Raimo Peltonen

RAIPE HENKILÖSTÖHALLINTAOHJELMA

Insinöörityö
Kajaanin ammattikorkeakoulu
Tekniikan ja liikenteen ala
Tietotekniikan koulutusohjelma
Kevät 2005

Kajaanin ammattikorkeakoulu

INSINÖÖRITYÖ
TIIVISTELMÄ

Osasto	Koulutusohjelma
Tekniikka	Tietotekniikka
Tekijä(t) Raimo Peltonen	
Työn nimi RaiPe Henkilöstöhallintaohjelma	
Vaihtoehtoiset ammattiopinnot Konenäkö	Ohjaaja(t) Raili Simanainen
Aika Kevät 2005	Sivumäärä

Tiivistelmä

Insinöörityön tavoitteena oli suunnitella ja toteuttaa Kuljetusliike Veini Peltonen Ky:lle henkilöstönhallintaohjelmisto. Insinöörityön teoriaosuus käsittelee ohjelmistoprojektin etenemistä alkaen suunnittelusta ja päättyen ylläpitovaiheeseen.

Yrityksen henkilöstöhallinta on aikaisemmin hoidettu käsityönä. Nyt toteutetussa ohjelmistossa on Visual Basicilla toteutettu käyttöliittymä. Ohjelmisto tallentaa tiedot Access-tietokantaan, josta niitä voidaan helposti käsitellä ja käyttää yrityksen laskentatoimessa. Ohjelmiston suunnittelussa on käytetty apuna UML-mallinnusta. Ohjelmiston suunnittelussa ja toteutuksessa pyrittiin kiinnittämään huomio ohjelmiston helppokäyttöisyyteen ja toimintavarmuuteen.

Ohjelmisto on ollut käytössä 1.3.2005 alkaen. Ohjelmisto on toiminut vaatimusten mukaisesti. Työn tuloksena saatiin henkilöstöhallintaohjelma, joka on helpottanut henkilöstöhallinnan tehtäviä merkittävästi.

Luottamuksellinen	
Kyllä	
Ei	X
Hakusanat	
Säilytyspaikka Kajaanin AMK:n kirjasto	

Kajaanin ammattikorkeakoulu
Kajaani Polytechnic

ABSTRACT
THESIS

Faculty	Degree programme
Engineering	Information Technology
Author(s): Raimo Peltonen	
Title: The RaiPe Personnel Management Program	
Optional professional studies: Machine Vision	Instructor(s) / Supervisor(s): Raili Simanainen
Date: Spring 2005	Total number of pages

Abstract

The target of this Bachelor's thesis was to design and realize a personnel management program for Kuljetusliike Veini Peltonen Ky. The theory of the thesis discusses the progress of the software project from the design to the maintenance phase.

The personnel management of the company was previously implemented manually. The program which was realized by Visual Basic has a user interface. The program will save the data into the Access database, where it is easily processed and used in the accountancy of the company. UML modeling was used in the design of the program. In the design of the program the ease of use and operational certainty were taken into consideration.

The program has been in operation since 1 March 2005. So far the program has met the requirements. The result of this thesis is a personnel management program which has made personnel management significantly easier.

Confidential
Yes
No X

Keywords
Personnel management, Visual Basic, user interface, Access database, UML modelling

Deposited at
The Library of Kajaani Polytechnic

SÄLLYSLUETTELO

KÄYTETYT TERMIT

ADO.NET	Microsoftin tietokannoille kehittämä tiedonkäsittelymalli.
Funktionaalinen riippuvuus	(functional dependency) Kenttä A on funktionaalisesti riippuvainen kentästä B, jos A:ta kohti on korkeintaan yksi B:n arvo kunakin ajanhetkenä. Käytetään normalisoinnissa. [1, s.337]
Relaatio	(relation) Relaatiomallin termi taululle; rakenteeltaan samanlaisten monikoiden joukko [1, s. 344]
Relaatiotietokanta	(relational database) 1. Relaatiomallia noudattava, tietokannan hallintajärjestelmän avulla muodostettu tietokanta. 2. Relationaalinen tietokannan hallintajärjestelmä, relaatiotietokantatuote (esim. Oracle, DB2). [1, s. 344]
SQL	(Structured Querry Language) Lähes kaikkien relaatiotietokantatuotteiden tukema ja tietokannan määrittely- ja käsittelykieli [1, s.345].
UML	(Unified Modeling Language) Oliomallintamisessa käytetty standardinotaatio [1, s. 347].

1 JOHDANTO

Kuljetusliike Veini Peltonen KY on pieni perheyritys, jonka päätoimialona on puutavaran ja maa-ainesten kuljettaminen. Yritys on henkilöstön kasvaessa etsinyt henkilöstöhallintonsa hoitamiseen sopivaa ohjelmistoa. Ohjelmiston tulisi olla yksinkertainen ja helppohoitoinen. Koska pienen yrityksen henkilöstönhallinta ei työllistä ketään täysipäiväisesti, täytyy henkilöhallinnasta vastaavan henkilön pystyä hoitamaan kyseinen työ ohjelmiston avulla muiden töidensä ohessa. Lisäksi ohjelmiston tulisi antaa tietoa kustannusseurantaan. Esimerkiksi autojen palkkamenot tulee saada ulos vaikka päivittäin, koska on tärkeää tietää, mitä jonkin palvelun tuottaminen todella maksaa.

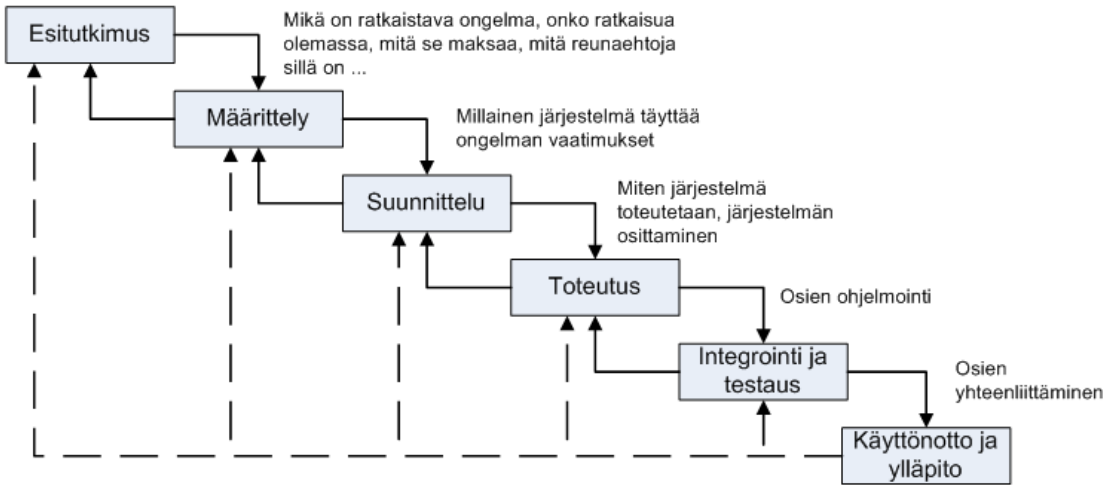
Markkinoilta ei ole löytynyt tällaista ohjelmistoa. Ohjelmistot ovat olleet joko liian kalliita hankittavaksi tai eivät ole antaneet riittävää tietoa. Jotkut ohjelmistot ovat olleet liian monimutkaisia ja hankalia käyttää. Lisäksi jos henkilöstöhallinnan palkanlaskenta annettaisiin tilitoimistolle hoidettavaksi, tulisi se liian kalliiksi ja heikentäisi näin kilpailukykyä.

Suunnitellun ja toteutetun ohjelmiston tuli täyttää yrityksen antamat vaatimukset, joita olivat ohjelman keveys, muuteltavuus, helppohoitoisuus ja kustannusseurantaan soveltuva raportointi. Ohjelmisto päätettiin toteuttaa relaatiotietokantana, jolloin ohjelmiston ylläpidettävyys helpottuu ja ohjelmistosta saa helposti yrityksen haluamia tietoja. Samoin ohjelmiston päivitettävyys on helpompaa verrattuna tavalliseen tietokantaan.

Insinöörityöhöni kuului suunnitella ja toteuttaa tällainen henkilöstöhallinnanohjelmisto. Tässä työssä käytettiin seuraavia ohjelmia: tietokannan luomiseen Microsoft Access 2003 -ohjelmistoa, tietokannan ylläpitämiseen Visual Basic .NET –kehitysjärjestelmää. Ohjelmisto suunniteltiin UML:ää apuna käyttäen. Näin ohjelmistosta tuli johdonmukainen ja tehdynnäköinen.

2 OHJELMISTON KEHITYSPROSESSI

Haikalan mukaan ohjelmiston elinkaarella tarkoitetaan aikaa, joka kuluu ohjelmiston kehittämisen aloittamisesta sen poistamiseen käytöstä. Vaihejakomallilla tarkoitetaan tapaa, jolla ohjelmiston kehitystyö tai koko elinkaari jaetaan vaiheisiin. Tavallisin vaihejakomalli on vesiputousmalli, jonka eräs versio on esitetty kuvassa 1. Käytännössä projektit eivät etene suoraan vesiputousmallin mukaisesti, koska ohjelmiston vaatimukset selviävät lopullisesti vasta projektin aikana. [2, s. 36.]



Kuva 1. Esimerkki vesiputousmallista[2, s.36]

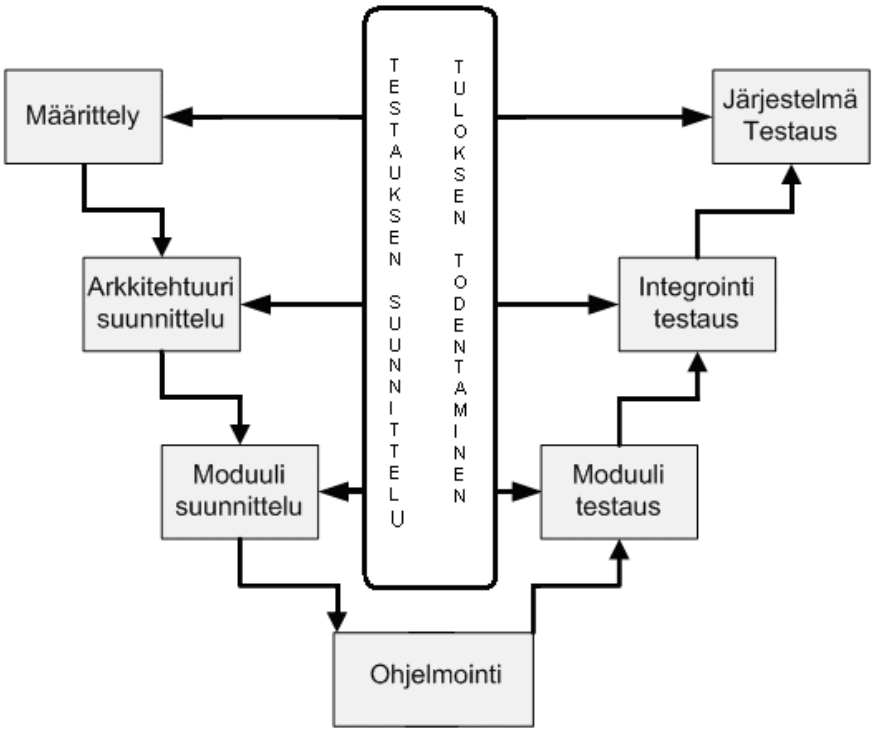
Esitutkimuksen tehtävänä on asettaa järjestelmätason vaatimukset. Tällaisia vaatimuksia kutsutaan usein asiakasvaatimuksiksi, koska ne määrittävät asiakkaan tarpeet, mutta eivät mitenkään ota kantaa siihen, millainen järjestelmä täyttää asiakkaan vaatimukset. Esitutkimus vastaa kysymykseen, miksi ohjelmisto tai järjestelmä tulisi tehdä tai miksi sitä ei kannata tehdä. Esitutkimus on siinä mielessä elinkaaren tärkein vaihe, että vääristä asiakasvaatimuksista ei voi päätyä hyvään järjestelmään. Sen suurin ongelma on asiakkaan todellisten tarpeiden selville saaminen ja perusteellinen ymmärtäminen. Esitutkimus usein ymmärretään myös osaksi määrittelyvaihetta, koska käytännössä asiakastarpeiden analysointi ja tarkentaminen yleensä jatkuu koko määrittelyvaiheen ajan. [2, s.37.]

Määrittelyvaiheessa analysoidaan asiakasvaatimuksia. Niistä laaditaan ohjelmistovaatimukset, jotka määrittelevät toteutettavan järjestelmän. Tuloksena syntynyttä dokumenttia kutsutaan toiminnalliseksi määrittelyksi. Dokumentissa kuvataan ohjelmiston toiminnalliset ja ei-toiminnalliset vaatimukset ja rajoitukset. Määrittelyvaihe vastaa kysymykseen, mitä järjestelmä tekee.

Suunnitteluvaiheessa suunnitellaan määrittelyn kuvaamat toiminnot. Suunnittelun tuloksena syntyy dokumentti tekninen suunnitelma. Suunnitteluvaihe osioidaan usein 2:teen tasoon. Ensimmäisessä osiossa suunnitellaan ohjelmiston arkkitehtuuri ja jaetaan ohjelmisto moduuleihin. Toisessa osiossa suunnitellaan moduulien sisäinen rakenne. Suunnittelu vaihe vastaa kysymykseen, miten tehtävät hoidetaan.

Ohjelmointivaihe sisältää toiminnot ohjelman kirjoittamisen aloittamisesta aina ensimmäiseen virheettömään käännökseen asti.

Testausvaiheessa etsitään ohjelmiston virheitä [2, s.40]. Testaus suoritetaan useassa eri tasossa ja yleisin testaustapa on V-malli. V-mallin mukaisesti erilaisia testaustasoja ovat moduuli-, integrointi- ja järjestelmätestaus. Moduulitestauksessa etsitään virheitä ohjelman moduuleista, integrointitestauksessa moduuleiden yhteyksistä ja järjestelmätestauksessa koko järjestelmän toiminnasta ja suorituskyvystä [2, s.40]. Testauksen suunnittelu tapahtuu vastaavalla ohjelmiston suunnittelutasolla. Kuvassa 2 on esitetty V-mallin mukainen testaussuunnittelu ja tulosten todentaminen. Järjestelmätestaus suunnitellaan määrittelyvaiheessa, integrointitestaus arkkitehtuurisuunnitteluvaiheessa ja moduulitestaus moduulisuunnitteluvaiheessa [2, s.286].



Kuva 2. Testauksen V-malli

Moduulitestauksessa on testattavana yksittäinen moduuli. Moduulin toimintaa verrataan tekniseen määrittelydokumenttiin. Testauksen suorittaa moduulin toteuttaja. Integrointitestauksessa on testattavana yhdistettyjä moduuleita tai moduuliryhmiä. Testauksen pääpaino on moduulien rajapintojen toimivuuden toimivuudessa. Tuloksia verrataan myös tekniseen määrittelydokumenttiin. Integrointitestaus etenee usein rinnan moduulitestauksen kanssa. Järjestelmätestauksessa testattavana on koko järjestelmä ja tuloksia verrataan määrittelydokumentaatioon. Järjestelmätestaukseen voi liittyä kenttätestaus ja hyväksymistestaus. Järjestelmätestauksen suorittajan tulisi olla ohjelmiston kehitystyöhön kuulumaton henkilö. [2, s.287.]

Käyttöönotto- ja ylläpitovaiheet ovat, varsinkin asiakasprojekteissa, tärkeitä ja keskeisiä vaiheita, koska asiakas on se taho, joka tuo pelimerkit taloon. Tämän vuoksi tämä vaihe kannattaa suunnitella ja toteuttaa kunnolla. Käyttöönotto- ja ylläpitovaiheessa ratkotaan asiakkaiden ohjelmistoon liittyviä ongelmia, korjataan virheitä ja päivitetään ohjelmisto uusien vaatimusten mukaiseksi.

Ohjelmistotuotteilla ei yleensä ole varsinaista ylläpitovaihetta, koska muutokset, korjaukset ja lisäykset toteutetaan yleensä uudessa omassa projektissa.

Ohjelmointivälineiden käyttö ohjelmistoprojektissa

UML

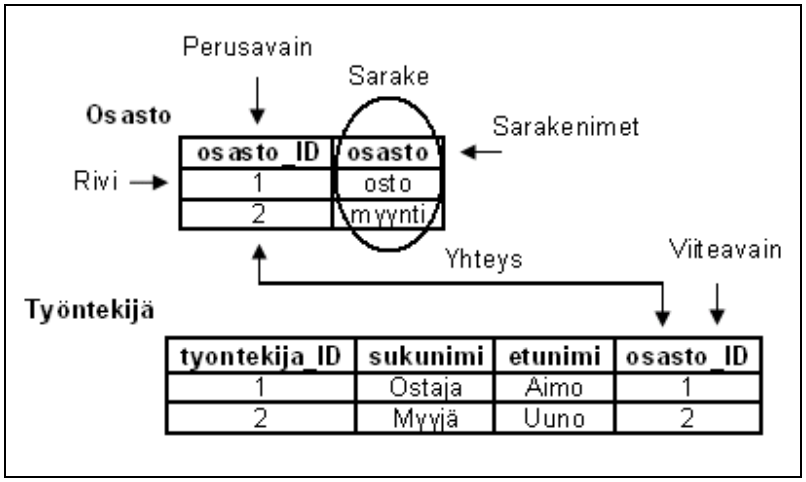
UML (Unifield Modeling Language) on graafinen oliopohjainen mallinnuskieli, jonka avulla voidaan mallintaa tapahtumien kulku toimintojen eri vaiheissa. UML sisältää mallinnuselementit, notaatiot ja yleisohjeet, suositukset, kuinka notaatioita tulee käyttää. UML:n kehitys on alkanut 1996, ja siihen on yhdistetty kolme johtavaa mallinnustekniikkaa, OMT (Rumbaugh), Booch (Booch) ja OOSE (Jacobson). Nykyinen standardi on vuodelta 2001, UML 1.4. UML:n käyttötarkoituksia on ohjelmistosuunnittelu, ajettavan koodin luominen, järjestelmien arkkitehtuuri, prosessitekniikka, Internet-portaalien rakenteet, työvirtojen kuvaus ja liiketoimintaprosessien mallintaminen[3].

UML:n kuuluu erilaisia kaavioita, joiden avulla järjestelmää mallinnetaan. Vuorovaikutuskaaviolla kuvataan, kuinka eri objektit toimivat yhteistyössä. Sekvenssikaavio kuvaa toimintojen tapahtumisjärjestystä. Yhteistyökaavio kuvaa objektien välistä vuorovaikutussuhdetta. Luokkakaavio kuvaa järjestelmän rakennetta. Luokkakaaviosta on olemassa erilaisia versioita tarpeen mukaan. Tilakaaviolla kuvataan objektin muuttumista eri tilojen kautta lopputilaan. Vuokaavio on samankaltainen kuin tilakaavio. Komponenttikaavio kuvaa järjestelmän komponentit. Sijoittelukaavio kuvaa, kuinka eri komponentit sijoittuvat järjestelmään.

Relaatiotietokanta

Relaatiotietokannat perustuvat IBM:n tutkija E. F. Coddin v. 1970 julkaisemaan relaatiomalliin (the relational model), joka määrittelee relaatiotietokannan teoreettisen pohjan. Relaatiomalli perustuu joukko-oppiin, matematiikkaan ja predikaattilogiikkaan. Relaatiomalli ei ota kantaa relaatiokannan fyysiseen toteutustapaan, vaan se on jätetty tietokantatuotteiden toimittajille.[1, s. 7.]

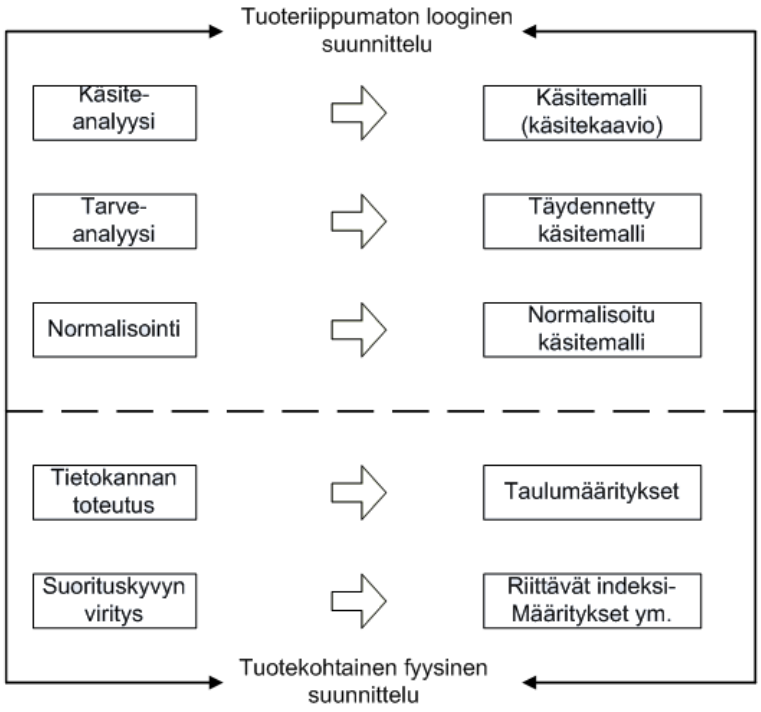
Relaatiokantojen tietomalli on relaatiomalli, jossa on vain kahdenlaisia objekteja: tauluja ja tietoja [1 s.104]. Relaatiomallin mukaisesti relaatiotietokanta rakentuu tauluista. Taulu koostuu sarakkeista ja riveistä. Yksi rivi sisältää yhden tietojoukon. Yksi taulun sarakkeesta on asetettu perusavaimeksi, joka yksilöi taulun rivin muista saman taulun riveistä. Taulu voi sisältää lisäksi viiteavaimen tai –avaimia, joiden avulla yhdistetään eri taulujen tietoja toisiinsa. Kuvassa 3 on esitetty kaksi taulua, Osasto ja Työntekijä, jotka liittyvät toisiinsa. Kuva selventää käsitteiden ymmärtämistä.



Kuva 3. Esimerkki kahdesta taulusta.

Relaatiomalli ottaa kantaa myös tietokannan eheyteen (integrity). Tietokanta on eheä, kun sen tiedot ovat oikein, ristiriidattomia ja vastaavat reaalimaailmaa [1 s. 11]. Relaatiotietokannassa noudatetaan avaineheys- ja viite-eheyssääntöä. Avaineheyssääntö kieltää perusavaimen tyhjän arvon. Viite-eheyssääntö estää orpojen tietojen esiintymisen tietokannassa. Lisäksi tarkastellaan tietojen oikeellisuutta ja sisäisen tiedon ristiriidattomuutta. Tällöin asioilla on vain yksi oikea tieto.

Tietokannan suunnittelu koostuu tietyistä vaiheista, jotka tavalla tai toisella on tehtävä kaikissa tietokantojen suunnitteluhankkeissa [1, s.24]. Tätä vaihejakomallia voidaan kutsua suunnitteluputkeksi. Kuvassa 4 on esitetty eräs suunnitteluputkimalli.



Kuva 4. Tietokantojen suunnitteluputki[1, s.24]

Käsiteanalyysissä kuvataan sitä reaalimaailmasta rajattua osaa eli kohdealuetta, joka on tarkoitus kuvata tietokannassa [1, s.32]. Käsiteanalyysin tuloksena saadaan ER-kaavio, jota kutsutaan käsitemalliksi, ja kuvattavia asioita ovat käsitteet, tiedot ja yhteydet. Käsitemalli on tulevan tietokannan runko. Käsitemallia pöytätestataan ja täydennetään, eli suoritetaan tarveanalyysi. Tämän jälkeen lopputuloksena on pöytätestattu ja toimivaksi todistettu tietokantamalli. Käsitemallissa on kolmenlaisia objekteja: käsite, tieto ja yhteys. Käsite kuvaa asiaa, jonka tietoja aiotaan säilyttää tietokannassa. Tieto kuvaa käsitettä, ja yhteys puolestaan kertoo käsitteiden väliset riippuvuudet.

Yhteysmalleja on yksi-yhteen, yksi-moneen ja moni-moneen. Yksi-moneen-yhteyttä kutsutaan tietolähteestä riippuen isä-lapsi- tai äiti-lapsi-yhteydeksi. Yhteyttä voidaan kuvata verbillä tekee, kuuluu tai jakaantuu. Esimerkkinä yksi-moneen-yhteydestä voidaan pitää työntekijää ja yritystä. Työntekijällä on yrityksessä yhdet henkilötiedot, mutta yrityksellä on monet eri työntekijätiedot. Moni-moneen-yhteyden esimerkkinä voidaan pitää henkilöä ja puhelinyhteyksiä. Henkilöllä voi olla monta puhelinyhteyttä, ja yhtä puhelinyhteyttä voi käyttää moni henkilö. Yksi-yhteen-yhteydet ovat harvinaisempia. Tällaisesta esimerkki

voi olla henkilö ja sosiaaliturvatunnus. Henkilöllä voi olla yksi sosiaaliturvatunnus ja sosiaaliturvatunnuksella voi olla vain yksi henkilö. Yhteyksiä tulee katsoa molempiin suuntiin, jotta nähdään, millainen yhteys on todella käsiteltävänä.

Relaatiotietokanta pyritään normalisoimaan kolmanteen normaalimuotoon. Normalisointi lisää yleensä taulujen määrää. Normalisoinnilla saavutetaan tietorakenteen joustavuus, jolloin rakennetta ei tarvitse aina muuttaa, kun lisätään tietoa. Normalisoinnilla haetaan tietokantaa, jossa tietojen toistaminen on minimoitu, päivitykset ovat tehokkaita ja rakenne on yhdenmukainen ja muutosjoustava.

Ensimmäisen normaalimuodon sääntö kuuluu: Poista toistuvat ryhmät ja moniarvoiset kentät [1 s.88]. Kuvassa 5 esitetyssä taulussa on toistuva ryhmä puhelinnumero.

Henkilö					
ID_Henkilo	Sukunimi	Etunimi	puhelin1	puhelin2	puhelin3
1	Puhelias	Veikko	123456	234567	345678
2	Hiljanen	Unto	97654		

Kuva 5. Henkilötaulussa esiintyvä toistuvaryhmä

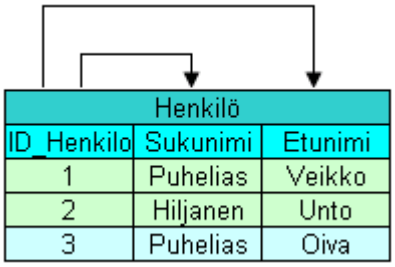
Taulu saadaan ensimmäiseen normaalimuotoon jakamalla taulu kahteen osaan. Ensimmäiseen osaan laitetaan henkilötiedot ja toiseen osaan puhelinnumerot. Muutokset on esitetty kuvassa 6. Henkilön puhelinnumero haetaan henkilötunnuksella ja puhelintunnuksella.

Henkilö		
ID_Henkilo	Sukunimi	Etunimi
1	Puhelias	Veikko
2	Hiljanen	Unto

Puhelin		
ID_puhelin	ID_Henkilo	puhnro
1	1	123456
2	1	234567
3	1	345678
4	2	97654

Kuva 6. Ensimmäisen normaalimuodon mukaiset taulut

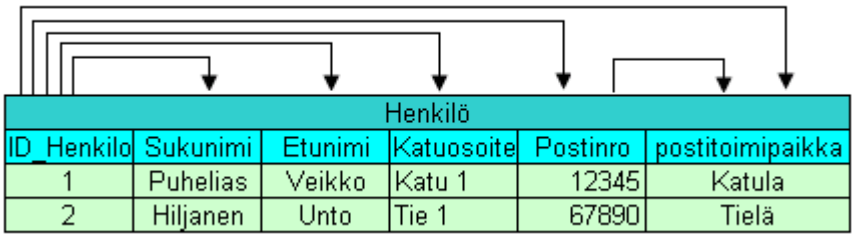
Toisen normaalimuodon sääntö kuuluu: Jos taulussa on moniosainen perusavain, niin kaikkien sarakkeiden tulee olla funktionaalisesti riippuvia koko perusavaimesta (eikä siis vain osa-avaimesta)[1, s.91]. Tutkimalla kuvan 7 henkilötaulua voidaan selventää käsitettä funktionaalinen. Sukunimi on funktionaalisesti riippuvainen ID_henkilöstä, koska jokaista henkilötunnusta kohden on yksi sukunimi. ID_henkilö ei taas ole funktionaalisesti riippuvainen sukunimestä, koska sukunimeä kohti voi olla useita ID_henkilötunnuksia. Kuvassa oleva nuoli luetaan seuraavasti: Henkilötunnusta kohti on nolla tai yksi sukunimi ja vastaavasti etunimi.



Henkilö		
ID_Henkilo	Sukunimi	Etunimi
1	Puhelias	Veikko
2	Hiljanen	Unto
3	Puhelias	Oiva

Kuva 7. Funktionaalinen riippuvuus

Kolmannen normaalimuodon sääntö kuuluu: Jokaisen sarakkeen pitää olla funktionaalisesti riippuvainen vain perusavaimesta, eikä siis mistään tavallisesta sarakkeesta [1, s. 93]. Kuvassa 8 on esitetty henkilötietojen taulu. Taulussa esiintyy 2 riippuvuussuhdetta. Ensimmäinen funktionaalinen riippuvuus on henkilötunnuksen ja kaikkien muiden sarakkeiden välillä. Toinen funktionaalinen riippuvuus on postinumeron ja postitoimipaikan välillä.



Henkilö					
ID_Henkilo	Sukunimi	Etunimi	Katuosoite	Postinro	postitoimipaikka
1	Puhelias	Veikko	Katu 1	12345	Katula
2	Hiljanen	Unto	Tie 1	67890	Tielä

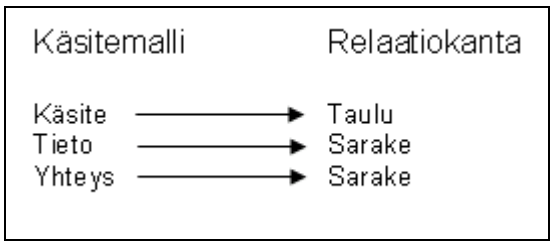
Kuva 8. Taulun sisäisiä riippuvaisuuksia

Tämän riippuvuuden voi poistaa tekemällä uuden taulun posti. Sen perusavaimeksi laitetaan postinumero ja normaaliksi sarakkeeksi postitoimipaikka.

Näin tietokantaan voi ladata etukäteen postitaulun, jossa on valmiina postinumerot ja postitoimipaikat. Henkilötauluun jää viiteavaimeksi postinumero, jonka avulla löydetään postitaulusta postitoimipaikka.

Tietokannan voisi vielä denormalisoida. Tämä tarkoittaa peruuttamista kolmannesta normaalimuodosta. Näin menetetään normalisoinnin etuna saavutettu tietojen toiston minimointi, mutta etuna saavutetaan tietokannasta hakujen nopeutuminen ja virhemahdollisuuksien väheneminen. Nopeus saadaan varsinkin, jos tietoja joudutaan lukemaan enemmän levyltä. Taulujen määrän vähentäminen vähentää hakuja yhdistellessä taulujen liitoksista aiheutuvaa virhe mahdollisuutta.

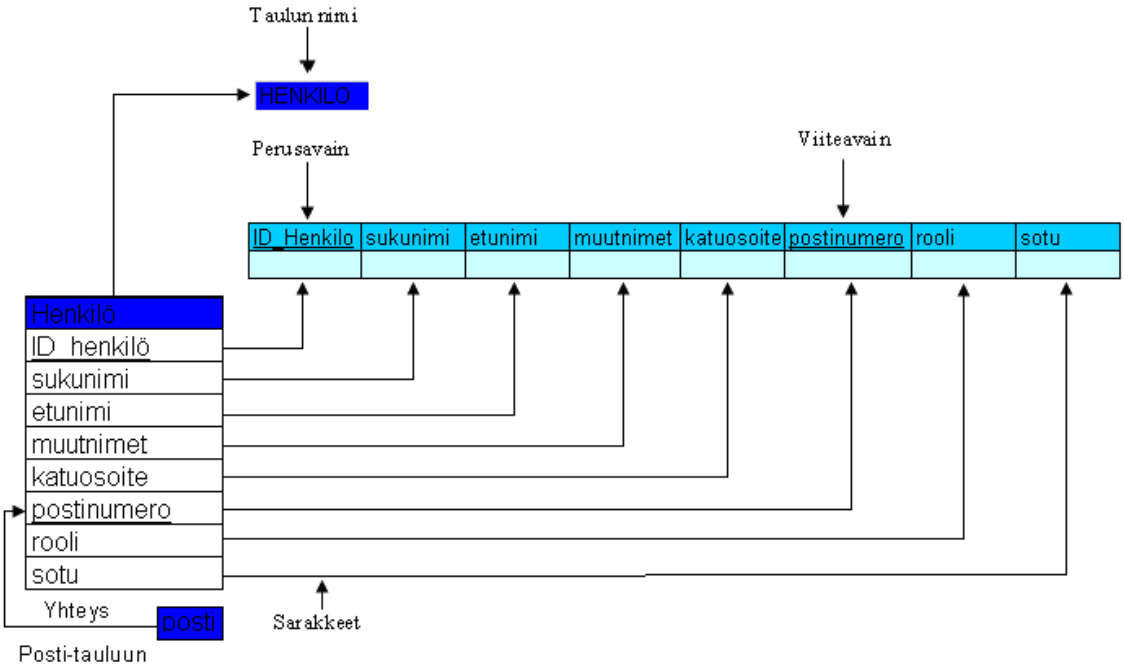
Normalisoinnin jälkeen täydennetyn käsitemallin käsitteestä muodostetaan tietokannan taulu, tiedosta muodostetaan sarake ja yhteydestä muodostetaan sarake, joka toimii viiteavaimena. Kuvassa 9 on esitetty käsitemallin objektien muuttuminen relaatiokannan objekteiksi.



Kuva 9. Käsitemallista relaatiokantaan[1, s. 104]

Tauluja muodostettaessa aluksi luodaan jokaisesta käsitteestä relaatiokannan taulu. Taulun nimi kannattaa harkita huolella, koska myöhemmässä vaiheessa taulun nimen vaihtaminen on vaikeaa melkein mahdotonta. Toisessa vaiheessa luodaan käsitteiden tiedoista taulujen sarakkeita. Perusavaimet on määritelty aiemmin tarveanalyysin teon aikana. Yksi-moneen-yhteydessä isä-taulun perusavain tulee lapsi-tauluun viiteavaimeksi. Yksi-yhteen-yhteydessä laitetaan viiteavain siihen tauluun, joka on pakollinen [1, s.106]. Sarakkeiden nimet kannattaa harkita huolella, koska nimien vaihto jälkeenpäin on vaikeaa. Kolmannessa vaiheessa määritetään tietotyypit ja eheysehdot. Eheysehdot estävät tietokannasta isä-taulun tietojen poiston, jos lapsi-taulussa on vielä kyseiseen tietoon liittyvää

tietoa. Kuvassa 10 on esitetty Henkilö-käsitteen muuttuminen relaatiotauluksi. Relaatiotauluun tulee perusavaimeksi ID_henkilo ja viiteavaimeksi postinumero. Molemmat on alleviivattu kuvassa. Lopuista tiedoista tulee vain normaaleja taulun sarakkeita, joihin tieto tallennetaan ja joista tieto haetaan tarvittaessa.



Kuva 10. Käsitteen muuttaminen relaatiotauluksi

Kun tietokantaan on luotu taulujen väliset yhteydet, voidaan tietokanta vielä virittää toimimaan mahdollisimman nopeasti ja joustavasti. Viritys suunnitellaan tietokannan suunnittelun yhteydessä. Virityksessä voidaan vaikuttaa indeksiin, taulun rivien järjestykseen, uudelleenjärjestykseen, sovelluksen rakenteeseen, SQL-käskyjen rakenteeseen, tietokannan parametreihin, denormalisointiin, summaamiseen ja johdettuihin tietoihin, laiteratkaisuihin, lukko-odotusten minimointiin ja priorisointiin. Yleensä indeksien suunnittelulla saavutetaan riittävän hyvät tulokset, jolloin muita virittämisiä ei enää tarvita. Indeksien parantelun verraton etu on se, ettei ohjelmiin tarvitse koskea [1, s.144].

Microsoft Visual Basic .NET

Visual Basic .NET –ohjelmien kirjoittamiseen käytetään Microsoft Visual Studio .NET -kehitysympäristöä. Microsoft Visual Basic .NET on uusi päivitys ja laajennus Visual Basic –kehitysjärjestelmään. Tämän kehitysjärjestelmän suurimpia etuja on tietokantojen ja Internet ratkaisujen helppo tuottaminen. Visual Basic .NET jakaa kehitysympäristön Visual C++ .NETin kanssa, mutta ne ovat edelleen eri ohjelmointikieliä. Visual Basic .Net on täydellinen olio-ohjelmointikieli, joka tarjoaa mahdollisuuden käyttää kaikkia .NET -Frameworkpalveluita. Kielen syntaksi on säilynyt selkeänä, mutta ohjelmointimalli on siirtynyt proseduraalisesta ohjelmoinnista olio-ohjelmointiin. Visual Basic .NETiä ei ole suunniteltu tietokantojen luomiseen vaan niiden sisällön tuottamiseen ja käsittelemiseen [4, s.499]. Visual Basic .Net tukee kaikkia yleisimpiä tietokanta-ohjelmia. Visual Basic .NET käyttää tietomallia nimeltään ADO.NET' ja sen sisäinen tietomuoto on XML-standardi, jonka on kehittänyt Word Wide Web Consortium.

Michael Halvorsonin mukaan[4, s.498] Visual Basic .NET mahdollistaa nopeasti ja helposti muodostettavat yhteydet relaatiotietokannan ja käyttöliittymän välille. Tämä on seurausta siitä, koska Visual Basic .NET on suunniteltu tietokantojen mukautettujen käyttöliittymien suunnittelemiseen. Visual Basic .NET käyttää tiedonkäsittelymallina ADO.NET -mallia. Tämä sisältää laajat valikoimat tiedonkäsittelytoimintoja ja perustuu Microsoftin tekniikkaan nimeltään ADO+. ADO.NET on vakiotietomalli kaikille Visual Studio .NETin alaisuudessa tehtäville ohjelmille. Visual Basic .NETissä tietokannan tietojoukon sisältöä edustaa tietojoukko-objekti, dataset, joka on käsiteltävän tietokannan itsenäinen kopio.

Visual Basic -ohjelmassa voi esiintyä kolmenlaisia virheitä: syntaksi-, ajonaikaisia ja loogisia virheitä. Syntaksivirhe on ohjelmointivirhe, joka rikkoo Visual Basicin sääntöjä. Visual Basic havaitsee monenlaisia syntaksivirheitä jo ohjelmalauseita kirjoittaessa. Se ei anna käynnistää ohjelmaa ennen kuin virheet on korjattu. Ajonaikainen virhe on virhe, joka aiheuttaa ohjelman suorittamisen keskeytymisen. Se aiheutuu esimerkiksi ulkoisesta tapahtumasta tai havaitse-

matta jääneestä syntaksivirheestä. Looginen virhe on inhimillinen erehdys eli ohjelmointivirhe, joka saa ohjelman tuottamaan vääriä tuloksia. [4, s. 216.]

Microsoft Access 2003

Tietokantasovellusten tuottamiseen Access on helppokäyttöinen. Access käyttää ohjelmointikielenään Visual Basicia, jota käytetään muissakin Microsoft Office -sovelluksissa. Access sisältää joukon ohjattuja toimintoja, joiden avulla tietokannan luominen on helppoa ja nopeaa. Tämä helppous voi kostautua tuottamaan ensin tietokantasovellus ja suunnittelemaan se vasta tuottamisen jälkeen. Access ei ole aito palvelinpohjainen tietokannan hallintajärjestelmä vaan tiedostopohjainen ns. Desktop-kanta [5, s. 5]. Tämän vuoksi se ei sovi laajempiin monen käyttäjän sovelluksiin.

SQL

SQL ei ole pelkästään kyselykieli, vaan se kattaa tietokannan rakenteen määrittelyt, kyselyt, päivitykset lisäyksistä poistoihin, tapahtumien ohjaamisen, valtuuksien ja turvallisuuden hoidon, upotettu SQL ja kohdistimien hallinta sekä API-rajapinnat ohjelmointikieliin [5, s. 14]. SQL on ei-proseduraalinen kieli. SQL:llä ei kerrota, miten tietokantaoperaatiot tehdään, vaan mitä halutaan tehdä. SQL-kysely tuo vastaukseksi yleensä joukon tietueita, kun taas proseduraaliset kielet käsittelevät tietoja tietue kerrallaan. SQL:llä on tärkeä rooli asiakas-palvelinjärjestelmissä. Asiakkaiden työasemilla sijaitsevat ohjelmat kyselevät SQL-lauseilla palvelimilla olevilta tietokanoilta tietoja. Tietokannassa tapahtuu tietojen luku ja lähetys takaisin asiakaspäätteelle tai tietojen päivitys tietokantaan.

3 OHJELMISTON VAATIMUSTEN MÄÄRITTELY

Työn tilaajan kanssa sovittiin, että Henkilöstöhallintaohjelma toimii Kuljetusliike Veini Peltonen Ky:n henkilöstöhallinnan apuvälineenä. Ohjelmiston avulla ylläpidetään kirjaa henkilöstön henkilötiedoista, palkoista, työsuhteen kestosta ja yleensä kaikista henkilöstöhallintoon liittyvistä asioista. Ohjelmistoon syötetään henkilötiedot, palkan perusteet ja työaika. Ohjelmisto laskee ja tulostaa henkilön palkan, palkkayhteenvedon tiettynä ajankohtana esim. vuosittain, työajan kertymän tiettynä ajankohtana esim. vuosittain ja ohjelman tietoja voi käyttää apuna yrityksen kustannusseurannassa. Ohjelmiston käytön tulee olla selkeää ja helppoa. Ohjelmistoon tulee pystyä syöttämään tiedot vaivattomasti. Ohjelmiston tulee antaa selkeitä tulosteita. Tulosteet tulee voida katsoa näytöltä ennen tulostamista. Kaikkia tulosteita ei tarvitse pystyä tulostamaan suoraan paperille, koska yrityksen laatujärjestelmä edellyttää paperitulosteiden vähentämistä. Ohjelmistoa tulee voida käyttää vaivattomasti pienen koulutuksen jälkeen.

Ohjelmistossa tulee olla seuraavat toiminnot:

1. Ohjelmisto ylläpitää henkilötietoja.
2. Ohjelmisto laskee työajan.
3. Ohjelmisto ylläpitää työaikakertymää.
4. Ohjelmisto laskee palkan.
5. Ohjelmisto tulostaa palkkaerittelyn.
6. Ohjelmisto ylläpitää palkkakertymää.
7. Ohjelmisto ylläpitää ennakonpidätyskertymää.
8. Ohjelmisto ylläpitää palkanlisien kertymää.
9. Ohjelmisto tulostaa palkkakertymän.
10. Ohjelmisto tulostaa työnantajamaksut.
11. Ohjelmisto ylläpitää tietoa työnantajamaksuista.

Luettelon järjestys on samalla ohjelmiston toimintojen tärkeysjärjestys.

Käyttäjät ja toimintaympäristö

Lopullinen pääkäyttäjä on Kuljetusliike Veini Peltonen Ky:n palkkahallinnosta vastaava henkilö. Toimintaympäristö on toimistoympäristö tavallisine kiinteine tietokoneineen. Tietokone, johon valmis ohjelmisto asennetaan, ei ole liitetty Internetiin. Tämän vuoksi ohjelmisto ei tule liittymään muihin tietokoneisiin esim. Internetin tai sisäisen verkon kautta. Tästä johtuen ohjelmiston suojaamiseen ei tarvitse kiinnittää niin paljon huomiota.

Rakenteet

Tietokannan rakenteen tuli olla selkeä ja tarkasti tarpeeseen sovitettu. Taulujen nimien tuli olla sisältöä kuvaavia ja selvästi suomen kielellä ilmaistu. Tietokannan tuli olla muutosjoustava, mikä tarkoittaa, että tietokannan hoitajan tulee voida lisätä uusia tauluja ja sarakkeita koskematta olemassa oleviin ohjelmiin. Tietokannan rakenteen tuli pysyä yksinkertaisena, josta päästään helposti hyvään suorituskykyyn. Lisäksi rakenteessa noudatettiin avaineheys- ja viiteeheys-sääntöä.

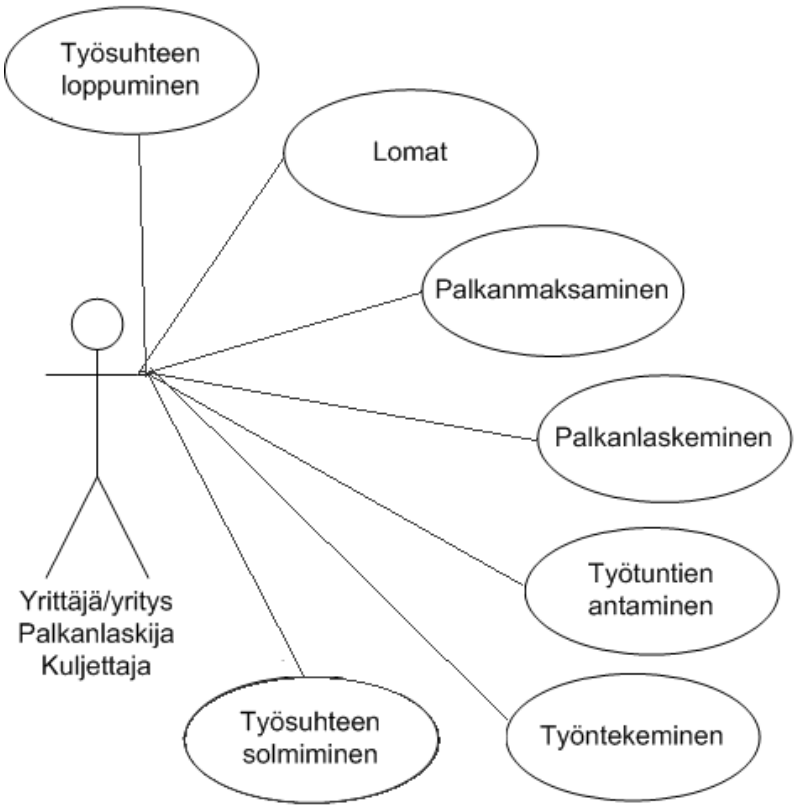
Käyttöliittymän rakenteen tuli olla johdonmukainen ja selkeä. Sen tuli ohjata toimintaa eteenpäin. Näkymien nimien tuli olla suomen kielellä ilmaistu ja kuvata sitä tapahtumaa, mitä näkymässä hoidetaan.

Lisäksi palkanlaskennassa, joka tapahtuu käyttöliittymässä, tuli ottaa huomioon työehtosopimuksen määräykset. Se määrittää palkanmaksuväliksi 2 viikkoa, kertoo vähimmäistuntipalkan ja päivärahan suuruudet, lomien ja työajanlyhennyksen pituudet.

3.1. Ohjelmiston määrittäminen UML-mallinnuksen avulla

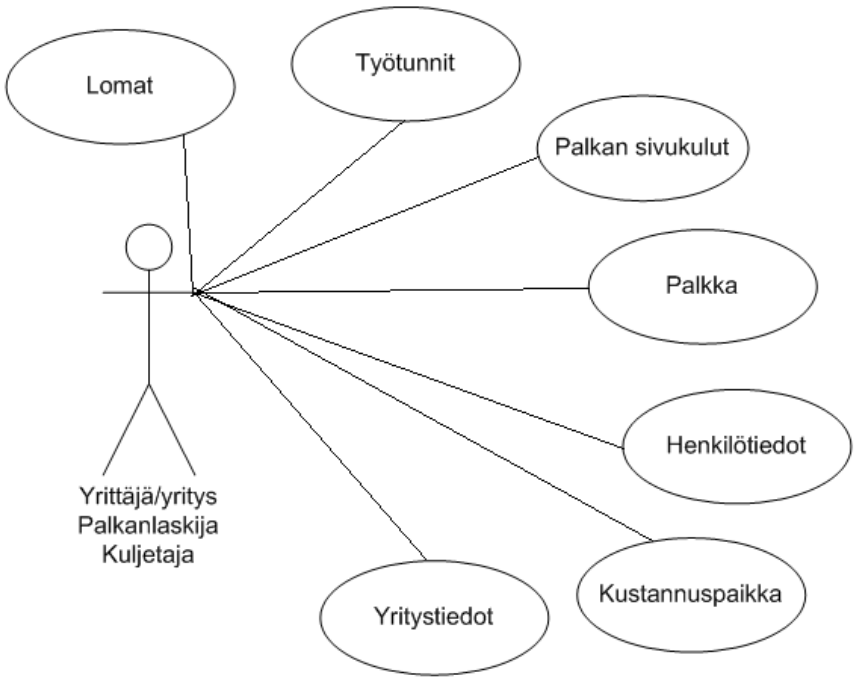
Yrityksellä tuli henkilöstöhallintaa seuraavia käyttötapauksia: työsuhteen solmiminen, työskentely, lomien pitäminen ja työsuhteen loppuminen. Käyttäjinä

näissä käyttötapauksissa on yrittäjä, palkanlaskija ja työntekijä. Kuvatuista käyttötapauksista tehtiin kuvan 11 mukainen käyttötapauskaavio.



Kuva 11. Henkilöhallinnan käyttötapaukset ja käyttäjät

Käyttötapauksista etsittiin toimintoja, jotka liittyvät käyttötapauksiin. Toimintoja etsiessä mietittiin samalla, kuinka ne tulevat liittymään tulevaan ohjelmistoon ja mitä vaiheita ne sisältävät. Toiminnoiksi saatiin esimerkiksi seuraavat: Yritystietojen ylläpito, henkilötietojen ylläpito, työtuntien antaminen, palkan laskeminen, palkan maksaminen ja palkan sivukulujen hoitaminen. Käyttötapauksien toiminnoista laadittiin kuvan 12 mukainen kaavio. Kaavio ei ole tyhjentävä, vaan suuntaa antava. Toiminnot tarkentuivat suunnitelman edetessä.



Kuva 12. Henkilöhallinnan toiminnot ja käyttäjät

Käyttötapaustaulukot

Edellisten kuvien avulla päästiin laatimaan käyttötapaustaulukoita. Ne kertovat sanallisesti mitä tarvitaan, jotta kaikki käyttötapaukset tulevat tehtyä ja mihin päästään käyttötapauksen tultua suoritettua. Käyttötapaustaulukot helpottavat tietokannan suunnittelua ja toteutusta. Käyttötapaukset on esitetty seuraavissa taulukoissa. Taulukoita ei ole nimetty ja numeroitu erikseen, vaan käyttötapauksen nimi löytyy taulukosta itsestään.

KÄYTTÖTAPAUS:	***Työsuhteen solmiminen***
YHTEENVETO:	Yritykseen tulee uusi työntekijä.
AKTORIT:	Yrittäjä, palkanlaskija ja kuljettaja
EHDOT:	Yritys tarvitsee uuden työntekijän. Yrittäjä haluaa palkata työntekijän. Työntekijä haluaa työskennellä yrityksessä.
KUVAUS:	Yrittäjä ja kuljettaja solmivat työsopimuksen. Työsopimuksessa sovitaan työsuhteen ehdot: palkanperuste ja määrä. Kuljettaja antaa verokortin yrittäjälle, joka toimittaa sen muiden papereiden kera palkanlaskijalle. Palkanlaskija tallentaa tietokantaan kaikki tarpeelliset tiedot.
POIKKEUKSET:	Työehdoista ei sovita. Tietokanta ei toimi.
LOPPUTULOS:	Kuljettaja aloittaa työt yrityksessä. Kuljettajan tiedot ovat tietokannassa.

KÄYTTÖTAPAUS:	***Työn tekeminen***
YHTEENVETO:	Työntekijät ja yrittäjä työskentelee yrityksessä.
AKTORIT:	Yrittäjä, palkanlaskija ja kuljettaja
EHDOT:	Yrittäjä on laatinut työvuorolistan ja työntekijät työskentelevät sen mukaisesti.
KUVAUS:	Kaikki työntekijät työskentelevät yrittäjän johdon alaisina. Palkkajakso tulee täyteen. Työntekijät kirjaavat työtuntinsa ylös ja käyttävät ajotehtävissä ajopiirturia, jonka kiekko toimiin palkan tuntimäärän perusteena.
POIKKEUKSET:	Lakko, ajoneuvo rikki
LOPPUTULOS:	Työntekijöillä on työtunnit toimitettavaksi palkanlaskijalle..

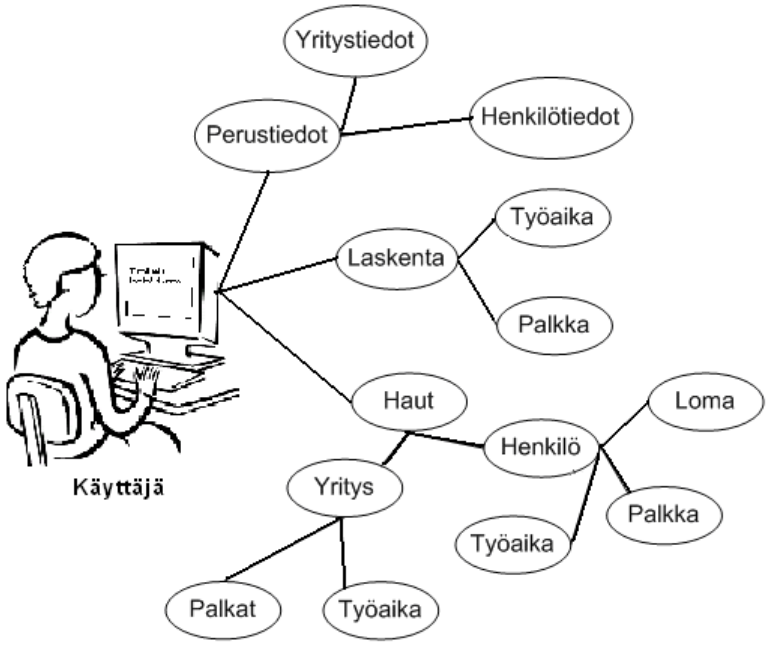
KÄYTTÖTAPAUS:	***Palkanlaskeminen***
YHTEENVETO:	Palkanlaskija laskee työntekijöiden palkat.
AKTORIT:	Palkanlaskija
EHDOT:	Työntekijät ovat toimittaneet palkkalistat tai ajopiirturinkiekot palkanlaskijalle.
KUVAUS:	Palkanlaskija laskee työtunnit ja työntekijöiden palkat. Samalla palkanlaskija kirjaa tiedot palkkakirjanpitoon ja työaikaseurantaan.
POIKKEUKSET:	Lakko, tietokanta ei toimi.
LOPPUTULOS:	Työntekijöiden palkat on laskettu. Palkkatiedot kirjattu tietokantaan.

KÄYTTÖTAPAUS:	***Palkanmaksaminen***
YHTEENVETO:	Yrittäjä maksaa työntekijöiden palkat.
AKTORIT:	Yrittäjä
EHDOT:	Palkanlaskija on laskenut palkat ja pankkitilillä on katetta.
KUVAUS:	Yrittäjä tekee pankkisiirtokuitit ja maksaa työntekijöiden palkat ja palkan sivukulut. Tiedot yrittäjä saa palkanlaskijan tulostamista tulosteista ja tietokanasta.
POIKKEUKSET:	Palkkoja ei ole laskettu. Tietokone ei toimi.
LOPPUTULOS:	Työntekijöiden palkat ja sivukulut on maksettu.

KÄYTTÖTAPAUS:	***Lomat***
YHTEENVETO:	Pidetyt lomat tallennetaan tietokantaan.
AKTORIT:	Yrittäjä, palkanlaskija ja työntekijä.
EHDOT:	Yrittäjä on hyväksynyt pidettävän loman. Loman pitäjä on pitänyt loman. Lomasta on tullut tieto palkanlaskijalle.
KUVAUS:	Lomalaisen loma on päättynyt ja siitä on tullut tieto palkanlaskijalle. Palkanlaskija tallettaa tiedot tietokantaan.
POIKKEUKSET:	Lomasta ei ole tietoa palkanlaskijalla.
LOPPUTULOS:	Loman tiedot on merkitty tietokantaan.

KÄYTTÖTAPAUS:	***Työsuhteen päättyminen***
YHTEENVETO:	Työntekijän työsuhde päättyy ja tiedot merkitään tietokantaa.
AKTORIT:	Yrittäjä, palkanlaskija ja työntekijä.
EHDOT:	Työsuhde on irtisanottu ajoissa tai työsuhteen purkuun on muu pätevä syy.
KUVAUS:	Työntekijän työsuhde on päättynyt. Palkanlaskija laskee lopputilin ja kirjaa sen tietokantaan. Samalla palkanlaskija kirjaa työsuhteen päättymispäivän tietokantaan.
POIKKEUKSET:	Tietokone ei toimi.
LOPPUTULOS:	Työsuhteen päättymisen tiedot (lopputili ja päättyminen) on merkitty tietokantaan.

Käyttöliittymän ja tietokannan käyttötapaukset voitiin johtaa käyttötapaustaulukoista ja vaatimuksenmäärittely dokumentista. Toteutettu ohjelmisto sisältää Accessilla toteutetun tietokannan ja Visual Basicilla toteutetun käyttöliittymän tietokantaan. Käyttöliittymän ja tietokannan käyttötapaukset on esitetty kuvassa 13. Tämä mallinnus voitiin jalostaa käsiteanalyysin kautta ER-kaavioksi tietokannan suunnittelussa. Lisäksi mallinnusta käytettiin apuna käyttöliittymän suunnittelussa.

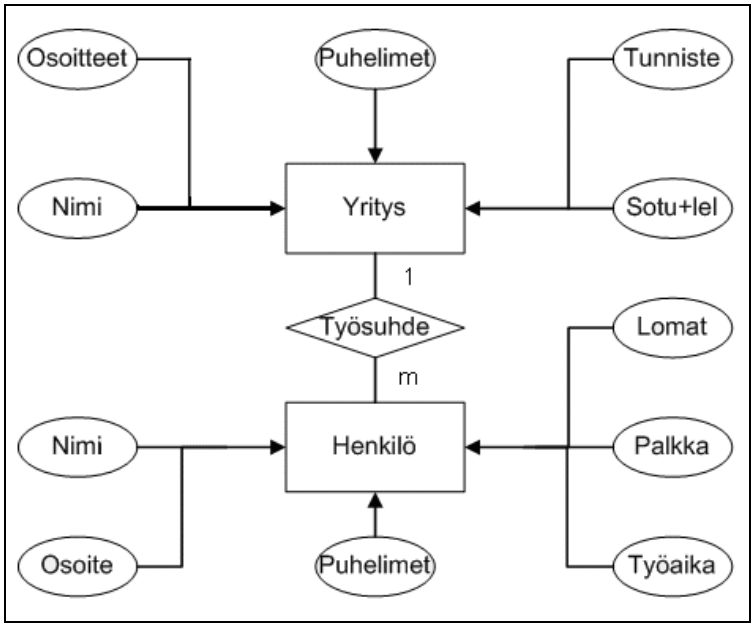


Kuva 13. Käyttöliittymän ja tietokannan käyttötapaukset

3.2 Tietokannan määrittely

Tietokannan määrittelyvaatimuksena oli eheys, muutosjoustavuus, yksilöitävyys ja suomenkielisyys. Käyttötapauksista määriteltiin käsitteet ja niille tarpeellinen tieto. Nimet mietittiin valmiiksi selvällä suomen kielellä ja kohdeasiaa kuvaaviksi. Käsitteiksi saatiin esimerkiksi yritystiedot, henkilötiedot ja työtiedot. Luettelo ei ole tyhjentävä. Seuraavaksi merkittiin käsitteiden väliset yhteydet. Käsitemalliin pyrittiin merkitsemään kaikki tarpeellinen tieto, jolloin määritelmästä tuli vaatimukset täyttävä. Aluksi käsitteistä löytyi iso kokonaisuus, jonka pystyi jakamaan pienempiin osiin. Esimerkiksi työntekijäkäsitteen pystyi jakamaan palkkaan, ve-

roihin jne. Samalla voitiin varmistua, että tuleva tietokanta palvelisi kunnolla asetettuja vaatimuksia. Tässä vaiheessa ei tarvinnut vielä miettiä, kuinka tämä kaikki lopulta toteutettaisiin, ainoastaan tehdä kunnolliset määritykset tietokannasta. Käsitemalli kuvattiin ER-kaaviona. Kuvassa 14 on esitetty pelkistetty ER-kaavio toteutetusta tietokannasta.

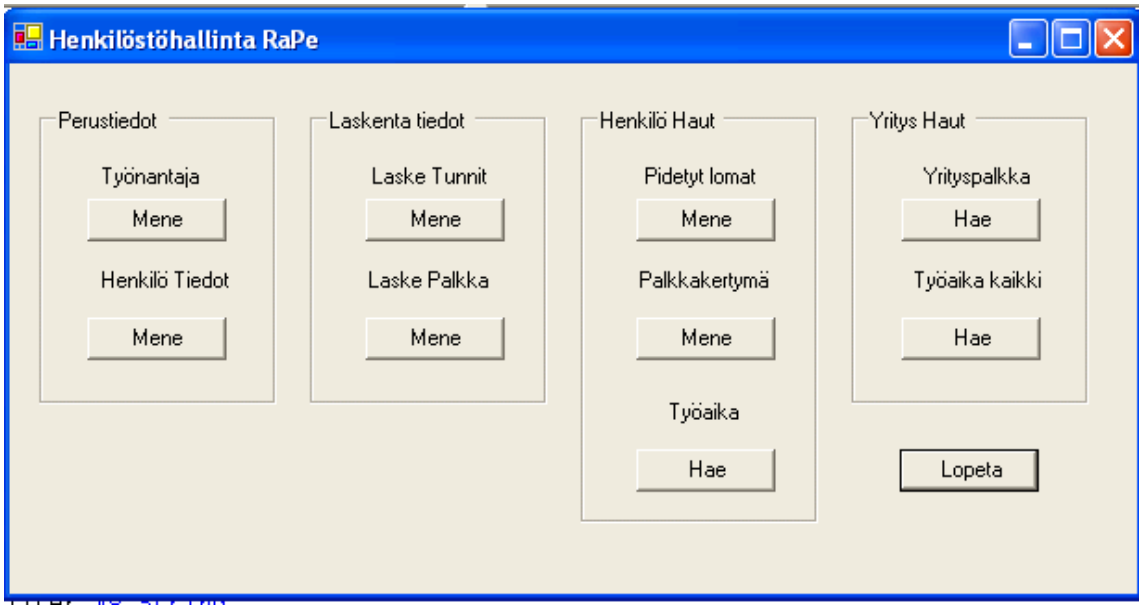


Kuva 14. Pelkistetty ER-kaavio toteutetusta tietokannasta.

3.2 Käyttöliittymän määrittely

Käyttöliittymällä oli vaatimuksena selkeys, yksinkertaisuus ja toiminnan ohjaaminen. Nämä kaikki vaatimukset oli mahdollista saavuttaa hyvällä määrittelyllä ja suunnittelulla. Käyttöliittymää määriteltäessä otettiin malliksi jakaa toiminnot 3:een eri alueeseen: perustieto, laskeminen ja raportointi. Lisävaatimuksen toi asiakasyrityksen laatukäsikirjassa oleva maininta ”vältetään paperitulosteiden käyttöä”, mikä tarkoitti tässä projektissa ohjelmiston tulostamisen minimointia. Ohjelmisto ei tulosta paperille muuta kuin palkkalistan. Muut tulosteet näkyvät tietokoneen näytöllä, josta niitä voidaan kopioida muihin ohjelmiin ja lomakkeisiin.

Käyttöliittymään määriteltiin etusivu, jonka avulla ohjelmiston toimintaa hoidetaan. Lisäksi määriteltiin avautuvien näkymien järjestys. Järjestyksellä on tärkeä merkitys varsinkin tallennettaessa tietoja tietokantaan. Tallennusjärjestyksen tulee noudattaa viite-eheys sääntöä. Jos yritetään tallentaa lapsitauluun tietoa, jota ei ole isätaulussa, tulee suorituksen aikainen virhe. Lisäksi on hyvä muistaa, että Visual Basic käsittelee alkuperäisen tietokannan kopiota. Tästä johtuen uusi tai käsitelty tieto täytyy aina tallentaa myös alkuperäiseen tietokantaan. Liitteessä A on esitetty vuokaavio käyttöliittymän näkymien avautumisjärjestyksestä. Käyttöliittymän näkymien yhteyksistä tietokantaan määriteltiin osa tehtäväksi Visual Basicin ohjatun toiminnon avulla ja osa luomalla yhteys ohjelmallisesti itse. Käyttöliittymän aloitussivulle laitettiin selkeästi, mihin toimintoon kukin komentopainike kuuluu, ja painiketta painaessa tulee näkyviin seuraava näkymä. Käyttöliittymän aloitusnäkymä on esitetty kuvassa 15.



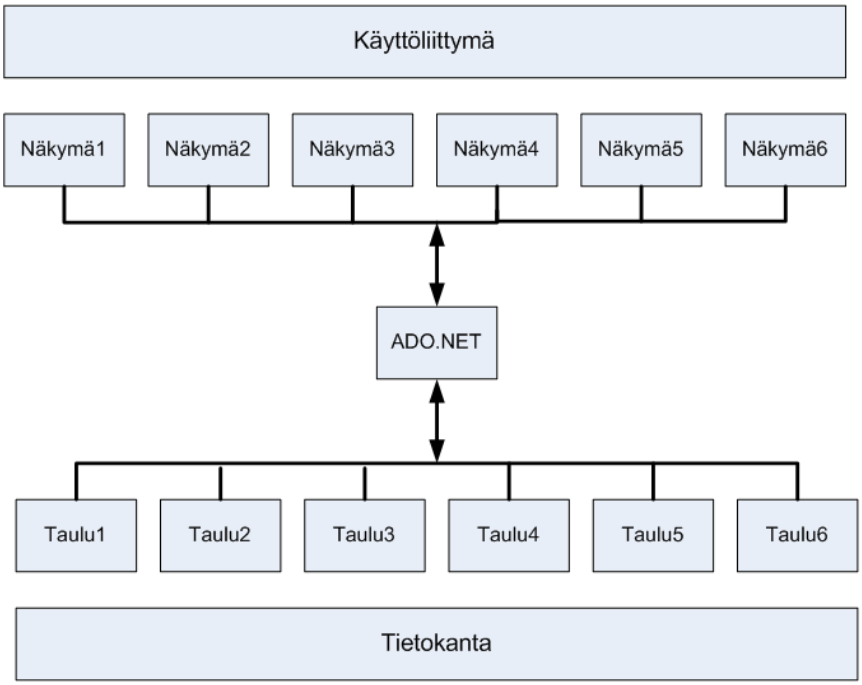
Kuva 15. Tietokannan käyttöliittymän aloitusnäkymä.

Käyttöliittymään määriteltiin tarvittavaa sisältöä (painonappeja, textbox jne.) näkymiin. Laskemiseen määriteltiin ohjelmien kulku, jotka kuvattiin vuokaavion avulla. Työajan laskemiseen käytettävän ohjelmiston vuokaavio on esitetty liitteessä B ja palkanlaskemiseen käytettävän ohjelmiston vuokaavio on esitetty

liitteessä C. Näkymien sisältöä ja ohjelmisto-osien sopivuutta ja tietokattavuutta pöytätestattiin. Sisältöä ja ohjelmia täydennettiin ja lisättiin puuttuvia osia.

4 OHJELMISTON SUUNNITTELU

Arkkitehtuurisuunnittelussa suunniteltiin tietokannan ja käyttöliittymän rakenteet ja niiden väliset yhteydet. Suunnitelmassa tietokanta jaettiin lopullisesti tauluihin, joiden tietoja käsiteltiin käyttöliittymän näkymien kautta. Visual Basic .Net käyttää tietokantarajapintana ADO.NETiä, joka mahdollistaa monipuoliset yhteydet tietokannan ja käyttöliittymän välillä. Kuvassa 16 on esitetty periaatekuva käyttöliittymän ja tietokannan liittymisestä toisiinsa ja niiden välisestä yhteydestä.



Kuva 16. Käyttöliittymän ja tietokannan yhteydet

Moduulisuunnittelussa ohjelmisto jaettiin osiin, jolloin ohjelmisto oli helpompi testata, muuttaa ja ymmärtää. Ohjelmiston modulaarisessa suunnittelussa pyrittiin käyttämään hyväksi joko valmiita tai ohjatun toiminnon avulla toteutettuja ohjelmistorakenteita. Aivan kaikkea näin ei voinut toteuttaa, joten sellaisille osille toteutettiin oma toiminto.

Tietokannan moduulisuunnittelu

Suunnittelussa tietokannan toteutusmalliksi otettiin taulu kerrallaan ohjelmointi. Näin taulusta tulee kerralla valmis testattavaksi. Kun kaikki taulut oli saatu suunniteltua, laitettiin taulujen väliset yhteydet kuntoon. Tauluun sisällön saamiseksi käsitemallin ER-kaaviota laajennettiin ja käsitteiden sisälle lisättiin tarvittavaa tietoa. Tässä vaiheessa ei vielä ollut pakko mennä aivan tarkalle tasolle, mutta suunnitelma voitiin jo nyt suunnitella alustavasti kolmanteen normaalimuotoon. Tällöin taulut eivät myöhemmässä vaiheessa hajoa enempää ja taulujen määrä ei kasva. Lisäksi jo nyt voitiin miettiä valmiiksi perusavaimet ja viiteavaimet, jolloin säästyttiin myöhemmin niiden lisäämiseltä. Perus- ja viiteavaimia voi suunnittelun edetessä muuttaa, jos tarvetta ilmenee.

Tarveanalyysi

Tarveanalyysissä pöytätestattiin ja tarkistettiin, että toteutettuun tietokantaan tuli kaikki tarpeellinen tieto. Puutteellisuudet korjattiin lisäämällä puuttuvat käsitteet ja yhteydet. Pöytätestauksessa käytettiin käsitemallin ER-kaaviota. Tarveanalyysin lopputuloksena oli toteutettavan tietokannan taulujen sisältö. Käsitemallista voitiin nyt esittää käsitemallitaulut. Tässä vaiheessa voitiin suunnitella käsitteiden tiedoille sisällön rakenne ja merkitä se käsitemallitauluihin. Rakenteessa kerrottiin tietotyyppi, pakollisuus, oletusarvo ja rajoitukset. Kuvassa 17 on esitetty malli käsitemallitaulusta. Tämä poikkeaa teoriasta siinä suhteessa, että käsitemallitauluihin on jo laitettu viiteavaimet.

Yritys	tietotyyppi	pakollinen	oletusarvo	rajoitus
ID_yritys	luku	kyllä	ei	uniikki
nimi	teksti	kyllä	kyllä	ei
katuosoite	teksti	kyllä	kyllä	ei
halliosoite	teksti	kyllä	kyllä	ei
postinumero	teksti	kyllä	kyllä	uniikki
teleyht	luku	kyllä	ei	uniikki
kaupparekNum	teksti	kyllä	ei	uniikki
ALVnum	teksti	kyllä	ei	uniikki
Lytunnus	teksti	kyllä	ei	uniikki

Kuva 17. Malli käsitemallitaulusta

Normalisointi

Toteutetulle tietokannalle suoritettiin normalisointi kolmanteen normaalimuotoon. Tietokannalle ei tarvinnut toteuttaa monta muutosta, koska tietokanta oli jo valmiiksi suunniteltu kolmanteen normaalimuotoon. Eräs miettimistä aiheuttava kohta oli puhelimien liittäminen omistajiinsa. Kyseessä on poissulkeva-yhteys, mikä tarkoittaa, että puhelinnumerolla on vain yksi omistaja. Ongelmaa aiheutti omistajan merkitseminen puhelimet-tauluun, koska merkitsemällä yritystunnus ja henkilötunnus viiteavaimiksi olisi tauluun jäänyt paljon tyhjää tilaa, ja puolestaan jos yritystietoihin ja henkilötietoihin olisi tuonut erittelevän tiedon, olisivat taulut kasvaneet yhdellä sarakkeella. Ratkaisin ongelman tekemällä kaksi taulua yrityspuhelimet ja henkilöpuhelimet. Näin pääsin mielestäni helpoimmin tyydyttävään lopputulokseen.

Käyttöliittymän moduulisuunnittelu

Käyttöliittymä jaettiin moduuleihin näkymien mukaan, ja näkymä jaettiin moduuleihin sen toimintojen mukaan. Lisäksi suunniteltiin julkiset muuttujat, aliohjelmat ja funktiot, joilla voitiin välittää tietoa ja tehdä toimintoja. Lisäksi tapahtumat toistuvat ohjelmistossa useamman kerran. Tällainen toiminto on esimerkiksi työntekijän nimen haku tietokannasta avautuvaan näkymään. Funktioissa ja aliohjelmissa käytettiin hyväksi Visual Basic .NETiin tullutta uutta ominaisuutta,

Return-lausetta. Sen avulla voidaan palauttaa kutsuvaan ohjelmiston osaan aliohjelmassa käsitelty arvo.

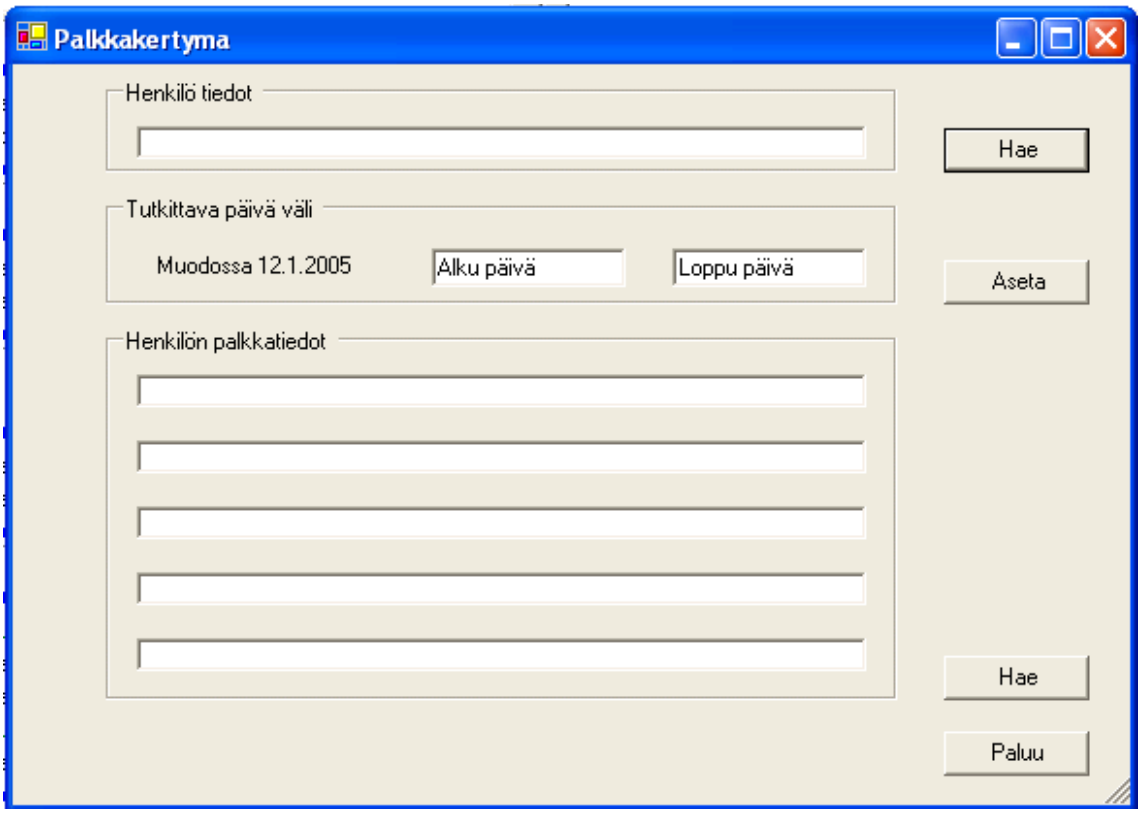
Tietokantakyselyjä suunniteltiin toteutettavaksi ohjatun toiminnon (Data Form Wizardin) avulla tai suunnittelemalla oma yhteys tietokantaan ja siihen oma kysely. Oma tietokantakysely suunniteltiin suoritettavaksi kirjoittamalla peruskysely ohjelmistoon ja sitä täydennettiin näkymästä otetulla tarkennetulla tiedolla. Seuraavassa esimerkissä on selvennetty edellistä. Kyselyllä halutaan saada tietyn aikavälin palkkatiedot ryhmiteltynä henkilön mukaan. Kysely on summa SQL-kysely, mutta käyttäjä ohjaa kyselyn tulosta kirjoittamalla näkymään varattuun tilaan kyselyn alkamispäivämäärän ja loppumispäivämäärän. Nämä muutetaan ohjelmallisesti oikeaan formaattiin Accessille sopiviksi ja tallennetaan päivämääriä varten nimettyihin muuttujiin esimerkiksi alkPv ja lopPv. Muuttujat lisätään kyselyyn oikeaan kohtaan ja kysely tallennetaan kyselyä varten nimettyyn muuttujaan esimerkiksi haku. Kyselyyn päivämäärät saadaan liitettyä katkaisemalla lainausmerkeillä rakennettava kysely ja lisäämällä muuttuja & -merkkien väliin. Tämän jälkeen suoritetaan kysely. Seuraavaksi on esitetty edelliseen liittyen ainoastaan hakukomennon rakentaminen. Tämä selventää edellä kerrottua.

```
haku =”SELECT Palkka.ID_henkilo, Sum(Palkka.palkkaYht) AS
palkkaYht, Sum(Palkka.vero) AS vero, Sum(Palkka.sotu) AS
sotu, Sum(Palkka.lel) AS lel, Sum(Palkka.paivaRaha) AS pai-
vaRaha FROM(palkka) WHERE lopPvm between #" & alkPv & "#
and #" & lopPv & "# GROUP BY Palkka.ID_henkilo;”
```

Käyttöliittymä suunniteltiin ohjaamaan käyttäjää siten, että seurataan julkisella muuttujalla, mistä näkymästä tullaan uuteen näkymään ja toimitaan muuttujan arvon mukaisesti. Näkymästä poistetaan näkyvistä tarpeettomia painonappeja tai tarpeetonta tietoa. Näkymästä palataan edelliseen näkymään tai suoraan aloitusnäkymään. Ohjelmiston sulkemista ei laiteta muualle kuin aloitusnäkymään.

Perustiedoissa tallennetaan ja päivitetään tietokantaan henkilötietoja ja yritystietoja. Tietoja ovat esimerkiksi henkilönnimi, sukunimi, osoite jne. Laskenta-

tiedoissa suoritetaan kaikenlainen laskenta ja käsittely tietokannan tiedoille. Laskentaa on työtuntien laskenta ja palkanlaskenta. Käsitellyn tiedon esittämiseen suunniteltiin omat näkymät. Nämä ryhmiteltiin henkilö- ja yrityshakujen mukaan. Henkilöhakuihin kuuluvat kaikki yksittäistä henkilöä koskeva ja yrityshakuihin kaikkia henkilöitä yhdessä koskevia tietojen kokonaismääriä. Tietoja ovat esimerkiksi palkka, vero ja päiväraha. Käsitellyn tiedon näkymä esimerkki on esitetty kuvassa 18.



Kuva 18. Henkilön palkkakertymän näkymä

Ohjelmiston ajonaikaisille virheille suunniteltiin try-catch-rakenne, jota voitiin tarpeen mukaan muuttaa käytön mukaisesti toimivaksi. Rakenteen Try-osaan laitetaan toiminto, jonka arvellaan tuottavan vaikeuksia. Catch-osaan kirjoitetaan lauseet, jotka suoritetaan virheen sattuessa. Lisäksi on finally-osa, johon kirjoitetut lauseet suoritetaan aina. Rakenteen avulla on helppo paikallistaa virheen paikka ohjelmistossa.

Paperille tulostaminen suunniteltiin suoritettavaksi Word-dokumenttina. Ohjelmisto kirjoittaa Word-dokumentin. Käyttäjä voi täydentää ja lisätä dokumenttiin haluamiaan asioita. Lopuksi käyttäjä voi tallentaa dokumentin haluamaansa tallennuspaikkaan. Tulostamiselle suunniteltiin rakenne ja ohjelmalauseet. Tulostamisessa käytettiin apuna Microsoft Office –sovellusten automatisointia ja prosessien hallintaa.

Ohjelmiston testaaminen

Ohjelmiston testaamiseksi suunniteltiin suunnittelun yhteydessä testausmateriaali. Testaus suunniteltiin suoritettavaksi polkutestauksena. Ohjelmistosta käydään läpi kaikki henkilöstönhallinnassa tarvittavat ohjelmistopolut. Ehtokattavuuteen pyrittiin ottamalla kaikkien ehtojen kaikki vaihtoehdot. Materiaaliin otettiin sisältö asiakas yrityksen edellisen vuoden todellisista henkilöhallinnan tiedoista. Testaus suunniteltiin etenemään ensin yhden henkilön tiedot oikein ja sen jälkeen kahden ja lopuksi kolmen henkilön todelliset tiedot. Todellisten tietojen käyttö helpotti testauksen suunnittelua, koska testauksen oikea lopputulos oli näin jo tiedossa.

Käyttöliittymän syntaksi ja ajonaikaisista virheistä suunniteltiin päästävän helpoimmin eroon suorittamalla käännös heti ohjelmamoduulin valmistuttua ja korjaamalla heti mahdolliset virheet. Loogiset virheet ajateltiin löydettävän helpoimmin suunnitellun testausmateriaalin avulla. Virheen syy etsitään heti ja korjataan. Tällöin virheiden mahdolliset paikat on rajattu vähiin ja virheen aiheuttaja on mahdollista löytää nopeasti ja helposti.

5. OHJELMISTON TUOTTAMINEN VALMIIKSI OHJELMISTOKSI

Tietokanta toteutettiin Microsoft Access -tietokannalla, ja tietokantaa käytetään Visual Basic .NETillä tuotetulla käyttöliittymällä. Accessin avulla on helppo luoda uusi tietokanta, ja siihen voi vaivattomasti luoda käyttöliittymän Visual Basic .NETillä, koska Access käyttää Visual Basicia ohjelmointikielenä. Ohjelmisto testattiin aina rakenteilla olevan moduulin valmistuttua. Testaaminen on vaikeaa erottaa ohjelmistoprojektissa aivan itsenäiseksi osaksi.

Tietokannan ohjelmointi

Tässä työssä tietokanta ohjelmoitiin Microsoft Accessissa olevan ohjatun toiminnan avulla ja ohjelmoimisen jälkeen määriteltiin taulujen väliset yhteydet. Tietokannan taulut ja niiden väliset yhteydet on esitetty liitteessä D. Kaikki yhteydet saatiin muodostettua yksi-moneen-yhteyksiksi. Luotu tietokanta tallennettiin tietokoneen muistiin ja otettiin varmuuskopio CD-levylle. Tietokantaa ei testattu vielä tässä vaiheessa enempää, koska tietokanta saatiin luotua virheettömästi.

Käyttöliittymän ohjelmointi

Käyttöliittymän tekeminen aloitettiin luomalla yhteys relaatiotietokannan ja käyttöliittymän välille. Seuraavaksi toteutettiin aloitusnäkymä suunnitelman mukaan. Näkymään lisättiin kaikki tarpeelliset komponentit ja nimettiin ne. Nimeämisessä on hyvä käyttää yhtenäistä nimeämiskäytäntöä ja merkitä käytetyt nimet vaikka ruutupaperille myöhempää ja parempaa dokumentointia varten. Ruutupaperilta on helppo katsoa, mitä nimeä missäkin kohtaa käyttiin. Nimettäessä samaa asiaa eri näkymissä ja eri ohjelmalohkoissa kannattaa käyttää samaa nimeä, koska nämä ovat paikallisia muuttujia ja eivät näy muualla ohjelmassa, mutta koodi ohjelmalohkoihin voidaan siirtää kopioimalla. Tämä helpottaa ja nopeuttaa ohjelmankoodaamista.

Seuraavaksi luotiin aliohjelmat ja funktiot, joilla käsitellään ohjelmassa toistuvaa tiedon hakemista tai käsittelyä. Sen jälkeen luotiin muut näkymät ja tarvittavat koodin lisäykset. Perustietoja käsitellessä kannattaa käyttää Visual Basicin ohjattua toimintoa tietokantanäkymiin. Tällöin kehitysympäristö toteuttaa kaikki tarpeelliset tietokantayhteydet molempiin suuntiin. Käsitellyn tiedon hakemiseen ja käsittelyyn luotiin omat tietokantakyselyt ja tulostamiset suunnitelman mukaisesti.

Tietokannan tietoja käsitellessä on syytä muistaa, että käyttöliittymässä käsitellään alkuperäisen tietokannan kopiota. Tiedot on aina muuttamisen jälkeen talletettava relaatiotietokantaan, tai muuten uudet tiedot eivät ole käytössä myöhemmin niitä haettaessa. Kyselyissä on tärkeää luoda oikeanlainen kysely ja välittää se tietokantaan ja saada haun tulos näkyviin. Tietokantakyselyssä alkuperäisestä tietokannasta kannattaa tuoda suurempi tietojoukko käyttö-liittymään. Käyttöliittymässä tietojoukko käsitellään sopivasti lajittelemalla ja lopputuloksena esitetään oikea tulos hakuun. Aikaa säästyy, kun lajittelu suoritetaan käyttöliittymässä eikä itse tietokannassa. Lisäksi samaa tietojoukkoa voidaan käyttää käyttöliittymässä hyödyksi toisenlaisella suodatuksella. Esimerkiksi palkkahaku muodostettiin seuraavasti:

Esitellään näkymälle yhteiseksi yhteys ja tiedontallennuspaikka:

```
'yhteys tietokantaan
Dim yhdista As New System.Data.OleDb.OleDbConnection
'kopio basicissa access tietokannasta
Dim dataset As New System.Data.DataSet
```

Ensin luodaan yhteys tietokantaan:

```
yhdista.ConnectionString = "Pro-
vider=Microsoft.Jet.OLEDB.4.0;Data source=" &
"D:\Raimo\inssityö\työpaperit\kopio rape.mdb"
```

Luodaan tarpeelliset muuttujat, joilla haetaan ja käsitellään tietoa:

```
Dim haku As String
Dim filter As String
Dim toka As DataRow
Dim Paikka() As DataRow
```

Luodaan haku tietokannasta, huomaa alkPv ja lopPv on kysytty ohjelmallisesti näkymässä:

```
haku = "SELECT Palkka.ID_henkilo, Sum(Palkka.palkkaYht) AS
palkkaYht, Sum(Palkka.vero) AS vero, Sum(Palkka.sotu) AS
sotu, Sum(Palkka.lel) AS lel, Sum(Palkka.paivaRaha) AS pai-
vaRaha FROM(palkka) WHERE lopPvm between #" & alkPv & "#
and #" & lopPv & "# GROUP BY Palkka.ID_henkilo;"
```

Laitetaan haku käyntiin:

```
Dim komento As New System.Data.OleDb.OleDbCommand(haku, yh-
dista)
'liitin tietokantaan
Dim adapter As New Sys-
tem.Data.OleDb.OleDbDataAdapter(komento)
'Haetaan tiedot tietokannasta
Try
 'avataan tietokanta yhteys
 Me.yhdista.Open()
 dataset.Clear()
 adapter.Fill(dataset, "Palkka")
 'Suljetaan tietokanta yhteys
 Me.yhdista.Close()
Catch ex As Exception
 MsgBox("Ei yhteyttä tietokantaan")
 End Try
```

Laitetaan tiedot näkymässä oikeaan paikkaan:

```
Try
 Paikka = dataset.Tables("Palkka").Select(filter)
 'Käydään kopio taulun rivit läpi ja etsitään sopiva
 Dim id As Integer, yhtPalkka As Decimal, vero As Decimal,
 sotu As Decimal
 Dim lel As Decimal, paiva As Decimal
 For Each toka In Paikka
     'laitetaan tiedot listboxiin
     id = toka("ID_Henkilo").ToString()
     yhtPalkka = toka("palkkaYht").ToString()
     vero = toka("vero").ToString
     sotu = toka("sotu").ToString()
     lel = toka("lel").ToString()
     paiva = toka("paivaRaha").ToString
      lsbPalkkaYht.Items.Add(id & " henkilön ajanjakson
      palkka " & yhtPalkka)
     lsbVero.Items.Add(id & " henkilön ajanjakson vero " &
      vero)
     lsbSotu.Items.Add(id & " henkilön ajanjakson sotu " &
      sotu)
      lsbLel.Items.Add(id & " henkilön ajanjakson lel " &
      lel)
     lsbPaivaRaha.Items.Add(id & " henkilön ajanjakson Päi-
      väraha " & paiva)
 Next
Catch
```

```
            MsgBox("Virhe, ota yhteys tekniseen tukeen")
        End Try
```

Samalla tavalla voidaan muodostaa kaikki tarpeelliset haut. Useamman taulun haut muodostetaan periaatteessa samalla tavalla, mutta taulujen nimet lisätään varatun sanan FROM jälkeen. Lisäksi täytyy olla tarkka rakentaessa hakua, jotta hakutulos tulee oikein ja sisältää sitä tietoa, mitä todella tarkoitettiin hakea. Apuna hakujen rakentamisessa voi käyttää Accessin ohjattua toimintoa. Sieltä voi katsoa, miltä haku näyttää SQL-kielellä. Kaikki hakukomennot oli suunniteltu alustavati suunnitteluvaiheessa. Mahdolliset ajonaikaiset virheet käsiteltiin try-catch-rakenteella, joka suunniteltiin alustavasti suunnitteluvaiheessa.

Ohjelmiston testaus ja käyttöönotto

Ohjelmointivaiheessa tietokantaa ja käyttöliittymää testattiin aina osavaiheen valmistuttua. Näin ohjelmointivaiheessa saatiin poistettua syntaksi- ja ajonaikaiset virheet ja osa loogisista virheistä. Polku- ja ehtokattavuus on helpoin saavuttaa juuri moduulitestauksessa. Esimerkiksi napin painamisen jälkeen tuli avautua uusi näkymä. Testauksessa kokeiltiin ohjelmaa ajossa ja katsottiin, että oikea näkymä avautui. Näin testattiin ohjelmiston kaikki toiminnot jo koodaamisen aikana ja varmistuttiin oikeasta lopputuloksesta. Ohjelmiston valmistuttua järjestelmätestauksella haettiin loogisia virheitä. Looginen virhe voi olla virheellinen lopputulos laskutoimituksessa tai virheellinen uusi näkymä näkymän vaihdon yhteydessä. Testaus suoritettiin ohjelmiston suunnitteluvaiheessa suunnitellulla materiaalilla, ja testauksen antamia tuloksia verrattiin suunniteltuihin tuloksiin. Virheen aiheuttama ohjelmalohko paikallistettiin ja virhe korjattiin. Testausta toistettiin niin kauan, että lopputulokseksi saatiin virheetön tulos. Loogisten virheiden etsintään Visual Basicissa on askellustila, jonka avulla ohjelmisto voidaan suorittaa lause kerrallaan. Ohjelmaa testattaessa käytetään Debug-työkaluriviä, jonka toiminnot on tarkoitettu pelkästään virheiden etsimiseen [4, s.218].

Virheettömän oman testauksen jälkeen järjestelmätestauksen suoritti yrityksen puolelta henkilö, joka tulee käyttämään ohjelmistoa. Testauksessa käytettiin samaa aineistoa, jolla suoritettiin oma järjestelmätestaus. Aineisto käsitti 3 hen-

kilön 3 kuukauden tiedot. Testaajalle annettiin alkukoulutus ja ohjeet kuinka ohjelma toimii. Testaaja suoritti testin ja merkitsi tulokset käytöstä liitteen E mukaiseen lomakkeeseen. Ohjelmisto toimi pienen alkulämmittelyn jälkeen hyvin ja tulokset olivat edellisen testauksen mukaiset. Testauksen tuloksena ohjelmaan päivitettiin näkymässä käsiteltävän henkilön nimitietojen näkyminen avautuvassa näkymässä. Näin käyttäjä näkee suoraan, kenen tietoja on käsittelemässä.

Testauksen onnistuttua virheettömästi käyttöliittymästä ja tietokannasta tehtiin käyttöönotto-CD-levy, jonka avulla ohjelmisto asennettiin asiakkaan tietokoneelle. Ohjelmiston toiminta testattiin samalla, kun annettiin käyttäjäkoulutusta ohjelmiston tuleville käyttäjille. Ohjelmisto on toiminut moitteettomasti 1.3.2005 alkaen. Ohjelmistoon on tulossa pieniä päivityksiä käyttäjän toiveiden mukaan. Päivitykset parantavat käytettävyyttä ja monipuolistavat tietokannasta saatavia raportteja.

6 OHJELMISTOPROJEKTIN DOKUMENTOINTI

Ohjelmistotyölle on tyypillistä, että siinä kirjataan projektin aikana kertynyttä tietoa dokumentin muotoon. Minidokumentaation muodostavat projektisuunnitelma, määrittelydokumentti ja suunnitteludokumentti. Ylläpidon helpottamiseksi dokumentaatio kannattaa koota mahdollisimman helposti ylläpidettävään muotoon. [2, s.50.]

Ohjelmistosta on laadittu vaatimusten määrittely, tekninen suunnitelma, testaus- ja käyttöönottodokumentit. Dokumentointia suoritettiin koko ajan ohjelmiston edistyessä. Dokumentointi oli tärkeä apu ohjelmiston toteutusvaiheessa ja myöhemmin ylläpitovaiheessa, kun ohjelmaa päivitettäessä muistellaan kuinka jokin asia on hoidettu. Nyt tarvitsi seurata vain valmista suunnitelmaa ja laittaa se toimivaan muotoon. Muutokset kirjattiin aluksi ruutupaperille ja myöhemmin päivitettiin kyseiseen dokumenttiin, mitä muutos koskee.

Vaatimustenmäärittelydokumentti sisältää asiakkaan ohjelmistolle asettamat vaatimukset, ja sen laatimisen aikana esille tulleet lisäykset, kuten esimerkiksi, miten tulostukset hoidetaan. Tekninen suunnitelma sisältää tiedot ohjelmiston sisällöstä ja sisällön tuottamisesta. Tässä projektissa oli tavallaan kaksi eri ohjelmistoa, tietokanta ja käyttöliittymä, jotka toteutettiin toimimaan yhdessä. Tekninen suunnitelma sisältää myös, miten tämä ohjelmistojen välinen rajapinta hoidetaan ja toteutetaan. Testaus- ja käyttöönottodokumentti sisältää testauksen ja käyttöönoton suunnittelun. Testauksesta kerrotaan, mitä testataan ja miten testataan. Testauksesta on luotu materiaali, jonka avulla testaaminen voidaan suorittaa myöhemmin uudelleen. Käyttöönotosta kerrotaan, miten ohjelmiston käyttöönotto tapahtuu ja miten käyttäjiä koulutetaan ohjelmiston käyttöön.

7 TULOKSEN ARVIOIMINEN JA OHJELMISTON KEHITTÄMINEN

Insinöörityön tuloksena syntyi henkilöstöhallintaohjelmisto, joka täytti kaikki sille asetetut vaatimukset. Työ valmistui ajallaan ja on toiminut moitteettomasti. Työn tilaaja on tyytyväinen ohjelmiston toimintaan ja sen antamiin tulosteisiin. Ohjelmiston avulla henkilöhallinnan työ on helpottunut ja henkilöstöön liittyvät tiedot ovat nopeasti saatavilla.

Ohjelmistolle asetettiin vaatimuksiksi työsuhteen tietojen ylläpito, palkkaukseen liittyvien asioiden ylläpito ja laskeminen ja lomiin liittyvien tietojen ylläpito. Ohjelmisto hoitaa yrityksen tiedot ja henkilöstön tiedot, joten työsuhteen tiedot on hoidettu näiden asioiden mukana. Ohjelmisto laskee henkilöstön työajan ja sen avulla myös henkilöstön palkan ja palkan sivukulut. Ohjelmisto antaa palkkaerittelyn käyttäjän haluamalla aikavälillä ja näyttää palkan sivukulut kyseisellä aikavälillä. Henkilöstön lomista ohjelmisto pitää kirjaa vuositasolla ja näyttää käyttäjän haluamat tiedot lomakausittain. Näin ollen ohjelmisto täyttää kaikki sille asetetut vaatimukset.

Käyttäjätestauksen tuloksena ohjelmaa päivitettiin ja lisäkehitysideoita saatiin. Henkilön nimen lisääminen käsiteltävään lomakkeeseen oli ohjelman käytettävyyttä parantava päivitys. Ensimmäisessä versiossa käyttäjä tunnisti henkilönumeron perusteella käsiteltävän henkilön. Nyt käyttäjä tietää tarkalleen, kenen tietoja käsitellään. Näin virheen mahdollisuus pienenee. Kehitysideaksi saatiin myös menu-valikon käyttömahdollisuus ohjelmiston kulun ohjaukseen. Menuvalikosta käyttäjä voisi ohjata ohjelmiston toimintaa.

Ohjelmistoon on suunnitteilla erilainen tiedonhaku tietokannasta. Käyttäjä voisi antaa hakuehtoja näkymässä ja ohjelmallisesti rakennettaisiin tietokantakysely, joka tuottaa käyttäjän haluaman lopputuloksen. Työajan laskentaan on suunnitteilla ohjelmisto, jonka avulla henkilöstö tallentaa itse työtuntinsa tietokantaan ja palkanlaskijalle jäisi vain työtuntien tarkastaminen ja palkanlaskeminen. Lisäksi ohjelmistoon on suunnitteilla osio, jonka avulla voidaan ohjata ajoja ja seurata suoritettuja ajoja.

8 YHTEENVETO

Tavoitteena oli luoda henkilöstönhallintaohjelma, jonka tuli helpottaa ja nopeuttaa yrityksen henkilöstönhallinnan toimia. Työ saavutti nämä vaatimukset hyvin. Nyt ohjelman toimiessa henkilöstönhallinnan tehtävät voidaan hoitaa silloin, kun ne tekijälleen sopivat parhaiten. Ohjelmiston avulla on voitu vähentää paperitulosteiden määrää, joten tämäkin vaatimus tuli toteutettua.

Ohjelmiston toteutuksessa käytettiin Access -tietokantaa ja Visual Basicilla toteutettua käyttöliittymää. Tietokantaan tallennetaan kaikki tieto. Käyttöliittymän avulla tietoa lisätään tai käsitellään tarpeen mukaan. Nyt toteutetuilla käsittelyillä voidaan tulostaa tietokoneen näytölle palkka, työaika ja loma tietoja. Nyt yritys voi tarkastella vaikka päivittäin, mitä ja miten kauan työntekijät ovat työskennelleet. Työn kohdistaminen on helpompaa ja tarkempaa laskentatoimen tarpeisiin. Näin yritys saa helposti tarpeellista tietoa palveluidensa hinnoitteluun.

Tällä hetkellä käyttäjänä on vain yrityksen laskentatoimesta vastaava henkilö, mutta jatkosuunnitelmissa on kehittää autoihin ja autohallille ohjelmaversio. Tämä mahdollistaisi kuljettajien suoran työtuntien tallentamisen tietokantaan. Näin laskentatoimelle jäisi vain palkan ja sen lisien laskeminen palkkahallinnosta ja työajan seuranta. Lisäksi kehittämisen alla on suunnitelmat tietokannan hakujen lisääminen ja yhdistely. Tätä varten on suunnitteilla uusi näkymä, jossa käyttäjä antaa enemmän hakuehtoja ja ohjelmisto rakentaa haut ja tulostaa tuloksen.

Ohjelmiston tekeminen tuntui ajatuksena suuritöiseltä, mitä se olikin, ja haastavalta. Vähäisen ohjelmointikokemuksen ja tietokantatietämyksen vuoksi joutui monille tehtäville etsimään ratkaisua kirjoista ja Internetistä. Lisäksi Visual Basic . NET on muuttunut niin paljon edeltäjästään, että projektia varten tuli opiskella se uudestaan. Nyt kokemus on tätä harjoitusta rikkaampi ja asioihin ottaa kantaa eri tavalla. Tiedon etsimisen taito on lisääntynyt ja esivalmistelun merkitys on selvinnyt töitä tehdessä. Sanonnalle ”hyvin suunniteltu on puoliksi tehty” on avautunut uusi merkitys.

LÄHDELUETTELO

1 Hovi, Huotari, Lahdenmäki. Tietokantojen suunnittelu & indeksointi. 1. painos kesäkuu 2003. Jyväskylä: Docendo Finland Oy, 2003. ISBN: 951-846-178-3.

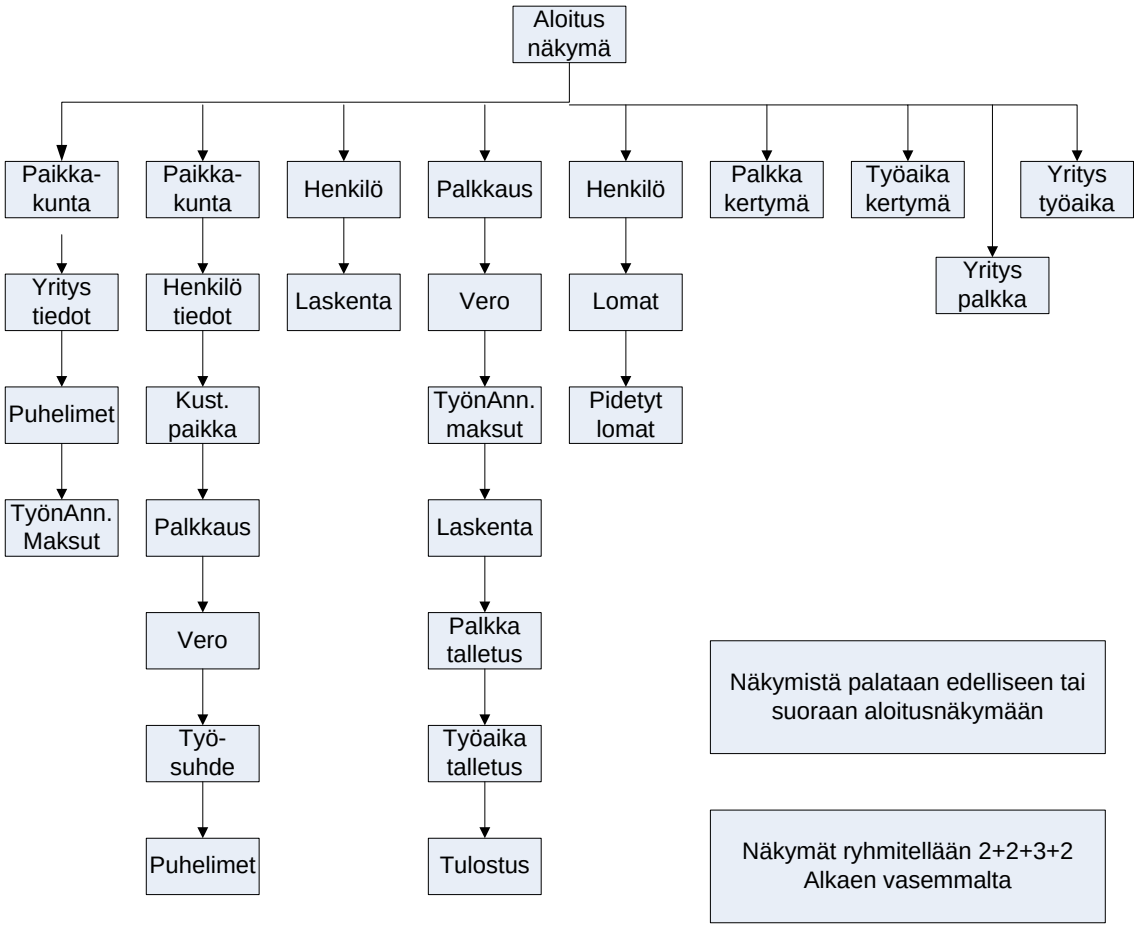
2 Haikala I, Märijärvi J. Ohjelmistotuotanto. 9. painos. 2002 Talentum Media Oy. ISBN: 952-14 -086-8.

3 UML-MALLINNUS [WWW-dokumentti] http://www.media.hut.fi/~as75112/luennot_14022002_TMJ.pdf (12.3.2005)

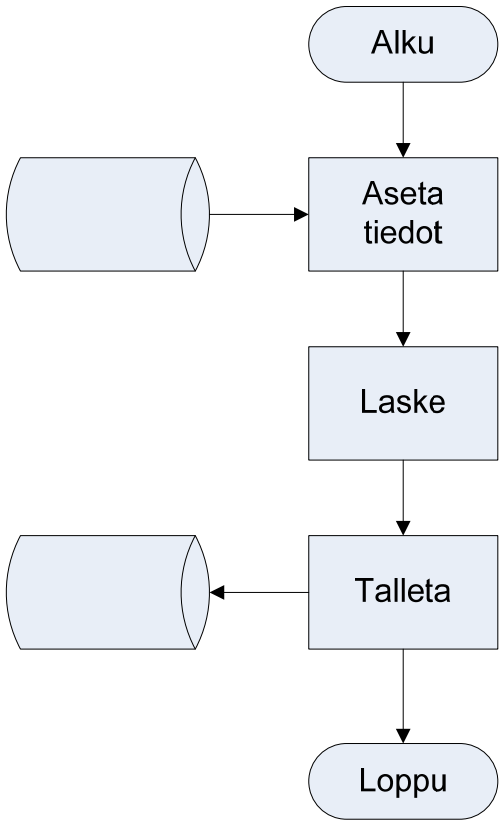
4 Halvorson, M. Microsoft Visual Basic .NET TRAINER KIT. 2. painos. IT Press 2002. ISBN: 951-826-653-0

5 Hovi, A. SQL-opas. 1. painos marraskuu 2004. Docendo Finland Oy. ISBN: 941-846-228-3.

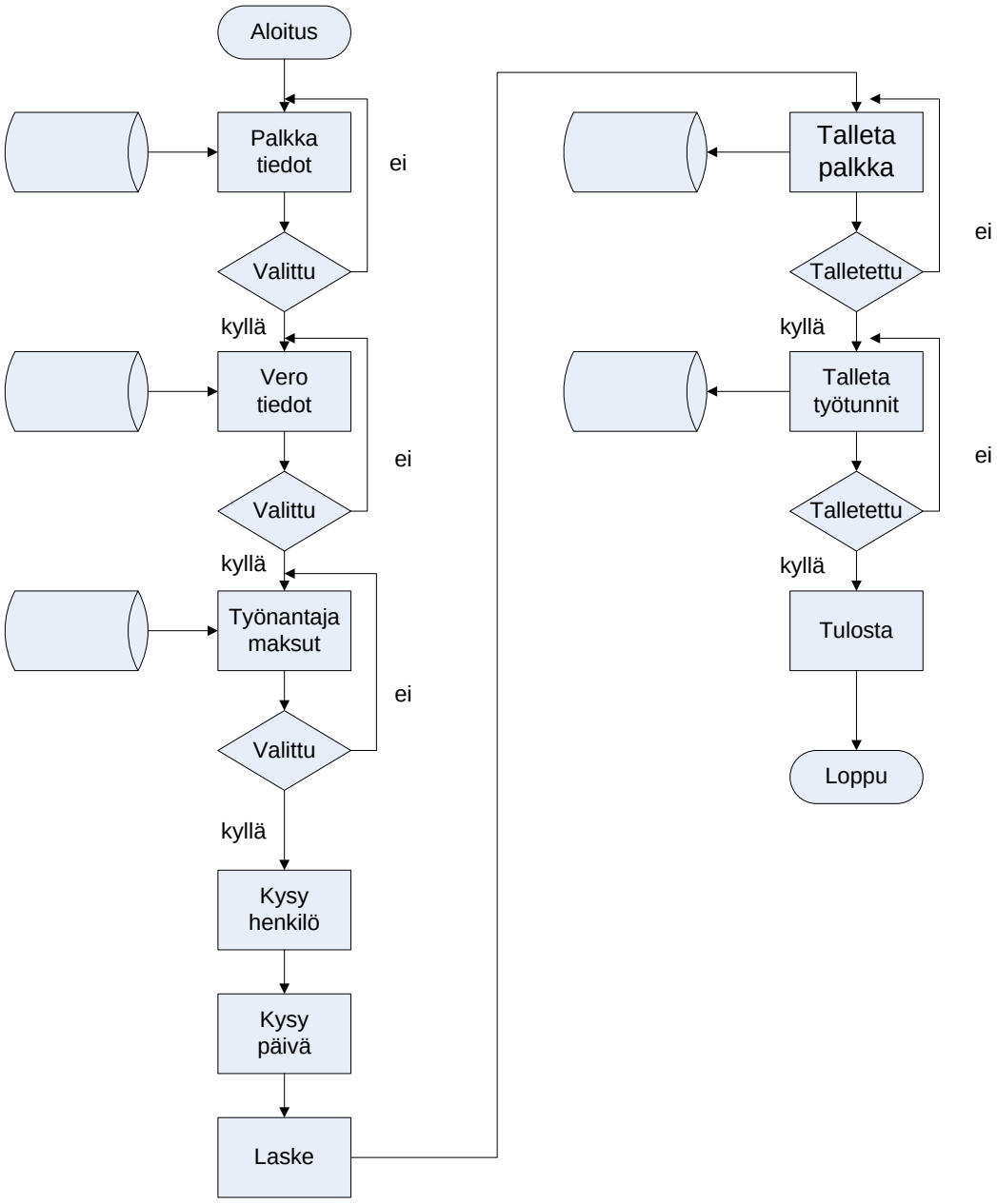
Käyttöliittymän näkymiä kuvaava vuokaavio



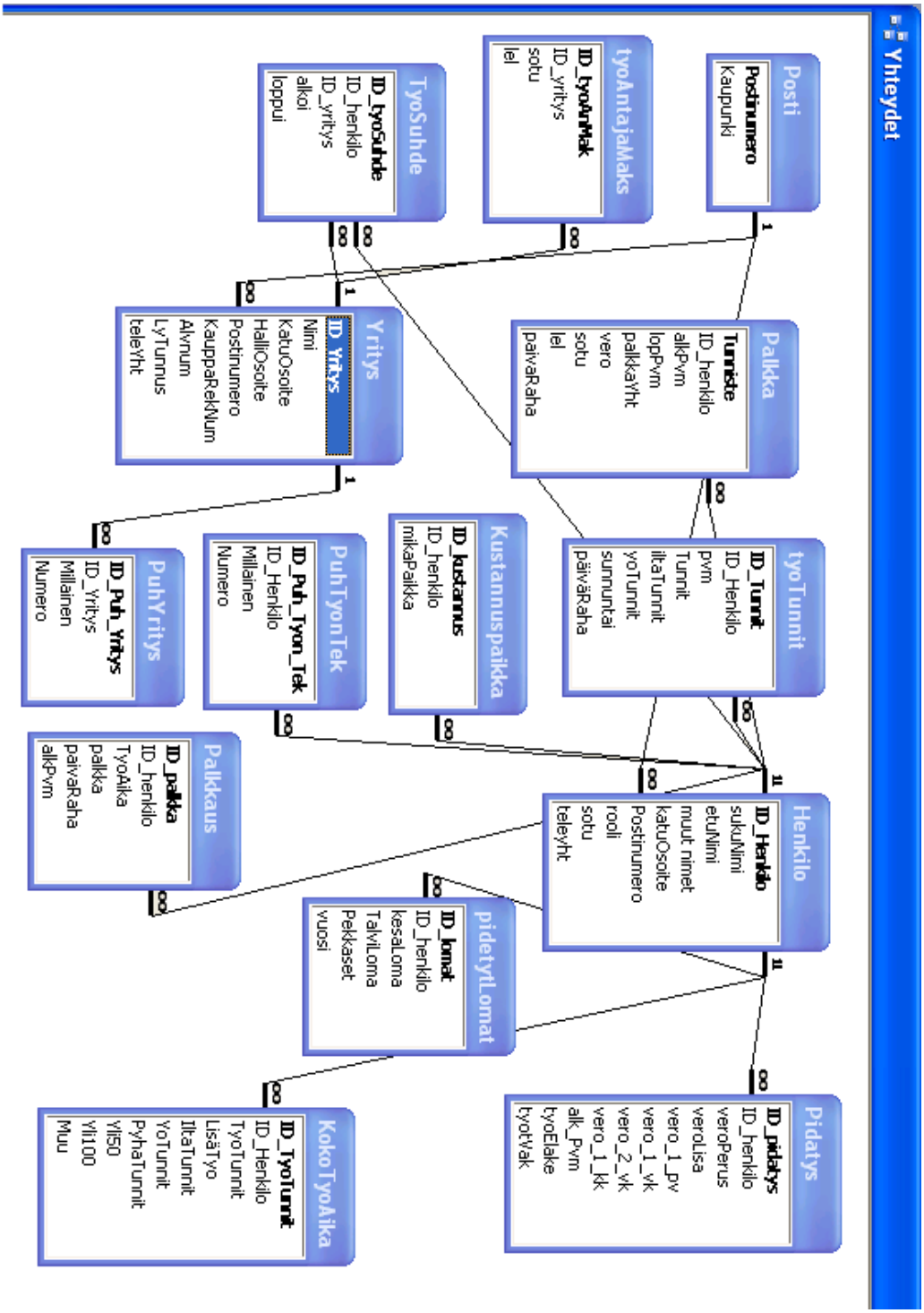
Työtuntien laskentaa kuvaava vuokaavio



Palkan laskentaa kuvaava vuokaavio



Tietokannan taulut ja niiden väliset yhteydet



RaiPe Henkilöstöhallintaohjelmisto

TESTAUSTRAPORTTI

Testaaja:_____

Asema:_____

Testauksen suoritus pvm:_____

Testauksen tulos: ___ OK ____ Virheellinen

Ohjelmiston käytettävyys:_____

Ohjelmiston selkeys:_____

Näkymien selkeys:_____

Tiedon esittämistapa:_____

Tiedon riittävyys:_____

Tulosteet:_____

Kehitysideat:_____

_____

_____

_____

Yleisarvosana:_____

Muuta:_____

_____