Haaga-Helia
ammattikorkeakoulu Oy

ETL-prosessin suunnittelu

Janne Kemppainen

Opinnäytetyö
Tietojenkäsittely
2015

Abstract

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Janne Kemppainen

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
ETL-prosessin suunnittelu

Sivu- ja liitesivumäärä
22 + 1

Tiedon muokkaamisessa käytetään ETL-prosesseja jotka automatisoidaan ja tieto päivitetään omien tarpeiden mukaan. Opinnäytetyössä käydään läpi prosessin eri vaiheita.

Opinnäytetyö on kevyt käsikirja, jonka avulla aloittavat prosessin kehittäjät pääsevät alkuun. Prosessin pilkkominen hallittaviin kokonaisuuksiin onnistuu tuntemalla kuinka tietoa hyödynnetään eri vaiheissa. Tämä prosessin pilkkominen helpottaa virhetilanteissa ja ongelmien korjaamisessa.
Vaiheittainen prosessi on tarpeellinen hajanaisen infrastruktuurin myötä, kaikkia asioita ei kannata tehdä yhdessä työssä.

Prosessin suurena osana on tietojen rikastaminen ja muokkaaminen. Tietoa voidaan muokata tietokannassa tai ETL-työkaluilla. ETL-työkalua valittaessa omien tarpeiden mukaan on asioita jota tulee pohtia. Näissä työkaluissa on eri ominaisuuksia, mahdollisuus on muun muassa valita avoimen lähdekoodin tai suljetun koodin välillä. Tietojen muokkaamiseen tulee työkalun mukaan eri komponentteja joita voidaan hyödyntää.

Valmiin prosessin muokkaaminen on haasteellinen tehtävä kun dokumentointi on olematon ja infrastruktuurin ollessa hajanainen. Uudelleen kehittäminen vaatii koko prosessin vahvaa tuntemista. Samalla on otettava huomioon kehittämisen liittyvät tekniset ja taloudelliset rajoitukset. Prosessin työt on hyvä suunnitella käyttämään vahvuuksia infrastruktuurissa. Ylläpidon tulee kuitenkin olla helppoa ja täytyy onnistua myös muiden toimesta.

Valmiin prosessin muokkaaminen vaati alkuun vanhan prosessin kuvaamisen, ja sovelluksen tietomallin purkamisen tietokannasta. Tulevat muutokset on syytä kehittää jo olemassa oleviin prosesseihin. Prosessin päivitys muodostuu uusista moduuleista jotka voidaan liittää sellaisenaan olemassa oleviin töihin.

Asiasanat
ETL-prosessi, kehittäminen, ETL, integraatio

Abstract

Haaga-Helia ammattikorkeakoulu Oy

Author(s)
Janne Kemppainen

Degree program
Business Information Technology

Thesis title
Planning ETL-process

Page- and attachmentpage
22 + 1

Automatic ETL-processes are used to transform data to own purposes. This thesis points out things that needs to be considered during developing processes.

This thesis is a light manual for fresh ETL-developers to get started. Knowledge of the data helps it to split the process to manageable pieces. Phasing the process helps in the situation on errors, and fixing them. A phased process is necessary also in a scattered infrastructure. All things in a process isn't worth doing in one job. During the process the data is enriched and transformed. The transform can be done in the database or with the ETL-tool.

Choosing a proper ETL-tool to cover the requirements there are the things that needs to take in consider. ETL market is filled with open- or closed source products. ETL-tools have many different components to transform the data, and each component has its own purpose.

Modifying a process is a challenging task when the documentation is slim and the infrastructure is scattered. Further development requires knowledge of different parts in the infrastructure. The process is good to plan to use the strengths in the systems. However the maintenance should still be easy and others manage it.

To model a ready process without any documentation requires reverse engineering from the database. Describe the process and the future integrations. The modifications are implemented to the old processes.

Asiasanat
ETL-prosessi, kehittäminen, ETL, integraatio

Sisällys

1 Johdanto

Opinnäytetyössäni on tarkoitus osittain käydä läpi eri vaiheita ETL-prosessin kehittämistä. Kaikki teknisiä asioita ei ole tarkoitus käydä läpi. Opinnäytetyössä käydään muutama tekninen asia johon olen törmännyt työssäni. Kehittämäni prosessi on käytössä, mutta kuvaus on tarpeeksi yleinen jota voi soveltaa monessa eri paikassa.

Tiedon kerääminen oman päätöksentekoon kasvaa koko ajan. Tietoa on saatavilla paljon eri paikoista. Yksittäinen tietue voi olla itsessään käyttökelvotonta mutta muokattuna tai rikastettuna se voi olla arvokasta. Kerätessä tieto omiin tietosäilöihin tieto on itsessään arvotonta. Yhdistämällä ja rikastamalla tieto voi muuttua arvokkaaksi. ETL-prosessilla voidaan tietoa muokata omiin liiketoiminta tarpeisiin. Prosessi voidaan kehittää eri tavoin ja prosessin aikaiset tehtävät voidaan jakaa eri osiin tai voidaan suorittaa eri paikoissa.

Automaattisia ETL-päivitysprosesseja kehitetään usein siellä missä tietoa liikkuu paljon ja siihen kohdistuu suuria määriä muutoksia ja sääntöjä. Prosessit liittyvät joko yhteen tai useampaan tietotarpeeseen. Tiedon alkuperä tai lähde saattaa tuottaa tietoa moneen eri tarpeeseen liiketoiminnalle ja näin ollen prosessitkin jakautuvat tarpeen mukaan. Tiedon käyttö ei ole opinnäytetyön tarkoitus vaan tarkoitus on perehtyä muutamaan ongelmaan yhden prosessin kehityksen aikana.

Kehittäessä prosessia on monia seikkoja jota tulee ottaa huomioon. Prosessin eri vaiheet ja tiedon muokkaamisessa tarvittavat resurssit vaihtelevat eri tarpeista. Vaiheet ovat teknisiä liittymiä jotka suoritetaan yksittäisinä töinä tai ketjutetaan yhteen sarjaan. Teknisissä töissä arkkitehtuuri ja eri tekniset rajoitusten vaikuttavat kuinka prosesseja kehitetään.

Tarpeet integroida tietoa eri lähteistä tulevat liiketoiminnalta, harvoin IT-osasto rupeaa toteuttamaan automaattisia päivityksiä omiin tarpeisiin. Liiketoiminnalle tärkeä asia on esimerkiksi Master Data Management (MDM). MDM on yrityksen avaintietoa. Avaintieto saattaa olla esimerkiksi asiakastiedot. Asiakastiedot saattavat sijaita eri järjestelmissä, ja järjestelmillä ovat omat tietokannat. Tietokanta rakenteet ovat erilaiset, ja tietomallit saattavat poiketa hyvin paljon toisistaan. ETL-prosesseja käytetään tietojen muokkaamisessa yhtenäisen tietomallin mukaiseksi.

Opinnäytetyössä on tarkoitus käydä läpi eri mahdollisuuksia kehittäessä automaattista ETL-prosessia. Eri asiat vaikuttavat yksittäisten töiden suorituskykyyn ja hallintaan. Opinnäytetyössäni on tarkoitus hieman pohtia vaikuttavia asioita.

ETL-prosessista itsessään ei paljon kirjallisuutta löydy. Tieto ja taito ovat enemmän yleistä osaamista, kun tarvitaan osaamista monesta eri asiasta. Tietoa tarvitaan ohjelmoinnista, tietokannoista, palvelimista, tiedonsiirrosta jne.

Tavoitteena on peilata valmiiksi kehitettyä prosessia, ja kuinka hyvin se noudattaa eri tarpeiden parhaita käytäntöjä.

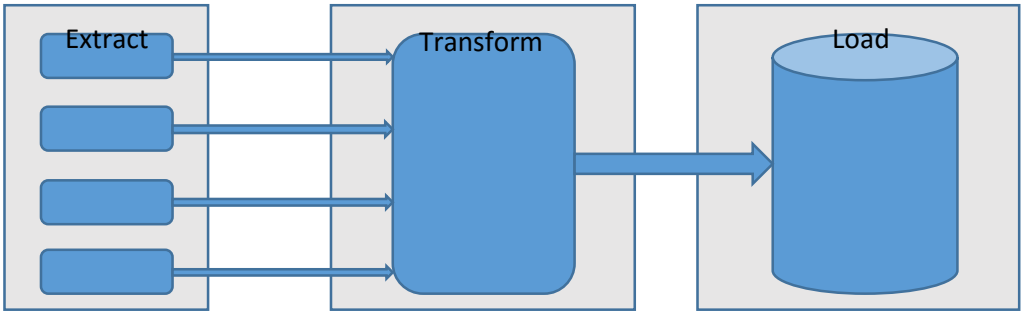
1.1 Sanasto

kohde	päivitettävä tietokanta
työ	työkalussa suoritettava yksittäinen työ
päätyö	yksittäinen työ työkalussa
alityö	yksittäinen työ työkalussa jota päätyö kutsuu
stage-taulu	tietokantataulu jota käytetään prosessin vaiheessa
työkalu	prosessissa käytetty väline integroida ja muokata tietoa
”truncate table”	poistaa taulun tiedot kaikki kerralla
UML	Unified Modeling Language, standardi mallinnuskieli

2 ETL

ETL on automaattisen tiedon muokkaamiseen tarkoitettu prosessi. Kyseessä ei ole ainoastaan liittymä joka siirtää tietoa. Tiedolle tehdään prosessin aikana monimutkaisia eri muutoksia, rikastuksia tai laskentaa. Usein prosessia käytetään kun lähteitä on useita ja tieto tulee yhdistää toiseen lähteeseen. Lähde voi olla tietomalliltaan eri muodoissa, prosessin aikana se muokataan vastaamaan omia tietotarpeita. MDM projekteissa käytetään monesti ETL työkaluja yhdistämään oman liiketoiminnan kriittinen tieto.

ETL (”Extract”, ”Transform” , ”Load”) kuvastaa automaattisen prosessin eri vaiheet (kuvio 1).
Vaiheilla on eri tehtävät automaattisen prosessin aikana. Prosesseja kehitetään monimuotoisen tiedon muokkaamiseen. Parhaan suorituskyvyn takaamiseksi laskentatehoja käytetään monessa eri paikassa.



Kuvio 1. Kuvaus prosessista

2.1 Extract

”Extract”-vaiheessa tieto ladataan lähteistä käyttöön jatkoa varten. Vaiheen tarkoitus on kerätä kaikki tarvittavat tiedot jota tarvitaan. Lähteinä voi toimia tietokannat, erilaiset tiedostot (xml, csv, määrämittaiset, jne), avoimet rajapinnat(esimerkiksi API) ja uutena Internet of Things (IoT).

Tätä tietoa voidaan kerätä eri tavoin lähteistä, riippuen siitä miten lähde tuottaa tietoa.
Tieto pyritään muokkaamaan tässä vaiheessa muotoon jossa attribuutit ovat omissa kentissä tietokannassa. Tässä vaiheessa aineistoa myös siivotaan käyttökelpoiseksi seuraavaan vaiheeseen. Siivoamiseen kuuluu muun muassa viitearvojen lisääminen, nimet puhdistetaan ”JANNE KEMPPAINEN” > ”Janne” ”Kemppainen”, ”mies” käännetään ”male” tai ”m” tai muotoon joka on oman käyttötarpeen mukaan määritelty.

2.1.1 Inkrementaalinen päivitys

Tietueita kertyy lisää jatkuvasti lähdettä ladatessa. Aineistot on syytä ladata vanhimmasta alkaen, ensin ladataan aineisto joka on muodostunut ensin. Aineistojen aikaleimoja voidaan käyttää hyväksi tässä vaiheessa kun tutkitaan mikä on vanhin ja mikä uusin. Ladattua aineistoa ei saa ladata uudestaan. Uudelleen lataus voidaan estää käyttämällä esimerkiksi aineiston MD5 summa. Lataus on nopeaa kun kaikki rivit kirjoitetaan suoraan tietokantaan sisään.

Esimerkiksi myyntitiedot joita käytetään laskennassa. Myytävän tuotteen tapahtumat lisääntyvät jatkuvasti, ja jokainen tapahtuma halutaan talteen.

2.1.2 Päivitys

Lähde tuottaa tietoa päivittämällä olemassa olevaa tietoa. Päivitys tapahtuu aineiston avaimilla. Tiedot luetaan vanhimmasta uusimpaan, päivittäen olemassa olevaa tietoa. Jos tietoa ei löydy avaimen perusteella, rivi kirjoitetaan uutena. Päivitys on hitaampi kun pelkkä sisäänkirjoittaminen, kun jokainen sisään kirjoitettava rivi verrataan olemassa oleviin riveihin.

Esimerkiksi henkilötiedot tuottavat päivityksiä. Kun henkilö muuttaa ja tiedoissa on henkilötunnus. Henkilötunnus toimii avaimena ja osoitetiedot voidaan muuttaa sen perusteella.

2.1.3 Kaikki tieto kerralla

Kaikki rivit kirjoitetaan joka kerta. Yleisesti kun kaikki tiedot tulevat kerralla lähde tuottaa omasta tietokannasta kaiken tiedon sen hetken tilanteesta. Päivitys saattaa olla hidasta riippuen tiedoston koosta. Ja tämä vaikuttaa suorituskykyyn, aineistosta riippuen tiedosta saattaa olla valtava. Ladatessa luetaan ainoastaan viimeisin tiedosto. Tietokantaan kirjoitetaan rivit aina vanhojen päälle. Ladattavat taulut voidaan tyhjentää ennen tietojen latausta joka nopeuttaa latausaikaa.

Esimerkiksi kun otetaan taulusta kaikki tieto yhteen tauluun valitsematta mitään pois. Usein tätä vaihtoehtoa käytetään kun ladataan uusi lähde tietokantaan. Kaikki tieto toimii pohjaaineistona.

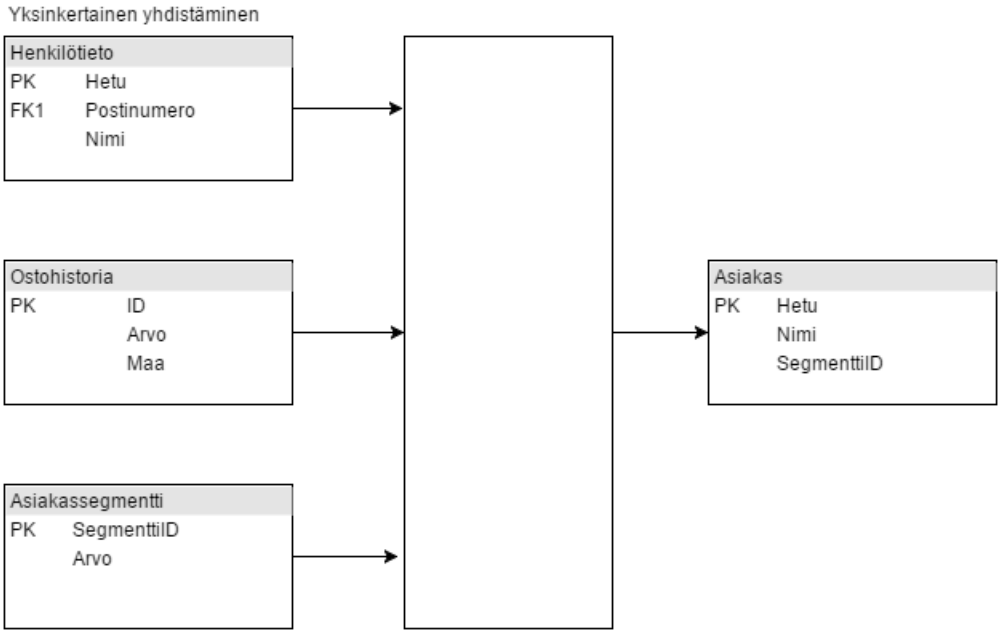
Se kuinka usein lähde tuottaa uutta tietoa vaikuttaa myös päivitystiheyteen. Usein tiedostot muodostuvat kerran vuorokaudessa. Kerran vuorokaudessa saatavat tiedot ladataan yleisesti eräajoina. Tietokantaliipaisimet ja esimerkiksi api-rajapinnat mahdollistaa rivikohtaisen päivityksen. Ja päivitykset voidaan suorittaa saman tien.
Lähteet tarvitset erilaiset tavat liittää prosessiin, riippuen missä muodossa ovat.

(ETL (Extract-Transform-Load))

2.2 Transform

Vaiheessa valitaan edellisen vaiheen siivottu tieto joka halutaan viedä lopulliseen kohteeseen. Kohteen entiteetit muodostuvat eri attribuuteista ja tässä vaiheessa yhdistäminen tapahtuu eri avaimien perusteella. Prosessin määrityksessä kerrotaan mistä eri lähteistä entiteetit muodostetaan. Laskennalliset toimenpiteet joka tiedolle tehdään, tapahtuu tässä vaiheessa. Aineisto muodostetaan avaimia myöten valmiiksi seuraavaan vaiheeseen.

Kuvio 2 kuvastaa Asiakas entiteetin muodostamisen kolmesta eri taulusta. Henkilötiedot muodostavat asiakkaan perustiedot. Asiakkaan ostohistoriasta lasketaan arvo, ja asiakas luokitellaan historian perustein segmenttiin.



Kuvio 2: Yksinkertainen yhdistäminen

2.3 Load

Vaiheessa muokattu ja rikastettu tieto viedään kohteeseen. Lataukset pyritään suorittamaan mahdollisimman tehokkaasti ja nopeasti. Latauksen nopeuteen vaikuttavia asioita pyritään tässä vaiheessa poistamaan, esimerkiksi avaimet usein poistetaan ja luodaan uudestaan valmiin päivityksen jälkeen.

2.4 Työkuorman jakaminen

Prosessien suorituskyky määräytyy monesta eri asiasta. Tiedon määrä, laatu ja käsittelysäännöt ovat lähtökohtaisesti asioita jotka vaikuttavat minne laskentateho sijoitetaan.
Tiedon määrän ollessa pientä voidaan toteuttaa monimutkaisia käsittelyitä yhden prosessin aikana ilman että suorituskyky kärsii paljon. Käsittelyt voidaan toteuttaa tietokannassa tai antaa valitun työkalun komponentin suorittaa käsittelyt.

Tietokannassa tehdyt käsittelyt ovat tehokkaita. Lisäksi SQL-kielellä pystyy tekemään lähes kaikki tarvittavat tehtävät. Tietokannoissa on erilaiset suorituskyvyt riippuen valitusta tietokannasta ja tietokannan tilasta. Suorituskykyyn vaikuttaa myös tietokannan palvelimen tila sekä se kuinka tietokanta on määritelty. SQL-kielellä tehdyt kyselyt ja muokkaamiset saattavat muodostua raskaiksi tietokannalle ja onkin syytä tiedostaa tietokantojen käyttöaste. Transform-vaiheessa tietokantoja voi olla useampi. Lisäksi tiedon muokkaamisen esteenä tietokannassa voi muodostua tietokantojen sijainnit sekä tekemättömät integraatiot. Tietokantojen välillä tehdyt muokkaamiset ovat pääsääntöisesti huonoja vaihtoehtoja.

Työkalussa tehtävä käsittely vie kuormituksen työtä ajavaan tietokoneeseen. Muokkaaminen tehdään työkalun ohjelmointikielellä. Valitut rivit siirtyvät suorittavaan koneeseen ja muistinkäyttö lisääntyy. Mikäli suorittavalla koneelle on huono kapasiteetti, työ saattaa hidastua. Oikein määritelty työpalvelin kestää työn suorituksen. ETL-työkalussa on monia eri mahdollisuuksia muokata tietoa. Nämä valmiit komponentit antavat mahdollisuuden muokata tietoa eri tavoin. Eri komponenteilla on eri tarkoitus, ja ovat suorituskyvyltä erilaisia. Mikäli valmista komponenttia ei ole, voidaan kehittää oma komponentti joka suorittaa oman haluamansa muokkauksen. ETL-prosessi sisältää kolme eri vaihetta, mutta työt voidaan vaiheistaa useampaan eri vaiheeseen. Vaiheistukset voidaan jakaa eri tavoin jos tuntee prosessin.

Latausvaiheen nopeus riippuu monesta eri asiasta. Mikäli tietokanta on sama jossa tietoa muokataan, siirtämiseen ei kulu aikaa vaan nopeus riippuu tietokannasta ja tietomallinnuksesta. On kuitenkin suositeltavaa että muokkaaminen tehdään toisessa tietokannassa kun

tuotannollinen kanta sijaitsee. Jos tietokannat sijaitsevat eri palvelimilla latausvaiheen tietoliikenneyhteys voi hidastaa latausta. Jos muokkaamiset ja lataus suoritetaan samassa työssä, useampi tietokanta resurssi on käytössä. Kohteen tietokanta onkin hyvä eristää muokkaamisen ajaksi jotta lataus voidaan suorittaa omana erillisenä työnä kun muokkaaminen on suoritettu.

2.5 Virhetilanteita

Automaattisissa prosesseissa tapahtuu virheitä eri syistä. Virheet saattavat olla teknisiä tai inhimillisiä. Tekniset virheet rajoittuvat laite rikkoihin ja laitteiden vikoihin. Suurimmaksi osaksi virheet tulevat inhimillisistä syistä.
Päivittävä työ tulee kehittää että kukin suorittava komponentti kirjoittaa lokiin tapahtumat. Tämä mahdollistaa nopeamman virheen löytämisen.

2.6 Päivitystiheys

ETL-prosessia kehitetään ajettavaksi toistuvasti. Päivitystarve täytyy suunnitella sen mukaan miten tietoa käytetään ja miten päivityksiä on saatavilla. Usein päivitykset ajetaan päivittäin eräajoina. Eräajot ovat ajoitettu usein toimistoaikojen ulkopuolelle, koska ei haluta päivitysten vaikuttavan tiedon käytettävyyteen päivän aikana. Tietoja voidaan myös päivittää tiedon muuttuessa. Päivitystiheys määritellään ennen prosessin kehittämistä.

2.7 Teknisiä termejä

Automaattisissa päivityksissä on toistuvia asioita jotka teknisesti tulevat vastaan, ja niitä joutuu tai voi käyttää automaattisten päivitysten aikana.

2.7.1 Transaktio

Transaktio on yksi tai useampi tietokantatapahtuma sidottu toimenpideketjuun josta jokaisen on onnistuttava. Tapahtumaketju suoritetaan kokonaan tai ei ollenkaan. Mikäli yksi tapahtuma ketjusta ei onnistu kaikki edeltävät tapahtumat peruuntuvat tilaan jossa olivat aikaisemmin. Peruutetut tiedot luetaan takaisin transaktiolokista. Transaktion onnistumisen ja toipumisen edellytykset ovat että tapahtumissa ovat ACID-ominaisuudet

- Atomicity (jakamattomuus) Kaikki tai ei mitkään transaktion tapahtumista suoritetaan.
- Correctness (oikeellisuus) Transaktion tulokset ovat oikein muista ulkopuolisista tapahumista riippumatta.
- Isolation (eristyvyys) Transaktion suorituksen aikana muut eivät pääse tuloksiin kiinni.
- Durability (pysyvyys) Vahvistuksen jälkeen, kaikki kirjoitetut tiedot pysyvät tietokannassa. (Lahtonen 2002.)

2.7.2 Surrogaatti avain

Surrogaatti avain on tietokannassa automaattisesti muodostettu avain. Avain ei sisällä informaatiota. Avain on keinotekoinen.
(Turkumäki Toukokuu 2013)

Automaattisen avaimen muodostaminen tietokannassa, esimerkkinä Sql server luonti lauseke.

```
create table taulu (
ID int IDENTITY(1,1) PRIMARY KEY,
…
);
```

”Identity” luo juoksevaan numeroinnin. ”(1,1)” ensimmäinen numero kertoo mistä juokseva numerointi alkaa, pilkun jälkeinen numero kertoo kuinka monella juokseva numero kasvaa seuraavalle riville. ”Primary key” kertoo kentän olevan taulun pää avain.

2.7.3 Entiteetti

Entiteetti on asia tai konsepti josta voidaan kerätä tietoa yhteen.
(Entities)
Otetaan esimerkiksi entiteetiksi Henkilö. Henkilötieto saattaa sisältää attribuutit etunimi, sukunimi, katuosoite, katunumero, rappu, asunto, postinumero, paikkakunta, maa ja sähköposti. Attribuutteja voi olla monta erilaista, riippuen omasta tietomallista. Attribuutit kuvastavat entiteettiä

2.7.4 MD5 summa

MD5 summalla tarkoitetaan tiedostosta tai tekstistä laskettu 32 merkkinen tiivistesumma heksakoodi muodossa, joka muuttuu heti kun jokin merkki muuttuu. Summan tarkoitus on salata tiedoston tai tekstin selkokielinen lukeminen. MD5 summaa käytetään salasanojen tallentamisessa.
(Repo 2014)

3 Prosessin kehitys

ETL-prosessia kehittäessä tulee tuntea kehitettävän ympäristön infrastruktuurin. Moni asia vaikuttaa kuinka automaattiset prosessit suunnitellaan. Kehittäessä voidaan alkuun hyödyntää esimerkiksi ajatuskarttaa (liite 1).

Prosessi muodostuu monesta seikasta. Alkuun on syytä tietää miksi prosessi kehitetään. Ketä prosessi palvelee, ja mitä hyötyä tiedosta on. Mihin tietoa käytetään? Analysoidaanko tietoa vai onko tieto osa jotain sovellusta, ja kuinka sovellusta käytetään? Tieto saattaa olla yrityksen avain tietoa, Master Dataa. Tiedon saatavuus, ja kuinka usein tieto päivitetään kun osia tiedoista saadaan kerran vuorokaudessa ja kun taas osa tiedosta saattaa päivittyä sovelluksen kautta monta kertaa päivässä.

Tiedon määrä ja siihen tehtävät rikastukset tai muutokset vaikuttavat kuinka prosesseja kehitetään. Muutosprosessiin saatetaan sitoa hyvin monimutkaisia sääntöjä, jotka hidastavat tiedon käsittelyä. Säännöt voidaan kirjata ja piirtää UML kaavioihin, jonka perusteella tekninen kehitys voidaan toteuttaa.

Tiedon tuntemisella on suuri vaikutus päivitykseen suunnittelussa. Entiteettien rakentaminen eri attribuuteista saattaa olla riippuvaisia toisen prosessin suorituksesta. Töitä voidaan sitoa yhteen kutsumalla töitä onnistuneen työn jälkeen tai kutsumalla alityönä yhtä tai useampaa työtä. Töitä voidaan suorittaa samanaikaisesti kutsumalla päätyöstä alitöitä.
Tekninen ympäristö vaikuttaa suuresti miten päivitykset tehdään. Tietokannoilla on erilaiset suorituskyvyt, sen ollessa pieni lähtökohtaisesti. Tietokannan rakenne, sijainti ja palvelin johon tietokanta on asennettu vaikuttavat suurelta osin suorituskyvyn muodostumiseen. Tietokannan käyttöaste vaikuttaa prosesseihin, esimerkiksi siksi että tiedot ovat käytössä monessa prosessissa tai jokin sovellus käyttää tietokantaa. ETL-prosessien käyttämät tietokannat ovat hyvä eriyttää muiden käytöstä.

Rivien määrä vaikuttaa suuresti prosessien suunnittelussa, suurien massojen muokkaaminen saattaa kestää kauan kun pienet määrät suoriutuvat nopeammin.
Mahdolliset virhetilanteet tulee ottaa huomioon kehittäessä prosesseja. Virheitä tulee, ja ne voivat johtua monista eri asioista. Virhetilanteesta toipuminen on syytä suunnitella prosessissa. Virheet eivät saa estää kohteen tietokannan käyttöä. Hyvin suunniteltu prosessi edesauttaa virheen löytämisessä. Kun prosessi on suunniteltu hyvin, tekninen toteutus voidaan pilkkoa sopiviin palasiin ja muutosten tekeminen töihin on helpompaa.

Tekninen toteutus prosessille on hyvä toteuttaa niin että se on helppo ylläpitää. Yhdellä työllä suoritettu prosessi on monesti valtavan kokoinen ja sen ylläpito on huomattavan vaikeaa. Prosessi on syytä pilkkoa sopiviin kokonaisuuksiin, tiedot voidaan tallentaa eri vaiheissa ”stage” tauluihin. Pilkottu prosessi on myös helpompi virhetilanteessa uusia. Lisäksi yhden työn prosessi varaa tarvitsemansa resurssit työn suorituksen ajaksi jonka takia tämä ei ole suotavaa.
Työn epäonnistuessa jossakin vaiheessa tai jäädessä jumiin, resurssit ovat lukossa kunnes työ loppuu täysin. Työn jumittaessa tietokanta resurssit jäävät lukkoon, eikä muut työ pääse kyseisiin resursseihin kiinni. Helppo tapa pilkkoa prosessi on alkuun noutaa lähteet omana työnä, usein lähteet noudetaan ftp-siirrolla. Siirron tekemiseen on monia eri tapoja, jokaiselle lähteelle voidaan tehdä omat työt jotka noutavat tiedostot tai voidaan tehdä yksi työ joka noutaa jokaisen lähteen tiedostot.

Riippuen tiedostojen tiedon laadusta ja käytöstä voidaan puhdistaminen suorittaa tietokantaan ladatessa. Usein ladataan tiedostot sellaisena kun ovat tietokantaan. Jo ladatut tiedostot usein arkistoidaan omiin tiedostopalvelimiin. Arkistoidut tiedostot voidaan nimetä uusiksi esimerkiksi ”tiedosto.txt” => ”tiedosto20102015.txt”, numeroiden viitaten latauspäivämäärään muodossa ppkkvvvv (pp=päivä, kk= kuukausi, vvvv= vuosi). Tietojen arkistointi määritetään kun ETL-prosesseja otetaan käyttöön. Tiedostojen puhdistaminen tietokantaan lataamisvaiheessa edellyttää sääntöjen luomisen puhdistamiseen. Puhdistus säännöt riippuvat omista laatuvaatimuksista. Normaali puhdistus on esimerkiksi nimen puhdistaminen suurista kirjaimista pieniin kirjaimiin muodossa ”JANNE” => ”Janne”. Puhdistukset edellyttävät että kunkin tiedoston lataus tehdään erikseen. Lataustöiden määrä kasvaa sitä myöten suureksi. Lataukset voidaan suorittaa samalla työllä, jolloin puhdistusta ei yleensä tehdä tietokantaan ladatessa. Tavan valinta riippuu omasta tavasta hallita tietoa eri vaiheessa ja kuinka lähde tuottaa tietoa. Tiedoston muoto määrää usein myös kuinka lataus suoritetaan, usein lähde tuottaa omat tiedostot samassa muodossa. Tuolloin voidaan lataukset suorittaa lähteittäin.

Kaikkien tiedostojen lataaminen yhdellä työllä vaatii suuren määrän suunnittelua ja lataus on omalainen prosessi. Tietokantaa voidaan hyödyntää työn hallinnassa. Työhön voidaan lukea sisään tietokannasta tietoa muuttujiin. Ja muuttujat työssä määräävät työn kulun. Tietoihin voidaan merkitä esimerkiksi aktiiviset lähteet. Tietokannassa merkityt tiedot mahdollistavat että prosessia voidaan ohjata esimerkiksi omalla sovelluksella.

On nopeampaa kehittää alkupäässä tietojen siivoaminen erillisinä töinä kun tehdä siivoaminen samassa työssä jossa yhdistetään ja lasketaan. Erilliset työt ovat usein itsenään nopeampia suorituskyvyltä ja virheiden hallinta on helpompaa.

Suunnitelma tietojen ylläpitäminen eri vaiheessa on tärkeä osa prosessin jatkossa. Lähteiden latauksessa on syytä tietää tyhjennetäänkö latausvaiheen taulut aina ennen vai kerätäänkö sinne tietoa. Jos taulut tyhjennetään ennen latausta, suoritus on nopeampaa kun voidaan tyhjään tauluun ladata tietoa. Mikäli tauluja ei tyhjennetä ennen latausta, on hyvä merkitä rivin lataus ajankohta tai merkitä rivit jotenkin uudeksi. Uusien rivien merkitseminen helpottaa prosessin seuraavan vaiheen suoritusta. Tietokanta taulun sisältäessä miljoonia rivejä halutaan prosessin seuraavassa vaiheessa hyödyntää ainoastaan ne rivit jotka ovat viimeksi luettu sisään.

Prosessin seuraava vaihe on tehdä tiedon valinta ja muutos. Vaiheessa on tyypillistä muodostaa kohdetta vastaavia entiteettejä. Jos latausvaiheessa ei ole suoritettu puhdistusta, se suoritetaan tässä vaiheessa. Puhdistettu tieto voidaan ylläpitää omassa paikassa ennen sen vientiä eteenpäin. Muutokseen viedään ainoastaan valitut tiedot, ja valinta tehdään määrityksissä. Vaiheeseen liittyy paljon liitoksia sekä rikastamista eri lähteistä, tai lähteen omista tiedostossa keskenään. Vaiheessa voidaan käyttää omia tai yleisiä käytössä olevia viitearvoja. Omat viitearvot saattavat olla esimerkiksi asiakkaan taso, joka on laskettu asiakkaan ostamista tuotteiden arvosta ja muista kriteereistä. Viitearvoja ylläpidetään omissa tauluissa. Näissä lähteissä on jokin viitetaulun tieto sisällään.

Toinen esimerkki viitearvosta on maakoodit. Maakoodit halutaan ylläpitää omissa järjestelmissä yleisillä koodeilla. Lähteen tuottaessa maatunnukset selkokielisenä tekstinä, esimerkiksi ”suomi” tai ”finland” ja muunnetaan tieto yleiseksi koodiksi ”FI”. Selkokielinen teksti verrataan viitetaulussa oleviin tietoihin ja saadaan yleinen koodi lähteen tuottaman tekstin sijaan. Viitearvoja käytetään saadakseen omat tiedot siivottua. Lähde saattaa tuottaa selkokielisen tekstin väärin kirjoitettuna tai eri tavoin kirjoitettu. Suomi voi olla kirjoitettu monin eri tavoin vaikka kyseessä kuuluu olla sama teksti, kirjoitusasu saattaa vaihdella ”somi” / ”sumi” / ”soumi”. Entiteetit muodostuvat oman tietomallin mukaan, joka on syytä olla määritetty tarkalla tasolla.

Valmiit entiteetit voidaan ylläpitää omissa tauluissa, joka on oma vaihe tiedon ylläpidossa. Päivitykseen viedään ainoastaan ne rivit jotka ovat määritetty. Määrityksessä saattaa olla eri valintoja jotka suodattavat tietoa jotka viedään päivitykseen. Valintoja voidaan ylläpitää tietokannassa, joka mahdollistaa ylläpidon sovelluksesta. Tietue saattaa olla oikea mutta sitä ei haluta viedä kohteeseen esimerkiksi sen ollessa ei aktiivinen maa tai siitä ei haluta asiakaita kohteeseen jotka eivät vastaa omien viitearvojen kehyksiä. ETL prosesseissa on usein paljon eri valintoja tietueisiin jotka suodattavat päivitykseen vietävät tietueet. Valinnat saattavat muuttua ja sen takia on hyvä pilkkoa prosessi eri palasiin. Tietojen ollessa oikein,

ja ovat prosessin vaiheissa eri paikoissa, tiedot voidaan helposti viedä eteenpäin määritysten muuttuessa.

Kohteen lataus on prosessin näkyvin asia, valmiit tietueet viedään päivitykseen. Rivien määrä ratkaisee paljon kuinka lataus suoritetaan. Lähteistä ladattu tietue määrä saattaa olla hyvinkin suuri, ja prosessin aikana kohteeseen päivitettävien rivien määrä on voinut valintojen jälkeen pudota huomattavan paljon. Kohteen sijainti ja rivien määrä vaikuttaa paljon suorituskykyyn. Kohteen ollessa eri sijainnissa tulee ottaa huomioon tietoliikenne rajoitukset päivittäessä kohdetta. Eri sijainnin kohteella voidaan käyttää omia tauluja johon päivitettävät rivit siirretään ennen päivitystä. Taulut sijaitsevat samalla palvelimella kun kohteen tietokanta.

Transaktion käyttö prosessin aikana on hyvin suotavaa, oma päivitys halutaan että se onnistuu kerralla eikä muut pääse vaikuttamaan tulokseen. Noudatettaessa ACID periaatteita tietokannan kunto saattaa vaikuttaa prosessiin. Suurien rivimäärien prosessit saattavat viedä tietokannan suorituskyvyn äärirajoille. Ja mahdollinen päivitys saattaa loppua kesken transaktio lokin täyttyessä.

Prosessin aikana on hyvä saada läpinäkyvyyttä eri vaiheista. Jokaisen prosessin työn tulee kirjoittaa lokia. Lokitietoon voidaan valita mitä halutaan. Työkalun komponentit kirjoittavat lokiin suorituksen aikaista tietoa.

Prosessia kehittäessä ei ole yhtä oikeata tapaa toimia, arkkitehtuuri vaikuttaa kehitystyöhön. Prosessi voidaan teknisistä rajoitteista huolimattaa pilkkoa eri vaiheisiin. Nämä vaiheet ovat enemmän tiedosta riippuvaisia. Vaiheiden ollessa selkeät teknisten vikojen tullessa on helpompi uusia yksi osa prosessista. Eri vaiheiden työt muuttuvat määritysten muuttuessa, ja kaikkia määrityksiä ei voida hallita tietokannasta. Muutoksia ei tehdä olemassa olevaan työhön, vaan työstä tehdään uusi versio johon muutokset tehdään.

Päivitysprosessien kehityksen jälkeen työt viedään tuotantoon suoritukseen. Samalla uudet prosessit siirtyvät ylläpitoon. Tämän jälkeen tuotannossa olevia prosesseja ei kehitetä enää samalla versionumerolla. Muutoksia tehdessä tuotannossa oleva versio pysyy tuotannossa kunnes uusi versio on kehitetty.

Prosessia kehittäessä tulee ottaa huomioon ylläpidettävyys. Talend työkalussa on mahdollista kehittää prosessi jossa työssä on ainoastaan yksi komponentti joka suorittaa kaiken. Kustomoidulla komponentilla riskiksi muodostuu ylläpidettävyys, mikäli komponentin vaiheita ei ole dokumentoitu selkeästi. SQL syntaksi on vahva kieli jolla pystyy toteuttamaan

muutokset, mutta jos koko muutosprosessi on kirjoitettu yhteen saattaa ylläpito olla hankalaa.

3.1 ETL työkalun valinta

Ennen kun voi ruveta rakentamaan automaattisia ETL-prosesseja pitää valita jokin työkalu. Sitä valittaessa täytyy olla tietoinen omasta IT-arkkitehtuurista. Ja minkälainen työkalu sopii omiin tarpeisiin.

Avoimen ja suljetun lähdekoodin työkaluja on monia. Valitessa työkaluja on Mustakallion (2015) mukaan mietittävä kumpaa suosii avointa tai suljettua.

Avoimen lähdekoodin etuina ovat matalat tekniset rajoitukset, lisäksi tunnetut ongelmat on tunnistettu yleensä avoimissa yhteisöissä. Teknologiat perustuvat aikaisemmin tunnistettuihin ongelmiin. Myös valmiita komponentteja on suuri määrä. Kustannukset avoimissa työkaluissa perustuvat työkustannuksiin lisenssi maksujen puuttuessa, lisenssiversion ollessa kuitenkin mahdollinen.

Miinuksina avoimessa lähdekoodissa voidaan pitää luotettavuutta kun ei tunneta työkaluja tarpeeksi hyvin. Ongelmien ilmetessä osaamista tarvitaan enemmän, ja niihin on vaikeampi löytää ratkaisuja selvän dokumentoinnin puuttuessa. Suurten yritysten tarpeisiin on vaikeampi sopeuttaa avoimen lähdekoodin ratkaisua, ja osaavaa työvoimaa on vaikeampi löytää.

Suljetun lähdekoodin etuina ovat tuotteiden nopea ja helppo käyttöönotto. Tuotteet ovat valmiiksi versioituja, ja niihin löytyy kattavampi tuke ja tekijöitä helpommin. Dokumentointia löytyy helposti ja se on kattavaa.

Suljetun lähdekoodin kattava tuki maksaa paljon ja sen räätälöinti on hidasta ja tulee kalliiksi. Kaikki lisäominaisuudet maksavat. Lisenssin skaalautuvuus on raskasta ja kallista, ja valittu tuote voi pahimmillaan lukita ratkaisut pitkälle tulevaisuuteen.

Sekä avoimia ja suljettuja työkaluja löytyy useita ja niitä vertailemalla löytää parhaiten soveltuvan ratkaisun. Esimerkki työkaluna opinnäytetyössäni käytän Talend työkalua.

3.2 Talend

Talend on avoimesti ladattava ETL-työkalu, mutta siitä voi myös maksaa lisenssimaksun. Lisenssin myötä toimintoja saa enemmän. Ohjelmointikielenä Talend käyttää Javaa, ja studio töiden kehittämiseen on tehty Eclipsen päälle.

Työkalusta on eri versioita erilaisiin käyttötarpeisiin. Liittymät eri lähteisiin ovat monipuolisia, lisäksi komponentit suorittavat eri osia tiedon muokkaamisen aikana. Jos valmista komponenttia omaan tarpeeseen ei löydy sellaisen voi itse ohjelmoida.

Versioiden käyttötarpeet kertovat mitä niillä on tarkoitus tehdä. Tämän hetken avoimen lähdekoodin versiot ovat:

- Big Data: ei rakenteellisen tieto kerääminen
- Data Quality (DQ): tiedon profilointi
- Enterprice Service Bus (ESB): palveluiden ja sovellusten integrointiin
- Data Integration (DI): integrointiin
- Master Data Management (MDM): avaintiedon hallinta

(Talend)

Työt voidaan ajaa studiossa tai viedä omina töinä ajettavaksi. Lisenssi version myötä tulee Talend Administration Center (TAC), jossa voi hallita töitä ja sisältää tietovaraston.

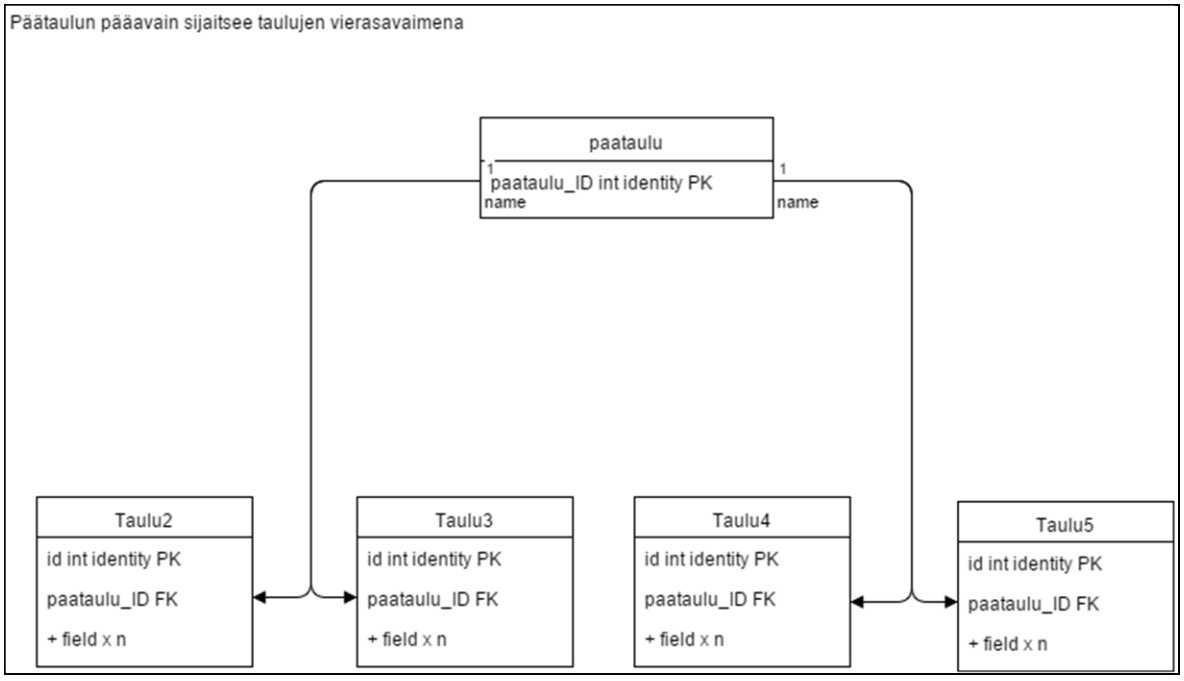
4 Kehitetty prosessi

Tässä luvussa kuvataan asiakkaalle kehitetty prosessi.

4.1 Tausta ja tavoitteet

Asiakkaan käytössä on tämän itse kehittämä sovellus, joka saa tiedon ulkopuolisista lähteistä. Tietoa kerätään eri maista, ja sovellus näyttää maakohtaisesti tiedon. Kaikki tiedot sijaitsevat maasta riippumatta samassa tietokannassa. Tavoitteena on luoda staattinen avain.

Sovellus tarvitsee tauluissa pääavaimen jokaisessa taulussa toimiakseen. Pääavain on muodostettu surrogaattina ”int”. Avaimen tietotyypissä loppuu aikanaan uudet arvot kesken rivimäärän ollessa valtava. Tietotyyppi tulee muokata uuden prosessin aikana, sekä olemassa oleviin tietoihin tulee luoda uudet avaimet. Päätaulun alla on 4 taulua jossa on pääavain vierasavaimena (kuvio 3).



Kuvio 3. Päätaulun pääavain sijaitsee vierasavaimena

4.2 Menetelmät

Ennen uuden prosessin kehittämästä vanha prosessi piti kuvata, ja kohteen tietomalli selventää. Dokumentointi sovelluksen osalta oli olematon, eikä päivitysprosessia ole oltu kuvattu lainkaan.

Tietomallin purkaminen onnistuu sovelluksella joka tuottaa taulujen relaatiot. Päivitykseen tulevien taulujen luontilauseista löytyy myös relaatiot sekä tietotyypit. Sovelluskehittäjän tiedossa on kuinka sovellus käyttää näitä tietokantatauluja. Taulujen rakennetta ei voi muuttaa ilman keskustelua sovelluskehittäjän kanssa. Sovellukset saattavat tarvita sarakkeita taulussa vaikka niitä ei ylläpidettäisi päivitysprosessissa. Taulun rakenteet pysyvät samoina muutaman tietotyypin muuttuessa, ainoastaan avaimet luodaan toisella tavalla.

Sovelluksen koodiin tarvitsee tehdä muutos kun tietotyypin muutos tehdään tietokantaan. Yleensäkin sovelluskehityksen lainalaisuudet tarvitsee tuntea muokatessa tietokannan rakenteita.

4.3 Uusi päivitys

Päivitys on jo toteutettu kertaalleen, ja nyt päivitykseen tulee muutos avaimen muodostamiseen. Avain halutaan pysyvän staattisena. Staattiseksi avaimeksi muodostu kahden attribuutin yhdistelmä. Päivitykseen tulee muutos siirtoon sekä lataukseen.

4.4 Lähteiden lataaminen

Lähdetietona toimii ostettu tieto joka noudetaan ftp-siirrolla. Nämä lähdetiedostot ovat txt-muodossa. Prosessi aloitetaan noutamalla tiedostot palvelimelle. Tiedostoja on useita, ja ostettua tietoa on useasta maasta. Työ onnistuu yhdellä liittymällä. Työ käynnistyy tarkistamalla mitkä tiedot haetaan. Tarkistus tehdään tietokannasta lukemalla aktiiviset lähteet.

Taulukko 1. Tietokantataulu

lahde	aktiivinen	iposoite	kayttaja	salasana	polku
lahde1	1				
lahde2	0				
lahde3	1				
n..	1				

Taulukko 1 kuvastaa tietokantataulua. Sarake ”lahde” nimittää lähteen, ”aktiivinen” kertoo onko lähde aktiivisena. Tietokannasta saatu tieto mahdollistaa lähteen passivoinnin, jos lähteen palvelin sattuu olemaan pois käytöstä tai tietoa ei ole voitu noutaa hetkeen. Taulusta voidaan myös helposti nähdä kaikki omat lähteet.

Työssä haetaan aktiiviset lähteet SQL-kyselyllä joissa on aktiivinen ”1”. Työhön palautuu nyt rivit aktiiviset rivit. Rivit siirtyvät komponenttiin, joka iteroi jatkossa tiedot muuttujina eteenpäin ftp-yhteyden avaukseen.

Yhteyden avaamisen jälkeen lähteestä noudetaan kaikki tiedostot omalle palvelimelle. Tiedostot tallennetaan kunkin lähteen kohdalla omaan kansioon: ”\ftp\lahde\noudetut\tiedostonimi.txt”. Polun ”lahde” muodostuu iteraation muuttujasta. Työ loppuu kun kaikista lähteistä on tiedostot noudettu. Työ kirjoittaa lokiin onnistuneet siirrot. Loki on myös luettavissa töiden hallinta konsolista.

Seuraavassa työssä tiedostot ladataan tietokantaan. Työn alussa tehdään sama tarkistus aktiivisista lähteistä. Aktiiviset lähteet iteroidaan työssä eteenpäin tiedostotarkistukseen. Kaikki lähteen tiedostot tarkistetaan siltä varalta ovatko ne ladattu jo tietokantaan. Tarkistus suoritetaan MD5 summan tarkistuksella. Jokainen ladattu tiedosto kirjoitetaan tietokantaan kertoen tiedoston tiedot. Jo ladatut tiedostot jätetään lataamatta tietokantaan ja siirretään pois lataus kansiosta. Uudet tiedostot ladataan tietokantaan. Latauksen jälkeen tiedosto siirretään: ”\ftp\lahde\ladattu\tiedostonimippkkvvvv.txt”. Tiedostoon lisätään päivämäärä latauspäivämäärän mukaan.

4.5 Laskennalliset muutokset

Prosessiin tehtävä muutos kohdistuu avaimiin, jotka luodaan ladattavaan kohteeseen. Pääavaimena toimii surrogaatti-avain, joka muodostuu lisätessä rivi tietokantaan. Ongelmaksi muodostuu avaimen tietotyyppi (int) sekä latausvaiheessa taulujen tyhjennykset. Pääavain halutaan pysyväksi eikä joka kerta muodostaessa aineistoa. Prosessissa on valmiina työt eri maille, jotka suorittavat ”transform”-vaiheen muutokset. Maat ovat vielä omissa tietokannoissa. Muutokset tulee suorittaa maa kerralla.

Eri maiden prosessit koostuvat omista päätöistä jotka kutsuvat alitöitä. Tarkoitus on kehittää muutama alityö jotka voidaan moduuleina liittää maiden päätöihin. Nämä moduulit liitetään vaiheittain eri maiden töihin, ja aktiivisten maiden tulee päivittyä, joko uusilla avaimilla tai vanhoilla. Uuden sekä vanhan avaimen muodostuksen tulee toimia samaan aikaisesti kohteen ollessa sama. Uudeksi avaimeksi muodostuu looginen kahden attribuutin yhdistäminen, jonka avulla saadaan yksilöivä avain. Avaimen muodostus ei ole mahdollinen tietotyypillä ”int”, vaan tietotyypiksi vaihdetaan ”bigint”. Tietokanta täytyy muuttaa vastaamaan uutta tietotyyppiä. Uusi avain voidaan syöttää tauluun vaikka taulu luo avaimen vanhan prosessin mukaan, käyttämällä ”IDENTITY_INSERT” syntaksia. Työn perään lisätään moduuli joka kirjoittaa tietokantatauluun tiedon onko maakohtainen työ suoriutunut onnistuneesti.

4.6 Siirto

Kohde sijaitsee eri palvelimella ja eri maassa kuin muutokset tehdään. Kohteen latausta ei kannata tehdä samassa työssä kuin muutokset tehdään. Työ aloitetaan tarkistamalla edeltävän prosessin vaiheen onnistuneet työt. Maakohtaisen prosessoinnin onnistuessa tiedot siirretään yhteen tietokantaan. Kunkin maan taulut saavat tässä vaiheessa vielä jälkiliitteen, joka on maan viitearvo (”tauluFI”). Tietokanta ei ole sovelluksen käyttämä tietokanta, vaan ”stage” kanta.

Työ iteroi maakohtaisesti kunkin onnistuneen maan. Työssä käytetään transaktiota maakohtaisesti. Jos jonkun maan yksi siirto ei onnistu, tietoa ei viedä loppuun vaan palautetaan edeltävään tilaan epäonnistuneen maan osalta. Onnistunut siirto merkitsee tietokantaan tiedon onnistumisesta. Epäonnistunut siirto kirjoitetaan myös. Onnistumisen ja epäonnistumisen merkintä on samassa taulussa eri arvolla.

4.7 Lataus

Kohteen taulut pitävät sisällään suuren määrän rivejä ja päivitys on hankala tietokannan ollessa huonossa kunnossa. Latausta edeltävä asia on ”truncate table” joka nopeuttaa latausta. Taulut päivitetään kahden prosessin avulla, uuden ja vanhan. Vanha prosessi on niille maille jotka ajetaan vielä vanhan avaimen mukaan ja uusi prosessi uuden avaimen mukainen. Vanha prosessin ei kirjoita lokiin tietoa. Päivitykset ajetaan muutoksen jälkeen ketjussa. Uusi päivitys ajetaan ketjussa, samalla tarkistetaan lokiin kirjoitetuista tiedoista kunkin vaiheen onnistumiset.

Lataava työ alkaa tyhjentämällä edellisen suorituksen yhdistetyt taulut. Näihin tauluihin yhdistetään kaikki päivitettävät rivit kaikista maista. Taulut ovat rakenteeltaan samanlaiset kuin sovelluksen taulut, mutta eivät sisällä avaimia tai indeksejä. Sovelluksen taulut ladataan lopulta näistä tauluista.

Seuraavassa vaiheessa tarkistetaan siirron onnistuneet maat. Onnistuneet maat iteroidaan läpi seuraavassa vaiheessa. Taulut joihin siirto kirjoitti tiedot edellisessä prosessin vaiheessa muodostavat syötteen. Tietue joko päivitetään tai kirjoitetaan uusi rivi. Päivittäessä leimataan tieto päivitetyksi aikaleimalla, ja lisätty aikaleima pysyy samana. Uuden rivin ollessa kyseessä päivitysleima on tyhjä, ja lisättyyn aikaleimaan tulee ajonaikainen kellonaika.

Tauluja on muutama joten helpoin tapa olisi päivittää suoraan taulut, mutta päivitys kestäisi liian pitkään työn käydessä läpi jokaisen päivitettävän rivin. Siksi prosessissa yhdistetään

avainten avulla tulevat rivit vastaanottavan taulun riveihin työkalussa ja kirjoitetaan ne tauluun joka työn alussa tyhjennettiin. Kaikki päivitettävät taulut ajetaan alitöinä samanaikaisesti ajan säästämiseksi.

Yhden maan osalta kun kaikki taulut ovat ajettu alitöinä, suoritetaan seuraava maa. Kun kaikki maat on suoritettu, työn seuraava vaihe on tarkistaa ne maat, jotka eivät ole uuden prosessin mukaisia. Maiden tiedot kopioidaan samoihin tauluihin, joihin alityöt muodostivat tiedot.

Kopioimisen onnistuessa työn seuraava vaihe tarkistaa, että mikään edeltävän vaiheen alityö ei päättynyt epäonnistuneesti. Epäonnistunut työ keskeyttää koko päivityksen. Kaikkien töiden onnistuessa, työ jatkaa sovelluksen taulujen päivitykseen. Tässä vaiheessa käytetään työn tauluja, jotka ovat alitöissä muodostettu.

Sovellus ei toimi mikäli viite-eheys on rikki tietokannassa. Transaktio on pakollinen päivityksessä. Suorituskyvyn parantamiseksi indeksit ja avaimet poistetaan päivityksen alussa. Kaikkiin tauluihin ajetaan ”truncate table”, jolloin lataaminen nopeutuu. Taulujen päivityksen jälkeen luodaan avaimet ja indeksit. Mikäli avainten luominen epäonnistuu, koko päivitys palautuu transaktion edeltävään tilaan.

Työn päätteeksi tietokantaan kirjoitetaan lopetusaika ja onnistui tai epäonnistui.

5 Pohdinta

Tekninen kehittäminen on huomattavasti nopeampaa kuin tiedon mallintaminen ja sovelluksen tietotarpeiden ymmärtäminen. Päivitysprosessin kehittäjällä tarvitsee olla muutakin osaamista kuin tiedon parsiminen ja siirtäminen. Kehittäjä on enemmän tiedon arkkitehti. Automaattisen ETL-prosessin kehitys voi kuulostaa helpolta valitsemalla muutama komponentti työhön ja vetämällä viivat niihin. Tietoa voidaan siirtää helposti paikasta toiseen, mutta onko tieto sen jälkeen käytettävää? Ja mikä meni pieleen päivityksessä. Tiedon siirtämisen helppous tekee päivityksestä hyvin vaarallisen. Ilman kohteen tuntemista ja tietomallia, päivitysten tekeminen on todella vaikeata. Tietueet saattavat näyttää nimellisesti oikein, etunimet oikeissa paikoissa, ja osoitteetkin ovat kohtuullisen hyvännäköisiä päällisin puolin. Ongelmaksi muodostuu, että onko etunimi ja osoite oikein tietueella. Väärin tehdyt JOIN-liitokset tietokannassa saattavat vääristää tiedon tai saattavat jättää jotain pois.

Toteutustapoja asioihin on monta. Myös pulmakohtia voidaan kiertää monin eri tavoin. Miten pulmia sitten ratkoa ja missä kohtaan prosessia? Hyvä kehittäjä ratkoo monta ongelmaa kerralla. Sanonta ”on helppo tehdä vaikeata ja vaikea tehdä helppoa” pätee hyvin myös ETL-prosessien kehittämisessä. Harvoin kehittäjällä on itsellään ylläpitovastuu, ja se täytyy muistaa kehittäessä prosesseja. Kun on korjannut muutaman jättikokoisen ETL työn tietää että kuinka vaikeata on päästä toisen suunnittelemaan työhön käsiksi.

Kehittäessä prosessia on hyvä lähteä tiedosta käsin luomaan eri vaiheita. Vaiheittainen tiedon muokkaaminen mahdollistaa virheiden korjaamisen prosessin kohdassa jossa virhe tapahtuu. Virhe voi johtua huonosta määrityksestä tai teknisestä virheestä. Tekniset virheet eivät pysty korjaamaan virheitä määrityksessä. Hyvä määrittelijä tuntee tiedon rakenteen.

Opinnäytetyön tarkoitus oli kuvata prosessiin tehty muutos. Prosessin muutos sinänsä onnistui, ja muutetut moduulit ja työt vietiin tuotantoon. Kehityksen aikaiset ongelmat muodostuivat tietomallin puutoksesta. Tärkeimpänä oppina koin haasteen kuinka käänteisesti puretaan tietokannan tietomalli. Hyvä IT-infrastruktuuri ei anna suurempia haasteita kehittämisessä. Huonommin suoriutuvat työt eivät koe pahempia suorituskyky ongelmia. Kun infrastruktuurissa löytyy pullonkauloja, jokaista kohtaa prosessissa joutuu teknisesti tarkastelemaan hyvin tarkkaan. Kaikki sql kyselyt ja proseduurit joutuu käymään läpi ja optimoida, ettei tietokanta tietokone termein mene kyykkyyn huonon syntaksin takia.

Suurin haaste prosessin kehityksen aikana oli aikataulu, ja dokumentointi perusta ”lue koo-

dista”. Prosessi piti purkaa teknisestä kehityksestä ylätasolle, ja suunnitella muutos. Helposti prosesseja korjataan tuomalla uusia ominaisuuksia päälle, jolloin prosessi muodostuu sekavaksi. Kehittäjälle tulee helposti mieleen ruveta ensin tekemään jotain, ennen kuin alkaa suunnitella. Joskus on helpompaa, ja järkevämpää kehittää prosessi uudestaan, kuin korjata vanhaa.

Vanhan prosessin kuvaaminen teknisestä toteutuksesta olisi aihe jota voisi tutkia enemmän. Ja minkälaisia menetelmiä ja pohjia tarvitaan kuvaamaan jo olemassa oleva päivitysketju?

Tietojen siirto tietokannasta toiseen harvoin on pullonkaulana, mutta kun tietokannat sijaitsevat eri maissa nopeuteen on syytä perehtyä. Myös tietokannan palvelimen tila ja taso vaikuttavat suorituskykyyn. Lisäksi levynopeus vaikuttaa sisäänkirjoitukseen ja lukemiseen.

Ongelmat johtuivat infran erilaisista ongelmista, ja tarvitsin eri osaamisalueiden tietämystä ratkaisuissa. Koin syventäneeni osaamista teknisellä tasolla monessa eri asiassa.

ETL-prosessit tapahtuu suurelta osin tietokantapainotteisesti ja tietokannan optimointiin on hyvä paneutua enemmän. Prosessin kehittäjä toimii ohjelmointi- ja tietokantamaailman rajapinnassa joten osaamista tarvitaan molemmista osa-alueista. Jokaista valmista prosessia voidaan optimoida eri keinoin.

Tietomallin kehittämistä on mielestäni hyvä tutkia ennen prosessin suunnittelua.

Lähteet

Entities. Luettavissa:http://www.teach-ict.com/as_a2_ict_new/ocr/AS_G061/315_database_concepts/attributes_entities/miniweb/pg2.htm. Luettu: 20.10.2015.

ETL (Extract-Transform-Load). Luettavissa:http://www.dataintegration.info/etl. Luettu: 20.10.2015.

Lahtonen, T. 2002. SQL (sivu 11)

Mustakallio, J. 2015. Vapaan lähdekoodin integraatiot - Mitä voi saada ilman lisenssikuluja?. Luettavissa https://prezi.com/euv_tpngmyiw/vapaan-lahdekoodin-integraatiot-mita-voi-saada-ilman-lisenssikuluja/?utm_campaign=share&utm_medium=copy. Luettu: 16.10.2015.

Repo, J. 2014. Kansallisarkiston tiff valokuvien siirto ja metatietojen päivittäminen. Luettavissa:http://www.theseus.fi/bitstream/hadle/10024/80883/Repo_Jouni_2014_10_7.pdf?sequence=1. Luettu: 20.10.2015

Talend. Luettavissa: https://www.talend.com/products/talend-open-studio. Luettu: 30.11.2015

Turkumäki, M. Toukokuu 2013. Tietokantamallinen laatukustannusmittari. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/62382/Tietokantamallinen%20laatukustannusmittari.pdf?sequence=1. Luettu: 20.10.2015.

Liitteet

Liite 1. mindmap etl kehittämisestä