitysehdotukset mieleen kuin lyhyellä testikäytöllä osasta tuotantodataa.

Jos toteuttaisin sovelluksen nyt uusiksi, toteuttaisin käyttöliittymän ulkoasun eri tavalla. Itse rakenne toimii ja tukee muitakin palvelintekniikoita kuin Microsoftin SQL Serveriä ja mahdollistaisi tarvittaessa suurimman osan SSIS-lataajien toiminnoista toteutettavan SQL-lauseilla. Vikasietoisuuden ohjelmoimiseen SSIS-lataajien sisään tietovuon käsittelyyn saattaisin panostaa enemmän, sillä siihen panostaminen kasvattaa latausten toimintavarmuutta. Eniten käyttöä on kuitenkin hankaloittanut sovelluksen epäintuitiivinen ulkoasu. Käyttäisin välilehtien sijaan listausta sovelluksen vaiheista vasemmassa reunassa ja oikealla puolella näytettäisi valintaa vastaava toiminto. Toiminnon yllä voisi olla kuvaukset, mitä se tekee ja mitä vaaditaan, jotta toimintoa voidaan käyttää.

Sovellusta tullaan kehittämään vielä lisää jatkossa, jotta ohjelman toiminnan vikasietoisuuden parantamisen lisäksi myös virheellisestä datasta aiheutuvat tilanteet voidaan käsitellä paremmin SSIS-lataajissa. Käyttöliittymääkin tullaan parantamaan käyttäjien palautteen myötä helppokäyttöisemmäksi ja sisältämään yleisimmin käytetyt optiot valmiiksi päälle kytkettyinä. Tällaista tietoa ei pystytty sovelluksen kehitysvaiheessa keräämään, sillä idea sovelluksen käytöstä oli melko uusi eikä sen käyttäjäryhmää ollut kohdennettu.

Data vault ja muut tietovarastointikeinot tulevat hyvin todennäköisesti olemaan kasvava trendi tiedon hallinnassa ja pohjana vaativammille analysointi- ja raportointitarpeille. Logican ratkaisu automatisoida rakenteiden luominen nopeuttaa teknisen puolen toteutusta ja nopeuttaa asiakkaiden muutoksiin sopeutumista sekä jättää enemmän aikaa asiakkaiden liiketoiminnan prosessien kartoittamiseen ja raportointiin. DV-mallia ei kuitenkaan pidä käyttää uusissa projekteissa ja asiakkuuksissa vain siksi, että se on nyt mahdollista, vaan siksi, että se vaaditaan ratkaisun toimimiseksi.

Logican ja kilpailijoiden onkin tärkeää kehittää henkilöstönsä tietovarasto-osaamista, jotta ne pärjäävät lähitulevaisuudessa asiakkaista kilpaillessa. Se yritys, jossa ymmärretään tietojen varastointi, mallinnus ja tiedon metadata abstraktilla tasolla hyvin, osaa suunnitella paremmin tietovarastosovelluksia ja varautua muutoksiin, kun nyt käytössä olevat tekniikat vanhenevat ja uudet tulevat tilalle. Toisin sanoen tällä hetkellä ohjelmistokehityksen sitominen tietyn valmistajan, kuten esimerkiksi Microsoftin, ohjelmistoihin olisi lyhytnäköistä ja teettäisi pidemmällä tähtäimellä enemmän työtä, kuin alun perin jo abstraktilla tasolla tekniikkariippumattomasti toimivan tekniikan kehittäminen.

LÄHTEET

Hovi, A., Koistinen, H. & Hervonen, H. 2009. Tietovarastot ja Business Intelligence. Porvoo: WSOY.

Korhonen, T. 2010. The choosing of BI reporting tool to present DW data. Aalto-yliopisto, informaatio- ja luonnontieteiden tiedekunta. Diplomityö. Espoo.

Lindstedt, D. 2002. Data Vault Series 1 – Data Vault Overview [viitattu 21.1.2012]. Saatavissa: http://www.tdan.com/view-articles/5054/

Mishra, S. 2009. Data Compression: Strategy, Capacity Planning and Best Practices [viitattu 24.4.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/dd894051%28v=sql.100%29.aspx

Wikipedia. 2012. SQL Server Integration Services [viitattu 24.4.2012]. Saatavissa: http://en.wikipedia.org/wiki/SQL_Server_Integration_Services

Rämö, A. 2010a. Data Vault tietokannan generointi – ratkaisukuvaus v.1.4, yrityksen sisäinen dokumentti

Rämö, A. 2010b. DW generoijan toteutuksesta, yrityksen sisäinen dokumentti.

LIITTEET

LIITE 1. Lindstedtin lista data vaultin luomissäännöistä

The Data Vault should be built as follows:

1. Model the Hubs. This requires an understanding of business keys and their usage across the designated scope.
2. Model the Links. Forming the relationships between the keys – formulating an understanding of how the business operates today in context to each business key.
3. Model the Satellites. Providing context to each of the business keys as well as the transactions (Links) that connect the Hubs together. This begins to provide the complete picture of the business.
4. Model the point-in-time tables. This is a Satellite derivative, of which the structure and definition is outside the scope of this document (due to space constraints).

Reference rules for Data Vaults:

1. Hub keys cannot migrate into other Hubs (no parent/child like Hubs). To model in this manner breaks the flexibility and extensibility of the Data Vault modeling technique.
2. Hubs must be connected through Links.
3. More than two Hubs can be connected through Links.
4. Links can be connected to other Links.
5. Links must have at least two Hubs associated with them in order to be instantiated.
6. Surrogate keys may be utilized for Hubs and Links.
7. Surrogate keys may not be utilized for Satellites.
8. Hub keys always migrate outward.
9. Hub business keys never change, Hubs primary keys never change.
10. Satellites may be connected to Hubs or Links.
11. Satellites always contain either a load date-time stamp, or a numeric reference to a stand-alone load date-time stamp sequence table.
12. Stand-alone tables such as calendars, time, code and description tables may be utilized.
13. Links may have a surrogate key.
14. If a hub has two or more satellites, a point-in-time table may be constructed for ease of joins.
15. Satellites are always delta driven, duplicate rows should not appear.
16. Data is separated into Satellite structures based on: 1) type of information 2) rate of change.

These simple components Hub, Link and Satellite combine to form a Data Vault. A Data Vault can be as small as a single Hub with one Satellite, or as large as the scope permits. The scope can always be modified at a later date and scalability is not an issue (nor is granularity of the information)