Tampereen yliopisto

Jani Valo

TIETOKANTAJÄRJESTELMÄN VAIHTO

Tekniikan ja luonnontieteen tiedekunta
Diplomityö
Helmikuu 2020

TIIVISTELMÄ

Jani Valo: Tietokantajärjestelmän vaihto
Diplomityö
Tampereen yliopisto
Johtaminen ja tietotekniikka
Helmikuu 2020

Tietokantajärjestelmät pitävät sisällään monenlaista ja kokoista dataa. Niitä käytetään lukemattomissa erilaisissa toteutuksissa ja erilaisia käyttötarkoituksia varten löytyy useita eri tavalla rakennettuja tietokantajärjestelmiä. On kuitenkin mahdollista, että tietokantajärjestelmän rajat tulevat vastaan esimerkiksi sen hallitseman datamäärän suhteen niin, että datan hallinnan kannalta on järkevintä siirtää se toiseen tietokantajärjestelmään, jonka kautta suuremmankin datamäärän hallinta onnistuu helpommin.

Tässä tutkielmassa selvitetään millaiseen tietokantajärjestelmään kohdeyrityksen olisi järkevintä siirtyä siinä vaiheessa, kun kohdeyrityksen alun perin käyttämän tietokantajärjestelmän rajat tulevat vastaan. Selvitys aloitetaan tutkimalla tietokantajärjestelmiä yleisesti sekä käsittelemällä uuden tietokantajärjestelmän valintaan liittyviä aiheita. Lisäksi käsitellään Data Warehouse ja Business Intelligence -konsepteja, joita on sovellettu kohdeyrityksen alkuperäisessä järjestelmässäkin.

Seuraavaksi käydään läpi kohdeyrityksen tämän hetkinen tilanne. Tutkielmassa esitellään aiemmin rakennettuja järjestelmiä, jotka eivät lopulta täyttäneet sitä tarkoitusta, jota varten järjestelmää alun perin rakennettiin. Nykyisin käytössä oleva järjestelmä käydään läpi ja esitellään syyt miksi tietokantajärjestelmän vaihtaminen on kohdeyrityksessä tullut ajankohtaiseksi.

Tietokantajärjestelmän voi toteuttaa monella eri mallilla, joten työssä käydään läpi sitä, millainen malli sopisi kohdeyrityksen käyttöön. Samalla käsitellään rajoituksia, joita yrityksen uuden tietokantajärjestelmän valinnassa on hyvä ottaa huomioon. Mallin valinnan ja rajoituksien käsittelyn jälkeen tarkastellaan tarkemmin kolmea erilaista valittua mallia käyttävää tietokantajärjestelmään (IBM Db2, Microsoft SQL Server ja MySQL) ja tutustutaan niiden ominaisuuksiin.

Tutkielman lopuksi käsitellään tietokantajärjestelmän valintaan liittyviä ensisijaisia ja toissijaisia kriteereitä. Kriteereitä käsitellään kohdeyrityksen näkökulmasta ja havaitaan, että yrityksen tarpeet ja rajoitteet muokkaavat hyvinkin paljon sitä, millaisia kriteereitä tietokantajärjestelmän valinnassa on syytä käyttää. Tietokantajärjestelmän valintaa varten luodaan arviointikehys, jota käytetään lopullisen valinnan tekemisessä. Arviointikehyksen avulla päästään tulokseen, että kohdeyrityksen kannalta tutkituista järjestelmistä paras vaihtoehto uudeksi tietokantajärjestelmäksi olisi Microsoft SQL Server.

Avainsanat: Tietokantajärjestelmä, data warehouse, business intelligence, IBM Db2, Microsoft SQL Server, MySQL

ABSTRACT

Jani Valo: Changing a database management system
Master's Thesis
Tampere University
Master's Degree Programme in Management and Information Technology
February 2020

Database management systems contain data in many sizes and types. These systems are used in countless different implementations and different kind of systems can be found for different purposes. However it is possible that the limits of a single system are reached for example within the data size that the system can handle. In this situation it might be the best course of action to move the data into a new database management system in which it is easier to handle the size of the data needed.

In this thesis it will be examined what kind of database management system would be the best one for the target company to change to when the limits of the current system are met. Examination will start by researching the database management systems on a general level and by dealing with topics pertaining with changing a database management system. Additionally concepts like Data Warehouse and Business Intelligence are covered as those concepts are in use in the system currently used by the target company.

Next the current situation of the target company is covered. Previously built systems, that ultimately did not fulfill the purpose that they were built for, will be introduced in the thesis. The system that is currently in use at the target company and the reasons why changing the database management system has become relevant are presented.

Database management systems can be implemented in many different models so the thesis will cover what kind of a model would be best suited for the target company. At the same time the limitations that the company has for choosing a database management system are examined. After the database model has been chosen and limitations for the system have been examined the thesis will further investigate three different database management systems that use the chosen model (IBM Db2, Microsoft SQL Server and MySQL) and the properties of these systems are examined.

In the end of the thesis primary and secondary criteria affiliated with the selection of the new database management system are examined. The criteria are covered from the view of the target company and it is observed that a company's needs and limitations heavily influence what kind of criteria are to be used when choosing a new database management system. For the selection of the database management system an evaluation framework is created that will be used when making the final decision. With the help of the created framework a result is reached that from the researched database management systems the best one for the target company would be Microsoft SQL Server.

Keywords: Database management system, data warehouse, business intelligence, IBM Db2, Microsoft SQL Server, MySQL

SISÄLLYSLUETTELO

1. JOHDANTO

Tietokantajärjestelmät ovat olennainen osa nykyajan tietotekniikkaa ja ne ovatkin laajasti käytössä monenlaisissa erilaisissa ympäristöissä aina kirjastoista, sisällönhallintajärjestelmistä, verkkosivuista ja varausjärjestelmistä erilaisiin kassajärjestelmiin asti, sekä monissa muissa toteutuksissa. Tietokantajärjestelmät ovatkin tulleet kiinteäksi osaksi elämäämme, vaikka normaalisti ne ovatkin peruskäyttäjältä piilossa ja ne huomataan yleensä vain silloin, kun tapahtuu jonkinlainen virhetilanne.

Viimeisten vuosikymmenien aikana tietokantajärjestelmät ovat kehittyneet perinteisistä relaationaalisista tietokantajärjestelmistä moniin erityyppisiin tietokantajärjestelmiin. Tällaisia muita järjestelmiä ovat niin dokumentti-, graafi- kuin NoSQL-tietokantakin. Oli tavoitteena hallita sitten pieniä tai suuria tietomääriä, löytyy eri tietokannoista varmasti tilanteeseen sopiva vaihtoehto. On kuitenkin mahdollista, että käytetyn tietokantajärjestelmän rajat tulevat jossain kohtaa vastaan ja tällöin joudutaan pohtimaan yhtenä mahdollisena ratkaisuna vaihtoa toiseen tietokantajärjestelmään.

Tämän työn tarkoitus on käsitellä erilaisia tietokantajärjestelmiä ja valita niistä yrityksen käyttämää järjestelmää varten yksi vaihtoehto. Yrityksen nykyinen järjestelmä on tulossa tiensä päähän ja tätä varten tarvitaan vaihtoehtoinen tietokantajärjestelmä, johon tiedot voidaan siirtää ja käyttöä jatkaa. Tätä tehtävää varten työssä perehdytään kolmeen erilaiseen tietokantajärjestelmään ja luodaan arviointikehys, jota vasten voidaan valita yrityksen käyttöön sopiva tietokantajärjestelmä.

Toisessa luvussa käsitellään tietokantajärjestelmien toiminnallisuutta yleisesti ja esitellään siihen liittyviä osia. Luvussa käsitellään myös tietokantajärjestelmiin kytkeytyvät termit data warehouse ja business intelligence. Lisäksi käydään läpi tietokannan valitsemiseen liittyviä asioita.

Kolmannessa luvussa käydään läpi siitä millaisia iteraatioita yrityksessä on käyty läpi jotta on päästy siihen pisteeseen, jossa nykyinen järjestelmä on. Yrityksen käytössä oleva järjestelmä esitellään ja käydään läpi miksi tarve muutokselle on syntynyt.

Neljännessä luvussa käsitellään ensin sitä, millainen tietokantamalli yrityksen kannattaisi ottaa käyttöönsä ja millä tavalla vaihtoehtoisia järjestelmiä otetaan mukaan tutkittavaksi. Tutkittavien järjestelmien valinnan jälkeen nämä käydään yksitellen läpi ja

perehdytään niiden ominaisuuksiin sekä siihen, miten ne voisivat toimia yrityksen käytössä.

Viidennessä luvussa luodaan uuden järjestelmän valitsemista varten arviointikehys. Ensin kerrotaan arviointikehyksen kriteerien valinnasta ja perustellaan miksi kyseiset kriteerit oli kehykseen valittu. Tämän jälkeen käydään kriteerit vielä läpi, kertoen mitä ne pitävät sisällään arvioinnin suhteen. Lopulta kriteerit jaetaan painotuksen suhteen ensisijaisiin ja toissijaisiin kriteereihin, käyden tämän jälkeen kunkin kriteerin kohdalla läpi, miten jokainen mukana ollut vaihtoehto pärjäsi, kun tarkastellaan kyseistä kriteeriä. Viimeiseksi suoritetaan lopullinen arviointi perustuen siihen, miten eri vaihtoehdot pärjäsivät annettujen kriteerien kanssa ja valitaan tämän perusteella ehdotus uudeksi tietokantajärjestelmäksi.

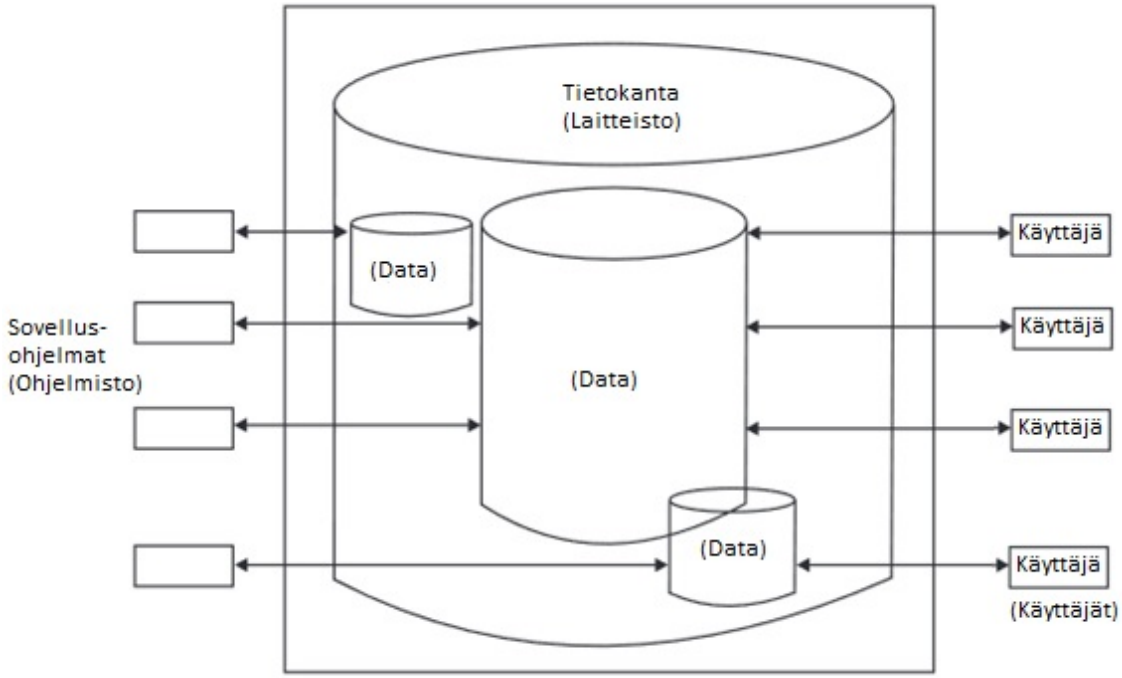
Kuudes ja seitsemäs luku vievät työn päätökseen. Kuudennessa pohditaan miten projektin tavoite onnistui, mitä olisi voitu vielä kehittää ja mitä olisi voitu tehdä toisin. Seitsemäs luku on taas yhteenveto, jossa kerrataan työn tarkoitus, miten sitä käsiteltiin, mitkä olivat tulokset ja millaisia rajoituksia työhön liittyi.

2. TIETOKANTAJÄRJESTELMÄT YLEISESTI

Tietokantajärjestelmä on tietokantaohjelma tai teknisesti sanottuna ohjelmistojärjestelmä, joka käyttää normaalia menetelmää, jolla voidaan luetteloida, hakea ja suorittaa kyselyjä dataan. Tietokantajärjestelmä hallitsee tulevaa dataa, organisoi sitä ja tarjoaa tapoja joilla sitä voidaan muokata tai poimia joko käyttäjien tai ohjelmien toimesta. Esimerkkejä tietokantajärjestelmistä ovat muun muassa MySQL, PostgreSQL, Microsoft Access ja SQL Server. Koska tietokantajärjestelmiä on saatavilla monia erilaisia, on tärkeää, että on olemassa tapa jolla ne voivat kommunikoida keskenään. Tätä varten useimmat tietietokantajärjestelmät sisältävät Open Database Connectivity -ajurin (ODBC), joka mahdollistaa tietokannan integroinnin toisiin tietokantoihin. Yleiset SQL-lauseet kuten SELECT ja INSERT voidaan näin muuttaa tietokannan omasta syntaksista sellaiseen syntaksiin jota toinen tietokanta pystyy käsittelemään. [1]

Tietokantajärjestelmässä mekanismit joilla määritellään sen rakenteet datan tallentamiseen ja palauttamiseen, tunnetaan tiedonhallintana (data management). Tietokanta itsessään on kokoelma loogisesti toisiinsa liittyvää dataa joilla on jonkinlainen luontainen tarkoitus. Kaikenlaisissa yrityksissä, joissa data ja informaation ovat oleellisessa asemassa, tietokannoilla on tärkeä rooli. Tarvittavan tiedon määrä on usein erittäin suuri ja tietokantojen avulla suurta määrä tietoa voidaan hallita helposti ja tehokkaasti. Ensimmäiset kaupalliset tietokannat saatiin 1960-luvun loppupuolella ja ne kehittyivät perinteisistä tiedostojärjestelmätietokannoista. [2]

Tietokantajärjestelmät voidaan luokitella käytetyn tietomallin mukaan joko relationaaliseen, verkolliseen (network), hierarkiseen, olio-orientoituneeseen tai olio-relationaaliseen. Tietokantajärjestelmän komponentit voidaan taas jakaa neljään osaan: laitteisto, ohjelmisto, data ja käyttäjät. Joissain tapauksissa on viidenneksi komponentiksi laskettu mukaan proseduurit, mutta nämä mainitut neljä ovat ne, jotka yleisimmin lasketaan varsinaisiksi komponenteiksi. Alla olevassa kuvassa on tietokantajärjestelmän komponenttien sijainti ja vaikutukset toisiinsa nähden. [1]–[4]



Kuva 1. *Tietokannanhallintajärjestelmän komponentit.*

Kuvasta 1 voidaan havaita tietokantajärjestelmän eri osat ja niiden suhteet toisiinsa. Laitteisto pitää sisällään datan, jota voi olla useammassa eri paikassa. Ohjelmisto ja käyttäjät taas vaikuttavat tietokannassa olevaan dataan joko pelkästään tarkastelemalla tai sitten muokkaamalla sitä. Käydään näitä osia seuraavaksi hieman tarkemmin läpi. [1]–[4].

2.1 Laitteisto

Tietokantajärjestelmissä laitteistolla (hardware) tarkoitetaan kaikkea fyysisiä laitteita, joita tietokantajärjestelmän rakentaminen vaatii. Yleisellä tasolla tämä voidaan rajata pelkästään tietokantaa pyörittävään palvelinkoneeseen ja sitä käyttävään asiakaslaitteeseen. Laitteisto pitää kuitenkin sisällään myös mahdolliset näppäimistöt, hiiret, tulostimet kuin koneiden sisällä olevat prosessorit, muistit, ynnä muut vastaavat. Lisäksi laitteistolla viitataan myös tallennusmediaan joka sisältää tietokantaan tallennetun tiedon. Yksinkertaisimmillaan tämä voi tarkoittaa CD/DVD-levyille tai flash-muistille tallennettua tietoa, mutta käytännössä yleisin tallennustapa on käyttää magneettikiintolevyjä (HDD), jotka ovat kustannustehokkaita ja yleisesti saatavilla. Liikkuvien osiensa vuoksi nämä kiintolevyt ovat myös alttiita hajoamisille, mutta rikkoutumisten aiheuttamaa vahinkoa voidaan pienentää tai kokonaan välttää muun muassa varmuuskopioinnilla tai yhdistämällä erillisiä kiintolevyjä yhdeksi RAID-levyksi. RAID-tekniikkaa hyväksikäyttäen yhden (tai useamman) levyn hajoaminen ei

välttämättä aiheuta merkittävää hävikkiä datan suhteen. Parhaimmissa tapauksissa kaikki menetetty tieto pystytään palauttamaan toisilta levyiltä. Laitteiston valinta näyttelee merkittävää roolia tietokantajärjestelmän valinnassa sillä tietokantojen skaala voi vaihdella aina yhden käyttäjän kokoonpanoista monien yhtäaikaisten käyttäjien tietokantoihin ja kutakin tapausta varten on hyvä valita oikeanlainen laitteisto. [1], [2]

Yhden hengen tietokantaa varten voi hyvinkin riittää omista nurkista löytyvä vanha laitteisto, jolle annetaan uusi elämä palvelinkoneena. Vastaavasti ison yrityksen tarpeisiin voidaan joutua ostamaan useamman laitteen verran tilaa palvelinsalista. Pienemmille yrityksille saattaa taas olla kustannustehokkaampaa sijoittaa omaan paikalliseen varta vasten palvelinkäyttöön tarkoitettuun koneeseen.

2.2 Ohjelmisto

Tietokannoissa ohjelmisto-osa (Software) pitää sisällään varsinaisen tietokantahallintajärjestelmän (DBMS) ja sen tarkoituksena on toimia välittäjäkerroksena käyttäjän ja tietokannan välissä. Tässä merkityksessä se on tietokantajärjestelmän tärkein osa, sillä käyttäjän hallitseman ohjelmiston avulla välitetään käskyjä tietokantajärjestelmään sen hallitsemiseksi. Ohjelmiston avulla on mahdollista hakea, muokata ja poista tietoa tietokannasta. Ohjelmisto suorittaa näitä toimintoja joko oman kyselykielensä (esimerkiksi SQL, QUEL, LINQ) tai sovellusohjelmiston avulla (esimerkiksi Visual Basic). Tietokantajärjestelmä toimii tietyn käyttöjärjestelmän alla ja sen asiakasohjelmisto voi sijaita tietokoneella jossa sitä voi käyttää joko yksi tai useampi henkilö. Verkossa tietokantajärjestelmä taas sallii dataa käyttävien ohjelmien sijaita palvelimella ja asiakasohjelmistojen jokaisella erillisellä työpöydällä. Tietokantajärjestelmä voi käyttää myös muita apuvälineitä kuten raportin luojia, sovelluskehitystyökaluja ja muita kehitystyökaluja. [1], [2], [4]

Ohjelmiston osuus on kriittinen varsinkin sellaisten tietokantajärjestelmän käyttäjien kannalta, jotka käyttävät tietokantaa vain saadakseen sieltä informaatiota jolla he voivat tehdä päätöksiä liittyen yrityksen liiketoimintaan. Tällaiset käyttäjät eivät todennäköisesti ole muuten tekemisissä tietokantajärjestelmien kanssa, joten ohjelmiston olisi hyvä pystyä antamaan tarpeeksi selkeät hallintatyökalut, joilla käyttäjä pääsee käsiksi tarvitsemaansa tietoon. Kuitenkaan ilman, että käyttäjän tulisi tietää sen enempää esimerkiksi SQL-kielestä.

2.3 Data

Dataa voitaneen pitää tietokantajärjestelmän tärkeimpänä osana ja tietokanta itsessään pitää sisällään kahdenlaista dataa. Ensimmäisenä on tietysti käyttäjädata joka koostuu yhdestä tai useammasta taulusta joissa on vaihteleva määrä sarakkeita ja rivejä. Tämä data on sitä mitä käyttäjä normaalisti tarvitsee ja pitää sisällään esimerkiksi kaupan alalla toimivilla yrityksillä tuoterekisterin, johon on syötetty tiedot yrityksen välittämistä tuotteista (EAN-koodi, nimi, hinta, toimittaja, ynnä muut vastaavat.). Toisena datana on taas tietokannan metadata-tieto eli ”dataa datasta”. Metadata pitää sisällään tietokantaan itseensä liittyvää tietoa, kuten taulujen määrä ja niiden nimet, tauluissa olevien kenttien määrä ja sarakkeiden otsikot sekä pääavainkentät. Lisäksi metadataan sisällytetään informaatio tietokannan käyttäjistä sekä heidän käyttöoikeuksistaan tietokannassa. Näin voidaan tietyiltä käyttäjiltä poistaa vaikkapa mahdollisuus tiedon poistamiseen tietokannasta. Tietokannan tiedoissa niitä osia, jotka ovat käyttäjille tarpeellisia, kutsutaan entiteeteiksi, kun taas tuntomerkkejä, jotka kuvaavat entiteettiä kutsutaan ominaisuuksiksi (attribute). Esimerkkinä voidaan tarkastella tietokantaa, joka pitää sisällään tietoa yliopisto-opiskelijoista. Perustentiteetti tällaisessa tietokannassa on opiskelija. Ominaisuuksia jotka kuvaavat tätä entiteettiä voivat olla muun muassa opiskelijan etu- ja sukunimet, opiskelijanumero, synnyinaika tai kurssien arvosanojen keskiarvo. Tiedoilla voidaan antaa myös rajoitteita. Esimerkkiä jatkaen voitaisiin rajata, että jokaisella opiskelijalla tulee olla tiedossa etunimi, sukunimi ja keskimmäisen nimen alkukirjain. Joidenkin opiskelijoiden kohdalla voi tietysti olla, ettei ehtoa voida täyttää jos opiskelijalla ei ole kuin etu- ja sukunimi. Tällaisissa tapauksissa tietoihin yleensä syötetään jonkinlainen täytetieto jolla ehto saadaan täytettyä (esimerkiksi syöttämällä nimen alkukirjaimeksi X). [1], [2]

Kaikkien tietokannan kohteiden kuvausta kutsutaan kaavioksi (schema). Tällaisen kaavion kuvaus riippuu siitä millaista datamallia on käytetty ja spesifioitu tietokannan suunnitteluvaiheessa. [2]

2.4 Käyttäjät

Kukin tietokanta on rakennettu tiettyä tarkoitusta varten joten käyttäjillä tarkoitetaan kaikkia niitä henkilöitä jotka tarvitsevat tietokannassa olevaa dataa käyttöönsä. Tällaisia käyttäjiä voivat olla muun muassa työntekijät, esimiehet tai johtohenkilöstö. Kuten aiemmin todettiin, käyttäjällä voi olla tietokantaan joko täydellinen tai osittainen pääsy. Tietokannassa käyttäjät voivat käsitellä tai noutaa dataa tarpeen vaatiessa käyttäen tietokantajärjestelmän tarjoamia ohjelmistosovelluksia tai käyttöliittymää. Yleisenä luokitteluna tietokannan käyttäjät voidaan jakaa kolmeen eri ryhmään: Pääkäyttäjä (Database Administrator), sovelluskäyttäjä (application programmer) ja

loppukäyttäjä (end user). Sovelluskäyttäjä kommunikoi tietokannan kanssa erilaisten sovellusten avulla, jotka ovat yleensä kehitetty tuomaan uusia ominaisuuksia tietokantajärjestelmään tai kustomoimaan jo olemassa olevia. Loppukäyttäjä ovat taas henkilöitä, joilla on pääsy olemassa olevaan dataan tietokantajärjestelmän tarjoamien mahdollisuuksien kautta. Loppukäyttäjät voivat hyödyntää joitain tietokantajärjestelmän tarjoamia käyttöliittymiä datan hakemiseen tai päivittämiseen. Tällaiset käyttöliittymät sallivat käyttäjän yhdenlaisen toiminnan tietystä joukosta mahdollisia toimenpiteitä tai sitten käyttäjä voi määrittää tietyt kriteerit jonka datan tulee täyttää kun sitä haetaan tietokannasta. Yleensä käyttäjän vaaditaan antamaan jonkinlainen syöte valitun toiminnon suorittamiseksi, kuten syöttämällä haetun henkilön puhelinnumero kun haetaan tämän osoitetta tietokannasta joka on tehty sisällyttämään tietoa puhelintilaajista. Käyttäjät saavat tietokannasta tietoa kysymällä kysymyksiä tietokantajärjestelmältä. Näitä kysymyksiä taas kutsutaan kyselyiksi (query). Käytetty kyselykieli taas riippuu siitä, mikä tietokantajärjestelmä on milloinkin käytössä. [1], [2]

2.5 Data Warehouse ja Business intelligence

Data warehouse on yleisnimi isolle kokoelmalle dataa, joka on koostettu yrityksen sisällä useasta eri lähteestä ja jota käytetään yrityksen johdon päätösten ohjaamisen. Data warehouse -järjestelmän konsepti voidaan jäljittää vuoteen 1988, jolloin IBM:n tutkijat Barry Devlin ja Paul Murphy kehittivät termin *information warehouse* ja IT-yritykset alkoivat rakentaa kokeellisia data warehouse -järjestelmiä. Käytännöllisiä data warehouse -järjestelmistä taas alkoi tulla vuonna 1991, kun W.H Inmon julkaisi oppaansa ”Building the Data Warehouse”. Alun perin data warehouse -järjestelmän kaltaiselle tekniikalle oli kysyntää sillä vallalla oleva teknologia – jota kutsuttiin nimellä naturally evolving architecture – oli moneltakin osalta ongelmallinen yrityksien käytössä. Vallitsevalla tekniikalla oli ongelmia muun muassa tuottavuuden, datan luotettavuuden sekä kyvyissä muuttaa saatavilla oleva data informaatioksi. [5]–[7]

Muita järjestelmiä käyttäneet asiantuntijat saattavat data warehouse -järjestelmää tutkittaessa tuntea olevansa oman mukavuusalueensa ulkopuolella. Tämä johtaa juurensa siihen, että data warehouse -järjestelmän käyttäjät ovat ensisijaisesti yritysmaailman henkilöitä ja vasta toissijaisesti teknikkoja. Data warehouse -järjestelmän käyttäjän päätehtävä onkin yrityksen liiketoimintaan ja päätöksentekoon liittyvän informaation löytäminen ja määritteleminen. [7]

Kuten aiemmin mainittiin, data warehouse -järjestelmä koostaa datansa useammasta eri lähteestä, minkä vuoksi järjestelmien tärkein yksittäinen puoli on tämän datan integraatio. Järjestelmään tulevaa dataa muokataan, muutetaan, uudelleen järjestetään

ja summataan. Näissä operaatioissa voidaan käyttää apuna ETL-työkaluja (Extract, transform, load), mutta se voidaan toteuttaa myös ilman niitä. Lopputuloksena järjestelmässä oleva data kuvastaa aina sen omistavaa yritystä. [7]

Business intelligence -termillä viitataan eri teknologioihin, applikaatioihin ja käytäntöihin, joilla kerätään, integroidaan, analysoidaan ja esitetään liiketoimintaan liittyvää informaatiota (business information). Business intelligence -järjestelmät antavat liiketoimintaan liittyvää tietoa niin historiasta, nykyhetkestä kuin ennakoitavissa olevasta tulevaisuudesta, käyttäen tietoa joka todennäköisimmin on koottu käyttäen data warehouse -järjestelmää. [8]

DW/BI (Data Warehouse / Business Intelligence) -ratkaisuja on olemassa useita erilaisia. Tällä hetkellä vallitsevia vaihtoehtoja ovat muun muassa Kimballin arkkitehtuuri, Independent Data Mart arkkitehtuuri ja Hub-and-Spoke Corporate Information Factory Inmon arkkitehtuuri. Näitä erilaisia sovellusarkkitehtuureita tarkastellaan seuraavaksi.

2.5.1 Kimballin DW/BI arkkitehtuuri

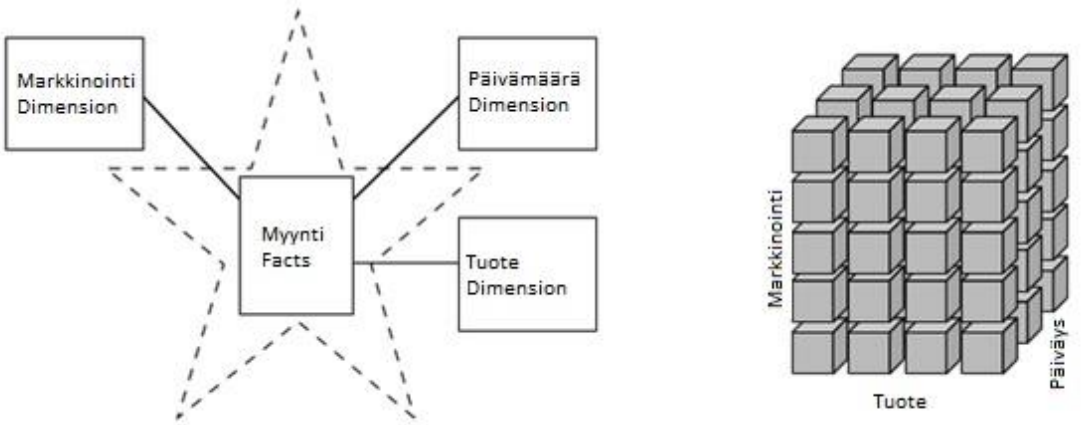
Ralph Kimball julkaisi ensimmäisen versionsa DW/BI-arkkitehtuuria käsittelevästä kirjastaan ”The Data Warehouse Toolkit” vuonna 1996. Kimballin tuolloin esittelemät käytännöt luoda DW/BI-tietokantajärjestelmä ovat sen jälkeen nousseet yhdeksi käytetyimmistä DW/BI-arkkitehtuureista. Kimballin opetusten mukaan yksi minkä tahansa organisaation tärkeimmistä voimavaroista on informaatio ja tätä käytetään melkein aina kahteen eri tarkoitukseen: operatiiviseen rekisterinpitoon ja analyyttiseen päätöstentekoon. Yksinkertaistettuna operatiiviset järjestelmät ovat alueita joihin dataa syötetään sisään, kun taas DW/BI-järjestelmät ovat niitä joista dataa otetaan ulos. DW/BI-järjestelmät ovat yrityskohtaisia ja Kimball painottaakin järjestelmän luomisessa, että DW/BI-järjestelmän pitää aina ottaa huomioon yrityksen tarpeet. [9]

Kimballin menetelmiä on kuitenkin myös kritisoitu siitä, että vaikka yksi sen vahvuuksista on analyyttisen raporttitietokantarakenteen suunnittelun perustan pohjautuminen yksittäisille liiketoimintaprosesseille, on se samalla sen heikkous koska menetelmä ei tarkastele miten nämä prosessit ovat yhteydessä toisiinsa. Joillain aloilla tällainen prosessien toisiinsa yhdistäminen on taas hyvinkin tärkeää (esimerkiksi terveyteen liittyvät tutkimukset, joissa pitää yhdistää syy ja seuraus). [10]

Yhtenä keskeisimmistä konsepteista Kimballin arkkitehtuurissa on niin sanottu *dimensional modeling*. Dimensional modeling on analyyttisen datan esittämiseen käytetty tekniikka ja se käsittelee samanaikaisesti kahta eri vaatimusta:

- Tuota dataa sellaisessa muodossa, että se on yrityskäyttäjien ymmärrettävissä.
- Tuota nopeita tuloksia kun tietokantaan tehdään kyselyitä.

Relationaaliseen tietokannan hallintajärjestelmään toteutettuja dimensional model -malleja kutsutaan nimellä *star schema* (tähtiskeema). Monidimensionaaliseen tietokantaympäristöön toteutettuja malleja taas kutsutaan nimellä *OLAP kuutio*. Näitä kahta mallia on havainnollistettu kuvassa 2.



Kuva 2. *Tähtiskeema ja OLAP-kuutio*

OLAP-kuutioon ladattu data on tallennettu ja indeksoitu käyttäen formaatteja ja tekniikkoita jotka on suunniteltu dimensionaalista dataa varten. OLAP-kuution moottori taas usein luo ja hallitsee suorituskyvyn aggregoinnin tai esilasketut yhteenvetotaulut. Kuutiot mahdollistavat myös SQL-funktioita vahvemmat ja analyyttisemmät funktiot. Haittapuolena on, että varsinkin isompien datamäärien kanssa maksetaan hintaa suorituskyvyn kanssa. Vaikka OLAP-teknologian mahdollisuudet kehittyvät jatkuvasti, kehottaa Kimballkin vielä käyttämään tähtiskeemaa ja tarvittaessa OLAP-kuutioita voidaan täyttää tähtiskeemasta. Kuvan tähtiskeemassa näkyvät dimension- ja facts-termit ovat Kimballin arkkitehtuurissa luotavia tauluja ja kaksi koko tähtiskeeman avainkomponenttia. Näitä käsitelläänkin hieman tarkemmin seuraavassa luvussa. [9]

Liiketoiminnalla on monenlaisia tarpeita, joista osa saattaa pysyä ajankohtaisina vielä vuosikymmentenkin jälkeen. Toistuvia teemoja yritysmaailmassa voivat olla muun muassa toteamukset ”Keräämme valtavasti dataa, mutta emme pääse siihen käsiksi”, ”Näytä vain se, mikä on tärkeää” tai ”Haluamme ihmisten käyttävän informaatiota tukemaan enemmän faktoihin perustuvaa päätöksentekoa”. Kimballin DW/BI-

järjestelmässä tällaiset yritysmaailman teemat on käännetty järjestelmän tavoitteiksi ja sen perusteella DW/BI-järjestelmän tulisi täyttää seuraavat vaatimukset:

- Informaation pitää olla helposti saatavilla DW/BI-järjestelmässä;
- Informaatio tulee esittää DW/BI-järjestelmässä yhdenmukaisesti;
- DW/BI-järjestelmän täytyy sopeutua muutoksiin;
- DW/BI-järjestelmän tulee esittää informaatio sopivassa muodossa;
- DW/BI-järjestelmän tulee olla turvallinen linnake, joka suojelee siinä olevaa informaatiota;
- DW/BI-järjestelmän täytyy toimia arvovaltaisena ja luotettavana perustana päätöksenteolle; ja
- Liiketoimintayhteisön tulee hyväksyä DW/BI-järjestelmä jotta se voidaan luokitella onnistuneeksi.

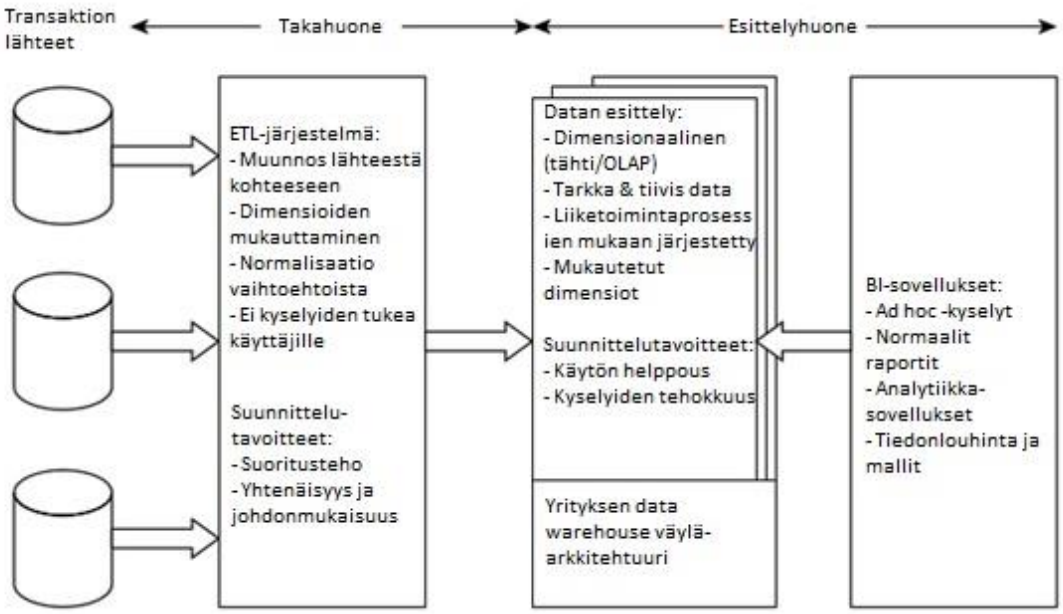
Nämä kohdat pitävät tietysti vielä sisällään useita erilaisia tavoitteita ja vaatimuksia. Jotta informaatio olisi helposti saatavilla, tulisi datan olla intuitiivista ja selkeää myös yrityskäyttäjälle eikä pelkästään kehittäjälle. Kyselytulokset tulisi palauttaa käyttäjälle minimoiduilla odotusajoilla ja datan rakenteen ja otsikoinnin tulisi kopioida yrityskäyttäjien ajatusmaailmaa ja sanastoa. Järjestelmän sopeutuminen muutoksiin taas pitää sisällään oletuksen siitä, että käyttäjien tarpeet, yrityksen liiketoiminta, data ja teknologia ovat kaikki alttiita muutoksille ja DW/BI-järjestelmän tulisi olla suunniteltu niin, että tällaiset väistämättömät muutokset eivät mitätöi olemassa olevaa dataa tai sovelluksia tai muuta niitä. [9]

Turvallisuuden puolesta järjestelmässä on hyvä ottaa huomioon, että minimissäänkin DW/BI-järjestelmä pitää sisällään tiedon siitä mitä tuotteita yritys myy ja millä hinnalla yritys niitä itselleen hankkii. Mahdollisesti järjestelmä voi pitää sisällään paljon arkaluonteisempaakin tietoa, mutta joka tapauksessa tällaisen tiedon päätyminen ulkopuolisille olisi yrityksen kannalta haitallista, joten DW/BI-järjestelmän tulisi tehokkaasti hallinnoida pääsyä organisaation luottamukselliseen informaatioon. [9]

Jokainen listalla mainittu kohta on järjestelmän kannalta tärkeä, mutta kaksi viimeisintä ovat Kimballin DW/BI-järjestelmän kannalta ne, joita ovat kriittisimpiä. Onnistunut DW/BI-järjestelmä vaatii kehittäjältään muutakin kuin sen, että on erinomainen arkkitehti, teknikko, mallintaja tai tietokantaylläpitäjä. DW/BI-järjestelmissä on yksi osa aina tuttua IT-alan toimintaa, mutta toinen osa on tekijälleen todennäköisesti tuntemattomampaa liiketoiminnan osa-aluetta. Järjestelmän kannalta onkin siis

oleellista, että liiketoiminnan puolen käyttäjät otetaan mukaan heti alusta alkaen, jolloin voidaan varmistua siitä, että valmis järjestelmä sisältää sen käyttäjien tuntemaa termistöä ja mahdolliset sovellukset toimivat sitä käyttävien henkilöiden logiikan mukaan. [9]

Kimballin arkkitehtuuriin perustuvassa DW/BI-ympäristössä on neljä erillistä ja selvästi erottuvaa komponenttia. Nämä komponentit ovat operational source- ja ETL-järjestelmä (Extract, Transformation, Load), data presentation area sekä business intelligence -sovellukset. Näiden komponenttien vaikutukset toisiinsa ja niiden sijainti järjestelmässä on kuvattuna alla olevassa kuvassa.



Kuva 3. *Kimballin DW/BI-arkitehtuurin ydinelementit.*

Arkkitehtuurissa on kaksi erilaista niin kutsuttua *huonetta*, joista takahuone (back room) on käyttäjältä piilossa oleva osa ja pitää sisällään ETL-järjestelmän. Esittelyhuone (front room) taas on se osa, joka on käyttäjän nähtävissä/hallittavissa ja pitää sisällään datan esittelyn (data presentation area) sekä BI-sovellukset (business intelligence applications). Operational source -järjestelmä on taas kuvassa näkyvä transaktion lähteet.

Operational source -järjestelmällä (operatiivinen lähdejärjestelmä) tarkoitetaan niitä järjestelmiä jotka tallentavat yrityksen liiketoiminnasta aiheutuvaa dataa. Tällaista järjestelmää voi tavallaan ajatella rakennetun data warehouse -järjestelmän ulkopuolisena osana, sillä todennäköisesti DW/BI-järjestelmän tekijällä ei ole juurikaan kontrollia siihen sisältöön ja formaattiin jossa nämä ulkopuoliset järjestelmät tuottavat dataa. On myös turvallista olettaa, että näihin lähdejärjestelmiin ei kohdenneta

tietokantakyselyitä niin laajoilla ja epätavallisilla tavoilla kuin DW/BI-järjestelmiin. Monissa tapauksissa lähdejärjestelmät ovat erityistarpeisiin tehtyjä sovelluksia, joihin ei ole sisäänrakennettu datan jakamista muiden organisaatiossa käytettävien operatiivisten järjestelmien kanssa. Tällaisissa tapauksissa voidaan turvautua esimerkiksi kolmannen osapuolen ohjelmistoihin joiden avulla lähdejärjestelmästä saadaan tarvittava data ulos halutussa formaatissa. [9]

ETL-järjestelmä toimii operatiivisen lähdejärjestelmän ja datan esittelyalueen välillä ja pitää sisällään niin työalueen, toteutetut datarakenteet kuin kokoelman prosesseja. Tiedon tuominen (extract) on luonnollisesti ETL-järjestelmän ensimmäinen tehtävä. Tiedon tuominen tarkoittaa, että järjestelmä pystyy lukemaan ja ymmärtämään lähdetiedostoissa olevaa dataa sekä tuomaan sen sisälle tarkempaa käsittelyä varten. Käytännössä tieto tuodaan tässä kohtaa sisälle data warehouse -järjestelmään. Datan sisään tuomisen jälkeen sille on mahdollista tehdä useita erilaisia toimenpiteitä (transformation). Todennäköisin vaihtoehto lienee datan puhdistus, sillä alkuperäiset lähdetiedostot saattavat sisältää virheellisiä rivejä, väärää tietoa, kirjoitusvirheitä tai puuttuvia elementtejä. Dataa saatetaan myös joutua yhdistelemään useamman lähdetiedoston kautta, mikäli vaikkapa asiakas- ja myyntitiedot ovat eri tiedostoissa ja halutaan koostaa tieto asiakkaan tekemistä ostoista. ETL-vaiheen tärkein tehtävä on kuitenkin luoda Kimballin arkkitehtuurissa käytettävät dimension- ja facts-taulut ja ETL-vaihe on juurikin se, missä nämä taulut tehdään. Viimeisenä vaiheena on datan vienti (load) seuraavaan vaiheeseen eli datan esittelyyn. [9]

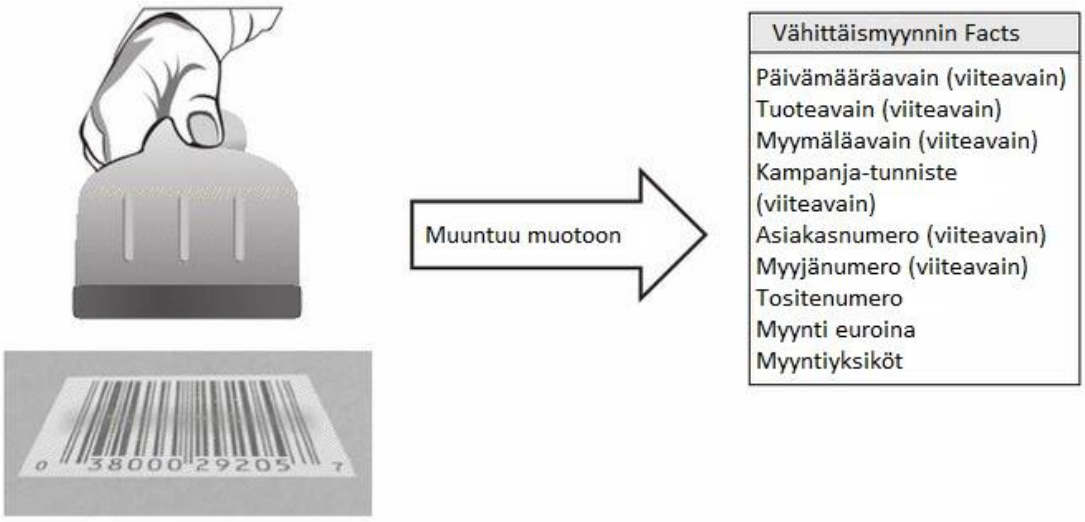
Datan tuonnin jälkeen on vuorossa datan esittely (data presentation area). Tässä vaiheessa kerätty data organisoidaan, talletetaan sekä mahdollistetaan käyttäjien, raportointityökalujen ja muiden BI-sovellusten tekemät kyselyt dataan. Kimballin arkkitehtuurissa datan esittelyyn on annettu muutamia painotettuja suuntaviivoja. Ensimmäiseksi datan tulee olla esitettynä, tallennettuna ja saatavilla joko relationaalisena tähtiskeemana (relational star scheme) tai OLAP-kuutioina (OLAP cubes). Tämän lisäksi datan esittelyalueen tulisi sisältää yksityiskohtaista tietoa (atomic data). Tämän datan tulee kestää käyttäjiltä tulevat odottamattomat kyselyt. Arkkitehtuurin logiikan mukaan missään nimessä ei tässä vaiheessa tulisi tallentaa pelkästään yhteenvetoja datasta dimensionaalisiin malleihin (dimensional model) niin että yksityiskohtainen tieto on lukittuna normalisoituihin malleihin. DW/BI-järjestelmän käyttäjät ja sovellukset saattavat toisinaan kyllä katsoa esimerkiksi yksittäistä riviä asiakkaan tilaukselta, mutta todennäköisimmin heitä kiinnostaa tieto vaikkapa kuinka paljon tiettyä tuotetta on myyty viime viikolla asiakkaille jotka ovat ostaneet jotain viimeisen kuuden kuukauden aikana. Tässä vaiheessa tarvitaan siis mahdollisimman

hyvin hienojakoista dataa jotta käyttäjien on mahdollista kysyä mahdollisimman tarkkoja kysymyksiä. [9]

Viimeinen suuri komponentti Kimballin DW/BI-arkkitehtuurissa on BI-sovellukset (Business Intelligence applications). Tällä termillä viitataan löyhästi kaikkiin niihin mahdollisuuksiin joita yrityskäyttäjille on tarjolla analyyttiseen päätöksentekoon käyttäen hyväksi aiempaa datan esittelyvaihetta. Kaikki BI-sovellukset suorittavat kyselyitä DW/BI-järjestelmän datan esittelyvaiheeseen. BI-sovellus voi olla vain yksinkertainen ja tilapainen kyselytyökalu tai se voi olla hyvinkin monimutkainen ja kehittynyt työkalu jolla datasta voidaan kaivaa tietoa monella eri tavalla ja mallintaa se edelleen haluttuun muotoon. Todennäköisesti useammat yrityskäyttäjät pääsevät dataan käsiksi sellaisen työkalun kautta joka ei vaadi käyttäjää rakentamaan kyselyitä suoraan. [9]

2.5.2 Facts- ja dimension-taulut

Olennaisena osana Kimballin arkkitehtuuria on aiemmin mainittu dimensional modeling, johon liittyvät konseptit facts ja dimensions. Dimensional modeling -mallissa facts-alkuisiin tauluihin tallennetaan organisaation liiketoiminnasta aiheutuvien tapahtumien suorituskyvyn mittaustulokset. Terminä ’fact’ edustaakin nimenomaan liiketoiminnan mittaustulosta. Alla olevassa kuvassa nähdään miten liiketoiminnan tapahtuma muuttuu facts-tauluun tallennettavaan muotoon.



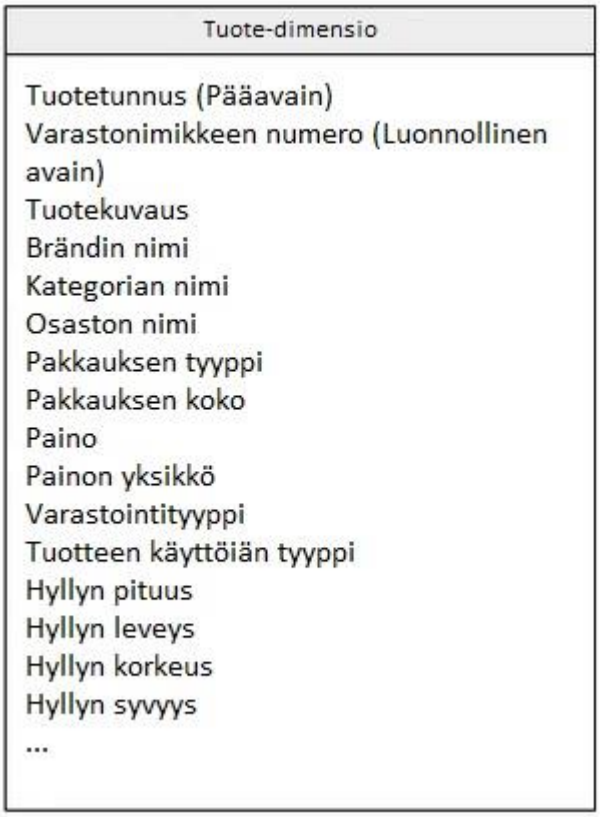
Kuva 4. *Yritystoiminnan tapahtuman muuntaminen facts-tauluksi.*

Kuvassa 4 on perinteinen kassatapahtuma jossa tuotteen viivakoodi luetaan kassalle, jotta asiakasta voidaan veloittaa. Kassatapahtumassa tallentuu todennäköisesti

muutakin tietoa kuten tuotekoodi, myyty kappalemäärä, myyjän kassatunnus, päivämäärä ja mahdolliset alennukset. Jokainen facts-taulussa oleva rivi vastaa jotain mitattua tapahtumaa. Tämä yksi-yhteen-suhde, jossa jokaisella mittaustapahtumalla fyysisessä maailmassa on vastaava yksittäinen rivi sitä kuvaavassa facts-taulussa, on dimensional modeling -mallin kulmakivi ja kaikki muu rakentuukin sen päälle. Hyödyllisimmät facts-tauluihin tallennettavat tiedot ovat numeerisia ja yhteenlaskettavia, kuten kokonaismyyntimäärä euroissa. Tämä yhteenlaskettavuus on tärkeää sillä BI-sovellukset palauttavat vain harvoin yhtä facts-taulun riviä. Useimmin palautettavia rivejä on vähintään satoja – parhaimmillaan – jopa miljoonia jolloin hyödyllisin asia, jota niille voidaan tehdä, on yhteen laskeminen. Toisinaan rivit voivat kuitenkin olla vain osittain yhteenlaskettavia (tilien saldot), tai sellaisia joita ei voida yhteen laskea (tuotteen yksikköhinta). Sellaisten rivien kohdalla, joita ei voida laskea yhteen, pitää käyttää funktioita kuten count tai average, muutoin joudutaan tyytymään palauttamaan facts-taulun rivejä yksi kerrallaan. Lisäksi facts-tauluihin pyritään tallentamaan vain todelliset tapahtumat. Mikäli tuotteella ei ole kassatapahtumassa myyntiä, ei sille myöskään tallenneta tauluun nollarivejä kuvaamaan tätä. Kaikilla facts-tauluilla on vähintään kaksi vierasavainta (foreign key) jotka viittaavat dimensions-taulujen pääavaimiin (primary key). Jokaisella facts-taululla pääavain joka koostuu vierasavainten osajoukosta ja jota kutsutaan nimellä composite key (yhdistelmäavain). Jokainen taulu jolla on tällainen yhdistelmäavain, on facts-taulu. Moni-moneen-suhteet ovat facts-tauluille ominaisia ja kaikki muut ovatkin dimension-tauluja. [9]

Teoriassa on mahdollista, että mittaustapahtuma tuottaa tuloksen tekstimuodossa, mutta käytännössä tällaiset tekstimuodossa olevat tapahtumat tulisi kaikin keinoin tallentaa dimension-tauluun facts-taulun sijaan. Facts-taulussa ei tulisi säilyttää turhaa tekstimuodossa olevia rivejä, ellei kyseinen tekstikolumni ole uniikki jokaisella facts-taulun rivillä.

Dimension-taulut sisältävät tekstimuodossa olevan kontekstin joka liitetään liiketoiminnan prosesseista mitattaviin tapahtumiin. Näitä tapahtumia kuvaillessa taulut vastaavat kysymyksiin ”kuka, mitä, missä, koska, miten ja miksi”. Alla olevassa kuvassa 5 on esimerkki yhdestä dimension-taulusta.



Kuva 5. *Esimerkki dimension-taulussa olevista attribuuteista.*

Ei ole epätavallista, että yhdessä dimension-taulussa on suuri määrä attribuutteja. Taulut myös pitävät yleensä sisällään facts-tauluja vähemmän rivejä, mutta saattavat sisältää useita isoja tekstikolumneja. Jokainen dimension-taulu sisältää yhden pääavaimen joka toimii viite-eheyden perustana jokaisen facts-taulun kanssa johon kyseinen dimension-taulu liitetään. Dimension-taulujen attribuutteja käytetään tietokantakyselyjen hakuehtoina ja rajoittimina ja tästä syystä tauluilla onkin merkittävä rooli koko DW/BI-järjestelmässä. Näin ollen attribuuttien tulisi koostua oikeista sanoista eikä lyhenteistä joita ei välttämättä ymmärrä muu kuin asiaan perehtynyt henkilö. Koodin käyttöä dimension-tauluissa tulisi myös minimoida ja sen sijaan korvata niitä kuvaavilla tekstimuotoisilla attribuuteilla. Monessa mielessä data warehouse on vain yhtä hyvä kuin dimension-tauluista löytyvät attribuutit. Koko DW/BI-ympäristön analyyttinen teho on suoraan verrannollinen näiden attribuuttien laatuun ja syvällisyyteen. Mitä enemmän käytetään aikaa attribuuttien kuvaamiseen yritysmaailman terminologiaa käyttäen tai mitä kauemmin käytetään attribuuttikolumneissa olevien arvojen laadun varmistamiseen, sen parempi.

Luokiteltaessa lähteenä olevaa dataa, ei kuitenkaan aina ole selvää kuuluisiko olemassa oleva numeerinen data facts- vai dimension-tauluun. Usein päätökset

tehdäänkin selvittämällä onko kyseinen kolumni mittatapahtuma, joka sisältää paljon arvoja ja ottaa osaa laskutoimituksiin (jolloin se on facts-taulu) vai onko se erillinen kuvaus, joka on tuotteen enemmän tai vähemmän muuttumaton attribuutti ja ottaa osaa kyselyiden rajoituksiin sekä rivien nimikkeisiin (jolloin se on dimension-taulun attribuutti).

2.6 Tietokannan valitsemisesta

Tietokannan vaihtaminen tai valitseminen kulloinkin vallallaan olevaan projektiin on varsin tapauskohtaista, mikä tulee vastaan aiheesta kirjoitetuista tutkielmista, jotka vaikuttavat olevan hyvinkin tarkasti rajattuja aihepiireiltään. Aihe voi olla vaikkapa tehokkaimman tietokannan valinta web-pohjaiseen järjestelmään silmätautioppiin liittyvien terveystietojen tallentamiseen ja jakamiseen. Aihealueeseen on siis tehty tutkimusta, mutta koska tilanteet joihin tietokantajärjestelmää valitaan voivat vaihdella suurestikin vaatimuksiltaan, voi tarvittavaan tilanteeseen liittyvää tutkimusta olla hankalampi löytää.

Tämä ei tietysti tarkoita, ettei aiheeseen liittyvää yleistä tutkimusta olisi lainkaan. Tämänkin tutkielman aihealueeseen liittyvää tutkimusta on tehty muun muassa artikkelissa ”*Choosing Right Database System: Row or Column-Store*”. Artikkelissa käsitellään juuri nykyajan tilannetta, jossa DW/BI-järjestelmät sekä erilaiset sosiaaliset verkostot vaativat suurien tietomäärien prosessointia (parhaimmillaan tera- tai petatavuittain) ja tarkoituksena on nimenomaan analysoida dataa ja tarjota sitä eteenpäin. Artikkelissa tutkitaan olisiko tämänkaltaisten järjestelmien tietokanta järkevämpi toteuttaa rivi- vai kolumnipohjaisesti. Perinteiset relaatiopohjaiset tietokantajärjestelmät kuten Oracle, IBM ja MySQL ovat rivipohjaisia tietokantoja ja edelleen yleisimmin käytössä data warehouse -järjestelmissä. Vastaavasti kolumnipohjaisia tietokantajärjestelmiä ovat muun muassa BigTable, MonetDB ja VectorWise. Perinteisissä relaatiomallisissa tietokantajärjestelmissä on kuitenkin ongelmia, kuten indeksointi, optimointi tilanteissa joissa on useita indeksejä, kiinteä tietuerakenne ja ajoittain tapahtuva tietokannan uudelleenjärjestely. [11]

Indeksointi tulee ongelmaksi kun relaatiomallinen tietokanta kasvaa suuremmaksi. Mitä isommaksi tietokanta kasvaa, sitä hitaammaksi indeksointi käy hakuaikojen kasvaessa. Tämä taas johtuu siitä, että tietoa haetaan rivi kerrallaan. Toinen indeksointiin liittyvä ongelma on, ettei perinteisissä tietokantajärjestelmissä voida suorittaa kyselyjä ja optimoida prosessia niin, että se käyttäisi vain indeksoituja kenttiä, mikäli suoritettavassa kyselyssä on mukana liian monta eri indeksiä. Tämä taas johtaa osaltaan suorituskyvyn heikkenemiseen. Kiinteä tietuerakenne on rivipohjaisten

tietokantajärjestelmien ominaisuus, mikä tarkoittaa uuden tiedon tai tietokentän lisäämisessä, että tietokantataulun rakennetta joudutaan muokkaamaan tai sitten tieto pitää lisätä jonnekin muualle. Tästä johtuen tietokantataulun rakennetta on hankala ylläpitää jatkuvasti. Data muuttuu tai sitä muokataan tietyllä aikavälillä ja tämä muutos tulisi sisällyttää tietokantaan. Tällaisista muutoksista johtuen tulee usein vastaan tilanteita joissa indeksejä joudutaan muokkaamaan, mikä taas voi osaltaan olla hankalaa. Näin ollen tietokannan rakenteen ylläpitämisestä tulee hankalaa. [11]

Kolumnipohjaisessa tietokantajärjestelmässä on myös omat ongelmakohtansa. Se ei muun muassa sovellu hyvin ETL-operaatioihin ja näissä tapauksissa tietokannan suorituskyky heikkenee. Kolumnipohjaisen tietokantajärjestelmän suorituskyky heikkenee niin ikään tapauksissa joissa dataa tuodaan järjestelmään tai viedään siitä ulos (export & import). [11]

Kaiken kaikkiaan tietokannan valinta on hyvin tapauskohtaista ja rivi- tai kolumnipohjaisia järjestelmiä vertailtaessa voidaan todeta, että rivipohjaiset järjestelmät sopivat tapauksiin joissa tietokannasta otetaan suuri määrä raportteja ulos ja erilaisten tapahtumien määrä on suuri. Mikäli taas ollaan tilanteessa jossa otetaan käyttöön yhä suurempia data warehouse -järjestelmiä joiden tarkoituksena on hoitaa perustason raportointia, liiketoiminnan analytiikkaa, monimutkaisten tapahtumien prosessointia ja datankeruuta (data mining), voidaan todeta että suorituskyvyn kasvava tarve ylittää perinteisen relaatiomallisen tietokantajärjestelmän suorituskyvyn ja tällöin kolumnipohjaisella tietokantajärjestelmällä voidaan kattaa nämä kasvavat tarpeet rivipohjaisia järjestelmiä paremmin. [11]

Toisessa artikkelissa (”*Choosing a Database Architecture: An Essential Guide for Data Warehousing Professionals”*) käsitellään myös tietokannan valintaa rivi- ja kolumipohjaisten vaihtoehtojen välillä, mutta tällä kertaa mukana on kaksi erilaista kolumnipohjaista arkkitehtuuria (disk-based & in-memory). Rivipohjaisten relaatiotietokantojen kohdalla artikkelissa nostetaan esiin kaksi ongelmakohtaa:

- Tilapäisten kyselyiden suorittaminen hidastaa suorituskykyä, koska kaikki kolumnit eivät ole indeksoituja.
- Datan päällekkäisyys jos sama tieto esiintyy useammalla rivillä.

Artikkelin mukaan relaatiotietokannat soveltuvat parhaiten OLTP-työkuormiin (Online Transaction Processing), koska tällöin kaikki data on sarjallistettu kerralla. Kahdesta eri kolumnipohjaisesta tekniikasta disk-based on vastaava tekniikka kuin mitä aiemmassakin artikkelissa käsiteltiin. Sen eduiksi nähdään tällä kertaa kolme kohtaa:

- Arkkitehtuuri sopii parhaiten OLAP-työkuormalle (Online Analytical Processing), missä data voi olla harvaa, mikä taas vähentää fyysisesti tarvittua tallennustilaa.
- Kyselyiden suorituskyky paranee koska kaikki kolumnit on indeksoitu arvojen perusteella.
- Kolumneissa olevalla datalla on yhdenmukainen tyyppi joten kolumnikohtainen kompressio on paljon suurempi kuin rivipohjaisilla järjestelmillä.

Heikkoutena taas pidetään tilanteita joissa kokonainen tietue pitää noutaa (kuten OLTP-järjestelmien kyselyissä). Välimuistin tietokantaa (in-memory database) käyttävät kolumnipohjaiset tietokantajärjestelmät taas erottuvat edukseen sillä, että niissä kaikki informaatio on ladattu muistiin, mikä poistaa tietokannan optimoinnin tarpeen. Käyttäjät suorittavat kyselyitä tähän muistissa olevaan dataan, mikä poistaa kokonaan sen latenssin mikä menisi datan etsimiseen kiintolevyltä. Tämä nopeampi datan haku on välimuistin tietokantojen isoin etu, mutta käyttäjien lukumäärän ja datamäärän kasvaessa tarvitaan myös yhä kasvava määrä keskusmuistia (RAM) pitämään sisällään tarvittava data. Tämä taas osaltaan kasvattaa laitteistokustannuksia. Välimuistin tietokanta ei näin ollen todennäköisesti ole ratkaisu ainakaan suuria datamääriä sisältäviin järjestelmiin, ellei yrityksen IT-budjetissa ole varattu selvää osuutta tarvittaville laitehankinnoille. [12]

3. NYKYISEN JÄRJESTELMÄN TAUSTA

Tarve nykyiselle järjestelmälle lähti samasta tarpeesta kuin varmasti monella muullakin kaupan alalla olevalla yrityksellä; kohdeyritys halusi saada tarkempaa tietoa yrityksen varasto- ja myyntianalytiikasta. Yrityksen käyttämässä kassaohjelmistossa (**Jeemly**) on itsessään myös mukana erilaisia raportointityökaluja, joilla voidaan ottaa muun muassa tuotemyyntiraportteja (esimerkiksi tuote- tai tuoteryhmäkohtaisia) tai varastonkiertoraportteja. Nämä raportit eivät kuitenkaan kaikilta ominaisuuksiltaan täytä niitä tarpeita joita yrityksellä on oman myyntinsä ja varastonhallinnan kehittämisessä. Eräs ongelma kassaohjelmiston kautta otettavissa raporteissa on se, että ne ovat aina vain läpileikkauksia kulloinkin valitusta tilanteesta, eikä niiden avulla pysty seuraamaan trendejä. Tämä olikin yksi suurimmista syistä, jonka takia järjestelmään lähdettiin kehittämään. Lisäksi tavoitteena oli hankkia tarkempaa tietoa muun muassa. hinnoittelun kehityksestä, tuotteiden ja toimittajien suorituskyvystä sekö varasto- ja katekierrosta. Tarkoitus ei kuitenkaan missään kohtaa ole ollut korvata kassaohjelmiston raportointiominaisuuksia vaan täydentää niitä.

Lopullisena tavoitteena oli päästä tilanteeseen, jossa kassaohjelmiston tietokannasta voitaisiin ottaa kuukausittain uudet tiedot ja yhdistää ne aiemmin otettuun dataan. Tällä tavalla kerätystä datasta pystyttäisiin luomaan halutunlaisia raportteja. Kassaohjelmisto käyttää tietokantajärjestelmänään Pervasive SQL -kantaa. Tähän tietokantaan on mahdollista päästä käsiksi **Pervasive Control Center** -ohjelmiston (PCC) avulla. Lisäksi kassaohjelma tukee myös itsessään puhtaita SQL-kyselyitä export-toimintonsa kautta. Kyselyitä on mahdollista tehdä PCC-ohjelman kauttakin, mutta ohjelmiston export-ominaisuus taas tukee tuloksien viemistä eri formaatteihin (muun muassa Excel ja csv). Alkuperäinen suunnitelma olikin käyttää PCC-ohjelmaa sekä kassaohjelmiston export-toimintoa ja saada niiden avulla tarvittava aineisto käytettäväksi Excel-muodossa.

3.1 SQL-kyselyt

Ongelmaa lähdettiin ensin ratkaisemaan ohjelmiston itsessään tarjoamilla työkaluilla. PCC-ohjelmalla oli aluksi tarkoitus selvittää missä tietokannan tauluissa on tarvittavat tiedot, luoda niitä käyttävä SQL-kysely ja lopulta käyttää kassaohjelmiston export-toimintoa SQL-kyselyn suorittamiseen ja tuoda tiedot Excel-muotoon.

Melko pian alkoi kuitenkin selvitä, millaisia ongelmia halutun tietomäärän esiin saaminen piti sisällään. Kassaohjelmiston tietokanta koostuu hieman yli 200 taulusta, joten tieto on hyvinkin pitkälle hajautettua. Tietoa joudutaan parsimaan kokoon hyvin useista tauluista. Esimerkkinä yrityksen tekemien ostotilausten lisääminen raportille tuotteittain vaatii kyselyn viiteen eri tauluun. Ensimmäisessä taulussa on ostotilausten ylätason tiedot (ostotilauksen tilausnumero, toimittajatieto, ynnä muut vastaavat). Toisessa taulussa on taas kunkin ostotilauksen rivien sisältö, kolmannessa tieto siitä paljonko kutakin tuotetta on tilattu, neljännessä tieto siitä moniko näistä tilatuista tuotteista on saapunut ja viidennessä vielä tilaukseen liittyvät ylimääräiset merkinnät (vaikkapa maininta, jos jokin tuote on tilattu tietylle asiakkaalle). Mikäli halutaan vielä, että raportissa olisi vielä listattuna toimittajatiedot selkokielisenä, pitää tehdä kysely toimittaja-tauluun ja verrata ostotilauksen tiedoissa olevaa toimittajan id-numeroa toimittajat-taulussa vastaavaan tietoon ja hakea tätä kautta tekstimuotoinen nimi toimittajalle.

Tämän lisäksi monet tietokannan taulut koostuvat useista kymmenistä sarakkeista. Yrityksellä ei myöskään ole käytössä minkäänlaista tietokantamallia, josta olisi ollut mahdollista selvittää eri taulujen suhteet avainkenttineen. Käytännössä selvitystyötä tehtiin käymällä tietokantaa läpi ja päättelemällä taulujen suhteita ja sitä mistä taulusta voitaisiin kulloinkin tarvittua tietoa löytää. Tietokannan taulut on pääosin nimetty kuvaavasti, mikä osaltaan helpotti tutkimustyötä (muun muassa varastosaldotiedot löytyvät varsaldo-taulusta). Tästä huolimatta eri tiedot oli kuitenkin hajautettu niin moneen tauluun, että tarvittavan tiedon saaminen yhdellä SQL-kyselyllä osoittautui erittäin haasteelliseksi.

Lopulta todettiin, ettei haluttua lopputulosta pystytä itse toteuttamaan järkevässä ajassa. Näin ollen päätettiin pyytää apua kassaohjelmistoa jälleenmyyvältä yritykseltä, joka puolestaan hoitaa ohjelmistossa käytettävän verkkokauppa-alustan kehitystä. Heillä on asiantuntemusta myös ohjelmiston tietokannasta, joten ratkaisun uskottiin löytyvän tehtävän ulkoistamisella. He toimittivatkin yrityksen käyttöön räätälöidyn SQL-kysely, jolla kaivattuja tietoja olisi ollut tarkoitus hakea tietokannasta. Kyselyn avulla saatiin osittain haluttua tietoa käsiteltäväksi, mutta tässäkin tapauksessa ongelmaksi muodostui vain yhden SQL-kyselyn käyttäminen. Kaikkea haluttua tietoa ei tälläkään kertaa saatu kyselyn avulla käytettäväksi. Valitettavasti aikoinaan käytetty kysely ei ole enää tällä hetkellä tallessa, joten tätä työtä varten ei voitu enää tarkastaa, mitä tietoja tuolloin kyselystä jäi puuttumaan. Kyselyn kehityksestä käytiin sen tekijöiden ja yrityksen välillä keskustelua ja muutama päivitetty versio toimitettiinkin, mutta nämä päivitetytkään kyselyt eivät tuoneet lopullista ratkaisua asiaan. Koska yksikään kysely

ei pystynyt tuottamaan kaikkea haluttua tietoa ja monen eri kyselyn tekemistä ei koettu järkeväksi ratkaisuksi, alettiin etsiä vaihtoehtoisia ratkaisuja.

3.2 Excel-lomake

Lopulta päädyttiin kokeilemaan yrityksen ulkopuolisen henkilön tarjoamaa ratkaisua. Kyseinen henkilö oli vakituisen työnsä ohella tehnyt muutamia hieman vastaavanlaisia projekteja, joissa tuotetaan erilaista analytiikkatietoa yritysten käyttöön. Hänen ratkaisuehdotuksensa oli, että tietokannasta tuotaisiin kokonaisia taulua sellaisinaan Excel-tiedostoiksi. Nämä tiedostot kasattaisiin sitten yhdeksi Excel-tiedostoksi ja käytettäisiin Excelin työkaluja halutun datan saamiseksi. Tämä malli helpottikin asiaa siinä mielessä, että taulujen tuominen tietokannasta oli export-toiminnon vuoksi hyvin yksinkertainen operaatio ja näin voitiin vain keskittyä siihen, miten tuodusta datasta saataisiin haluttu tieto ulos.

Excel-lomakkeeseen perustuvassa ratkaisussa tulivat kuitenkin hyvin pian vastaan omat ongelmansa. Tietoa oli tarkoitus ottaa tietokannasta lisää kuukausittain ja tämän tiedon määrä oli kohtuullisen suuri. Näihin aikoihin puhuttiin noin 60 000 uudesta rivistä per kuukausi. Tämä johti siihen, että vaikka Officen 2007 -versiosta asti Excel on tukenut 1 048 576 rivin taulukoita, alkoi kyseinen raja tulla hyvinkin pian vastaan siinä, kuinka paljon rivejä pystyttiin käsittelemään. Käytännössä tietokannasta otetulla tietomäärällä raja olisi tullut vastaan 18 kuukauden päästä ja tämäkin siinä tilanteessa, että tietojen ottaminen olisi aloitettu tyhjästä (toisin sanoen, historiatiedot olisi jätetty ottamatta). Viimeistään rajan tullessa vastaan olisi sitten pitänyt pohtia, jätetäänkö joitain menneitä vuosia pois analytiikasta, jotta tieto saadaan lomakkeeseen mahtumaan, mikä ei tietysti yritykselle ole kovin houkutteleva vaihtoehto, sillä menneellä tiedolla voidaan esimerkiksi ennustaa kausivaihteluita. Tämän ongelman lisäksi kaikkea haluttua analytiikkaa ei pystytty tässä ratkaisussa tuottamaan tai sitten saatu tieto oli selvästi virheellistä, kun samaa asiaa tutkittiin kassaohjelmiston kautta (esimerkiksi tuotteen myyntihistoria ei pitänyt paikkaansa).

Excel-lomakkeeseen perustuva ratkaisu ei kovin montaa iteraatiota käynytkään läpi, ennen kuin todettiin, ettei siitäkään saatu sellaista tulosta, jota oltiin etsimässä. Jälleen yritys alkoi etsiä vaihtoehtoisia ratkaisutapoja, joilla järjestelmä voitaisiin toteuttaa.

3.3 Microsoft Access ja nykytilanne

Kolmantena vaihtoehtona päädyttiin taas käyttämään yrityksen ulkopuolisen toimijan apua. Henkilö oli yritykselle jo entuudestaan tuttu sekä hän oli myös toteuttanut vastaavan kaltaisia projekteja muille yrityksille, joten valinta oli sinällään helppoa. Hänen kanssaan käytiin aluksi keskusteluja siitä, miten projekti tultaisiin toteuttamaan ja mikä valittaisiin toteutusalustaksi. Microsoft Access valikoitui käytettäväksi alustaksi hyvinkin pikaisesti, sillä tietokantajärjestelmän toteuttajalla oli ennestään suurin kokemus VBA:sta ja Office-työkaluista, joten projektia oli hyvä lähteä toteuttamaan näiden pohjalta. Alusta asti oli kuitenkin selvää, että Access toimisi vain eräänlaisena prototyyppinä ja se tultaisiin vaihtamaan johonkin muuhun järjestelmään sen jälkeen, kun ensin saataisiin toiminta halutunlaiseksi Access-järjestelmässä. Tavoitteena oli myös, että lopullinen järjestelmä tulisi olemaan täysin automatisoitu eikä raporttien saamiseksi pitäisi kuukausittain käyttää henkilötyötä.

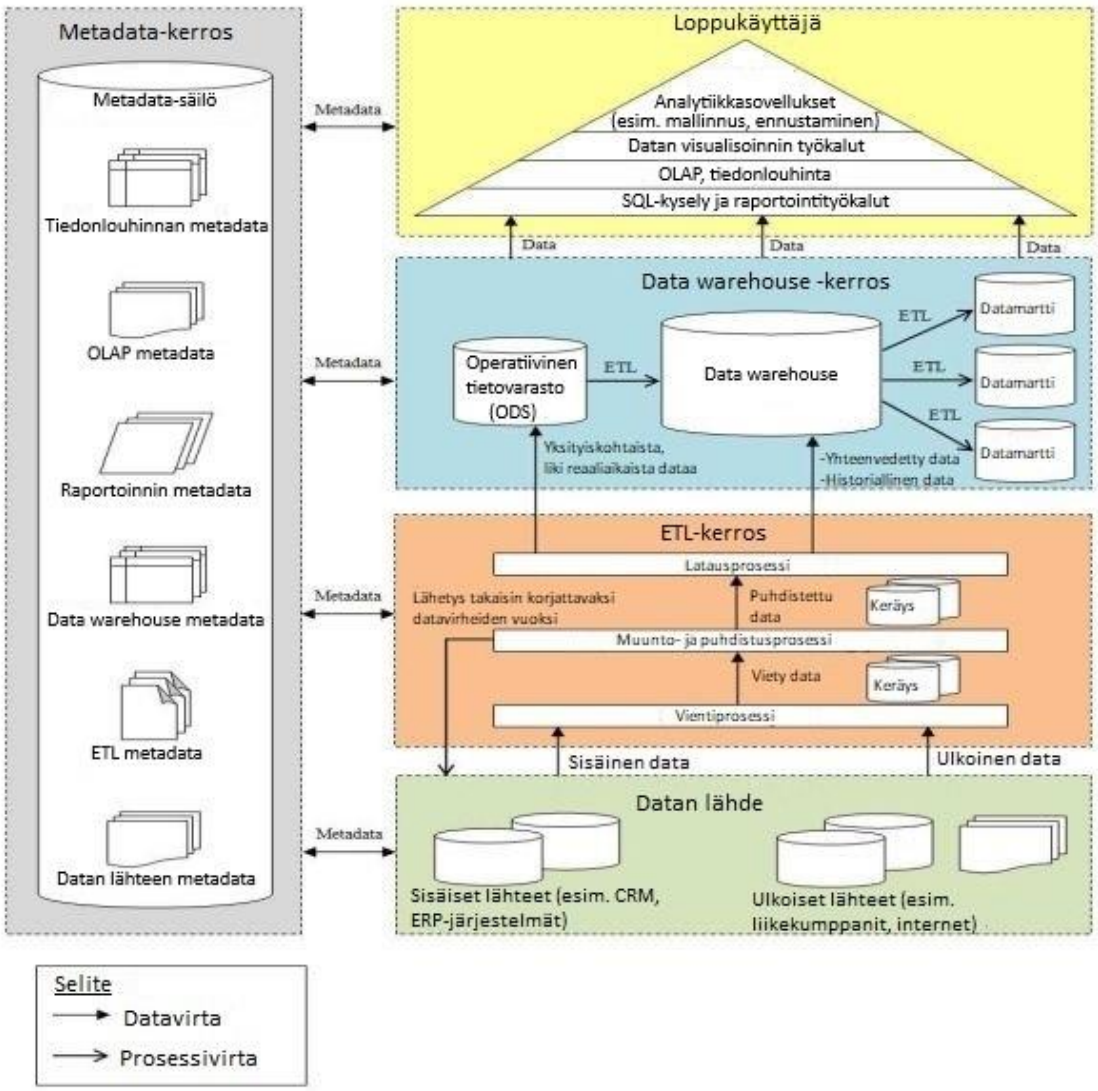
Heti alkuun jouduttiin tekemään iso urakka tietokantamallin luomisessa. Kuten aiemmin mainittua ei valmista mallia ollut saatavilla, joten sellainen piti luoda alusta alkaen. Järjestelmän tekijäkin itse totesi tietomallin luomisessa olleen haasteita, koska taulujen nimeäminen ja sisältö eri järjestelmissä perustui täysin erilaisiin lähtökohtiin ja ohjelmistojen kehittäjien näkemyksiin. Keskeiseksi ongelmaksi tietomallin rakentamisessa hän myös kertoi olleen taulujen viite-eheyden, mistä syystä taulujen liittäminen toisiinsa järjestelmän ulkopuolella osoittautui mahdottomaksi.

Käytännössä työtä tehtiin sekä tutkimalla tietokannassa olevaa tekstimuotoista dataa, että kassaohjelmiston graafisen käyttöliittymän tietoja. Käyttöliittymän selväkielisenä näyttämiä arvoja pyrittiin yhdistelemään tietokannasta löytyviin arvoihin. Näissä oli toisinaan omat ongelmansa, kuten tuotteen varastohistoriaa tutkittaessa. Käyttöliittymän puolella käyttäjälle tarjotaan muun muassa tapahtuman tyyppi -sarakkeessa selväkielinen tieto siitä, mikä tapahtuma on kyseessä (osto/myynti/inventointi/tms). Tietokannassa tieto on taas koodattu numeromuodossa (esimerkiksi ostotapahtuma on numero yksi). Nämä tiedot piti yhdistää toisiinsa hakemalla kassaohjelmistosta ensin tuote, jolla ei ollut isoa varastohistoriaa ja tästä historiasta valita jokin selvästi erottuva tapahtuma kuten inventoinnin nollaus.

Tämän jälkeen tutkittiin vastaavaa riviä siitä tietokannan taulusta, jossa se oli tallennettuna ja selvitettiin mikä numero tietokannassa vastasi tutkittua inventoinnin nollausta. Muitakin vastaavia ongelmia on tullut vastaan ja välillä on jouduttu ottamaan yhden tarvitun tiedon takia kokonaan uusia tauluja mukaan kerättyyn tietomäärään. Kassaohjelmistossa on mahdollista merkitä tuote niin sanotuksi multituotteeksi

(eräänlainen sateenvarjotuote vaikkapa vaatteen eri kokoluokille). Raportoinnin kannalta on tietysti hyödyllistä tuntea, mikäli tuote on osa jotain multituotetta, mutta tietokannassa tämä tieto on irrallaan tuotteen muista tiedoista. Tätä yhtä tietoa varten jouduttiin ottamaan mukaan kokonaan uusi taulu, jossa multituotetieto oli tallennettuna. Käytännössä olisi ollut tietysti mahdollista ottaa taulusta mukaan vain tämä yksittäinen sarake, mutta raportoinnin tulevaisuuden kannalta on usein hyödyllisempää, että tietoa on enemmän kuin tarpeeksi. Mikäli samasta taulusta tarvitaan joskus muutakin saraketta, on siitä olemassa jo historiatietoa eikä keräämistä tarvitse aloittaa tyhjästä. Tästä on järjestelmän kehityksen aikana ollut jo aiemminkin apua, kun on haluttu kehittää raportointia vaikkapa varastoanalytiikan puolelta ja tiedot, joita siinä kohtaa tarvittiin lisää, olivat jo valmiiksi kerätyssä tiedossa mukana.

Järjestelmän arkkitehtuurimalli perustuu Kimballin arkkitehtuuriin, mutta vaikutteita on otettu myös Ongin, Siewin ja Wongin tutkielmassa ”*A Five-Layered Business Intelligence*” esitettyyn viisikerroksiseen BI-malliin. BI:ssä on kyse siitä miten voidaan kerätä, ottaa käyttöön, ymmärtää, analysoida ja muuntaa yksi yritysten tärkeimmistä voimavaroista – raakadata – hyödylliseksi informaatioksi liiketoiminnan tehokkuuden kehittämiseksi [13]. Tutkielman arkkitehtuurimalli on esitetty alla olevassa kuvassa 6.



Kuva 6. *Käytetty arkkitehtuurimalli.*

Tämän tutkielman kannalta datan lähde -kerros (data source) tarkoittaa kassaohjelmiston tietokantaa, josta saadaan käytettävä data. Tällä hetkellä tämä tarkoittaa varsinaisen toteutuksen kannalta csv-tiedostoja, jotka otetaan kassaohjelmiston tietokannasta DMT-ohjelman (Data Moving Tool) avulla ja luetaan sitten sisään Access-kantaan. Datan siirtäminen csv-tiedostoista Access-kantaan on vielä tällä hetkellä manuaalisesti tehtävä työvaihe, eikä tapahdu automaattisesti. ETL-kerroksella kassaohjelmistosta saatu data käsitellään sellaiseksi, että sitä voidaan hyödyntää. Tietokannasta otetun datan siivoamiseksi on jouduttu tekemään iso määrä työtä. Alun perin data yritettiin tuoda ensin sellaisenaan, mutta se sisälsi suuren määrän ”roskamerkkejä” (muun muassa rivinvaihtomerkit asiakas-taulun tiedoissa ja välilyönnit EAN-koodien edessä tai takana), jotka tietoa tuotaessa sekoittivat järjestelmän niin, ettei dataa pystytty enää kunnolla käsittelemään. Osa näistä ongelmatapauksista pystyttiin korjaamaan suoraan kassaohjelmiston tietokantaan

(esimerkiksi turhat välilyönnit EAN-koodeissa), mutta toisten kohdalla tämä ei ollut mahdollista, joten korjaus tehdään erikseen ETL-kerroksella. Data warehouse -kerros on varsinainen Access-kanta, joka toimii DW-tyyppisenä ja on normalisoitu kolmanteen asteeseen. Loppukäyttäjän -kerros (End user) tarkoittaa taas varsinaisia raportteja, joita järjestelmän avulla pystytään luomaan. Loppukäyttäjän kannalta tämä tarkoittaa yhtä Excel-tiedostoa, johon on koostettu otetun ja käsitellyn tiedon tarjoama informaatio ja luotu lomake josta on mahdollista tutkia näitä tietoja erilaisia kriteereitä valitsemalla.

Järjestelmä aiheutti ongelmia myös sen jälkeen, kun oikeat tiedot oli löydetty ja tuotu Access-kantaan. Järjestelmää luotaessa kävi selväksi, että kassaohjelmistossa itsessään löytyi katelaskentaan liittyvä virhe, eivätkä ohjelmiston tuottamat arvot vastanneet oikeita. Tämän johdosta katelaskenta piti tehdä kokonaan uudestaan DW:n puolella. Katelaskentaa varten piti luoda neljä eri katelaskentatapaa ja lisäksi jouduttiin luomaan kokonaan uusi taulu, jossa katelaskentaa suoritettiin. Lisähaasteita laskentaan toi se, että kun kuukausittain otettiin mukaan aina uudet tiedot tietokannasta, tuli mukana myös kokonaan uusia tuotteita, joita oli menneen kuukauden aikana luotu kassaohjelmiston kautta. Näitä uusia tuotteita saadaan keskimäärin 500 per kuukausi ja jokaiselle tuotteelle pitää keksiä oikea tapa laskea kate sekä mihin myyntilajiin kukin tuote kuuluu.

Tällä hetkellä ollaan kuitenkin siinä tilanteessa, että järjestelmä vastaa melko lailla niitä spesifikaatioita, joita alussa lähdettiin tavoittelemaan. Access-kannasta luoduilla raporteilla pystytään tutkimaan niin varastoa kuin myyntiäkin. Näiden raporttien avulla on saatu apua niin hinnoitteluun, varastonkiertoon, myymättömien tuotteiden arvostukseen kuin useampaan muuhunkin myynti- tai varastoanalytiikan kohteeseen. Lisäksi raporttien avulla on pystytty tekemään yritykselle muun muassa tuoteryhmäkohtaista ABC-analyysia, mitä ei kassaohjelmiston kautta olisi mahdollista tehdä. Analyysin avulla on saatu yritykselle arvokasta tietoa, siitä paljonko on analyysissä käytettävien ryhmien (A-E) osuus myynnistä ja varastoarvosta ja näin on päästy käsiksi tietoon, mitkä tuotteet pitävät sisällään suurimman osan yrityksen myynnistä. Yhtenä tavoitteena oli järjestelmän täysi automaatio, mutta tähän päämäärään ei ole vielä aivan täysin päästy. Nykyisessä tilassaan kunkin kuukauden datan lisääminen järjestelmään vaatii vielä noin tunnin verran manuaalista työtä. Kuitenkin verrattuna siihen paljonko työtä datan lisäämiseksi projektin alkuvaiheissa piti tehdä töitä, ollaan huomattavasti paremmassa tilanteessa. Täyttä automatisointia onkin tarkoitus lähteä tavoittelemaan, kun järjestelmä on vaihdettu seuraavaan versioon.

Järjestelmän käytön kannalta Access-kannan rajat alkavat pikkuhiljaa tulla vastaan. Access-kannan sisäänrakennettu raja kannan koolle on 2 Gt:a ja järjestelmässä

käytettävän Access-tiedoston koko oli tätä työtä kirjoitettaessa 1,8 Gt:a. Ilman järjestelmän vaihtoa toiseen vaihtoehtoon jouduttaisiin tietokanta hajauttamaan toisiin tauluihin tai kokonaan toiseen Access-kantaan kun 2 Gt:n raja tulee vastaan. Accessissa on myös oma työkalu, jolla tietokannan kokoa voidaan jossain määrin hallita (Compact & repair database) ja työkalun avulla esimerkiksi mainittu 1,8 Gt:n tiedosto voidaan vielä kutistaa 1,2 Gt:n kokoiseksi. Joka tapauksessa projektissa ollaan nyt siinä vaiheessa, että prototyyppi on valmis ja voidaan aloittaa siirtyminen toiseen järjestelmään.

4. VAIHTOEHTOISET TOTEUTUSTAVAT

Yhtenä kriteerinä uuden järjestelmän valinnalle voitaisiin käyttää DB-engines-sivun (https://db-engines.com/en/) listaa suosituimmista tietokantajärjestelmistä (kuva 7), jotka tällä hetkellä ovat käytössä. Suuri käyttäjämäärä ei tietysti sinällään tarjoa parasta ratkaisua joka tilanteeseen, mutta mitä enemmän käyttäjiä tietokantajärjestelmällä on, sitä todennäköisimmin siihen on saatavilla tukea ongelmatilanteissa. Mikäli ollaan tilanteessa, jossa valinta on kahden muutoin tasaisen tietokantajärjestelmän välillä, mutta toinen on selvästi suositumpi, lienee järkevä valita vaihtoehdoista enemmän käytetty.

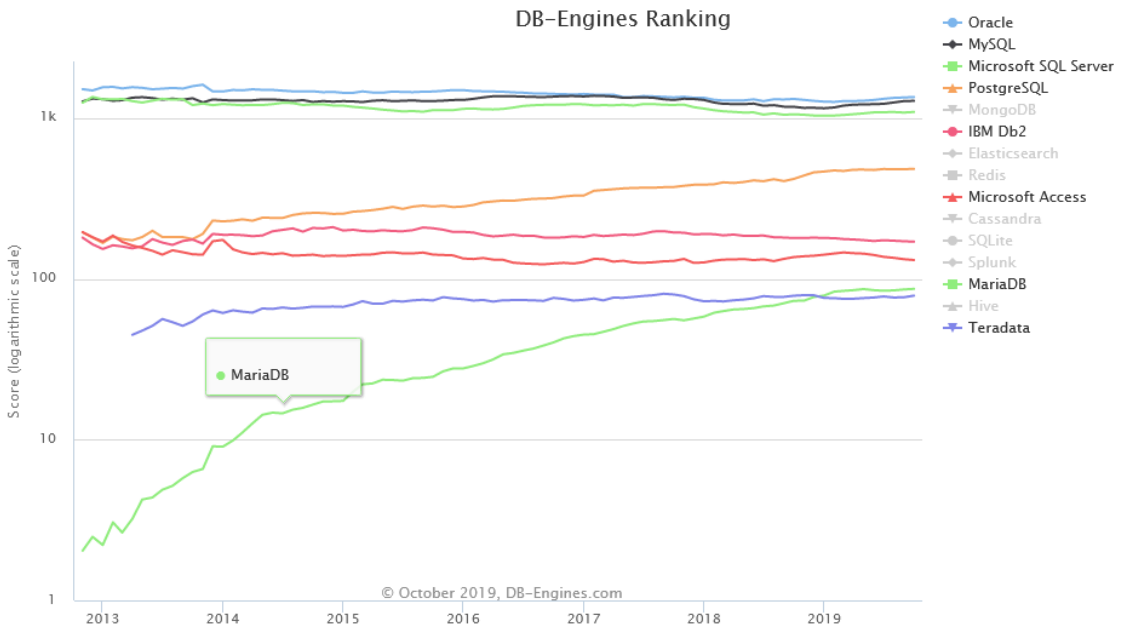
Mainittakoon, että DB-engines-sivuston lista mittaa suoraan sitä, kuinka monessa järjestelmässä eri tietokantajärjestelmät ovat käytössä. Heidän listauksensa perustuu enemmänkin järjestelmien suosioon, jota mitataan muun muassa tutkimalla tietokantajärjestelmästä tehtyjä hakuja eri hakukonesivustoilla (Google, Bing, Yandex), seuraamalla järjestelmien trendiä Google Trends -sivulla, tutkimalla järjestelmään liittyvien avointen työpaikkojen määrää ja paljonko järjestelmästä liikkuu mainintoja sosiaalisessa mediassa. Tämä ei varmastikaan tuota täysin oikeaa kuvaa eri järjestelmien käyttömääristä, mutta vertailtaessa muihin listauksiin, päästään tälläkin tekniikalla pitkälti samoihin tuloksiin.

Scalegrid-sivuston tekemän tutkimuksen mukaan, jossa he kyselivät suoraan kehittäjiltä, mitä järjestelmää he käyttävät, päästiin hyvin paljon samankaltaisiin tuloksiin. Heidän listassaan suosituimmat relaatiomallin tietokantajärjestelmät olivat järjestyksessä: MySQL, PostgreSQL ja Oracle. Heidän otantansa on suhteellisen pieni – muutama sata kehittäjää –, mutta tulos luo kuitenkin samanlaista kuvaa eri järjestelmien suosiosta kuin DB-engines-sivun listaus. Ainoat erot tulevat siinä, että DB-engines-sivun listalla Oracle saavuttaa suosituimman järjestelmän paikan ja kolmantena tulee Microsoft SQL Server, mikä oli Scalgrid-sivuston listassa vähiten käytettyjen mukana. [14], [15]

Aiemmin käsiteltiin valintaa rivi- ja kolumnipohjaisten tietokantajärjestelmien välillä, mikä auttaa rajaamaan sen mitä eri vaihtoehtoja kannattaa käydä läpi. Järjestelmän tarkoituksena on nimenomaan tuottaa erilaisia raportteja yrityksen käyttöön ja ETL-prosessit ovat isona osana järjestelmän kokonaisuutta. Näillä kriteereillä voidaan todeta, että tällä hetkellä perinteinen relaatiomallinen rivipohjainen

tietokantajärjestelmä on vielä järkevämpi valinta vaihtoehtoiseksi toteutustavaksi. Lisäksi nykyisen kassaohjelmiston tietokanta on niin ikään rivipohjainen, joten siitä tuotua tietoa ei tarvitse ensin muokata toisenlaiseksi. Jotkin perinteiset relaatiomalliset tietokantajärjestelmät tukevat nykyään myös kolumnipohjaista tallennusta (esimerkiksi IMB Db2 versiosta 10.5 alkaen) joten muutos voi olla mahdollista tulevaisuudessakin valinnasta riippuen. [16]

Relaatiomalliin perustuvien tietokantajärjestelmien käyttö on myös säilyttänyt suosionsa vaikka erilaisia NoSQL-vaihtoehtoja onkin viime vuosina noussut jo haastamaan relaatiopohjaisia tietokantajärjestelmiä. Käyttömäärien suhteen tarkasteltuna kaupallisten ja avoimeen koodiin perustuvien järjestelmien välillä ei siinäkään ole merkittävää eroa. Avoimen koodin puolella käytetyimpiä järjestelmiä ovat MySQL, PostgreSQL ja MariaDB. Vastaavasti kaupallisella puolella käytetyimpien järjestelmien joukossa ovat Oracle, SQL Server ja DB2. Nämä kaikki kuitenkin ovat kaikki järjestelmät huomioon ottaen aivan suosituimpien joukossa ja ovat siellä pysyneet jo pitemmän aikaa, kuten alla olevasta kuvasta nähdään. Näin ollen ainakin tällä hetkellä on siis suhteellisen turvallista olettaa, että vaikka valitsisi minkä tahansa suosituimmista relaatiomallisista järjestelmistä, on sille saatavissa tukea vielä pitkälle tulevaisuuteen. [17], [18]



Kuva 7. *DB-engines-sivun trendilistaus relaatiotietokantojen suosiosta.*

Kuvasta 7 nähdään, että viimeisen kuuden vuoden aikana ei juuri muutoksia ole eri järjestelmien suosiossa tapahtunut. Oikeastaan ainoa isompi muutos on MariaDB-järjestelmän suosion nousu kärkikymmenikköön. Mukaan voidaan ottaa myös muutkin

tietokantamallit, mutta relaatiomallit ovat niihinkin verrattuna pitäneet suosionsa vuosien saatossa hyvin. Kuvasta voidaan todeta sekin, että alkuperäiset SQL-tietokannat eivät ole hävinneet vielä mihinkään, sillä kärkikymmeniköstä vain MariaDB on julkaistu 2000-luvun puolella (2009), kaikkien muiden alkuperäinen julkaisu on 80- tai 90-luvulta.

Kuten aiemmin mainittiin, on DW/BI-järjestelmään tuotua tietoa jouduttu muokkaamaan suuresti johtuen erilaisista ongelmatiedoista tuodun tiedon sisällä. Tällä hetkellä DW:n puolella tehtävät muutokset on toteutettu VBA-skriptien avulla, joita Access tukee. Nämä skriptit joudutaan joka tapauksessa muokkaamaan uuteen järjestelmään sopiviksi, sillä juuri muita VBA:ta tukevia järjestelmiä ei Accessin lisäksi ole. Eduksi voitaneen siis laskea, jos valittu tietokantajärjestelmä tukee kieltä, josta kehittäjillä on kokemusta, eikä tarvitse opetella uutta syntaksia.

Yrityksen käytössä on tällä hetkellä yksi palvelinkone, johon tietokanta on mahdollista asentaa. Kyseessä on Windows Server 2008 r2 -käyttöjärjestelmää käyttävä palvelin, joten valitun tietokantajärjestelmän tulisi tukea Windows-käyttöjärjestelmää. Tämä ei varmastikaan ole ongelma, sillä suurin osa tietokannoista tukee Windows-järjestelmiä. Windows Server 2008:n tukiaika on kuitenkin loppumassa tammikuussa 2020 joten yrityksellä on edessään väkisinkin palvelimen vaihto toiseen. Se, mikä käyttöjärjestelmä tulevassa palvelimessa tulee olemaan tai hankitaanko tulevaisuudessa tilaa vain jostain palvelinsalista, on vielä ratkaisematta. Tästä syystä on eduksi mitä useampaa käyttöjärjestelmää tietokantajärjestelmä tukee, jolloin vaihto voidaan tehdä niin, että järjestelmä toimii vielä uudellakin palvelimella.

Tietokantajärjestelmän lisensointi ei sinällään ole tiukka valintakriteeri. Valittu järjestelmä saa olla kaupallinen tai avoimeen koodiin perustuva. Yrityksellä ei kuitenkaan ole tarkoitus sijoittaa suuria summia rahaa kiinni uuteen järjestelmään, joten liian suuri kustannus voi olla esteenä valinnalle, vaikka järjestelmä muuten olisi muita vaihtoehtoja parempi. Tarkkaa summaa, joka järjestelmään on laittaa kiinni, ei kuitenkaan tällä hetkellä ole. Tietokantajärjestelmistä on usein eri versioita eri hinnoilla, joten niiden hintoja voidaan arvioida tapauskohtaisesti.

Datan turvallisuutta on nykypäivänä paljon painotettu, eikä varmasti turhaan. Erilaisia tietomurtoja on tullut ilmi ikävän tasaista tahtia, ja datan turvaaminen on noussut yhä suuremmaksi puheenaiheeksi. Toteutettu järjestelmä tulee pitämään sisällään niin yrityksen itsensä kannalta olennaista ja tärkeää tietoa, sekä asiakasdataa. Näitä kumpaakaan ei haluta joutuvan ulkopuolisten toimijoiden haltuun, joten painoarvoa voidaan järjestelmän valinnassa antaa sille järjestelmälle, jolla on mahdollista toteuttaa

mahdollisimman turvallinen kokonaisuus. Turvallisuutta voidaan tukea monilla tavoilla, kuten SSL-yhteydellä (Secure Socket Layer) server- ja client-päätteiden välillä, mikä huolehtii sellaisista turvallisuuskohdista kuin todennus, luottamuksellisuus ja eheys. Turvallisuutta voidaan tietysti lisätä monilla muillakin keinoilla, mutta tietokantajärjestelmät sisältävät usein myös työkaluja, joilla niiden käyttöä voidaan rajata tai muuten tehdä ympäristöä turvallisemmaksi, joten nämä kannattaa arvioinnissa ottaa huomioon.

Yksi tärkeistä kohdista koko järjestelmän suunnittelussa on datan vienti yrityksen tietokannasta toiseen. Kuten mainittua tällä hetkellä tieto otetaan yrityksen tietokannasta ulos csv-tiedostoina ja siirretään sitä kautta toiseen tietokantaan. Samaa käytäntöä olisi hyvä päästä jatkamaan myös uudessakin järjestelmässä, joten voidaan laskea valintaa puoltavaksi asiaksi, mikäli uusi järjestelmä tukee suoraan csv-tiedostojen tuontia tietokantaan tai jos järjestelmässä on muutoin hyvä valikoima erilaisia vaihtoehtoja datan tuomiseksi tietokantaan.

Vaihtoehtoja on siis useita, mutta joka ikistä vaihtoehtoa ei kuitenkaan ole järkevä lähteä tarkasti tutkimaan, joten jonkinlainen rajaus on hyvä tehdä heti alkuun. Tätä tutkielmaa varten valittiin potentiaalisiksi korvausvaihtoehdoiksi kolme eri tietokantajärjestelmää: IBM Db2, Microsoft SQL Server ja MySQL Nämä rajattiin lähinnä sillä perusteella, että mukana olisi sekä kaupallisia, että avoimeen koodiin perustuvia vaihtoehtoja. Lisäksi kaikki ovat maailmanlaajuisesti käytettyjä järjestelemiä, joten niihin on saatavissa hyvä määrä materiaalia niin tutkimisen kuin tuenkin puolesta järjestelmää rakennettaessa ja käytettäessä.

Vaikka Oracle on suosituimpia kaupallisia järjestelmiä, se rajattiin heti vaihtoehtojen ulkopuolelle, sillä lisensointikulut nostavat tietokantajärjestelmän hinnan välittömästi niin suureksi, ettei sitä valittaisi sen puolesta muutenkaan. Oracle tarjoaa joko käyttäjä- tai prosessorikohtaista hinnoittelua. Näidenkin kohdalla hinnoittelu vaihtelee sen mukaan, mitä versiota on hankkimassa (Standard Edition 2, Enterprise, Personal, Mobile tai NoSQL Database Enterprise). [19] Käyttäjäkohtainen hinnoittelu voisi muuten toimia kohdeyritykselle, mutta minimimäärä tälle on 25 käyttäjää ja yrityksellä ei kuitenkaan olisi tarve kuin muutamalle käyttäjälle. Halvin hinta 25 käyttäjälle olisi Standard Edition 2, mikä sekin olisi jo 8 750 dollaria ilman mitään muita kuluja. Prosessorikohtainen hinnoittelu vaatii vähintään Enterprise Edition -järjestelmän ja tulisi tässäkin mielessä jo kalliimmaksi kuin käyttäjäkohtainen hinta.

4.1 IBM Db2

Ensimmäisenä vaihtoehtona on IMB:n kehittämä Db2-tietokantajärjestelmä. Db2 oli ensimmäisiä kaupallisia relaatiopohjaisia tietokantajärjestelmiä ja sen ensimmäinen versio julkaistiinkin jo vuonna 1983. IBM tarjoaa Db2:sta kolmea erilaista versiota: community, standard ja advanced. Community on Db2:n ilmaisversio, mutta samalla hieman rajoitetumpi. Ilmaisversio on rajattu käytetyn keskusmuistin (4 Gt RAM), prosessoriytimien (neljä) ja tietokannan maksimikoon (100 Gt) mukaan. Tätä versiota tarjotaan käytettäväksi joko Docker-konttina, erillisenä latauksena tai pilvipalvelun kautta hallinnoituna (jolloin tietokannan koko on rajattuna 100 Gt:n sijaan 100 Mt:a). Standard ja advanced ovat IBM:n tarjoamia maksullisia versioita samasta järjestelmästä. Advanced on käytännössä Db2 ilman minkäänlaisia rajoituksia. Standardissa on taas community-versioon nähden hieman suuremmat rajat keskusmuistin (128 Gt RAM) ja prosessoriytimien (16) kanssa. Standard-version hinta on kiinteästi 1 850 $ per VPC (Virtual Processor Core) ja advanced-version hinta tilauspohjainen 7 800 $ per VPC. Advanced-versio voidaan suoraan hylätä liian kalliina, mutta standard-versio saattaa olla hinnastaan huolimatta mahdollinen, ellei community-versio osoittaudu riittäväksi. [20], [21]

Valituista vaihtoehdoista Db2 tukee suurimpaa määrää eri käyttöjärjestelmiä (AIX, HP-UX, Linux, Solaris, Windows, z/OS), joten yrityksen palvelintilanteen kannalta Db2 olisi tässä mielessä paras vaihtoehto. Käyttöjärjestelmä vaikuttaa kuitenkin hieman siihen, mitä vaihtoehtoja Db2:n asennuksessa on. Linux- ja Unix-ympäristöissä jokainen instanssi on luotu dedikoidun käyttäjän alle, joka on instanssin omistaja. Tällaisen instanssin luodakseen on oltava pääkäyttäjä (root). Näissä ympäristöissä se tarkoittaa, että instanssit ovat rajoitettuja yhteen per käyttäjä. Windows-ympäristössä on taas mahdollista luoda useampi instanssi yhden käyttäjän alle. Db2 tukee myös suurta määrää eri ohjelmointikieliä (C, C#, C++, Cobol, Delphi, Fortran, Java, Perl, PHP, Python, Ruby, Visual Basic), mikä antaa rutkasti vaihtoehtoja siihen, miten VBA-skriptit voitaisiin kääntää uuteen järjestelmään. [22], [23]

Db2 tarjoaa mahdollisuudeksi asentaa tietokanta useammalle osiolle, jolloin tietokanta on jaettu useampaan fyysiseen serveriin tai samalle SMP-palvelimelle (symmetric multiprocessor). Työkuorma voidaan tällä tavalla jakaa useamman tietokannan kesken, jotka toimivat kuitenkin yhtenä yhtenäisenä tietokantana. Datan prosessoinnissa tämä mahdollistaa hyvän skaalattavuuden. Tällaista osioitua tietokantaa suositellaan nimenomaan, mikäli suunnittelee BI-tietokantaa. Tällöin 300 Gt:a kohti voidaan laskea yksi prosessoriydin. Yrityksen tietokannan tämänhetkinen koko ei tosin ole lähelläkään niitä määriä, joita tämän kokoluokan BI-tietokannat ovat ja yhden osion tietokannat ovat

mitä todennäköisimmin riittäviä useimpiin tietokantasovelluksiin, mutta mahdollisuus muuhunkin on tietysti hyvä olla olemassa. Datan määrä kuitenkin kasvaa koko ajan ja jossain kohtaa voi tulla ajankohtaiseksi toteuttaa järjestelmä useammalle osiolle. Näin vältettäisiin vastaavanlainen tilanne kuin mikä on nyt Access-tietokannan kanssa, jossa tietokannan rajat järjestelmän käytettävyyden kannalta ovat tulossa vastaan. [23]

IBM tarjoaa Db2-tietokannan hallintaan sitä varten kehitellyn työkalun. IBM Control Center on ilmaiseksi ladattavissa oleva hallinnointityökalu. Työkalun avulla voidaan hallita kerralla kaikkia Db2-instansseja ja isäntäpalvelimia, joten hajautettuakin tietokantaa voidaan hallita yhdellä kertaa. Näitä instansseja voi ohjelman avulla sammuttaa ja käynnistää yksitellen, mikä on mahdollisten ongelmatilanteiden kannalta tervetullut apukeino niiden ratkaisemiseen. Myös perustason CRUD-operaatiot (create, read, update, delete) tauluille on mahdollista toteuttaa työkalun kautta. Toinen työkalu, josta saattaisi hyvinkin olla hyötyä tietokannan hallinnoimisessa, on IBM Data Studio (entinen Developer Workbench). Data Studio mahdollistaa niin ikään tietokannan taulujen hallintaa, mutta mielenkiintoisena toimintona ohjelma tukee SQL-kyselyiden tallennusta ja automaattista ajoa joko yksittäin tai ryhmissä. Järjestelmän kannalta tämä voisi tarkoittaa kyselyiden automaattista ajoa sen jälkeen, kun tieto on ensin tuotu (mahdollisesti automaattisesti) sisään tietokantaan, mikä edelleen tukisi tavoitetta täysin automatisoidusta ympäristöstä. Ohjelmaan tehtyjä SQL-kyselyitä on mahdollista tallentaa myös proseduureina ja välitetyillä arvoilla niin, että kyselyn voi toteuttaa muun muassa etäyhteyden kautta vain kutsumalla proseduuria halutuilla arvoilla. Muunkinlaisia aputyökaluja ohjelmasta löytyy, kuten hälytykset, joilla voidaan asettaa ohjelma lähettämään viestiä erilaisissa tilanteissa (esimerkiksi jos tietokanta on alhaalla). Näistä ei kuitenkaan ainakaan tällä hetkellä vaikuttaisi olevan konkreettista hyötyä järjestelmän kokonaisuuden kannalta. Toki jos koko ympäristö saadaan automatisoitua suunnitelman mukaan, tällöin muun muassa hälytyksistä voi virhetilanteissa saada lisähyötyä. Virhetilanteiden varalta työkalu tarjoaa mahdollisuuden koko tietokannan varmuuskopiointiin (joko online tai offline). [24]–[26]

Db2-tietokantajärjestelmässä on varmuuskopioinnin lisäksi mahdollista suorittaa tietokannan palautus. Roll forward -komento taas mahdollistaa tietokannan palauttamisen käyttämällä tietokannan lokitiedostoihin tallentuneita tapahtumia. Tietokannan varmuuskopiointi on mahdollista automatisoida. Varmuuskopiointia tuetaan joko kovalevylle, kasetille, TSM-muodossa (Tivoli Storage Manager) sekä toimittajien DLL-medialuokille. Kovalevylle tehty varmuuskopiointi lienee käytetyimpien joukossa ja siihen kohdistuu järjestelmän puolesta rajoitus. Kovalevylle suoritettava automatisoitu varmuuskopiointi poistaa tasaisin väliajoin niin ikään automaattisesti

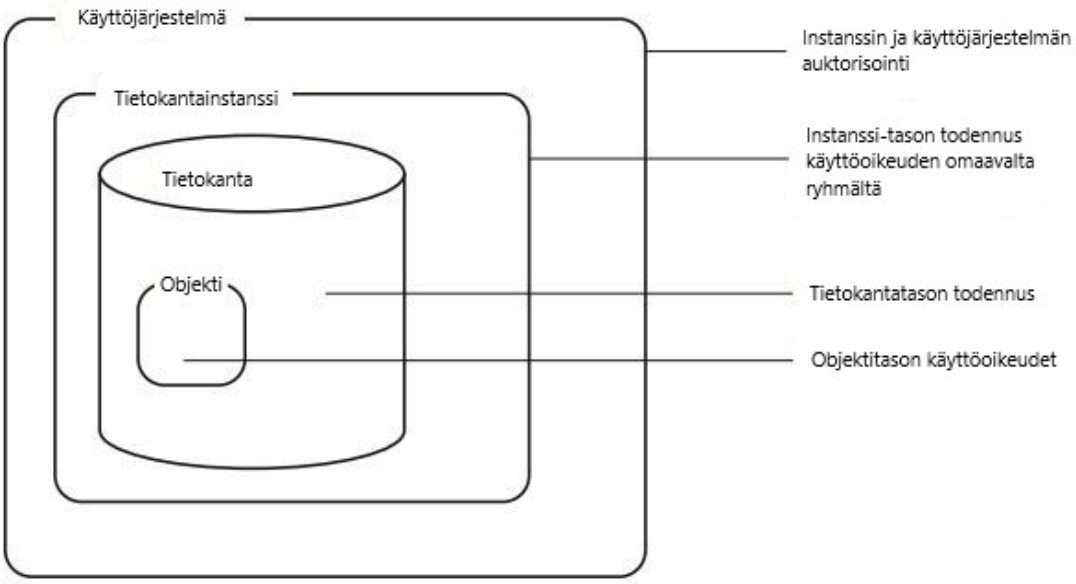
kaikki muut paitsi uusimman tietokannan varmuuskopion (sekä muut kansiossa olevat tiedostot). Tällainen toiminta voidaan lukea joko järjestelmän eduksi tai haitaksi. Mikäli ympäristö, johon tietokanta asennetaan, ei pidä sisällään suurta määrää tallennustilaa, voi olla kaivattu ominaisuus, etteivät vanhat varmuuskopiot ala varaamaan tilaa itselleen. Muutoin niitä pitäisi poistella manuaalisesti tai kirjoittaa jokin sovellus poistamaan varmuuskopioita tilan vähentyessä.

Toisaalta pelkästään yhdellä varmuuskopiolla on vaarana kohdata tilanne jossa tietokanta sekä varmuuskopio ovat viallisessa tilanteessa. Oletetaan esimerkkinä, että muut varmuuskopiot on juuri poistettu ja automatisoituun tietokantaan tulee tietoa illalla klo 23:00 varmuuskopioinnin alkaessa 0:00. Jos tämä uusi tieto sisältäisi dataa, mikä onnistuu korruptoimaan tietokannassa olevaa tietoa niin, ettei sitä pystytä enää käyttämään, voisi helpoin ratkaisu olla tietokannan palautus varmuuskopiosta. Mikäli varmuuskopiota ei kuitenkaan ole ehditty ottaa talteen klo 23:00 – 0:00 välisenä aikana on syntynyt tilanne, jossa sekä tietokanta, että varmuuskopio sisältävät korruptoituneen tiedon. Vastaavanlaisia tilanteita voidaan tietysti yrittää välttää asettamalla tietokantaan tulevan tiedon ja varmuuskopioinnin väli isommaksi ja estää uuden tiedon tuominen muun muassa viikonloppuisin jolloin henkilökuntaa ei välttämättä ole paikalla. Aiemmin mainitusta hälytysominaisuudesta voisi olla apua tällaisissa tilanteissa jos viesti korruptoituneesta tiedosta saadaan välittymään tietokannan ylläpitäjälle. Huolimatta varotoimenpiteistä on tällaisissa yhden varmuuskopion ratkaisuissa kuitenkin aina vaaransa ja vaikka tilanteita voidaan yrittää estellä, on inhimillisen erehdyksen mahdollisuus edelleen olemassa. Vaikka virhetilanteesta lähtisi ilmoitus, pitää jonkun se huomata ajoissa. Vastaavasti jos järjestelmä on toiminut pitkän aikaa ilman virhetilanteita voi ihminen helposti tuudittautua siihen, ettei tänäänkään mitään tapahdu ja jättää tietokannan uuden datan tarkastamatta heti sen tultua. Näin sitten voi myös jäädä huomaamatta datan korruptoituminen eikä varmuuskopiota ehditä ottaa talteen. Joka tapauksessa tällaiset Data Studion mahdollistamat varmuuskopioinnit ovat tärkeitä apuvälineitä, sillä niiden avulla on hyvinkin mahdollista onnistua korjaamaan vaikeitakin virhetilanteita. [26]

Komentokehotteen käyttö tietokannan hallinnassa on niin ikään mahdollista ja joissain tapauksissa mahdollisesti graafista työkalua parempi vaihtoehto. Komentokehotteen kautta on myös mahdollista ajaa erilaisia, komentoja, skriptejä ja suorittaa työtehtäviä sarjana. Tällainen mahdollisuus voi tuoda myös apua tilanteissa, joissa järjestelmää täytyy päästä käyttämään etäyhteyden kautta tai jos graafinen työkalu ei ole syystä tai toisesta käytettävissä. Db2:n komentokehotteen työkalut ovat liki identtisiä eri Linux-,

UNIX- ja Windows-ympäristöissä, joten ympäristöstä toiseen siirtyminen ei heti aiheuttaisi uusien komentojen opettelua, mikä taas säästää arvokasta aikaa. [24]

Kuten monessa muussakin tietokannassa, Db2:ssa pääsynhallinta ja oikeudet on jaettu kahteen erilliseen alueeseen (todennus ja auktorisointi). Todennus on prosessi, jonka avulla selvitetään, kuka käyttäjä on hänen suorittaessaan toimintoa, joka on rajattu vain tietyille käyttäjille. Mikäli todennus suoritetaan onnistuneesti (käyttäjä on kuka hän ilmoittaa olevansa) tapahtuu seuraavasti auktorisointi, jolloin selvitetään mitä käyttäjän on sallittua tehdä.



Kuva 8. *Db2:n todennus- ja auktorisointi-kerrokset.*

Kuvassa 8 nähdään miten todennus ja auktorisointi on toteutettu Db2-järjestelmässä. Oletusarvoisesti Db2 suorittaa todennuksen käyttäen hyödyksi käyttöjärjestelmässä olevia käyttäjiä ja ryhmiä. Tämä tarkoittaa samalla sitä, että mikäli tietokantaan halutaan luoda uusi käyttäjä, tämän käyttäjän pitää olla käyttäjänä myös käyttöjärjestelmässä. Turvallisuuden puolesta tätä voi pohtia kahdeltakin tasolta. Onko turvallisempaa, että jos ulkopuolinen taho saa käsiinsä minkä tahansa käyttöjärjestelmän tunnuksen, hän pääsee myös sisään tietokantaa vai onko parempi, että tällaisessa tilanteessa pitäisi vielä erikseen saada haltuunsa tietokannan käyttöön oikeuttava käyttäjätunnus salasanoineen? [24], [27]

Puhtaasti tietokannan käyttäjien kannalta katsottuna ensimmäinen tapa on varmasti se mieluisampi tilanne käyttömukavuudeltaan, sillä tällöin ei tarvitse muistaa kahta eri tunnusta ja salasanaa vaan pärjää yhdellä parilla. Käyttäjille voidaan myös asettaa eritasoisia oikeuksia tietokannan käyttöön, joten vaikka käytössä olisi pelkkä käyttöjärjestelmätason todennus, ei se välttämättä tarkoita, että millä tahansa

käyttäjätunnuksella saadaan koko tietokanta käyttöön. Db2:ssa on kuitenkin mahdollisuus ottaa käyttöön kehittyneempiä tapoja todennuksessa. Nämä pitää toteuttaa lataamalla kolmansien osapuolien lisäosia (plug-in), kuten Kerberos. Näiden lisäosien toteutus riippuu hieman siitä, asennetaanko niitä asiakas- vai palvelinpäätteelle. Asiakaspäätteet tukevat yhtä ryhmähaku-lisäosaa ja yhtä käyttäjä/salasana-todennuslisäosaa. Vaihtoehtona asiakaspäätteet joissa on käytössä GSS-API todennuslisäosa (Generic Security Service Application Program Interface) pystyvät selvittämään mitä lisäosaa tulee käyttää, skannaamalla palvelinpäätteellä olevan listan toteutetuista GSS-API-lisäosista. Ensimmäinen todennuslisäosa, joka täsmää asiakaspäätteellä toteutettuun GSS-API-todennuslisäosaan, tulee valituksi. Palvelinpäätteet taas tukevat yhtä ryhmähaku-lisäosaa, yhtä käyttäjä/salasana-todennuslisäosaa sekä useita GSS-API-lisäosia. [24], [27]

Ellei näitä lisäosia kuitenkaan haluta käyttää on Db2:n instanssi-kerroksella vielä erikseen mahdollista määrittää todennustyyppi, mikä vaikuttaa siihen millä tavalla käyttäjätunnus/salasanaparia kysytään tietokannan puolesta. Oletusarvona on ”server”, mikä tarkoittaa juuri aiemmin kuvattua käyttöjärjestelmätyyppistä todennusta. Muita vaihtoehtoja ovat muun muassa ”server_encrypt” joka on muunnos server-arvosta ja käyttäjien salasanat sekä ID:t kulkevat kryptattuina kun niitä lähetetään client-päätteeltä server-päätteelle. Data kulkee tässäkin tapauksessa selkokielisinä. Kryptaustavaksi on mahdollista valita AES 256 -algoritmin, mikä asiallisesti toteutettuna on varmasti tarpeeksi hyvä suoja salasanoille. Jos halutaan kryptata myös data, tämä on mahdollista asettamalla todennustyypiksi ”data_encrypt_cmp”, jolloin SQL-kyselyt, muuttujat (variables), palautuva data sekä viestit ja koodit liikkuvat kryptattuna. Kuvassa alimmaisena olevat object-tason auktorisointi tarkoittaa käyttäjille tai käyttäjäryhmille annettavia oikeuksia tietokannan tauluja kohtaan. Tällaisia ovat muun muassa CRUD-operaatiot, mutta myös muunlaisia vaihtoehtoja on olemassa, kuten alter-oikeus, mikä sallii muutokset taulun rakenteeseen ja määrittelyyn. [24], [27]

Käyttäjänhallinnassa Db2 erottuu hieman useimmista muista tietokannoista. Käyttäjille voidaan yhtä lailla laittaa erilaisia oikeuksia tietokannan tauluihin, kuten muokkaus-, poisto-, tai lukuoikeudet, mutta Db2-tulkitsee oikeuksia niin, että mikäli oikeutta ei ole myönnetty, se on automaattisesti kielletty. Järjestelmässä ei siis erikseen tarvitse kieltää vaikkapa taulujen poistoa käyttäjältä vaan jos hänellä ei ole oikeutta tauluja poistaa, ei järjestelmä sitä mahdollisuutta hänelle anna. Toisiin järjestelmiin verrattuna tällainen lähestymistapa estää tilanteet, joissa on tarve löytää hukassa oleva kieltoehto. Tällainen tilanne saattaa muissa järjestelmissä tulla vastaan, jos käyttäjä kuuluu ryhmään jolla on oikeus muokata tauluja, mutta käyttäjällä tämä on kiellettynä.

Säännön tulkitseminen tässä kohtaa riippuu tietysti järjestelmistä (ylittääkö ryhmän oikeudet käyttäjän oikeudet vai toisinpäin). Peruskäyttäjän puolesta on myös perusteltua, ettei hänellä ole enempää oikeuksia järjestelmään kuin ne mitä hän varsinaisesti tarvitsee. Ellei käyttäjällä ole tarvetta poistaa tauluja, on sellaista oikeutta hänelle turha antaa sillä se voi johtaa vain siihen, että jossain kohtaa käyttäjä poistaa tahattomasti taulun yrittäessään tehdä jotain muuta. [24]

Db2 pitää sisällään auditointiominaisuuden, mikä auttaa havaitsemaan tuntematonta tai odottamatonta pääsyä dataan. Tämä ominaisuus luo ja sallii ylläpidon niin kutsutulle auditointijäljelle (audit trail) ennalta määrätyille tietokantatapahtumille, kuten onnistuneet- tai epäonnistuneet tietokantaoperaatiot ja virhetilanteet. Tästä aiheutuvat tiedot tallennetaan erityisiin auditointilokeihin, joita ylläpito voi sitten analysoida paljastaakseen käyttötapahtumia, jotka täyttävät järjestelmän väärinkäytön kriteerit. Auditointi toimii erikseen sekä instanssi- että tietokantakerroksilla ja toiminnat molemmista tallennetaan niin ikään eri lokitiedostoihin. Ominaisuus on hyvä lisä turvallisuustyökaluihin, sillä vaikka kaikki olisi säädetty oikein, voi ulkopuolinen henkilö siltikin päästä järjestelmään jollain tasolla käsiksi. Tällöin hyvin kerätyistä lokitiedostoista on varmasti hyötyä tilanteen selvittämiseksi. Lisäksi lokitiedostoja voidaan käyttää muidenkin virhetilanteiden tutkimisen apukeinona. [23], [28]

Koska turvallisuutta on painotettu, lienee hyvä erikseen mainita, että Db2:n natiivit kryptausominaisuudet käyttävät hyväkseen IBM:n Global Security Kit:in (GSKit) SSL/TLS ja kryptografisia ominaisuuksia. Mikäli client- ja server-pääte on asennettuna samalle koneelle, myös GSKit on asennustiedoston mukana, mutta mikäli käytössä on eri koneille asennetut päätteet, voidaan GSKit joissain tapauksissa joutua asentamaan erikseen.

Db2 tukee Windowsin Active Directory -ominaisuuksia (AD), mikä on projektin kannalta ehdottoman myönteinen asia. Yrityksen käytössä oleva palvelinkoneella on käytössä nimenomaan AD-palvelu. Mikäli projekti saadaan toteutettua nykyiselle palvelimelle tai jos uusi palvelin on niin ikään Windows-palvelin AD-palvelulla, voidaan todennus tarvittaessa hoitaa sen kautta, eikä tarvita erikseen lisäosia.

Turvallisuuden puolesta voidaan mainita vielä yksi ominaisuus mikä saadaan Control Center -apuohjelman kautta. Tietokannan tauluja on mahdollista laittaa uinuvaan (quiesce) tilaan tai ottaa ne pois tästä tilasta. Käytännössä tämä tarkoittaa, että taulu otetaan pois normaaleiden toimintojen vaikutusalueelta ja taulun käyttöä tai siihen pääsyä rajataan joko käyttäjä- tai toimintokohtaisesti. Taulu siis laitetaan eräänlaiseen hallintatilaan, jolloin vain pääkäyttäjällä on oikeudet suorittaa tauluun toimenpiteitä.

Todennäköisesti tällaista toimintaa vaativat tilanteet eivät ole kovin usein tapahtuvia, mutta tämä on kuitenkin nopeasti suoritettava toimenpide, jonka voi tehdä etähallinnan kautta, joten on vähintään hyvä lisä turvallisuuspakkiin. [24]

Datan tuomiseen ja viemiseen Db2-kantaan on IBM:n puolesta tarjottu laajat vaihtoehdot. Db2 tukee datan tuomista/viemistä useassa formaatissa (ASCII erotinmerkillä tai ilman, WSF, PC/IXF, Cursor). Datan tuominen suoraan csv-muodossa on siis näin ollen mahdollista. Operaatio onnistuu Db2:n omilla työkaluilla (muun muassa Control Center), mutta ainakaan suoraan Db2 ei tue automaattista tiedon tuomista esimerkiksi kerran kuussa. Tämä on kuitenkin varmasti toteutettavissa vaikkapa Db2:n itse luotavien funktioiden kautta tai luomalla automaattisesti suoritettava skripti Linux-ympäristössä crontab-käskyn avulla. Data Studion automaattinen kyselyiden ajo voisi mahdollisesti tarjota tähänkin yhden tavan suorittaa automaatio. Kuitenkaan datan tuominen tietokantaan ei Db2:n kohdalla ole varmastikaan ongelmana. [24]

Vielä yhtenä valintaa puoltavana asiana Db2:n kohdalla voidaan mainita IBM:n Knowledge Center -sivu, joka pitää sisällään Db2:n dokumentaation. Dokumentaatiota tietysti löytää muidenkin valmistajien osalta, mutta IBM:n kohdalla pitää erikseen mainita sivujen toimivuus. Ne ovat selkeät ja navigoitavuus on äärimmäisen toimivaa. Sivujen kautta voi nopeasti vaihtaa muun muassa Db2:n versionumeroa ja mikäli sama ominaisuus, jota luetaan, löytyy myös vanhemmalta versiolta, ohjautuu sivu automaattisesti siihen sivuun. Ellei näin ole niin tästäkin mainitaan, eikä vastaan tule ”sivua ei löydy” -ilmoitusta. Tällaisesta voi mahdollisesti olla apua tilanteessa, jossa ei haluta päivittää tietokantaa uusimpaan mahdolliseen vaan pelkästään seuraavaan versioon, josta haluttu ominaisuus löytyy. Lisäksi tätäkin tutkielmaa tehtäessä eri instanssien ja tietokannan parametrien tarkoitusta selvitellessä ei tarvinnut kuin laittaa tutkittavan parametrin nimi sellaisenaan (esimerkiksi dftdbpath) sivun hakukenttään ja joka kerta päädyttiin halutulle sivulle. Helpottaa varmasti työtaakkaa kun tarvittava tieto löytyy muutamassa sekunnissa valmistajan omilta sivuilta eikä tarvitse turvautua ulkopuolisiin hakukoneisiin tai hakea käsin ohjekirjasta. Tämä on todennäköisesti asia, jossa Db2:n kaupallisuus näkyy hyvänä puolena; kun tuotetta myydään rahaa vasten, on sillä kannattavaa olla hyvin toimiva tukisivusto.

4.2 Microsoft SQL Server

Toisena vaihtoehtona on Microsoftin SQL Server. SQL Server on niin ikään suhteellisen vanha tietokantajärjestelmä, sen ensimmäinen versio julkaistiin jo vuonna 1989. Db2:n tapaan SQL Server on kaupallinen tietokantajärjestelmä ja Microsoft

tarjoaa siitä jopa kuutta erilaista versiota. Kaksi versiota on ilmaiseksi ladattavissa: Express ja Developer. Näistä kahdesta Express on tietokannan pienoisversio, jonka avulla tietokantajärjestelmää on mahdollista testata tai käyttää pienempiin projekteihin. Express-versio on rajattu 10 Gt:n kokoisiin sovelluksiin, joten vielä tällä hetkellä olisi mahdollista ennen hankintaa testata SQL Serverin toimivuutta ilmaisversion avulla. Developer-versio on taas kehittäjille tarkoitettu versio erilaisten SQL Serveriin perustuvien sovellusten testaamiseen, joten sillä ei tämän projektin kannalta ole suurta merkitystä. Maksullisia versioita SQL Serveristä on kolme erilaista: Enterprise sekä kaksi erilaista Standard-versiota: ydinkohtainen ja palvelin + CAL (Client Access License). Enterprise on näistä laajin ja suuriin tietomääriin suunniteltu versio. Samalla se on tietysti myös kallein vaihtoehto 14 256 $:n hinnallaan (per ydin). [29]

Microsoftin hinnoittelussa on todettava sekin, että annetut myyntihinnat ovat Open No Level (NL) -tason arvioituja jälleenmyyntihintoja ja tarkan hinnan saadakseen, pitää tarkistaa jälleenmyyjien hintoja. Lisäksi ydinkohtaiseen hinnoitteluun perustuvat mallit ovat saatavilla kahden ytimen paketteina. Enterprise-versio voidaan Db2:n kalleimman version tapaan rajata suoraan ulos vaihtoehdoista liian kalliina. Kaksi jäljelle jäävää standard-versiota eroavat toisistaan hinnoittelun perusteella. Ydinkohtainen versio maksaa samalla NL-hinnoittelumallilla 3 717 $:a per ydin. Server + CAL -versio on taas lisenssikohtainen ja jokainen käyttäjä vaatii oman lisenssinsä. Lisenssejä voi ostaa laite- tai käyttäjäkohtaisina. Hinta tälle versiolle on 931 $:a, mutta mukaan pitää laskea myös lisenssit. Lisenssin hinta riippuu jälleenmyyjästä, mutta Microsoft tarjoaa omassa Microsoft Store -kaupassaan yhtä käyttäjä tai laitekohtaista CAL-lisenssiä 243 €:n hintaan. Aivan alkuun järjestelmässä ei ole kuin yhdelle tai maksimissaan kahdelle käyttäjälle ja on myös mahdollista, että lopullinen versio pyörii vain yhdellä laitteella, jolloin lisenssipohjainen versio tulisi halvimmaksi vaihtoehdoksi. Laitteistotasolla standard-versio tukee enintään neljää prosessoria ja 2 Tt:a keskusmuistia. Viimeisenä versiona on Web-versio, mutta kyseinen versio on ainoastaan julkisia verkkosivuja varten ja saatavilla ainoastaan kolmansien osapuolien ohjelmistopalveluntarjoajille. Web-versio voidaan siis suoraan rajata pois. [29], [30]

Tuettujen käyttöjärjestelmien puolesta SQL Server on listan viimeisenä. Tuettuina ovat vain Windows- ja Linux-alustat. Nämä kaksi vaihtoehtoa kyllä itsessäänkin tarjoavat useita erilaisia käyttöjärjestelmäversioita (varsinkin Linuxin puolella), joten tilanne ei kuitenkaan ole huono tässäkään tilanteessa. Lisäksi Microsoft tarjoaa Docker-konttia yhtenä asennusvaihtoehtona. Tuetuissa ohjelmointikielissä SQL Server on niin ikään listan viimeisenä, häviten Db2:lle yhdellä (C#, C++, Delphi, Go, Java, JavaScript (Node.js), PHP, Python, R, Ruby, Visual Basic). Vaihtoehdot ovat liki identtiset

Db2:een ja merkittävin ero lienee C-kielen tuen puuttuminen SQL Serveristä. Toisaalta SQL Serveristä löytyy JavaScript-tuki jota ei Db2:ssa ollut. Käyttöjärjestelmä- ja kielituesta ei siis jää valinta kiinni SQL Serverinkään kanssa. [31]

Microsoft tarjoaa useampiakin hallintatyökaluja SQL Serveriä varten. Näitä ovat SQL Server Management Studio (SSMS), SQL Server Configuration Manager, Database Engine Tuning Advisor, Data Quality Client ja SQL Server Data Tools (SSDT). Näistä ohjelmista kaikki muuta paitsi SSMS ja SSDT tulevat SQL Server -asennuspaketin mukana. SSMS on ladattavissa ilmaiseksi, mutta ohjelma on vain Windows-käyttöjärjestelmille. SSDT on taas käytettävissä yhdessä Visual Studion kanssa. Projektin kannalta mielenkiintoisimmat ohjelmat ovat SSMS ja SQL Server Profiler. [30]

SSMS vertautuu Db2:n Data Studio -ohjelmaan ja SSMS-ohjelmankin käyttötarkoituksena on SQL Serverin ylläpito ja hallinta. SSMS-ohjelman kaksi päätarkoitusta on tietokantapalvelimien (instanssien) ylläpito ja tietokannan objektien (taulut, näkymät, indeksi, ynnä muut vastaavat) hallinnointi. Data Studion tavoin SSMS mahdollistaa yhdellä hallintaikkunalla yhteydet yhteen tai useampaan palvelimeen, joten useampiin instansseihin on mahdollista päästä kiinni yhdellä kerralla. Tietokantapalvelimiin on mahdollista kohdistaa SSMS-ohjelmalla useita ylläpitotoimia, joista muutamia ovat seuraavat: palvelimen rekisteröinti, palvelimeen yhdistäminen, uuden palvelinryhmän luonti sekä palvelimien käynnistäminen ja pysäyttäminen. Projektin kannalta hyödyllisimpiä ovat tässä kohtaa palvelimen rekisteröintimahdollisuus asennusvaiheessa sekä mahdollisuus palvelimen käynnistämiseen ja pysäyttämiseen, mikä tuo vikatilanteiden selvittelyyn hieman lisäapua. SSMS:n hallintatyökaluihin kuuluvat ne perinteiset: tietokannan luominen ja muokkaaminen, tietokannan taulujen hallinta ja SQL-kyselyiden luominen ja suorittaminen. Huomioitavaa on, että SQL-kyselyitä lukuun ottamatta, näiden toimenpiteiden suorittaminen ei vaadi Transact-SQL:n käyttöä, mikä on Microsoftin ja Sybase-yrityksen omistama jatke SQL-kieleen ja muutoin hyvinkin sidoksissa SQL Serveriin. Toki vastaavien toimenpiteiden suorittaminen T-SQL-kieltä käyttäen on täysin mahdollista. SQL-kyselyitä varten SSMS pitää sisällään myös SQL debuggerin. Debugger on aina hyvä ominaisuus missä tahansa ohjelmistossa sillä kyselyihin tulee väkisinkin aika ajoin virheitä ja debuggerin avulla niitä on monta kertaa helpompi selvitellä kuin vain silmämääräisesti koodia tutkimalla. [30], [32]

Kyselyiden automaattiseen ajamiseen SSMS ei suoraan sovellu, mutta SQL Serverin asennuksessa tulee mukana SQL Server Agent -palvelu, jolla voi ajastaa erilaisia työtehtäviä (kuten SQL-kyselyiden ajon). Kyselyiden automatisointi voidaan siis suorittaa tallentamalla ensin kyselyt SSMS-ohjelmaan ja tämän jälkeen ajastamalla

niiden suoritus SQL Server Agent -palvelun avulla. Oletuksena SQL Server Agent on poissa päältä, joten se pitää ensin aktivoida, mikäli sitä halutaan käyttää. SQL Server Agentin kautta on mahdollista ajastaa myös tietokannan varmuuskopiointi, joten sen avulla on mahdollista automatisoida kaksi hyvinkin tärkeää projektin toimenpidettä. Server Agent pitää sisällään myös hälytysominaisuuden virhetilanteiden varalta (esimerkiksi varmuuskopioinnin epäonnistuessa), mikä sekin on oiva työkalu tietokannan toimivuuden ylläpitämisessä. [33]

SQL Server Profiler on taas käyttöliittymä, jonka avulla on mahdollista seurata tietokantainstanssia tai analysointipalveluita (analysis services). Seurantaa on mahdollista tallentaa tapahtuma kerrallaan tiedostoon tai tietokannan tauluun ja se on näin analysoitavissa jälkikäteen. Server Profileria käytetään normaalisti vaikkapa hitaasti tapahtuvien SQL-kyselyiden löytämiseen ja diagnosoimiseen tai tallentamaan sarja T-SQL-kyselyitä, jotka johtavat johonkin ongelmatilanteeseen. Tällä tallennetulla sarjalla voidaan toistaa ongelmatilanne toisella testikäyttöön tarkoitetulla palvelimella ja näin tutkia mikä virhetilanteen aiheuttaa. Server Profiler tukee myös SQL Serverin eri instansseissa suoritettujen toimenpiteiden auditointia. Turvallisuuteen liittyvissä tapahtumissa auditointi tallennetaan myöhempää tarkastelua varten. Server Profiler pystyy seuraamaan useita tapahtumia, mutta turvallisuuden kannalta merkityksellisimmät ovat sisäänkirjautumisten (onnistuneiden ja epäonnistuneiden), tietokannan objektien lukkojen tilan ja turvallisuusoikeuksien tarkistusten seuranta. [34]

Todennukseen SQL Server tarjoaa kahta eri vaihtoehtoa: Windows- ja mixed-tila. Käytännössä nämä kaksi tilaa hoitavat asiat kutakuinkin samalla tavalla kuin Db2. Windows-tilassa todennukseen käytetään käyttöjärjestelmän käyttäjätunnuksia eikä SQL Serveriä varten vaadita omaa käyttäjätunnus/salasanayhdistelmää. Windows-tila käyttää oletuksena Kerberos-suojausprotokollaa, tarjoaa salasanojen vahvuutta valvovan käytännön, tukee tilien lukitsemista ja mahdollistaa salasanojen vanhenemisen (pakottaa käyttäjän vaihtamaan salasanansa). Mixed-tilassa taas vastavuoroisesti käytetään todennukseen SQL Serverissä luotuja käyttäjätunnuksia, eikä käyttöjärjestelmän tunnuksilla pääse sisään. Mixed-tila tukee niin ikään salasanojen vanhenemista sekä vahvaa salasanaa valvovan käytännön, mutta se ei tue Kerberos-suojausprotokollaa. Microsoft toteaa itsekin omassa ohjeistuksessaan, että Windows-tila on näistä kahdesta turvallisempi ja sitä tulisi käyttää aina kun mahdollista. Mixed-tilaa joutuneekin käyttämään lähinnä siinä tilanteessa, että SQL Server on asennettu jollekin muulle kuin Windows-käyttöjärjestelmää käyttävälle koneelle tai jos käytetään jotain vanhempaa kolmannen osapuolen ohjelmaa, jotka vaativat SQL Server todennuksen toimiakseen. Yrityksen palvelintilanteen kannalta

tämä on hyvä ottaa huomioon uuttaa palvelinta hankittaessa, mikäli valinta osuu SQL Serveriin. [30], [35]

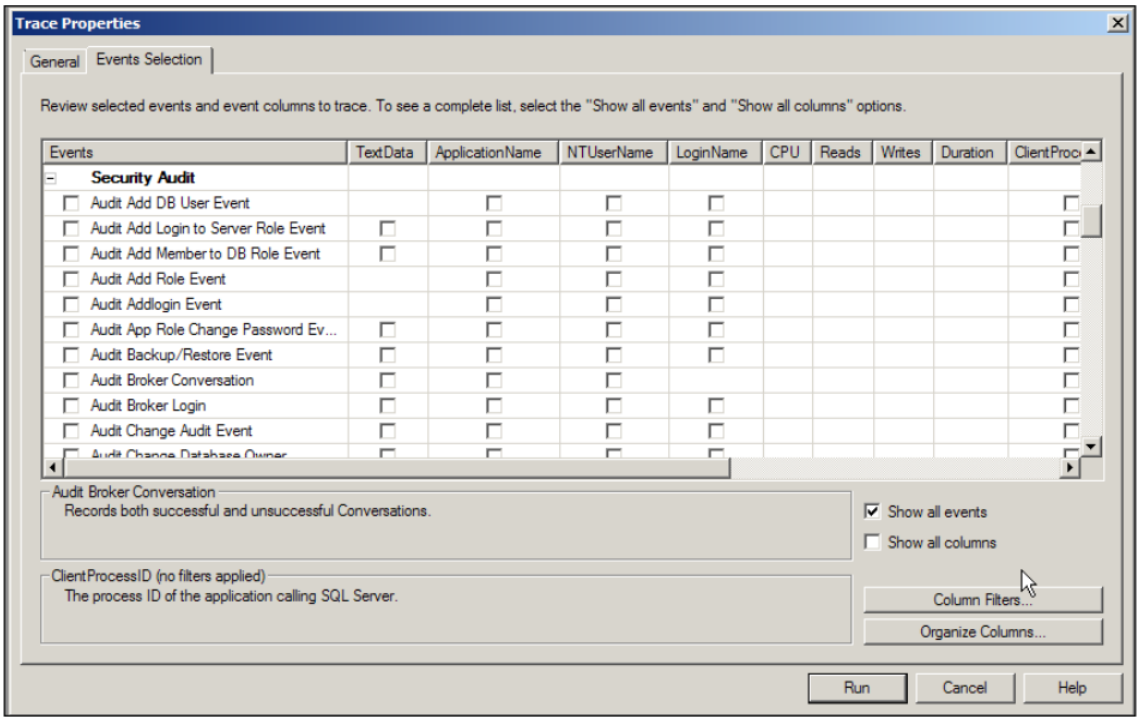
Active Directory ja SQL Server ovat molemmat Microsoftin tuotteita, joten ei olekaan yllättävää SQL Serverin tapauksessa, että se tukee myös AD-ympäristöä ja voi todentaa käyttäjiä sitäkin kautta. AD-ympäristöissä on SQL Serverin kohdalla mahdollista myös aktivoida epäonnistuneiden sisäänkirjautumisten tapauksessa tilien lukitseminen. Epäonnistuneiden kirjautumisten määrä ennen lukitusta on asetettavissa itse haluttuun määrään. [36]

Datan kryptaukseen SQL Server tarjoaa kahta eri vaihtoehtoa: transparent data encryption (TDE) ja always encrypted. Ensimmäinen vaihtoehto kryptaa tietokannan tiedostot automaattisesti, eikä se vaadi muutoksia muihin sovelluksiin. Tietokanta on tällä tavoin suojattu niin, että vaikka ulkopuolinen taho saisi sen haltuunsa, hän ei automaattisesti pääse siihen käsiksi. Jälkimmäinen vaihtoehto taas kohdistuu tiettyihin tietokannan kolumneihin. Tämä on tarkoitettu nimenomaan arkaluontoisen tiedon (henkilötunnukset, luottokorttien numerot, ynnä muut vastaavat) suojaamiseen ja tällä tavalla suojattuna niin tietokanta kuin tietokantapalvelinkaan eivät näe kryptaamattomia arvoja suojatuissa kolumneissa. Yrityksen projektissa tietokantaan tallennettava tieto ei ole kuitenkaan näin arkaluontoista joten TDE-salaus on projektin tapauksessa riittävä. Salausalgoritmeiksi SQL Server tarjoaa useampaakin eri vaihtoehtoa (DES, Triple DES, TRIPLE DES 3KEY, RC2, RC4, RC4 128, DESX, AES 128, AES 192, ja AES 256), joskin näistä kaikki muut paitsi AES 128, AES 192, ja AES 256 on määritelty SQL Server 2016 versiosta alkaen vanhentuneiksi. Kuten Db2:n kohdallakin todettiin, AES 256 on varmasti riittävä salausalgoritmi, joten pelkästään nämä kolme vaihtoehtoakin ovat riittävät projektin kannalta. [30], [37]–[39]

Käyttäjänhallinnassa SQL Serverissä on asiaankuuluvat vaihtoehdot. Käyttäjille voidaan antaa oikeuksia tai rajauksia eri tietokantojen tai taulujen suhteen. Voidaan luoda omia käyttäjäryhmiä tai käyttää esimerkiksi käyttöjärjestelmästä jo löytyviä. Itse tehtyjen roolien luominen on myös mahdollista, mikä voi tietyissä tilanteissa tuoda helpotusta käyttäjänhallintaan. Voidaan luoda vaikkapa rooli jolla on oikeus käyttää tiettyä tietokantaa ja sitten lisätä tämä rooli halutuille käyttäjille. Sama tietysti onnistuu luomalla uusi ryhmä ja osittain itse tehdyt roolit ja ryhmät menevätkin päällekkäin. Käyttäjänhallintaa pystyy hoitamaan SSMS-ohjelman avulla. Mikäli käyttäjällä on estettynä jokin tietty toiminto, mutta hänet on samalla laitettu rooliin, jolla oikeus on olemassa ottaa tämä kielto korkeamman prioriteetin. Tilanne on siis vastakkainen Db2:een verrattuna, jossa kaikki oli estettynä, ellei sitä ollut erikseen kielletty. SQL Serverissä oikeuksia voidaan antaa tai niitä voidaan kieltää, joten tällaiset tilanteet,

joissa käyttäjältä on kiellettynä toimenpide, jonka hänen roolinsa puolesta pitäisi pystyä tekemään, ovat mahdollisia. Tällöin ainoa ratkaisu on poistaa käyttäjältä toimenpiteen estävä kielto. Jos tietokannassa on useita käyttäjiä asian selvittäminen voi viedä oman aikansa. Poikkeuksen tähän toimintojen estämiseen tuo se tilanne, mikäli käyttäjällä on ylläpitäjän rooli. Tämä ottaa kaikkein korkeimman prioriteetin, jolloin ylläpitäjältä ei voi poistaa oikeuksia. Tämä estää sen, että ylläpitäjä voisi vahingossa lukita itsensä ulos tietokannasta. [30], [36], [40]

SQL Server tarjoaa myös työkaluja, joilla voidaan hoitaa auditointia. Yksi vaihtoehto on käyttää kytkimiä (trigger) jotka reagoivat tauluun tehtyihin insert, update tai delete -kyselyihin. Käyttäjät eivät voi ohittaa näitä kytkimiä ja ellei kytkin itsessään lähetä viestiä käyttäjälle, ei käyttäjä myöskään ole tietoinen kytkimen toimenpiteistä. Kytkimiä käyttäessä pitää kuitenkin huomioida, että niillä voi olla vaikutusta tietokannan suorituskykyyn. Muitakin vaihtoehtoja auditointiin kuitenkin löytyy muun muassa SSMS-ohjelmasta.



Kuva 9. *SSMS-ohjelman auditointimahdollisuuksia.*

Kuten kuvasta 9 nähdään, ohjelma tarjoaa monia erilaisia tapahtumia, jotka voidaan asentaa seurantaan. Projektin kannalta tärkeimpiä tai mielenkiintoisimpia ovat tiedot varmuuskopioinnin tai palautuksen tapahtumisesta, tietokannan muuttumisesta, luomisesta tai poistamisesta, tietokannan omistajatiedon muutoksesta sekä GDR-komennon (grant, deny, revoke) suorittamisesta tietokannan objektille. Näillä

vaihtoehdoilla voidaan jo tukea tietokannan turvallisuutta ja pienentää virheiden mahdollisuutta muun muassa varmuuskopioinnin puutteellisuudessa. [36], [40]

T-SQL pitää sisällään bulk insert -komennon, mikä mahdollistaa tiedon viemisen tietokannan tauluun tai näkymään. Tämä tieto on mahdollista tuoda ulkopuolisesta tiedostosta (muun muassa csv-formaatista), joten sitä voidaan käyttää yrityksen tietokannasta otettujen tiedostojen tuomiseen. Alla on Microsoftin esimerkki komennon toiminnasta tiedoston tuomisessa.

```
BULK INSERT Sales.Orders
FROM '\\SystemX\DiskZ\Sales\data\orders.csv'
WITH ( FORMAT='CSV');
```

Ohjelma 1. *Esimerkki bulk insert -komennosta T-SQL-formaatissa.*

Tiedoston tuominen on siis suhteellisen yksinkertainen toimenpide ja koska kyseessä on puhdas tietokantakysely, voidaan tätä käyttää hyväksi automatisoinnissa. Tarvittavat tuontikyselyt voidaan tehdä valmiiksi ja tallettaa SSMS-ohjelman avulla, minkä jälkeen niiden automaattinen suorittaminen voidaan toteuttaa SQL Server Agent -palvelun avulla. [41]

IBM:n tapaan myös Microsoft tarjoaa omaa dokumentaatiosivustoaan (Microsoft Docs), josta voi etsiä apua ja ohjeita SQL Serverin käyttämiseen. Microsoftin versio tällaisesta ohjesivustosta on myös tuotteen käyttäjän kannalta hyvin toteutettu. Sivusto on selkeä ja IBM:n sivuston tapaan hakupalkilla löytää normaalien hakutermien lisäksi hakemalla pelkästään parametrien nimellä. Sivustolta voi IBM:n sivun tapaan myös vaihtaa käytössä olevaa SQL Serverin versiota tarkastaakseen onko ominaisuutta tuettu halutussa versiossa. Tämän ominaisuuden vaikutus SQL Server 2014 -versiota vanhempien versioiden kanssa on tosin kyseenalaista. Tätäkin tutkielmaa varten Microsoft Docs -sivusto toi korvaamatonta apua, mutta yhtään sellaista sivua ei tullut missään vaiheessa vastaan, joka olisi ollut olemassa myös vanhemmille versioille, vaan tällöin ohjattiin aina uusimman version sivulle. Tämä ei kuitenkaan ole kovin merkittävä ongelma, sillä mikäli valinta osuu SQL Serveriin, tulee hankittava versio olemaan 2014-versiota uudempi jo saatavuudenkin puolesta. Tällöin version tarkastusta tehdäänkin lähinnä toiseen suuntaan siinä kohdassa kun SQL Serveristä tulee uusia versioita ja tulee tarve tarkistaa löytyikö jokin ominaisuus vielä siitä versiosta mikä on käytössä.

4.3 MySQL

Kolmantena vaihtoehtona uudeksi järjestelmäksi on MySQL. Käsitellyistä tietokantajärjestelmistä MySQL on nuorin, mutta senkin ensimmäinen versio nähtiin jo vuonna 1995. Muista poiketen kyseessä on avoimen lähdekoodin järjestelmä (GNU GPL -lisenssi), mutta järjestelmästä on saatavilla kaupallisiakin versioita. Lisäksi MySQL on mahdollista asentaa Docker-kontin avulla. Kaupallisia versioita on kolme: Standard Edition, Enterprise Edition ja Cluster CGE. Nämä eroavat toisistaan hinnaltaan sekä sisällöltään. Sisällöltään suurin ero Standard ja Enterprise versioiden välillä on, että Enterprise-version mukana tulee graafisen käyttöliittymän työkaluja, joita on suunniteltu muun muassa helpottamaan tietokantajärjestelmän tarkkailua (MySQL Enterprise Monitor), suojaamaan sen turvallisuutta (MySQL Enterprise Security) ja hallitsemaan varmuuskopiointia (MySQL Enterprise Backup). Cluster CGE -versio eroaa taas Enterprise-versiosta siinä, että Enterprise-version ominaisuuksien lisäksi siinä on mukana NDB-tietokantamoottori ja MySQL Cluster Manager -ohjelma. [42]

Hinnoittelun osalta MySQL saa vertailun selvimmän kandidaatin palkinnon. Kaikki kaupalliset versiot ovat vuosimaksullisia, eikä mukana ole mitään rajauksia tehojen tai prosessorien ytimien suhteen. Standard-versio maksaa 2 000 dollaria, Enterprise-versio 5 000 dollaria ja Cluster CGE -versio 10 000 dollaria vuodessa. Kuten hinnastakin huomataan, Cluster CGE -versio on tarkoitettu suuren luokan tietokantajärjestelmiin ja se voidaan tämän projektin osalta sivuuttaa niin hinnan kuin kapasiteetinkin puolesta, joka projektin mittasuhteisiin verrattuna on suuresti ylimitoitettu. Enterprise-version hinta mitä todennäköisimmin menee yli budjetin ja sekin voidaan projektin osalta ohittaa. Standard-versio taas voisi hintansa puolesta olla mahdollisuus, mutta sen erot ilmaiseen Community Edition -versioon ovat varsin pienet. Käytännössä Standard-versiolla saa Commynity-versioon verrattuna kolme lisäsertifikaattia ja muutaman ominaisuuden lisää MySQL Workbench -ohjelmaan liittyen skeeman ja mallin validointiin sekä dokumentaatioon luomiseen keskittyvän osan (DBDoc). Tämän tutkielman kannalta keskitytäänkin pääasiassa ilmaiseen Community-versioon ja sen ominaisuuksiin. Koska MySQL ei rajoita tietokantansa suorituskykyä versioiden mukaan on Community-versiokin varmasti riittävä projektin tarpeisiin. [42], [43]

Käyttöjärjestelmätukea tarkastettaessa MySQL sijoittuu juuri Db2:n taakse, joka tukee lukumäärällisesti yhtä järjestelmää enemmän. MySQL:n tuetut käyttöjärjestelmät ovat FreeBSD, Linux, OS X, Solaris ja Windows. Samalla se on vertailun ainoa järjestelmä joka tukee OS X -käyttöjärjestelmää. Tuettujen kielien osalta MySQL nappaa taas kirkkaasti kärkipaikan 19 tuetun kielen turvin (Ada, C, C#, C++, D, Delphi, Eiffel,

Erlang, Haskell, Java, JavaScript (Node.js), Objective-C, OCaml, Perl, PHP, Python, Ruby, Scheme ja Tcl). Puhtaasti tuettujen järjestelmien ja kielten kannalta katsottuna MySQL olisi näin ollen järkevin ratkaisu.

MySQL on muidenkin tietokantajärjestelmien tavoin hallittavissa komentoriviltä ja se saattaa olla MySQL:n yleisin hallintatapa edelleen. Graafisen käyttöliittymän hallintaohjelmia on kuitenkin olemassa myös MySQL:n tarpeisiin niin virallisia kuin kolmannen osapuolen tekeminä. Avoimen lähdekoodin vaikutus näkyneekin juuri suuressa määrässä kolmannen osapuolen ohjelmia, joita on saatavilla. Muutamia suosittuja ovat muun muassa Adminer, DBeaver ja phpMyAdmin. Osa näistä ohjelmista tukee muitakin järjestelmiä (esimerkiksi DBeaver tukee kaikkia kolmea tässä tutkielmassa tarkasteltua tietokantajärjestelmää), joten niitä on mahdollista käyttää valinnasta riippumatta. Virallinen hallintatyökalu MySQL-järjestelmään on taas MySQL Workbench, joka on hyvinkin verrattavissa aiemmin käsiteltyihin Data Studioon ja SSMS-ohjelmaan.

Workbench-ohjelman kautta onnistuu muiden käsiteltyjen ohjelmien tavoin yhteydet yhteen tai useampaan palveluun, joten eri instanssien hallinta hoituu MySQL-järjestelmässäkin tarvittaessa yhdellä ohjelmalla. Workbench-ohjelma tarjoaa aloitussivullaan pienen yhteenvedon tietokannan tilasta, jonka avulla voidaan tarkastaa niin palvelimen tämän hetkinen tila, kuin statistiikkaa sen käytöstä (prosessorin ja muistin käyttö, verkkoliikenne, ynnä muut vastaavat). Palvelimien käynnistäminen ja sammuttaminen onnistuu ohjelman kautta, mikä on edelleen hyvä tilanne mahdollisten huoltotoimenpiteiden tai vikatilanteiden kannalta. Tietokantoihin voidaan kohdistaa ohjelman kautta normaaleja toimenpiteitä CRUD-operaatioiden, käyttäjänhallinnan (luominen, poistaminen, muokkaaminen) ja SQL-kyselyeditorin kautta. SQL-editori pitää sisällään SQL-debuggerin koodin tarkastelua varten. Workbench ei itsessään tue kyselyiden automaattista ajamista, mutta se saadaan toteutettua Event Schedulerin avulla, mikä tulee suoraan MySQL version 8 mukana. Oletuksena ominaisuus on kuitenkin kytketty pois päältä, joten se pitää ensin ottaa käyttöön jos sitä halutaan käyttää. Kuten aiemmin mainittu, varmuuskopioinnin mahdollisuus ja sen automatisointi on vain Workbench-ohjelman Enterprise-versiossa. Mikäli Community-versiossa halutaan suorittaa automatisoitu varmuuskopiointi, pitää se tehdä kolmannen osapuolten ohjelmien avulla (esimerkiksi AutoMySQLBackup). Mikäli automatiikkaa ei tarvita, onnistuu varmuuskopiointi myös phpMyAdmin-ohjelman kautta. [44]–[46]

Yksi erikoisemmista Workbench-ohjelman ominaisuuksista on Database Migration -ominaisuus. Tämän ominaisuuden avulla voidaan tuoda olemassa oleva tietokanta toisesta järjestelmästä suoraan MySQL-tietokantaan. Microsoft Access on niiden

tietokantajärjestelmien joukossa, joita Database Migration -ominaisuus tukee, joten projektin kannalta on mahdollista yrittää tuoda koko nykyinen tietokanta sellaisenaan suoraan MySQL-järjestelmään. Operaation onnistumisesta ei tietysti voi sanoa mitään ennen sen kokeilemista, mutta jos valinta osuu MySQL-järjestelmään, on se varmasti ainakin yrittämisen arvoista sillä siinä voidaan parhaassa tapauksessa säästää merkittävästi aikaa. SQL Server on niin ikään tätä ominaisuutta tukevien tietokantajärjestelmien joukossa, joten jos projektin alustaksi valittaisiin SQL Server ja jälkeenpäin siirryttäisiin kuitenkin MySQL-järjestelmään, olisi tämä vaihtoehto edelleen olemassa. [44]

MySQL tukee AD-ympäristön kautta tapahtuvaa todennusta, mutta tuki on rajattu vain Enterprise-versioon. Community-version kanssa joudutaan siis tyytymään todennuksen toteutuksessa muihin vaihtoehtoihin. Niin ikään Kerberos-suojausprotokolla, joka MySQL-järjestelmään saadaan asentamalla PAM-laajennus (PAM Pluggable Authentication), on rajattu vain Enterprise-versioon. Vakiona MySQL-järjestelmässä on versiosta 8 alkaen ollut käytössä SHA-256 todennus. SHA-256 ei ole turvattomimmasta päästä, mutta tehtyjen tutkimusten mukaan se on murrettu 52 kertaa 64 yrityksestä preimage-hyökkäysmenetelmällä ja 46 kertaa 64:stä collision-hyökkäysmenetelmällä. Mikäli siis ehdottomasti halutaan toteuttaa todennus virallisten työkalujen kanssa AD-ympäristössä tai Kerberos-suojausprotokollalla, on ainoana vaihtoehtona hankkia Enterprise-versio. [47]–[49]

Käyttäjänhallinnassa voidaan turvallisuutta tukea muun muassa asettamalla käyttäjien salasanoihin vanhenemisaika. Salasanalle voidaan antaa muitakin kriteereitä, kuten minimipituus ja pienten sekä isojen kirjainten minimimäärä. Käyttäjille annettavat oikeudet MySQL jakaa kolmeen eri ryhmään: ylläpitäjien oikeudet, tietokantaoikeudet ja oikeudet tiettyihin tietokantaobjekteihin. MySQL lukee käyttäjien oikeudet muistiinsa niitä koskevasta taulusta käynnistyksen aikana, joten jos käyttöoikeuksia muutetaan, pitää tietokanta uudelleenkäynnistää. MySQL ei oletuksena sisällä käyttäjille suunniteltuja rooleja, mutta niiden luominen on mahdollista. MySQL-järjestelmässä voidaan luoda rooli, joka merkitään pakolliseksi (mandatory role). Palvelin käsittelee tällaista roolia sellaisena, mikä on automaattisesti annettuna kaikille käyttäjille. Näin voidaan antaa vaikkapa luku- ja kirjoitusoikeudet tiettyyn tauluun kaikille tietokannan käyttäjille yhtäaikaisesti, eikä roolia tarvitse erikseen kaikille antaa. [46], [50]

MySQL-järjestelmän turvallisuutta voi parantaa MySQL:n omilla työkaluilla. Asennuksen jälkeen voi ajaa mysql_secure_installation-työkalun. Työkalu käy läpi turvallisuuteen liittyviä asioita tietokannassa ja mahdollistaa muun muassa sellaisten root-tunnusten poistamisen, joihin on mahdollista päästä käsiksi paikallisen koneen

ulkopuolelta sekä sellaisten tilien poistamisen, joilla on mahdollista päästä käsiksi tietokantaan ilman, että heille pitää luoda käyttäjätunnusta (esimerkiksi testaamiseen tarkoitetut tilit). Toinen työkalu (mysql_config_editor) taas tarjoaa parannusta komentokehotteen kautta tapahtuvaan todennukseen. Työkalu tallentaa kirjautumiseen käytettävän salasanan kryptatussa muodossa tiedostoon, jota voi sitten käyttää tietokantaan kirjautumisessa ilman, että komentokehotteelle syöttää salasanaansa selväkielisenä. Tällainen kirjautuminen on mahdollista ilman työkaluakin, mutta tällöin tiedosto, jota kirjautumiseen käytetään sisältää käyttäjän salasanana selkokielisenä. Oletuksena MySQL-järjestelmä käyttää krypattuja yhteyksiä, mutta mikäli krypatattu yhteys epäonnistuu, palaa MySQL käyttämään kryptaamatonta yhteyttä. MySQL voidaan konfiguroida käyttämään SSL-yhteyksiä käyttäen hyväksi X509:ä, jolloin voidaan varmistua siitä, ettei yhteys lähetä tietoja kryptaamattomana. Kaikki tiedostot, joita yhteyden luomiseksi käytetään, tulevat MySQL-asennuksen ohessa, joten mitään ylimääräistä ei tarvitse tätä varten ladata. [46]

MySQL Workbench -ohjelma tukee suoraan csv- ja JSON-tiedostojen tuontia tietokantaan. Tätä ei kuitenkaan ole mahdollista automatisoida Event Schedulerin kautta, joten tällöin csv-tiedostojen tuonti tietokantaan olisi edelleen käsityötä. Mikäli automaatio halutaan toteuttaa MySQL-järjestelmällä, voidaan korvaavana vaihtoehtona kokeilla SQL:n LOAD-komentoa. LOAD-komento tukee vastaavasti tiedon tuomista csv-tiedostoista (tai muista formaateista) ja koska kyseessä on SQL-kysely, se on mahdollista tallentaa Workbench-ohjelman avulla ja tämän jälkeen automatisoida Event Schedulerin kautta. [51], [52]

Dokumentaationsa puolesta MySQL jatkaa samalla linjalla kuin aiemmatkin vertailukohteensa. Sivusto on selkeästi toteutettu ja sieltä löytää helposti etsimänsä tiedon, joko suoraan parametrien nimellä hakemalla tai sitten normaaleilla hakutavoilla. Niin ikään versiontarkistus on mahdollista, jos on tarvetta tutkia löytyykö ominaisuus tai työkalu siitä versiosta joka on asennettu. MySQL saa varmasti avoimen lähdekoodinsa ansiosta pientä etua siitä, että se on varmasti tämän johdosta levinnyt vuosikymmenien aikana lukemattomiin eri käyttökohteisiin. Näin ollen sen kohdalle on todennäköisemmin osunut monenlaisia ongelmatilanteita. Mikäli projektin edetessä siis ongelmia jonkun osa-alueen kanssa, on hyvin todennäköistä, että jollain toisella on ollut jo vastaavanlainen ongelma jossain kohtaa ja vastaus ongelmaan löytyy joltain internetin useilta ongelmien ratkaisuun erikoistuneelta sivustolta (esimerkiksi Stackoverflow).

5. ARVIOINTIKEHYS

Tarkastelluista vaihtoehdoista kaikki kolme ovat sellaisia, joilla projekti olisi mahdollista toteuttaa, mutta pieniä eroja järjestelmien välillä saadaan aikaan kun tarkastellaan niitä kriteereitä mitä edellisen kappaleen alussa annettiin. Nämä kriteerit voidaan jakaa kuuteen ryhmään: hinta, levinneisyys, käyttöjärjestelmät ja kielet, apuohjelmat, turvallisuus sekä datan vienti järjestelmään.

5.1 Kriteerien valinta

Kriteerien valinnassa yrityksen tarpeet ja rajoitukset määrittävät sitä, miksi juuri nämä kriteerit valittiin määrittämään tietokantajärjestelmän valintaa. Yrityksen IT-henkilöstömäärä on varsin pieni (1-2 henkilöä), joten tämä vaikuttaa siihen, että levinneisyys ja apuohjelmat sekä datan vienti päätettiin ottaa kriteereiksi. Pienen henkilöstämäärän kanssa on mahdollista, ettei henkilöstöllä itsellään ole esimerkiksi ratkaisua mahdollisiin ongelmatilanteisiin tai siihen miten tietty ominaisuus pitäisi toteuttaa. Tällöin tilannetta helpottaa järjestelmän levinneisyys, yleensä korreloi saatavilla olevan tuen määrässä.

Apuohjelmat ovat kriteerinä myös samasta syystä mukana. Koska järjestelmän tuntevaa henkilöstä ei välttämättä ole aina paikalla, mutta joitain yksinkertaisia tehtäviä voi tietokantaan silti joutua tekemään, on hyvä jos tietokantajärjestelmä tarjoaa apuohjelmia jolla tällaiset yksinkertaiset operaatiot voisi olla mahdollista toteuttaa helposti ilman alan koulutusta.

Datan vienti otettiin taas kriteeriksi mukaan, koska haluttiin selvittää millaisia mahdollisuuksia datan automaattiseen tuontiin on tutkittavissa tietokantajärjestelmissä. Edelleen tämä liittyy henkilöstömäärään, sillä datan vienti olisi hyvä saada automatisoitua, jotta siihen ei tarvitse erikseen käyttää työaikaa joka kuukausi. Mitä paremmat automatisointimahdollisuudet ovat olemassa, sitä helpommalla päästään tarvittavan työn määrässä ja resurssit voidaan kohdistaa muualle.

Kriteerinä hinta taas tulee mukaan hyvinkin selvästi yrityksen budjetin kautta. Jo ennen kuin järjestelmiä oli valittu tutkittavaksi tai koko projektia aloitettu, oli selvää, ettei yrityksen ole tarkoitus sijoittaa uuteen tietokantajärjestelmään suuria summia rahaa. Näin ollen hinta valikoitui hyvinkin selvänä kriteerinä mukaan jo alusta asti.

Käyttöjärjestelmät ja kielet otettiin mukaan nimenomaan siinä tarkoituksessa, että laaja määrä laskettaisiin tietokantajärjestelmän eduksi. Tämä perustuu siihen, ettei yritystä haluttu väkisin lukita johonkin tiettyyn ympäristöön, eikä yritykselle itselleen ollut vaatimusta, että uuden tietokantajärjestelmän tulisi toimia juuri tietyssä ympäristössä ja tietyllä käyttöjärjestelmällä.

Yritys on painottanut toimintatavoissaan tietoturvallisuutta ja alusta alkaen oli selvää, että uudenkin tietokantajärjestelmän haluttiin olevan mahdollisimman turvallinen. Tietoturvallisuus itsessäänkin on nykyaikana hyvin tiedostettu vaatimus ihmisten keskuudessa ja erilaiset tietomurrot aiheuttavat nopeasti haittaa yrityksille. Turvallisuus tulikin hinnan ohella toisena heti alusta alkaen selvänä kriteerinä mukaan projektin toteutukseen.

5.2 Kriteerien tarkastelu

Hinta on sinällään hankala kriteeri arviointikehyksessä, ettei tarkkaa rajasummaa sille ole, kuinka paljon järjestelmä saa maksaa. Hintaa yritetäänkin arvioida sen suhteen, minkä hintaisia palveluita yritys on tähän asti hankkinut.

Tietokantajärjestelmien levinneisyyttä ei pystytä tietysti mittaamaan pilkuntarkasti, eikä tätä kriteeriä ole tarkoitus painottaa suuresti, mutta järjestelmien erot ovat kuitenkin suhteellisen pieniä, joten tällainenkin näkökulma on hyvä ottaa valinnassa huomioon. Levinneisyydessä eri vaihtoehtojen välillä tarkastellaan välillisesti myös saatavilla olevan tuen määrää.

Käyttöjärjestelmät ja kielet otetaan huomioon yrityksen palvelintilanteen sekä uudelleentehtävien skriptien kannalta. Uusi palvelin on jossain kohtaa hankittava ja mitä enemmän siihen on tietokantajärjestelmän puolesta vaihtoehtoja, sen parempi. Vanhassa järjestelmässä olevat skriptit on tehtävä uudelleen ja tätä varten on parempi mitä enemmän tuettuja kieliä on.

Apuohjelmat pitää sisällään erilaiset graafiset käyttöliittymät joita tietokantajärjestelmän kehittäjä tarjoaa järjestelmän tueksi. Järjestelmiä voi käyttää komentokehotteen kauttakin, mutta on mahdollista, että yrityksen henkilökunnasta voi joskus joutua sellainenkin henkilö tekemään tietokantaan toimenpiteen (esimerkiksi tietokannan uudelleenkäynnistyksen), joka ei tietokantajärjestelmien kanssa ole juuri ennen toiminut. Tällöin on avuksi, mikäli järjestelmään löytyy graafisia työkaluja, sillä niiden avulla on helpompi kouluttaa erilaisia toimintoja kuin komentokehotteen kautta.

Turvallisuus on arviointikehyksessä yksi painotetuimmista kriteereistä valinnan suhteen. Tässä kriteeri on kuitenkin se, mitä vaihtoehtoja turvallisuuteen järjestelmässä

on, eikä se kuinka turvallinen järjestelmä on oletusasetuksineen heti sen jälkeen kun se on asennettu.

Datan vienti liittyy kassaohjelmiston tietokannasta otettujen tiedostojen tuontiin uuteen tietokantaan. Tiedostot saadaan csv-muodossa ja tuonnin automatisointi on tavoitteena joten sen toteutuksen käytännöllisyys on arvioinnissa olennaista. Annetuista kriteereistä turvallisuus, datan vienti, apuohjelmat ja hinta ovat eniten painotettuja ja niitä voidaan pitää ensisijaisina kriteereinä. Levinneisyys sekä käyttöjärjestelmä ja kielet ovat vähemmän painotettuja ja näin toissijaisia kriteereitä.

5.3 Ensisijaiset kriteerit

Hintaa tarkastellessa voisi äkkiä ajatella, että MySQL on ilmeinen valinta. Sekä Db2:sta että SQL Serveristä löytyvät kuitenkin ilmaiset vaihtoehdot, joita voi vähintään kokeilla miten tietokantajärjestelmä tuntuisi toimivan, ennen kuin tekee varsinaista päätöstä. Db2:n ilmainen Community-versio voisi mahdollisesti jopa riittää sellaisenaan projektin tarpeisiin, sillä 100 Gt:n rajattu kanta riittäisi kyllä pitkälle tulevaisuuteen. Kysymysmerkiksi jää laitteistorajausten riittävyys. Kerran kuussa ajettavat SQL-kyselyt voivat olla melko raskaita, joten Community-version rajattu teho saattaa johtaa siihen, että kyselyiden ajaminen vie rutkasti aikaa. SQL Serverin ilmaisversio riittäisi sellaisenaan projektin tarpeisiin, mutta tietokannan koon rajaus 10 Gt:n kokoiseksi aiheuttaa selvästi enemmän pohdintaa tulevaisuuden kannalta. Vaikka tällä hetkellä tietokanta menisi helposti rajojen sisälle, tulisi raja silti selvästi nopeammin vastaan kuin Db2:ssa ja silloin viimeistään pitäisi hankkia kaupallinen versio SQL Serveristä tai käynnistää koko projekti taas kerran uudelleen.

Voitaneen siis todeta, että mikäli valittaisiin Db2 tai SQL Server, otettaisiin jokin niiden maksullisista versioista. Projektin kannalta se mitä todennäköisimmin tarkoittaisi edullisimpia mahdollisia versioita (Db2 minimissään 1850 $ ja SQL Server 931 $ sekä lisenssi). Db2:n Standard-versiossa on siinäkin hieman laiterajoituksia, kun taas SQL Serverissä ei näitä ole. Pelkästään hinnan puolesta mietittynä voidaan todeta, että paras vaihtoehto olisi MySQL ja sen jälkeen SQL Server hieman Db2:sta halvemmalla hinnallaan ja rajoittamattomuudellaan. MySQL:stä on tietysti myös kaupalliset versiot ja niistä varsinkin Enterprise-versio tuo lisää turvallisuusvaihtoehtoja, mutta pelkästään tämän otsikon alla mietittynä MySQL vie kärkipaikan. SQL Serverin ja Db2:n kanssa voitaneen olettaa, että vaikka ensin aloitettaisiin ilmaisversiolla, joudutaan jossain kohtaa kuitenkin siirtymään kaupalliseen versioon. Tähän pohjaten SQL Server vie hieman halvempana kakkospaikan ja Db2 tulee kolmantena vaihtoehtona.

Microsoft tarjoaa suurimman määrän erilaisia apuohjelmia aina tietokannan ylläpidosta varmuuskopiointiin. Samalla kuitenkin se tärkein eli SSMS on saatavilla vain Windows-alustoille. Mikäli uusi palvelin siis asennetaan käyttäen jotain muuta käyttöjärjestelmää, ei tätä apuohjelmaa päästä käyttämään. Tämä ei tietysti estä apuohjelman mahdollistamien toimenpiteiden suorittamista, mutta joissain tilanteissa graafinen käyttöliittymä voi olla parempi ratkaisu. Toisaalta asian voi kääntää toisinkin päin ja nimenomaan Windows-alustalle asennettuna SQL Server on hyvinkin vahva vaihtoehto varsinkin, kun tarjotuilla työkaluilla on mahdollista automatisoida toimenpiteitä, jotka on tähän asti jouduttu tekemään käsityönä.

IBM tarjoaa niin ikään hyvän kattauksen apuohjelmia ja tietokannan hallinnan puolesta esimerkiksi Data Studio ei häviä SQL Serverin tai MySQL:n vastaaville. IBM:n apuohjelmilla ei kuitenkaan pystytä suoraan hoitamaan eri tehtävien automatisointia. Automatisointi on myös Db2:lla varmasti toteutettavissa (esimerkiksi crontabin avulla), mutta jos muiden järjestelmien ohjelmilla se voidaan hoitaa ilman välikäsiä, on se laskettava niiden eduksi.

MySQL-järjestelmään on tarjolla vain yksi virallinen apuohjelma (ilmaisversioon), mutta kyseessä on eräänlainen all-in-one-ratkaisu. Workbench-ohjelmalla onnistuvat samat tietokannan hallintaoperaatiot kuin muiltakin ja vaikka sekään ei suoraan tue tehtävien automatisointia, on MySQL:llä tarjolla Event Scheduler. Lisäksi MySQL oli ainoa vaihtoehto, jossa oli data migration -ominaisuus, vanhojen tietokantojen tuomiseksi uuteen järjestelmään. Toisaalta apuohjelmien kannalta MySQL ottaa hieman takapakkia siinä, että turvallisuuteen ja auditointiin liittyvät apuohjelmat on ”lukittu” maksullisen Enterprise-version taakse, joka oli vieläpä MySQL:n tietokantaversioista toiseksi kallein.

Apuohjelmissa voidaan SQL Server todeta parhaaksi vaihtoehdoksi sen tarjoamien useiden vaihtoehtojen ja automatisaatiomahdollisuuksien puolesta. MySQL ottaa seuraavan sijan lähinnä sen vuoksi, ettei ilmaisversiossa ole tarjolla kaikkia vaihtoehtoja. Toki MySQL:n kannalta myös kolmannet osapuolet tarjoavat erilaisia vaihtoehtoja, mutta tämän tutkielman osalta perehdyttiin vain viralliseen tarjoamaan. Db2: jää kolmannelle sijalle automatisaation suoran tuen puuttumisen vuoksi.

Turvallisuusominaisuuksiltaan SQL Server ja Db2 erottuvat edukseen. Molemmissa on hyvät mahdollisuudet niin käyttäjänhallinnan kuin todennuksenkin suhteen. Varsinkin Db2:n tapa hoitaa käyttäjien oikeuksia niin, että mikäli käyttäjällä ei oikeutta ole annettu, on se automaattisesti kielletty, voi joissain tilanteissa pelastaa pitkältä

kiellettyjen oikeuksien suhteen. SQL Server taas toimi perinteisemmällä tavalla, jossa oikeuksia voidaan joko myöntää tai kieltää erikseen.

Todennuksessa sekä SQL Server, että Db2 tukevat suoraan AD-ympäristöä, mikä on edelleen yrityksen palvelintilanteen kannalta hyvä, sillä tämä mahdollistaa sen, että niin halutessa voidaan uusi palvelin asentaa tutulla käyttöjärjestelmällä ja saada AD-ympäristön tuki todennuksessa käyttöön. SQL Server tarjoaa Microsoftin tuotteena vielä lisämahdollisuuksia AD-ympäristössä käyttäjätilien lukitsemismahdollisuudellaan. Ilman AD-ympäristöäkin SQL Serverissä todennuksessa käytetään automaattisesti Kerberos-suojausprotokollaa. Db2:ssa Kerberos-protokolla on niin ikään mahdollinen, mutta se pitää asentaa lisäosan kautta. MySQL:n puolella sitä taas ei saada käyttöön ilman Enterprise-version hankintaa ja lisäksi se pitää erikseen asentaa laajennuksen kautta.

Datan kryptaykseen kaikilla järjestelmillä on hyvät vaihtoehdot ja varsinkin SQL Serverin tarjoama always encrypted -vaihtoehto riittää salaamaan arkaluontoisemmankin tiedon. MySQL tosin kärsii taas hieman siitä, että luotettavimmat salausalgoritmit on taas laitettu maksullisen Enterprise-version taakse. Sama koskee MySQL-järjestelmää auditoinninkin osalta, vaikka muun muassa triggereitä on mahdollista luoda ilmaisversiossakin, on isommat turvallisuus- ja auditointiratkaisut laitettu Enterprise-version alle. SQL Server ja Db2 taas tarjoavat monipuoliset auditointimahdollisuudet tietokannan turvallisuuden tukemiseksi ja varsinkin SQL Serverin SSMS-ohjelma tarjoaa useita erilaisia mahdollisuuksia tähän.

Näiden jälkeen voidaan todeta, että turvallisuuden puolesta SQL Server olisi paras vaihtoehto, varsinkin Windows-alustalle asennettuna. Se tarjoaa hyvät vaihtoehdot tietokannan turvaamiseksi ja turvallisuuden valvomiseksi. Db2 tulee hyvänä kakkosena, mutta koska se vaatii muutaman lisäosan asentamisen saman toiminnallisuuden saamiseksi ja koska SQL Server tarjoaa AD-ympäristössä lisävaihtoehtoja, jää Db2 SQL Serverin taakse. MySQL on tässä kohtaa kolmantena ja taas kerran ilmaisversion rajoitteiden johdosta.

Datan viennissä kyse on enemmän siitä miten uudet järjestelmät toimivat yhteen vanhojen menetelmien kanssa ja miten ne tukevat automatisointia. On selvää, että vaikka uuteen järjestelmään ei olisi voitu tuoda dataa samalla tavalla kuin vanhaan, olisi se siihen jollain muulla tavalla tuotu. Mikäli on kuitenkin mahdollista käyttää jo kehitettyä mallia, säästää se taas aikaa projektin kehityksessä.

Kaikki tutkitut järjestelmät pystyvät suoraan lukemaan csv-tiedostoja ja tuomaan ne sitä kautta kantaan, joten projektin kannalta tiedon ottamista kassaohjelmiston

tietokannasta ei tarvitse miettiä uudelleen. Uuden järjestelmän täydellinen automaatio on ollut yksi projektin tavoitteista ja tätä se tarkoittaa tiedon tuomisenkin osalta. Vaikka järjestelmien hallintaohjelmat tukevat SQL-kyselyiden tallennusta ja ajoa (jopa massana) niin silti ei yhdelläkään järjestelmällä voi niiden avulla suoraan automatisoida kyselyjä. SQL Serverin ja MySQL:n tapauksessa ratkaisu ei kuitenkaan ollut kaukana, sillä ensimmäisessä on SQL Server Agent -työkalu ja jälkimmäisessä Event Scheduler -työkalu, joiden avulla voidaan automatisoida muun muassa kyselyiden ajo. Koska molemmissa järjestelmissä tiedon tuonti onnistuu suoraan SQL-kyselyllä, voidaan nuo kyselyt luoda ennakkoon ja sen jälkeen ajastaa työkalujen avulla. Db2 saa puolestaan hieman huonomman arvosanan tässä kategoriassa. Automatisointi saadaan varmasti Db2-järjestelmässäkin onnistumaan, mutta sen hallintaohjelma ei sitä suoraan tue, eikä toisia järjestelmiä vastaavaa työkalua ole. Db2:n tapauksessa joudutaan siis turvautumaan joko käyttöjärjestelmästä löytyviin työkaluihin kuten Linuxin crontabiin, mikä tietysti vaatii uuden palvelimen asentamista Linux-alustalle. Toinen vaihtoehto on etsiä jokin kolmannen osapuolen ohjelma, jolla saadaan vastaavat toimenpiteet suoritettua. SQL Server ja MySQL ovat siis melko tasaväkisiä tässä kriteerissä. SQL Serverin tiedon tuomiseen vaadittava bulk insert -syntaksi on kuitenkin hieman MySQL:n load-syntaksia yksinkertaisempi, joten valitaan sen perusteella SQL Server tässä kriteerissä parhaaksi vaihtoehdoksi. MySQL toisena vaihtoehtona ja Db2 kolmantena.

5.4 Toissijaiset kriteerit

Kuten aikaisemmin todettiin, on levinneisyyttä hankala mitata tarkasti. Tätä kriteeriä voikin käyttää enemmän tasapelin ratkaisuun ellei muuten valintaa järjestelmien välillä saada aikaa. Kaikki kolme tarkasteltua vaihtoehtoa ovat joka tapauksessa suosituimpien relaatiotietokantajärjestelmien joukossa, joten kovin suurta eroa ei tässäkään tapauksessa saada eri järjestelmien välille. DB-Engines sivun listaus on usein referoituna muuallakin ja heidän käyttämänsä tekniikat eri järjestelmien käyttömäärien selvittämiseksi antanevat tarpeeksi tarkan kokonaiskuvan. Käytetään siis DB-Enginesin listausta tämän kriteerin ratkaisemiseksi. Suosituimman järjestelmän paikan ottaa MySQL, kakkosena SQL Server ja kolmantena Db2.

Kaikkien kolmen järjestelmän tukisivustot olivat järjestelmiä tutkittaessa suureksi avuksi. Levinneisyys lienee vaikuttanut myös virallisten tukisivustojen ulkopuolisen dokumentaation määrään ja varsinkin MySQL:n tapauksessa apua varmasti löytyy jos ei viralliselta sivustolta, niin joltain useista IT-ongelmiin erikoistuvista sivustoista. SQL Server oli niin ikään hyvin edustettuna, kun apua piti etsiä virallisen sivuston

ulkopuolelta, eikä Microsoftin omakaan tukisivusto ole missään nimessä huonoimmasta päästä. Db2 tuskin näille suurissa määrin häviää, mutta järjestelmiä tutkiessa omien kokemusten perusteella Db2:n kohdalla apu löytyi useammin viralliselta tukisivustolta kuin sen ulkopuolelta. Tämä voi kertoa siitäkin, että IBM on toteuttanut tukisivustonsa äärimmäisen hyvin, mutta mahdollista on myös, että Db2 ei hintansa puolesta ole aivan niin käytetty kuin muut vaihtoehdot. Tuen määrässä voitaneen näin ollen todeta MySQL:n ja SQL Serverin olevan suurin piirtein tasaväkiset ja Db2 tulee hieman näiden perässä.

Käyttöjärjestelmissä Db2 tukee suurinta määrää eri vaihtoehtoja (kuusi kappaletta), mutta samalla käyttöjärjestelmän valinta vaikutti hieman siihen millainen asennus on mahdollista tehdä. Linux- ja Unix-ympäristöissä jokainen instanssi on luotu dedikoidun käyttäjän alle, joka on instanssin omistaja. Windows-ympäristössä on taas mahdollista luoda useampi instanssi yhden käyttäjän alle. Ainakaan alkuun ei ole tarkoitus luoda yhtä instanssia enempää, joten tämäkään rajoitus ei suoraan estä Db2:n valintaa. MySQL tukee liki yhtä monta käyttöjärjestelmää (viisi kappaletta), eikä siinä ole Db2:n kaltaisia rajoituksia asennusalustan suhteen. SQL Server tukee vain Windows- ja Linux-alustoja, mutta tämäkin pitää sisällään tietysti monta eri versiota. Ohjelmointikielien puolesta kaikki vaihtoehdot tukevat kieliä jotka ovat käytetyimpien joukossa. Kaikkien järjestelmien listoista löytyy kieliä jotka ovat sen verran tuttuja, ettei projektin aikana tarvitse kokonaan uutta kieltä ruveta opettelemaan. Sen pohjalta vaihtoehdot voidaan tässä tapauksessa laittaa järjestykseen puhtaasti vaihtoehtojen määrän perusteella. MySQL ja Db2 olisivat periaatteessa tasatilanteessa (Db2 tukee useampaa käyttöjärjestelmää ja MySQL useampaa ohjelmointikieltä), mutta koska Db2 on asennusalustansa suhteen hieman MySQL:ää rajoittavampi, voidaan MySQL nostaa tämänkin kriteerin ykkössijalle. Db2 toisena ja SQL Server kolmantena

5.5 Lopullinen valinta

SQL Server ja MySQL veivät siis keskenään kolme kärkisijoitusta kuuden kriteerin kesken. SQL Server on paras vaihtoehto kun tarkastellaan apuohjelmia, turvallisuutta ja datan vientiä. MySQL nappaa taas ensimmäisen sijan kun kriteereinä ovat levinneisyys, hinta sekä käyttöjärjestelmä ja kielet. SQL Server tosin nappaa omista kärkisijoistaan kaikki ensisijaisista kriteereistä. MySQL on ensisijaisista kriteereistä ykkösvalintana vain kun kriteerinä on hinta. Kriteereissä oli järjestelmien välillä kuitenkin hieman vaihtelevuutta muun muassa sen suhteen minkä käyttöjärjestelmän alla tietokantajärjestelmä toimii. Muutamassa kriteerissä ykkös- ja kakkosvaihtoehdot olivat niin ikään hyvin lähellä toisiaan ja valinta oli melko kosmeettinen. Varsinaista

valintaa voidaan helpottaa laittamalla järjestelmät kriteereineen taulukkoon ja arvostella ne kriteereittäin asteikolla huono - välttävä - kohtalainen - hyvä - erinomainen. Näin saadaan hieman lisää tukea valintaa varten. Asteikon arvosanat määräytyvät seuraavanlaisesti.

- Huono: Järjestelmä ei täytä yrityksen tarpeita kriteerin määrittelemällä tavalla.
- Välttävä: Järjestelmä voi pieniltä osin täyttää yrityksen tarpeet kriteerin määrittelemällä tavalla.
- Kohtalainen: Järjestelmässä voi olla muutamia puutteita, mutta muutoin se täyttää yrityksen tarpeet kriteerin määrittelemällä tavalla.
- Hyvä: Järjestelmässä voi olla pieniä puutteita, mutta muutoin se täyttää yrityksen tarpeet kriteerin määrittelemällä tavalla.
- Erinomainen: Järjestelmä täyttää yrityksen tarpeet kriteerin määrittelemällä tavalla.

Järjestelmä	**Hinta**	**Levinneisyys**	**K&K**	**Apuohjelmat**	**Turvallisuus**	**Datan vienti**
Db2	Kohtalainen	Hyvä	Erinomainen	Kohtalainen	Hyvä	Hyvä
SQL Server	Hyvä	Erinomainen	Hyvä	Erinomainen	Erinomainen	Erinomainen
MySQL	Erinomainen	Erinomainen	Erinomainen	Hyvä	Kohtalainen	Erinomainen

Taulukko 1 *Tietokantajärjestelmien arviointi.*

Tällä tavalla arvioituna huomataan taulukon 1 avulla, että Db2 saa selvästi huonoimman arvioinnin, vaikka samalla pitää huomata, että varsinaiset erot järjestelmien välillä ovat kuitenkin suhteellisen pieniä. SQL Server ja MySQL vaikuttavat tälläkin tavalla tutkittuna tasaisen vahvoilta. Erot tulevat hinnassa, käyttöjärjestelmässä ja kielissä, apuohjelmissa sekä turvallisuudessa. Molemmilla järjestelmillä on kaksi kohtaa jolla ne eivät saa erinomaista arvosanaa, mutta SQL Serverin kohdalla huonommatkin arvosanat ovat vähintään arvosanalla hyvä. MySQL saa turvallisuuden kohdalla arvosanaksi taas kohtalaisen, johtuen osaltaan ilmaisversion rajoituksista. Turvallisuus on kuitenkin ensisijainen kriteeri ja toisista ensisijaisista kriteereistä (apuohjelmat ja hinta) johtuvat erot menevät ”tasan” SQL Serverin ja MySQL:n välillä. Tälläkin tavalla tarkasteltuna voidaan siis vetää lopulliseksi tulokseksi, että näillä kriteereillä tarkasteltuna projektin kannalta järkevin vaihtoehto uudeksi järjestelmäksi olisi Microsoft SQL Server.

Seuraavana askeleena projektin osalta on uuden järjestelmän hankinta ja sen jälkeen asennuksen valmistelu. Vanhojen skriptien valmistelu uuteen järjestelmään on hyvä aloittaa myös samalla. Seuraavat askeleet eivät valitettavasti kuitenkaan ehtineet tämän tutkielman aikamääreisiin joten tutkielman osalta jäädään tilanteeseen jossa uusi järjestelmä on valittuna.

6. POHDINTA

Tavoitteena oli valita kohdeyritykselle uusi tietokantajärjestelmä, jolla voitaisiin korvata vanha. Valintaa helpottamaan luotiin myös arviointikehys, mikä osoittautuikin hyväksi apukeinoksi tietokantajärjestelmien erojen luokittelemisessa. Arviointikehys oli tässä työssä sanallinen, mutta sen olisi voinut toteuttaa tietysti myös numeroihin perustuvana kehyksenä. Tällöin kehykselle olisi voitu luoda vaikkapa jonkinlainen painotuskerroin ensisijaisille kriteereille (esimerkiksi 1,0 ensisijaisille ja 0,8 toissijaisille). Kehys olisi voinut olla vaikka välillä 0,0 – 5,0 ja arvosana olisi kerrottu joko ensisijaisen tai toissijaisen kriteerin kertoimella todellisen arvosanan saamiseksi. Tällä tavalla kertoimien avulla tietokantajärjestelmien vertaileminen olisi saattanut olla hieman helpompaa tai ainakin erot olisivat olleet selvästi nähtävillä, kun olisi mahdollista verrata suoraan yhtä arvosanaa sellaisenaan jokaiselle tietokantajärjestelmälle, eikä tarvitse miettiä jokaisen kriteerin kohdalla sanallisen arvioinnin merkitystä.

Tällainenkin arviointikehys olisi pitänyt kiinnittää johonkin logiikkaan, minkä laatiminen olisi todennäköisesti vaatinut selvästi sanallista arviointia enemmän työtunteja. Lisäksi olisi ollut vähintäänkin hankala sanoa, olisiko tällaisella kehyksellä saatu selvästi parempi ero sanalliseen arviointiin verrattuna. Joka tapauksessa, numeroarvosanaan perustuvaa arviointia voi vähintään harkita, jos kohdeyritykselle tulee samanlainen vaihtotarve toistamiseen tai mikäli vastaavassa työssä pitäisi vertailla selvästi useampaa eri tietokantajärjestelmää.

Sanallinen arviointikehys oli mielestäni projektin tavoitteeseen peilattuna kuitenkin riittävä. Arviointikehyksen avulla saatiin lopputuloksena kohdeyritykselle ehdotus uudesta tietokantajärjestelmästä, joten siltä osin projektin tavoite onnistui odotusten mukaisesti.

Projektin kannalta olisi tietysti ollut aina parempi, mitä useampaan erilaiseen tietokantajärjestelmään olisi tutustuttu. Yhteen tietokantajärjestelmään perehtyminen vie kuitenkin huomattavan paljon aikaa, joten se ei valitettavasti työn aikataulun suhteen ollut mahdollista. Lisäksi olisi ollut ainakin mielenkiintoista perehtyä myös muihin kuin relaatiomallisiin tietokantoihin ja siihen miten niitä olisi voitu mahdollisesti hyödyntää projektin kannalta. Tässäkin tosin aikarajoitteet tulivat siinä kohtaa vastaan, kun todettiin, ettei relaatiomallisen tietokannan valintaa vastaan ollut juurikaan esteitä.

7. YHTEENVETO

Tämän työn tarkoituksena oli saattaa alulle kohdeyrityksen projekti tietokantajärjestelmän vaihdosta valitsemalla tietokantajärjestelmä, joka tulisi korvaamaan vanhan. Järjestelmän vaihto koostui tietokantajärjestelmälle asetettujen valintakriteerien laatimisesta, mahdollisten korvaavien tietokantajärjestelmien valitsemisesta, näihin järjestelmiin perehtymisestä ja lopulta valinnasta, siltä osin, mikä järjestelmä olisi sopivin vaihtoehto kohdeyrityksen tarkoituksiin.

Tietokantajärjestelmä on mahdollista toteuttaa monella eri tavalla, joten ensimmäiseksi tehtiin valinta siitä tulisiko uusi valinta toteuttaa samalla relaatiotietokantamallilla kuin aiempikin järjestelmä, vai olisiko jokin muu tietokantamalli (esimerkiksi NoSQL) mahdollisesti parempi vaihtoehto. Asiaa tutkittua kävi kuitenkin selväksi, että vaikka relaatiotietokanta on jo hyvinkin vanha malli, on se silti edelleen hyvin tuettu ja suuressa käytössä eri puolilla maailmaa, eivätkä muut tietokantamallit ole sitä vielä onnistuneet täysin syrjäyttämään. Kohdeyrityksen käyttämillä tietomäärillä muista tietokantamalleista tuskin saataisiin kovin suurta hyötyä. Lisäksi vanhasta järjestelmästä olisi todennäköisimmin hyvin paljon yksinkertaisempaa tuoda tiedot samanlaiseen tietokantajärjestelmään. Näin ollen voitiin tehdä valinta relaatiotietokantamallin puolesta ja jättää muut mallit valintaprosessin ulkopuolelle.

Tietokantajärjestelmän valintaa tarkasteltiin käyttömääriltään myös avoimen ja suljetun lähdekoodin järjestelmien välillä. Mikäli erot olisivat suuria, olisi valinta voitu kohdistaa joko pelkästään avoimen tai suljetun lähdekoodin ratkaisuihin. Näiden välillä ei kuitenkaan huomattu minkäänlaista suurta eroa, joten valinnan kannalta ei ollut väliä olisiko uusi järjestelmä avoimeen vai suljettuun lähdekoodiin perustuva.

Tietokantajärjestelmän valintaa varten painotettiin valinnassa tiettyjä kriteereitä. Näistä kriteereistä turvallisuus, datan vienti, apuohjelmat ja hinta arvioitiin enemmän painotettaviksi ensisijaisiksi kriteereiksi. Levinneisyys sekä käyttöjärjestelmä ja kielet arvioitiin taas vähemmän painotettuja ja näin ollen ne olivat toissijaisia kriteereitä.

Valitut kriteerit osoittautuivat toimiviksi, vaikka toissijaiset kriteerit olivatkin sellaisia, joilla ei normaalisti järjestelmien eroja välttämättä haettaisi. Kriteereillä saatiin kuitenkin luotua muuten keskenään hyvin tasoissa olevien järjestelmien välille tarvittavia, päätöksentekoa tukevia, eroja.

Tietokantajärjestelmän ja sen valintaa koskevien kriteereiden kanssa on kuitenkin hyvä huomata, että nämä ovat varsin sidottuja yrityksen tarpeisiin. Tästä johtuen työssä esitetyt kriteerit eivät sovi kaikkiin tapauksiin ja tietokantajärjestelmää valittaessa kohteena olevalla yrityksellä on oma vaikutuksensa valittavan järjestelmän kriteereihin. Toisella yrityksellä voisi olla käytössään isompi budjetti tai yritys voisi olla lukittu tiettyyn ympäristöön jolloin käytetyt valintakriteerit muuttuisivat sen mukaan.

Tarkempaan tutkintaa varten valitut tietokantajärjestelmät (IBM Db2, Microsoft SQL Server ja MySQL) valittiin niin, että mukana olisi sekä avoimen, että suljetun lähdekoodin vaihtoehtoja. Lisäksi järjestelmät valittiin niin, että ne olisivat suhteellisen yleisessä käytössä, jolloin niille olisi saatavilla hyvä tukiverkosto mahdollisesti pitkälle tulevaisuuteen.

Tutkittavien tietokantajärjestelmien valinnan jälkeen käytiin yksitellen läpi jokainen tarkasteltava tietokantajärjestelmä ja käsiteltiin niiden toteutusta ja ominaisuuksia valittujen kriteerien osalta. Eri vaihtoehtoja tutkittaessa kävi nopeasti ilmi, että järjestelmät olisivat kaikki sellaisia joilla projekti olisi mahdollista toteuttaa. Erojakin tietysti vaihtoehtojen välillä oli, mutta ne olivat lopulta suhteellisen pieniä.

Db2-tietokantajärjestelmä sai valintakriteerien osalta kolmikon huonoimman arvosanan, mikä johtui lähinnä Db2:n korkeasta hankintahinnasta sekä huonoimmasta apuohjelmavalikoimasta. Käyttöjärjestelmää ja kieliä lukuun ottamatta Db2:sta ei myöskään arvioitu erinomaisella arvosanalla missään kriteerissä. Arviointia tehdessä huomioitiin kuitenkin, että vaikka Db2 arvioitiin kolmikon huonoimmaksi, ei se tietokantajärjestelmien pienistä eroista johtuen tarkoittanut, ettei järjestelmää olisi mahdollista Db2-tietokannalla toteuttaa.

MySQL ja SQL Server osoittautuivat taas varsin tasaväkisiksi vaihtoehdoiksi. Sekä MySQL että SQL Server arvioitiin erinomainen-arvosanalla neljässä eri kriteerissä, sai SQL Server kuitenkin nämä arvosanat pääasiassa ensisijaisissa kriteereissä, kun MySQL taas pärjäsi paremmin toissijaisissa. Ainoa ensisijainen kriteeri jossa MySQL pärjäsi SQL Serveriä paremmin, oli hinta. SQL Serverin hinta ei kuitenkaan ollut liian suuri kohdeyrityksen budjetin rajoille, joten senkään puolesta SQL Serveriä ei tarvinnut hylätä vaihtoehtona.

Tietokantajärjestelmien tutkinnan jälkeen tultiin näin ollen siihen lopputulokseen, että SQL Server olisi järkevin vaihtoehto, jolla toteuttaa kohdeyrityksen tietokantajärjestelmän vaihto. Kaikista käsitellyistä vaihtoehdoista on kuitenkin saatavilla jonkinlainen ilmainen testiversio, tai MySQL:n tapauksessa koko

tietokantajärjestelmä, joten halutessaan kohdeyritys voi suorittaa pienimuotoisen testikäytön jokaisella tietokantajärjestelmällä ennen varsinaista päätöstä.

LÄHTEET

[1] A. B. Bhatia and V. Bansal, *Database Management System*, vol. 1, no. 3. 2015.
[2] R. A. Mata-Toledo, “Database Management System,” vol. 1, no. 1, pp. 1–11, 2018.
[3] T. Connolly, C. Begg, and R. Holowczak, *Business Database Systems*. 2008.
[4] S. B. Gupta and A. Mittal, *Introduction to Database Management System*, 2nd ed. 2017.
[5] Lexico, “No Title.” [Online]. Available: https://www.lexico.com/en/definition/data_warehouse. [Accessed: 23-Sep-2019].
[6] F. Hayes, “The Story So Far,” *Computerworld*, 2002.
[7] W. H. Inmon, *Building the Data Warehouse*, 4th ed. 2005.
[8] OLAP.com, “No Title.” [Online]. Available: https://web.archive.org/web/20190617155613/https://olap.com/learn-bi-olap/olap-bi-definitions/business-intelligence/. [Accessed: 22-Sep-2019].
[9] M. Kimball, Ralph; Ross, *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, 3rd editio. John Wiley & Sons, Incorporated, 2013.
[10] R. Hart and A. M. H. Kuo, “Meeting health care research needs in a kimball integrated data warehouse,” *Proc. - 3rd IEEE Int. Conf. Data Sci. Adv. Anal. DSAA 2016*, pp. 697–703, 2016.
[11] A. S. Kanade and A. Gopal, “Choosing right database system: Row or column-store,” *2013 Int. Conf. Inf. Commun. Embed. Syst. ICICES 2013*, pp. 16–20, 2013.
[12] S. Samaddar, “Choosing a Database Architecture: An Essential Guide for Data Warehousing Professionals.,” *Bus. Intell. J.*, vol. 19, no. 2, pp. 16–19, 2014.
[13] I. Ong, P. Siew, and S. Wong, “A Five-Layered Business Intelligence Architecture,” *Commun. IBIMA*, 2011.
[14] “DB Engine ranking method.” [Online]. Available: https://db-engines.com/en/ranking_definition. [Accessed: 17-Oct-2019].
[15] “2019 Database Trends.” [Online]. Available: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/. [Accessed: 17-Oct-2019].
[16] IBM, “IBM Knowledge Center - Column-organized tables,” 2019. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0060592.html. [Accessed: 17-Oct-2019].
[17] “DB Engine ranking.” [Online]. Available: https://db-engines.com/en/ranking. [Accessed: 20-Oct-2019].
[18] Scalegrid, “2019 Open Source Database Report: Top Databases, Public Cloud vs. On-Premise, Polyglot Persistence.” [Online]. Available: https://scalegrid.io/blog/2019-open-source-database-report-top-databases-public-cloud-vs-on-premise-polyglot-persistence/. [Accessed: 20-Oct-2019].
[19] Oracle, “Oracle Technology Global Price List Software Investment Guide,” 2012. [Online]. Available: http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf. [Accessed: 20-Oct-2019].
[20] IBM, “Db2 Community edition.” [Online]. Available: https://www.ibm.com/cloud/blog/announcements/ibm-db2-developer-community-edition. [Accessed: 20-Oct-2019].
[21] IBM, “IBM Db2 pricing.” [Online]. Available: https://www.ibm.com/products/db2-database/pricing. [Accessed: 20-Oct-2019].
[22] “DB Engine IBM Db2.” [Online]. Available: https://db-engines.com/en/system/IBM+Db2. [Accessed: 21-Oct-2019].
[23] A. Neagu and R. Pelletier, *IBM DB2 9.7 Advanced Administration Cookbook*. Packt Publishing, 2012.
[24] G. Allen, *Beginning DB2*. 2008.
[25] IBM, “IBM Data Studio.” [Online]. Available: https://www.ibm.com/fi-en/marketplace/ibm-data-studio. [Accessed: 21-Oct-2019].
[26] IBM, “IBM Knowledge Center - IBM Data Studio.” [Online]. Available:

https://www.ibm.com/support/knowledgecenter/en/SS62YD_4.1.1/com.ibm.datatools.ds.nav.doc/topics/ds_landing.html. [Accessed: 21-Oct-2019].

[27] IBM, “IBM Knowledge Center - Security plug-ins.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.5.0/com.ibm.db2.luw.admin.sec.doc/doc/c0011970.html. [Accessed: 25-Oct-2019].

[28] IBM, “IBM Knowledge Center - Audit facility administrator tool command.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0002072.html. [Accessed: 27-Oct-2019].

[29] Microsoft, “Microsoft SQL Server pricing.” [Online]. Available: https://www.microsoft.com/en-us/sql-server/sql-server-2017-pricing. [Accessed: 28-Oct-2019].

[30] D. Petkovic, *Microsoft SQL Server 2016: A Beginner’s Guide, Sixth Edition, 6th Edition*, 6th ed. McGraw-Hill, 2016.

[31] “DB Engine Microsoft SQL Server.” [Online]. Available: https://db-engines.com/en/system/Microsoft+SQL+Server. [Accessed: 28-Oct-2019].

[32] Microsoft, “Microsoft Docs - What is SQL Server Management Studio.” [Online]. Available: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15. [Accessed: 29-Oct-2019].

[33] Microsoft, “Microsoft Docs - SQL Server Agent.” [Online]. Available: https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver15. [Accessed: 29-Oct-2019].

[34] Microsoft, “Microsoft Docs - SQL Server Profiler.” [Online]. Available: https://docs.microsoft.com/en-us/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-ver15. [Accessed: 30-Oct-2019].

[35] Microsoft, “Microsoft Docs - Choose an Authentication Mode.” [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/choose-an-authentication-mode?view=sql-server-ver15. [Accessed: 29-Oct-2019].

[36] R. Bruchez, *Microsoft SQL Server 2012 Security Cookbook*. Packt Publishing, 2012.

[37] Microsoft, “Microsoft Docs - Transparent Data Encryption.” [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver15. [Accessed: 30-Oct-2019].

[38] Microsoft, “Microsoft Docs - Always Encrypted (Database Engine).” [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-ver15. [Accessed: 30-Oct-2019].

[39] Microsoft, “Microsoft Docs - Choose an Encryption Algorithm.” [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/choose-an-encryption-algorithm?view=sql-server-ver15. [Accessed: 30-Oct-2019].

[40] A. Jorgensen, J. Segarra, P. LeBlanc, D. Cherry, A. Nelson, and J. Chinchilla, *Microsoft SQL Server 2012 Bible*, 1st ed. 2012.

[41] Microsoft, “Microsoft Docs - BULK INSERT (Transact-SQL).” [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/statements/bulk-insert-transact-sql?view=sql-server-ver15. [Accessed: 30-Oct-2019].

[42] MySQL, “MySQL Editions.” [Online]. Available: https://www.mysql.com/products/. [Accessed: 31-Oct-2019].

[43] MySQL, “MySQL Workbenc features.” [Online]. Available: https://www.mysql.com/products/workbench/features.html. [Accessed: 31-Oct-2019].

[44] MySQL, “MySQL Workbench.” [Online]. Available: https://www.mysql.com/products/workbench/. [Accessed: 31-Oct-2019].

[45] MySQL, “MySQL Event Scheduler.” [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/event-scheduler.html. [Accessed: 31-Oct-2019].

[46] K. Appigatla, *MySQL 8 Cookbook*. Packt Publishing, 2018.

[47] MySQL, “MySQL Enterprise Authentication.” [Online]. Available: https://www.mysql.com/products/enterprise/security.html. [Accessed: 31-Oct-2019].

[48] D. Khovratovich, C. Rechberger, and A. Savelieva, “Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7549 LNCS, pp. 244–263, 2012.

[49] M. Lamberger and F. Mendel, “Higher-Order Differential Attack on Reduced SHA-256,”

IACR Cryptol. ePrint Arch., vol. 37, pp. 1–16, 2011.
[50] T. Malepati, B. Shah, and E. Vanier, *Advanced MySQL 8*. Packt Publishing, 2019.
[51] MySQL, “MySQL Table Data Export and Import Wizard.” [Online]. Available: https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-table.html. [Accessed: 01-Nov-2019].
[52] MySQL, “MySQL LOAD DATA Syntax.” [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/load-data.html. [Accessed: 01-Nov-2019].