TAMPEREEN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma
Jukka Kaupinmäki

Opinnäytetyö

Tietokannasta riippumattoman sovelluksen tarpeellisuus

Työn ohjaaja: Paula Hietala, FL
Työn tilaaja: KPM-Engineering Oy; Martti Laurila, ICT-manageri, DI
Tampere X/2009

TAMPEREEN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Tekijä(t)	Jukka Kaupinmäki
Työn nimi	Tietokannasta riippumattoman sovelluksen tarpeellisuus
Sivumäärä	37(40)
Työn valmistumiskuukausi ja vuosi	29/5/2009
Työn ohjaaja	Paula Hietala
Työn tilaaja	KPM-Engineering Oy; Martti Laurila

TIIVISTELMÄ

Esittelen perinteiseen vesiputousmalliin pohjautuvia ohjelmistomenetelmiä, joilla voidaan toteuttaa tietokannasta riippumaton sovellus. Työ on kuitenkin toteutettu paremminkin ketterästi havaintoesitysohjelmoinnin periaatteella, joten käyttäjät ovat voineet puuttua erilaisiin ongelmakohtiin ohjelman versioita kehitettäessä.

Sovellus on tarkoitettu rakennussuunnitteluyrityksen käyttöön. Sovelluksella suunnittelijat sekä yrityksen asiakkaat voivat selailla rakennuksiin asennettavien naulalevyristikoiden eli kattotuolien tietoja ja kuvia ja laskea niiden hinta-arvioita.

Toteutin aluksi Accessilla tietokantasovelluksen, joka ei kuitenkaan ole tietokannasta riippumaton. Sen vuoksi aloin suunnitella ja toteuttaa varsinaisen sovelluksen havaintoesitystä Visual Studiolla. Toteutin säikeistystekniikalla ohjelman, joka siirtää tekstitiedostossa olevat tiedot tietokantaan. Sovellus sisältää myös abstraktin tehdasjärjestelmän, jolla voidaan valita oikea tietokanta koneella olevista vaihtoehdoista. Sovellus käyttää DataGridView-luokkaa tietokannan mallintamiseen näytölle. Sovelluksessa on myös rajaus- ja hakutoimintoja naulalevyristikoille.

Sain havaintoesityksen toimimaan ODBC- ja OLE-pohjaisesti. Havaintoesityksen toteutus vei kuitenkin suunniteltua pitemmän ajan ja käytössä havaintoesitys oli myös epäkäytännöllisen hidas, joten varsinaisen sovelluksen toteuttaminen keskeytettiin. Sovelluksella saattaa olla tulevaisuudessa kuitenkin käyttöarvoa ja kehitystyötä voidaan jatkaa, kunhan tietokannat nopeutuvat.

Avainsanat	tietokanta, ADO, ohjelmointi, tietokantariippumaton, tuottajat, rajapinnat, abstraktit tehtaat

TAMK University of Applied Sciences
Business Information Systems

Writer(s)	Jukka Kaupinmäki
Thesis	Planning Database Independent Software
Pages	37(40)
Month and Year of Completion	29/5/2009
Thesis Supervisor	Paula Hietala
Co-operating Company	KPM-Engineering; Martti Laurila

ABSTRACT

For this work I have carried out research on programming database independent software. I introduce the programming methods, and how to write database independent software. I will introduce this by using the traditional waterfall-view. In practice, however, I have used the agile demo-construction method, so that users can be involved with countering any problems.

The Software is designed to be used in a structural engineering company. With the help of the software, a user can browse a roof chair's parameters and pictures and calculate its estimated cost.

In the beginning, I produced database software with Access, which is not database-independent. Because of that, I started to design and produce a demo-program with Visual Studio. I produced a threading program that transfers information from a text-file into the database. The software also includes an abstract factory system through which the user can choose the right database from the options which are stated in the computer's screen. The software uses DataGridView-class for modelling a database to the user's display. The software also includes cropping and search functions for roof chairs.

I managed to make the demo-software work with ODBC and OLE-based database systems. Writing the demo-software took more time than I had planned and it was too slow. So the production of the actual software producing had to be halted. The software could be usable in the future, however, and its development can be continued once databases become faster.

Keywords: database, ADO, programming, database-independent, providers, interfaces, abstract factories

TAMPEREEN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma
Jukka Kaupinmäki

Sisällysluettelo

Sanasto

Suomeksi	**Selitys**	**Englanniksi**
Tietokannasta riippumaton sovellus	Sovellus, joka ei toimiakseen tarvitse mitään erityistä tietokantaa.	Database/provider-independent
SQL-standardi	ANSI:n ylläpitämä yhteisen tietokantakielen malli.	SQL standard
ODBC-rajapinta	Microsoftin toteuttama järjestelmä, joka palvelee sovellusten ja tietojenkantojen välistä yhteydenpitoa.	ODBC interface
OLE-tuottaja	Microsoftin toteuttama järjestelmä, joka tuottaa sovellusten ja tietojenkantojen väliselle yhteydelle tiedonviestintäratkaisuja.	OLE provider
Tietokantaolio	Olio, jota tietokantaohjelmistot käyttävät mallintaessaan tietokannan tietuetta.	Database-object
Mfa- ja mfc-linkitysjärjestelmät	Tahdistavat ulkoisten ohjelmien toimimisen Visual Studio -ympäristössä.	Mfa- ja mfc linkage systems

1 Johdanto

1.1 Tausta

Olin 25.4.–30.9.2008 työharjoittelijana KPM Engineering OY -nimisessä rakennussuunnitteluyrityksessä. Yrityksessä suunnitellaan muun muassa betoni-, teräs- ja puurakennuksia sekä naulalevyristikoita. Naulalevyristikot ovat yleensä omakoti- tai rivitaloihin asennettavia valmiita kattoristikoita, jotka kootaan tehtailla asentamalla kattoristikon puuosat erityiseen kehikkoon. Puut painetaan kiinni toisiinsa naulalevyiksi kutsutuilla metallisilla levyillä, joissa on valmiiksi naulat. Naulalevyjä on monenkokoisia eri käyttötarkoituksiin. Naulalevyt nopeuttavat kattoristikoiden kokoamista, kun jokaista naulaa ei tarvitse vasaroida puuristikkoon erikseen.

Työharjoittelussa tehtävänäni oli suunnitella ja toteuttaa työnantajalleni sovellus, jolla voi selata naulalevyristikoiden tietoja ja kuvia. Lisäksi ohjelmalla voidaan laskea naulalevyristikoiden työ- ja raaka-ainekustannuksia. Ohjelma on tarkoitettu sekä työnantajani että tämän asiakkaiden käyttöön niin, että työnantaja pystyy selaamaan kaikkia kohteita mutta asiakkaat omien kohteidensa lisäksi vain ICT-päällikön määrittelemää pohjatietokantaa, joka sisältää asiakkaiden perustöitä ilman yksityiskohtaisia kohdetietoja.

Alkuperäinen työ tuli tehtäväksi Access 2007 -ympäristöön käyttäen sen sisäistä VBA-skriptausta. Myöhemmin vertaisohjelmoija ehdotti, että ohjelma voisi toimia muillakin tietokanta-alustoilla kuin vain Accessissa.

Työnantajallani ja tämän asiakkailla oli ohjelmasta ennestään käytössä DOS-pohjainen Paradox-tietokantasovellus, joka käytti naulalevyristikkokuvien katseluun erillistä ohjelmaa. Työnantajani halusi päivittää ohjelmansa nykytekniikan tasolle eli siirtyä komentorivipohjaisesta käyttöliittymästä graafiseen käyttöliittymään. Lisäksi työnantajani halusi päivittää silloista ohjelmaansa. Vanhan järjestelmän tietokantakapasiteetti oli käytetty loppuun eikä se siksi enää vastannut käytännön tarpeita.

1.2 Tavoite

Tarkoituksena oli siis aluksi tutkia, oliko ylipäätään järkevää tehdä ohjelmaa kustannustehokkaasti. Miten ohjelman voisi tehdä edullisesti, ja kuinka toimiva tietokannasta riippumaton sovellus olisi? Tietokannasta riippumattoman sovelluksen etuna olisi, että asiakas saisi vapaasti valita, mille alustalle tietokantansa sijoittaisi, ja voisi siten käyttää hyödykseen jo käytössään olevia tietokantaratkaisuja. Koska sovellus on vapaa ulkoisesta tietokantasovelluksesta, ohjelmoijan ei tarvitse ottaa huomioon tietokantasovelluksen yksityiskohtaisia tarpeita tai muutoksia tietokantasovellusten kehittyessä.

Sovelluksen tulisi pystyä käsittelemään isoja tietokantoja suhteellisen vaivattomasti. Koska ohjelma on tarkoitettu naulalevyristikkoja kokoaville rakennusalan yrityksille, jotka eivät yleensä ole tietotekniikan ammattilaisia vaan talonrakentajia tai rakennusalan alihankkijoita, tulisi ohjelman olla asiakasystävällinen ja helppo käyttää. Ohjelman tulisi toimia suhteellisen vaivattomasti nykylaitteissa, joten ohjelman sisällä suoritettavien tehtävien vasteajan tulisi olla mahdollisimman pieni.

Ohjelman on tarkoitus toimia kolmitasoarkkitehtuurilla, jossa tietokantaosio on yhdistetty Visual Studion C++ -rajapinnan läpi pääohjelmaan. Tietokanta on ohjelmassa mallinnettu olioiden avulla. Looginen osio sisältää ohjelman toiminnot ja toimii mallinnuksen, tietokannan ja graafisen ympäristön välisenä tiedonviejänä.

1.3 Menetelmät

Tietokannan suunnittelu olisi tarkoitus aloittaa alusta. Vanhan naularistikko-ohjelman yhden taulun järjestelmästä pitäisi päästä moniulotteiseen relaatiotietokanta-järjestelmään. Samalla tutustun Visual Studion C++ -ohjelmointiin ja sen sisältämään palveluun tietokannasta riippumattomille alustoille. Tarkoitukseni on lukea tietokannasta riippumattomiin sovelluksiin liittyvää aineistoa, jota tuntuu suoraan tällä nimikkeellä olevan saatavilla todella heikosti.

Tehtäväni on kartoittaa ohjelman vaatimusmäärittely ja tehdä sen pohjalta havaintoesityssovellus. Havaintoesityksen testauksessa voidaan käyttää hyödyksi yrityksen tulevia käyttäjiä ja asiakasyhteyshenkilöitä.

2 Vaatimusmäärittely

2.1 Aikaisemmat sovellukset

NR-Mikro

NR-Mikro-sovelluksen on suunnitellut ja toteuttanut nykyinen ICT-päällikkö Martti Laurila vuonna 1992. Sovellus on ohjelmoitu Paradox-Pal-kielellä. Paradox-Pal sisältää omaan työpöytäympäristöön toteutettavan sisäisen tietokantaympäristön. Pal on Pascalista syntyisin oleva funktio-pohjainen tehokkuutta vaativa ohjelmointikieli. Paradox-tietokantajärjestelmä ei ole SQL-pohjainen vaan se on itsenäinen tietokantajärjestelmänsä.

Sovelluksen perusnäkymiä ovat Selaus- ja Lomake-näkymät. Selausnäkymä näyttää kaikki ristikkokohteet riveittäin listattuna. Rivillä näkyy ristikkokohteen tiedot omiin sarakkeisiinsa jaoteltuna. Selaus-näkymässä on pikahakutoiminto, joka hakee valittuna olevan sarakkeen kentästä syötettyä hakuarvoa. Lomake-näkymässä kohteen tiedot on aseteltu samalle sivulle. Lomake-näkymä sisältää erikseen avattavan hintaosion, joka laskee kohteelle kokonaishinnan. Liitteessä 1 on kuva NR-Mikron Selaus-näkymästä.

Sovelluksen hakunäkymiä ovat Kysely- ja Erikois-näkymä. Kysely-näkymässä on määritelty ristikkokohteiden yleisimmin käytettyjä rajausehtoja riveittäin. Erikois-näkymässä kohteen kaikki tiedot ovat sarakkeittain rajattavana.

NR-Mikro-sovelluksessa on oma kaavionpiirto-ohjelma, joka näyttää puutavaran, naulalevyjen ja menekin sekä yhteiskustannukset. Sovelluksessa on toiminto, jolla voi laskea ristikoiden keskiarvomenekit. Sovelluksen muita toimintoja ovat raporttien tulostukset ja tietokannan päivittäminen.

NR-Mikro käyttää kuvien katseluun erillistä NR-Kuvat-piirto-ohjelmaa. NR-Mikro lähettää piirto-ohjelmalle tarvittavat koordinaatit kuvista ja piirto-ohjelma piirtää koordinaattien mukaan kuvan näytölle. Liitteessä 2 on kuva NR-Kuvat-ohjelmasta.

2.2 Ongelmat

Käyttäjän ongelmat

Tilaajan käyttämä vanha järjestelmä oli DOS-pohjainen sovellus, jossa ei ollut graafisuutta eikä hiiritukea. Asiakkaat eivät ole tietotekniikan ammattilaisia, ja graafisella järjestelmällä sovellus saadaan vastaamaan nykyaikaisia odotuksia.

Lisäksi tietokanta ei enää vastannut tietovaatimuksiltaan käyttäjien tarvetta, vaan se kuvasi puutteellisesti naulalevyjen ja puutavaroiden yksityiskohtia. Naulalevyistä oli vanhaan ohjelmaan laskettu neliökoon yhteissumma ja kappalemäärä, mutta yksityiskohtaiset nimet ja kokotiedot puuttuivat. Puumateriaalin osalta vanha ohjelma kertoi, kuinka monta kuutiota puuta kohteeseen tarvittiin nimellisesti ja kuinka paljon sitä tarvittiin todellisuudessa, kun mukaan laskettiin myös puutavarasta sahattaessa ja höylättäessä irtoava aines. Vanhassa ohjelmassa oli jokaiselle puutavaran lujuudelle oma rivinsä, jossa oli laskettuna kunkin lujuusluokan kuutiomäärä erikseen.

Vanha Paradox oli nopea käymään läpi kuvia ja tekemään raportteja. Accessilla tekemäni sovellus toimii kuitenkin hitaasti kuvienkäsittelyohjelman kanssa. Lisäksi monitasoisten kyselyiden käsittely hidastaa tietokantaa liikaa.

Kuvien katseluun käytetty erillinen ohjelma ei toimi interaktiivisesti pääohjelman kautta, vaan saa tiedot pääohjelmalta käynnistettäessä ja lähettää tietoja sammutettaessa.

Ohjelmoinnin ongelmat

Vanhoista skripteistä, jotka on kirjoitettu Paradoxilla, Visual C++ 6.0:lla ja Visual Basicilla, ei ole tehty kunnollisia dokumentteja. Skripti on funktiopohjaista. Ohjelmointia ei ole tehty oliopohjaisesti paitsi C++:ssa osittain käyttäen Visual Studion omaa vanhaa graafista MVC (Model View Controller) -kehystä kuvien selaukseen.

Ohjelman päivittäminen Accessissa tuntui olevan todella hankalaa. Toki Accessiin on ladattavana oma Development Kitinsä. Tämä on tarkoitettu helpottamaan ohjelmoijaa tekemään Access-sovelluksesta oman irrallisen ohjelman, joka ei tarvitse koko Access-ohjelmaa toimiakseen, vaan toimii vapaasti levitettävän lukijaohjelman eli Run-Time-ohjelman kautta. Development Kitissä on valmiina jonkinlainen päivityksen hallintajärjestelmä, joka osaa tehdä Run-Time-ohjelman kautta toimivan oman ohjelmansa, mutta ei sisällä päivityksen muutoshallintaa. Accessiin pystyin toki ohjelmoimaan tietokantapäivityksen lukemalla tiedostolistan tietokantapäivityksistä, mutta tässä tapauksessa muutoksen hallinta on edelleen olematonta varsinaisen ohjelman päivityksiin.

2.3 Sisällölliset vaatimukset

Sovelluksen sisällölliset vaatimukset ovat määrittyneet kolmessa vaiheessa: Perustavoitteena oli sisällyttää sovellukseen kaikki edelliseen versioon sisältyneet

ominaisuudet. Projektia käynnistäessäni ICT-päällikkö asetti kehitystavoitteita. Kehitystyön aikana asiakkaat esittivät lisätoivomuksia**.**

Perustavoitteet

Asiakkaat voivat säätää tietoja naulalevyjen ja puutavaroiden hinnoista sekä työkustannuksista. Asiakkaan tulee voida määritellä omat asennustyökustannuksia koskevat funktionsa, joilla lasketaan kokoonpano asete ja naulalevysarjan valmistamiseen kuluvaa työkustannusaikaa. Funktiot voivat olla riippuvaisia muun muassa naulalevyristikon tyypistä, alapaarteen pituudesta ja tietysti naulalevyjen määristä.

Alapaarre on rakennusalan termi, joka tarkoittaa naulalevyristikkotuolin alaosaa. Alapaarteen pituus vastaa naulalevyristikon alaosan pituutta leveyssuunnassa (Erkkilä & Metsäranta 1989: 5). Ohjelman tulee piirtää kaavioita, jotka voivat olla käyttäjän rajaamia kohteita ja alapaarteen pituuden mukaan järjestetty. Sovelluksessa kaavioita voidaan piirtää puutavaramenekistä, naulalevymenekistä, puutavarakustannuksesta, naulalevykustannuksesta, asennustyökustannuksesta ja yhteiskustannuskaaviosta.

Ohjelman tulisi toteuttaa toivotut ohjelmanäkymät. NR-Mikron selausnäkymässä kaikki ristikkokohteet on lajiteltu ja niistä ovat näkyvissä tarvittavat tiedot. Kullakin kohteella tulisi olla oma tuotenäkymä, josta näkyvät kohdekohtaiset tiedot. Lisäksi kohteella on puutavaroista ja naulalevyistä tarkemmat tietonäkymänsä.

Jokaisella asiakkaalla on erillinen tietokanta omista kuvistaan sekä yhteinen tietokanta yrityksen määrittelemistä yleisistä kuvista. Tämä asettaa vaatimuksia tietokannan päivitysten hallinnalle. Yritys toisaalta pääsee käsiksi kaikkien asiakkaiden tietokantoihin, mikä asettaa tietokannan kapasiteetille tiukat vaatimukset.

Muita perustavoiteotteita sovellukselle oli perushakufunktio, rajaushaku ja ristikkokuvienkatselu. perushakufunktio voi hakea tietoa eri sarakkeista, ja yksityiskohtaisempi hakusivu, jolla voi rajata selauksessa näkyvien kohteitten määrää. Rajaushaku vaikuttaa kaavioitten ja kustannusten yhteenvetojen tuloksiin. Rajausarvoja tulisi voida tallentaa muistiin. Ohjelman naularistikkokuvien katselut ovat käyttäjäkohtaisia.

Kehitystavoitteet

Paradox-pohjainen sovellus on tietokantaan täysin sidottu, samoin kuin harjoittelussa tekemäni Access-sovellus. Sovelluksen toivottiin kuitenkin toimivan yrityksen asiakkaiden omilla tietokantajärjestelmillä.

Naularistikko kuville haluttiin kaksi kehitystavoitetta. Ensiksi naularistikkokuvien selailu vaikuttaa tietokannan ristikkokohteen tietoihin. Toiseksi naularistikon piirto-ohjelma kuuluu osana lopulliseen sovellukseen.

Lisätoivomukset

Ohjelmassa tulee olla hintalaskuri, jolla kyetään poimimaan tiedot aiemmasta tilauksesta ja muokkaamaan niitä käyttäjäkohtaisesti. Eräs yrityksen asiakas toivoi tätä ominaisuutta Access-havaintoesitysohjelman käyttöönoton yhteydessä.

Lomake- ja Selausnäkymän yhtenäistämistä samalle sivulle pidettiin hyvänä ratkaisuna Access-ohjelmassa. Lisäksi toivottiin kuvanpiirtonäkymän yhdistämistä samaan näkymään.

Rajaushakulauseeseen haluttiin oma umpiräystäsrajaus nopeuttamaan tällaisten hakujen tekemistä. Periaatteessa ristikko on umpiräystäs silloin, kun ristikon räystään pääty ja alapaarre ovat kiinni toisissaan. Käytännössä umpiräystään määrittelemiseksi käytetään kaavaa: Jänneväli <= Alapaarteen pituus – 500 mm.

3 Suunnittelu

Ensimmäinen tekninen vaihe ohjelmistotuotannossa on suunnittelu, joka koostuu arkkitehtuurin suunnittelusta, yksityiskohtien suunnittelusta ja perussuunnittelusta. Arkkitehtuurin suunnittelu sisältää ohjelmiston osituksen, kontrollisuunnittelun ja rajapintasuunnittelun. Yksityiskohtien suunnittelu sisältää tietosuunnittelun, algoritmisuunnittelun ja käyttöliittymäsuunnittelun. Viimeisessä suunnitteluvaiheessa käydään läpi komponentteihin, rinnakkaisuuteen, resurssien hallintaan ja tietoturvaan liittyvät asiat.

3.1 Arkkitehtuuri

Ohjelmistoarkkitehtuurin tarkoituksena on kuvata ohjelman komponenttien ja aliohjelmien välisiä suhteita monesta näkökulmasta. Komponentit voivat olla mitä tahansa ohjelmoinnissa mallinnettavia asioita. Näkökulmat taas keskittyvät tietyn arkkitehtuurikuvauksen abstraktiokerroksen toiminnallisten ja ei-toiminnallisten vaatimusten ratkaisemiseen. Abstraktiokerroksen ratkaisumalli kuvaa tietynlaiselle järjestelmälle tehtyä ratkaisumallia. Se voi toimia järjestelmäperhettä yhdistävänä arkkitehtuurikuvauksena tai tarkentaa yhden järjestelmän kuvausta. (Koskimies 2005: 31–32.)

Arkkitehtuurinäkymät

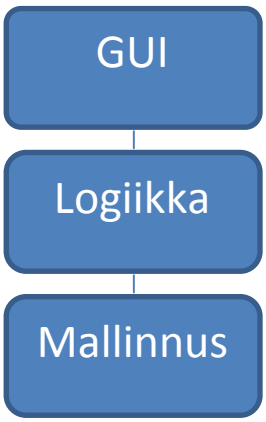
Kruchtenin (1995) esittämä 4+1 -näkymämalli jakaa suunnittelun viiteen näkymään: loogiseen, prosessoivaan ja fyysiseen näkymään sekä komponentteihin ja käyttötapauksiin liittyvään näkymään. Looginen näkymä kuvaa ohjelman toiminnallisuuden toteuttamista. Prosessinäkymä kuvaa ei-toiminnallisia asioita ja prosessien etenemistä. Fyysinen näkymä kuvaa laitteiden fyysistä sijoittelua. Komponenttinäkymä kuvaa komponenttien riippuvuuksia toisistaan; sitä käytetään erityisesti työn osittamisessa. Käyttötapausnäkymän puolestaan on tarkoitus yhdistää muut näkymät yhteen kuvaamalla käyttäjän tilaa ohjelmassa. (Lindell 2008: 5–6.)

Kerrosarkkitehtuuri

Kerrosarkkitehtuuri on yksi arkkitehtuurityylin ilmentymä. Buschmannin & al. (1996) mukaan arkkitehtuurityyli on korkeimman tason esitetty suunnitelma. Kerrosarkkitehtuurissa komponentit ryhmitellään käyttösuhteiden mukaan eri tasoihin. Kerrosarkkitehtuurilla voidaan kuvata ohituksia ohjelman loogisten komponenttien välillä ja ulottuvuuksia kerrosten sisällä. Ohittamisessa ylempi kerros ottaa yhteyttä

suoraan alimpaan kerrokseen, käymättä läpi välissä olevia kerroksia. Kaaviokuviossa 1 käytetään tyypillistä leipäviipalekerrosarkkitehtuuria, jossa ei ohituksia ole esitetty. Kaavio kuvaa vain loogista ulottuvuutta eikä sisällä erikoistamisulottuvuutta, jolla kuvataan tietyn tason yksityiskohtaisempia toimintoja.

Koskimies (2005) esittää, että kerrosarkkitehtuuria voidaan soveltaa kaikissa järjestelmissä. Arkkitehtuuri jakaa järjestelmän korkealla tasolla osiin, joiden muodossa järjestelmää on helpompi ymmärtää. Kerrosarkkitehtuuri ohjaa vähentämään riippuvuuksia suunnittelusta, sillä ylemmät kerrokset rakentuvat alempien varaan. Tämä tuo järjestelmään ketteryyttä. Ongelmana on tehokkuushäviö, kun joudutaan käyttämään alimman osion toimintoja ylemmän tarpeisiin. (Koskimies 2005: 130–131.)



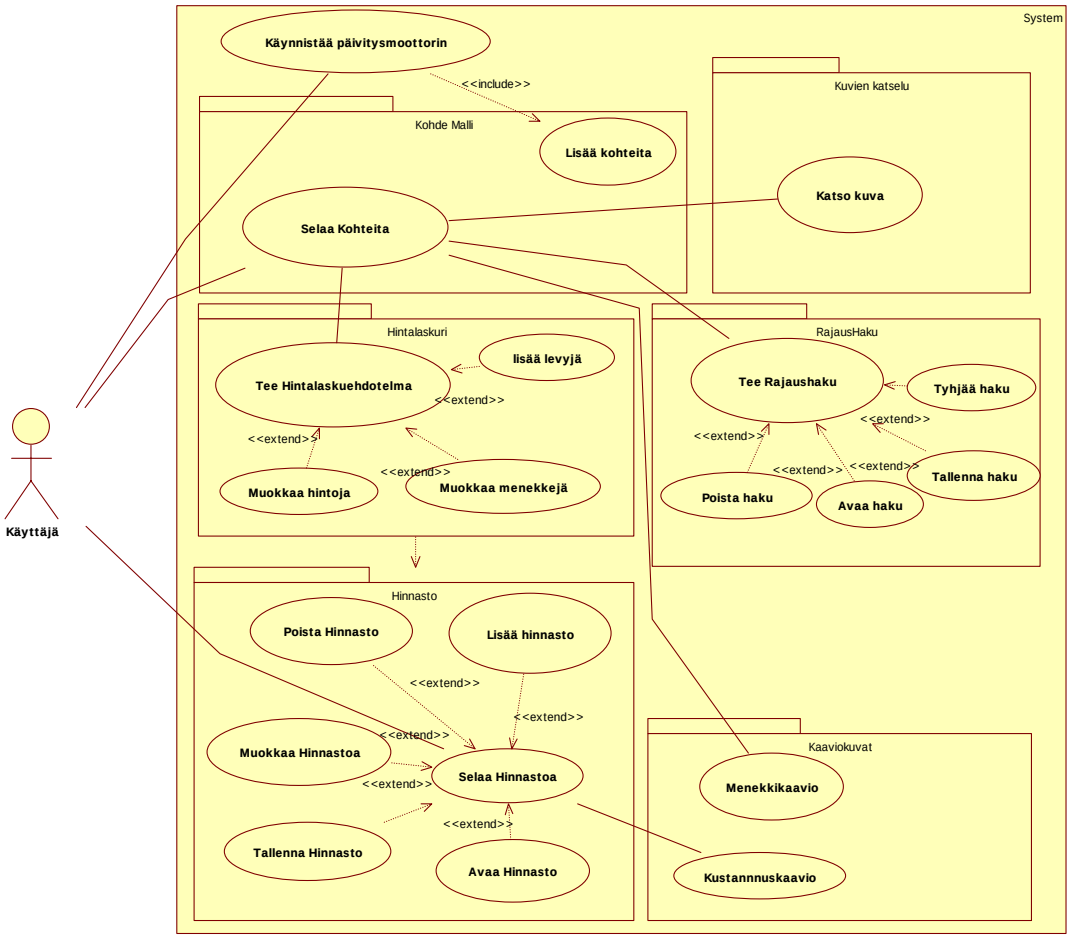
Kuvio 1. Arkkitehtuuri

3.2 Kaaviot (4+1 -näkymämalli)

Käyttötapauskaavio

Käyttötapauskaavion on tarkoitus havainnollistaa vaatimuksenmäärittelyssä esiin tulleiden ongelmien ratkaisua käyttäjän näkökulmasta ja esittää ratkaisua siihen. Lähdin kuvion 2 käyttötapauskaaviossa jaottelemaan varsinaisen pääohjelman aliosioihin. Aliosioissa luettelen niiden tarvitsemat suuntaa antavat toiminnot ja niiden yhteydet toisiinsa.

Ryhdyn tarkastelemaan kaaviossa käyttäjän kannalta oleellisia toimintoja. Jaan käyttäjän polun kolmeen päähaaraan. Ensimmäisessä haarassa käyttäjä voi päivittää tietokantaa lisäämällä siihen paketteja.



Kuvio 2. Käyttötapauskaavio

Toisessa haarassa käyn läpi pääohjelman kohteitten lukutoiminnon. Koska kuvien haku haluttiin kehitysneuvottelujen yhteydessä osaksi tätä pääominaisuutta, tulee se sisällyttää eikä linkittää tämän ominaisuudeksi. Selaa Kohteita -toiminnosta päästään Hintalaskuri-, Rajaushaku- ja Kaavio-osioiden omiin toiminnallisiin haaroihin. Hintalaskuriehdotelman teko sisältää puutavaran menekkien, naulalevyjen lisäyksien ja eri hintojen muokkaustoiminnot. Hintalaskuri käyttää valitun hinnaston hintoja hyväkseen tehdessään oletuskaavakkeen käyttäjälle. Kuvion 2 Tee Rajaushaku -toiminnolle olen määritellyt lisäominaisuuksia, joilla voidaan tallentaa hakuehdot, avata tallennettu haku, poistaa tallennettu haku tietokannasta ja tyhjentää nykyiset haut yksittäisestä tietueesta.

Kolmannessa haarassa käsittelen hinnaston eri toimintoja. Hinnaston selauskäyttäjä-toimintoon lisään toiminnoiksi tietojen lisäämisen, poistamisen ja muokkauksen. Lisäksi hinnastot voidaan tallentaa erilliseen varmuuskopiotiedostoon, ja varmuus-kopiotiedostosta voidaan avata hinnastoa koskevat tiedot.

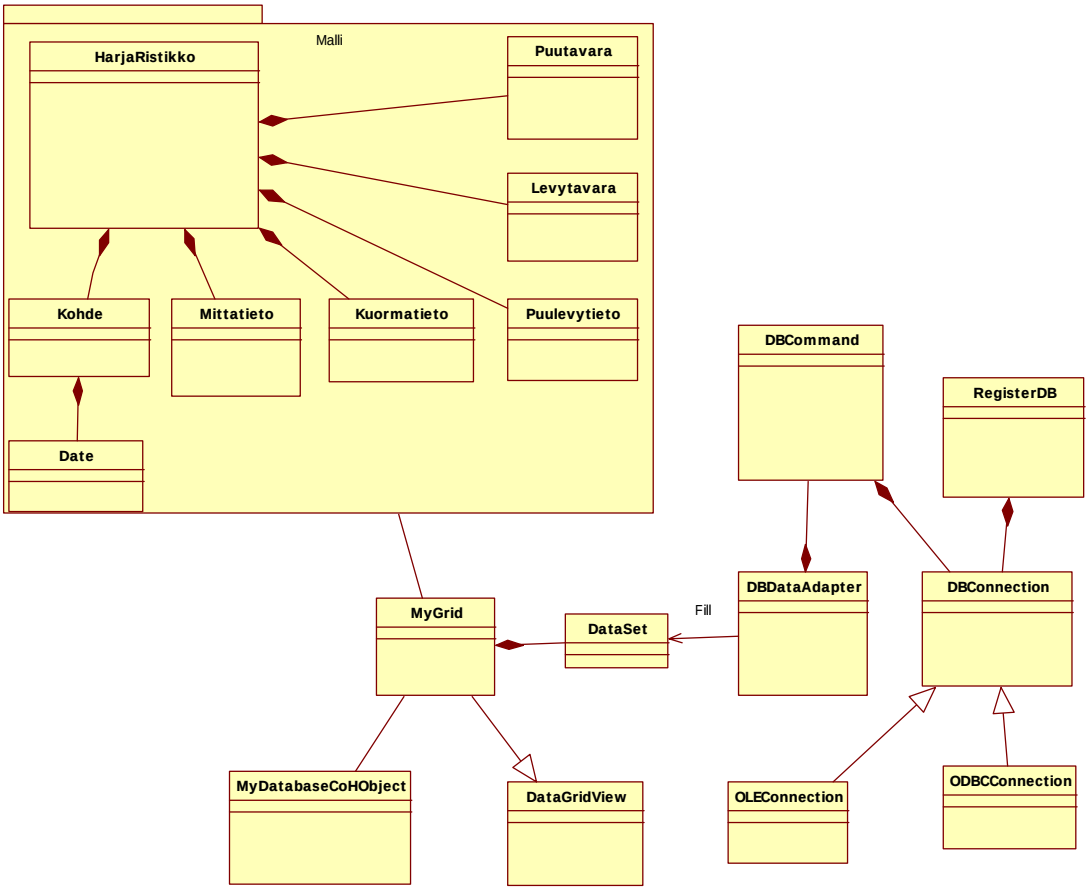
Komponenttikaaviossa aihealueet voidaan jaotella eri kansioihin. Komponenttikaavio on tarkoitettu ohjelmoijille jakamaan työalueet itsenäisiin osiin. Komponenttikaaviossa kansiot kuvaavat ohjelman eri osien alueita. Kuviossa 2 olen liittänyt käyttötapaus-kaavioon kansiot osaksi järjestelmää. Komponenttikaavion tulee myös näyttää eri osien suhteita toisiinsa. Kaaviossa tämä on toteutettu katkoviivanuolilla.

Luokkakaavio

Luokkakaaviolla kuvataan ohjelman loogista näkymää. Havaintoesitystoteutuksessa luokkakaavion naularistikon mallia ei suoranaisesti hyödynnetä, vaan ohjelma lataa tiedot taulukkoruutuun suoraan tietokannasta. En ole vielä päättänyt, miten se toteutetaan lopullisessa versiossa. Kuvion 3 luokkakaavio on jaoteltu kahteen osaan.

Ensimmäinen osa kuvaa ristikon mallinnusta sovelluksessa. Mallinnuksessa olen jakanut ristikon alempiin olioihin sen perusteella, minkä tyyppistä tietoa se sisältää. Kohde-luokan olio sisältää ristikon kohteesta yleistietoa, kuten tilauspäivämäärän, suunnitteluhinnan, rakennusnumeron, tilaajan nimen ja paikkakuntatiedot. Tilaajasta voisi tehdä oman olion. En ole kuitenkaan vielä toteuttanut tätä ratkaisua. Mittatieto sisältää ristikon kokoon vaikuttavia laskelmia, kuten kokonaispituuden, harjakorkeuden sekä alapaarteen pituuden. Kuormatieto sisältää painolla mitattavia asioita, kuten yläpaarteen ja alapaarteen oman painon, lumikuorman ja tuulikuorman. Puutavaratieto kertoo käytetyssä ristikossa olleitten puutavaroiden lujuusluokat ja mitat. Levytavara kertoo käytetyssä ristikossa olleitten naulalevyjen mitat ja tyypit. Puulevytieto koostuu puutavaraan ja naulalevyihin liittyvistä laskettavista asioista. Puulevytieto-olioon on laskettu yhteen puutavaroiden kuutiot lujuusluokittain ja kokonaissummana sekä naulalevyjen kappale- ja neliökoot.

Toinen osa luokkakaaviosta kuvaa karkeasti ohjelmassa toimivan logiikan yhteyksiä. DBConnection-luokka toimii abstraktina yläluokkana ODBConnection- ja OLEConnection-luokille. DBConnection saa arvonsa RegisterDB-nimiseltä luokalta, johon on tallennettu käytettävän tietokannan yhteystiedot ja yhteystekniikka. DBConnection toimii osana DBCommand-oliota, joka taas toimii DBDataAdapter-oliossa. DBDataAdapter-olio hoitaa tiedon keruun ohjelmassa käytettävän tiedon syöttämiseksi DataGrid-taulukkoon. DataGrid-taulukosta olen periyttänyt oman ratkaisuni. Myös hakutoimintoa varten olen tehnyt oman DatabaseCollection-olion.

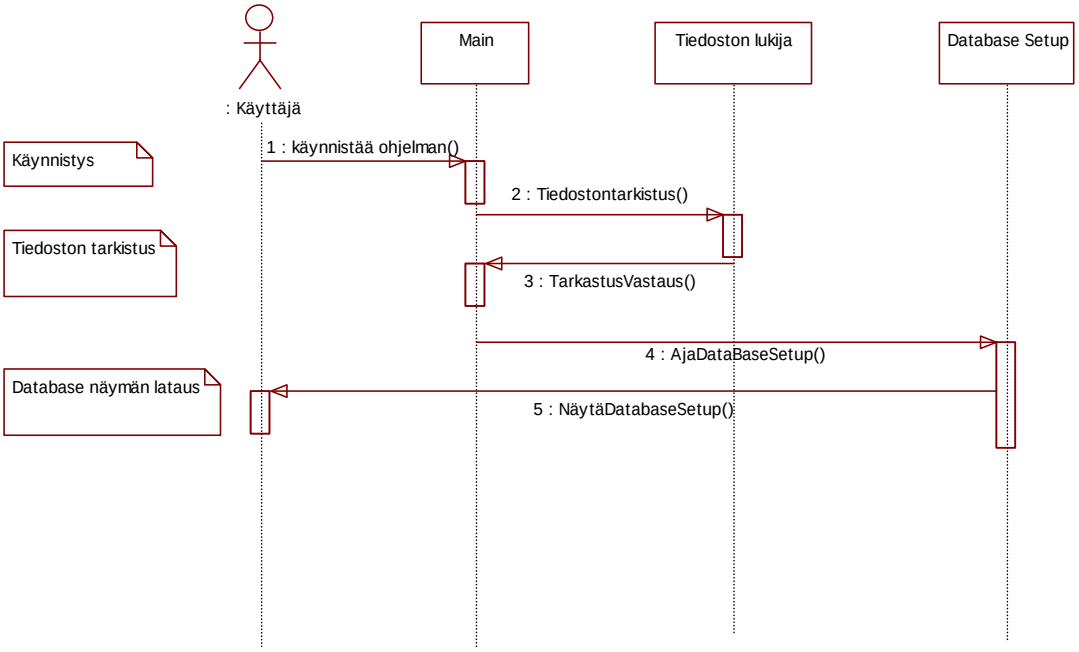


Kuvio 3. Havaintoesityksen luokat

Sekvenssikaavio

Sekvenssikaaviota voidaan käyttää prosessinäkymän toteutuksena. Sekvenssikaavio kuvaa toimintojen aikajärjestystä sovelluksen toiminnan aikana. Sekvenssikaaviossa voidaan hyvin havainnollistaa, mitä funktioita tarvitaan ja mitä parametreja tietyn komponentin funktio vastaanottaa sekä mitä attribuutteja se palauttaa.

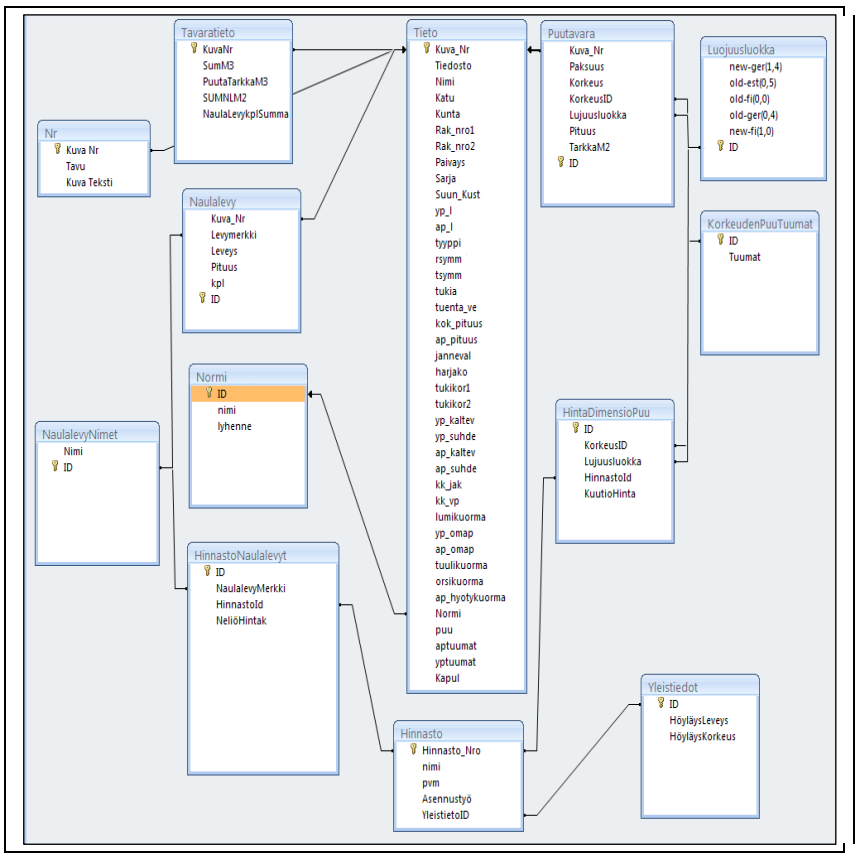
Kuvaan kuvion 4 sekvenssikaaviossa ohjelman käynnistystä ja Database Setup -ohjelman käsittelyjärjestystä. Aluksi käyttäjä käynnistää ohjelman. Pääohjelma tarkistaa ensiksi tietokannan olemassaolon tiedostosta, ja jos tietoa ei löydy, pääohjelma ajaa Database Setup -nimisen aliohjelman, joka tulee näkyviin ruudulle. DataBase Setup -ohjelmassa käyttäjä valitsee sopivan tietokantaratkaisunsa ohjelmalle.



Kuvio 4. Sekvenssikaavio Database Setup-ohjelman käynnistäminen

Tietokannan relaatiokaavio

Tietokannan relaatiokaavio nojautuu kahteen päätykeskipisteeseen. Relaatiokaavion ensimmäisenä päätykeskipisteenä toimii Tieto-taulu, joka sisältää keskeisimmät tiedot naulalevyristikosta. Normi-taulu ja NR-taulu ovat Tieto-taulun aputauluja. Toinen yhdistävä päätytaulu on Hinnasto-taulu joka sisältää hinnaston yleistietoja. Yleistiedot-taulu on Hinnasto-taulun aputaulu. Päätytaulut muodostavat kaksi haaraa. Ensimmäisen haaran muodostavat naulalevyjen yksityiskohtaiset tiedot ja naulalevyjen hinnaston tiedot. Toisen haaran muodostavat puutavaran yksityiskohtaiset tiedot ja puutavaran dimensiohinnasto. Puutavara-taulu riippuu sekä lujuusluokasta että korkeudesta puutavaran Hintadimensio-tauluun. Lujuusluokka- ja KorkeudenPuuTuumat-taulut ovat Puutavara-taulun aputauluja, jotka antavat tarkempaa tietoa. Lujuusluokka-tauluun on eroteltu lujuusluokkien norminimitykset.



Kuvio 5. Relaatiokaavio tietokannasta Accessissa

Koska isoja kyselyitä saattaa olla todella hidasta käyttää suoraan, tätä varten tehdään omat kyselytaulunsa (Bruni & al. 2003: 209). Relaatiokaaviokuvassa on jo yksi tällainen taulu. Tavaratietotauluun on laskettu puutavaran tarkat kuutiomäärät sekä naulalevyjen neliöt ja kappalemäärä.

Yhteishinnan laskemisessa käytän yleistä naularistikon asennuskaavaa. Kysely käy läpi koko relaatiotietokannan päädystä päätyyn ja on tietokantaohjelmoinnin vaativin osio. Tämä hidastaa sovelluksen ajoa huomattavan paljon. Kyselyssä voisi käyttää hyödyksi kyselytaulua, mutta koska hintatietoja voi muutella, pitäisi kyselytaulu päivittää useasti, mikä taas hidastaisi sovellusta. Tämä ratkaisu pitäisi pystyä toteuttamaan varsinaisessa ohjelmassa.

4 Toteutusvälineiden valinta

Tässä luvussa tarkastelen, mitä vaihtoehtoja tietokannan toteuttamiseksi on tarjolla. Käyn läpi vaihtoehtoja, jotka tukevat toisiansa. Aloitan tarkastelun tietoteknisiltä vaatimuksiltaan vähäisimmästä ja vanhimmasta tekniikasta ja kuljen kohti nykyisiä tekniikoita. Ensin luon katsauksen SQL-tekniikkaan, sitten ODBC- ja OLE-tekniikoihin ja lopuksi Visual Studion tarjoamaan C++ -tekniikkaan. Jo pelkästään SQL:n tarjoama nykyiseen standardiin pohjautuva tekniikka on riittävä, kun halutaan tehdä tietokannasta riippumaton sovellus. ODBC- ja OLE-rajapintojen ratkaisut ovat välitaipaleita kohti nykyistä kehitystä.

Visual Studio tarjoaa abstraktin tehtaan palvelua, joka käyttää hyödykseen rajapintojen tukia eri tietokannoille. Abstrakteissa tehtaissa suositellaan käytettäväksi kuitenkin SQLClient-pohjaista tuottajaratkaisua sen nopeuden takia (Ramadurai 2005). SQLClient ottaa yhteyden suoraan tietokantaan ilman rajapintatoteutusten hidastavaa vaikutusta. En kuitenkaan ryhtynyt tarkemmin tutkimaan SQLClient-pohjaista tuottajaratkaisua, vaikka nykyinen tekniikka tukee sitä ja täyttää hyvin vaatimukset tietokannasta riippumattomasta sovelluksesta. Itse halusin käyttää mahdollisimman tuoretta ja tuttua tekniikkaa sovelluksen toteuttamiseen ja samalla oppia uutta ohjelmoinnin saralla. Tehdas-malli tarjoaa laajan ja tehokkaan tietokantoja tukevan ympäristön ja on ohjelmistoarkkitehtuurisesti katsottuna nykyaikainen ja älykäs ratkaisu.

4.1 SQL

Sovellus voidaan toteuttaa ilman tietokantaa käyttämällä hyväksi tekstitiedostoa. Tekstin lukeminen tekstitiedostosta on kuitenkin todella hidasta verrattuna tietokantaan. Tietokanta antaa paljon monipuolisemmat mahdollisuudet käsitellä ja tutkia tietoa. Nykyään SQL on ainoa varteenotettava vaihtoehto.

4.1.1 SQL-standardien historia

ANSI (American National Standard Institute) alkoi vuonna 1986 kehittää yhteistä SQL (Structured Query Language) -kieltä, jonka ensimmäinen versio julkaistiin vuonna 1989. Vuonna 1992 standardista julkaistiin uusi versio, joka sisälsi tietokantayhteydet ja sisäisen yhteyden (INNER JOIN). Vuoden 1999 versio lisäsi taulukot, uusia tietotyyppejä ja näiden lisäksi uusia rakennustoimintoja. Vuonna 2008 julkaistun uusimman version pitäisi olla täysin yhteensopiva edellisten kanssa. (Jones & al. 2005: 2–3.)

Tietokantajärjestelmien valmistajat toteuttavat standardia eri tavoin. Oracle kertoo toteuttavansa tietokantansa ANSI:n uusimpien standardien mukaan. Microsoft puolestaan kertoo taas, että heille on tärkeämpää toteuttaa ohjelmistoympäristöön toimivampaa ja nopeampaa ratkaisua, ja Microsoftin Server toteuttaakin sen omalla T-SQL -kielellä. Sunin MySQL taas pyrkii seuraamaan tarkasti uusinta SQL-standardia. Muita suuria tietokantoja ovat IBM:n DB2 ja PostgreSQL. (Jones & al. 2005: 3–4.)

4.1.2 SQL-standardien vertailut tietokantoihin

Tietokantataulut muodostavat tietueelle kehykset, joiden raameissa tauluja luodaan ja käytetään. ANSI:n mukaan tietokantataulut luodaan CREATE TABLE -lauseilla, kun taas ALTER TABLE -lauseilla muokataan valittua taulua. Kyselyjä varten käytetään SELECT-lausetta. Datan muokkaamiseen tietokannassa käytetään INSERT-, UPDATE- ja DELETE-lauseita. INSERT-lause lisää tietokantaan tietueita. UPDATE muokkaa valittua tietuetta ja DELETE poistaa valitun tietueen tietokannasta.
Tietokantavalmistajat tukevat nykyään näitä peruslauseita hyvin. Tosin SQL-92 -standardiin ei vielä kuulunut esimerkiksi UPDATE-lausetta. (Jones & al. 2005: 2–4.)

Seuraavat vertailut pohjautuvat Jonesin & al. vuonna 2005 tekemiin vertailuihin. Tiedot pohjautuvat SQL-2003 -standardiin, joten ne saattavat olla osittain vanhentuneita. Aggregaatio-funktiot keräävät yhteenvedon solujen ryhmästä ja käyttävät yleensä Group By -lausetta. Tietokantojen valmistajat tukevat kaikkia ANSI:n mukaisia aggregaatio-funktioita. Aggregaatio-funktioita ovat AVG, Count, MAX, MIN ja SUM. (Jones & al. 2005: 60–61.)

Eroja tietokantaohjelmistojen tukemissa funktioissa alkaa ilmetä, kun tarkastellaan String-funktioita. Kaikissa tietokantaohjelmistoissa ovat kuitenkin tavanomaisimmat String-funktiot, tosin eri tavoin toteutettuina. Ainoastaan PostGreSQL ja MySQL

toteuttavat LEN-funktion nimellä CHAR_LENGTH. Funktio laskee merkkien määrän tekstikentässä. (Jones & al. 2005 65–66.)

Matemaattiset funktiot, kuten Round, Power ja Floor, voidaan toteuttaa hyvin monin tavoin. Oracle jättää matemaattisista funktioista Pii-, Cot-, Rand-, Square- ja Degrees-funktiot kokonaan toteuttamatta. Cot laskee yhtälöstä cotangentin. Square laskee lauseelle toisen potenssin, mutta yhtälön voi toteuttaa myös käyttämällä Power-funktiota. Rand arpoo luvun väliltä 0 ja 1, ja Degrees-funktio muuttaa radiaanit asteiksi. Vain Oracle ja IBM toteuttavat hyperboliset cosinin, sinin ja tangentin. Tosin IBM jättää Oraclen tavoin Piin toteuttamatta. Ainoastaan MySQL toteuttaa kaikki ANSI:n mukaiset logaritmifunktiot. MySQL:kin jättää Squaren toteuttamatta. (Jones & al. 2005: 69–72.)

Muista funktioista COALESCE ja NULLIF toimivat kaikilla tietokanta-alustoilla, tosin IBM käyttää COALESCE-funktiosta nimitystä VALUE. Funktio palauttaa ensimmäisen ei-Null-arvon listasta. Yleisesti ottaen kuitenkin tietokanta-alustat tukevat hyvin tavallisimpia funktioita, tai ainakin ne voidaan kiertää jollain tavalla toimiviksi kaikilla. (Jones & al. 2005: 85.)

ANSI toi käyttäjäkohtaiset funktiot vuoden 2003 standardiinsa, ja tämän vuoksi sen tuki eri tietokantavalmistajien tuotteissa on hyvin vaihtelevaa. ANSI:ssä Create Function -komennolla luodaan oma käyttäjäkohtainen funktio. SQL-rutiini on joko SQL-herätetty funktio tai proseduuri, mikä tarkoittaa, että pääherätys tapahtuu jonkin kutsun kautta. Koska proseduuri ei palauta arvoa, se vaatii parametreikseen attribuutin in, out tai inout. Contains SQL on oletusarvo sille, että voi olla muitakin esityksiä kuin tyypilliset SELECT, INSERT, UPDATE ja DELETE. SQL Data määrittelee, että koodi sisältää vain SELECT- tai FETCH-lauseita. ALTER FUNCTION -komennolla voidaan muuttaa jo valmiina olevaa omaa funktiota. DROP FUNCTION -komennolla poistetaan funktio tietokanta-alustasta. (Jones & al. 2005: 415–419.)

4.2 ODBC ja OLE

Microsoft kehitteli ODBC:n (Open DataBase Connectivity) vuonna 1992. Nykyään suurin osa tietokantajärjestelmien tarjoajista tukee omilla ajureillaan ODBC:tä. ODBC sisältää yleisen yhteiskielen ja tarjoaa erillisen oman laajennuksen tietokannoille (Elmasri & Navathe 2004: 41). Tietokantarajapinnat kehiteltiin asiakkaille, jotta heidän olisi helpompi vaihtaa tietokantaratkaisunsa toiseen, koska kieli on samantyyppinen, ja jotta he voisivat käyttää samaa välipalvelua sen toteuttamiseen.

ODBC:llä voidaan käytännössä nykyään toteuttaa kaikki relaatiotietokantaratkaisut. Ongelmana on lähinnä sen hitaus, joka suuremmissa ympäristöissä muodostaa pullonkaulan. Nopeus riittää kuitenkin nykyisellään perustietokannoissa, ja tietokantavalmistajat tukevat sitä hyvin. ODBC alkaa tosin olla jo vanhanaikainen järjestelmä. Microsoft on toki kehitellyt siitä uudempia versioita, mutta tuki on varovaista. Microsoft kuitenkin lisäsi ODBC:n ajurille oman tuen OLE:hen, mikä antaa ODBC:lle lisää elinikää. (Otey & Otey 2005: 256.)

DLL (Dynamic-link library) on Microsoftin kehittelemä kirjastojärjestelmä, joka toimii samalla suoritettavana ohjelmana. MS Administrative Tools Data Sources ODBC on DLL-pohjainen Microsoftin tekemä ODBC-käyttöympäristö ODBC:n tarjoajille. Ohjelma tarjoaa kahdentyyppistä tietokantayhteysratkaisua, jotka ovat DNS ja suora yhteys. DNS on nimipalvelu, ja se ottaa yhteyden tietokantaan siihen tallennetun nimen kautta. DNS-nimi kertoo, miten tietokantaan saadaan yhteys.

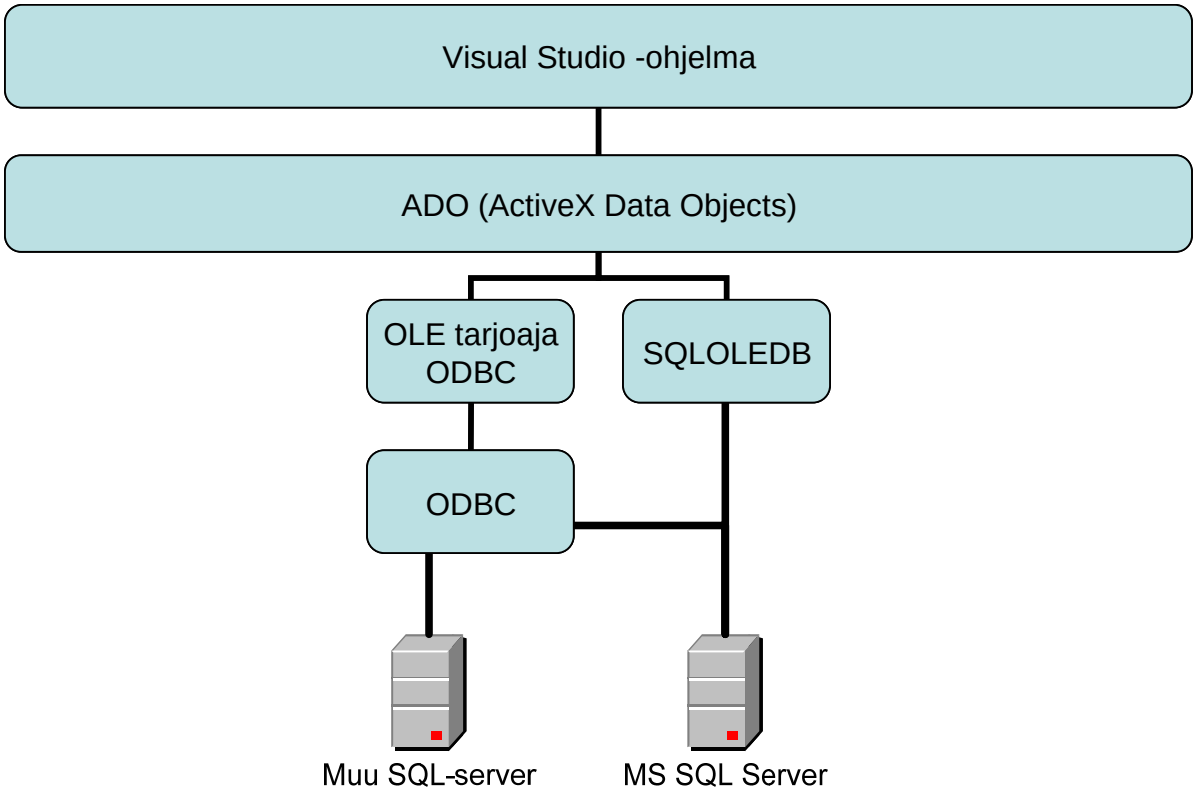
ODBC:n seuraajaksi Microsoft kehitti OLE:n (Object Linking and Embedding). ODBC oli tarkoitettu toimimaan relaatiotietokantojen kanssa SQL:ää hyväksi käyttäen, mutta OLE:ssä pyrittiin lisäämään ominaisuuksia, jotka tukisivat yhteyksiä relaatiotietokannan ulkopuolisiin tietolähteisiin. ADO (ActiveX Data Object) on Microsoftin toteuttama tietokantaohjelmiston palveluohjelma, joka tarjoaa sovelluksille yhteyden tietokantoihin. OLE tukee muun muassa Excel-tiedostoja, sähköposteja ja Active Directoryä. ODBC tarjoaa DLL-pohjaisen käyttöliittymän. ADO tarjoaa OLE:lle COM (Component Object Model) -pohjaisen käyttöliittymän. COM on komponentti-pohjainen oliomalli käyttöliittymäpohjaisten sovellusten luomiseen. (Otey & Otey 2005: 256.)

Kuviossa 6 esitetään tarkemmin OLE:n arkkitehtuuria. Ohjelmissa OLE:tä käytetään tarjoamaan tai kuluttamaan tietokantapalveluita. Kuluttajapuolen OLE-järjestelmät ovat perusohjelmistoja, jotka käyttävät OLE-rajapintaa. OLE-tarjoajat huolehtivat siitä, että tietokantayhteydet ja käyttäjäohjelma toimivat. Tarkemmin katsottuna on siis kahdentyyppisiä OLE-tarjoajia. Tiedon tarjoajan tarkoituksena on vain käsitellä tietolähteitä, kun taas palvelun tarjoaja hoitaa tiedon kuljetuksen ja muuntamisen. (Otey & Otey 2005: 257.)

Ohjelmisto eli OLE-kuluttaja.
OLE-rajapinta
OLE-tarjoaja
Tietokanta

Kuvio 6. OLE:n arkkitehtuuri

Kuviossa 7 esitetään Microsoftin tietokanta-arkkitehtuuria tarkemmin. Kuten ODBC:ssä myös OLE:ssä ajurit tukevat yleistä rajapintaa niin, että jokaisella ajurilla on omat laajennetut toimintonsa. OLE tukee myös ODBC-ajureita, tosin tällöin järjestelmä hidastuu eivätkä kaikki erikoispalvelut, kuten osoittimet, ole käytettävissä. (Otey & Otey 2005: 258.)



Kuvio 7. Microsoftin tietokanta-arkkitehtuuri

OLE:n nopeuden takaa ADO. ADO toimii OLE:n automaatiopalvelimena. Automaatio-palvelin toimii tietokantayhteyksien tarjoajana ohjelmistosovelluksille. ADO käyttää ActiveX-tieto-olioita nopeuttamaan tiedonsiirtoa. ADO kehiteltiin Visual Studio 6.0:lle. Uusimmissa Visual Studioissa ADO kuuluu suosittuun vakiovarustukseen. (Otey & Otey 2005: 259.)

ADO eli ActiveX Data Objects

ADO on kehitetty ja ohjelmoitu käyttämään hierarkkista oliokehystä. Microsoft laajensi ja samalla yksinkertaisti oliokirjastojaan suhteessa edellisiin kehyksiin, joita olivat DAO ja RDO. ADO:n kolme perustavaa oliota ovat Connection, Recordset ja Command. Connection-olio vastaa yhteydestä ulkoiseen tietokantaan. Tämän lisäksi Connection-oliota voidaan käyttää transaktiotarkastelussa. Recordset vastaa tietokannasta tulevasta vastauksesta. Se voi joko itse luoda yhteyden tietokantaan tai käyttää tähän Connection-oliota. Recordset-oliolla voidaan sekä tehdä kyselyjä tietokannasta että muokata tietokannassa olevaa tietoa. Jokainen Recordset-olio sisältää joukon Field-olioita, jotka vastaavat saraketta tietokannan taulussa. Command-olion tarkoituksena on antaa käskyjä ja tehdä parametreja SQL-lauseista. Command-oliota voidaan myös käyttää hakemaan tallennettuja proseduureja ja SQL-toimintolauseita. Kuten Recordset-olio, Command-olio voi joko itse luoda yhteyden tietokantaan tai käyttää Connection-olion palvelua. (Otey & Otey 2005: 260–261.)

Vähemmän käytettyjä olioita ovat Record- ja Stream-olio. Record-oliota voidaan käyttää edustamaan yhtä tietuetta Recordsetissä. Record-olio voi edustaa myös tiedostojärjestelmän tai sähköpostijärjestelmän hierarkiarakenteisia olioita, kuten kansioita ja tiedostoja. Stream-oliota käytetään Stream-tyyppisen tiedon, kuten xml- ja binaaritiedostojen, kirjoittamiseen ja lukemiseen. (Otey & Otey 2005: 262.)

4.3 Visual C++

Visual Studio 2003:ssa on multi-dll-tuki, joka mahdollistaa vanhojen Visual Studio 6.0:lle tehtyjen dll-tiedostojen käytön. Siirtyessäni Visual Studio 2008:n käyttäjäksi koin yllätyksen, sillä vanhan ohjelman kääntäminen sille ei ollutkaan mahdollista. Tämän vuoksi on aika siirtyä linkityksestä nykyaikaisempaan tehdastietokanta-ajatteluun. Visual Studio 2008 sisälsi ADO.NET 3.5 -tietokantajärjestelmän, joka sisälsi paljon uutta tietokantaohjelmointiin käytettävää ohjelmakoodia vanhojen sovelluksien ratkaisujen korvaamiseksi.

Abstraktit tehtaat

Microsoft kehitteli jo ADO.NET 1.1:een tietokannasta riippuvaiset tehtaat. ADO.NET 1.1 toi OLE- ja ODBC-rajapinnoille omat ratkaisunsa tietokannan yhdistämiseksi kielellisesti. ADO.NET 1.1:n Connection-oliot pyrkivät toteuttamaan rajapintojensa toteutuksia. ADO.NET 2.0:n tehtaat periytyvät abstraktista yläluokasta (Queen 2006.)

Abstrakti tehdas on suunnittelumalli, joka tarjoaa luontioperaatiot määritellyille komponenttijoukoille (Koskimies & Mikkonen 2005 94). Abstraktin tehtaan tarkoitus on tarjota rajapinta olioperheille ilman tarkempaa määrittelyä. Kun abstraktin tehtaan yhteyteen kirjoitetaan rajapintoja, määritellyt tehtaat erotetaan koodista. Koodi taas luo olioita, joita tarvitaan ohjelmassa. Tämä antaa mahdollisuuden luoda samanniminen olio eri tehtaan avulla eri toiminnoilla. (Queen 2006.)

DbProviderFactory on ADO 2.0:n abstrakti luokka. Luokka sisältää tavan kerätä tarvittava tieto siitä, mitä tehdasta aiotaan käyttää sovelluksessa. Tehtaan toteuttava luokka voisi siis olla joko OleDbFactory, OdbcFactory, SqlClientFactory tai OracleClientFactory. Jokainen tehdas sisältää tavat luoda omantyyppisiä tietokanta-olioita eli connection-, command- tai adapter-olioita. Toisin sanoen OleDbFactory tuottaa ainoastaan OLEDB-olioita. (Queen 2006.)

Oikean tehtaan määrittelemistä varten tarvitaan staattinen luokka. Koska staattiselle luokalle ei tarvitse erikseen tehdä oliota, sitä voidaan kutsua suoraan luokan kautta. Microsoft on toteuttanut sitä varten DbProviderFactories-luokan. Toinen tarvittava asia on yhteinen nimiavaruus; tätä varten Visual Studiossa käytetään System.Data.Common-nimiavaruutta. DbProviderFactories-luokka sisältää funktion GetFactory, joka noutaa rivi- tai tekstimuotoisena tiedot oikeasta tietokantatarjoajasta ohjelmalle. (Queen 2006.)

5 Toteutus

NR-Pakki

Suunnittelin ja toteutin VBA-Accessilla työharjoittelun aikana yritykselle NR-Pakki-sovelluksen. VBA on yksinkertaistettu Visual Basic-ohjelmointikieli Office-sovellusten sisäiseen ohjelmointiin. Access on työpöytä-ympäristössä toimiva T-SQL:ään pohjautuva tietokantajärjestelmä. Kuten NR-Mikro, NR-Pakki käyttää NR-kuvat-piirto-ohjelmaa ristikkokuvien piirtämiseen ja niiden selaamiseen.

NR-Pakki-sovelluksessa on pyritty toteuttamaan NR-Mikro-sovelluksessa olevat näkymät ja toiminnot. Sovellukseen on suunniteltu lomakkeelle, selaukselle ja kuvan piirrolle yhteistä näkymää. Liitteessä 3 on kuva yhteisestä näkymästä.

Sovellukseen on toteutettu hintalaskurinäkymä, jolla voi laskea uuden ristikon koko-naiskustannuksia. Sovellukseen on myös toteutettu omat näkymät ristikkokohteen naulalevyille ja puutavaralle.

Sovelluksen Erikoishaku-näkymä on suunniteltu NR-Mikron näkymää mukaillen. Erikoishakuun on lisätty uusia toimintoja, joita sovellukseen haluttiin.

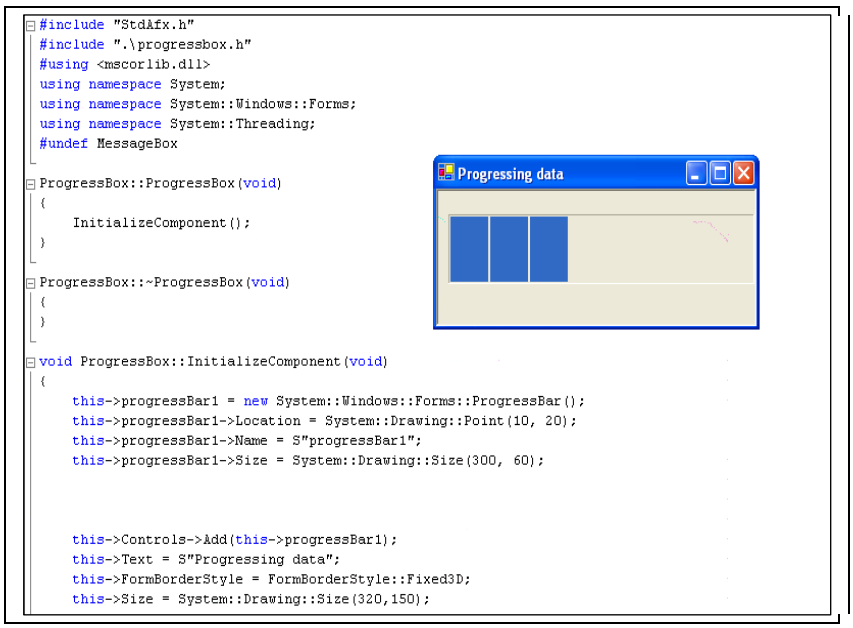
Sovelluksen Hinnasto-osio on laajennettu käsittelemään puutavaralle oman hintadimensionäkymän ja naulalevyille yksittäiskohtaisen neliöhinnastonäkymän.

Uusi NR-Sovellus

Ryhdyin tekemään tietokannasta riippumatonta ohjelmaa ensiksi Visual Studio 2003 C++:lla, koska se oli kuitenkin opintojeni pohjalta tuttu kieli. Tosin Visual Studion käyttämä C++ on erilaista kuin yleisesti käytetty C++. Microsoft suunnitteli omaan kehykseensä sopivan laajennutun C++ -kielen, joka on yhdenmukainen muitten Visual Studio -sovellusten kanssa. Olisin kuitenkin päässyt paljon vähemmällä, kun vain olisin opetellut C#:n. Visual Studio 2003 käyttää ADO.NET 1.1 -tietokantaohjelmistoa, joka tuki vielä vanhalla Visual Studio 6.0:lla tehtyjä sovelluksia. Myöhemmistä versioista eli Visual Studio 2005:stä alkaen Microsoft poisti tuen, koska katsoi sen hidastavan liiaksi ohjelmistoansa ja olevan ristiriidassa vanhan järjestelmänsä kanssa.

Käytännössä tein aluksi tietokantaan tekstin siirto-ohjelman käyttäen Visual Studio 2003:ssa odbccp32.dll -tiedostoa. Teksti sisälsi tietokantaan asetettavien kohteitten tiedot riveittäin. Graafinen ympäristö sisältää näyttöön piirtyvän osion komponentteja. Visual Studio 6.0:n säikeistettyä ohjelmointia olen käyttänyt lähinnä latauspalkkien

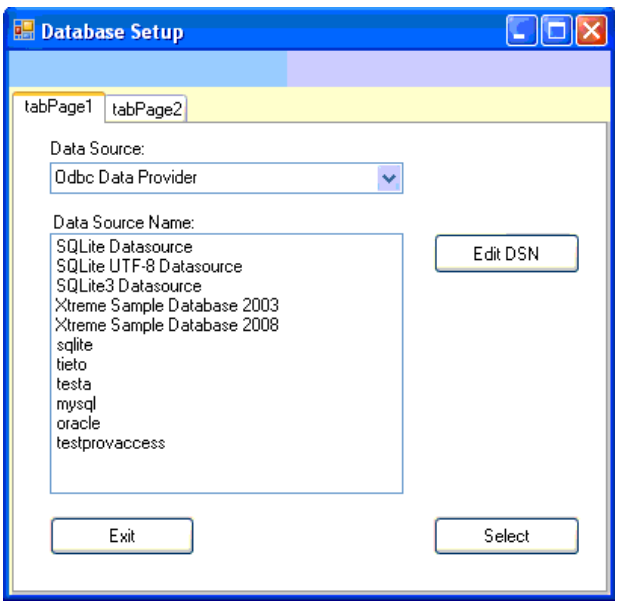
yhteyksissä tietokannan tietojensiirtämisen aikana. Säikeistetyssä ohjelmoinnissa säikeistysohjelma suorittaa useaa ohjelmaa samanaikaisesti. Todellisuudessa säkeistetty ohjelmointi käyttää suoritinta vuorottelemaan eri ohjelmien suorittamista määritellyin aikavälein. Kuviossa 8 näkyy säikeistetyä ohjelmointia ja sen toimintaa käytännössä.

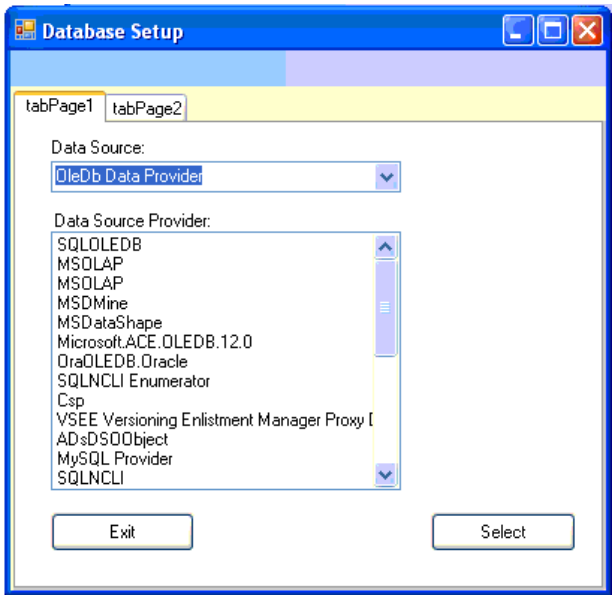


Kuvio 8. Säikeistystä vanhassa Visual Studio 2003:ssa

ODBC/OLE-vaihtaja

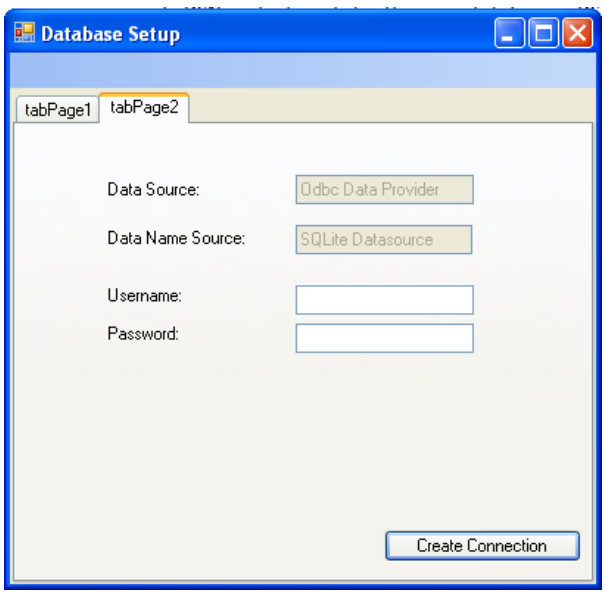
Ohjelmaa varten toteutin havaintoesityksen, jolla voidaan valita tietty tietokanta listasta. Visual Studio 2008 sisältää oman systeemin, jolla se listaa käytössä olevat tietokannat. Kokeilin aluksi tehdä omaa sisäistä DSN-asetusta, josta käyttäjä voisi tehdä omia DSN-asetuksia. Totesin sen kuitenkin liian hankalaksi, kun uusin Visual Studio ei tukenut vanhemmalla Visual Studiolla tehtyä DSN-palvelun sisäistä dll-tiedostoa, joten aloin vain yksinkertaisesti käyttää Microsoftin omaa DSN-palvelua käynnistämällä sen suoraan komentokehotteella. Kuvioissa 9 ja 10 näkyvä Data Source Name -palkki listaa käytössä olevat palvelut. Ensimmäisessä kuvassa on valmiiksi tehty ODBC-puolen palvelulistaus. Toisessa kuvassa on palvelinlistaus OLE-puolelta.

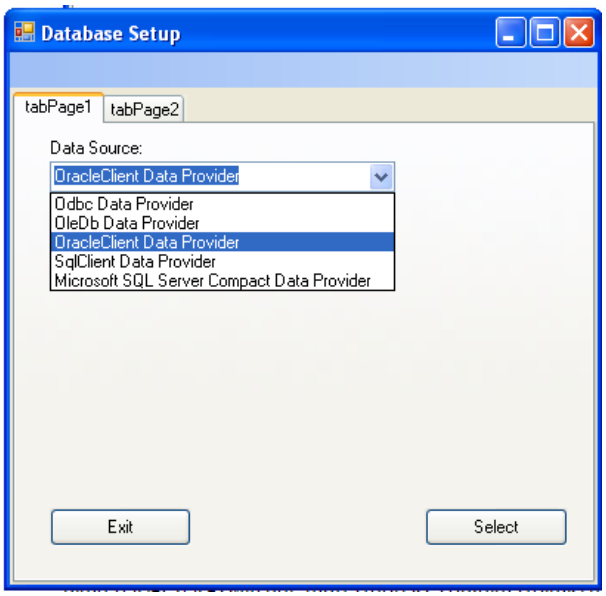




Kuviot 9 ja 10. ODBC- ja OLE-tehtaitten tietokannan tuottajalistat

Kun palvelu on valittu, voidaan tehdä yhteys ohjelmalle. Ohjelma tallentaa yhteystiedot erilliseen tekstitiedostoon, jonka ohjelma tarkistaa käynnistyessään. Mikäli ohjelma havaitsee tekstitiedoston tyhjäksi, se ajaa Database Setup -ohjelman. Käytän Database Setup -ohjelmasta nimeä ODBC/OLE-vaihtaja, mutta teoriassa ohjelma toimisi myös muiden tietokantajärjestelmien kanssa, koska Data Source -palkki listaa muutkin tietokoneen sisältämät tietokantapalvelujen tarjoajat. Näitä en kuitenkaan havaintoesitykseen erikseen toteuttanut. Kuvassa 12 näkyy lista muista tietokoneella olevista tietokantajärjestelmistä.



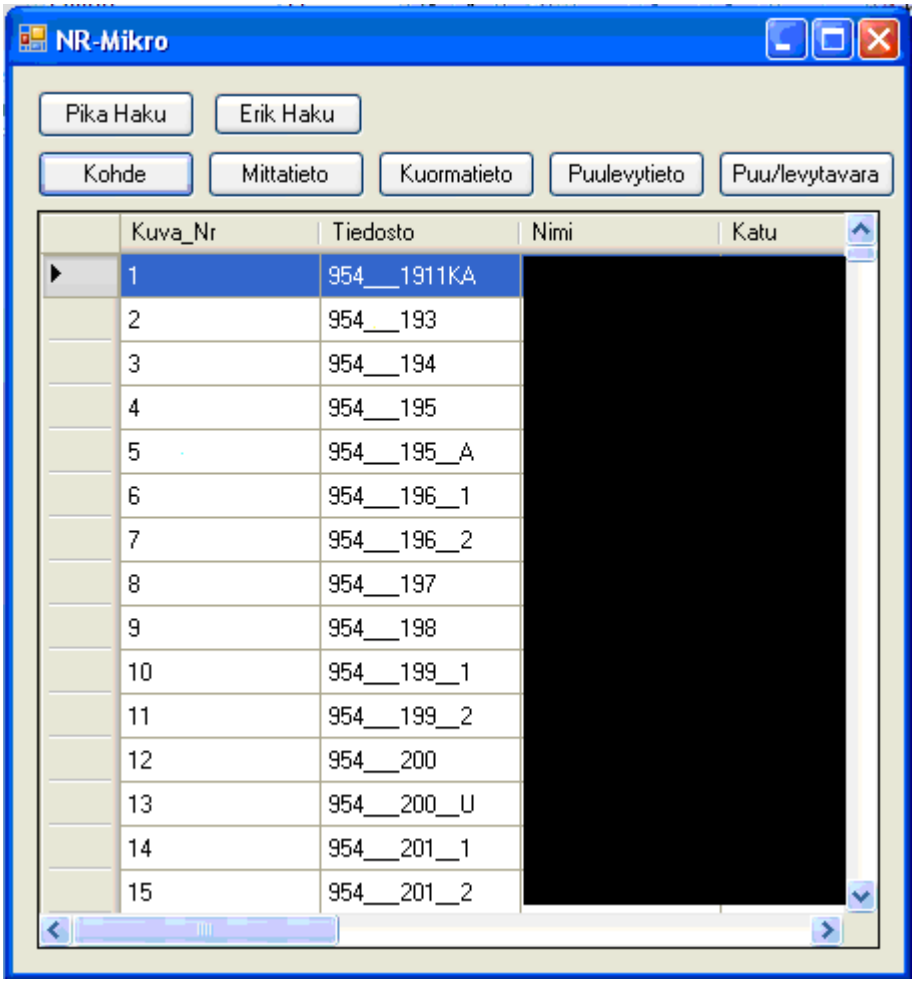


Kuviot 11 ja 12. ODBC:n tarkemmat tiedot, kuva muista tietokantatehtaista.

Selaus-näkymä

Kaavailin ohjelmalle samanlaista ulkoasua kuin oli ollut ohjelman aiemmissa versioissa eli eräänlaista tietokohteitten selaus-näkymää. Tämän toteutin pitkälti DataGridin avulla. Koska ohjelman valmis DataGrid-luokka ei toteuttanut kaikkia haluamani toimintoja, lähdin periyttämään tästä omaa toteutusta. DataGrid toimii sovelluksen perusselausalustana. DataGrid esittää tietokantataulun ilmentymää sovelluksessa. DataGrid käyttää ADO.NET:n tarjoamaa tietokantapalvelua hyväkseen ottaakseen yhteyttä tietokantaan. DataGrid-ohjelma mahdollistaa sarakkeen mukaiset järjestämisen. Lisäsin MyDataGrid-toteutukseeni pari uutta funktiota: SetMySelectedRowCore- ja SetMySelectedCellAddressCore-funktiot. Kummankin funktion pohjana käytetään isäntäluokan suojattuja funktioita asettamalla niille oletusarvot. Funktiot toteutettiin hakutoimintoja varten.

Kuvion 13 sovelluksen selaus-näkymässä jaottelin kohteen eri tietoryhmät omiksi tauluiksi. Lisäksi kokeilin tehdä perus- ja erikoishakuja vanhoja ohjelmia mukaillen. Kuviossa 13 olen toteuttanut molemmille hauille oman painikkeen yläreunaan. Kaikkien naularistikkokohteitten täyttäminen taulukkoon kuitenkin hidasti ohjelmaa, koska tietueiden määrä saattoi olla suuri. Toinen ratkaisu olisi ollut hakea tietokannasta vain osa kohteista kerralla näkyviin sivuttamalla tietueet lukumäärien mukaan. Tämä ratkaisu saattaisi nopeuttaa ohjelmaa ratkaisevasti käyttäjän kannalta.



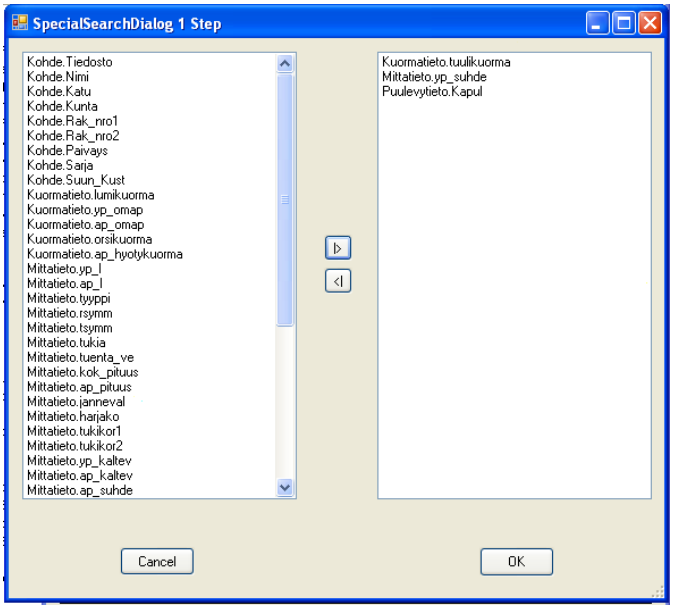
Kuvio 13. Havaintoesityksen Selaus-näkymä (osa tiedoista peitetty)

Erikoishaun näkymät

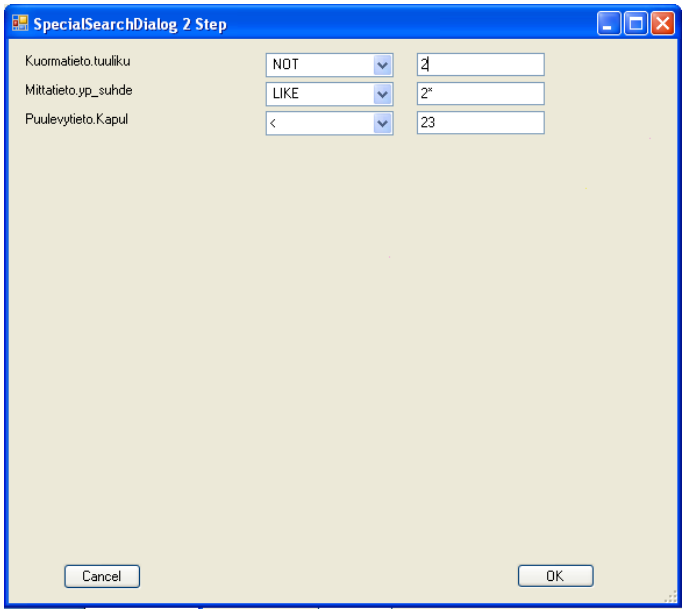
Erikoishaun tarkoituksena on rajata tietokannassa olevia naulalevyristikkokohteita. Hauissa pystyy käyttämään kätevästi SQL:n sisältämiä rajausmetodeja. Toki nämä vaativat jonkinlaista syötteen tarkastusta, kun käyttäjä sitten lähettää rajauksen tietokantapalvelimelle. Syötteentarkastuksessa käytin Regular Expression -menetelmää, jota tukevat useimmat ohjelmointikielet.

Ohjelmoin havaintoesitysversioon käyttöliittymälistauksen, jossa käyttäjä voi itse rajata tarpeettomia tietoja hakuehtojen ulkopuolelle. Kuvion 14 vasemmanpuoleinen listaus näyttää valittavissa olevat naulalevyristikon alkiot. Oikeanpuoleisessa listauksessa näkyvät jo valitut naulalevyristikon alkiot. Listojen keskellä on nuoli-näppäimet alkioiden siirtämiseksi listasta toiseen. Kun käyttäjä on saanut valittua tarvittavat alkiot, hän painaa OK-nappia suorittaakseen lopullisen rajaushaun. Tämän jälkeen ohjelma siirtyy hakuvaiheen toiseen osaan, jossa määritellään raja-arvot alkiolle. Kuviossa 15 näkyy valittujen alkioiden toinen hakuvaihe. Rivillä on alkion nimi, rajausehto ja

rajausarvo. Käyttäjä voi valita valintalaatikosta sopivan rajausehdon alkiolleen ja kirjoittaa rajausehdon tekstikenttään.



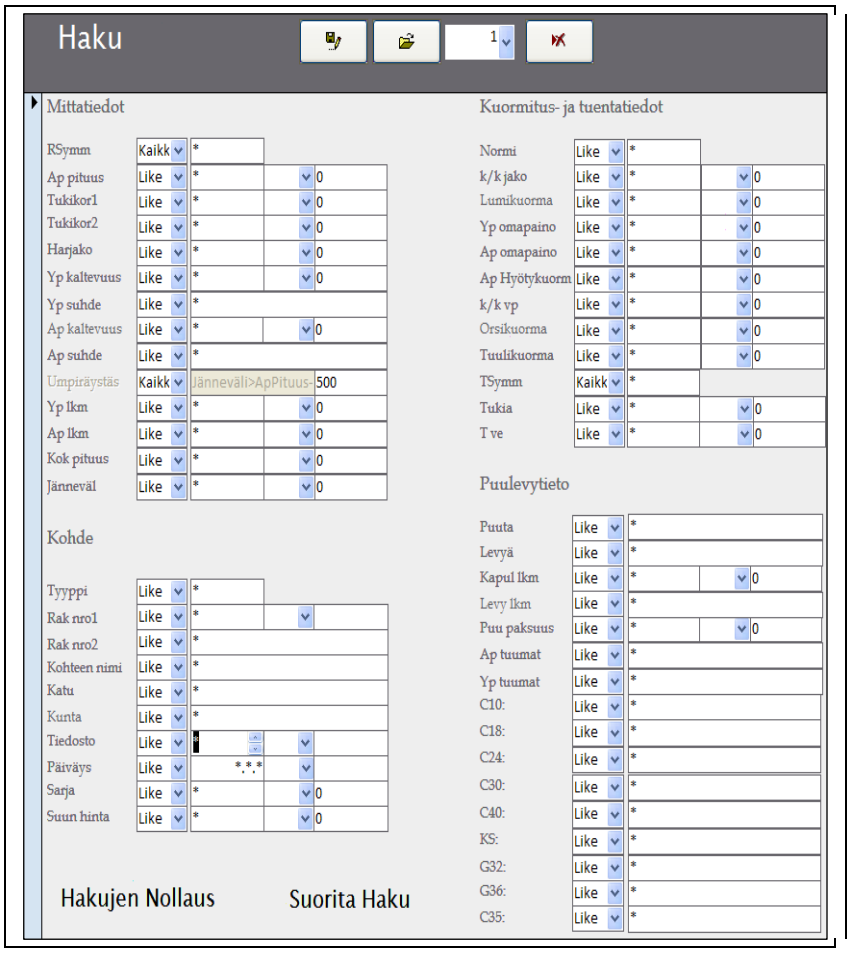
Kuvio 14. Erikoishaun ensimmäinen vaihe, jossa valitaan haettavat alkiot.



Kuvio 15. Erikoishaun toinen vaihe, jossa määritellään haettavien kohteitten tietoja.

Vanhassa Access-versiossa kaikki oli suoraan haettavissa yhdeltä sivulta. Sitä käyttäjät pitivät käytettävämpänä ja nopeampana ratkaisuna. Kuvio 16 näyttää, miten rajaushaku on toteutettu Access-versiossa. Alkiot, hakuehdot ja hakuarvot ovat kaikki näkyvissä samalla sivulla. Osille alkioista haluttiin rajata arvoja tietyltä lukuväliltä. Tätä varten tein ohjelmaan alkiolle toisen rajausehdon ja rajaushaun. Lisäksi käyttäjä voi tyhjentää kentät. Ohjelmaa varten toteutettiin umpiräystäsrajaus nopeuttamaan sen tyyppisiä

hakuja. Rajaushaun ylänurkassa olevilla painikkeilla hakuja voidaan tallentaa, avata ja poistaa.



Kuvio 16. Accessilla tehdyn ohjelman erikoishaku.

Hintalaskuri

Ohjelmaan haluttiin myös hintalaskuri. Se toimii apuohjelmana pääohjelman sisällä. Hintalaskurin alkuperäinen, Paradox-versio suunniteltiin ja toteutettiin alun perinkin yrityksen asiakkaiden käyttöön. Asiakkaat halusivat kyetä laskemaan hintoja naula-levyristikoillensa jo ennen tilausta.

Kuvion 17 hintalaskuri näyttää pitkälti sitä, mitä sen ohjelmassakin tulisi pystyä tekemään. Laskuri ottaa talteen kohteen tiedot, minkä jälkeen käyttäjä pääsee muokkaamaan niitä. Laskurin tarkoituksena on antaa käyttäjän laskea suunnitellun tuotteen kustannuksia vanhojen tilausten mallin mukaan.

Hintalaskuri on jaettu viiteen osaan: perustietoihin, puutavaraan, naulalevyyn, asennukseen ja kokonaiskustannukseen. Perusosiossa tarkastellaan, mistä

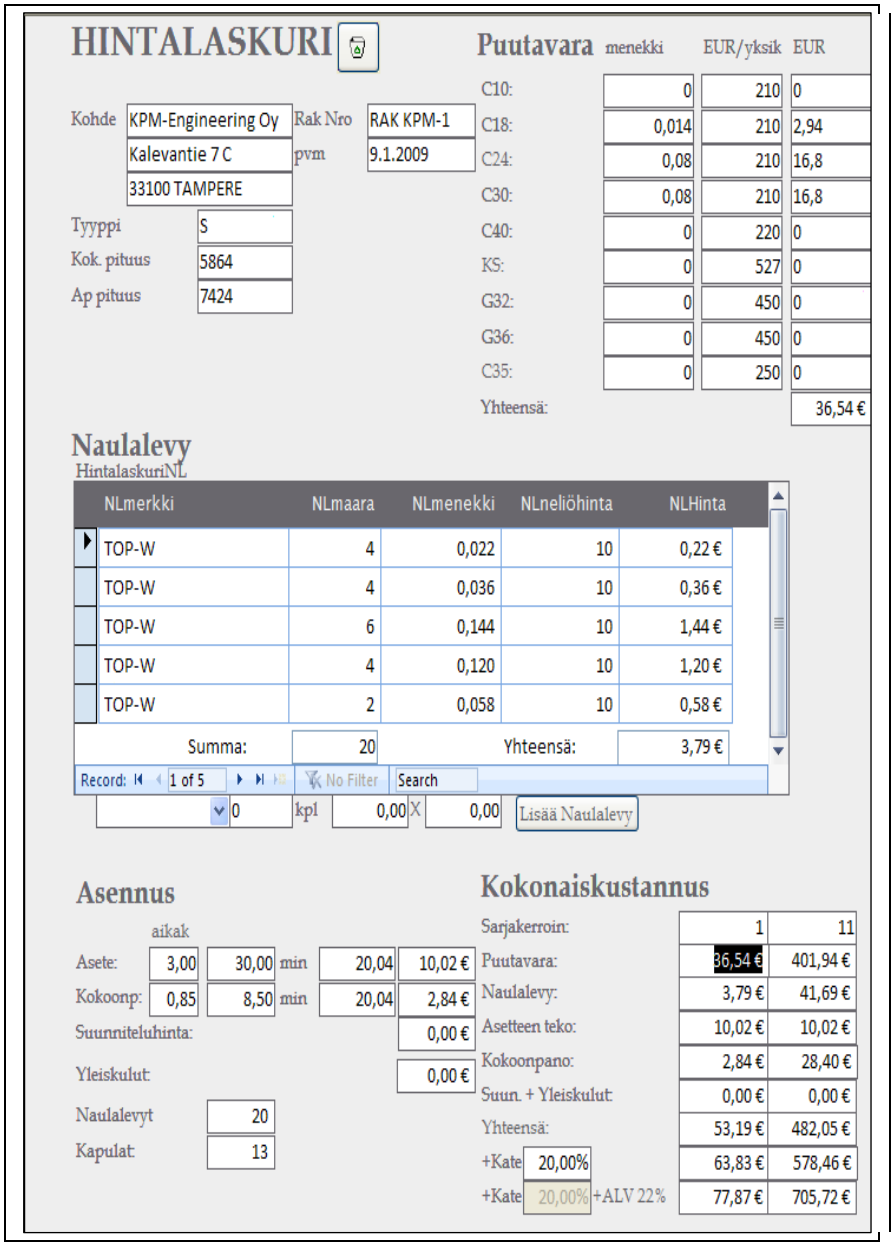
ristikkokohteesta tuote on alun perin otettu ja mitä keskeisimpiä tietoja se sisältää. Näitä tietoja ei tässä hintalaskurissa voida muuttaa. Hintalaskurissa on myös poistopainike, jolla voidaan tuhota kyseinen hintalaskuriehdotelma tietokannasta. Hintalaskuriehdotelmat tallentuvat tietokantaan samalla tavoin kuin hakutiedot. Kuviossa 17 poistopainiketta kuvaa roskakori.

Puutavaraosiossa lujuusluokkien kuutiomäärät eli menekkitiedot haetaan valitun ristikkokohteen puulevytieto-osiosta. Euroyksiköt lasketaan lujuusluokkien valitun hinnaston hintadimensioiden keskimääräisistä hinnoista. Ohjelma laskee lujuusluokkakohtaiset yhteishinnat menekistä ja yksiköistä. Käyttäjä voi vapaasti muokata menekin ja euroyksiköiden kohteiden lukuja. Lisäksi ohjelma laskee yhteissumman kaikelle puutavaralle.

Naulalevyosioissa näytetään valitun ristikkokohteen käytetyt naulalevyt. Naulalevytaulu sisältää kappalemäärän, neliökoon ja neliöhinnan ja laskee näistä naulalevykohtaisen hinnan ja kokonaishinnan. Taulu hakee valitut tiedot ristikkokohteen naulalevytaulusta. Lisäksi taulu hakee naulalevyjen hinnat käytössä olevan oletushinnaston hinta-arvon mukaan. Naulalevyosioissa on myös naulalevyn lisäysosio, jossa käyttäjä voi lisätä naulalevyn naulalevytauluun.

Asennusosioissa käydään läpi asetukseen ja koonpanoon liittyviä hintoja. Asetuksessa käytetään oletuskaavaa, jossa aikakerroin 3 kerrotaan naulalevyparien määrällä. Tästä saadaan aika, joka kuluu yhden asetteen asennukseen. Aseteaika kerrotaan asennus-hinnalla, josta saadaan asetehinta. Asetteessa kootaan ensimmäinen ristikko valmiiksi sarjaa varten. Kokoonpanon kaava toimii samalla tavalla kuin asetteen, mutta aika-kerroin on luonnollisesti pienempi, koska asetekehikko on valmiina. Käyttäjä voi muokata alikertoimia ja asetuksen tuntihintaa. Suunnitteluhinnan tiedot haetaan valitusta ristikkokohteesta. Yleiskulut voi käyttäjä itse määritellä itselleen sopivaksi. Naulalevyjen ja kapuloiden lukumäärät antavat lisätietoa asennuksessa käytettävistä tavaroista.

Kokonaiskustannusosio laskee yhteen muiden osioiden summat. Sarjakerroin haetaan valitusta kohteesta, mutta käyttäjä voi itse vaikuttaa siihen. Sarjakerroin kuvaa sitä, kuinka monta naularistikkoa tarvitaan katon laittamiseen. Käyttäjä voi muokata kateprosenttia ja sarjakerrointa haluamakseen suoraan kyseistä kentistä.



Kuvio 17. Hintalaskurin Access-malli.

6 Johtopäätökset

Työn tarkastelu

Kokosin alkuperäisen Access-havaintoesityksen kahdessa viikossa ja sen jälkeen aloin toteuttaa koodia Visual C++:lla. Tämän havaintoesityksen tekemiseen meni kaksi kuukautta. Se toimi, mutta käytti vain ODBC-rajapintaa hyödyksi. DataGrid-toteutus oli melko hidas jo 200.000 tietueen kanssa. OLE:n kanssa ohjelma toimi suhteellisen nopeasti.

Uuden, tietokannasta riippumattoman ohjelman hyödyt olisivat olleet kuitenkin vähäiset tarpeeseen nähden. Ohjelma ei olisi pystynyt nopeudessa juurikaan auttamaan. Yksi syy oli Visual Studion kehyksien kankeus ja hitaus. Toki ohjelma on hidas nykyisellä Accessillakin, vaikka käyttää aputaulukoita hyväksi. Visual Studion kehyksien hyötynä olisi ollut ohjelman valmius tulevaisuuteen ja tietokantojen muuttumiseen. Nyt ohjelma on täysin riippuvainen Accessista ja sen kehityksestä tulevaisuudessa. Microsoft on kuitenkin lupautunut pitämään tukeansa myös Accessille SQL Serverinsä lisäksi.

Visual Studion havaintoesityksen toteutukseen meni nelinkertainen aika Access-versioon verrattuna. Tämän perusteella arvioin, että Visual Studio C++:lla valmiin sovelluksen tekemiseen olisi kulunut 24 kuukautta.

Kesällä esiteltyäni havaintoesitystä ICT-päällikölle ohjelman kehitystyö keskeytettiin, ja palasin kehittämään ensimmäistä Access-versiotani, joka alkaa nyt olla valmiina. Access-version toteutukseen meni kuusi kuukautta.

Visual Studiolla toteutettavan ohjelman kehittämiseen voidaan palata myöhemmin, jos se katsotaan ajankohtaiseksi ja tarpeelliseksi. Nykyisen havaintoesityksen ohjelman-pätkät ovat tallessa koneella ja pystyn tarvittaessa nopeasti tarttumaan työhön uudestaan.

Ohjelman sallima yhtaikainen kuvienpiirto saattaa tulla kompastuskiveksi nykyisessä tekniikassa. Access kun ei sisällä kunnollista dynaamista vektorin piirto-ohjelmatukea. Visual Studion tarjoama valmis piirtoratkaisu olisi yksi uutta järjestelmää puoltava tekniikka.

Ohjelma kuitenkin tulisi kehittää SQLClient-tietokantajärjestelmälle tai jollekin tulevaisuuden tietokantatekniikalle sopivaksi, koska järjestelmä vaatii mahdollisimman

nopean tietokantatarjoajan. Mikään muu tietokantatarjoaja ei selviä näin massiivisista tietokannan tietueista ja laajoista kyselyistä.

Loppupäätelmät

Esittelin aluksi työn taustoja työn tilaajan, toimintojen ja menetelmien osalta. Kävin läpi vaatimusmäärittelyn ongelmia ja vaatimuksia. Kerroin suunnittelussa korkean tason arkkitehtuurisista näkymistä ja esittelin tietokannan relaatiosuhdekaavion. Menetelmissä kävin läpi tietokannasta riippumattomia ratkaisuja ja valitsin niistä sopivimman. Lopuksi esittelin ohjelman näkymiä ja pohdin jatkokehitysmahdollisuuksia.

Valmistelin havaintoesityksen valmiiksi jo heinäkuussa. Pääsin kuitenkin aloittamaan dokumentaation ja opinnäytetyön kirjoittamisen vasta marraskuussa 2008. Opin työn aikana paljon Visual Studion tarjoamasta C++ -tekniikasta. Kaiken kaikkiaan sain kirjoitettua oleellisimmat asiat ylös aihealueestani.

Koska työstä oli heikosti tietomateriaalia tarjolla, on työ edellä kävijä tämän tyyppisissä teknologioissa. Lisäksi työllä on varmasti tulevaisuuden arvoa, koska se pohjautuu uusimman Visual Studion tarjoamaan yleispätevään tekniikkaan.

Lähteet

Bruni, Paolo & al. (2003). DB2 UDB for z/OS Version 8: Technical Preview. IBM, USA

Buschmann, F. & al. (1996). Pattern-Oriented Software Architecture: A System of Patterns. Wiley, New York, USA.

Elmasri, Ramez & Navathe, Shamkant B. (2004). Fundamentals of Database Systems. 4th Edition. Pearson Addison Wesley, Boston, USA

Heikkilä, Arto & Metsäranta, Jarmo (1989). Naulalevyrakenteista. Tampere.

Jones, Arie & Stephens, Ryan & K. Plew & Ronald R. (2005). SQL Functions Programmer's Reference. John Wiley & Sons. USA

Koskimies, Kai & Mikkonen, Tommi (2005). Ohjelmistoarkkitehtuurit. Talentum, Helsinki.

Kruchten, Philippe (1995). Architectural Blueprints – the 4+1 View Model of Software Architecture. IEEE Software.

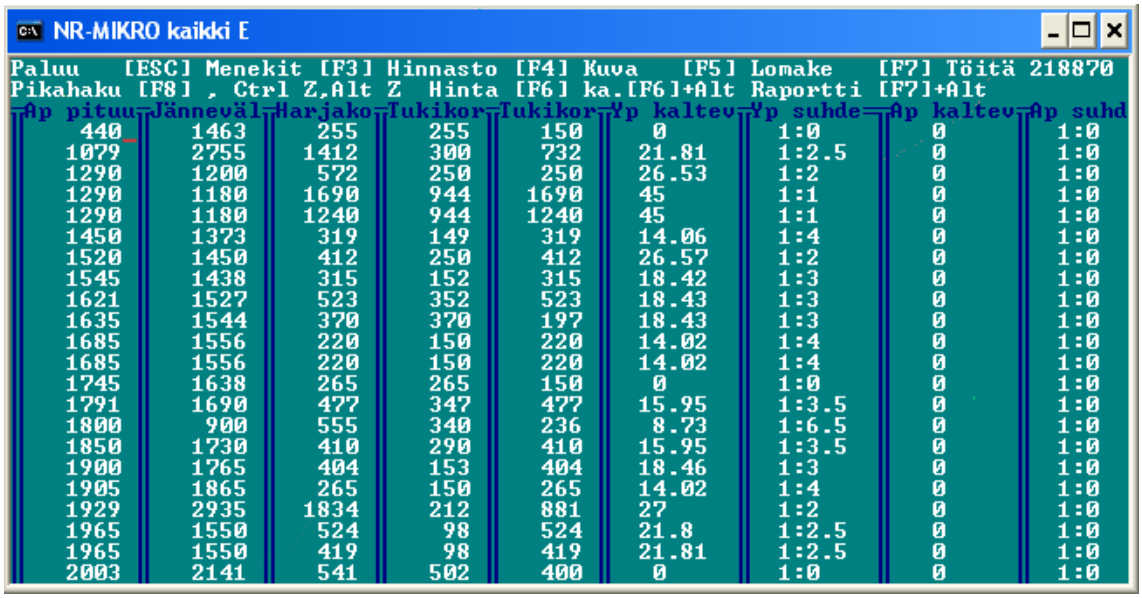
Lindell, Matti (2008). Monikerrosarkkitehtuuria noudattavan Java-sovelluksen toteutus. Pro gradu -tutkielma. Joensuun yliopisto. ftp://cs.joensuu.fi/pub/Theses/2008_MSc_Lindell_Matti.pdf (15.3.2009 verkossa)

Otey, Michael & Otey, Denielle (2005). Microsoft SQL Server 2005 Developer's Guide. McGraw-Hill, Osborne. USA

Queen, Anthony (11.7.2006). Provider-independent code simplified in ADO.NET 2.0 http://www.codeproject.com/KB/vb/ProviderIndependentCode.aspx (17.3.2009 verkossa)

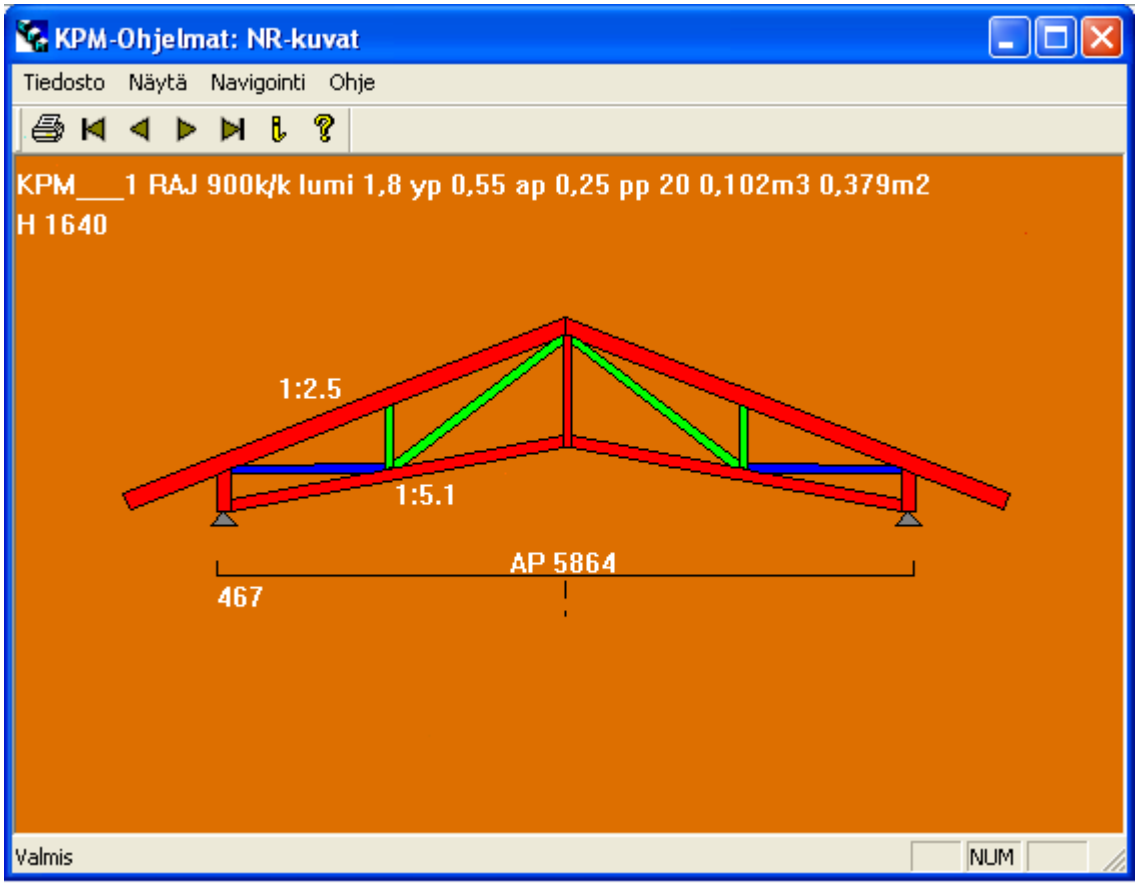
Ramadurai, SeenivasaRagavan (17.3.2005). Database Provider-Independent Data Access Layer Using ADO.NET 2.0, http://www.codeguru.com/csharp/.net/net_data/sortinganditerating/article.php/c9373 (17.3.2009 verkossa)

Liite 1



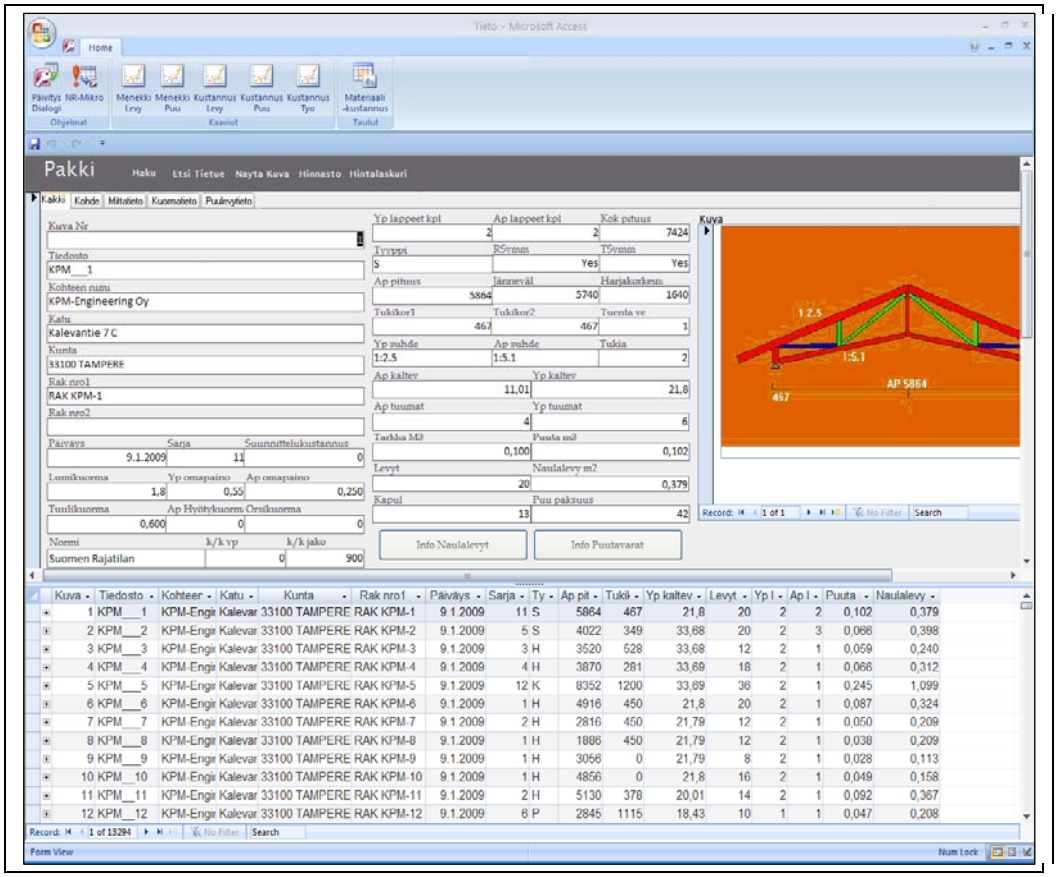
NR-Mikro ohjelma Selausnäkymä

Liite 2



NR-kuvat-ohjelman näkymässä saksiristikko.

Liite 3



Accessilla tehty naularistikon selausohjelma.