HAAGA-HELIA
ammattikorkeakoulu

Tietokannan relaatioiden mallinnusohjelman ja datan tulostusohjelman kehittäminen Digia Enterprise toiminnanohjausjärjestelmään

Antti Mikkonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2016

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

2.5.2016

Koulutusohjelma

Tekijät Mikkonen Antti	**Ryhmä** 2012
Opinnäytetyön nimi Tietokannan relaatioiden mallinnusohjelman ja datan tulostusohjelman kehittäminen Digia Enterprise toiminnanohjausjärjestelmään	**Sivu- ja liitesivumäärä** 19 + 0
Ohjaajat Outi Virkki	

Tämä opinnäytetyö toteutettiin toimeksiantona Digia Oyj:lle, joka on Suomalainen ohjelmisto- ja palveluyritys.

Opinnäytetyön tavoitteena on tuottaa tulostusohjelma Digia Enterprise toiminnanohjausjärjestelmään. Opinnäytetyössä myös suunniteltiin mihin ja miten tietokantataulujen väliset yhteydet mallinnetaan. Tulostusohjelmalla tulostetaan mallinnettujen tietokantataulujen yhteyksien mukaisesti tietoa tietokannasta.

Työssä ajateltiin myös tulevaa jatkokehitystä, jossa tarkoituksena on monipuolistaa tulostusohjelmaa niin, että sillä voidaan tulostaa tietoa myös XML-muodossa. Tämän työn pohjalta voidaan myös edistää olemassa olevaa tietokantojen kopiointityökalua. Lisäksi tarkoitus on työn jälkeen kehittää automaattisesti yhteyksiä mallintava ohjelma, joka tallentaa tietokantataulujen väliset yhteydet tämän työn aika suunniteltuun paikkaan.

Projekti aloitettiin syksyllä 2015, jolloin tehtiin määrittelyä ja suunnittelua. Näkyvämmät osiot työstä tehtiin keväällä 2016.

Valmiilla ohjelmalla voidaan tulostaa annetun tietokantataulun ja sen yhteystaulujen dataa ja valmiissa työssä on määritelty oma paikkansa, johon yhteystaulut määritellään.

Asiasanat
OpenEdge ABL, Digia Enterprise, Relaatiotietokanta

HAAGA-HELIA
ammattikorkeakoulu

Abstract

2.5.2016

Degree programme

Authors Mikkonen Antti	**Group** 2012
The title of thesis Programming a database relations modeling program and relational data printing program for Digia Enterprise ERP	**Number of pages and appendices** 19 + 0
Supervisors Outi Virkki	

This thesis was commissioned by for Digia Ltd which is Finnish software and services company.

The aim of the thesis is to develop a printing program for Digia Enterprise ERP. The study also planned how and where the relations between database tables are modeled. With the printing program one can print data according to the modeled database relations.

The thesis also paved the way for further development, where the aim would be to diversify the printing program so that it can be used to print information in the XML format. This study has revealed that it is also possible to advance already existing replication program. In addition, the purpose of the further development is to develop automatic modelling program for database relations. Such program would save the database relations like it is planned during this study.

This project started with the definition and planning phase at autumn 2015. More significant parts were implemented in the spring of 2016.

As a result the program can be used to print out data from the given database table and the tables that have relations with it. The database relations can be saved like it is planned in a ready study.

Key words
OpenEdge ABL, Digia Enterprise, Relational database

Sisällys

1 Johdanto

Tämä opinnäytetyö toteutettiin toimeksiantona Digia Oyj:lle, joka on Suomalainen ohjelmisto- ja palveluyritys. Digian liiketoimintana on tuottaa palveluita, toiminnan ohjaamista ja tiedon hyödyntämistä alan johtaville toimijoille. Digia työllistää Suomessa lähes 1 000 henkilöä ja liikevaihto on lähes 100 miljoonaa euroa (Digian vuosikertomus 2013).

Toimeksianto toteutettiin Digia Enterprisen tuotekehitykselle. Enterprise on toiminnanohjausjärjestelmä, joka soveltuu parhaiten tukkukauppaan ja valmistavaan teollisuuteen. Enterprisestä löytyy myös omat versionsa elintarviketeollisuudelle ja leipomoille.

Digia Enterprise käyttää Progress OpenEdge tietokantaa, joka koostuu useasta sadasta tietokantataulusta. Tietokantataulujen väliset yhteydet on rakennettu, mutta näitä yhteyksiä ei ollut mihinkään mallinnettu. Tämän takia aiemmin dataa tulostaessa piti ohjelmallisesti kertoa kaikki taulut, joiden data haluttiin tulostaa.

Jotta tietokannasta voidaan tulostaa dataa yhden taulun ja siihen liittyvien taulujen kentistä, tuli yhteydet mallintaa tietokantaan ja ohjelmoida tulostusohjelma, jolla yhteystauluja käydään läpi. Yhteyksien mallinnusta varten suunniteltiin oma ohjelmansa, joka suorittaa yhteyksien mallintamisen olemassa olevien indeksikenttien avulla. Tätä ohjelmaa ei kuitenkaan toteutettu kokonaisuudessaan vaan sen tulevaa toteutusta varten tehtiin selvitystyötä.

Opinnäytetyö hyödyttää Enterprisen kehityksen parissa työskenteleviä henkilöitä, jotka haluavat tulostaa isompaa datamassaa kerralla. Tätä datamassaa voidaan esimerkiksi siirtää ympäristöstä toiseen. Varsinkin asiakasprojektilta kehittäjän versioon testausta varten tietokantoja siirrettäessä tästä työstä on suuri hyöty.

Tämä opinnäytetyö jättää hyvät mahdollisuudet jatkokehittämiselle ja joitakin ideoita on keretty jo miettimään. Esimerkiksi tämän toiminnollisuuden pohjalta voidaan kehittää XML-tulostus ja tätä XML-tulostetta voitaisiin hyödyntää vaikka Microsoft Vision kanssa niin, että Visio piirtäisi tietokannasta tietokantakaavion. Toinen mahdollinen jatkokehityshanke olisi tietokannan replikoinnin jälleen kehittäminen niin, että voitaisiin replikoida asiakkaan kanta kehityskantaan käyttäen tietokantaulujen yhteyksiä. Nykyisin kun replikointi pitää hoitaa käsin, kenttä kerrallaan.

Tämän opinnäytetyön ensimmäisessä kappaleessa kerrotaan relaatiotietokannoista ja siihen liittyvistä ominaisuuksista. Seuraavassa kappaleessa kerrotaan ohjelmointikielestä, Progress OpenEdge ABL, jolla toiminnanohjausjärjestelmä ja opinnäytetyönä tehdyt ohjelmat on kirjoitettu. Tämän jälkeen kerrotaan Enterprisen tietokannasta ja miten se on toteutettu. Lopuksi käsitellään kehitystyö ja työn tulokset, sekä pohdinnat.

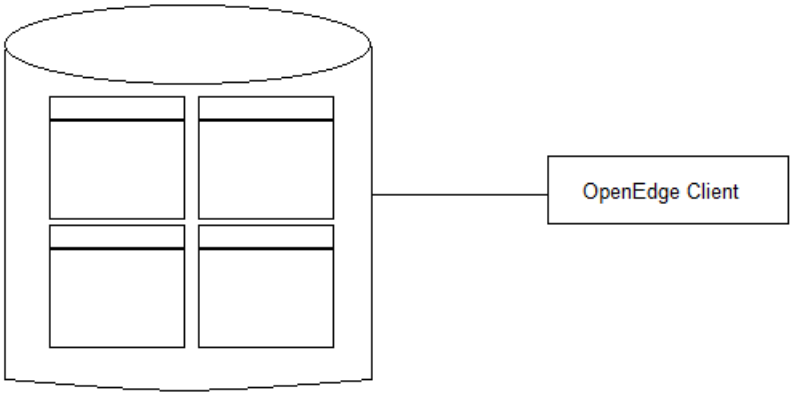
2 Relaatiotietokanta

Tietokannan tarkoitus on säilöä ohjelmassa tai järjestelmässä käytettyä tietoa. Tämä tieto voi olla esimerkiksi asiakastiedot, tuotteet, myyntitilaukset ja kaikki muu yrityksen liiketoiminnan pyörittämiseen tarvittava tieto. Tietokannan sisältämä tieto tulee olla käytettävissä, eli tietoa voidaan viedä tietokantaan ja sitä voidaan myös sieltä lukea. Järjestelmä on aina yhteydessä käytössä olevaan tietokantaan, josta se osaa tulostaa näkyviin halutut tiedot tietokannasta. (Oracle)

2.1 Tietokanta

Tietokanta koostuu tietokantatauluista, joihin voidaan tallentaa tietoa. Se mitä tietoa tietokantaan tallennetaan, riippuu tietokannan käyttötarpeesta. Esimerkkinä käytetään myynti- ja tilaustietokantaa, johon on tallennettu myös asiakasrekisteri.

Tietokanta on yhteydessä järjestelmään tai ohjelmaan, joka tietokantaa hyödyntää. Järjestelmä tai ohjelma johon tietokanta on kytketty, voi muuttaa tietokannan sisältöä lisäämällä tai poistamalla tietueita, muuttamalla tallennettua tietoa tai näyttää halutut tiedot käyttäjälle.



Kuva 1. Tietokanta sisältää tietokantatauluja ja on yhteydessä ohjelmaan tai järjestelmään, joka sitä hyödyntää.

2.2 Kentät

Tietokantataulu koostuu kentistä (kuva 2), joihin tallennetaan tietueen tiedot. Jokaiselle tietokantataululle määritellään oma yksilöivä kenttä, johon tallennetaan tietueen yksilöivä tieto. Tämä voi olla esimerkiksi asiakasnumero, joka on jokaisella asiakkaalla eri. Loput asiakkaaseen liittyvät tiedot tallennetaan omiin kenttiinsä asiakastaulussa. (Cengage)

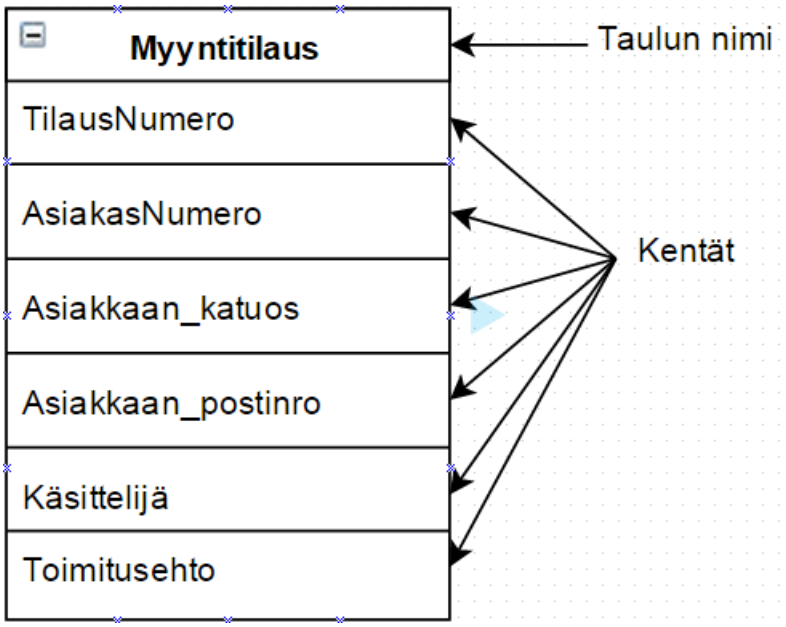
Yhteen kenttään tulee tallentaa yhden muotoista tietoa, esimerkiksi postinumero ja postitoimipaikka tulee tallentaa omiin kenttiinsä, etunimi ja sukunimi ovat omat kenttänsä jne. Tästä käytetään termiä kentän atomisuus (Jyväskylän yliopiston IT-tiedekunta).

Kentälle määritellään aina nimi ja tietotyyppi. Tietotyyppi voi olla esimerkiksi merkki, numeraalinen, päivämäärä tai looginen. Käytettävissä olevat tietotyypit ja niiden nimitykset vaihtelevat käytettävästä tietokannan hallintajärjestelmästä riippuen (OKOL).

2.3 Tietokantataulut

Tietokantataulut koostuvat kentistä ja tietueista.

Tieto tallennetaan tietueisiin, jotka koostuvat kentistä ja sisältää kaiken tiedon yhdestä tietystä henkilöstä, yrityksestä tms. Esimerkin myyntitilaustauluun voidaan tallentaa yksi tietue yhdestä myyntitilauksesta. Seuraava myyntitilaus tallennetaan seuraavaan tyhjään tietueeseen.



Kuva 2. Esimerkki myyntitilaustaulusta.

Tietueisiin tallennetut tiedot voidaan kuvata taulukko 1:n mukaisesti. Tällöin näemme selvästi, mihin kenttään on tallennettu mitäkin tietoa.

Taulukko 1. Jokainen rivi on oma tietueensa myyntitilaus -taulussa.

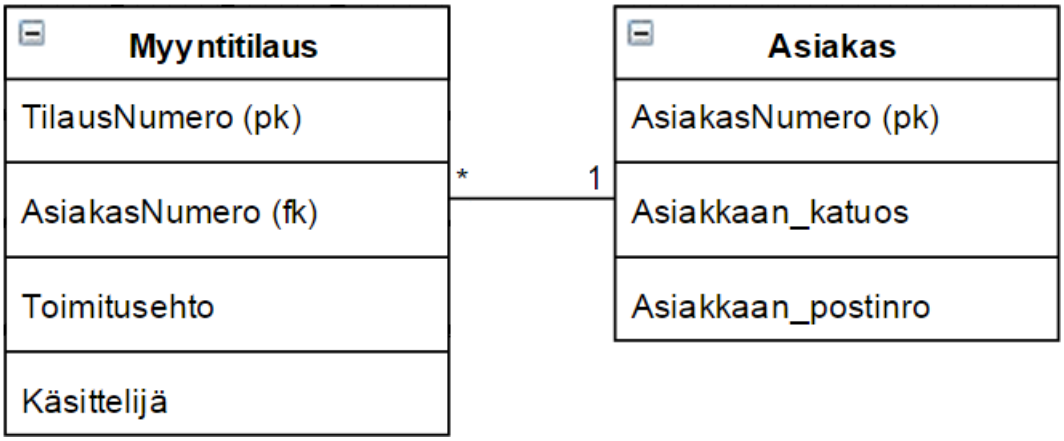
Myyntitilaus -taulu					
TilausNumero	**AsiakasNumero**	**Asiakkaan_katuos**	**Asiakkaan_postinro**	**Käsittelijä**	**Toimitusehto**
1001	1	Tie 3	00100	AnMi	FCA
1002	2	Katu 2	00200	AnMi	DAP
1003	1	Tie 3	00100	AnMi	EXW
1004	3	Kuja 10	32010	KeKo	FCA
1005	4	Väylä 253	01300	SiHo	DAP

2.4 Avaimet

Relaatiotietokannassa jokaiselle taululle tulee määritellä pääavain (primary key, pk). Pääavain yksilöi kyseisen taulun sisältämät tietueet. Jokaisella asiakkaalla on oma asiakasnumeronsa, joka ei koskaan ole kahdella asiakkaalla sama. Tällöin asiakastaulusta voidaan etsiä asiakasnumerolla varmasti oikea tietue. Tätä yksilöllistä arvoa kutsutaan uniikiksi. Jokaisen tietueen pääavaimella tulee olla arvo, eli se ei saa olla tyhjä (null). Tätä kutsutaan pääavaimen eheydeksi (Jyväskylän yliopiston IT-tiedekunta.)

2.5 Tietokantatauluyhteydet

Tietokantataulujen kentistä voidaan viitata jonkin toisen taulun pääavaimeen. Tällöin pääavaimeen viittaavan toisen taulun kenttää kutsutaan viiteavaimeksi (foreign key, fk). Näiden kenttien arvon tulee siis olla sama. Esimerkissä myyntitilaustaulun asiakasnumero-kenttään on viitattu asiakastaulusta asiakasnumero-kentällä. Tällöin tämä kenttä on viiteavain. Näin tekemällä ei tarvitse tallentaa myyntitilaustauluun kaikkia asiakastietoja, vaan ne voidaan hakea asiakastaulusta. Tämä taas estää saman datan esiintymisen monessa myyntitilaustaulussa, koska yhdellä asiakkaalla voi olla useampi myyntitilaus. Tämä on esimerkissä kuvattu yhteydessä merkeillä 1 ja *. Ilman omaa asiakastaulua, kirjoitettaisiin saman asiakkaan samat tiedot siis useaan myyntitilaustauluun.



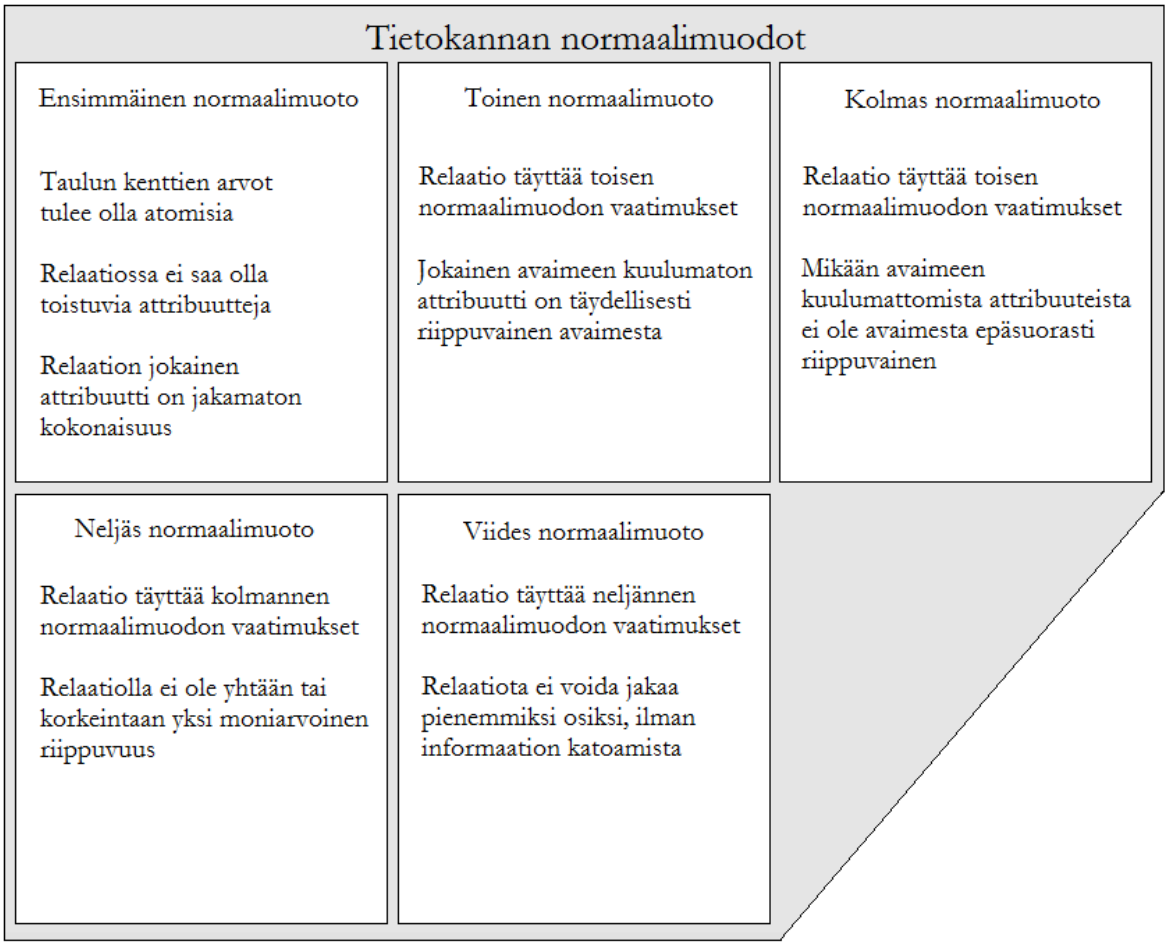
Kuva 3. Taulujen välinen yhteys on nyt mallinnettu viiteavaimella.

2.6 Tietokannan normalisointi

Normalisointi on tietokannan tietojen järjestämisen prosessi. Siihen sisältyy taulukoiden luomista ja suhteiden järjestämistä taulukoiden välille noudattamalla sääntöjä, jotka on suunniteltu sekä suojaamaan tietoja että tekemään tietokannasta entistä joustavamman poistamalla redundanssin ja epäyhtenäiset riippuvuussuhteet (Microsoft 2016).

Toistuvat tiedot, redundantit, kuluttavat turhaan levytilaa ja lisäksi näistä aiheutuu tietokannan ylläpidollisia ongelmia. Esimerkissä annetussa myyntitilaus -taulussa on asiakkaan tiedot viety omaan tauluunsa. Näin asiakkaan muuttuneita tietoja ei tarvitse jokaisesta myyntitilaus -taulusta muuttaa, vaan ne muutetaan asiakkaan tauluun ja näkyvät muuttuneina kaikissa niissä taulun tietueissa, joissa on viittaukset tämän asiakkaan taulun tietueeseen.

Tietokannan normalisoinnissa on sääntöjä, joista puhutaan normaalimuotoina. Näitä normaalimuotoja on viisi ja ne on kerrottu kuvassa 4.



Kuva 4. Tietokannan viisi normaalimuotoa (Virkki 2012).

Kuvassa mainitun, neljännen normaalimuodon, moniarvoinen riippuvuus tarkoittaa kun taulussa yksi tai useampi kenttä viittaa yhteen tai useampaan muuhun kenttään samassa taulussa (SQL Destination 2013).

Mikäli tietokanta ei ole ensimmäisessä normaalimuodossa, saattaa se aiheuttaa epätäydellisen tiedostorakenteen (Microsoft 2016).

2.6.1 Normalisoinnin esimerkki

Taulukko 2. Esimerkki normalisoimattomasta myyntitilaus-taulusta.

Myyntitilaus					
Asiakasnumero	Nimi	Osoite	Puhnro	Myyntitilaus 1	Myyntitilaus 2
123	Mikko Esimerkki	Katu 3, 00010 Helsinki	0404040404	112	113
142	Nelli Esimerkki	Kuja 2 B 1, 01300 Vantaa	0983955874	114	115

Kuten esimerkistä huomataan, on asiakastiedot määritetty suoraan myyntitilaus -tauluun. Tällöin joudutaan myyntitilaus -tauluun luomaan sarakkeita niin monta, että kaikkien asiakkaiden myyntitilausten määrä riittäisi. Tämän arvaaminen on käytännössä mahdotonta. Tässä tapauksessa myyntitilaus on toistuva tieto. Lisäksi osoitteeseen on tallennettu useampaa tietoa; osoite, postinumero ja postitoimipaikka.

Taulukko 3. Esimerkki tietokantataulusta ensimmäisessä normaalimuodossa.

Myyntitilaus						
Asiakasnumero	Nimi	Osoite	Postinumero	Postitoimipaikka	Puhnro	Myyntitilaus
123	Mikko Esimerkki	Katu 3	00010	Helsinki	0404040404	112
123	Mikko Esimerkki	Katu 3	00010	Helsinki	0404040404	113
142	Nelli Esimerkki	Kuja 2 B 1	01300	Vantaa	0983955874	114
142	Nelli Esimerkki	Kuja 2 B 1	01300	Vantaa	0983955874	115

Nyt tauluun on luotu jokaiselle myyntitilaukselle oma rivinsä ja se on ensimmäisessä normaalimuodossa, mutta voimme todeta, että se sisältää redundantteja tietoja. Asiakkaan tiedot toistuvat monella rivillä.

Taulukko 4. Esimerkki asiakastaulun toisesta normaalimuodosta.

Asiakas					
Asiakasnumero	Nimi	Osoite	Postinumero	Postitoimipaikka	Puhnro
123	Mikko Esimerkki	Katu 3	00010	Helsinki	0404040404
142	Nelli Esimerkki	Kuja 2 B 1	01300	Vantaa	0983955874

Taulukko 5. Esimerkki myyntitilaustaulun toisesta normaalimuodosta.

Myyntitilaus	
Tilausnumero	Asiakasnumero
112	123
113	123
114	142
115	142

Nyt taulu on jaettu kahteen tauluun, asiakas ja myyntitilaus, jotta redundanttia tietoa ei olisi. Myyntitilaus-tauluun on tuotu vierasavaimena asiakasnumero, joten tiedot haetaan aina asiakas-taulusta. Samalla taulut tulivat kolmanteen normaalimuotoon, koska kaikki taulujen kentät liittyvät taulun avaimeen. Jos myyntitilauksella olisi esimerkiksi kaikki tuotetiedot, tulisi niistä vielä tehdä oma itsenäinen taulu. Tämä voitaisiin toteuttaa niin, että myyntitilaukselta tehtäisiin yhteydet tilausrivi-tauluun, josta taas viitattaisiin tuote-tauluun.

3 OpenEdge ABL

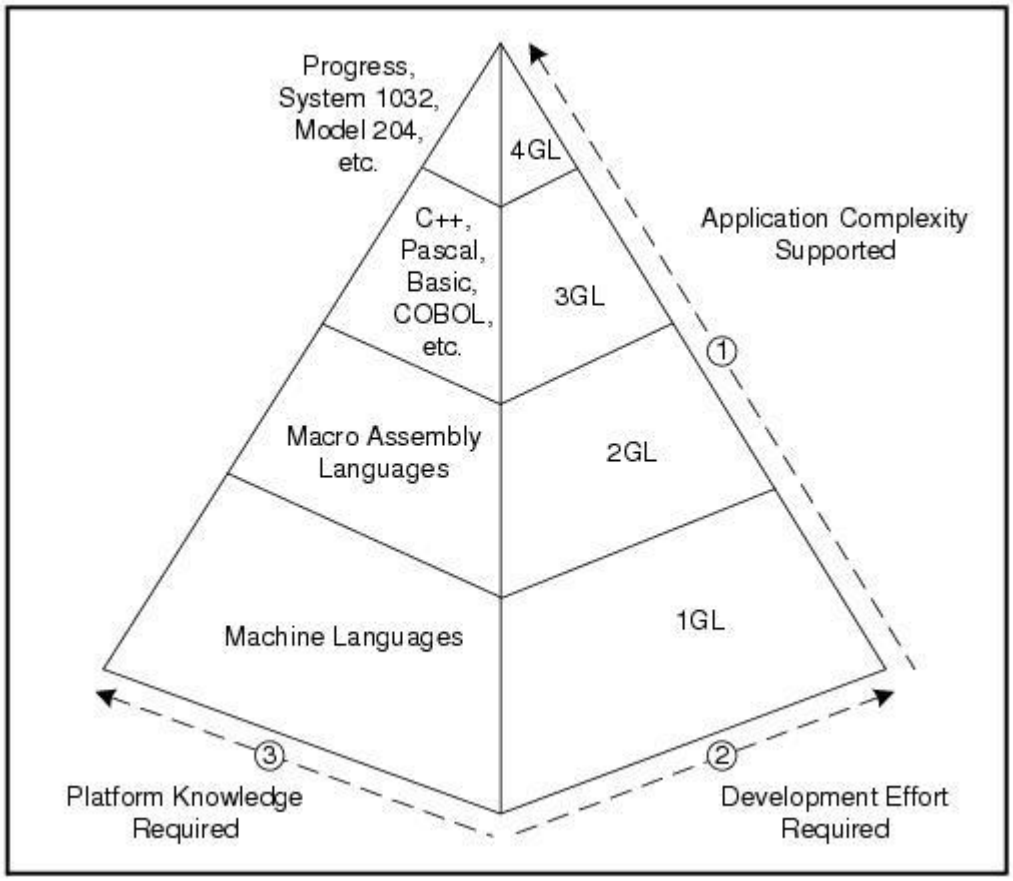
OpenEdge ABL (Advanced Business Language), aiemmin Progress 4gl, on Progress Software Corporation:in kehittämä neljännen sukupolven ohjelmointikieli yritysten sovellusten tai ohjelmistojen kehittämiseen. Ohjelmointikielen syntaksi on hyvin samankaltaista englannin kielen kanssa ja se sisältääkin hyvin vähän erikoismerkkien käyttöä.

ABL on proseduurillinen ohjelmointikieli. Tällä tarkoitetaan sitä, että ohjelmoija voi kirjoittaa lausekkeita, jotka voidaan tallentaa omiin yksilöityihin proseduureihin. Yleensä nämä lausekkeet on suoritettu tai prosessoitu siinä järjestyksessä, kun ne on proseduureihin kirjoitettu (Sadd 2006).

ABL:ssä proseduuri muodostuu lohkoista. Proseduuri itsessään on päälohko itselleen ja kieli tukee monia tapoja määrittää muita lohkoja päälohkon sisälle. Hyvä esimerkki sisäkkäisistä lohkoista (nested block) on FOR EACH lauseke ja siihen liittyvä END lauseke. Tällä käydään läpi kaikki tietueet tietokannasta ja suoritetaan kaikki koodi, jota lausekkeiden väliin on kirjoitettu (Sadd 2006). Tästä kerrotaan lisää myöhemmin tässä työssä.

3.1 Kevyempi ohjelmointiratkaisu

Campbellin (2004) mukaan OpenEdge ABL on ohjelmointiratkaisuna kevyempi kuin muut tunnetut kielet. Se vaatii vähemmän tuntemusta alustasta, sekä vähemmän vaivaa ohjelmointiratkaisuissa.



Kuva 5. ABL:n vaativuus muihin ohjelmointikieliin verrattuna (Campbell 2004).

Kuvasta 5 voimme päätellä, että jokainen ohjelmointikielen sukupolvi:

- Tuottaa edellistä sukupolvea suuremman kapasiteetin monimutkaisten sovellusten hallintaan
- Vaatii vähemmän vaivaa ohjelmoijalta
- Vaatii vähemmän tietämystä ja hallinnointia alustasta, jolla sovellus ohjelmoidaan ja ajetaan.

ABL tukee monimutkaisia sovelluksia tarjoamalla sovellusorientoituja ratkaisuja ohjelmointikielessä (Campbell 2004). Alkeellisen aritmetiikan ja loogisten toimenpiteiden lisäksi ABL mahdollistaa sovellusorientoituneet toimenpiteet muun muassa tiedon lukemiseen ja kirjoittamiseen, tietokannassa navigoimiseen, etsimiseen, lajitteluun, kellonajan ja päivämäärän muuntamiseen, sekä käyttäjäympäristön hallintaan. Jotta näiden hallinta olisi mahdollisimman helppoa ja luotettavaa, tarjoaa ABL ajonaikaisen managerin (runtime manager).

Vaikkakin neljännen sukupolven ohjelmointikieli tuo mukanaan paljon hyötyä ja helpotusta, tuo se myös mukanaan paljon rajoitteita, joihin ohjelmoija ei törmää alemman sukupolven ohjelmointikielissä. Kuten Voelker (2010) artikkelissaan mainitsee, on kolmannen sukupolven

kielissä etuna modulaarisuus ja laajennettavuus. Laajennettavuuden ansiosta esimerkiksi apurungot (framework) ovat yleistyneet ohjelmoijien keskuudessa.

3.1.1 Vähemmän vaivaa ohjelmointiratkaisuissa

Kuten aiemmin mainittiin, ABL vaatii vähemmän vaivaa ohjelmointiratkaisuissa, tarjoamalla perusjoukon toimintoja, joilla voidaan tehdä paljon työtä. Kieli kuitenkin tarjoaa mahdollisuuden näiden perusjoukon toimintojen hallintaan. Tämä mahdollistaa ohjelmoijalle toimintojoukon käyttämisen yhä laajempaan ja monipuolisempaan käyttöön, koska jokainen ratkaisu voidaan räätälöidä kulloisenkin tarpeen mukaan.

Yhden sovellusikkunan tukemiseen voidaan käyttää perusjoukon toimintoja ja tämä eroaa kun verrataan esimerkiksi kieliin C tai COBOL, joissa tarvitaan monia erilaisia input/output–tapahtumia, sekä prosessoinnin tarkastelua operaation eri kohdissa. ABL:n yhdellä perusjoukon toiminnolla voidaan hallita yhtä tai useampaa sovellusikkunaa, riippuen ohjelman vaatimuksista (Campbell 2004).

Esimerkiksi yksi yksinkertaisimmista ja helpoimmista, mutta samalla tehokkaimmista työkaluista ABL:llä on FOR-lauseke (statement). Tämä lauseke tarjoaa iteroivan tietueiden lukemisen ja niin vahvan näytön hallinnan (display management), että sillä voidaan luoda oma itsenäinen sovelluksensa. Kuvassa 6 on kuvattu klassinen esimerkki FOR-lausekkeesta sovelluksena.

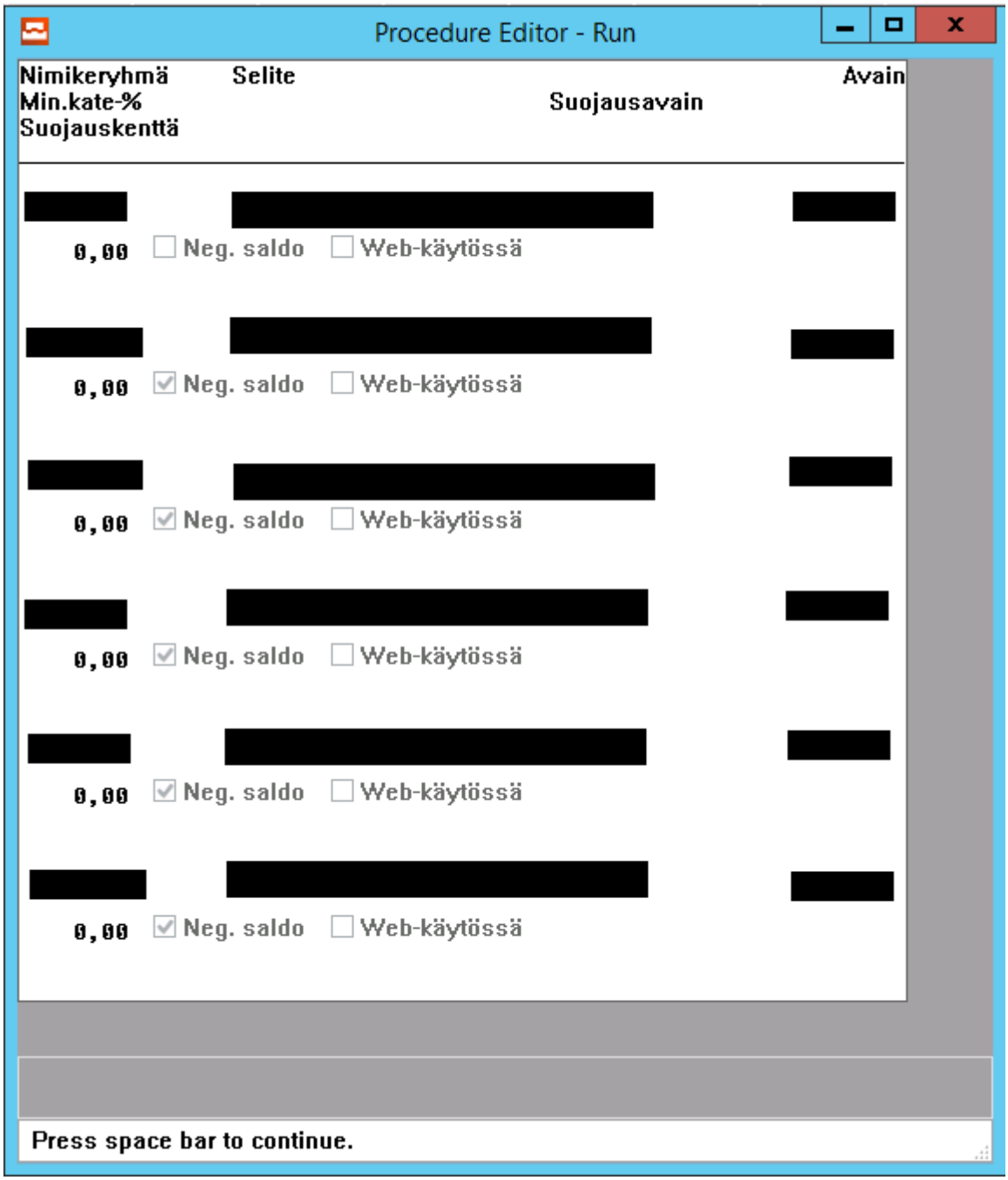
```
FOR EACH Customer:
    DISPLAY Customer.
END.
```

Kuva 6. FOR-lauseke.

Tätä lauseketta yksinkertaisempaa ei juurikaan voi tehdä. Sadd (2006) sanoo, että sen selittämiseen mitä kaikkea tuo lauseke tekee, menisi tunteja. Hän on kuitenkin summannut lausekkeen seuraavasti:

- ”Customer” on taulun nimi Sports2000 harjoitustietokannassa. FOR EACH lauseke aloittaa koodilohkon, joka avaa kyselyn annettuun tietokantatauluun ja palauttaa jokaisen tietueen tietokantataulusta iteroivasti yksi kerrallaan.
- Jokainen tietue Customer-taulusta näytetään tulostusnäytöllä. Koodi hakee muotoilun ja otsikkotiedot tietokannan mallin (schema) määrityksistä ja käyttää näitä tietoja näyttääkseen kaikki kentät tietokantataulusta. DISPLAY komento siis tarkoittaa, että se tulostaa tulostusnäytölle kaikki kentät annetusta taulusta.

- Kun jokainen kenttä on tulostettu, siirrytään seuraavalle riville tulostamaan seuraavaa Customer –tietuetta.
- Koko koodilohko FOR EACH lausekkeesta END komentoon käy iteroiden komennot näiden väliltä. Näin ollen koodi tulostaa DISPLAY-komennolla yhden tietueen Customer-taulusta, niin kauan kun tietueita on tulostamatta.
- Kun näyttö on tulostanut tulostusnäytön alalaitaan asti tietoa, koodi pysähtyy automaattisesti ja tarjoaa viestin, jossa pyydetään painamaan välilyöntipainiketta näppäimistöltä, jotta voidaan siirtyä seuraavalle sivulle.
- Välilyöntipainikkeen painettua koodi tyhjentää tulostusnäytön ja aloittaa tulostamaan seuraavan datasetin.
- Kun ABL Virtual Machine (AVM) havaitsee tietueiden loppumisen, se lopettaa koodin suorittamisen ja tulostaa tulostusnäytön alalaitaan tekstin ”Procedure complete. Press space bar to continue.”.
- Jos toimenpiteen haluaa keskeyttää ennen tietueiden loppumista, voidaan ESC-painikkeella lauseke päättää.



Kuva 7. Kerätyn datan näyttäminen. Kuvan data on tulostettu Enterprise tietokannasta ja se on sensuroitu mustilla palkeilla.

Managerin toiminta tuodessa tietueiden sisältö näkyville kuvataan kuvassa 7. Näitä sivuja tuodaan näyttöön niin monta, kun tietuissa on dataa. Kuten kuvasta voimme havaita, on näyttöön tuotu ensimmäinen sivu ja alalaitaan syötetty teksti ”Press space bar to continue”. Painamalla välilyöntiä tulee seuraava sivu näkyviin, ja näitä sivuja näytetään kunnes kaikki tietueitten data on tulostettu. Tällöin viimeinen välilyönnin painallus sulkee tulosteikkunan.

3.1.2 Vähemmän tietämystä alustasta

ABL:n ajonaikainen ympäristö poistaa kehittäjän tarpeet huomioida monia alustan hallinnointiin liittyvistä huolista. Sovelluskehittäjän ei tarvitse juurikaan opetella alustan tai kehitysympäristön, jolle sovellus tai ohjelma on tehty, kääntäjistä (compiler), linkkaajista (linker) tai debuggereista (debugger).

3.2 ABL:n hyödyt

ABL on etevä seitsemällä osa-aluella, jotka kaikki tuetaan tässä ohjelmointikielessä (Campbell. 2004).

Kuva 8. ABL:n vahvuudet Campbellin mukaan.

Monia kuvan 8 ominaisuuksia tuetaan myös muilla ohjelmointikielillä ja myös ABL:stä löytyy omat heikkoutensa. Omat kokemukset ovat osoittaneet, että OpenEdge ABL ohjelmiston asennukset menevät välillä kuin itsestään sekaisin. Tämän johtaa uudelleenasennukseen, joka vie aina aikaa puolesta tunnista ylöspäin. Vastaavia ongelmia en ole esimerkiksi Javalla kohdannut.

4 Digia Enterprise

Digia Enterprise käyttää kahta Progress OpenEdge –tietokantaa, POWDB ja PGDB. PGDB on ohjaustietokanta (control database), joka sisältää ohjelmamääritykset, valikot, käyttäjät, luvat, kielikäännökset, automaatioketjut, periytyvyydet jne. Tämän tietokannan koko on noin 80 taulua. POWDB taas on operatiivinen tietokanta, sisältäen mm myyntitilaukset, nimikkeet ja asiakkaat. POWDB on kooltaan suuri tietokanta, sisältäen noin 800 tietokantataulua. Kumpaankaan tietokantaan ei ole taulujen välisiä yhteyksiä mallinnettu. Tauluilla on kyllä keskenään viittaavia tietoja, esimerkiksi myyntitilauksella on tilausnumero, joka on myös myyntitilausrivillä. Tämä tieto on ohjelmoijan pääteltävä itse, koska mallinnettuja yhteyksiä ei ole. Joskus taulujen väliset suhteet ovat haastavia tai jopa mahdottomia kehittäjän pääteltäviksi.

Enterprisen tietokanta ei myöskään ole normalisoitu. Tietokannasta löytyy esimerkiksi tauluja, joissa pääavaimen arvoksi on tallennettu pilkkueroteltuja listoja. Yksi tällaisista tauluista on SysLink. Tämä rikkoo normalisoinnin ensimmäistä normaalimuotoa, kuten kappaleessa 2.6 kerrottiin. Näitä tauluja on koko kannassa vain muutama ja muut taulut ovatkin pääsääntöisesti vähintään kolmannessa normaalimuodossa.
Koska tietokanta ei ole kokonaisuudessaan edes ensimmäisessä normaalimuodossa, tulee tämä ongelma tiedostaa jatkokehittäessä mallinnusohjelmaa. Jos esimerkiksi SysLink-taulu viitataan toiseen tauluun ohjelmassa, aiheuttaa se koko pilkkuerotellun listan päätymisen viiteavaimeksi.

5 Kehitystyö

Tässä kappaleessa kerron kehitystyöstä vastaten kysymyksiin mitä tehtiin, miten ja miksi. Kerron myös mitä työkaluja ja viitekehyksiä työn edistämiseen käytettiin. Kehitystyö toteutettiin Digia Enterprisen tuotekehityksen työnä, jolle työtunnit kirjattiin.

5.1 Tarve

Tietokannoissa ei ollut mallinnettu tietokannan rakenteen relaatioita kuten normaaleissa relaatiotietokannoissa. Kannoissa on myös viittauksia tauluista toisiin, jotka eivät ole kannan rakenteesta pääteltävissä, tai joissa päättely on monimutkaista.
Digian Enterprise kehittäjillä ei ollut keinoa, jolla tietoa voitaisiin tulostaa tietokannasta suurempia kokonaisuuksia kerrallaan. Ei ollut myöskään selvää keinoa, jolla kannan rakenteita voisi tarkastella.
Mallinnettuja tietokantataulujen välisiä yhteyksiä halutaan hyödyntää myös tulevissa tuotekehityksen projekteissa.

5.2 Projektinhallinta ja työkalut

Tämä kappale kertoo projektissa käytetyistä työkaluista.

5.2.1 JIRA

Työtä hallittiin Atlassian tuoteperheen JIRA-projektinhallintatyökalulla. Työstä oli kirjattu omat JIRA-työt, joissa oli työstä tarpeet ja kuvaukset. JIRAsta käytettiin myös hyödyksi vanhempia JIRA-töitä, joilla etsin tähän työhön liittyviä töitä, joilta voisi löytyä kulloinkin vallitsevaan ongelmaani ratkaisu.

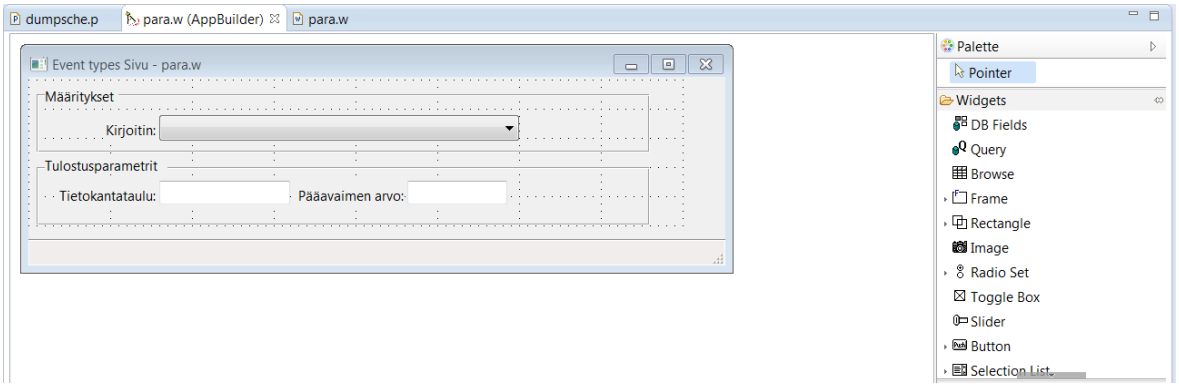
5.2.2 Tortoise SVN

Versionhallintana käytettiin yrityksessä käytettyä versionhallintaa, Tortoise SVN:ää. Kun sain jonkin ohjelmakokonaisuuden toimivana valmiiksi, vietiin se versionhallintaan. Tällä estettiin tiedostojen katoavuus, sekä tuotiin kullekin tiedostolle versiot, joihin voitaisiin tarpeen vaatiessa palata.

5.2.3 Progress 11.5

Ohjelmointikielenä toimi ABL (ent progress 4gl), jonka versio oli Progress 11.5. Progress 11.5 tuo mukanaan Developer Studion, joka perustuu Eclipse-ohjelmaan. Developer studiota käytettiin ohjelmointityökaluna.

Käyttäjänäkymä on rakennettu Progress 11.5 mukana tulleella AppBuilder-ohjelmalla. Tällä ohjelmalla voidaan rakentaa käyttäjänäkymä helposti lisäämällä painikkeet, kentät, valikot yms. haluttuihin paikkoihin. AppBuilder voidaan käynnistää suoraan Developer Studiosta.



Kuva 9. AppBuilder näkymä. Oikealla työkaluvalikko, josta voidaan valita lisättävät kentät yms.

5.3 Toteutusdokumentti

1 TARPEEN KUVAUS

Kehittäjille heräsi tarve tulostaa data relaatioittain tiedostoon. Tämä helpottaa tietokantataulujen välisten yhteyksien tarkastelua, sekä jatkossa datan siirtämistä tietokannasta toiseen.

Jotta dataa voidaan tulostaa tietokantataulujen välisiä yhteyksiä käyttäen, tarvitsee data joko tulostaa niin, että ohjelma ”haistelee” tietokantataulujen indeksikenttiä tai toisena vaihtoehtona taulujen yhteydet on mallinnettu omaan tauluunsa, jota voidaan käyttää apuna tulostaessa yhteystaulun dataa. Tässä on päädytty jälkimmäiseen ratkaisuun, jotta tarpeen vaatiessa yhteyksiä voidaan hallita käsin. Ei ole täyttä varmuutta, että tietokannassa kaikki indeksikentät on oikein tehty.

Tulostuksessa haluttiin voida tulostaa yhden päätaulun tietueen tiedot tai kaikkien päätaulun tietueiden relaatiossa olevien taulujen tiedot.

2 TOIMINNALLINEN RATKAISUN KUVAUS

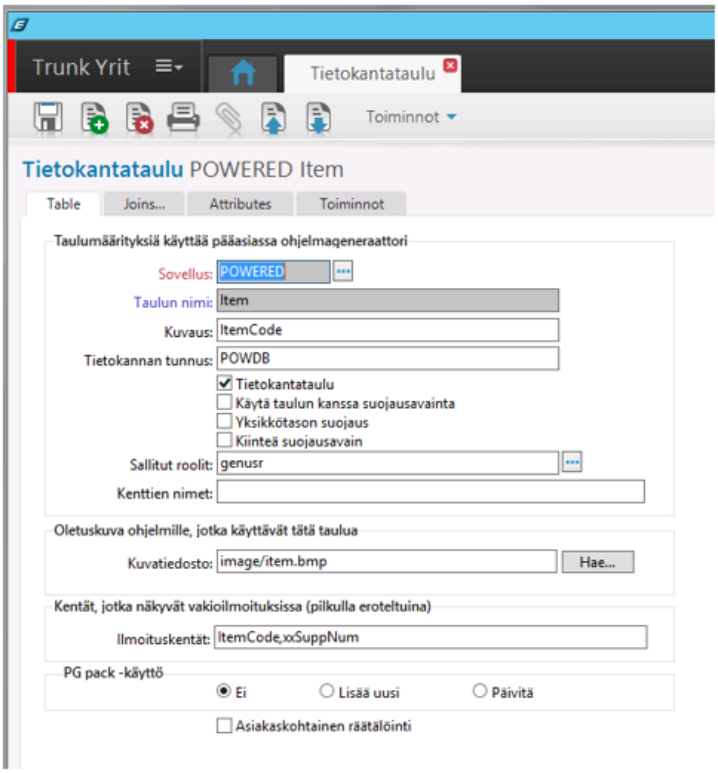
Käytettiin olemassa olevia tauluja, jotka eivät olleet käytössä. Nämä taulut ovat ”pgObjClass” ja sen alitaulu ”pgConnec”.

Tietokantatauluun ”pgObjClass” kerrotaan haluttu taulu, jonka yhteydet halutaan mallintaa. Yhteystaulujen mallinnus tehdään ”pgConnec” tauluun. Nämä näkyvät käyttäjälle käyttöliittymässä Tietokantataulu-ohjelman päävälilehdellä ja joins-välilehdellä.

3 TEKNINEN RATKAISUN KUVAUS

Tietokantataulu-ohjelman etusivulle määritellään haluttu taulu, jonka yhteydet halutaan mallintaa.

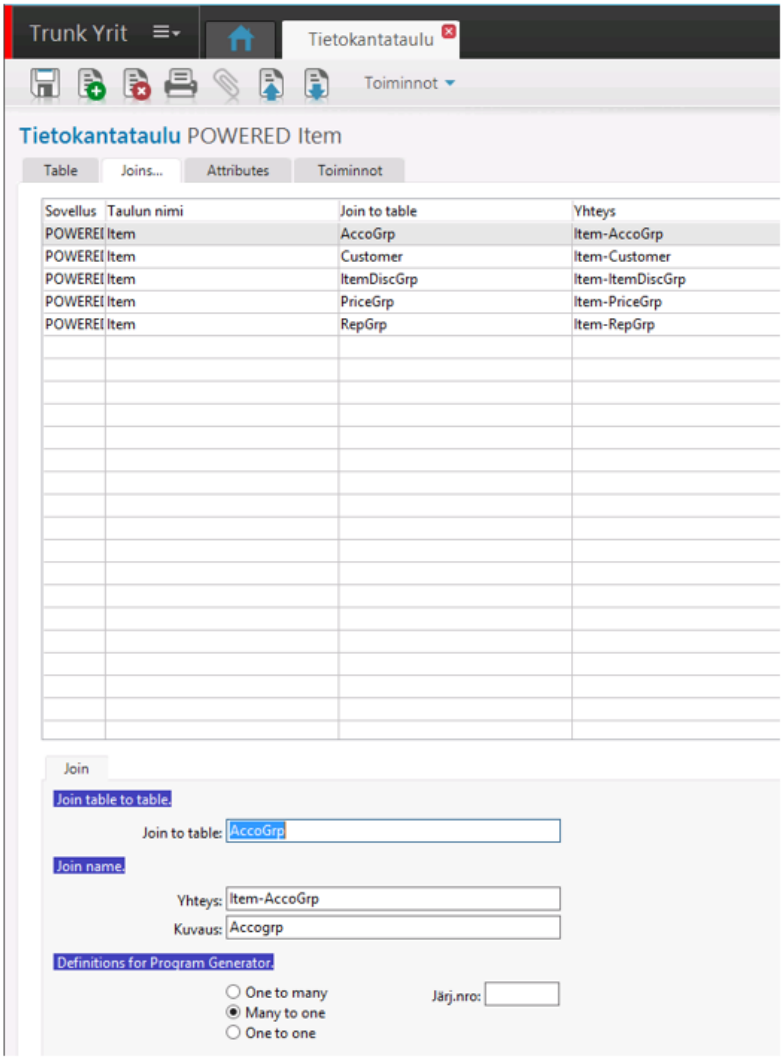
Kuva 10. Toteutusdokumentti. Sivu 1.



Tähän määritellään Tietokanta. taulun nimi, "kuvaus"-kenttään pääavain ja tietokannan tunnus. Muita tietoja ei käytetä tässä toteutuksessa.

Kentän tarkoitus	Kentän nimi	Kentän tyyppi
Sovellus eli tietokanta	Applid	Char
Tietokantataulun nimi	ObjectClass	Char
Kuvaus eli pääavain	ObjectDesc	Char
Tietokannan tunnus	DbaseName	Char

Kuva 11. Toteutusdokumentti. Sivu 2.

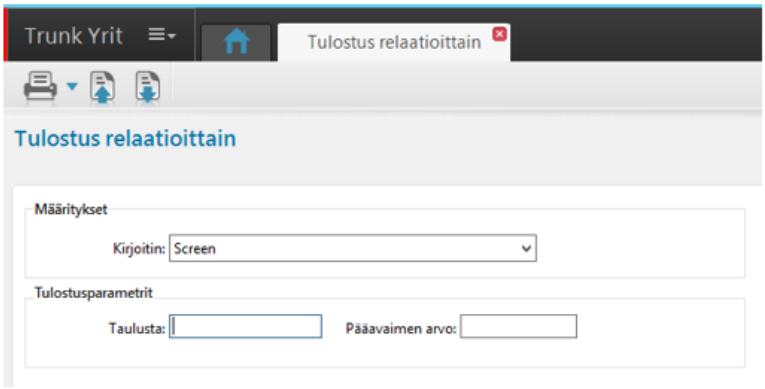


Joins-välilehdelle täytetään kaikki, tässä tapauksessa, Item-taulun yhteystaulut ja niihin liittyvät tiedot, jotka ovat: taulun nimi, yhteys, kuvaus (pääavain) ja suhde. Jokainen yhteystaulu näkyy listassa omana rivinään.

Kentän tarkoitus	Kentän nimi	Kentän tyyppi
Viittaus pgObjClass-tauluun (ei näkyvissä)	ObjectClass	Char

Kuva 12. Toteutusdokumentti. Sivu 3.

Join to table eli yhteystaulu	toObjectClass	Char
Yhteys	ConnName	Char
Kuvaus eli pääavain	ConnDesc	Char



Luotiin tulostusohjelma ”Tulostus relaatioittain”.

Kirjoittimen valintoina on yleisesti Enterprisessä käytetyt vaihtoehdot esim. screen, output to file, sähköposti, excel ja pdf.

Tulostusparametreiksi annetaan taulun nimi ja pääavaimen arvo-kenttään tietueen pääavaimen arvo tai *. Jos annettiin tietyn tietueen pääavaimen arvo, tulostetaan vain siihen tietueeseen liittyvät tiedot. *-valinnalla tulostetaan kaikkien tietueiden relaatiossa olevien taulujen data.

Kentän tarkoitus	Kentän nimi	Kentän tyyppi
Kirjoitin	fPrinterId	Char
Tietokantataulu, jonka yhteystauluista halutaan tulostaa	fTable	Char
Pääavaimen arvo annetulle taululle	fPrimaryKey	Char

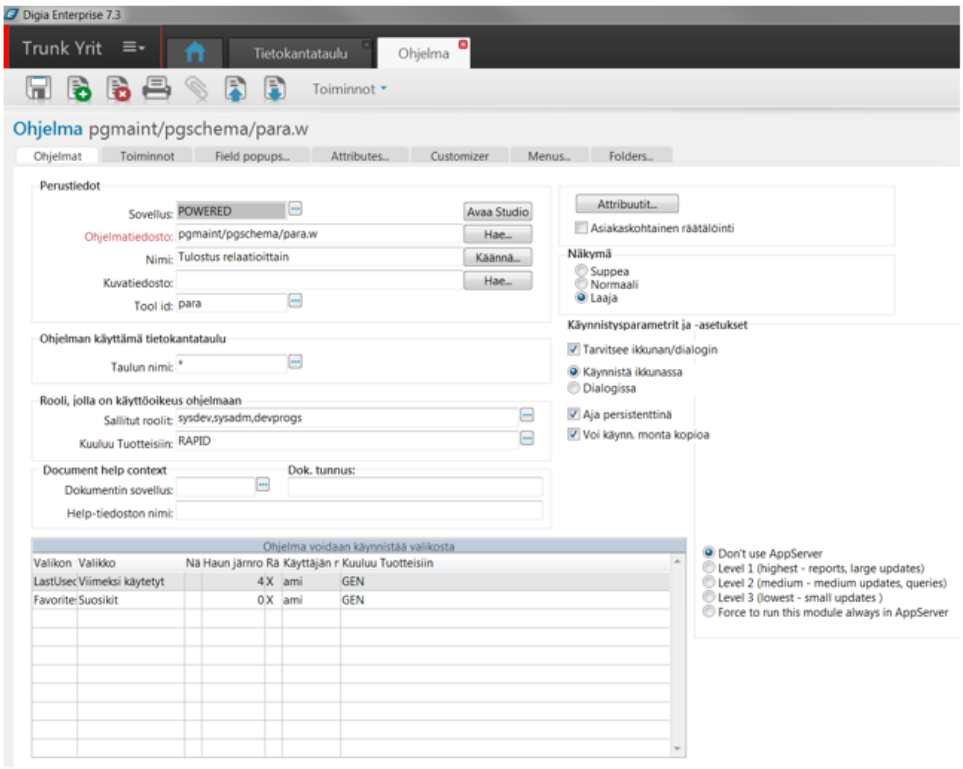
4 PARAMETRIT

Tässä kappaleessa kerrotaan, mitä kaikkia parametreja ohjelma tarvitsee toimiakseen oikein. Nämä parametrit määritellään Enterprise toiminnanohjausjärjestelmään.

Kuva 13. Toteutusdokumentti. Sivu 4.

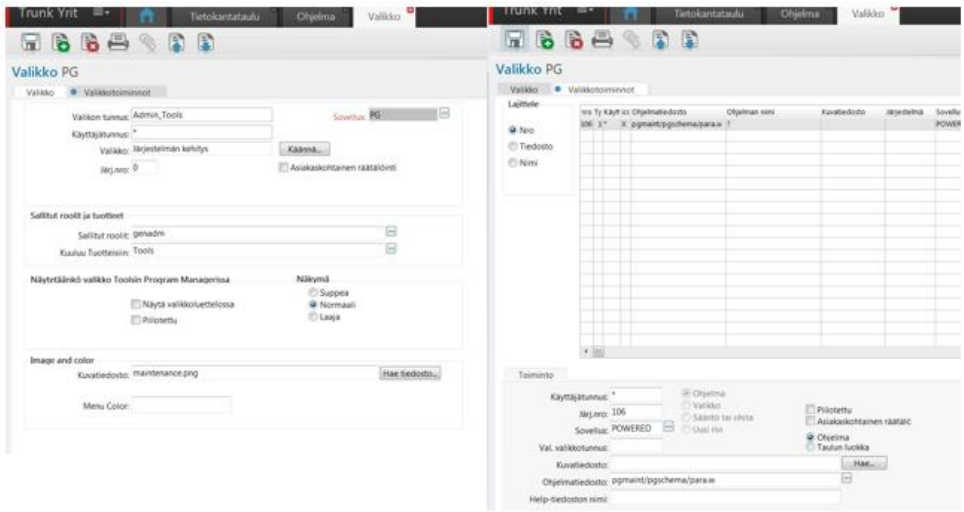
4.1 Ohjelmaparametrit

Tulostusohjelman käyttäjänäkymä on määritetty ohjelma-ohjelmaan.



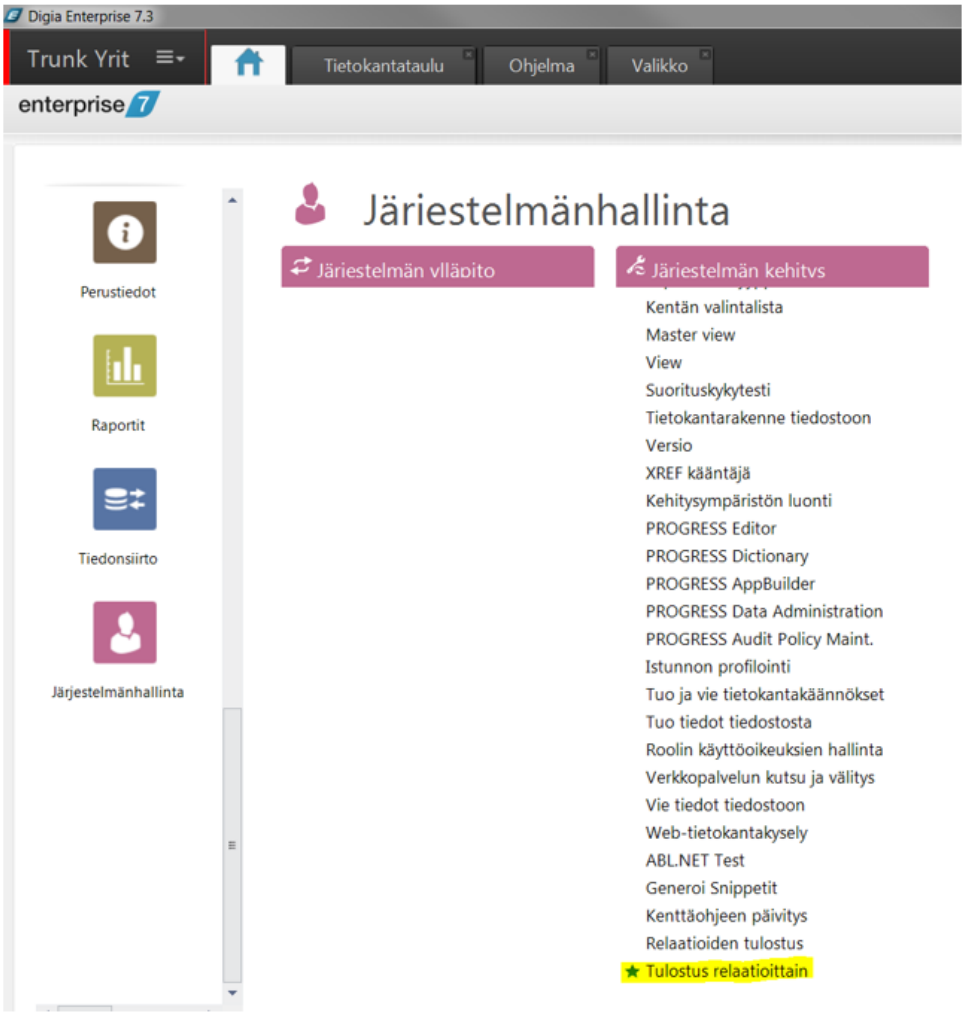
Tähän on kerrottu sovellus ja missä ohjelmatiedosto sijaitsee. Myös ohjelman nimi ja käyttäjäoikeudet kerrotaan tässä.

Kuva 14. Toteutusdokumentti. Sivu 5.



Valikko-ohjelmaan on määritelty ohjelman sijainti käynnistysvalikossa.

Kuva 15. Toteutusdokumentti. Sivu 6.



Käyttäjänäkymässä käynnistys näkyy Järjestelmänhallinta-osiossa.

5 TEKSTIT JA KÄÄNNÖKSET

Virhetilanteissa käyttäjälle tulostettaessa informaatiota on käytetty Enterprisessä olevia kielikäännöksiä. Näitä kutsutaan omilla ohjelmilla ja ne parametroidaan toiminnanohjausjärjestelmään.

Kuva 16. Toteutusdokumentti. Sivu 7.

5.1 Kielikäännökset

Kieli	Fraasi	Käännös
Suomi	POWERED%pgmaint/pgschema/dumpsche.p%No TableFound	Annettua tietokantataulua ei löytynyt määriteltynä tietokantataulu –ohjelmasta.

6 OHJELMAMODUULIT

Ohjelmat	Ohjelman kuvaus	Uusi/Muutettu /Käännettävä
pgmaint/pgschema/para.w	Tulostusohjelman käyttäjänäkymä	U
pgmaint/pgschema/dumpsche.p	Tulostusohjelman tulostuslogiikka	U
pgmaint/pgschema/repopara.i	Include tulostusohjelmalle, johon on määritetty muuttujat	U

Kuva 17. Toteutusdokumentti. Sivu 8.

5.4 Koodit

Tässä kappaleessa tarkastellaan tärkeimpiä osia ohjelmakoodeista.

AppBuilder-ohjelmalla on rakennettu käyttäjänäkymä (para.w) tulostusohjelmalle. Ohjelmaan on tehty pientä koodausta virheidenhallintaan tulostusohjelman käynnistysproseduuriin:

```
PROCEDURE PrintReport :
DO WITH FRAME {&frame-name}:
            ASSIGN {&LIST-1}.
    END.
    {pgincl/method/printre.i}
    /* Tarkistetaan, että tietokantataulu-ohjelmasta löytyy mallinnettuna käyttäjän antama taulu */
    IF NOT CAN-FIND(FIRST pgObjClass WHERE pgObjClass.ObjectClass = fTable:SCREEN-VALUE) THEN
    DO:
        RUN pgsubr/error.p(PGTransString("<<POWERED%pgmaint/pgschema/para.w%NoTableFound>>")).
        RETURN ERROR.
    END.
    /* Ajetaan tulostusohjelma */
    RUN RepRun("pgmaint/pgschema/dumpsche.p").

END PROCEDURE.
```

Kuva 18. Proseduuri PrintReport, jolla käynnistetään tulostusohjelma.

Tärkeimmät, tulostuslogiikan sisältävät koodit ovat dumpsche.p-ohjelmassa. Käydään ohjelma kohta kohdalta:

```
/* Määritellään temp-taulut */
DEFINE TEMP-TABLE wTables NO-UNDO
    FIELD TableName     AS CHAR
    FIELD JoinFieldName AS CHAR
    INDEX wTables IS UNIQUE TableName.

DEFINE TEMP-TABLE wDumped NO-UNDO
    FIELD TableName  AS CHAR
    FIELD TableRowId AS ROWID
    INDEX wDumped IS UNIQUE TableName TableRowId.

DEFINE TEMP-TABLE wStatements NO-UNDO
    FIELD TableName AS CHAR
    FIELD Statement AS CHAR
    INDEX wStatements IS UNIQUE TableName Statement.
```

Kuva 19. Väliaikaistaulujen (Temp-table) määrittely.

```
/* Määritellään muuttujat */
DEFINE VARIABLE xQuery       AS WIDGET-HANDLE NO-UNDO.
DEFINE VARIABLE xBuffer      AS HANDLE        NO-UNDO.
DEFINE VARIABLE xValue       AS CHAR          NO-UNDO.
DEFINE VARIABLE xStatement   AS CHAR          NO-UNDO.
DEFINE VARIABLE xRecordCount AS INT64         NO-UNDO.
DEFINE VARIABLE xProcedure   AS CHAR          NO-UNDO.
DEFINE VARIABLE wTable       AS HANDLE        NO-UNDO.
DEFINE VARIABLE xJoinToTable AS CHAR          NO-UNDO.

/* Haetaan includeina tarvittavat metodit, raportin otsikko ja
   tulostusohjelman muuttujien määrittelyt */
{pgrepo/method.i} /* Standard methods */
{pgrepo/rephead.i} /* Report header frame    */
{pgmaint/pgschema/repopara.i}

VIEW FRAME fRepHead. /* Header */
RUN RepDispParams. /* Display parameters */

FORM WITH FRAME fRepo.
RUN pgsubr/trans.p(FRAME fRepo:HANDLE, THIS-PROCEDURE:FILE-NAME).

VIEW FRAME fRepHead. /* Use generic report header */
```

Kuva 20. Muuttujien määrittely, sekä yleisten ohjelmien include-kutsut.

```
/* Jos käyttäjä jättää Pääavaimen arvo-kentän tyhjäksi, oletetaan
   että tulostetaan kaikki tietueet */
IF fPrimaryKey = "" THEN fPrimaryKey = "*".

/* Haetaan annettu taulu Tietokantataulu-ohjelmasta. */
FIND FIRST pgObjClass WHERE pgObjClass.ObjectClass = fTable NO-LOCK.
IF AVAILABLE pgObjClass THEN
DO:
    IF fPrimaryKey = "*" THEN
    DO:
        /* Luodaan kysely, jolla haetaan iteroivasti holleille kaikki taulun tietueet */
        ASSIGN
            xStatement = "FOR EACH " + fTable + " WHERE " + fTable + "." + pgObjClass.ObjectDesc + " <> " + QUOTER("") + " AND " +
                         "NOT " + fTable + "." + pgObjClass.ObjectDesc + " BEGINS " + QUOTER("_").
    END.
    ELSE
    DO:
        /* Luodaan kysely, jolla haetaan annettu tietue taulusta holleille */
        ASSIGN
            xStatement = "FOR EACH " + fTable + " WHERE " + fTable + "." + pgObjClass.ObjectDesc +  " = " + QUOTER(fPrimaryKey).
    END.

    RUN CreateTable(fTable,pgObjClass.ObjectDesc).
    FOR EACH pgConnec WHERE pgConnec.ObjectClass = pgObjClass.ObjectClass NO-LOCK:
        RUN CreateTable(pgConnec.toObjectClass,pgConnec.ConnDesc).
    END.
END.
RUN RunQuery(fTable,xStatement).
```

Kuva 21. Tarkastetaan, onko käyttäjä antanut tietueelle yhden arvon vai tulostetaan kaikki. Tämän perusteella luodaan kysely (xStatement). Lopuksi luodaan väliaikaistaulut ja ajetaan kysely.

```
/* Proseduuri, jossa ajetaan kysely ja haetaan oikeat tietueet */
PROCEDURE RunQuery:
    DEFINE INPUT PARAMETER iTable       AS CHAR NO-UNDO.
    DEFINE INPUT PARAMETER iStatement   AS CHARACTER NO-UNDO.
    DEFINE VARIABLE xMainKey AS CHAR NO-UNDO.

    IF CAN-FIND(FIRST wStatements WHERE wStatements.TableName = iTable AND wStatements.Statement = iStatement) THEN RETURN.
    CREATE wStatements.
    ASSIGN
        wStatements.TableName = iTable
        wStatements.Statement = iStatement.
    RELEASE wStatements.

    CREATE BUFFER xBuffer FOR TABLE iTable NO-ERROR.

    CREATE QUERY xQuery.

    /* Asetetaan kyselylle bufferi ja jo määritetty kysely */
    xQuery:SET-BUFFERS(xBuffer).
    xQuery:QUERY-PREPARE(iStatement).
    xQuery:QUERY-OPEN.
    /* Haetaan kyselyn ensimmäinen tietue */
    xQuery:GET-FIRST.

    IF xQuery:QUERY-OFF-END THEN
    DO:
        RETURN.
    END.

    QryLoop:
    REPEAT:
        IF xProcedure NE "" THEN RUN VALUE(xProcedure).
        IF iTable = fTable THEN
        DO:
            /* Otetaan taulun pääavaimen arvo talteen */
            FOR FIRST wTables WHERE wTables.TableName = fTable:
                xMainKey = xBuffer:BUFFER-FIELD(wTables.JoinFieldName):BUFFER-VALUE.
            END.
            /* Haetaan yhteystaulun nimi pääavaimen arvo talteen */
            FOR EACH wTables WHERE wTables.TableName <> fTable:
                xValue = xBuffer:BUFFER-FIELD(wTables.JoinFieldName):BUFFER-VALUE.
                xJoinToTable = wTables.TableName.
                /* Tulostetaan arvot */
                DISPLAY
                    xRecordCount VIEW-AS TEXT LABEL "TietueID"
                    xMainKey VIEW-AS TEXT FORMAT "X(20)" LABEL "Pääavain"
                    xJoinToTable VIEW-AS TEXT FORMAT "X(20)" LABEL "Yhteystaulu"
                    xValue VIEW-AS TEXT FORMAT "X(20)" LABEL "Arvo"

                    WITH
                    NO-BOX DOWN WIDTH 118 STREAM-IO USE-TEXT FRAME fRepo.
                RUN RepDispFrame(FRAME fRepo:HANDLE, 0). /* 12.7.2002 Jka XML-reporting 12616 */
                DOWN WITH FRAME fRepo.

            END.
        END.

        RUN CreateDumped(xBuffer).
        ASSIGN
            xRecordCount = xRecordCount + 1.
        xQuery:GET-NEXT (NO-LOCK).
        IF xQuery:QUERY-OFF-END  OR NOT xBuffer:AVAILABLE THEN LEAVE QryLoop.
    END.
    FINALLY:
        IF VALID-HANDLE(xQuery) THEN
        DO:
            IF xQuery:IS-OPEN THEN xQuery:QUERY-CLOSE().
            DELETE OBJECT xQuery.
        END.
        IF VALID-HANDLE(xBuffer) THEN DELETE WIDGET xBuffer.
    END.
END PROCEDURE.
```

Kuva 22. RunQuery-proseduuri, jossa luodaan kysely väliaikaistauluista ja tiedot tulostetaan DISPLAY-komennolla.

```
PROCEDURE CreateDumped:
/*------------------------------------------------------------------------------
 Purpose: Jatkokehityksen dumppiajoa varten luotu proseduuri
 Notes:
------------------------------------------------------------------------------*/
    DEFINE INPUT PARAMETER iBuffer AS HANDLE NO-UNDO.
    IF NOT VALID-HANDLE(iBuffer) OR iBuffer:TYPE <> "BUFFER" THEN RETURN.
    IF CAN-FIND(FIRST wDumped WHERE wDumped.TableName = iBuffer:TABLE AND wDumped.TableRowId = iBuffer:ROWID) THEN RETURN.
    CREATE wDumped.
    ASSIGN
       wDumped.TableName = iBuffer:TABLE
       wDumped.TableRowId = iBuffer:ROWID.
    RELEASE wDumped.
END PROCEDURE.

PROCEDURE CreateTable:
/*------------------------------------------------------------------------------
 Purpose: TEMP-TABLE for collected data.
 Notes:
------------------------------------------------------------------------------*/
    DEFINE INPUT PARAMETER iTable AS CHAR NO-UNDO.
    DEFINE INPUT PARAMETER iJoinFieldName AS CHAR NO-UNDO.
    IF CAN-FIND(FIRST wTables WHERE wTables.TableName = iTable) THEN RETURN.
    CREATE wTables.
    ASSIGN wTables.TableName = iTable
           wTables.JoinFieldName = iJoinFieldName.
    RELEASE wTables.
END PROCEDURE.
```

Kuva 23. Proseduurit CreateDumped ja CreateTable.

CreateTable-proseduuria käytetään luomaan välikaikaistaulut, joista RunQuery-proseduurissa tiedot tulostetaan. CreateDumped-proseduuri on jatkokehitystä varten tehty taulu, jota voidaan käyttää niin sanotun dumppitaulun luontiin.

Muuttujat, jotka otetaan talteen para.w:ltä ja joita halutaan käyttää dumpsche.p-ohjelmassa, on tallennettu repopara.i-includeen.

```
/* repopara.i for dumpsche.p */

DEFINE VARIABLE fPrintId    AS CHARACTER NO-UNDO.
DEFINE VARIABLE fDumpFile1  AS CHARACTER NO-UNDO.
DEFINE VARIABLE fTable      AS CHARACTER NO-UNDO.
DEFINE VARIABLE fPrimaryKey AS CHARACTER NO-UNDO.

RUN RepGetCHAR( INPUT "fPrintId",    OUTPUT fPrintId).
RUN RepGetCHAR( INPUT "fDumpFile1",  OUTPUT fDumpFile1).
RUN RepGetCHAR( INPUT "fTable",      OUTPUT fTable).
RUN RepGetCHAR( INPUT "fPrimaryKey", OUTPUT fPrimaryKey).
```

Kuva 24. Repopapra.i-include.

Tähän includeen on siis tallennettu käyttäjänäkymässä olevat kentät.

6 Tulokset ja pohdinnat

Tässä kappaleessa kerrotaan työn tulokset ja pohdintoja työn sujuvuudesta ja onnistumisesta. Tuloksissa kerrotaan myös työn testaamisesta.

6.1 Tulokset

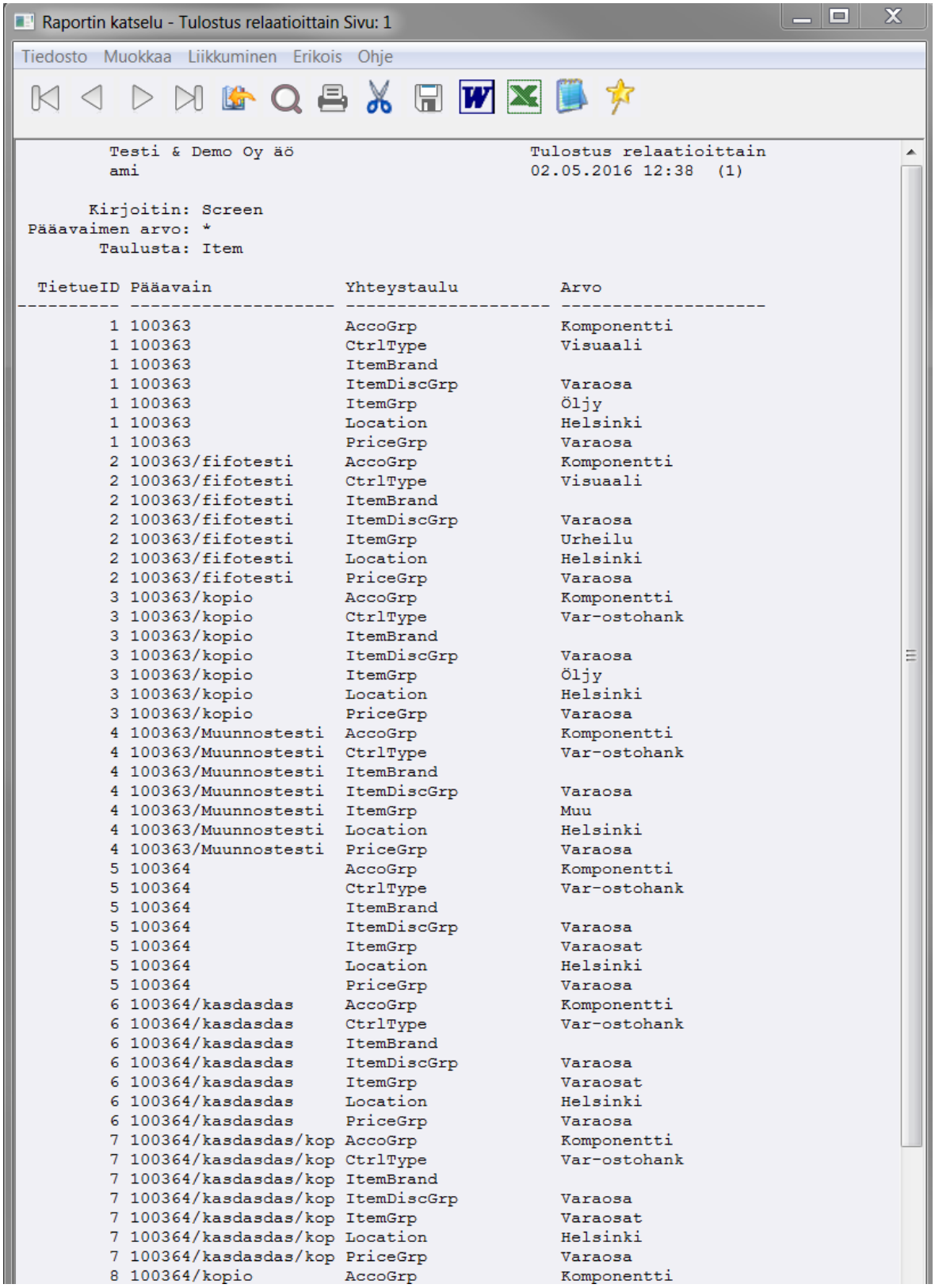
Tietokantarelaatioille löytyi hyvä ratkaisu, miten ne mallinnetaan tietokantaan ja kuinka niitä voidaan hyödyntää. Myös jatkokehitykselle syntyi hyvin paljon ideoita ja monet näistä ovat hyvinkin toteuttamiskelpoisia. Ihan täysin valmis ratkaisu ei kuitenkaan ole, vaan tietokantataulu-ohjelmaa tulee vielä kehittää niin, että sinne saadaan tallennettua tiedot myös niistä tietokantatauluista, joita ei ole normalisoitu ja sisältävät siis pääavaimena pilkkuerotullun listan.

Tulostusohjelma onnistui hyvin ja sillä onnistuu datan tulostus niin näytölle, tiedostoon kuin exceliin. Tätäkin voidaan vielä kehittää jatkossa niin, että saadaan tulostettua pelkän pääavaimen arvon sijaan koko taulun tiedot. Tulostusta on myös mietitty rekursiiviseksi, jotta samaa dataa ei toistuisi moneen kertaan.

Ihan kaikkia virhetilanteita ei tässä työssä otettu huomioon, koska ohjelma tulee pelkästään kehittäjien käyttöön, eikä sitä asiakkaiden ympäristöön edes viedä. Selkeimmistä virhetilanteista kuitenkin annetaan käyttäjälle selkeä viesti.

6.1.1 Testaaminen

Työlle toteutettiin kevennetty testaaminen, jossa todetaan se toimivaksi.
Järjestelmään parametroitiin testidataksi Item-taululle yhteystauluja seitsemän kappaletta. Kun tulostusohjelmalle annettiin tulostusparametreiksi ”Screen”, ”Item” ja joko ”P100” tai ”*”, tulosti ohjelma näkyville oikean määrän dataa.



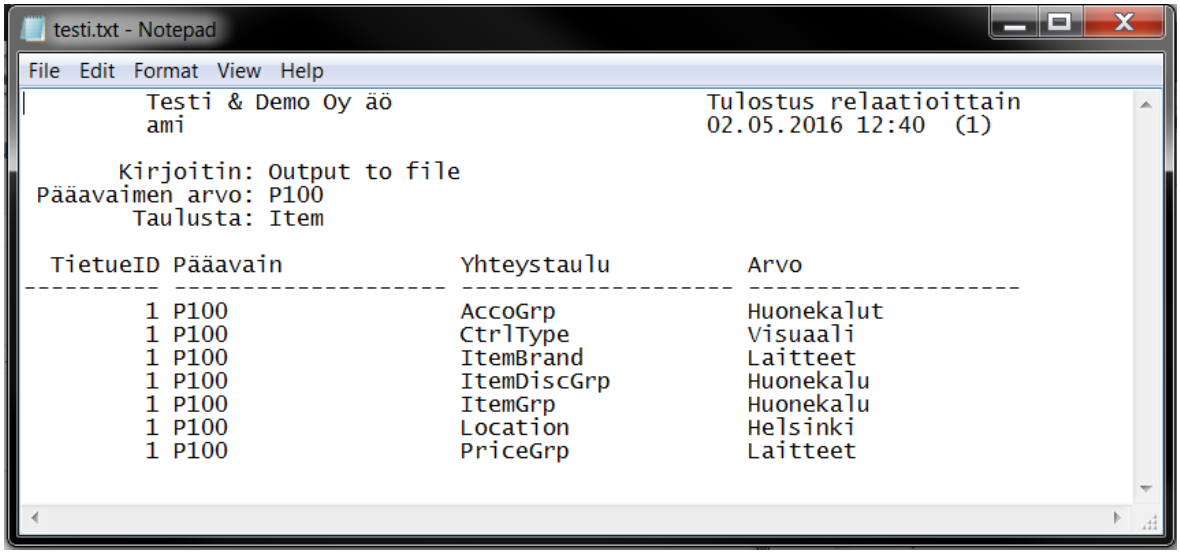
Testi & Demo Oy ää — Tulostus relaatioittain
ami — 02.05.2016 12:38 (1)

Kirjoitin: Screen
Pääavaimen arvo: *
Taulusta: Item

TietueID	Pääavain	Yhteystaulu	Arvo
1	100363	AccoGrp	Komponentti
1	100363	CtrlType	Visuaali
1	100363	ItemBrand	
1	100363	ItemDiscGrp	Varaosa
1	100363	ItemGrp	Öljy
1	100363	Location	Helsinki
1	100363	PriceGrp	Varaosa
2	100363/fifotesti	AccoGrp	Komponentti
2	100363/fifotesti	CtrlType	Visuaali
2	100363/fifotesti	ItemBrand	
2	100363/fifotesti	ItemDiscGrp	Varaosa
2	100363/fifotesti	ItemGrp	Urheilu
2	100363/fifotesti	Location	Helsinki
2	100363/fifotesti	PriceGrp	Varaosa
3	100363/kopio	AccoGrp	Komponentti
3	100363/kopio	CtrlType	Var-ostohank
3	100363/kopio	ItemBrand	
3	100363/kopio	ItemDiscGrp	Varaosa
3	100363/kopio	ItemGrp	Öljy
3	100363/kopio	Location	Helsinki
3	100363/kopio	PriceGrp	Varaosa
4	100363/Muunnostesti	AccoGrp	Komponentti
4	100363/Muunnostesti	CtrlType	Var-ostohank
4	100363/Muunnostesti	ItemBrand	
4	100363/Muunnostesti	ItemDiscGrp	Varaosa
4	100363/Muunnostesti	ItemGrp	Muu
4	100363/Muunnostesti	Location	Helsinki
4	100363/Muunnostesti	PriceGrp	Varaosa
5	100364	AccoGrp	Komponentti
5	100364	CtrlType	Var-ostohank
5	100364	ItemBrand	
5	100364	ItemDiscGrp	Varaosa
5	100364	ItemGrp	Varaosat
5	100364	Location	Helsinki
5	100364	PriceGrp	Varaosa
6	100364/kasdasdas	AccoGrp	Komponentti
6	100364/kasdasdas	CtrlType	Var-ostohank
6	100364/kasdasdas	ItemBrand	
6	100364/kasdasdas	ItemDiscGrp	Varaosa
6	100364/kasdasdas	ItemGrp	Varaosat
6	100364/kasdasdas	Location	Helsinki
6	100364/kasdasdas	PriceGrp	Varaosa
7	100364/kasdasdas/kop	AccoGrp	Komponentti
7	100364/kasdasdas/kop	CtrlType	Var-ostohank
7	100364/kasdasdas/kop	ItemBrand	
7	100364/kasdasdas/kop	ItemDiscGrp	Varaosa
7	100364/kasdasdas/kop	ItemGrp	Varaosat
7	100364/kasdasdas/kop	Location	Helsinki
7	100364/kasdasdas/kop	PriceGrp	Varaosa
8	100364/kopio	AccoGrp	Komponentti

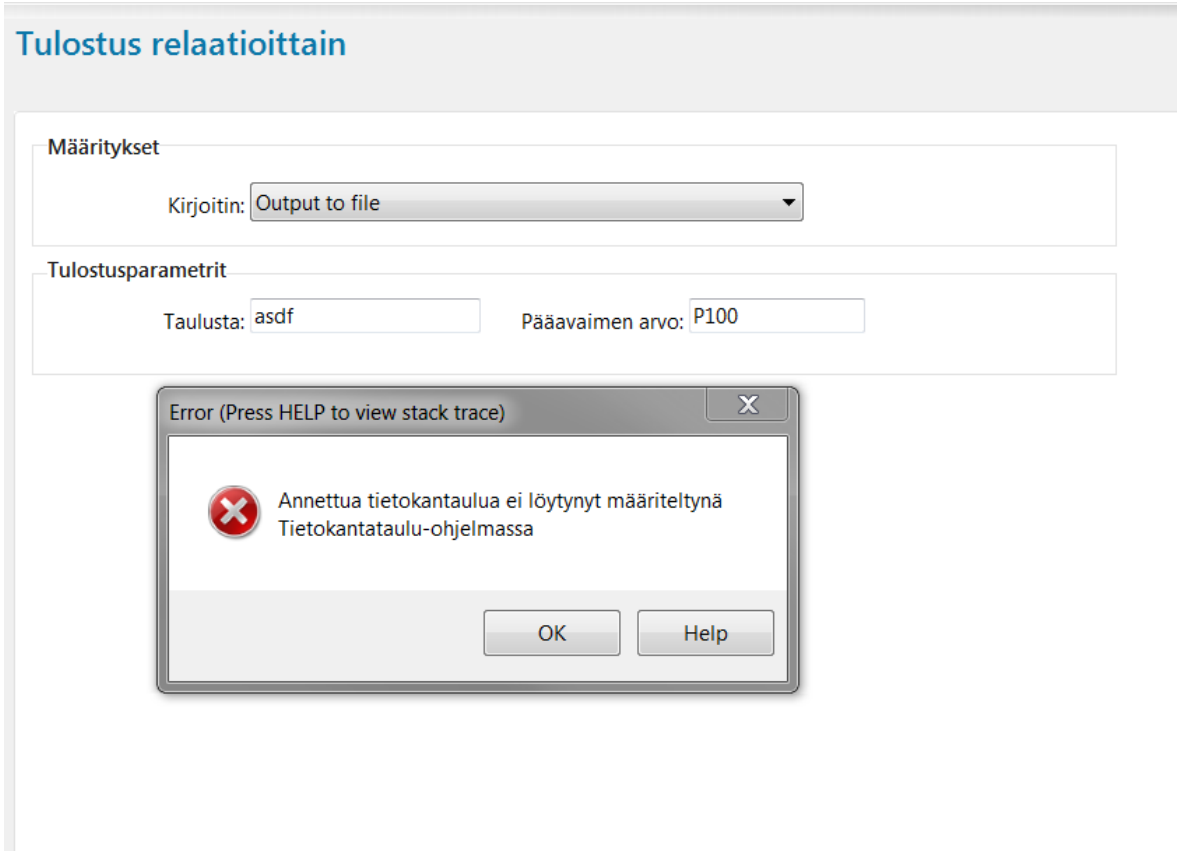
Kuva 25. Tulostus testidatalla.

Ohjelma tulostaa oikein neljä saraketta (tietueid, pääavain, yhteystaulu ja arvo), joihin tieto tulostetaan. Ohjelma käy iteroiden kaikkien yhteystaulujen annettuun pääavaimeen viittaavan tiedon läpi ja tulostaa sen näkyviin.



Kuva 26. Tiedostoon tulostettu, yhden tietueen tulostus.

Tiedostoon tulostaessa tekee ohjelma samannäköisen taulukon kuin näytöllekin. Kuvan 26 tulostuksessa annettiin pääavaimen arvoksi P100, jolloin ohjelma kävi vain Item-taulusta läpi vain tietueen, jonka pääavaimen arvo on P100 ja tulosti siihen liittyvien taulujen pääavaimen arvon.



Kuva 27. Virhetilanne tulostaessa.

Kun yritettiin tulostaa taulusta, jota ei ollut määritelty tietokantataulu-ohjelmaan, antoi järjestelmä selkeän virheen. Pääavaimen arvoon tätä tarkistusta ei ole, joten käyttäjän tulee olla varma, että syöttää arvon kenttään oikein. Tässä tapauksessa ohjelma yrittää tulostaa väärällä arvolla ja se ei löydä mitään tulostettavaa. Lopputuloksena on tiedosto, jossa on vain otsikkorivi.

Lopputulemana testauksesta voidaan todeta, että ohjelma toimi oikein ja antaessa väärää tietoa ohjelmalle, tuli järjestelmästä virheilmoitus tai dataa ei tulostunut.

6.2 Pohdinnat

Tämä opinnäytetyö oli itselleni opettava ja haastava. Työ vei minut aika kauas omasta mukavuusalueesta, joka yleensä on käyttäjärajapinnassa.
Aluksi olin suunnitellut hyvinkin tiukan aikataulun, jonka kuvittelin olevan mahdollinen toteuttaa. Tässä aikataulussa ei ollut edes huomioitu 3 viikon mittaista isyysvapaata, jolloin työtä ei edistetty. Ensimmäinen aikataulu kaatui ja lopullinen työn deadline selvisi vasta aika lopussa työtä, vain muutama viikko ennen palautusta. Työ olisi siis voitu määritellä tarkemmin ja aikatauluun olisi pitänyt kiinnittää enemmän huomiota.
Enterprise ja OpenEdge ABL ovat minulle tuttuja, mutta tiedon ja lähteiden löytyminen (varsinkin ABL:stä) osoittautui haastavaksi.

Työhön sain tarvittaessa hyvin tukea Enteprisen tuotekehityksestä ja omasta tiimistä. Haluankin kiittää kaikkia tämän työn edistämiseen osallistuneita henkilöitä!

Lopputulokseen olen jokseenkin tyytyväinen, vaikka odotankin jatkokehitykseltä enemmän. Dynaamisesti tietokantataulujen yhteyksien mallintaminen osoittautui tämän työn aikana todella haastavaksi ja se jäikin suunnitteluasteelle tässä työssä. Koko datan tulostus yhteystauluista taas on erittäin helppo lisätä tämän työn pohjalta.

Lähteet

Campbell, J. 2004. Progress Software Corporation. Progress Programming Handbook. Luettavissa: http://documentation.progress.com/output/Progress91E/wwhelp/wwhimpl/common/html/wwhelp.htm?context=proghand&file=proghand-01-1.html. Luettu 17.4.2016.

Cengage Learning. Database Elements. Luettavissa: http://www.cengage.com/school/corpview/RegularFeatures/DatabaseTutorial/db_elements/db_elements2.htm. Luettu 16.4.2016.

Digia. Vuosikertomus 2013. Luettavissa: http://vuosikertomus2013.digia.com/filebank/253-Digia_Vuosikertomus_2013.pdf. Luettu 15.4.2016.

Helsingin yliopiston tietojenkäsittelytieteen laitos. Tietokantojen perusteet. Luettavissa: https://www.cs.helsinki.fi/u/laine/tkp/tietomallit/ermalli.html. Luettu 16.4.2016.

Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto. 2004. Normalisointi. Luettavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/normalisointi/. Luettu 19.4.2016.

Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto. Relaatiotietokantojen peruskäsitteet. Luettavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index2.html#TOC3. Luettu 16.4.2016.

Microsoft. 2016. Tietokannan normalisoinnin perusteiden kuvaus. Luettavissa: https://support.microsoft.com/fi-fi/kb/283878. Luettu 18.4.2016.

Oracle. A Relational Database Overview. Luettavissa: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html. Luettu 16.4.2015

Oulun seudun ammattiopisto. Tietokantojen peruskäsitteet. Luettavissa: http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kehittaminen/tiedonhallintajarjestelmat/peruskasitteet.htm. Luettu 16.4.2016.

Sadd, J. 2006. OpenEdge Development: ABL Handbook. Progress Software Corporation.

SQL Destination. 2013. Database normalization and normal forms. Luettavissa: https://sqldestination.wordpress.com/2013/01/03/database-normalization-and-normal-forms/. Luettu 27.4.2016.

Virkki, O. 2008. Haaga-Helia ammattikorkeakoulu. Luento L: Normalisointi. Luettavissa: http://myy.haaga-helia.fi/~ict4td023/ict4td023a/mats/tks_L3_normalisointi.pdf. Luettu 27.4.2016.

Virkki, O. 2012. Haaga-Helia ammattikorkeakoulu. Normaalimuodot. Luettavissa: http://myy.haaga-helia.fi/~ict1tn005/materiaalit/ict05_S3_norm.pdf. Luettu 28.4.2016.

Voelker, A. 2010. Cure for the Plague: A Theoretical Progress / OpenEdge ABL Migration Plan. Luettavissa: https://blog.abevoelker.com/cure_for_the_plague_openedge_migration/. Luettu 28.4.2016.