Tuukka Malinen

Visual Studio 2008 ja NET 3.5

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Tradenomikoulutus
Tietojenkäsittelyn koulutusohjelma
Kevät 2009

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittelyn koulutusohjelma
Tekijä(t) Tuukka Malinen	
Työn nimi Visual Studio 2008 ja ASP.NET 3.5	
Vaihtoehtoiset ammattiopinnot Ohjelmistosuunnittelu	Ohjaaja(t) Veli-Pekka Piirainen
	Toimeksiantaja Veli-Pekka Piirainen / Kajaanin ammattikorkeakoulu
Aika Kevät 2009	Sivumäärä ja liitteet 121 + 2

Web-suunnittelu ja siihen käytettävät työkalut kehittyvät erittäin nopeasti. Kilpailu eri ohjelmistoalustoiden ja -tekniikoiden välillä on kiivasta ja tarjontaa on laajalti. Yleensä menetelmät keskittyvät ohjelmointityön tehostamiseen ja tiimityöskentelyn helpottamiseen, kuten Microsoftin Visual Studio -ohjelmointialusta, .NET -ohjelmointikehys ja siihen kuuluva ASP.NET.

Tämän opinnäytetyön tavoitteena on tutkia uusia ASP.NET-tekniikoita ja Visual Studio 2008 -ohjelmointialustan uusia ominaisuuksia, sekä perehdyttää lukija ASP-menetelmien taustaprosesseihin. Tarkoituksena on valottaa ASP.NET-tekniikoiden historiaa, sekä vanhempia ohjelmointikontrolleja, joita edelleen käytetään aktiivisesti. Työ on suunnattu erityisesti ASP-kursseja käyville ja työharjotteluun valmistautuville opiskelijoille.

Työn teoriaosuus keskittyy Visual Studio 2008:n lisäksi uuteen .NET-ohjelmointikehykseen ja erityisesti sen LINQ-tekniikoihin, jotka mahdollistavat eri tietovarastojen, kuten tietokantojen, nykyaikaisen käytön. LINQ -menetelmiä sovelletaan sekä vanhempiin että uusiin ASP.NET-datakontrolleihin, joiden avulla haettu data voidaan esittää Internet-selaimissa. Teoriaosuudessa käsitellään myös ASP.NET-ympäristön uusia AJAX -kontrolleja sekä niiden käyttökohteita. Lopuksi käsitellään lyhyesti ASP.NET-ohjelmoinnin yleisiä tietoturva -aukkoja ja kuinka ne estetään. Työssä ei käsitellä käytännön tasolla laajempia tietoturvamekanismeja, kuten käyttäjienhallintaa tai sisäänkirjautumis-menetelmiä.

Työn käytännön osana on ohjelmointiopas, jonka toteutusta teoriaosuus tukee. Opas on pyritty tekemään niin yksityiskohtaisesti, että teoriaosuuden lukeminen ei ole välttämätöntä, vaikka se on suositeltavaa. Oppaassa rakennetaan vaihe vaiheelta perustason SQL-tietokantaa käyttävä ASP-sovellus, jossa LINQ-menetelmillä toteutetaan tietokantaan kohdistettavat perustoiminnot. Oppaan, kuten koko työn taso on haastava, mutta pyrkimyksenä on ollut esittää ensin helpompia skenaarioita ja siirtyä sen jälkeen haastavampiin toimintoihin.

Kieli	Suomi
Asiasanat	ASP, ASP.NET, ASP.NET 3.5, Visual Studio 2008, LINQ, ohjelmointi
Säilytyspaikka	☒ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☒ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School	Degree Programme
Business	Data Processing

Author(s)
Tuukka Malinen

Title
Visual Studio 2008 and ASP.NET 3.5

Optional Professional Studies	Instructor(s)
Programming	Veli-Pekka Piirainen
	Commissioned by Veli-Pekka Piirainen / Kajaani Univercity of Applied Sciences
Date Spring 2009	Total Number of Pages and Appendices 121 + 2

Tools for web programming are developing fast. The competition between different software platforms and technologies is fierce, mainly because there are a lot of products to choose from. Usually all the products focus on productivity and teamwork, like Visual Studio programming platform, .NET Framework and ASP.NET, all made by Microsoft.

This thesis concentrates on the latest versions of ASP.NET and Visual Studio, as well as familiarizes the reader with the background processes of ASP.NET runtime. The history of ASP.NET is also presented, along with examples of older programming controllers still used on a regular basis. The thesis is specially meant for those students who are studying on ASP.NET courses and are getting prepared for the practice period.

In addition to Visual Studio 2008, the theory of this thesis explains the .NET Framework and its new LINQ techniques meant for handling different data sources with the same programming syntax. The LINQ techniques are adapted to ASP.NET data controls, which are used to show the gathered data in Internet browsers. The last part of the theory concentrates on the ASP.NET AJAX controls and the basics of data security in the ASP.NET environment.

This thesis includes a detailed programming guide with steps that are theorically possible to repeat without reading any of the theory. The guide shows step by step how to make a basic, SQL-oriented ASP program which uses the new LINQ techniques. The guide can be technically challenging, like the whole thesis, but the goal has been to first show simpler scenarios and move on to more difficult ones later.

Language of Thesis Finnish

Keywords ASP, ASP.NET, ASP.NET 3.5, Visual Studio 2008, LINQ, programming

Deposited at ☒ Kaktus Database at Kajaani University of Applied Sciences
☒ Library of Kajaani University of Applied Sciences

SYMBOLILUETTELO

ADO.NET	Osa .NET-Frameworkia, sisältää komponentit datavarastojen, kuten relaatiotietokantojen, käsittelyyn.
ASP.NET	Webkehitykseen tarkoitettu Framework, jonka avulla ohjelmoijat voivat luoda dynaamisia web-sivuja, -sovelluksia ja palveluja .NET-ympäristössä. ASP.NET on osa .NET -ohjelmistokehystä.
ASP .NET AJAX	ASP.NET-ympäristön AJAX (*engl.* Asynchronous JavaScript and XML). JavaSkriptillä toteutettu objektikirjasto, joka on sulautettu Microsoftin .NET Frameworkiin.
ASPX	Visual Studion ASP-ympäristöön luotujen web-sivujen tiedostopääte. Tämä aspx-tiedosto sisältää enimmäkseen sivun ulkoasun esittämiseen tarvittavaa XHMTL-merkkikieltä ja skriptejä. Aspx-tiedostolla on siihen sidottu .cs-tiedosto, johon sivun toimintalogiikka sijoitetaan. Windows Forms -projektien tapaan tämä luokka on myös .cs-päätteinen.
Breakpoint	Taukopiste, jota käytetään Visual Studion virheentarkistuksessa. Ohjelman ajo pysäytetään näissä taukopisteissä, jolloin esimerkiksi muuttujien sisältöjen tarkastelu on mahdollista ilman erillisiä tulostuslogiikoita.
C#	C-Sharp, Microsoftin kehittämä vahvasti tyypitetty ohjelmointikieli .NET-ohjelmointiympäristöön.
CLR	Common Language Runtime, Microsoftin virtuaalikone .NET -ympäristössä.

CSS	Cascade Style Sheets, erityisesti www-dokumenteille kehitetty tyyliohjejärjestelmä, jonka avulla sivujen ulkoasu voidaan määrittää. CSS-tekniikoita voidaan käyttää mihin tahansa merkkikieleen (*engl.* markup language), kuten esimerkiksi XML -tyyppisiin dokumentteihin.
Debug	Virheentarkistus, jolla voidaan tarkoittaa manuaalisia tai automatisoituja ohjelmistoalustan virheentarkistustoimintoja.
Heikko tyypitys	Yhteen muuttujatyyppiin voi tallettaa useita eri muuttujatyyppejä. Tästä esimerkkinä skriptikielet, joissa tehdyt virheet näkyvät vasta ajon aikana.
Manageroitu koodi	Managed code, Ohjelmakoodia jonka Microsoftin CLR -komentotulkki (tai jokin muu virtuaalikone) kääntää isäntäkoneen ymmärtämään muotoon.
Masterpage	Ylisivu, jonka avulla useilla sivuilla esiintyvä grafiikka ja toiminnallisuus voidaan keskittää kahteen tiedostoon.
IDE	Integrated Developement Environment, ohjelmointialusta, johon kuuluu usein koodieditori, kääntäjä, virheenkorjaustoiminnot ja muita aputyökaluja.
Intellisense	Ohjelmointiapu, joka listaa ohjelmoinin aikana mitä vaihtoehtoja ohjelmoijalla on missäkin tilanteessa käytettävissään.
Skripti	Komentosarja, jonka avulla automatisoidaan tehtäviä ilman varsinaista ohjelmointia.
Unmanaged code	Ohjelmakoodia jonka prosessori ajaa suoraan ilman kääntämistä.
Vahva tyypitys	Jokaisella muuttujalla on tyyppi ja muuttujat voivat saada ainoastaan tyyppinsä mukaisia arvoja (C++, C#, Java).

WF	Windows Workflow Foundation. Graafinen suunnittelu- ja ohjelmointiympäristö, joka on tarkoitettu vuon ohjauksille, dokumenttien tarkistuksille ja toimitusketjujen hallinnalle. Muistuttaa käyttötapauskaaviota.
XML	Extensible Markup Language, merkintäkieli, jonka avulla tiedon merkitys on kuvattavissa tiedon joukossa ja joka auttaa jäsentämään laajoja tietomassoja selkeämmin.
WCF	Windows Communication Foundation, mahdollistaa sovellustenvälisten viestinnän paikallisesti tai verkon yli.
Windows Cardspace	Microsoftin teknologia ja ohjelmointirajapinta käyttäjien digitaaliseen tunnistukseen. Käyttäjien tieto tallennetaan informaatiokortteihin (*engl.* information cards).
WPF	Windows Presentation Foundation. Windows Forms -tekniikoiden modernisoitu versio, joka keskittyy sovelluksen ulkoiseen muokattavuuteen ja animaatioihin.

SISÄLLYS

1 ALKUSANAT 9

2 JOHDANTO 1

3 VISUAL STUDIO 2008 3

3.1 IDE navigaattori 6

3.2 Käyttöliittymäikkunoiden uudistunut ankkurointi 6

3.3 Jaettu näkymä 7

3.4 Tyylien hallinta 9

3.5 Ylisivut 16

3.6 Virheentarkistus 18

3.7 Muita muutoksia 20

3.8 Visual Studio 2008 Service Pack 1 22

4 .NET FRAMEWORK 23

4.1 CLR (*engl.* Common Language Runtime) 24

4.2 .NET Luokkakirjastot 26

4.3 ASP.NET 28

4.4 LINQ 30

4.4.1 Var-tietotyyppi 34

4.4.2 LINQ-perusteet ja LINQ-oliotekniikat (*engl.* Linq to Objects, LtO) 34

4.4.3 LINQ ja XML (engl. Linq to XML, LtX) 47

4.4.4 LINQ ja SQL-tietokannat (*engl.* Linq to SQL, LtS) 52

4.4.5 LINQ ja suorituskyky 59

4.5 ASP.NET kontrollit ja LINQ 65

4.5.1 Taulukkonäkymä ja ilmentymänäkymä 66

4.5.2 Toistin 68

4.5.3 Listanäkymä 71

4.5.4 LINQ-datalähdekontrolli ja listanäkymä 73

4.5.5 Sivutuskontrolli 77

4.5.6 Sessio ja osoiteriviparametrit 78

4.6 .NET AJAX 80

4.7 Visual Studio 2010 ja .NET Framework 4.0 88

5 STEP-BY-STEP -OHJE LINQ TO SQL -OHJELMISTOON 90

6 ASP.NET JA OHJELMOIJAN TIETOTURVA 118

7 POHDINTA 121

1 ALKUSANAT

Tässä opinnäytetyössä käsitellään Visual Studio 2008:n ja ASP.NET 3.5:n uusia ominaisuuksia ja tekniikoita. Vaikka pääpaino on uusilla ohjelmointimekanismeilla, tarkoituksena on myös esitellä vanhempia malleja, varsinkin jos niiden esittely luo pohjan uudemmille.

Tämän työn toimeksiantajana toimii Kajaanin ammattikorkeakoulu ja työn aiheen käsittely on katsottu olevan tärkeä niille, jotka työskentelevät tai aikovat työskennellä ASP -ympäristöissä. VS 2008:n ja .NET 3.5:n julkaisusta on tämän työn kirjoitushetkellä kulunut melkein puolitoista vuotta ja todennäköisesti suuri osa ASP-sovelluskehittäjistä on siirtynyt näihin päivitettyihin ohjelmointialustoihin.

Tämän opinnäytetyön tehtävänä on tarjota aiheeseen sekä teoreettista pohjaa että runsaasti käytännön esimerkkejä. Tämän työn laatija oli, Kajaanin ammattikorkeakoulun työharjoittelu mukaanlukien, noin vuoden yrityksessä (Luovanet Oy), joka laati web-sovelluksia ja -portaaleja pääasiassa ASP-tekniikoilla. Yritys vaihtoi ohjelmointialustansa välittömästi VS 2008:aan sen ilmestyessä ja jo silloin oli selvää että tämä opinnäytetyö tulisi käsittelemään juuri näitä uusia sovellustekniikoita.

Tulleesta projektikokemuksesta huolimatta tässä työssä ei käytetä apuna eikä esitellä suurempaa työaikana laadittua ohjelmistoa tai sen osaa. Käytännön osuutena tässä työssä toimivat koodiesimerkit ja step-by-step -ohjelmointiohje, joka on opas nykyaikaiseen ADO -ASP -ohjelmointiin Visual Studio 2008 -ympäristössä. Kaikki ohjelmakoodi tässä työssä on toteutettu itse omilla ohjelmistoalusoilla, omalla ajalla.

Jos tämä työ olisi tehty jonkin työsuhteen aikana laadittujen projektien pohjalta, suuri osa tätä työtä käsittelisi kyseisen projektin toimintaa ja itse tekniikoiden esittelyyn jäisi vähemmän tilaa. Lisäksi pelkässä teoriaosuudessa itse ohjelmointityössä käytettyjen menetelmien selittäminen olisi jäänyt todennäköisesti lyhyemmäksi kuin nyt, kun pidempää ohjelmistoprojektia ei käytetä. Perustoimintojen lisäksi tekniikoista on tarkoitus tutkia ja esittää myös syvällisempiä skenaarioita.

2 JOHDANTO

Vuonna 2005 Visual Studio ja Visual Studio Team Systems muuttivat koko .NET -arkkitehtuurin ja siihen tehtyjen sovellusten laadinnan täysin uudenlaiseksi. Visual Studio 2005 toi mukanaan esimerkiksi valmiit koodikappaleet, itsemääritettävät projektipohjat, datayhteyksien luomista helpottavat velhot, ohjelmointiavut (*engl.* intellisense) sekä mallinnus-, testaus-, projekti ja tiimityökalut.

Visual Studio 2008 on rakennettu VS 2005:en päälle ja se sisältää monia parannuksia, uudistuksia sekä kokonaan uusia tekniikoita itse ohjelmointiytimeen ja ohjelmointiympäristöön. Näistä ehkä suurimmat kokonaisuudet ovat WPF, WCF ja WF. Nämä tekniikat tulivat suuren yleisön käyttöön jo Windows Vistan julkistuksen myötä .NET -ohjelmointikehyksen versiossa 3.0, mutta niitä ei kuitenkaan saanut päivityksinä Visual Studio -ohjelmointialustaan, vaan ne tuli asentaa VS:n versioon 2005 lisäosina. Lisäksi VS 2008 ja sen .NET-Framework 3.5 sisältävät natiivisti uusia ohjelmointikielellisiä työkaluja kuten LINQ (Language Integrated Query), lisää uusia tiimityökaluja, uusia ASP-kontrolleja ja -tekniikoita sekä pinnan alla suuren määrän uutta ohjelmakoodia. Kaikkien uudistusten tehtävänä on parantaa ohjelmoinnin tuottavuutta ja helpottaa projektien edistymistä.

Uusien työkalujen ja toiminnallisuuden lisäksi VS 2008:aan sisällytetty .NET Framework 3.5 sisältää mm. suoran tuen Office-tuoteperheen sovelluksiin, Windows CardSpace -tekniikat ja kokonaan uuden CLR-komponentin.

Uusi Visual Studio tekee entistä enemmän asioita käyttäjän puolesta. Parhaimmillaan tämä tarkoittaa sitä, että koska saman päämäärän saavuttamiseksi ohjelmoijan tarvitsee tehdä vähemmän koodirivejä, myös virheiden määrä ja niiden korjaamiseksi tarvittava aika pienenee. Näin päämäärään päästään entistä nopeammin. Vaikka nämä uudet, projektien etenemistä nopeuttavat työkalut vievät kokeneeltakin tekijältä aikaa oppia, on niiden tuoma hyöty monikertainen niiden opetteluun menevään aikaan verrattuna.

.NET Framework 3.0 ja 3.5 sekä Visual Studio 2008 ovat hyviä esimerkkejä siitä, miten ohjelmointiala muuttuu ja elää jatkuvasti. Tämä voi tehdä ohjelmoijan työstä välillä liiankin haastavaa ja koko ajan on kyettävä pysymään tuoreimpien tekniikoiden tasolla, mutta se on usein myös palkitsevaa.

Tässä työssä käsitellään Visual Studio 2008:n ja ASP.NET 3.5:n web-kehityksessä tarvittavia ominaisuuksia ja verrataan niitä aikaisimpiin versioihin. Keskipisteenä ovat uudet tekniikat kuten Visual Studion ulkoiset muutokset, sisäkkäiset ylisivut (*engl.* masterpage), uudistunut ohjelmointiapu, projektien luominen ja uudet virheentarkistustoiminnot (*engl.* debug) sekä lopuksi mitä uutta Visual Studio 2008 Service Pack 1 on tuonut mukanaan.

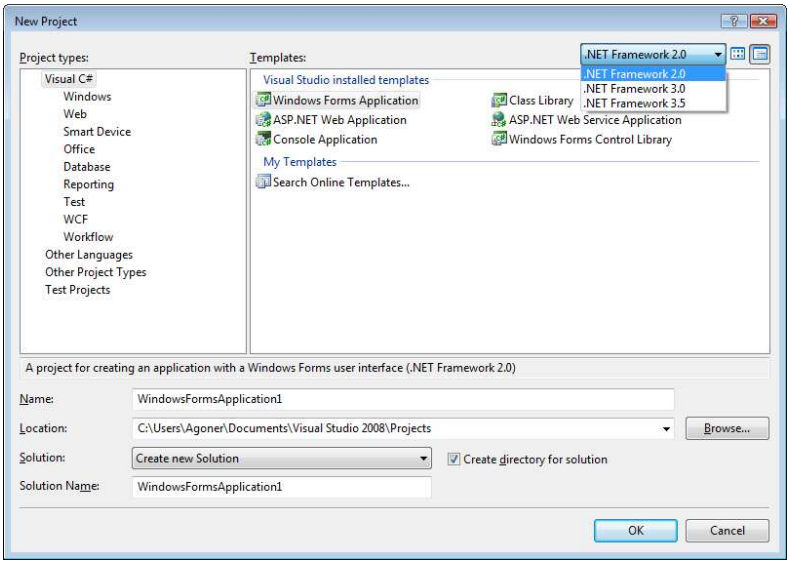
ASP.NET 3.5-ympäristöstä käsitellään LINQ-tekniikoiden lisäksi uusia AJAX- ja ASP -komponentteja, tehdään katsaus CLR-komponenttiin ja .NET luokkakirjastoihin sekä tutkitaan tulevaa .NET Framework versiota 4.0.

3 VISUAL STUDIO 2008

Ensi näkemältä uusi Visual Studio näytä versiosta 2005 paljoa muuttuneen. Microsoftin uusi IDE tuo .NET Frameworkin kanssa kuitenkin satoja uusia ominaisuuksia jo muutenkin laajaan ohjelmointiympäristöön. Visual Studio 2008 korostaa versioon 2005 verrattuna tehokasta ohjelmointia, projektinhallintaa ja tiimityöskentelyä. Tämän lisäksi uusi IDE painottuu yhä enemmän olemaan yksi työkalu moneen tehtävään. (Powers 2008, 6.)

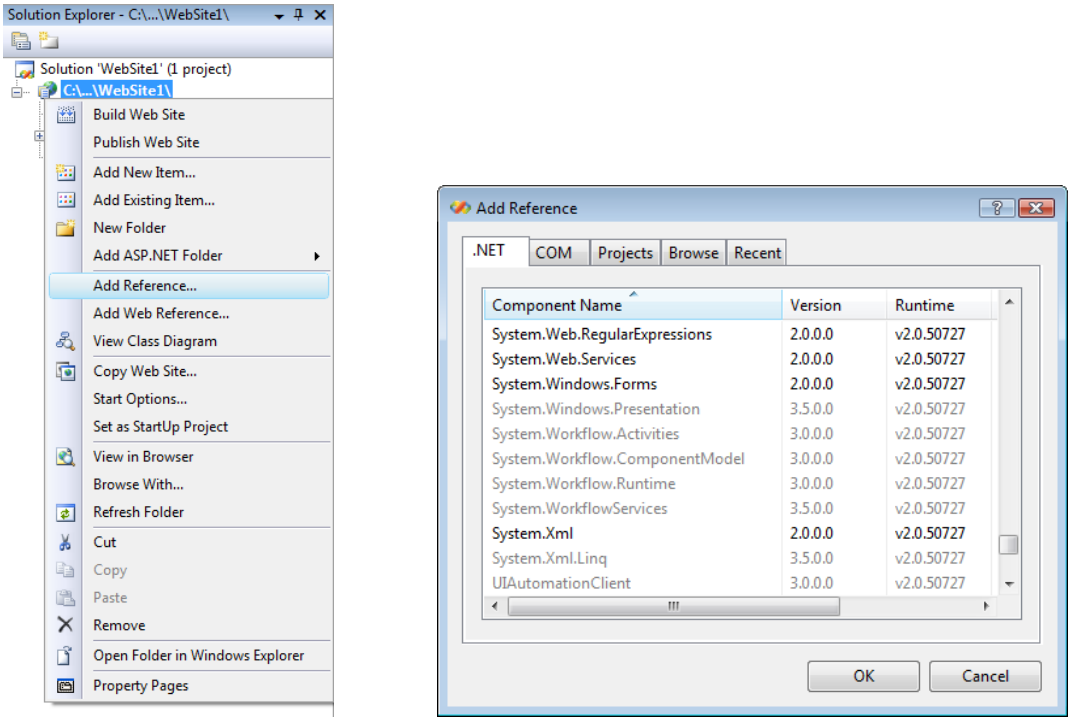
Usein .NET -ympäristön sovelluksia ja verkkopalveluita tehdään jo olemassa olevien kokonaisuuksien päälle. Tällöin on tärkeää että moduuleja voidaan rakentaa juuri siihen haluttuun .NET Frameworkiin, jota tarvitaan. Jos haluaa käsitellä usean eri Frameworkin projekteja, ei ole mielekästä, että tarvitaan Visual Studion eri versioita. Tämän vuoksi Visual Studio 2008 antaa mahdollisuuden luoda uusia projekteja ja projektitiedostoja juuri sen Frameworkin päälle mitä käyttäjä haluaa käyttää. Frameworkin versio määrää esimerkiksi sen, mitä työkaluja, projekti- ja tiedostotyyppejä sekä referenssejä (COM -komponentteja, aliohjelmia) ohjelmoijalla on käytettävissään. Framework vaikuttaa myös ohjelmointiavun sisältöön. (Powers 2008, 7.)

Kuviossa 1. näkyy uuden projektin luominen Visual Studio 2008:ssa (File - New - Project) ja .NET Framework -version valinta dialogin oikean ylänurkan alasvetovalikosta (Powers 2008, 7).



Kuvio 1. Uuden projektin luominen Visual Studio 2008:ssa

Kun projekti on luotu haluttuun Framework-versioon, voidaan referenssivalikko avata projektikokoelman (*engl.* solution) projektin nimen alta aukeavasta valikosta (kuvio 2.) (Powers 2008, 8).

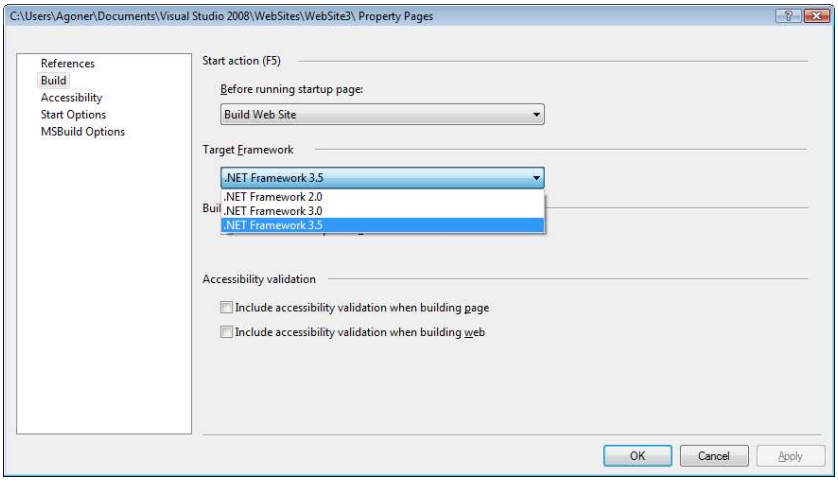


Kuvio 2. Referenssin lisääminen (vasemmalla) ja referenssivalikko

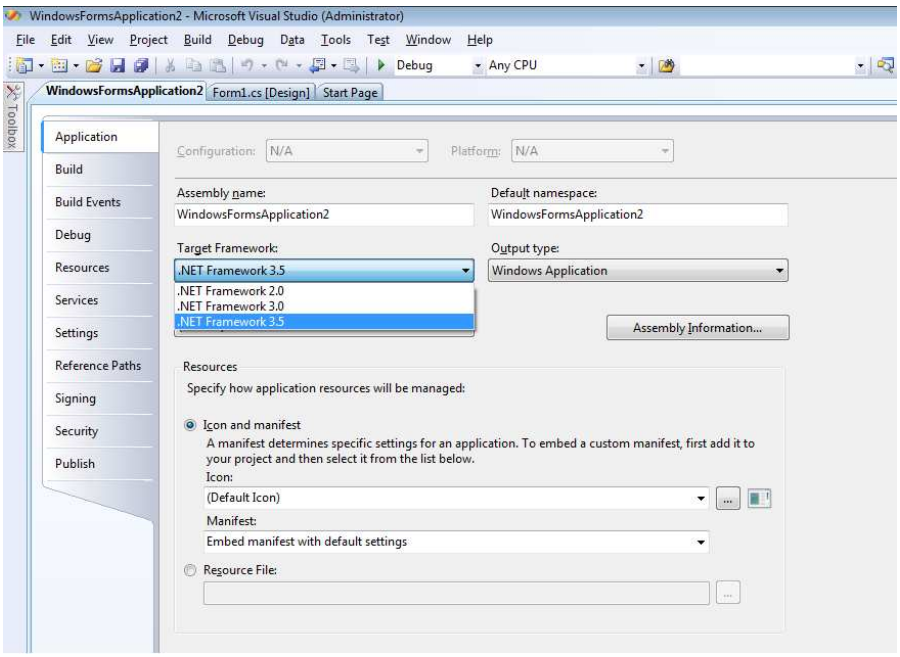
Kuviosta 2. nähdään, että jos projekti on luotu Frameworkin versioon 2.0, kaikki referenssit jotka vaativat toimiakseen Frameworkin versiota 3.0 tai 3.5 (tai uudempaa), ovat harmaana. Näitä referenssejä ei voi lisätä projektiin. Toisaalta, jos projekti olisi luotu versioon 3.0, Frameworkin versioon 3.5 kuuluvat referenssit olisivat harmaina, mutta version 2.0 ja 3.0 referenssit olisivat käytettävissä. Luonnollisesti Frameworkin version 3.5 projekteissa kaikki referenssit ovat valittavissa. (Powers 2008, 9.)

Projektin Framework-version voi muuttaa myös projektin luomisen jälkeen. Silloin Visual Studio muuttaa joidenkin tiedostojen määritteitä ja lisää tai poistaa referenssejä, riippuen siitä ollaanko projektia muuttamassa vanhempaan vai uudempaan Frameworkiin. Tämän vuoksi näiden Framework-päivitysten kanssa tulee olla erityisen huolellinen, koska vanhempien tekniikoiden kanssa voi syntyä ristiriitoja. Päivitettäessä aiempaan versioon, on mahdollista, että Visual Studio ei osaa tehdä kaikkia tarvittavia muutoksia, jolloin osa muutoksista on tehtävä itse. Koska Visual Studio ei voi poistaa jo mahdollisesti tehtyjä uudempien Frameworkien toiminnallisuuksia, myös nämä koodirivit on muutettava tai poistettava itse.

Visual Studion kääntäjän ansiosta projekti ei käänny, ennen kuin nämä virheet on korjattu. Kuvioissa 3. ja 4. näkyvät valikot on tarkoitettu sovellusten Framework-pohjan muuttamiseen jälkikäteen. Valikot löytyvät projektin ominaisuuksista painamalla sen päällä hiiren oikeaa painiketta ja valitsemalla valikosta properties-kohta. (Powers 2008, 9-10.)



Kuvio 3. Framework-version vaihtaminen web-sovelluksessa

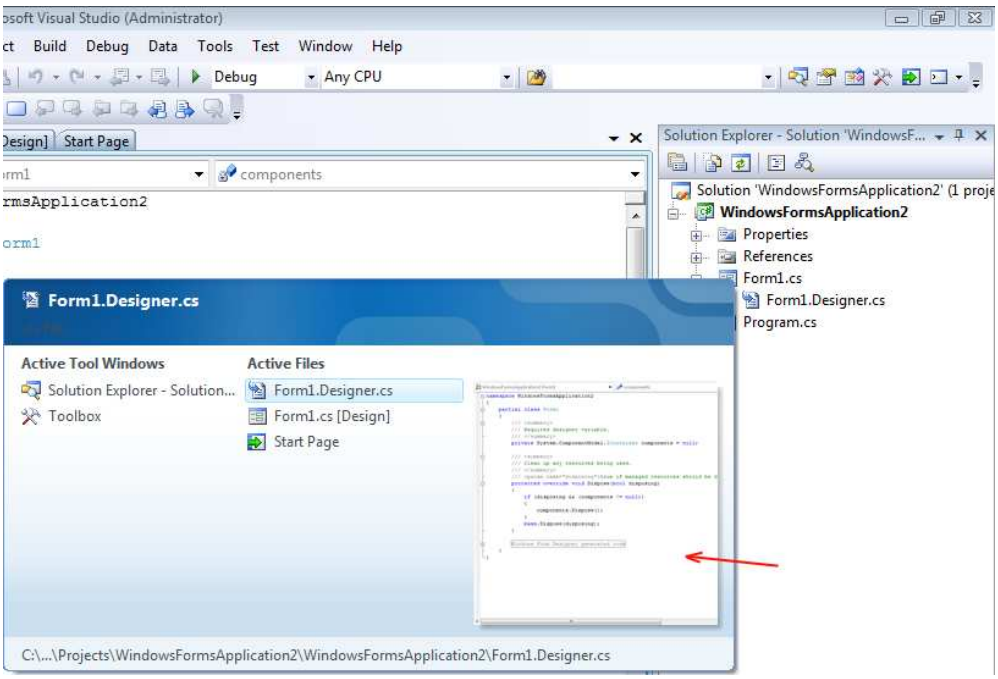


Kuvio 4. Framework-version vaihtaminen windows forms -sovelluksessa

Uudistuneen Frameworkin lisäksi myös itse Visual Studio on saanut ulkoisia ja toiminnallisia uudistuksia. Vaikka suurin osa myös Visual Studio 2008:n ulkoisista uudistuksista keskittyvät tukemaan Frameworkin versioita 3.0 ja 3.5, löytyy siitä muutakin uutta kuin natiivi projektituki LINQ-, WPF-, WCF- ja WF-projektityypeille. (Powers 2008, 8-10.)

3.1 IDE navigaattori

Visual Studiossa on nyt mahdollista selata avattuja tiedostoja Windows -käyttöjärjestelmien ALT-TAB -tyylisesti. Tämä tapahtuu Visual Studio 2008:ssa eteenpäin näppäinyhdistelmällä CTRL-TAB ja taaksepäin CTRL-SHIFT-TAB (kuvio 5.). (Powers 2008, 10-11.)

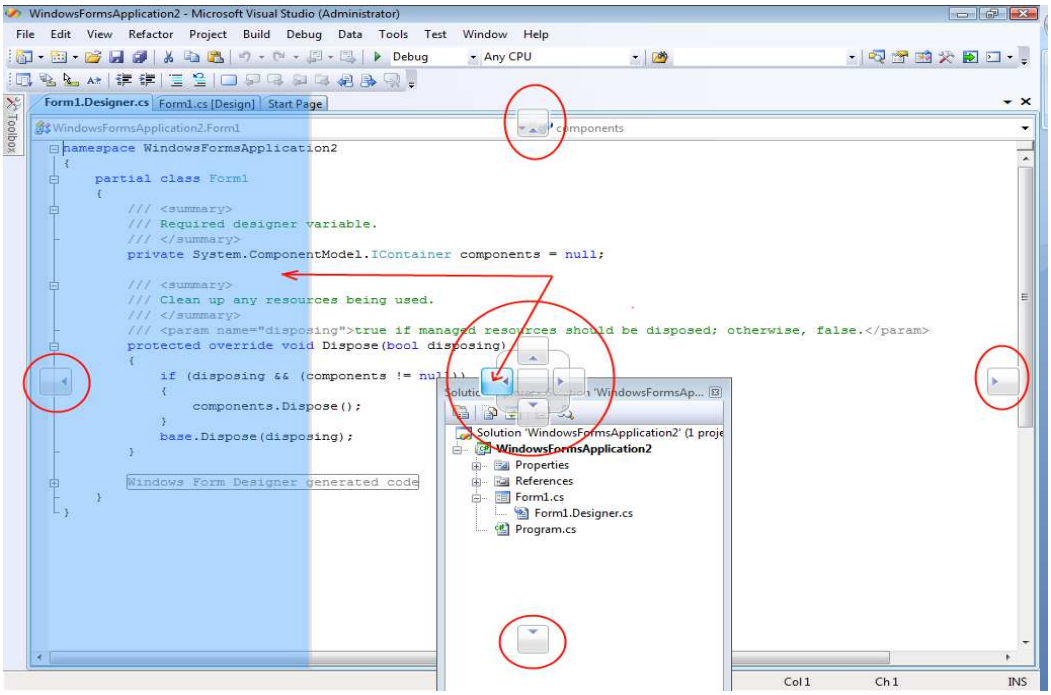


Kuvio 5. Avoimien tiedostojen selausvalikko

Kuviosta 5. näkyy aktiivisen tiedoston esikatseluruutu, joka helpottaa navigointia jos auki on suuri määrä kooditiedostoja. Näppäinten vapauttaminen aktivoi valitun tiedoston, vastaavasti kuin Windows-käyttöjärjestelmän ALT-TAB -toiminnallisuudessa. Valikon navigoinnissa voi käyttää apuna hiirtä tai nuolinäppäimiä. (Powers 2008, 10-11.)

3.2 Käyttöliittymäikkunoiden uudistunut ankkurointi

Visual Studio 2008 tarjoaa mahdollisuuden entistä monipuolisempaan käyttöliittymän ikkunoiden järjestelyyn ja ankkurointiin. Uusi ankkurointikäyttöliittymä näyttää selkeästi, minne irrotetun ikkunan voi raahata ja lukita (kuvio 6.). (Powers 2008, 20.)

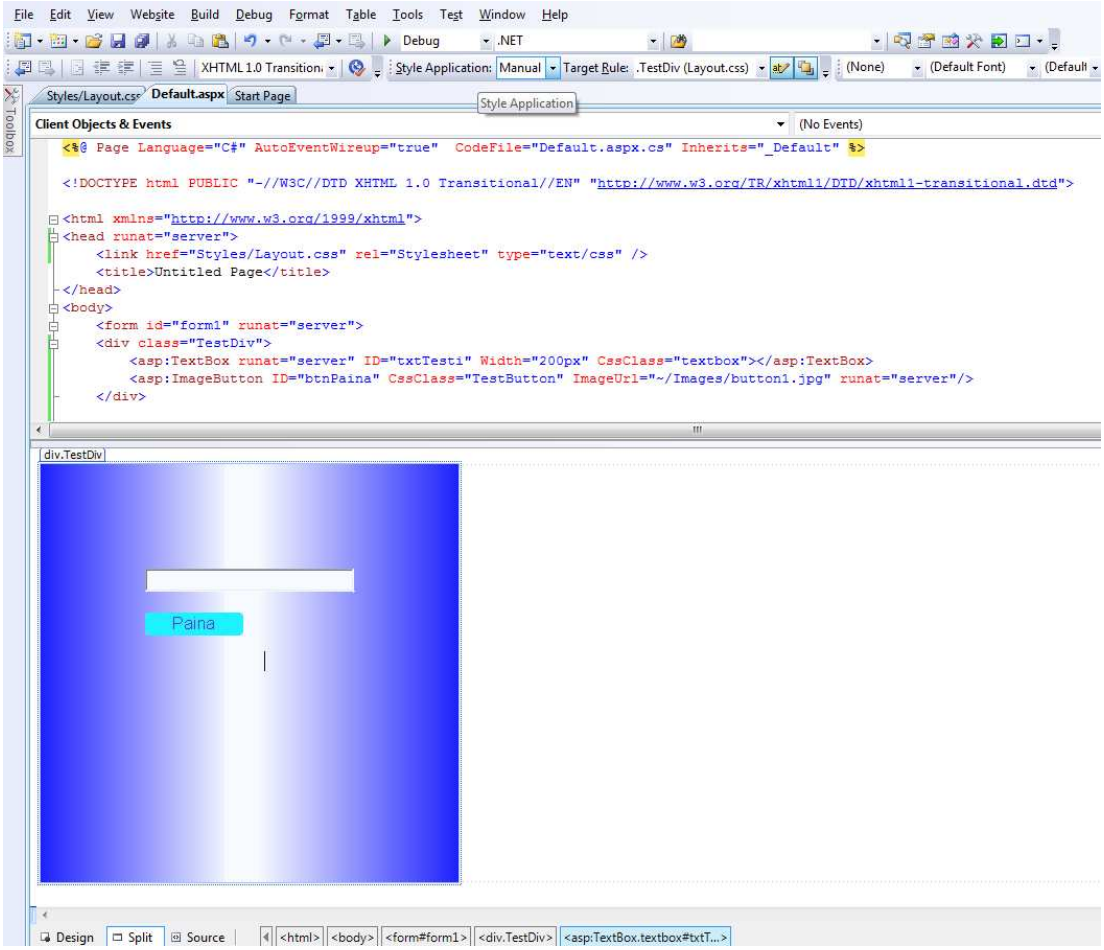


Kuvio 6. Ikkunoiden ankkurointinäkymä

3.3 Jaettu näkymä

Osa Visual Studiolla työskentelevistä web-kehittäjistä pitävät työskentelystä graafisen näkymän (*engl.* design view) ja koodinäkymän (*engl.* source view) välillä. Koodinäkymä antaa mahdollisuuden sivun XHTML-koodin täydelliseen hallintaan. Graafinen näkymä taas antaa kehittäjän nähdä sivun kehitysaikaisen ulkoasun sitä mukaa kun sivua rakennetaan. Graafinen näkymä antaa myös monia visuaalisia oikopolkuja sivun komponenttien ja asetusten muuttamiseen. (Powers 2008, 23.)

Visual Studio 2008 tekee vaihtelun graafisen näkymän ja ASP-koodin välillä helpommaksi, koska nyt niitä voi katsella samassa ikkunassa yhtäaikaisesti (kuvio 7.) (Powers 2008, 23).



Kuvio 7. Jaettu näkymä koodialueen (kuviossa ylhäällä) ja graafisen suunnittelutilan välillä

Sivun graafinen ulkoasu ei näytä aina samalta Visual Studiossa ja Internet-selaimessa. Graafisen näkymän toimivuus tai toimimattomuus ei sikäli ole vakava asia, koska sivut tulee aina testata asianmukaisilla selaimilla jo kehitysvaiheessa. Eri selainten grafiikanpiirtomoottorit eroavat toisistaan (myös saman selaimen eri versioiden), jolloin sama sivusto voi näyttää eri selaimilla hyvinkin erilaiselta. FireFox- ja Internet Explorer -selainten lisäksi sivut testataan myös usein Safarilla ja Operalla. (Microsoft 2009 i.)

Desing-näkymän riesana ovat myös suorituskykyongelmat, sillä esimerkiksi koodinäkymästä designeriin siirryttäessä edessä on pahimmassa tapauksessa koko ohjelman jumiutuminen vastaamattomaan tilaan. Visual Studio 2008:n Service Pack 1 -päivitys nopeuttaa ja parantaa myös graafisen näkymän toimivuutta. (Microsoft 2008 g.)

3.4 Tyylien hallinta

Tyylimääritteet ovat olleet arkipäivää web-ohjelmoinnissa jo HTML-tekniikoiden alkuajoista asti. Aluksi tyylit määritettiin suoraan HTML-komponenteissa erillisinä lohkoina, kuten seuraavassa esimerkissä, jossa 1-tason otsikon (h1) väri on muutettu siniseksi. (Powers 2008, 533.)

```
<font color="blue">
   <h1>Tämä on otsikko</h1>
</font>
```

Edellisen esimerkin tavoin määriteltyjen tyylien hallinta ja ylläpito on pienemmissäkin projekteissa lähes ylivoimaista. Tyylien hallinta siirtyi keskitetympään suuntaan kun CSS -tekniikat julkistettiin. Seuraavassa esimerkissä ensimmäisen tason otsikkoon (h1) on määritetty CSS-tekniikoilla sinisen kirjasinvärin lisäksi ominaisuus *cursor*. Tämä ominaisuus muuttaa hiiren osoittimen ulkoasua, kun hiiri viedään HTML-komponentin päälle. (Powers 2008, 533.)

```
<h1 style="color:Blue; cursor:pointer;">Tämä on otsikko</h1>
```

Edellisen esimerkin tapa ei helpota tyylien luettavuutta tai ylläpitoa, vaan se antaa ohjelmoijalle huomattavasti suuremman kirjon eri tyylivaihtoehtoja. CSS 2 sisältää yli sata tyyliominaisuutta, mutta yhtä tärkeä ominaisuus sillä on tyylien selvästi parantunut hallittavuus. CSS mahdollistaa tyylien määrittämisen keskitetysti joko suoraan sivulle <style> -lohkojen sisälle tai erilliseen CSS-tiedostoon. Näistä vaihtoehdoista tiedoston käyttäminen parantaa enemmän koodin luettavuutta ja toimivuutta, koska tyylimääritteet eivät ole samalla alueella varsinaisen HTML-koodin kanssa. Suoraan sivulle määritetyt tyylit ovat käytössä vain sillä sivulla, mutta CSS-tiedostoon määritetyt tyylit ovat käytettävissä koko projektissa. (Powers 2008, 533.)

Seuraavassa esimerkissä tyylit on määritetty suoraan ASP-sivulle (Powers 2008, 533).

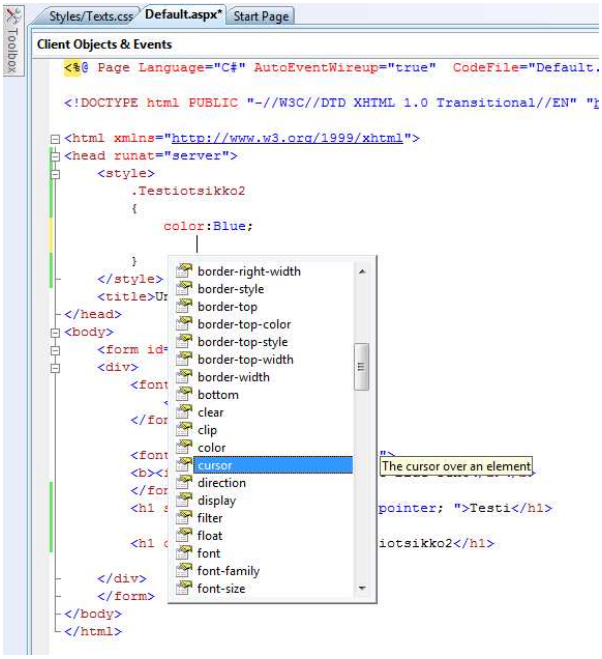
```
<head runat="server">
    <style>
        .Testiotsikko
        {
            color:Blue;
            cursor:pointer;
        }
    </style>
    <title>Untitled Page</title>
</head>
```

Seuraavassa esimerkissä edellisen esimerkin tyyliä (Testiotsikko) käytetään HTML -komponentissa (Powers 2008, 533).

```
<h1 class="Testiotsikko">Tämä on otsikkoteksti</h1>
```

Kuten edellä olevasta esimerkistä nähdään, tässä tyylinmääritystavassa tyylit kirjoitetaan HTML-sivulle style-lohkojen sisälle. Esimerkistä on hyvä huomata, että käytettäessä tyylimäärityksiin html-komponentin class-ominaisuutta, pitää tyylimäärityksessä tyylin nimen eteen sijoittaa piste. (Powers 2008, 533-534.)

Visual Studio 2008:ssa Style-tagien sisällä HTML- tai ASP-sivuilla on käytössä ohjelmointiapu (kuvio 8.) (Powers 2008, 534).



Kuvio 8. Ohjelmointiapu style-lohkojen sisällä

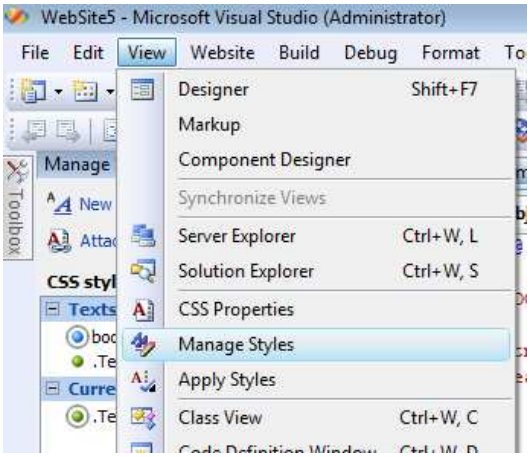
Style-lohkojen sisälle määritetyt tyylit ovat käytettävyydeltään keskitetympiä kuin kontrolleihin yksilöidyissä tyyleissä. Silti käytettävyyden kannalta paras tulos saadaan erillisen .css-tiedoston avulla, johon style-lohkon sisältö kirjoitetaan. Sivuille, joissa tyylitiedoston tyylejä aiotaan käyttää, tulee head-lohkon sisälle lisätä viittaukset tähän tyylitiedostoon link -komponentin avulla, seuraavan esimerkin tavoin. (Powers 2008, 533-534.)

```
<link href="/Styles/Texts.css" rel="Stylesheet" type="text/css" />
```

Edellisessä esimerkissä link-komponentin href-ominaisuus on projektissa oleva hakemistopolku, josta haluttu tyylitiedosto löytyy. (Powers 2008, 534.)

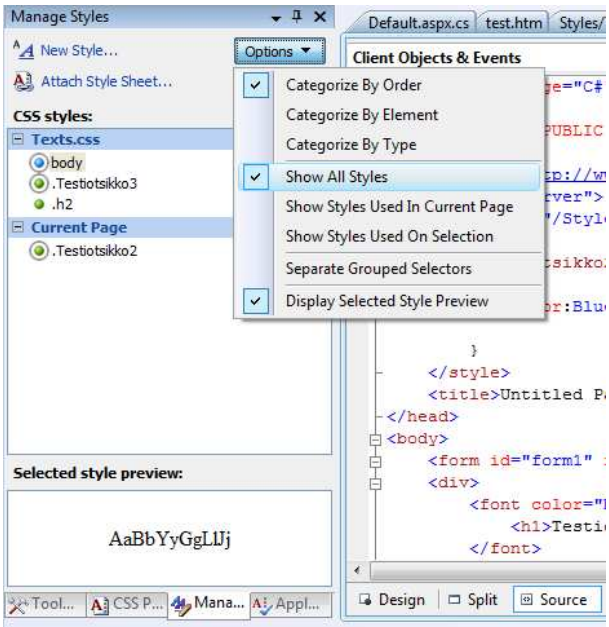
CSS-tekniikat ja parantuneet ohjelmointikehitysympäristöt ovat helpottaneet ja monipuolistaneet Internet-sivujen ulkoasujen laatimista. Nykyaikaiset Internet-sivut sisältävät kuitenkin valtavan määrän tyylimäärityksiä, jolloin niiden hallinta on haastavaa. Visual Studio 2008 tarjoaa tyylien hallintaan uusia työkaluja, jotka mahdollistavat tyylien luomisen ja muokkaamisen suoraan HTML- ja ASP-sivuille sekä tyylitiedostoihin. (Powers 2008, 24.)

Dialogit tyylien hallinnalle on löydettävissä Visual Studio 2008:n yläpalkin View -valikosta kohdista **CSS Properties**, **Manage Styles** ja **Apply Styles** (Kuvio 9.). Näiden avulla voidaan luoda ohjatusti uusia tyylejä, muokata ja esikatsella niitä sekä liittää niitä sivuille. Uudet työkalut auttavat määrittelemään mitä ja millaisia tyylejä sivuilla halutaan nähdä. (Powers 2008, 537.)



Kuvio 9. CSS-työkalut

Tyylien muokkausikkuna (*engl.* manage styles) mahdollistaa tyylitiedostojen liittämisen sivuille, uusien tyylien luomisen, niiden muokkaamisen ja esikatselun. Kuviossa 10. olevasta ikkunasta voidaan nähdä linkit uusien tyylien ja tyylitiedostojen luomiseen (*engl.* New Style) ja tyylitiedostojen liittämiseen halutulle sivulle (*engl.* Attach Style Sheet). Options-painikkeen alta löytyvän valikon ominaisuudet vaikuttavat vain tässä ikkunassa näkyvien tyylien näkyvyyteen ja lajitteluun. Valikosta voidaan valita esimerkiksi näytettäväksi vain aktiivisena olevaan sivuun vaikuttavat tyylit (*engl.* show styles used in current page) tai lajitella tehdyt tyylit tekojärjestyksen mukaan (*engl.* categorize by order). Nämä tyylit näkyvät valitussa järjestyksessä ikkunan keskiosassa CSS styles -otsikon alla (kuvio 10.). (Powers 2008, 537 -538.)



Kuvio 10. Tyylien muokkausikkuna

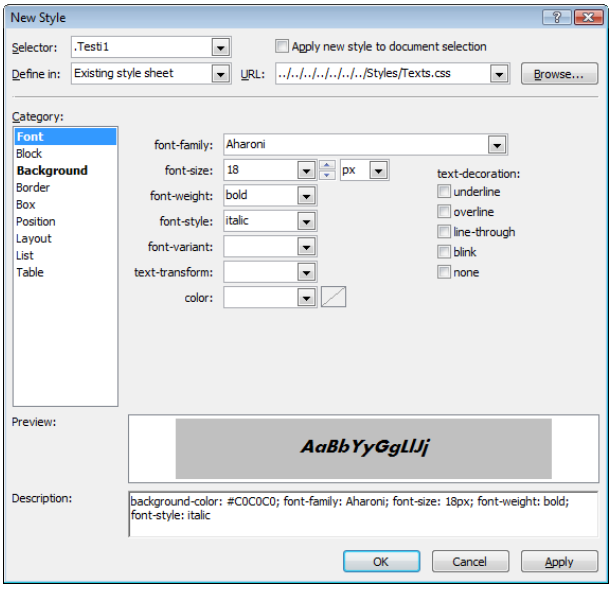
Omia tyylejä voi edelleen tehdä, mutta niiden laatiminen vaatii tyylisyntaksin hyvää hallintaa, koska Visual Studion ajonaikainen virheentarkistus ei kata CSS-tiedostoja ja tyylimäärityksiä. Jos tyylimääritykset on tehty virheellisesti, sivu toki toimii, mutta virheelliset tyylit eivät näy. Visual Studio näyttää myös tyylimääritysten kohdalla jos kyseisessä koodirivissä on jotain vikaa. Kun kursori viedään virhekohdan yläpuolelle, näytetään huomiotekstissä virheen syy (kuvio 11.). (Powers 2008, 539.)

```
test
{
    border-bottom:1px;
    border-bottom-style:1px;
}
```

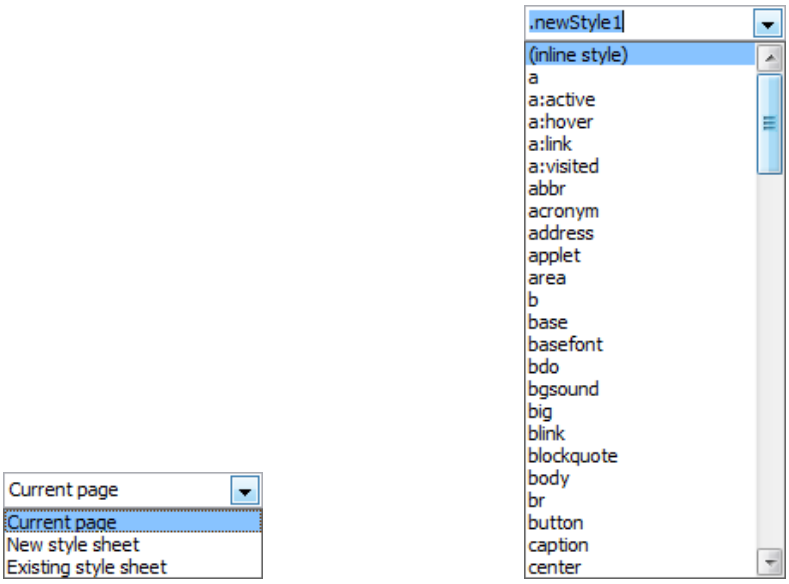
Validation (CSS 2.1): '1px' is not a valid value for the 'border-bottom-style' property.

Kuvio 11. Syntaksivirhe tyylimäärityksessä virhetekseineen

Tyylien muokkausikkunan New Style -linkin alta aukeava dialogi-ikkuna (kuvio 12.) on työkalu tyylimääritysten automaattiseen luomiseen. Kun halutut määritteet on valittu, voidaan tyyli hyväksyä dialogin Apply-painikkeesta, jolloin Visual Studio generoi valintoja vastaavat tyylikoodirivit. Tyylien luomisdialogin (kuvio 12.) Selector ja Define in -alasvetovalikot (kuvio 13.) määrittävät mihin ja millä nimellä nämä koodirivit kirjoitetaan. Kohdevaihtoehtoja koodille ovat aktiivinen sivu, uusi tyylitiedosto tai jo olemassaoleva tyylitiedosto. Jos valitaan toinen tyylitiedostovaihtoehdoista, URL-kenttään on määritettävä hakemistopolku, johon uusi tyylitiedosto luodan tai josta jo olemassa oleva tiedosto löytyy. (Powers 2008, 537-538.)



Kuvio 12. Dialogi uuden tyylimäärityksen luomiseen



Kuvio 13. Selector- ja Define In -alasvetovalikot

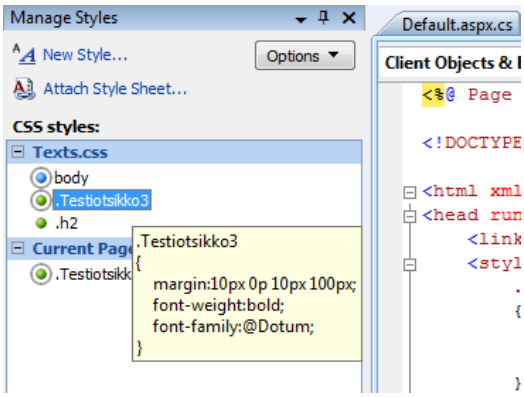
Tyylien luomisdialogin (kuvio 12.) keskiosan valikot on tarkoitettu tyylimäärityksille. Nämä valikot sisältävät käytännössä kaikki CSS-standardin tyylimääritykset jaoteltuna asianmukaisiin kategorioihin, jolloin tyylit on helpompi hahmottaa. Esimerkiksi kuvion 12. lihavotuihin Font ja Background -kategorioihin on jo asetettu jotain tyylimääritteitä. (Powers 2008, 536-537.)

Tyylien luomisdialogin alalaidassa sijaitsee luodun tai muokatun tyylin reaaliaikanen esikatselu (*engl.* preview), jonka ulkoasu muuttuu valittujen tyyliominaisuuksien mukaan. Esikatseluruudun alapuolella on tyylin kuvausalue (*engl.* description), joka ilmaisee miltä valituista ominaisuuksista koostuva tyylikoodi lopullisessa muodossaan näyttää. Esikatseluruudun tavoin myös tämän sisältö muuttuu reaaliaikaisesti muutettaessa tyylikategorioiden ominaisuuksia. OK-painike luo dialogissa määritetyn tyylin. (Powers 2008, 536-537.)

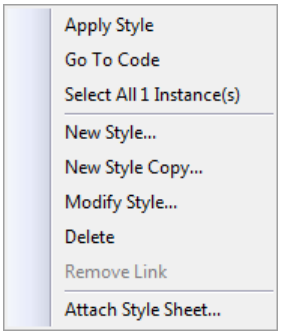
Tyylien muokkausikkunan (kuvio 10.) tyylien liittämislinkki (*engl.* attach style sheet) tuo esiin navigointi-ikkunan, jonka avulla voidaan liittää haluttu tyylitiedosto tietylle sivulle. Toimenpide lisää automaattisesti tyylitiedoston referenssin kyseiselle sivulle. (Powers 2008, 536.)

Muokkausikkunan keskellä oleva CSS styles -osio (kuvio 10.) näyttää mitä tyylejä aktiivisella sivulla on mahdollista käyttää. Tyylien sisällön näkee nopeasti ponnahdusikkunassa pitämällä hetken hiiren kursoria tyylin päällä. Itse tyylin sisältöön pääsee käsiksi joko tyylin otsakkeen

kaksoispainalluksella tai painamalla tyylin kohdalla hiiren oikeaa painiketta ja valitsemalla kohta ”siirry koodiin” (*engl.* go to code) (kuvio 15.). Tällöin tiedosto aukeaa ja siitä esitetään kohta, jossa kyseinen tyylikoodi sijaitsee, oli se itse sivulla tai erillisessä tyylitiedostossa. Jos haluaa muokata tyyliä sille tarkoitetussa dialogissa (kuvio 12.), on se mahdollista tehdä valitsemalla tyylin alavalikon muokkaa tyyliä -kohta (*engl.* modify style). (Powers 2008, 537 -538.)



Kuvio 14. Tyylien muokkausikkuna ja valitun tyylin sisältö ponnahdusikkunassa



Kuvio 15. Manage Styles -ikkunan tyylin alavalikko

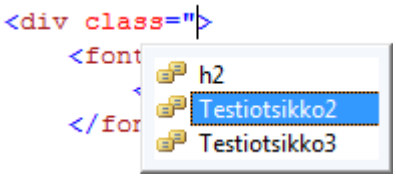
Tyylien liittäminen itse sivujen komponentteihin voidaan tehdä myös kuvion 14. tyylien hallintaikkunan kautta. Kun joku sivun HTML-lohkoista on aktivoitu, voidaan hallintaikkunasta valita halutun tyylin alavalikosta ”lisää tyyli” -kohta (*engl.* apply style). Silloin valittuun HTML-komponenttiin generoidaan tyylin luokkamääritys, kuten nähdään seuraavasta esimerkistä. (Powers 2008, 539.)

```
<div class="Testiotsikko3"></div>
```

Luokkamääritysten liittämiseen on Visual Studio 2008:ssa myös oma tyylien liittämisikkuna (*engl.* apply styles). Tämä ikkuna vastaa lähestulkoon tyylien hallintaikkunaa, mutta sen

toiminnallinen paino on hiukan erilainen. Tyylien liittämisikkunan avulla voidaan nopeasti sijoittaa tyylimäärityksiä haluttuihin web-komponentteihin valitsemalla ensin sivun koodiosiosta haluttu komponentti ja klikkaamalla haluttua tyylimääritystä. Tyylien hallintanäkymästä poiketen liittämisikkunassa olevat tyylimääritykset näytetään suoraan tyyliensä mukaisessa ulkoasussa. (Powers 2008, 539.)

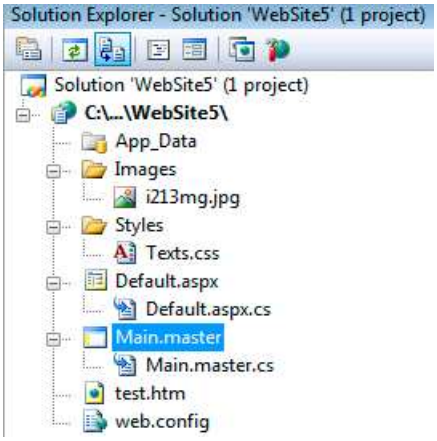
Vaikka Visual Studio 2008 tuo monta uutta valikkonäkymää tyylimääritysten käsittelyn helpottamiseksi, manuaalista sijoitusta ei ole unohdettu. Jos sivun viittaukset tyylitiedostoon on tehty asianmukaisesti, toimii tyylimääritysten apuna ohjelmointiapu, joka näyttää web -komponentteja alustettaessa käytettävissä olevat tyylit (kuvio 16.). (Powers 2008, 539.)



Kuvio 16. HTML-komponentin tyyliluokan määritys ja ohjelmointiapu

3.5 Ylisivut

Ylisivut (*engl.* master page) julkaistiin ensimmäistä kertaa Visual Studio 2005:ssä. Ylisivujen avulla on mahdollista sijoittaa useilla sivuston sivuilla näkyvän ulkoasun ja toiminnallisuuden toteutus kahteen tiedostoon. Nämä tiedostot, joiden päätteinä ovat .master ja .cs, toimivat ASP-sivujen tapaan sivun ulkoasun ja objektien esitysalustana (.master) ja niiden toimintalogiikoiden sijoituskohteena (.cs) (kuvio 17.). Tätä master-sivua ja sen grafiikkaa sekä toiminnallisuutta voidaan käyttää rajaamattomilla määrillä eri sivuja, sen sijaan että sama aspx-, html- ja/tai koodiluokan koodi toistetaan joka sivulle erikseen. Kun jokin aspx-sivu peritään master-sivusta, Visual Studio näyttää design-näkymässä tätä aspx-sivua katseltaessa myös master-sivun kautta sivulla näkyvät osat ja toiminnallisuudet. (Powers Lars 2008, 566.)

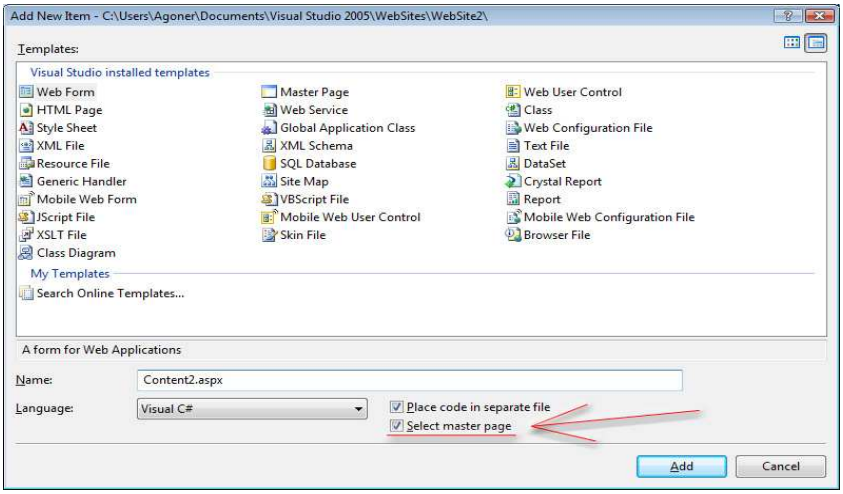


Kuvio 17. Web-projektiin luotu ylisivu ja sen koodiluokka

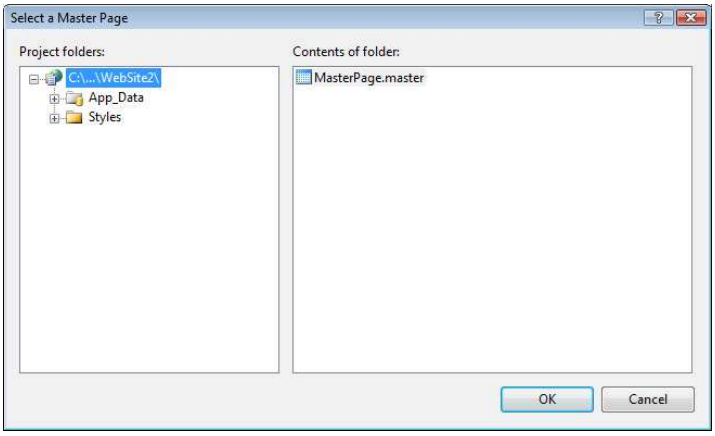
Ylisivu luodaan uuden tiedoston luomisdialogin kautta (*engl.* new item), kuten muutkin tiedostot. Visual Studio luo .master-päätteisen tiedoston sisällön lähes samanlaiseksi kuin tavallisen aspx-tiedoston, joitakin poikkeuksia lukuunottamatta. Erona on form-lohkon sisälle alustettu contentplaceholder-komponentti. (MacDonald 2007, 681-682.)

```
<asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
</asp:contentplaceholder>
```

Paras tapa tutkia ylisivujen toiminnallisuutta on luoda ylisivu, liittää uusi aspx-sivu siihen ja tutkia, mitä ohjelmakoodia Visual Studio tiedostoihin kirjoittaa. Kun sivustolle luodaan uusi aspx-sivu, voidaan tiedoston luomisdialogin alalaidasta (kuvio 18.) valita kohta ”Select master page”. Tämä valinta johtaa Add-painikkeen painamisen jälkeen toiseen dialogiin (kuvio 19.), josta tekeillä olevalle sivulle valitaan haluttu ylisivu. (Powers 2008, 567-568.)



Kuvio 18. Tiedoston luomisvalikko



Kuvio 19. Ylisivun valinta

OK-valinnan jälkeen ylisivun liittäminen aspx-sivuun on valmis. Ylisivu on laadittu samalla tavalla liitteessä 1, jonka esimerkki yhdistää ylisivun ja tyylitiedoston käytön.

Visual Studio 2005 tukee yksinkertaisia ylisivuja. Kuitenkin, jo ASP.NET 2.0 tuki kooditasolla useampia sisäkkäisiä ylisivuja, mutta niiden saaminen toimiviksi graafisessa näkymässä edellytti tarkempaa tietämystä ASP-sivujen latautumis- ja ajosykleistä. (Guthrie Scott 2005.)

Visual Studio 2008 tukee suoraan sisäkkäisiä ylisivuja, joten ylimääräistä toiminnallisuutta ei tarvitse laatia. Kun sivustolle luodaan uusi ylisivu, valintaruutu "Select Masterpage" ei ole Visual Studio 2005:n tapaan poistettuna käytöstä vaan se toimii myös ylisivujen kanssa samalla tavalla kuin luotaessa tavallisia aspx-sivuja. (Guthrie Scott 2005.)

3.6 Virheentarkistus

Ohjelmoijat käyttävät lähes poikkeuksetta yhtä paljon aikaa koodin virheenkorjaukseen ja ajonaikaiseen tutkintaan kuin itse koodin kirjoittamiseen. Tämän vaatimuksen seurauksena Visual Studion virheentarkistustyökalut (*engl.* debugger) ovat yksi Visual Studion laajimmista ja monimutkaisimmista kokonaisuuksista. Vaikka suurin osa Visual Studio 2008:n virheentarkistustyökalujen ominaisuuksista on säilynyt samanlaisena versiosta 2005, voidaan siitä poimia tärkeitäkin uudistuksia. Näitä ovat automaattinen saman projektikokoelman (*engl.* solution) prosessien liittäminen, virheentarkistus etäyhteydellä Windows Vistaan, parannettu tuki monisäikeisten sovellusten, skriptien sekä LINQ- ja WCF -projektien virheentarkistukseen. (Powers 2008, 316.)

Eri prosessien (kuten web-service -prosessien) ajonaikainen virheentarkistus on mahdollista, jos molemmat prosessit ovat kiinnitettyinä toisiinsa. Visual Studion versiossa 2005 tämä liitos (*engl.* attachment) täytyi tehdä käsin, mutta versiossa 2008 riittää, että nämä eri projektit ovat samassa projektikokoelmassa. Visual Studio 2008 tekee tarvittavat liitokset automaattisesti. Toisaalta virheentarkistustyökalut eivät automaattisesti tartu virheisiin, jotka aiheutuvat käynnistetyn projektin ulkopuolella. Tämän vuoksi on järkevää asettaa omia tarkistuspisteitä (*engl.* breakpoint) myös tähän ulkopuoliseen projektiin. (Powers 2008, 232.)

Verkon yli olevan laitteiston sovelluksen virheentarkistus onnistuu vain Visual Studion versioilla Pro ja Team. Tämän toiminnallisuuden käyttöönotto vaatii Remote Debugger -ohjelmiston (msvsmon.exe), joka tulee joko asentaa etäkoneelle tai ajaa se etänä jaetusta kansiosta. Lisäksi laitteiston palomuuri täytyy konfiguroida sallimaan liikenne näiden laitteiden ja Remote Debugger -ohjelmiston välillä. (Microsoft 2008 d.)

Monisäikeisten sovellusten virheentarkistus on mahdollista jo Visual Studion strandard -versiossa Visual Basic, C# ja C++ -kielillä sekä web-kehitysympäristössä. Paremman kielituen ja yhteensopivuuden lisäksi Visual Studio 2008:n säikeiden virheentarkistus ei juurikaan eroa versiosta 2005. Virheentarkistus-työkalupalkkiin on tullut kaksi uutta painiketta, joiden avulla voi merkitä haluamiansa säikeitä tarkempaa ajonaikaista tarkastelua varten. Painikkeiden toiminnot vaikuttavat virheentarkistus-työkaluvalikon säie -osioon ja siihen, miten säikeet tässä listassa esitetään. Visual Studio 2008:n virheentarkistusvalikko sisältää uuden, säikeiden näyttö -painikkeen (*engl.* show threads in source), jonka aktivoiminen näyttää kooditiedoston vasemmassa marginaalissa säie-symbolin jokaisen säikeen sisältävän rivin kohdalla. (Microsoft 2008 e.)

Visual Studio 2008:ssa on mahdollista ajaa virheentarkistukset myös asiakaspään skripteihin, jotka on toteutettu VBScript-, Jscript- tai JavaScript-kielillä. Skriptien virheentarkistuksessa on ajon aikana käytössä samat koodin tarkistuspisteet ja seurantamenetelmät (*engl.* watcher), kuten cs-tiedostojen ohjelmakoodissa. (Powers 2008, 361.)

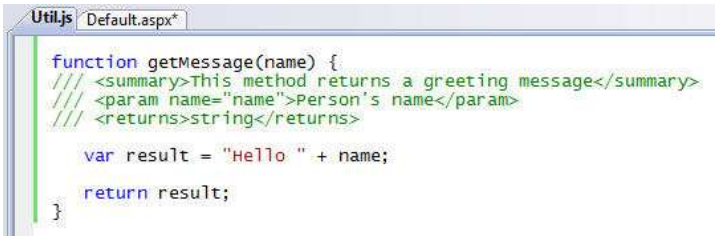
Kaikkien .NET Framework 3.0:n ja 3.5:n uusien projektimallien ja koodirivien (WPF, WCF, WF, LINQ) debuggaus on nyt natiivisti tuettuna (Powers 2008, 361).

3.7 Muita muutoksia

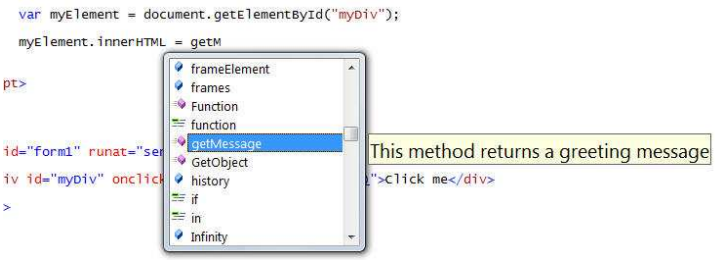
a) JavaScript-koodi ja ohjelmointiapu

Web-kehittäjille tervetullut uusi ominaisuus on sisäänrakennettu ohjelmointiapu .NET -ympäristön JavaScript-koodille (kuvio 20.). Tämä on käytettävissä kaikissa Visual Studio 2008:n versioissa. JavaScript-ohjelmointiapu tukee myös aliohjelmiin kirjoitettuja kommenttilohkoja, kuten käytettäessä muita Visual Studion ohjelmointikieliä (kuvio 21. ja 22.). (Guthrie 2008.)

Kuvio 20. JavaScript-ohjelmointiapu elementissä



Kuvio 21. JavaScript-koodi ja funktiokuvaus



Kuvio 22. JavaScript-koodi ja funktio-ohje

b) Dialogiruudut

Visual Studio 2005 sisälsi täysin sitä varten suunnitellut dialogiruudut. Kaikki tiedostojenkäsittelydialogit, kuten tallennus- ja avausdialogit saivat kritiikkiä, koska ne eivät noudattaneet Windows-käyttöjärjestelmän dialoginäkymiä. Nyt Visual Studio 2008:n dialogivalikot on samanlaisia kuin muissakin Windows-sovelluksissa. (Powers 2008, 12)

c) Globaalit fonttiasetukset

Asetuksissa (Tools-valikko - Environment - Fonts and Colors) on nyt uusi valinta, Globaalit fonttiasetukset (*engl.* environmental font). Tämä vaihtoehto mahdollistaa globaalit fonttiasetukset, jolloin käyttäjä voi halutessaan muuttaa koko käyttöliittymän fontteja ja väritystä, ei pelkästään ohjelmointialueilta, vaan myös valikoista, työkaluista ja ikkunoista. (Powers 2008, 12.)

d) Koodialueen valinta

Visual Studio 2008:ssa on mahdollista suorittaa koodirivien "maalaus" koodieditorin vasemman laidan marginaalista vetämällä. Oikea vetokohta marginaalissa on kapea alue jossa hiiren kursori muuttuu yläviistoon oikealle osoittavaksi nuoleksi. (Powers 2008, 12.)

e) Get ja Set -metodien alustus (Powers 2008, 12.)

Get ja Set -metodien alustus voidaan vanhan tavan lisäksi toteuttaa myös uudella, yksinkertaisemmalla tavalla:

```
public string Description { get; set; }
```

Ero vanhempaan malliin on selvä:

```
public string Description
{
    get { return _sDescription; }
    set { _sDescription = value; }
}
```

3.8 Visual Studio 2008 Service Pack 1

Lokakuussa 2008 julkistetut Visual Studio 2008:n ja ASP.NET 3.5:n Service Pack -päivitykset toivat .NET-ympäristöön runsaasti odotettuja parannuksia ja myös uusia ominaisuuksia. Odotetusti parannukset kohdistuivat enimmäkseen ohjelmistoalustan suorituskykyyn, etenkin WPF- ja WF -ympäristöihin, joiden ajon aikaista suorituskykyä parannettiin jopa 45%. Yleisesti ohjelmointialustan suorituskykyä parannettiin muutamia prosentteja. (Microsoft 2008 g.)

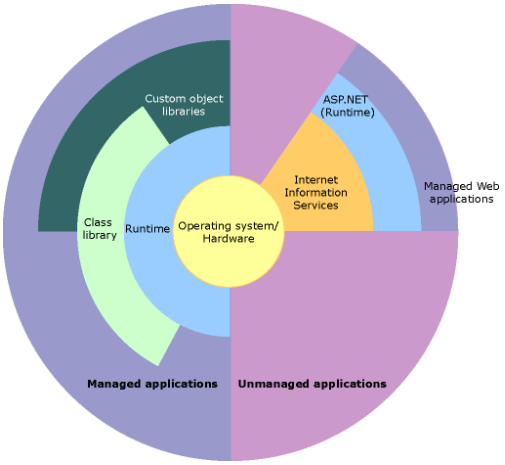
Visual Studio 2008 sai parannuksia esimerkiksi WPF-projektien graafiseen suunnittelunäkymään, JavaScript-tukeen ja Team Systems -ympäristöön. Päivityksen uusia ominaisuuksia ovat natiivituki SQL SERVER 2008:aan, ADO.NET Entity Designer -kokonaisuus ja DataRepeater-kontrolli Windows Forms -ympäristöön. Windows Forms -ympäristön graafiset kontrollit saivat uusia ominaisuuksia ja toiminnallisuuksia, joiden tehtävänä on yksinkertaistaa vanhoja grafiikanpiirtomalleja. Esimerkiksi piirretyn kappaleen häivytysvärien määrittäminen on nyt helpompaa. Service Pack 1 lisäsi Windows Forms -ympäristöön myös PrintForm-komponentin, jonka ansiosta Forms-sovellusten näkymien tulostus voidaan tehdä nyt samoin tavoin kuin ohjelmoitaessa Visual Basic 6.0 -kielellä. (Microsoft 2008 i.)

4 .NET FRAMEWORK

ASP.NET on osa Microsoftin .NET Frameworkia. Jos aikeena on toteuttaa Internet-sivuja ASP.NET-tekniikoilla, on myös ymmärrettävä ja hallittava .NET Framework -kokonaisuutta. Seuraavassa listassa on ne ominaisuudet, joita .NET Frameworkin kuuluu tarjota. (Microsoft 2008 a.)

- Oliokeskeinen kehitysympäristö niin web- kuin työpöytäohjelmistoihin, joita voidaan ajaa sekä paikallisesti että verkon yli
- Ohjelmisto- ja ohjelmointiympäristö, joka minimoi ohjelmistojen levityksessä, asennuksessa ja versioinnissa ilmenevät ristiriidat
- Ohjelmisto- ja ohjelmointiympäristö, jossa voi turvallisesti ajaa ohjelmakoodia oli se sitten omatekoista tai tuntematonta, kolmannen osapuolen laatimaa
- Koodinajoympäristö joka on suorituskykyinen myös skripti- ja tulkkausympäristössä
- Yhtenäinen toimintamalli ja koodaustapa laajalle arsenaalille eri ohjelmistotyyppejä aina web-sovelluksista työpöytä- ja kämmenlaiteohjelmistoihin
- Tietoliikennerajapinta, jonka avulla .NET-ohjelmakoodi voi kommunikoida minkä tahansa muun ohjelmointikielen ja -ympäristön kanssa

.NET Framework sisältää kaksi ASP-ohjelmoijalle keskeistä lohkoa, CLR-komponentin ja .NET-luokkakirjaston (Kuvio 23.) (Microsoft 2008 a).



Kuvio 23. .NET Framework 3.5 arkkitehtuuri

4.1 CLR (*engl.* Common Language Runtime)

CLR on koko .NET Frameworkin perusta. Se manageroi ohjelmakoodin ajonaikaisia tapahtumia ja tarjoaa ohjelmiston tarvitsemat ydinpalvelut, kuten automaattisen muistin- ja säiehallinnan. Se hoitaa myös ajettavan ohjelmakoodin ja toimintojen tietoturvan, sekä koodin eheyteen ja toimivuuteen liittyvät taustatoiminnot. Koodin ajonaikainen hallinta on koko CLR:n tärkein ominaisuus. Koodia, joka käyttää ajossaan CLR-komponenttia, kutsutaan manageroiduksi koodiksi (*engl.* managed code). Vastaavasti manageroimaton koodi (*engl.* unmanaged code) ei käytä CLR-komponenttia, vaan koodi ajetaan suoraan käyttöjärjestelmässä. (Microsoft 2008 a.)

Manageroidun koodin vahvuutena on edellä mainittujen ominaisuuksien lisäksi esimerkiksi roskien keräys (*engl.* carbage collection, automaattinen muistin vapauttaminen), muuttujatyyppien varmistus ja raja-arvojen tarkistus sekä virhetilanteiden käsittely. Ajon aikana manageroitu ohjelmakoodi käännetään aina alustasta riippumattomalle kielelle (*engl.* intermediate language), jonka lisäksi kääntäjä koostaa myös tarvittavan metadatan, joka sisältää symbolisen informaation kaikista koodin metodeista, muuttujista ja ominaisuuksista sisältöineen. Manageroituja koodikieliä ovat esimerkiksi J#, C# ja VB .NET. (Microsoft 2008 a.)

Manageroimaton koodi kohdistuu suoraan laitteiston prosessoriarkkitehtuuriin, eikä CLR -komponenttiin. Tästä johtuen manageroitu koodi käännetään aina etukäteen jollekin tietylle alustalle ja usein myös juuri tietylle arkkitehtuurille tämän alustan sisällä. Jos halutaan laatia manageroimaton koodi jollekkin toiselle alustalle, joudutaan koko tuotos kääntämään uudelleen. Manageroimaton koodi on aina natiivia koodia, joka on aina erilainen jokaisella laitekannalla. Manageroimatonta koodia käytettäessä ohjelmoijan on itse huolehdittava monista ajonaikaisista toiminnoista kuten muistinkäytöstä, tyyppiturvallisuudesta ja tietoturvallisuudesta. Manageroimattomalla koodilla toteutetut ajotiedostot sisältävät binääridataa, joka ladataan suoraan laitteiston muistiin. Manageroimattuja koodikieliä ovat esimerkiksi C++, VB 6.0 ja C. (Microsoft 2003.)

Manageroidun koodin vahvuutena on pitkälle viety automatisointi, joka tekee koodin käytöstä tehokasta. Manageroimattoman koodin vahvuutena on koodin taustatoimintojen täydellinen hallinta ja koodin ajonaikainen nopeus, koska sen ja prosessorikannan väliltä puuttuu yksi suuri käsittelijärajapinta. Nopeasti kehittynyt mageroitu koodi ja sen

ajoympäristöt ovat kaventaneet eroa manageroidun ja manageroimattoman koodin tehokkuuden välillä. Esimerkiksi Microsoftin uudet pelienkehitystyökalut (*engl.* XNA) ovat nostaneet manageroidun koodin käyttökelpoisuutta myös erittäin suurta reaaliaikaista suorituskykyä vaativien ohjelmistojen keskuudessa. (Microsoft 2008 b.)

.NET Framework voi ajaa sekä manageroitua että manageroimatonta koodia. Manageroimattomalla koodilla toteutetut komponentit lataavat CLR:n prosesseihinsa ja käynnistävät sen jälkeen manageroidun koodin ajon. (Microsoft 2008 a.)

CLR-komponentin tehtävänä on muistinhallinnan lisäksi säikeidenhallinta, koodin ajo, tarkistus ja kääntäminen sekä muita ”näkymättömiä” tehtäviä. Muistinhallinta hoitaa automaattisesti olioiden rakenteen ja niiden referenssit (pointtereita ei tarvita kuten käytettäessä C++ -kieltä), vapauttaen ne muistista sen jälkeen kun niitä ei enää tarvita. Hyvä muistinhallinta ratkaisee monet ohjelmointiongelmat, jotka ovat tavallisia käytettäessä manageroimatonta ohjelmistoympäristöä, kuten muistivuodot ja vialliset muistiviittaukset. (Microsoft 2008 a.)

Koska CLR hoitaa myös tietoturvaan ja -turvallisuuteen liittyviä prosesseja, on sen käsittelemällä koodilla eri turvatasoja riippuen niiden alkuperästä (Internet - Yritysten verkot - Intraverkot - paikallinen laite). Tästä tasosta riippuen tietyllä ohjelmakomponentilla voi olla estetty pääsy .NET-luokkiin, jotka mahdollistavat tiedostojen, rekistereiden tai muiden yksityisten tietojen käsittelyn. (Microsoft 2008 a.)

CLR sisältää CTS-infrastruktuurin (*engl.* Common Type System), joka määrittää miten tietotyypit määritellään, miten ne varaavat muistia, miten niitä käytetään ajonaikaisesti ja se on myös tärkeä osa CLR:n ajonaikasta ohjelmointikielitukea. CTS mahdollistaa oliomallisen suorituskykyisen ohjelmointiympäristön, joka sisältää useita eri ohjelmointikieliä. Se myös määrittää ne säännöt, joita eri ohjelmointikielien on noudatettava. Tämä varmistaa sen että eri ohjelmointikielillä toteutetut oliot voivat kommunikoida keskenään. (Microsoft 2008 a.)

Manageroitua koodia ei koskaan tulkata, vaan CLR:n JIT-tekniikka (*engl.* Just-In-Time compiling) mahdollistaa koodin ajamisen sillä kielellä, mitä ohjelman isäntälaite ymmärtää (Microsoft 2008 a).

CLR voi toimia yhdessä palvelinsovellusten (myös kolmannen osapuolen), kuten Microsoft SQL Server ja IIS-palvelujen (*engl.* Internet Information Services) kanssa. Tämän

infrastruktuurin ansiosta sovellusten bisneslogiikat voidaan laatia manageroidulla koodilla ja silti käyttää haluttuja palvelinratkaisuja. (Microsoft 2008 a.)

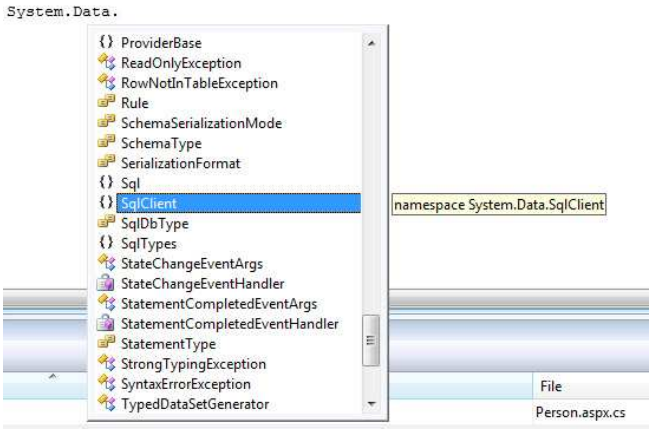
4.2 .NET Luokkakirjastot

.NET ohjelmointikehyksen luokkakirjastot sisältävät laajan kokoelman valmiita, oliopohjaisia tyyppiluokkia. .NET-luokkia on ohjelmointiavun ansiosta helppo käyttää ja niiden helppo ymmärrettävyys helpottaa uusien ominaisuuksien opettelemista. (Microsoft 2008 a.)

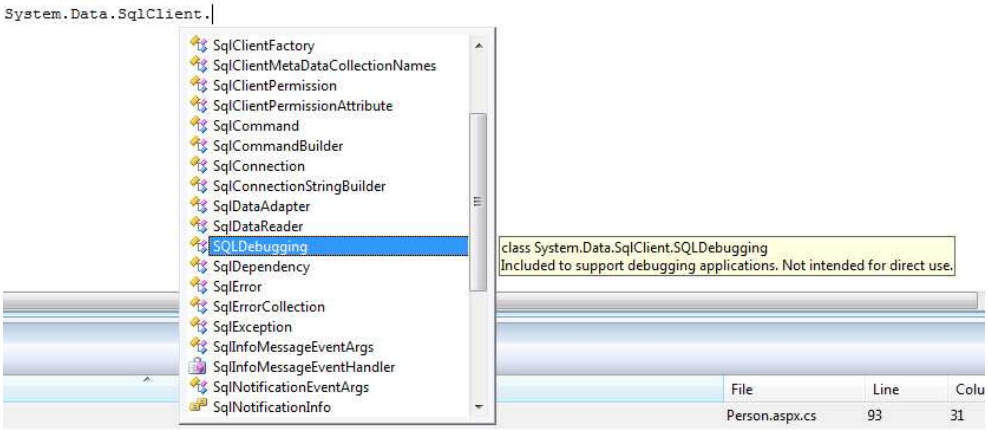
.NET 2.0 ja sen luokkakirjastot käsittivät noin 18 000 tyyppiä, 13 000 luokkaa, 400 000 julkista metodia, 93 000 julkista ominaisuutta ja noin 30 000 julkista tapahtumankäsittelijää. .NET-versioiden 3.0 ja 3.5 myötä ominaisuuksien määrä on noussut entisestään. Näillä luokkakirjastoilla voidaan esimerkiksi käsitellä tiedostoja (File-luokka), grafiikkaa, kuten kuvia ja vektoreita (Graphics-luokka) sekä generoida satunnaisnumeroita (Random-luokka) ja lähettää sähköposteja (SmtpClient-luokka). (Walther 2008, 12.)

Näin suuren luokkamäärän hallinta olisi ylivoimaista ilman nimiavaruuksia (*engl.* namespace). Nimiavaruudet sitovat koko .NET Frameworkin lukuisat ominaisuudet yhdeksi helpommin hahmotettavaksi kokonaisuudeksi, jonka sisältöä voidaan selata kuten käyttöjärjestelmien kansioita ja tiedostoja. Nimiavaruuden eri kategoriat erotetaan toisistaan pisteillä, jolloin saadaan aikaan kategoriapolku. Esimerkiksi kaikki tiedostojen käsittelyyn liittyvät luokat löytyvät **System.IO** -nimiavaruudesta. Nimiavaruudesta **System.Data.SqlClient** löytyvät kaikki SQL Serverin tietokantojen hallintaan kuuluvat luokat. (Walther 2008, 14.)

Nimiavaruuksia ei ohjelmointiavun ansiosta tarvitse muistaa ulkoa ja jos yhteenveto-lohko on määritetty, näyttää ohjelmointiapu ponnahdusikkunassa nimiavaruuden polun lisäksi myös valitun kohteen lyhyen selostuksen (kuvio 24. ja 25.) (Walther 2008, 14-15).



Kuvio 24. Ohjelmointiapu ja SqlClient-nimiavaruus



Kuvio 25. Ohjelmointiapu ja SQLDebugging-luokan lyhyt kuvaus

Selostuksen (*engl.* summary) voi liittää myös itse omiin luokkiin ja metodeihin, jolloin ohjelmointiapu näyttää ne kuvion 25. mukaisella tavalla. Selostus liitetään metodiin seuraavan esimerkin mukaisella syntaksilla. (Walther 2008, 15.)

```
/// <summary>
/// Palauttaa listan henkilö-olioita annetun PersonID:n perusteella
/// </summary>
public static List<CustomPersonItem> GetPerson(string sGuid)
{
    //metodin koodi tähän..
}
```

Jos selostus halutaan antaa koko luokalle yksittäisen metodin sijaan, on selostus kirjoitettava luokan juureen, ennen luokan varsinaista esittelyä. Edellisen esimerkin selostus näkyy ohjelmointiavussa kuvion 26. mukaisesti. (Walther 2008, 15.)

```
protected List<CustomPersonItem> GetSource()
{
    string sEmployerID = Request.QueryString["EmployerID"];

    if (!String.IsNullOrEmpty(sEmployerID))
    {
        return Persons.
    }

    return null;
}
```

Equals
GetPerson
ReferenceEquals

List<CustomPersonItem> Persons.GetPerson(string sGuid)
Palauttaa listan henkilö-olioita annetun PersonID:n perusteella

Kuvio 26. Ohjelmointiapu omien metodien ja niiden selosteiden kanssa

4.3 ASP.NET

.NET Framework sisältää suuren määrän valmista toiminnallisuutta, jonka tehtävänä on helpottaa ohjelmoijan työtä. Web-sivujen rakentamiseen tarkoitettu ASP.NET on vain osa .NET-arkkitehtuuria mutta se sisältää silti suuren määrän apuvälineitä, joiden määrä kasvaa versio versiolta. (MacDonald 2007, 4.)

ASP oli microsoftin ensimmäinen yritys parantaa perinteistä HTML-webkehitystä. Se tarkoitti käytännössä sitä, että web-sivustot koostuivat edelleen HTML -koodista, mutta sen joukkoon oli mahdollista sijoittaa ASP-skriptejä esimerkiksi tietokantojen käsittelyyn. Ero perinteisellä ASP:llä ja .NET ASP:llä on kuitenkin selvä. Vaikka ASP tarjosikin ratkaisuja joihinkin senaikaisen web-kehityksen ongelmiin, ASP myös loi niitä. Yksi näistä ongelmista oli niin sanottu ”spagettikoodi”, jonka nimi tuli <% -lohkoista, joilla skriptit erotettiin HTML-koodista. Seuraavassa ASP-esimerkissä alasvetovalikko (*engl.* dropdownlist) täytetään tietokannasta haetulla datalla. (MacDonald 2007, 4.)

```
<%
   Set dbConn = Server.CreateObject("ADODB.Connection")
   Set rs = Server.CreateObject("ADODB.Recordset")
   dbConn.Provider = "sqloledb"
   dbConn.Open "Server=SERVER_NAME; Database=Pubs; Trusted_Connection=yes"
%>

<select name="cboAuthors">
   <%
      rs.Open "SELECT * FROM Authors", dbConn, 3, 3
      Do While Not rs.EOF
   %>
   <option value="<%=rs("au_id")%>">
      <%=rs("au_lname") & ", " & rs("au_fname")%>
   </option>
   <%
      rs.MoveNext
      Loop
   %>
</select>
```

Heikon luettavuuden ja tietoturvan lisäksi tämän tekniikan suorituskyky on erittäin heikko, koska palvelimella oleva skriptimoottori joutuu käynnistymään useita kertoja pelkästään yhden palvelinpyynnön (*engl.* request) aikana. Lisäksi ASP-sivut käännetään aina, kun sivu ajetaan. Nämä nostavat koko sivun käsittelemiseen tarvittavaa aikaa. (MacDonald 2007, 5.)

ASP-tekniikoilla laaditut sivut voivat pituudeltaan kasvaa liian suuriksi. Jos sivulla käytetään lisäksi kolmannen osapuolen COM-komponentteja, sivun luettavuus heikentyy entisestään. Heikon luettavuuden lisäksi ASP:n virheentarkistus on hankalaa. Ammattimaisten virheentarkistustyökalujen toimivuus on ASP-ympäristössä erittäin heikko. (MacDonald 2007, 5.)

ASP:n skripteissä olevat muuttujat ovat heikosti tyypitettyjä. Ne vaativat vahvasti tyypitettyjä muuttujia enemmän muistia, niiden tila tarkistetaan vain ajon aikana ja niiden käyttö ei ole suorituskyvyn kannalta yhtä tehokasta kuin vahvasti tyypitettyjen. (MacDonald 2007, 6.)

Kehittyneemmissä ASP.NET -tekniikoissa bisnes- ja datankäsittelylogiikka tehdään olio -ohjelmoinnin kautta ja kontrollit toteutetaan samaan tapaan kuin työpöytäsovelluksissa. Tämän ansiosta HTML- ja logiikkakoodia ei tarvitse kirjoittaa samalle sivulle. (MacDonald 2007, 6.)

Kuten kaikki muutkin .NET-projektit, ASP.NET-projektit käännetään vain kerran ja kaytössä ovat kaikki ne ohjelmointikielet, jotka ovat yhteensopivia CLR-komponentin kanssa. ASP.NET-serverikontrollit renderoivat HTML-sisältönsä käytössä olevan Internet -selaimen ominaisuuksien mukaan, joka parantaa yhteensopivuutta eri selainten kanssa. (MacDonald 2007, 14.)

ASP.NET 1.0 tarjosi perustekniikat ja -kontrollit monipuolisempien web-sivujen laadintaan. ASP.NET 2.0 tehtiin versioiden 1.0 ja 1.1 päälle lisäämällä siihen monia uusia elementtejä, joista tärkeimmät on listattu seuraavaksi. (MacDonald 2007, 16.)

- Yli 40 uutta kontrollia kuten TreeView- ja JavaScript-pohjaiset Menu-kontrollit
- Ylisivut toistuvan koodin vähentämiseksi
- Teemat, jotka helpottavat tyylien keskittämistä ja koko sivuston ulkoasun muokkaamista
- Tietoturvaan ja käyttäjienhallintaan liittyviä tekniikoita ja kontrolleja
- Menetelmät, jotka mahdollistivat tietokantojen muokkauksen ilman tietokantojen omaa koodia

.NET Framework 3.0 julkaistiin Windows Vista -käyttöjärjestelmän mukana ja se sisälsi monia suuria uusia ohjelmointiympäristöjä kuten WPF-, WCF- ja WF-kokonaisuudet. NET Framework 3.0:n ja 3.5:n välissä julkaistiin myös uusi, Flashin kilpailijaksi asettunut web -teknologia SilverLight, joka tulevaisuudessa aiotaan liittää osaksi ASP.NET-komponenttia. (MacDonald 2007, 17.)

.NET Framework 3.5 on versioihin 2.0 ja 3.0 verrattuna enemmänkin päivityksenomainen, kuten versionumeroinnistakin voi päätellä. .NET Framework 3.5 ja ASP.NET 3.5 sisältävät kaksi uutta kokonaisuutta, LINQ- ja ASP.NET AJAX-tekniikat. (MacDonald 2007, 17.)

4.4 LINQ

LINQ on ohjelmointitekniikka joka on osa .NET Framework 3.5:tä. LINQ -tekniikoiden tehtävänä on yksinkertaistaa ja yhdistää tunnettuja datankäsittelymenetelmiä .NET-ympäristössä saman ohjelmointisyntaksin alle. Tähän mennessä kaikkien datavarastojen käsittely on vaatinut erilaisia työkaluja ohjelmointisyntakseineen. Datan käsittely voidaan tehdä suorituskyvyn ja tietoturvallisuuden kannalta hyvin tai hyvin huonosti, myös LINQ-ohjelmoinnissa. Jos siitä aikoo saada kaiken irti tehokkaasti ja turvallisesti, on se hallittava hyvin. (Pialorsi 2008, 1.)

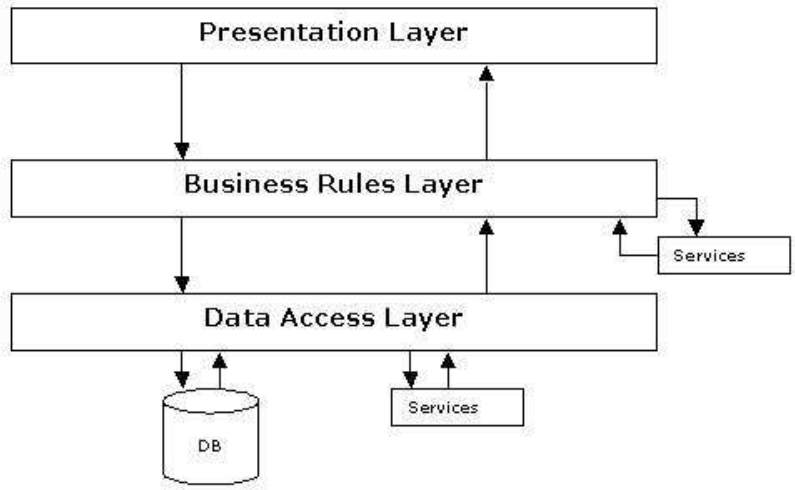
Datavarastoja, joita on hallittava eri tavoin, ovat esimerkiksi taulukot, oliot, XML -dokumentit, tietokannat, tekstitiedostot, rekisterit, sähköpostiviestit sekä Microsoft Office- ja Excel -tiedostot. LINQ mahdollistaa lähes täysin samanlaisen syntaksin näiden

tietolähteiden lukemiseen ja muokkaamiseen. Enää ei tarvitse käyttää SQL-datan käsittelyyn SQL-kyselyitä yhteysmerkkijonoineen (*engl.* connectionstring) ja parametreineen, eikä XMLReader-, Xpath- tai Xquery-toiminnallisuuksia XML-tiedostoihin. Enää ei ole pakko iteroida omia tietovarastoja ja laatia algoritmejä niiden lukemiseen ja järjestämiseen. Excel -tiedoston lukeminen on nyt helpompaa, koska ei tarvitse käyttää Excel-nimiavaruuksista löytyviä metodeja ja toimintoja. Sähköpostin ja Windows-käyttöjärjestelmien rekisterien luku- ja käsittelylogiikat voidaan nyt laatia samoilla LINQ-käskyillä. (Pialorsi 2008, 1; Rattz 2008, 301.)

LINQ-tekniikoiden avulla voidaan käsitellä sellaisia tietovarastoja, jotka tukevat IEnumerable- tai IQueryable -rajapintaa. Tuki löytyy suurimmasta osasta Windows-ypäristön tietovarastoja. (Walther 2008, 916.)

Yksi LINQ:n vahvuuksista on sen kyky parantaa tuottavuutta vähentämällä manuaalista ohjelmointityötä merkittävästi esimerkiksi toteutettaessa monikerrosmallin (*engl.* n-tier) mukaista ohjelmistoprojektia (Eichert 2008, 486).

Monikerrosmallin mukaisessa projektissa on esitystavasta riippuen kolme tai neljä kerrosta (kuvio 27.) *(Exforsys Inc 2006).*



Kuvio 27. Monikerrosmallin mukainen ohjelmistoprojektimalli (Exforsys Inc 2006.)

1) Esityskerros (*engl.* Presentation Layer)

   Esityskerroksessa haettu data käännetään esitettävään muotoon, oli kyseessä ASP -sivusto tai esimerkiksi konsolisovellus. Tässä kerroksessa ei suoriteta muuta logiikkaa tai algoritmejä kuin niitä, jotka liittyvät jo haetun tiedon esittämiseen. Tiedon hakemiseen, karsimiseen, muokkaamiseen ja muuhun käsittelyyn vaikuttava logiikka kirjoitetaan bisneslogiikkakerrokseen. (Eichert 2008, 487-488; Exforsys Inc 2006.)

2) Bisnesslogiikkakerros (*engl.* Business Logic Layer, BLL)

   Bisneslogiikkakerros käytetään niiden sääntöjen luomiseen, joilla data haetaan datakerroksen (*engl.* Data Access Layer, DAL) kautta. Tässä kerroksessa on myös suoritetaan haetun datan uudelleenkäsittely sekä sen mahdollinen muokkaus ja karsinta. Suorituskyvyn parantamiseksi on järkevää tehdä tietovarastoihin mahdollisimman tarkkoja hakuja, jolloin ylimääräisten loogisten prosessien ja hakujen määrä olisi mahdollisimman pieni. (Eichert 2008, 487-488; Exforsys Inc 2006.)

   Bisneslogiikkakerroksessa esitellään lisäksi oliot, eli niiden muuttujat, konstruktorit ja ominaisuudet. BLL-kerros sisältää myös tarvittavat metodit tietovarastojen datan tallentamiseen, päivittämiseen, lisäämiseen ja poistamiseen. (Eichert 2008, 488; Exforsys Inc 2006.)

3) Datakerros

   Tässä kerroksessa haetaan varsinainen data, BLL-kerroksesta tulleiden parametrien mukaisesti. Erillinen DAL-kerros lisää tietovarastojen tietoturvaan yhden lisäkerroksen, koska näin muiden kerrosten ei tarvitse ottaa kantaa tietovarastojen toiminnallisuuksiin, tai onko itse tietovarasto edes olemassa. (Eichert 2008, 488; Exforsys Inc 2006.)

4) Tietokanta tai muu tietovarasto

Kuviossa 27. olevat nuolet kuvaavat tiedon kulkua monikerrosmallin toteuttavassa projektissa. Kun esityskerros tekee tiedonhakukutsun bisneslogiikalle, se tekee kutsun DAL -kerrokselle, joka tekee kutsun tietovarastoon BLL-kerroksen esittämien ehtojen mukaisesti. Haun jälkeen DAL-kerros palauttaa datan BLL-kerrokselle ja sulkee avaamansa tietovarastoyhteydet. Seuraavaksi BLL-kerros muuttaa datan esityskerroksen ymmärtämiksi

ilmentymiksi ja palauttaa tiedon esityskerrokselle, joka suorittaa datanesitysrutiinit. (Eichert 2008, 488; Exforsys Inc 2006.)

Oikein laadittu monikerrosprojekti parantaa ohjelmiston tietoturvaa, koska ohjelmiston skaalautuvuus, muunneltavuus ja ylläpidettävyys paranee. Koska eri osa-alueiden ohjelmistokoodit sijaitsevat fyysisesti vähintään eri tiedostoissa (voivat sijaita myös eri laitteissa) ja ovat riippumattomia toisistaan, niitä voi käsitellä ja muokata omina kokonaisuuksinaan. Tärkeää on suunnitella kerrosten rajapinnat mahdollisimman hyvin. Jos esimerkiksi halutaan vaihtaa SQL-tietokanta Oracle-tietokantaan, vain DAL -kerroksen ohjelmakoodiin tulisi tarvita kirjoittaa muutoksia. (Eichert 2008, 489; Exforsys Inc 2006.)

Monikerrosmallin täyttävässä projektissa täytyy toteuttaa samat tietoturvamekanismit kuin muissakin projektimalleissa (Exforsys Inc 2006).

Monikerrosmallin toteuttaminen vaatii paljon työtä, taitoa sekä kokemusta ja sitä kautta sen toteuttaminen maksaa. Tästä huolimatta yritykset käyttävät sitä nykyään laajalti. (Eichert 2008, 489; Exforsys Inc 2006.)

LINQ helpottaa ja muuttaa monikerrosprojektin toteuttamista merkittävästi. Sen avulla on mahdollista jättää DAL-kerros kokonaan toteuttamatta, sillä tietokantoja käytettäessä voidaan datayhteydet alustaa LINQ to SQL -luokkien avulla, joiden sisällön Visual Studio 2008 luo automaattisesti. Nämä luokat sisältävät tarvittavat datayhteydet ja niiden käsittelyn. Tärkeää on myös tietää että kun LINQ to SQL -kutsu ajetaan, se generoi syntaksista automaattisesti SQL-kielisen kutsun, joka välitetään tietokannalle. Tämän vuoksi LINQ -kyselyt toimivat vain SQL Server -ohjelmiston kanssa. (Eichert 2008, 194.)

Uudet ASP.NET 3.5 kontrollit, datan sivutuskontrolli (*engl.* datapager) ja listanäkymä (*engl.* listview) on suunniteltu käytettäväksi erityisesti LINQ-tekniikoiden kanssa. Ennen uusien kontrollien käsittelyä, tässä dokumentissa tutustutaan LINQ-ohjelmointikieleen sekä ASP.NET 2.0 -kontrolleihin ja kuinka LINQ toimii niiden kanssa.

4.4.1 Var-tietotyyppi

Uusi var-muuttuja on heikosti tyypitetty, eli se voi olla mitä tyyppiä tahansa ja sen tyyppi määritetään ohjelman ajon aikana silloin, kun siihen ensimmäisen kerran sijoitetaan dataa. Var-tyyppisten muuttujien tyyppimääritys kannattaa toteuttaa jo muuttujan nimessä, jos tyyppi on tiedossa ennen sen käyttöä. (Walther 2008, 907.)

Seuraavassa esimerkissä on luotu kolme var-tyyppistä muuttujaa (int, string ja int -taulukko) (Eichert 2008, 26).

```
var intNumber = 12;
var strName = "Steve";
var intNumbers = new int[11,2,551];
```

Var-muuttujat sopivat hyvin olioiden luomiseen. Sen sijaan että kirjoitetaan vähintään yksi luokka tähän tarkoitukseen, voidaan olio luoda seuraavan esimerkin mukaisesti. (Ferracchiati 2008, 1.)

```
var objPerson = new {FirstName = "Steve", LastName = "Bishop", Age =
20};
```

Var-tyyppisillä muuttujat eivät voi olla metodien palautustyyppeinä tai luokkien muuttujina, eikä listoja tai taulukoita voi täyttää var-muuttujilla. Lisäksi var-tietotyyppiä ei voi alustaa null -arvoon. Paikallisesti var-tietotyypit kuitenkin oikein käytettynä vähentävät ohjelmointityön määrää. (Petricek Tomas 2008.)

4.4.2 LINQ-perusteet ja LINQ-oliotekniikat (*engl.* Linq to Objects, LtO)

Tämä kappale keskittyy LtO-tekniikoiden lisäksi LINQ-tekniikoiden perussyntakseihin. Kappaleessa 4.4.4 esitetään kuinka LINQ toimii SQL-tietokantojen kanssa ja kappaleessa 4.5 näytetään kuinka sitä käytetään erilaisiin ASP-kontrolleihin.

LINQ to Objects tarkoittaa LINQ-tekniikoiden käyttämistä paikallisesti luotuihin, muistissa oleviin tietovarastoihin. Näitä tietovarastoja ovat esimerkiksi taulukot, listat, kokoelmat ja oliot. Usein paikallisista tietovarastoista halutaan poimia tietoja jonkin niissä olevien ominaisuuksien eikä niinkään niiden järjestysnumeron perusteella. Tietovarastojen alkioita

joudutaan normaalisti käymään läpi yksi kerrallaan ja vertaamaan niiden sisältöä haluttuun arvoon. Esimerkiksi merkkijonotaulukoilla (*engl.* string array) ei ole sisäänrakennettua metodia, joka palauttaisi kaikki ne elementit jotka ovat tietyn pituisia. LtO-tekniikat antavat tähän uuden ratkaisun. (Ferracchiati 2008, 1.)

LINQ-kyselyt muistuttavat takaperin laadittua ACCESS- tai SQL-kyselyä, sillä myös LINQ -kysely sisältää from, where ja select-lausekkeet. From-lauseke määrittää datalähteen, where -lauseke datan karsinnan ja select-lauseke sen, mitä tietoa valitusta alkiojoukosta otetaan. Seuraavassa koodiesimerkissä käytetään LINQ-kyselyn kohteena string-taulukkoa. (Ferracchiati 2008, 1.)

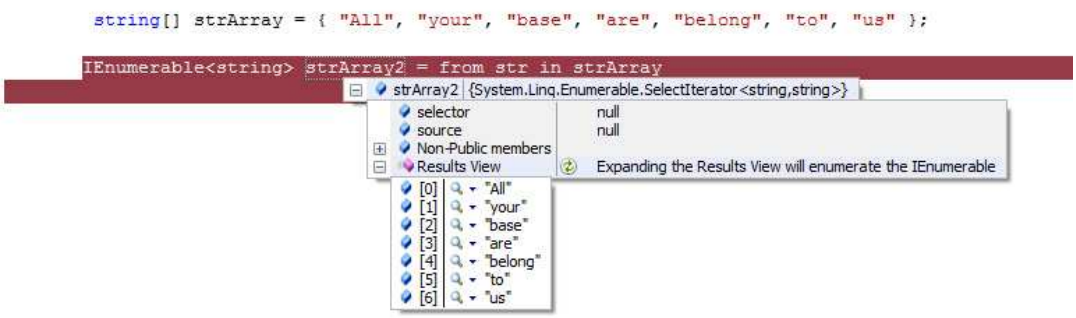
```
string[] strArray = { "All", "your", "base", "are", "belong", "to",
"us" };
```

Seuraavassa esimerkissä taulukkoon kohdistetaan LINQ-kysely, joka palauttaa kaikki edellisen taulukon merkkijonoilmentymät. (Ferracchiati 2008, 1.)

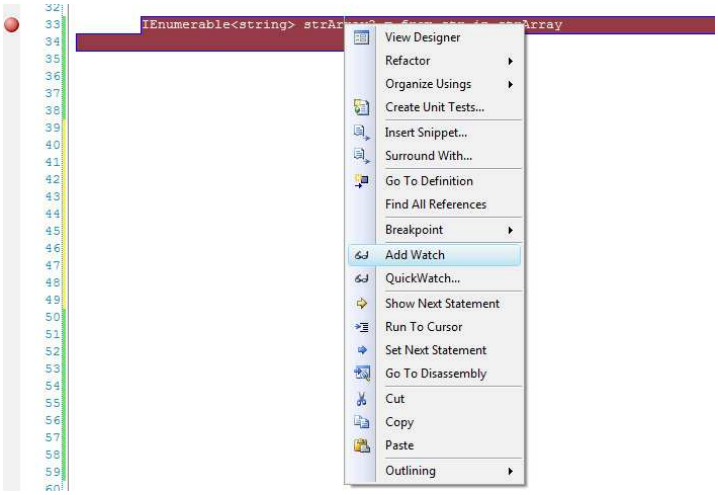
```
IEnumerable<string> strArrayFromLINQ = from str in strArray
                                       select str;
```

Edellisen esimerkin from-lauseessa esitellään itse tietovarasto, jota iteroidaan (strArray) ja itse nimetty muuttuja (str), joka esittää yhtä alkiota tietovarastosta. Select-lausekkeessa määritetään ne str-muuttujan ominaisuudet, joita hakutulokseen halutaan mukaan. Koska tässä esimerkissä haun kohteena on yksinkertainen merkkijonotaulukko, eikä esimerkiksi lista olioita, ei select-lausekkeeseen tule muuta. Myöhemmissä esimerkeissä esitellään, miten LINQ-tekniikoiden valintalauseketta voidaan tehokkaasti käyttää monimutkaisempia tietovarastoja käsiteltäessä. (Ferracchiati 2008, 1.)

LINQ-lauseita käytettäessä on usein tarpeellista tietää, miten kyselyt toimivat ja mitä ne palauttavat. Virheentarkistustilassa on mahdollista tarkastella tuloksia pitämällä hiiren kursoria muuttujan päällä, johon kyselyn tulos on talletettu (kuvio 28.). Vaihtoehtoisesti muuttuja voidaan lisätä virheentarkistuksen aikana seurantalistaan painamalla muuttujaa oikealla hiiren painikkeella ja valitsemalla kohta *add watch* (kuvio 29.). Tällöin muuttuja siirtyy koodialueen alapuolella sijaitsevaan seurantalistaan, jolloin sen sisältöä on helpompi seurata. (Ferracchiati 2008, 1; Powers 2008, 352.)



Kuvio 28. LINQ-kyselyn tuloksen tarkastelu virheentarkistustilassa



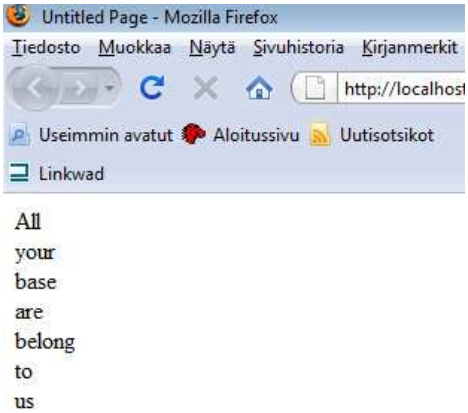
Kuvio 29. Virheentarkistus ja seuranta-toiminnon lisääminen

Konsolisovelluksissa voi kyselyiden tarkasteluun käyttää Microsoftin ObjectDumper -luokkaa, jonka voi imuroida osoitteesta http://code.msdn.microsoft.com/csharpsamples (ObjectDumper-luokka tulee CSharpSamples.zip-paketin mukana) (Ferracchiati 2008, 2).

Kolmantena vaihtoehtona on seuraavan esimerkin tapaan iteroida koko kyselyn tulos esimerkiksi foreach-lausekkeessa ja siirtää tulokset väliaikaisesti sivulle Response.Write -metodia käyttäen (Cox 2008, 103).

```
string[] strArray = { "All", "your", "base", "are", "belong", "to",
"us" };

IEnumerable<string> strArrayFromLINQ = from str in strArray
                                       select str;

foreach (string str in strArrayFromLINQ)
{
    Response.Write(str + "<br />");
}
```

Tämän ohjelman ajo tulostaa Internet-selaimeen kaikki `strArrayFromLINQ`-taulukon alkiot tekstinä omille riveilleen (kuvio 30.). Koska Response.Write-metodin sisältö ajetaan selaimessa HTML-koodina, HTML -rivinvaihtotagi (`<br />`)tekee rivinvaihdon. (Cox 2008, 103.)



Kuvio 30. Response.Write-käskyllä Internet-selaimeen tulostetut alkiot

Lähes yhtä vaivaton tapa tulosten tarkasteluun on lisätä aspx-sivulle esimerkiksi taulukkonäkymä (*engl.* gridview) ja sijoittaa kyselyn tulos siihen. Kyselyn liittäminen taulukkonäkymään tapahtuu kontrollin DataSource- ja DataBind-metodeja käyttäen. Taulukkonäkymä luo tulosten otsikot automaattisesti, jolloin tulosten luettavuus on hyvä. Taulukkonäkymän ja muiden ASP-kontrollien toiminnoista ja käytöstä enemmän kappaleessa 4.5. (MacDonald 2007, 532.)

Edellisen esimerkin LINQ-kyselyn tulos sijoitetaan IEnumerable-tyyppiseen muuttujaan. LINQ-kysely palauttaa aina IEnumerable-tyyppisen kokoelman alkioita, vaikka haun tuloksena palautuisi vain yksi alkio. Tähän palautustyyppiin voidaan vaikuttaa erilaisilla valmiilla metodeilla, joita LINQ pitää sisällään. (MacDonald 2007, 532.)

Seuraavassa esimerkissä käytetään LINQ-tekniikoiden ToArray-metodia, joka muuttaa kyselyn tuloksen taulukoksi. Seuraavassa esimerkissä koko LINQ-kysely sijoitetaan sulkuviivojen sisälle, jolloin ToArray-metodia voidaan käyttää. (MacDonald 2007, 553.)

```
string[] strArrayFromLINQ = (from str in strArray
                             select str).ToArray();
```

Edellisen esimerkin LINQ-kyselyn tulos voidaan sijoittaa myös Array-tyyppiseen taulukkoon. Jos kyselyn tulos halutaan sijoittaa string-muuttujilla täytettyyn listaan (*engl.* list),

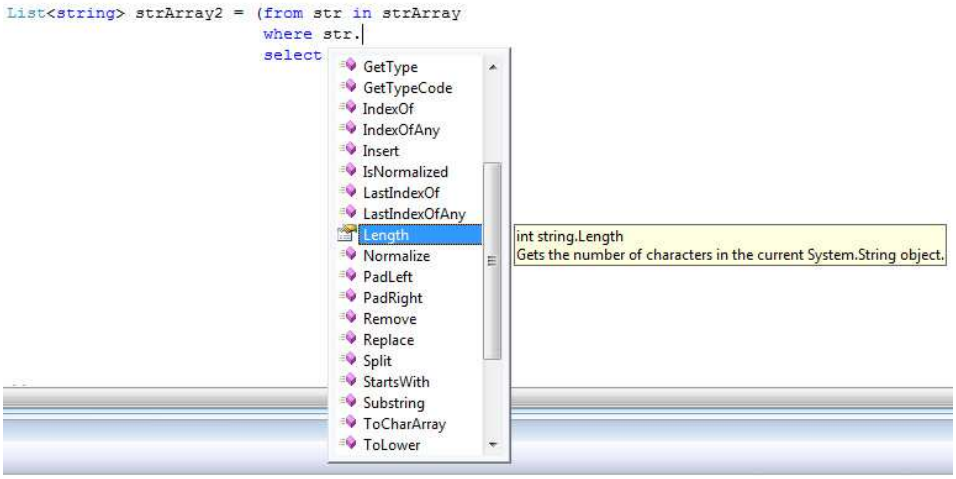
tulee kyselyyn käyttää ToList-metodia. Listoja käytettäessä tulee luokan alkuun lisätä seuraavan esimerkin nimiavaruusviittaus. (MacDonald 2007, 531-532.)

```
using System.Collections.Generic;
```

Edellisissä LINQ-esimerkeissä ei ole käytetty lainkaan tuloksia rajoittavaa where-ehtoa, joten kysely palautaa tietovarastosta kaikki alkiot. Seuraavassa esimerkissä samasta string -taulukosta haetaan ne merkkijonoalkiot, joiden pituus on yli viisi. (MacDonald 2007, 532.)

```
List<string> strArrayFromLINQ = (from str in strArray
                                 where str.Length > 5
                                 select str).ToList();
```

Koska edelliset esimerkit ovat käyttäneet tietolähteenään string-taulukkoa, kääntäjä osaa str -iteraattorimuuttujaa käsiteltäessä tarjota ohjelmointiavussa automaattisesti niitä metodeja, joita string-tyyppisellä muuttujalla on käytettävissään (kuvio 31.). (MacDonald 2007, 532.)



Kuvio 31. Ohjelmointiavussa näkyvä yhteys iteraattorimuuttujan ja tietolähteen välillä

Seuraava esimerkki palauttaa samasta tietolähteestä kaikki ne merkkijonot, jotka alkavat A- tai a-kirjaimella. (Walther 2008, 1095.)

```
List<string> lstFromLINQ = (from str in strArray Where
str.StartsWith("A",StringComparison.CurrentCultureIgnoreCase)
                            select str).ToList();
```

Jos ei haluta käyttää `StringComparison.CurrentCultureIgnoreCase`-määritettä, joka jättää huomioimatta kirjasimen koon, voidaan ehto täyttää TAI-operaattorilla, kuten seuraavassa esimerkissä (MacDonald 2007, 537).

```
List<string> strArray2 = (from str in strArray
                          where
                          str.StartsWith("A") ||
                          str.StartsWith("a")
                          select str).ToList();
```

Vastaavasti kyselyissä voi käyttää JA-operaattoria. Seuraavassa esimerkissä haetaan merkkijonot, jotka sisältävät kirjaimet 'a' JA 'e' TAI ne merkkijonot, joiden pituus on yli 5. (MacDonald 2007, 537-538.)

```
List<string> strArray3 = (from str in strArray
                          where
                          str.Contains('a') &&
                          str.Contains('e') ||
                          str.Length > 5
                          select str).ToList();
```

From-, where- ja select-lausekkeiden lisäksi LINQ-kyselyssä voi käyttää group by -, orderby-, join- ja let-lausekkeita (Walther 2008, 916).

Group by -lauseke mahdollistaa hakutulosten ryhmittelyn jonkin kokoelman esiintymän ominaisuuden mukaan. Seuraavassa esimerkissä hakutulokset on ryhmitelty henkilötaulun (*engl.* person) etunimi-ominaisuuden (*engl.* firstname) mukaan. Ryhmitystulokset on talletettu newPerson-väliaikaismuuttujaan, jota käytetään select-lausekkeessa. (Walther 2008, 916.)

```
var lstPersonFiltered = from per in lstPerson
                        group per by per.FirstName into newPerson
                        where per.Age >= 30
                        select newPerson;
```

Ryhmittely muuttaa kyselyn tuloksen IEnumerable-tyyppiseksi kokoelmaksi, joka sisältää tässä esimerkissä `System.Linq.IGrouping<string, Person>` -tyyppisiä olioita. Siksi kyselyn tulosta ei voi tallettaa listaksi vaan se on talletettava joko var-muuttujaan tai listaksi IGrouping-olioita, kuten seuraavassa esimerkissä. (MacDonald 2007, 538-539.)

```
List<IGrouping<string, Person>> lstPersonGrouped =

            (from p in lstPerson
            where p.Age >= 30
            group p by p.FirstName into g
            select g).ToList();
```

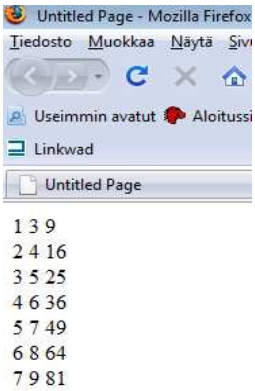
Hakutulokset voidaan järjestää laskevaan (*engl.* descending) tai nousevaan (*engl.* ascending) järjestykseen. Olioita käytettäessä tulee orderby-lausekkeessa ilmoittaa olion ominaisuus, jonka mukaan tulokset järjestetään. Seuraavassa esimerkissä tulokset on järjestetty henkilö -taulun ikä-ominaisuuden (*engl.* age) mukaan. (Walther 2008, 923.)

```
List<Person> lstPersonOrdered = (from per in lstPerson
                                 where per.Age >= 30
                                 orderby per.FirstName descending
                                 select per).ToList();
```

Let-lausekkeet mahdollistavat väliaikaisten, tyypittömien muuttujien luomisen LINQ -lauseiden sisällä. Seuraavassa esimerkissä muuttujiin 'j' ja 'k' on talletettu int-tyyppisiä arvoja. (Eichert 2008, 179.)

```
var lstNumbers = new List<int>() { 1, 2, 3, 4, 5, 6, 7 };

var lstNew = from i in list
             let j = i + 2
             let k = j * j
             select new { i, j, k };

foreach(var v in lstNew)
{
    Response.Write(string.Format("{0} {1} {2} <br />", v.i, v.j, v.k));
}
```

Edellisen esimerkin Response.Write-metodi tulostaa Internet-selaimeen kuvion 32. mukaisen syötteen (Walther 2008, 936-937).



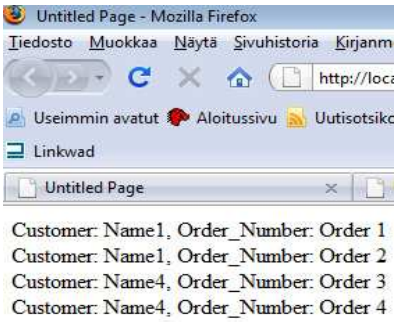
Kuvio 32. Muuttujalausekkeiden tulokset FireFox-selaimessa

Edellisestä esimerkistä on syytä poimia kohta `select new { i, j, k }`. Sen sijaan, että kysely palauttaisi jonkin tietyn ilmentymän, esimerkiksi i-muuttujan (`select i`), on palautteeksi luotu uusi olio (`select new { i, j, k }`). Olion ominaisuuksia käytetään myöhemmin Response.Write -käskyssä. Kun kyselyn tulos talletetaan var-muuttujaan, oliolle ei tarvitse kirjoittaa omaa luokkaa. (Eichert 2008, 179-180.)

Liitoslauseke mahdollistaa kahden eri datalähteen yhdistämisen yhteisen avaimen avulla. Seuraavassa esimerkissä käytetään olioiden luomiseen samaa menetelmää kuin edellisessä esimerkissä. (Walther 2008, 936-937.)

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        var customers = new List<Customer>() {
                            new Customer {Key = 1, Name = "Name1" },
                            new Customer {Key = 2, Name = "Name2" },
                            new Customer {Key = 3, Name = "Name3" },
                            new Customer {Key = 4, Name = "Name4" },
                            };

        var orders = new List<Order>() {
                        new Order {Key = 1, OrderNumber = "Order 1" },
                        new Order {Key = 1, OrderNumber = "Order 2" },
                        new Order {Key = 4, OrderNumber = "Order 3" },
                        new Order {Key = 4, OrderNumber = "Order 4" },
                        };

        var query = from c in customers
                    join o in orders on c.Key equals o.Key
                    select new {c.Name, o.OrderNumber};

        foreach (var order in query)
        {
           Response.Write(string.Format("Customer: {0} Order_Nro: {1}
           <br />", order.FirstName, order.OrderNumber));
        }
    }
}

public class Customer
{
    public int Key { get;set; }
    public string Name { get;set; }
}

public class Order
{
    public int Key { get;set; }
    public string OrderNumber { get;set; }
}
```

Edellisen esimerkin tuloksena Internet-selaimeen tulostuu kuvion 33. mukaiset tulokset (Walther 2008, 937).



Kuvio 33. Join-kyselyn tulokset

Kuten edellisestä liitosesimerkistä on nähtävissä, olioiden käsittely LINQ-tekniikoilla tapahtuu samoin tavoin kuin tämän kappaleen muissa esimerkeissä. Ymmärrettävin tapa on tehdä olioille oma luokka, joka sisältää vain get- ja set-metodit. Seuraavassa esimerkissä on luotu henkilö-luokka, jolla on ominaisuudet etunimi, sukunimi (*engl.* lastname), puhelinnumero (*engl.* phone) ja ikä (*engl.* age). (MacDonald 2007, 536-537.)

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Phone { get; set; }
    public int Age { get; set; }
}
```

Seuraavassa esimerkissä on alustettu henkilö-lista, johon lisätään kolme henkilö-oliota. Tämän jälkeen listaan kohdistetaan LINQ-kysely, jossa haetaan ”man”-sukunimen omaavia olioita. Lopuksi haun tuloksena olevien olioiden kaikki ominaisuudet kirjoitetaan `Response.Write`- ja `string.Format`-metodia apuna käyttäen Internet-selaimeen. Kyselystä palautuu yksi olio. (MacDonald 2007, 537.)

```
List<Person> lstPerson = new List<Person>();

lstPerson.Add(new Person { FirstName = "Donald", LastName = "Duck",
              Phone = "+358023654789", Age=22 });
lstPerson.Add(new Person { FirstName = "Super", LastName = "Man",
              Phone = "1946783", Age=30 });
lstPerson.Add(new Person { FirstName = "Rölli", LastName = "Peikko",
              Phone = "9764312", Age=50 });

List<Person> lstPersonFiltered = (from person in lstPerson
                                  where
                                   person.LastName.ToLower() == "man"
                                 select person).ToList();
```

```
foreach (Person p in lstPersonFiltered)
{
    Response.Write(string.Format("{0} {1} {2} {3} <br />", p.FirstName,
                    p.LastName, p.Phone, p.Age));
}
```

Jos tätä esimerkkiä kirjoitetaan Visual Studio 2008 ympäristössä, Visual Studio -ohjelmointiapu näyttää mitä ominaisuuksia oliolla on käytettävissä. Ominaisuuslista pienenee ominaisuuksien käytön mukaan (kuvio 34.). (MacDonald 2007, 537.)

```
List<Person> lstPerson = new List<Person>();

lstPerson.Add(new Person { FirstName = "Donald", LastName = "Duck", Phone = "+358023654789", Age=22 });
lstPerson.Add(new Person { FirstName = "Super", LastName = "Man", Phone = "1946783", Age=30 });
lstPerson.Add(new Person { FirstName = "Rölli", | });
```

Age
LastName
Phone
string Person.LastName

Kuvio 34. Olion luominen ja ohjelmointiapu

Tässä kappaleessa LINQ-kyselyihin kohdistettujen **ToArray**- ja **ToList**-metodien lisäksi kyselyissä voi käyttää esimerkiksi seuraavia metodeja (Walther 2008, 917).

- **Average**( ) - Palauttaa kyselyssä käytettyjen numeeristen arvojen keskiarvon.
- **Max**( ) - Palauttaa suurimman **numeerisen** arvon.
- **Min**( ) - Palauttaa pienimmän **numeerisen** arvon.
- **Count**( ) - Palauttaa kyselyssä palautuvien ilmentymien kokonaismäärän.
- **Select**( ) - Palauttaa hausta halutut ominaisuudet tai ilmentymät.
- **Where**( ) - Toimii suodattimena kuten edellisissä LINQ-esimerkeissä.
- **First**( ) - Palauttaa kyselyn ensimmäisen ilmentymän.
- **FirstOrDefault**( ) - Palauttaa kyselyn ensimmäisen ilmentymän, tai jos kysely ei löydä tuloksia, palautteena on null, eikä ohjelma kaadu.
- **Skip**( ) - Mahdollistaa kyselyn ilmentymien ohittamisen, jolloin palautteena on näiden ohitettujen ilmentymien jälkeiset ilmentymät.
- **Take**( ) - Tekee mahdolliseksi ottaa haluttu määrä kyselystä palautuneita ilmentymiä, jolloin loput ilmentymät jäävät pois.

Edellisen listan metodeista osa vaatii parametrejä (skip, take) tai lambda-lausekkeita (select, where). Muut listan metodit toimivat ilman parametreja tai ne eivät ole pakollisia. Lambda -lausekkeet eivät ole käytössä pelkästään LINQ-ympäristössä. .NET Framework 2.0 esitteli tyypittömät metodit, jossa delegaattien avulla voitiin luoda entistä vähemmän koodirivejä vaativia, anonyymejä toiminnallisuuksia. Lambda-lausekkeet vievät tätä ajattelutapaa vielä pidemmälle ja sen tavoitteena on tarjota minimalistinen lähestymistapa ohjelmointiin. (Walther 2008, 910.)

Seuraavassa esimerkissä aspx-sivulle on tehty painike- ja tekstikenttä-kontrollit, joiden nimet ovat btnButton ja txtText. Seuraavissa esimerkeissä painikkeelle tehdään toiminto, jonka ansiosta painikkeen painallus kirjoittaa tekstikenttään sen hetken päiväyksen ja kellonajan. Perinteisti painikkeeseen asetetaan tapahtuma (*engl.* event) ohjelmallisesti seuraavasti. (Walther 2008, 910-911.)

```
protected void Page_Load(object sender, EventArgs e)
{
    btnButton.Click += new EventHandler(btnButton_Click);
}

void btnButton_Click(object sender, EventArgs e)
{
    lblText.Text = DateTime.Now.ToString();
}
```

Visual Studio tekee käyttäjän halutessa tapahtumien rungot osittain automaattisesti. Kun edellisen esimerkin tilanteessa koodialueelle syötetään tapahtuman alku (`btnButton.Click +=`), voidaan painaa tabulaattoria, jolloin Visual Studio luo tapahtumalle tapahtumankäsittelijän (`new EventHandler(btnButton_Click)`). Kun tabulaattori-painiketta painetaan toisen kerran, generoidaan itse tapahtuma (seuraava esimerkki), josta virheen nostava `throw`-lause voidaan pyyhkiä pois ja korvata se halutuilla toiminnallisuuksilla. (MacDonald 2007, 264.)

```
void btnButton_Click(object sender, EventArgs e)
{
    throw new NotImplementedException();
}
```

Edellinen esimerkki voidaan toteuttaa anonyymisti, jolloin lisämetodeja ei tarvita. Koska seuraavassa esimerkissä käytetään delegaattia Page_Load-lohkon sisällä (jolla on samat

parametrityypit kuin delegaatilla), tulee delegaatin parametrit nimetä uniikeiksi (sender2, e2). (Walther 2008, 910-911.)

```
protected void Page_Load(object sender, EventArgs e)
{
    btnButton.Click += delegate(object sender2, EventArgs e2)
    {
        lblText.Text = DateTime.Now.ToString();
    };
}
```

Lambda-lauseena koodin määrä lyhenee entisestään, kuten seuraavasta esimerkistä nähdään (Walther 2008, 911).

```
protected void Page_Load(object sender, EventArgs e)
{
    btnButton.Click += (sender2, e2) => lblText.Text =
    DateTime.Now.ToString();
}
```

Lambda-lausekkeiden voidaan katsoa olevan mahdollisimman minimaalinen versio halutusta ohjelmallisesta toiminnallisuudesta. Seuraavassa esimerkissä on perinteinen LINQ-kysely, jossa yksi where-ehto. (Powers 2008, 90.)

```
List<Person> p = (from p in lstPerson
                  where p.Age >= 18
                  select p).ToList();
```

Seuraava esimerkki näyttää, kuinka vastaava kysely toteutetaan lambda-periaatteilla. (Powers 2008, 90.)

```
List<Person> p = (lstPerson.Select(em => em.BirthDate <= 18)).ToList();
```

Lambda-lausekkeita voi vapaasti yhdistellä LINQ-lausekkeiden kanssa. Seuraavassa esimerkissä on haettu kaikki henkilöt, joiden ikä on yli 18. Tästä joukosta kaksi ensimmäistä ilmentymää hypätään yli ja otetaan mukaan viisi seuraavaa, loput hylätään. (Powers 2008, 90.)

```
var q2 = (from p in lstPerson
          select p).SkipWhile(p => p.Age < 18).Skip(2).Take(5);
```

4.4.3 LINQ ja XML (engl. Linq to XML, LtX)

XML-tyyppinen data on merkittävä osa ohjelmistojen tietovarastoja. Se mahdollistaa esimerkiksi yleiset rajapinnat eri valmistajien Internet-sovellusten ja -sivustojen välillä tai XML-tyyppisten konfiguraatiodatan käsittelyn. XML-datan luettavuus on parempi kuin tekstimuotoisen datan. (Eichert 2008, 313.)

LINQ tarjoaa kevyen XML-ohjelmointirajapinnan, jolla voi tehdä LINQ-kyselyjä suoraan XML-dataan, samalla syntaksilla mitä käytetään olioiden käsittelyssä. Toimintamalli vaatii muutakin kuin LINQ-kyselyjen käyttöä, sillä kuten vanhempi DOM-malli, LtX sisältää omat metodinsa (load, parse, create, update, delete ja save), konstruktorit ja muuttujat. Silti lopputuloksena on DOM-mallia selkeämpi ja yksinkertaisempi toteutustapa. Tässä kappaleessa LINQ to XML -mallien rinnalla esitetään XML-datan käsittelyyn myös perinteisiä DOM-pohjaisia tyylejä. (Eichert 2008, 314.)

Koska XML-tyyppinen data on käytännössä vain hyvin järjesteltyä tekstiä, on sen käsittely monipuolisillakin työkaluilla vaikeampaa kuin tyypitettyjen olioiden. Ohjelmoijan on tiedettävä käsiteltävän XML-datan rakenne, sen nimikentät ja sisältö. (Eichert 2008, 315.)

Seuraavassa esimerkissä on esitelty tavoiteltava XML-data, jonka jälkeen näytetään kuinka data muodostetaan DOM-tekniikoilla (Eichert 2008, 317-318).

```
<games>
  <game>
    <title>Doom</title>
    <director>John Carmack</director>
    <coder>John Romero</coder>
    <coder>Sandy Petersen</coder>
  </game>
</games>
```

```
XmlDocument doc = new XmlDocument();
XmlElement games = doc.CreateElement("games");
XmlElement game = doc.CreateElement("game");
XmlElement director = doc.CreateElement("director");
director.InnerText = "John Carmack";
XmlElement coder1 = doc.CreateElement("coder1");
coder1.InnerText = "John Romero";
XmlElement coder2 = doc.CreateElement("coder2");
coder2.InnerText = "Sandy Petersen";
XmlElement graphics = doc.CreateElement("graphics");
graphics.InnerText = "Adrian Carmack";

game.AppendChild(director);
game.AppendChild(coder1);
```

```
game.AppendChild(coder2);
game.AppendChild(graphics);
games.AppendChild(game);
doc.AppendChild(games);
```

Seuraava esimerkki näyttää, kuinka edellisen esimerkin toiminnallisuus toteutetaan LINQ to XML -tekniikoilla. (Eichert 2008, 317.)

```
XDocument x = new XDocument
(new XElement("games",
     new XElement("game",
         new XElement("title", "Doom"),
             new XElement("director", "John Carmack"),
             new XElement("coder", "John Romero"),
             new XElement("coder", "Sandy Petersen"),
             new XElement("graphics", "Adrian Carmack"))));
```

Edellisistä esimerkeistä nähdään, että LtX-menetelmät tarvitsevat XML-datan käsittelyyn XElement- ja XDocument -tietotyyppien konstruktoreita, kun DOM-menetelmät tarvitsevat lisäksi useita metodeja. Esimerkistä nähdään myös, että LtX-ohjelmakoodi on rakenteeltaan samanmuotoinen kuin itse XML-data. (Eichert 2008, 319.)

Seuraavassa esimerkissä on LINQ-kysely, joka kodistetaan liitteen 2 mukaiseen XML -tiedostoon. Kyselyssä haetaan kaikki ne game-elementit, joiden tunniste (*engl.* id) on 1. (Ferracchiati 2008, 152.)

```
//ladataan xml-tiedoston data XDocument-tyyppiseen muuttujaan
XDocument xdoc = XDocument.Load(@"c:\games.xml");

//kysely, jossa haetaan elementit id:n perusteella
var query = from xml in xdoc.Elements("games").Elements("game")
            where (int)xml.Element("id") == 1
            select xml;

//kirjoitetaan haettu data selaimeen
foreach (var game in query)
{
    Response.Write(string.Format("Name: {0}, Price:{1} <br />",
                   game.Element("name"), game.Element("price")));
}
```

Edellinen esimerkki toteutetaan Xpath-toiminnallisuuksia käyttäen seuraavan esimerkin mukaisesti (tarvitaan `System.Xml.Xpath` -nimiavaruus). (MacDonald 2007, 605-606.)

```
//luetaan tiedosto ja luodaan navigaattori
XPathDocument doc = new XPathDocument(@"c:\games.xml");
XPathNavigator nav = doc.CreateNavigator();

//luodaan expression-muuttujat, jotka kuvaavat xml-polkuja
XPathExpression expr = nav.Compile("/games/game");
XPathExpression expr2 = nav.Compile("name");
XPathExpression expr3 = nav.Compile("price");

//luodaan Xpath-iteraattori, jota tarvitaan kun tiedostoa käsitellään
//silmukassa
XPathNodeIterator iterator = nav.Select(expr);

//tulostetaan data selaimeen
while (iterator.MoveNext())
{
    //iteraattorin kierroksen data
    XPathNavigator navCurrent = iterator.Current.Clone();

    //SelectSingleNode palauttaa node-listasta halutun noden
    Response.Write(string.Format("Name: {0}, Price: {1} <br />",
                   navCurrent.SelectSingleNode(expr2),
                   navCurrent.SelectSingleNode(expr3)));
}
```

Edellisistä XML-esimerkeistä voidaan nähdä LtX-menetelmien olevan selkeämpi, ainakin tarvittavien muuttujatyyppien määrän perusteella. Seuraavassa esimerkissä toteutetaan hiukan monimutkaisempi LtX-kysely. Kyselyssä haetaan liitteen 2 mukaisesta XML -datasta peli-elementit (*engl.* game) ja luodaan niistä uusi luokkatyyppi, jolla on ominaisuuksina nimi, hinta ja kokonaissumma siitä, miten peliä on eri vuosina myyty. (Ferracchiati 2008, 154)

Esimerkissä käytetään kahta sisäkkäistä LINQ-kyselyä, koska kokonaismyyntimäärä sijaitsee myynti-lohkossa (*engl.* sales) ja peli peli-lohkossa. Tilanne on sama kuin kahden erillisen

tietovaraston käsittelyssä. Kyselyssä ei ole toteutettu virheentarkistusta, joten jos parse -metodit saavat parametreinä null-arvoja (myynti-elementtejä ei löydy), ohjelma kaatuu. Esimerkin sisäkkäinen kysely voi suurilla datamäärillä olla rasite laitteiston suorituskyvylle. LINQ:n suorituskyvystä tarkemmin kappaleessa 4.4.5. (Ferracchiati 2008, 154.)

```
XDocument xdoc = XDocument.Load(@"c:\games.xml");

var query =

from game in xdoc.Descendants("game")
select new
{
    name = game.Element("name").Value,
    price = game.Element("price").Value,

    // haetaan game-elementin ID-arvoa vastaavat sales-elementit ja
    // palautetaan sales-elementtien yhteismyynti-arvojen (engl. sold)
    // summa (kaksi myyntiä)
    piecessold = (from sales in xdoc.Descendants("idgame")
                  where
                      (int)sales.Attribute("id") ==
                      (int)game.Element("id")
                  select
                  int.Parse(sales.Attribute("sold").Value)).Sum()
};

//tulostetaan kyselystä palautunut data selaimeen
foreach (var game in query)
{
    Response.Write(string.Format("name: {0}, price: {1}, pieces sold:
    {2}, sold overall: {3} € <br />", game.name, game.price,
    game.piecessold, game.piecessold *
    double.Parse(game.price.ToString())));
}
```

Load-metodin lisäksi XDocument sisältää metodit XML-dokumentin tallettamiselle (save) ja poistamiselle (delete). XElement sisältää esimerkiksi metodit elementtien sekä parametrien lisäämiselle ja päivittämiselle (SetElementValue, SetAttributeValue) sekä string -merkkijonon muuttamiseen XElement- tai XDocument-muotoon (parse). (Eichert 2008, 314.)

XDocumentin tallennus- ja poistometodit tarvitsevat parametriksi tiedoston polun (esimerkiksi `@"c:\test.xml"`), mutta XML-datan lohkojen sisällön muuttamiseen tarvitaan enemmän logiikkaa. Seuraavassa esimerkissä muutetaan tietyn pelin hinta-elementtiä (*engl.* price) ja varastotilanne-ominaisuutta (*engl.* instock). (Ferracchiati 2008, 174.)

```
XDocument xdoc = XDocument.Load(@"c:\games.xml");

// haetaan se game-elementti, jonka id=1 (doom-peli)
// SingleOrDefault -metodi muuttaa palautteen Xelement-tyyppiseksi
// Ienumerable-tyypin sijaan, jolloin sen voi tallettaa XElement-
// muuttujaan
XElement xe = (from xml in xdoc.Elements("games").Elements("game")
               where (int)xml.Element("id") == 1
               select xml).SingleOrDefault();

//muutetaan doom-pelin hinta-lohkon sisältöä
xe.SetElementValue("price", "10,00");

//muutetaan pelin ominaisuutta
xe.SetAttributeValue("instock", "23");
```

Edellinen koodiesimerkki ei ajettaessa vielä talleta muutoksia muuten kuin paikallisesti muistissa oleviin muuttujiin. Jos muutoksista haluaa pysyviä, tulee koko dokumentti tallettaa save-metodia käyttäen. (Ferracchiati 2008, 166.)

Edellisen esimerkin koodi muuttaa doom-nimisen pelin XML-datan seuraavan esimerkin kaltaiseksi (vertaa liite 2). (Ferracchiati 2008, 174.)

```
<game instock="23">
  <id>1</id>
  <name>Doom</name>
  <price>10,00</price>
  <idgenre>1</idgenre>
</game>
```

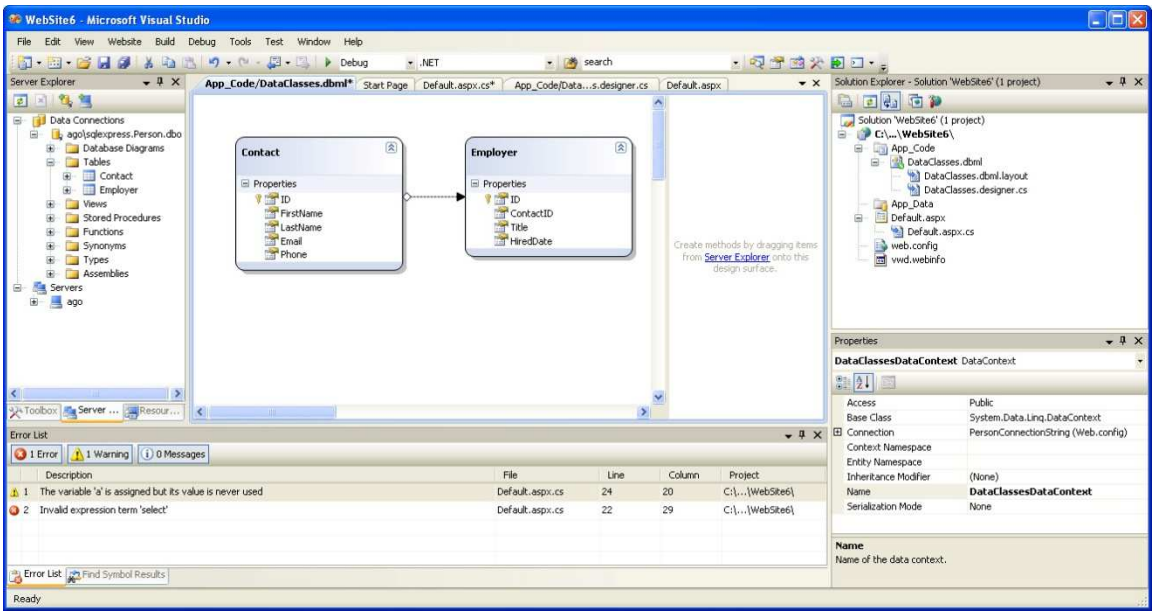
4.4.4 LINQ ja SQL-tietokannat (*engl.* Linq to SQL, LtS)

Tässä kappaleessa LINQ-tekniikoita laajennetaan käyttämään relaatiodataa, jolloin nähdään että pienillä muutoksilla LINQ-kyselyt voidaan kohdistaa tietokantoihin. Suurin tarvittava muutos on datalähteen vaihtaminen. (Eichert 2008, 206.)

LINQ to SQL toimii Microsoft SQL Server -ohjelmiston versioilla 2000, 2005 ja 2008. IQueryable-rajapinnan ansiosta LINQ-tekniikoilla on teoriassa mahdollista käsitellä myös muiden tietokantojen, kuten Oracle-, DB2- ja Access-tietokantojen dataa. (Ferracchiati 2008, 76.)

LtS-projektit aloitetaan aina muuttamalla tietokannan määritykset oliomalleiksi. Tämä onnistuu helposti oliomallieditorilla (*engl.* Object Relational Designer), joka mahdollistaa

oliomallin luomisen tietokannan tauluista vedä ja pudota -toiminnoilla (*engl.* drag and drop). Palvelinselausikkunasta (*engl.* server explorer) raahataan aluksi halutut taulut oliomallieditorin ikkunaan (kuvaa datakontekstia), jolloin Visual Studio kirjoittaa olioiden tarvitsemat koodiluokat. Nyt tauluille ja datakontekstille voidaan tehdä halutut ominaisuusmuutokset (kuvio 35.). Yleensä ainakin datakontekstin nimi-ominaisuus muutetaan selkeämmäksi. Tietokannan tauluista luotuja olioita voidaan käyttää ohjelmakoodissa, kun oliomalli tallennetaan. (Powers 2008, 742-745.)



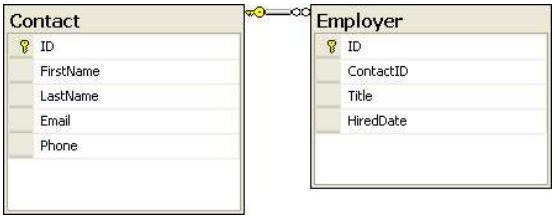
Kuvio 35. Palvelinikkuna, oliomallieditori ja datakontekstin ominaisuudet

Kuvion 35. palvelinikkunassa näkyvän henkilötietokannan teko-ohjeet löytyvät kappaleesta 5: SQL Server.

Ennen tietokantataulujen siirtämistä oliomallieditoriin, Visual Studiossa luodaan yhteys haluttuun tietokantaan. Tämä tapahtuu yhteyden luomistoiminnon (*engl.* add connection) avulla, joka löytyy palvelinikkunasta painamalla hiiren oikeaa painiketta datayhteydet-sarakkeen (*engl.* data connections) kohdalla. (Cox 2008, 79.)

Tietokantayhteyden luominen on selitetty tarkemmin kappaleessa 5: Visual Studio 2008.

Kuviosta 35. voidaan huomata, että luodut olioluokat ovat samanlaisia tietokannan taulujen rakenteiden kanssa (kuvio 36.) (Cox 2008, 79).



Kuvio 36. Person-tietokannan diagrammi SQL Server -ympäristössä

Nyt olioluokat ja tietokantayhteydet ovat valmiina LINQ-kyselyitä varten. Ennen ohjelmointiesimerkkejä tehdään lyhyt katsaus ohjelmakoodiin, jonka Visual Studio kirjoittaa oliomallieditorin kautta. (Powers 2008, 747.)

Oliomallieditorin kirjoittama luokka (DataClasses.designer.cs)

Edellisessä oliomalli-esimerkissä on käytetty kahta tietokantataulua. Niiden pohjalta oliomallieditori luo DataClasses.designer.cs-tiedoston, joka sisältää noin 500 ohjelmakoodiriviä. Luokassa on esimerkiksi tietokantayhteysmääritykset, tietokantataulujen esittelyt, konstruktorit tauluista luoduille olioille, sekä get- ja set-metodit kaikille taulujen ominaisuuksille. (Powers 2008, 747.)

DataClasses.designer.cs-tiedoston toiminnallisuus rakentuu oletuksena `DataClassesDataContext`-nimiseen luokkaan, joka perii `System.Data.Linq.DataContext`-luokan. Datakonteksti (*engl.* datacontext) hallitsee edellä mainittujen ominaisuuksien lisäksi toiminnot tietokantadatan päivittämiselle (insert, update, delete) ja virhetilanteiden käsittelylle. Esimerkiksi jos datakontekstille määritetään tietokantataulun rivin yhtäaikainen poistaminen ja päivittäminen, osaa datakonteksti ensin päivittää rivin ja vasta sitten poistaa sen, jolloin virhetilanteelta vältytään (Powers 2008, 746 -747.)

Insert-, update- ja delete-metodiesittelyiden jälkeen tiedosto sisältää konstruktorit `DataClassesDataContext`-luokalle. Seuraavassa esimerkissä oleva parametritön konstruktori käyttää tietokantayhteyksiin sitä yhteysmerkkijonoa, jonka oliomallieditori on luonut web.config-tiedostoon (tai työpöytäsovelluksissa app.config-tiedostoon). (Powers 2008, 747.)

```
DataClassesDataContext _dc = new DataClassesDataContext();
```

Seuraavassa esimerkissä on web.config-tiedoston yhteysmerkkijono, jota datakontekstin tyhjä konstruktori käyttää. Jos web.config-tiedoston yhteysmerkkijonoa ei haluta käyttää, se voidaan ylikirjoittaa asettamalla konstruktorille parametriksi oma yhteysmerkkijono. (Powers 2008, 745.)

```
<add name="DataClassesConnectionString" connectionString="Data
Source=<SERVERNAME>;Initial Catalog=<DATABASENAME>;Integrated
Security=True" providerName="System.Data.SqlClient"/>
```

Oliomallieditorin kirjoittamaa luokkaa voi muokata vapaasti ja sen voi jopa rakentaa kokonaan itse. On kuitenkin hyvä ymmärtää sen työn laajuus, jonka Visual Studio tekee ohjelmoijan puolesta. (Powers 2008, 747.)

Seuraavassa esimerkissä näkyy luokan alkuun kirjoitettu kommentti, joka kehottaa varovaisuuteen luokan muokkaamisessa (Eichert 2008, 211; Walther 2008, 1409).

```
//------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1434
//
//     Changes to this file may cause incorrect behavior and will be
//     lost if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------
```

LINQ to SQL -kyselyt

Ennen LINQ-kyselyiden kirjoittamista, koodissa esitellään datakonteksti. Tässä esimerkissä datakontekstin nimi on muutettu oliomallieditorissa arvoon PersonDataContext. Muuttujan edessä olevalla alaviivalla varmistetaan että muuttujan nimi ei ole jo varattu .NET Framework -ympäristössä tai oliomallieditorissa luoduissa olioissa. (Ferracchiati 2008, 96)

```
PersonDataContext _dc = new PersonDataContext();
```

Tämän jälkeen voidaan LINQ-kyselyt kohdistaa datakontekstin tauluihin, samalla syntaksilla kuin LtO- ja LtX -esimerkeissä. Seuraavassa esimerkissä on kaikki datakontekstin kontaktitaulun (*engl.* contact) rivit sijoitetaan var-tyyppiseen muuttujaan, jonka jälkeen rivien etunimi-sarakkeet tulostetaan Internet-selaimeen. (Ferracchiati 2008, 86-87)

```
PersonDataContext _dc = new PersonDataContext();

var _contacts = from con in _dc.Contacts
                select con;

foreach (var v in contacts)
{
    Response.Write(string.Format("{0}<br />",v.FirstName));
}
```

Datakonteksti huolehtii tietokantayhteyksien avaamisesta ja sulkemisesta. Silti on suositeltavaa pitää yhteyksiä auki vain sen aikaa kuin on tarpeellista. Linq to SQL -kyselyt voidaan sijoittaa using-lohkojen sisälle, jolloin tietokantayhteydet suljetaan automaattisesti lohkon loputtua. (Eichert 2008, 144.)

Koska var-tietotyyppiä ei voi asettaa null-arvoon, on seuraavassa esimerkissä kyselyn tulos talletettu kontaktioliota sisältävään listaan. Using-lohkon ulkopuolella luotua _contacts -kokoelmaa voidaan käyttää using-lohkon jälkeen, jolloin sinne ei tarvitse kirjoittaa muuta kuin tietokantakäsittelyihin tarvittavaa ohjelmakoodia. (Eichert 2008, 144.)

```
List<Contact> _contacts = new List<Contact>();

using (PersonDataContext _dc = new PersonDataContext())
{
    _contacts = (from con in _dc.Contacts
                 select con).ToList();
}
```

Nyt LINQ-kyselyä voi jalostaa samoin tavoin kuin edellisissä kappaleissa. Seuraavassa esimerkissä on haettu ne työntekijätaulun (*engl.* employer) oliot, joiden titteli (*engl.* title) on ”ohjelmoija”. (Ferracchiati 2008, 100)

```
List<Employer> _employers = new List<Employer>();

using (PersonDataContext _dc = new PersonDataContext())
{
    // SQL-käsky ajetaan tietokantaan heti, kun ToList-metodi
    // suoritetaan
    _employers = (from emp in _dc.Employers
                  where emp.Title == "ohjelmoija"
                  select emp).ToList();
}
```

LINQ-menetelmillä tehdyistä SQL-kyselyistä on usein tiedettävä, milloin varsinainen SQL -kysely ajetaan tietokantaan. Jos kyselyssä käytetään muunnosoperaattoreita (ToList, ToArray, Max, Min jne.), SQL-kysely ajetaan tietokantaan heti. Jos muunnosoperaattoreita

ei käytetä, SQL-kysely ajetaan vasta, kun muuttujaa esimerkiksi iteroidaan foreach -lauseessa. Jos muuttujaa iteroidaan uudelleen, tietokantahaku suoritetaan toisen kerran. Muunnosoperaattoreiden lisäksi LtS-lauseissa voi käyttää kappaleessa 4.4.5 esiteltyjä lambda -menetelmiä. (Taulty 2007 a.)

Jos muunnosoperaattoreita ei siis käytetä, hakulauseet vasta muodostetaan kyselyissä, eikä niitä vielä ajeta tietokantaan. Tällaisia LtS-kyselylauseita voi kirjoittaa vapaasti tarvitsematta välittää ohjelmiston suorituskyvyn heikkenemisestä. Seuraavassa esimerkissä iteroidaan hakutuloksia, joiden kyselylausekkeet on muodostettu kahdessa erillisessä LINQ-lauseessa. Esimerkissä tietokantakysely suoritetaan vasta, kun kokoelmaa iteroidaan foreach -lauseessa. (Taulty 2007 a.)

```
using(PersonDataContext _dc = new PersonDataContext())
{
   //SQL-lauseet muodostetaan kandessa LINQ-lauseessa
   var _contacts = from con in _dc.Contacts
                   select con;

   var _contactsFiltered = from con in _contacts
                           where con.FirstName.StartsWith('a')
                           select con;

   //SQL-käsky ajetaan tietokantaan vasta iteraatiossa
   foreach (var v in _ contactsFiltered)
   {
      Response.Write(string.Format("{0}<br />",v.FirstName));
   }
}
```

CLR-komponentti kääntää LINQ to SQL -kyselyt perinteisiksi SQL-kyselyiksi. Microsoft on luvannut niiden olevan niin tehokkaita kuin niistä on mahdollista tehdä. Seuraavassa esimerkissä on esitelty ohjelmakoodi, jossa työntekijän tietoja haetaan titteli-ominaisuuden mukaan. Esimerkin jälkeen näytetään koodia vastaava SQL-kysely. (Taulty 2007 b.)

```
using (PersonDataContext _dc = new PersonDataContext())
{
    //LINQ-kysely muodostaa SQL-kyselyn:
    var _employers = from emp in _dc.Employers
                     where emp.Title == "Projektipäällikkö"
                     select emp;

   //SQL-kysely ajetaan tietokantaan vasta iteroitaessa:
   foreach (var v in _ employers)
   {
      Response.Write(string.Format("{0}<br />",v.HiredDate));
   }
}
```

Edellisen esimerkin `_contacts`-muuttujan SQL-kielistä sisältöä voi tarkastella virheentarkistustilassa ennen muuttujan iterointia. Seuraavassa esimerkissä on esitetty muuttujaan tallentunut SQL-lause. (Taulty 2007 b.)

SELECT [t0].[ID], [t0].[ID], [t0].[ContactID], [t0].[Title], [t0].[HiredDate]
FROM [dbo].[Employer] AS [t0]
WHERE [t0].[Title] = @p0

Seuraavassa esimerkissä esitetään join-lausekkeen sisältävä LINQ-kysely, ja siitä käännetty SQL-lause. (Taulty 2007 b.)

```
from emp in _dc.Employers
join con in _dc.Contacts on emp.ContactID equals con.ID
where emp.Title == "ohjelmoija"
select new
{
     FirstName = con.FirstName,
     LastName = con.LastName,
     Title = emp.Title,
     HiredTime = DateTime.Now - emp.HiredDate
};
```

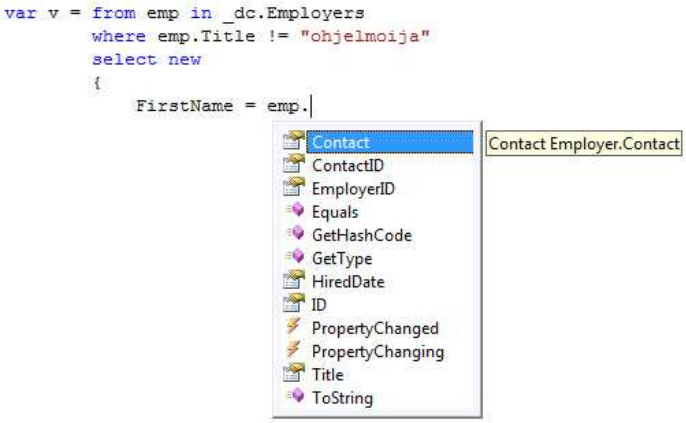
SELECT [t1].[FirstName], [t1].[LastName], [t0].[Title],
CONVERT(BigInt,(((CONVERT(BigInt,DATEDIFF(DAY, [t0].[HiredDate], @p1))) * 86400000) + DATEDIFF(MILLISECOND, DATEADD(DAY, DATEDIFF(DAY, [t0].[HiredDate], @p1), [t0].[HiredDate]), @p1)) * 10000) AS [HiredTime]

FROM [dbo].[Employer] AS [t0]
INNER JOIN [dbo].[Contact] AS [t1] ON [t0].[ContactID] = [t1].[ContactID]
WHERE [t0].[Title] <> @p0

Yksi oliomallieditorin ominaisuuksista on tietokantatauluihin tehtyjen liitosten automaattinen lisääminen myös tauluista muodostettuihin olioihin. Jos edellisen esimerkin työntekijä- ja kontaktitaulun välillä on liitos, ei LINQ-kyselyssä tarvitse käyttää join-lausekkeita. Liitosten ansiosta lapsitaulusta luotu olio löytyy isätaulusta luodun olion ominaisuuksista, joita voi selata ohjelmointiaputoiminnoilla. Seuraavassa esimerkin LINQ-kysely on kohdistettu kahteen tietokantatauluun ilman join-lausekkeita. (Guthrie 2008 b.)

```
from emp in _dc.Employers
where emp.Title == "ohjelmoija"
select new
{
    FirstName = emp.Contact.FirstName,
    LastName = emp.Contact.LastName,
    Title = emp.Title,
    HiredTime = DateTime.Now - emp.HiredDate
};
```

Edellisen esimerkin työntekijäolion ominaisuuksiin kuuluva kontaktiolio näkyy ohjelmointiavussa (kuvio 37.) (Guthrie 2008 b).



Kuvio 37. Kontaktiolio työntekijäolion ominaisuuksissa

Edellinen esimerkki, jossa ei käytetä join-lauseketta, luo saman SQL-kyselyn kuin sitä edellinen join-esimerkki.

Liitoksien tekemisestä SQL Server Management Studio -ohjelmassa enemmän kappaleessa 5: SQL Server.

4.4.5 LINQ ja suorituskyky

LINQ ei tarjoa valmiita ratkaisuja kaikkiin algoritmiongelmiin. Tyypillinen skenaario on etsiä listasta olioita yksittäinen ilmentymä, jolla jokin arvo, esimerkiksi ikä, on suurin. LINQ -kyselyyn kohdistettava Max-operaatio palauttaa numeerisen arvon, ei koko oliota, joten sen käyttö ei riitä. Ongelmaan on monta ratkaisua, joista muutamia esitellään tässä kappaleessa. Esimerkkien tietolähteenä käytetään Microsoftin AdventureWorks-tietokannan

kontaktitaulua, jossa on noin 20 000 riviä. Tietokannan voi kokonaisuudessaan ladata Microsoftin sivuilta (Microsoft 2007). (Eichert 2008, 192.)

Tässä kappaleessa testataan toimintoja, joilla etsitään viimeiseksi muokattu kontaktirivi. Nämä testit mittaavat LINQ to OBJECTS -toimintojen tehokkuutta, eikä niinkään tietokantahakujen nopeutta. Ensimmäisessä esimerkissä näytetään koko tarvittava ohjelmakoodi, mutta myöhemmissä esimerkeissä näytetään vain tiedon karsintaan tarvittava algoritmi. Operaatioihin kuluva aika ilmoitetaan millisekunteina ja tulos otetaan kymmenen ajon keskiarvona. (Eichert 2008, 192.)

Ensimmäisessä testissä käytetään yksinkertaista foreach-silmukkaa, jossa käydään läpi kaikki rivit ja palautetaan viimeiseksi muokattu rivi. Testissä on ajan mittaamiseen käytetty Stopwatch-toiminnallisuuksia. (Eichert 2008, 192.)

```
Stopwatch stopWatch = new Stopwatch();
Contact maxModDate = null;
long lMilliSeconds = 0;

using (DataClassesDataContext _dc = new DataClassesDataContext())
{
    // haetaan kaikki rivit tietokannasta
    List<Contact> contacts = (from con in _dc.Contacts
                              select con).ToList();

    // aloitetaan ajan mittaus
    stopWatch.Start();

    // selataan kaikki oliot läpi, ja talletetaan maxModDate
    // -oliomuuttujaan olio, jonka muokkausaika on suurin
    foreach (Contact c in contacts)
    {
        if ((maxModDate == null) || (c.ModifiedDate >
           maxModDate.ModifiedDate))
        { maxModDate = c; }
    }

    // lopetetaan ajan mittaus
    stopWatch.Stop();
}

lMilliSeconds = stopWatch.ElapsedTicks;
Response.Write(lMilliSeconds);
```

Kulunut aika: 2ms.

Seuraavassa testissä kontaktioliot järjestetään muokkausajan mukaan ja palautetaan listan ensimmäinen olio. (Eichert 2008, 192.)

```
var sortedList = (from c in contacts
                  orderby c.ModifiedDate descending
                  select c).FirstOrDefault();
```

Kulunut aika: 11ms.

Kolmannessa esimerkissä käytetään kahta erillistä kyselyä, joista ensimmäinen palauttaa viimeisimmän muokkausajankohdan ja toinen etsii sitä vastaavat oliot. Joukosta palautetaan ensimmäinen olio. (Eichert 2008, 193.)

```
var maxModifiedDate = contacts.Max(c => c.ModifiedDate);

var maxList = (from c in contacts
               where c.ModifiedDate == maxModifiedDate
              select c).FirstOrDefault();
```

Kulunut aika: 3ms.

Seuraavassa esimerkissä käytetään sisäkkäisiä kyselyitä. Menettelytapa on periaatteeltaan lähes sama kuin edellisessä esimerkissä. (Eichert 2008, 193.)

```
var maxList = (from c in contacts
               where c.ModifiedDate == contacts.Max(co =>
               co.ModifiedDate)
               select c).FirstOrDefault();
```

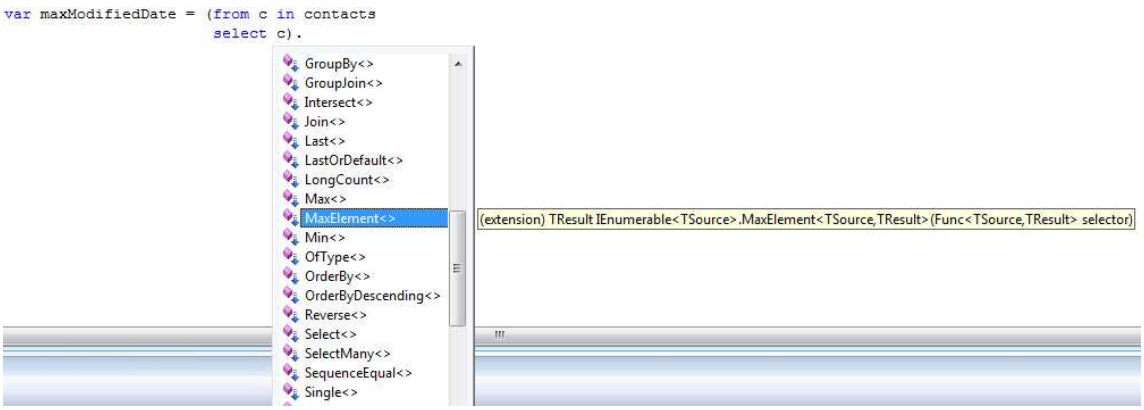
Kulunut aika: 22ms.

Viimeisessä testissä kirjoitetaan oma operaattori, MaxElement, käyttäen uusia C# 3.0 -ominaisuuksia luoda laajennusluokkia. (Eichert 2008, 193-194.)

```
public static class Extension
{
    public static TResult MaxElement<TSource, TResult>(this
IEnumerable<TSource> source, Func<TSource, TResult> selector) where
TResult : IComparable<TResult>
    {
        Boolean firstElement = true;
        TResult result = default(TResult);
        TSource maxValue = default(TSource);

        foreach (TSource element in source)
        {
            var candidate = selector(element);
            if (firstElement || (candidate.CompareTo(result) > 0))
            {
                firstElement = false;
                maxValue = element;
                result = candidate;
            }
        }
        return result;
    }}
```

Laajennusluokka MaxElement näkyy nyt ohjelmointiavussa (kuvio 38.) ja on käytettävissä sekä lambda- että LINQ-kyselyissä (Eichert 2008, 194).

```
var maxModifiedDate = contacts.MaxElement(c => c.ModifiedDate);
```



Kuvio 38. Itse laajennettu ohjelmointiapu LINQ-kyselyssä

Kulunut aika käytettäessä maxModifiedDate-laajennusluokkaa: 2ms.

Taulukosta 1. voidaan nähdä, että eri menetelmillä toteutettujen LINQ to Objects -kyselyiden suorituskyky voi vaihdella (Eichert 2008, 194).

Toteutustapa	Aika (ms)
Foreach-lause	2
Orderby + FirstOrDefault	11
kaksi erillistä kyselyä	3
sisäkkäiset kyselyt	22
oma laajennusluokka	2

Taulukko 1. Olion poimintanopeudet käytettäessä hakuehtona numeerista arvoa

Jos samat testit muutetaan etsimään olioita niiden merkkijonotyyppisen etunimi -ominaisuuden mukaan, suorituskykyerot kasvavat ajon aikana eksponentiaalisesti (taulukko 2.) (Eichert 2008, 194-195).

Toteutustapa	Aika (ms)
Foreach-lause	4
Orderby + first	53
kaksi erillistä kyselyä	5
sisäkkäiset kyselyt	28 000
oma laajennusluokka	4

Taulukko 2. Olion poimintanopeudet käytettäessä hakuehtona merkkijonoa

LINQ-tekniikoita käytettäessä voi eteen tulla suorituskyvyllisiä yllätyksiä, jos ei ole huolellinen ja tunne tekniikoita tarpeeksi hyvin. Sivustojen ohjelmoinnissa uusien tekniikoiden ansiosta saavutetut työtunnit eivät auta, jos sivut latautuvat hitaasti ja eivät toimi oikein. Tässä myös korostuu ohjelmointityön aikana tehtävän testauksen merkitys. (Eichert 2008, 195.)

Jos käytetään numeerisia arvoja, LINQ kuluttaa huolellisesti laadittunakin pahimmillaan kaksinkertaisen määrän aikaa samojen tehtävien tekemiseen (taulukko 3.) (Eichert 2008, 197).

Option	Average time (in ms)	Minimum time (in ms)	Maximum time (in ms)
foreach	68	47	384
for	59	42	383
List<T>.FindAll	62	51	278
LINQ	91	74	404

Taulukko 3. Iterointimenetelmät ja numeeriset ominaisuudet (50 milj. alkiota)

Kun käytetään olioiden merkkijono-tyyppisiä arvoja, aikaerot LINQ-tekniikoiden eduksi pienenevät (Eichert 2008, 198).

Option	Average time (in ms)	Minimum time (in ms)	Maximum time (in ms)
foreach	327	323	361
for	292	288	329
List<T>.FindAll	325	321	355
LINQ	339	377	377

Taulukko 4. Iterointimenetelmät ja merkkijonot (50 milj. alkiota)

String-operaatiot ovat huomattavasti raskaampia kuin numeeristen arvojen vertailu. String -vertailuja käytettäessä LINQ on enää noin 10 prosenttia hitaampi kuin perinteisemmät vertailuoperaattorit (taulukko 4.), kuten Microsoft lupasi LINQ-tekniikoita julkistaessaan. (Eichert 2008, 198.)

LINQ on olioympäristössä huolellisesti käytettynä hyvä vaihtoehto ohjelmistoalgoritmeille. Erityistä tarkkaavaisuutta tulee noudattaa toiminnoissa, jotka ovat ohjelmiston kannalta kriittisiä ja joita ajetaan esimerkiksi satoja kertoja sekunnissa. Koska ohjelmistokoodin selkeys, ylläpito ja tuottavuus paranee, on LINQ usein perinteisempiä algoritmimenetelmiä parempi vaihtoehto. (Eichert 2008, 198.)

4.5 ASP.NET kontrollit ja LINQ

Lähes kaikki Internet-sivut käyttävät tietolähteenään muuttuvaa dataa. Datahakujen lisäksi tarvitaan menetelmiä ja työkaluja, joilla haettu data saadaan visuaalisesti selkeään, toimivaan ja näyttävään muotoon. Lisäksi menetelmät tulee olla nopeasti toteutettavissa. (MacDonald 2007, 341.)

ASP.NET tarjoaa useita eri kontrolleja, joihin voidaan sitoa dataa ja jonka näyttämisen ne voivat tehdä automaattisesti. Kontrollien automatisoitujen toiminnallisuuksien ansiosta ohjelmoijan tarvitsee käyttää paljon vähemmän aikaa datan esityslogiikan laatimiseen. (MacDonald 2007, 341.)

Koska datakontrollit tarvitsevat datalähteen, sille on olemassa oma kontrollinsa. Näitä ovat esimerkiksi SQL-datalähde (engl. sqldatasource) ja uusi LINQ-datalähde (engl. linqdatasource). Datalähde-kontrollien tietokantahaut toteutetaan aspx-sivulla merkkijonona, joten niillä tehtyjen LINQ-kyselyiden virheenkorjaus on vaikeaa. (MacDonald 2007, 341.)

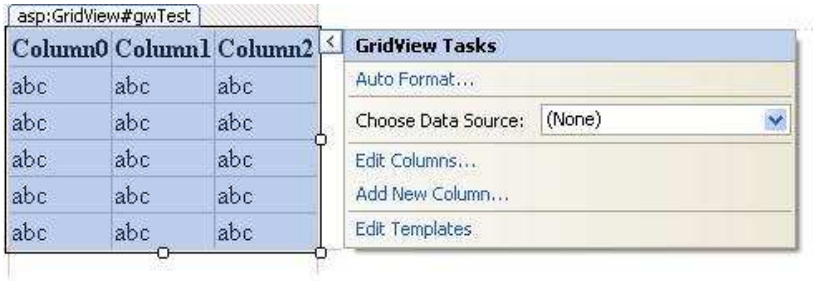
Tässä työssä kontrolleiden datasidonta toteutetaan kooditasolla .cs-luokassa. LINQ -datalähteen käyttöä esitellään kappaleessa 4.5.5: LINQ-datalähde ja listanäkymä.

ASP.NET sisältää kolme pitkälle automatisoitua datan esityskontrollia, taulukkonäkymä, ilmentymänäkymä (*engl.* detailsview) ja listanäkymä. Toistin (*engl.* repeater) ei sisällä lainkaan valmiita grafiikan esitysrutiineja, mutta se on suorituskyvyltään ja muokattavuudeltaan erittäin hyvä esityskontrolli. Datakontrollien lisäksi tässä kappaleessa esitetään osoiteriviparametrien käyttö, sessio-menetelmät (*engl.* querystring ja session) ja datan sivutuskontrolli. (MacDonald 2007, 354-355.)

Data sijoitetaan datakontrolleihin DataSource- ja DataBind-metodeilla. DataSource-metodi määrittää kontrollin datalähteen ja DataBind-metodi tekee datasidonnan. (MacDonald 2007, 355.)

Tämän kappaleen esimerkeissä datalähteenä on käytetty Microsoftin AdventureWorks -tietokannan osasto-taulua (*engl.* department). Tietokannan voi ladata Microsoftin sivulta (Microsoft 2007) ja ohjeet tietokannan käyttöönottoon löytyy kappaleesta 5: Visual Studio.

Jokainen datakontrolli sisältää tehtäväpaneelin (*engl.* tasks), jonka kautta voi muokata esimerkiksi kontrollin ulkoasua ja sisältöä valmiiden grafiikkamallien (*engl.* auto format) ja muokkaustoimintojen (*engl.* edit) avulla (kuvio 39.) (MacDonald 2007, 354).



kuvio 39. Taulukkonäkymä ja sen tehtäväpaneeli

Datakontrollit sisältävät ulkoasuun vaikuttavien tehtäväpaneelitoimintojen lisäksi omia ominaisuuksia, joita muokkaamalla voi vaikuttaa myös kontrollin toiminnallisuuteen. Kontrollit sisältävät lisäksi datan muokkaukseen tarkoitetut syöttö-, poisto- ja päivitystoiminnot. Toimintoihin pääsee käsiksi datakontrollin ominaisuuksien tapahtumalistasta (*engl.* events). (Walther 2008, 23)

4.5.1 Taulukkonäkymä ja ilmentymänäkymä

Taulukkonäkymä on toistimen lisäksi ehkä ASP.NET-ympäristön yleisimmin käytetty datakontrolli. Se korvasi ASP.NET 1.0:n yksinkertaisemman datataulukon (*engl.* datagrid). Taulukkonäkymän tehtävänä on tuottaa grafiikkaa, jossa kontrolliin sidottuja datailmentymiä esitetään riveittäin. Rivit on jaettu ilmentymien niiden ominaisuuksien mukaisiin sarakkeisiin ja taulukon solut täytetään datailmentymien arvoilla (kuvio 40.). (MacDonald 2007, 386.)

DepartmentID	Name	GroupName	ModifiedDate
1	Engineering	Research and Development	1.6.1998 0:00:00
2	Tool Design	Research and Development	1.6.1998 0:00:00
3	Sales	Sales and Marketing	1.6.1998 0:00:00
4	Marketing	Sales and Marketing	1.6.1998 0:00:00
5	Purchasing	Inventory Management	1.6.1998 0:00:00
6	Research and Development	Research and Development	1.6.1998 0:00:00
7	Production	Manufacturing	1.6.1998 0:00:00

Kuvio 40. Department-tietokantataulun sisältöä taulukkonäkymässä

Kuvion 40. mukaisen näkymän laatiminen on yksinkertaista. Halutulle aspx-sivulle vedetään ensin työkaluikkunasta (*engl.* toolbox) taulukkonäkymäkontrolli ja tarvittaessa muutetaan sen nimi ominaisuusikkunasta. Tämän jälkeen kontrolli täytetään LINQ-kyselyiden avulla sivun .cs-tiedoston Page_Load-lohkossa seuraavan esimerkin osoittamalla tavalla. (MacDonald 2007, 323.)

```
using (AWDataContext _dc = new AWDataContext())
{
    //haetaan datakontekstista kaikki Departments-taulun tiedot
    var v = from con in _dc.Departments
            select con;

    //asetetaan GridView-kontrollin datalähteeksi haetut tiedot
    gwTest.DataSource = v;

    //käsketään kontrollia tekemään datasidonta
    gwTest.DataBind();
}
```

Edellisessä koodiesimerkistä voidaan huomata, että aikaisemmin oliomallieditorissa luodun datakontekstin nimi-ominaisuus on muutettu arvoon AWDataContext. Tämä datakonteksti sisältää AdventureWorks-tietokannan osastotaulun. Taulukkonäkymän tunniste on muutettu muotoon gwTest. (MacDonald 2007, 323.)

Esimerkissä kontrollin sivutusominaisuus (*engl.* autogeneratecolumns) on oletuksena asetettu arvoon tosi (*engl.* true). Silloin kontrollin ulkomuodon määrää käytettävä datalähde ja sen ominaisuudet. Jos haluaa määritellä kontrollin sarakkeet ja niiden otsikot itse, sivutusominaisuus tulee asettaa arvoon epätosi (*engl.* false) ja kontrollin sarakkeet tulee määrittää tehtävälistan sarakkeenmuokkauslinkin (*engl.* edit column) kautta (kuvio 39.). (MacDonald 2007, 355.)

Kontrollien sarakkeiden muokkaaminen on selitetty tarkemmin kappaleen 5 ohjelmointioppaassa.

Ilmentymänäkymän tehtävänä on esittää yhtä datalähteen ilmentymää kerrallaan. Myös ilmentymänäkymä muodostaa selaimessa taulukon (kuvio 41.) (MacDonald 2007, 429).

DepartmentID	1
Name	Engineering
GroupName	Research and Development
ModifiedDate	1.6.1998 0:00:00

Kuvio 41. Ilmentymänäkymä Internet-selaimessa

Ilmentymänäkymän datasidonta tehdään Page_Load-lohkossa seuraavan esimerkin mukaisesti (MacDonald 2007, 429).

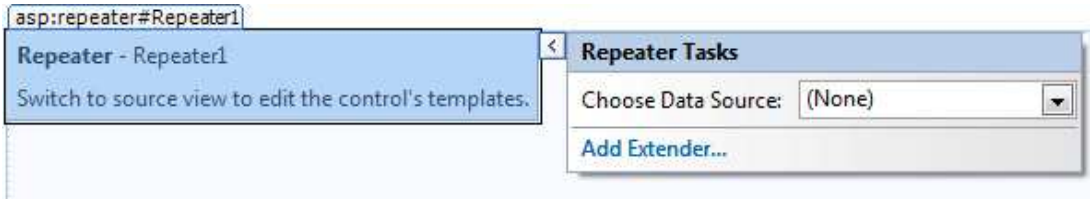
```
using (AWDataContext _dc = new AWDataContext())
{
    // koska ilmentymänäkymä näyttää vain yhden alkion kerrallaan,
    // tulee tietovarastosta hakea jokin tietty ilmentymä
    var dep =  from d in _dc.Departments
               where d.DepartmentID == 1
               select d;

    dwTest.DataSource = dep;
    dwTest.DataBind();
}
```

Taulukkonäkymän lisäksi ilmentymänäkymää ja sen ominaisuuksia käytetään kappaleen 5 ohjelmointioppaassa.

4.5.2 Toistin

Toistin-kontrolli on ollut ASP-NET-ympäristön mukana jo sen versiosta 1.0 asti. Toistin on monimutkaisin ASP-datakontrolli, sillä sen kaikki toiminnot ja graafiset elementit on rakennettava itse. Toisaalta toistin on kontrollina erittäin kevyt, koska siinä datalähteen sisältöä toistetaan ilman muunnoksia, niin monta kertaa kuin siinä on ilmentymiä. Taulukko- ja ilmentymänäkymä käännetään ajon aikana HTML-taulukoiksi. (Cox 2008, 208.)

Toistinta ei voi muokata design-näkymässä, koska sen tehtävälista on design-näkymässä erittäin rajallinen (kuvio 42.) (Walther 2008, 619).



Kuvio 42. Toistin graafisessa suunnittelunäkymässä

Kun näkymä vaihdetaan koodinäkymään, voidaan huomata, että IDE on generoinut graafiseen näkymään raahatusta toistimesta seuraavan esimerkin mukaiset koodirivit (Walther 2008, 619).

```
<asp:Repeater ID="Repeater1" runat="server">
</asp:Repeater>
```

Seuraavassa esimerkissä ASP-koodiin on lisätty pakollinen ItemTemplate-lohko, jonka sisälle kontrollin sisältö kirjoitetaan (Walther 2008, 623-624).

```
<asp:Repeater ID="Repeater1" runat="server">
    <ItemTemplate>
        koodi tähän
    </ItemTemplate>
</asp:Repeater>
```

Seuraavassa esimerkissä data liitetään kontrolliin, joka tapahtuu samoin tavoin kuin taulukkonäkymässä, DataSource- ja DataBind-metodeja käyttäen (Walther 2008, 622).

```
using (AWDataContext _dc = new AWDataContext())
{
    var dep =  from d in _dc.Departments
               select d;

    Repeater1.DataSource = dep;
    Repeater1.DataBind();
}
```

Jos sivu ajetaan nyt, toistin toistaa aspx-sivulla edellisen ASP-koodiesimerkin tekstiä ”koodi tähän” niin monta kertaa kuin kontrollin datalähteessä (muuttuja `dep`) on tietokantailmentymiä. Teksti ”koodi tähän” tulee korvata datakehyksillä, joihin data sijoitetaan. Seuraavassa esimerkissä kontrollin ItemTemplate-lohkoon on lisätty tekstikenttä, johon sijoitetaan datalähteen nimi-ominaisuus. (Walther 2008, 623.)

```
<asp:Repeater ID="Repeater1" runat="server">
    <ItemTemplate>
        <asp:Label ID="lblDepartmentName" runat="server"
           Text='<%# Eval("Name") %>'>
        </asp:Label>
        <br />
    </ItemTemplate>
</asp:Repeater>
```

Datakehykseen määritetty ominaisuus, tässä tapauksessa name, on löydyttävä datalähteestä tai muuten sivu ei toimi. Merkeillä <% ja %> erotettu lohko kuvaa kääntäjälle että kyseessä on tavallisen kontrollin sijasta aktiivista koodia, joka tulee suorittaa erikseen. Edellinen esimerkki listaa selaimeen osasto-taulusta haettujen osastojen nimet (kuvio 43.). (Walther 2008, 623.)

Kuvio 43. Toistin ja sen tuottama sisältö Internet-selaimessa

Toistimen sisällä oleviin kontrolleihin on mahdollista sitoa datalähteen ominaisuuksia myös ilman aspx-sivun aktiivista ohjelmakoodia ja sen Eval-määritteitä lisäämällä ensin kontrollille ItemDataBound-tapahtuma. Seuraavan esimerkin koodi kirjoitetaan kooditiedostoon helpoimmin etsimällä aluksi kontrollin ominaisuusikkunan tapahtumalistasta (salama-ikoni) tapahtuma ItemDataBound. Tapahtuman kaksoispainallus avaa sivun kooditiedoston, ja tapahtuma on kirjoitettu sinne automaattisesti. (Walther 2008, 99.)

```
protected void Repeater1_ItemDataBound(object sender,
RepeaterItemEventArgs e)
{
}
```

RepeaterItemEventArgs-komponentin FindControl- ja DataItem-ominaisuuksien avulla voidaan löytää haluttu kontrolli toistimen sisältä ja kirjoittaa siihen toistimeen sidotun datan ominaisuuksia, kuten seuraavassa esimerkissä (Walther 2008, 99-100).

```
protected void Repeater1_ItemDataBound(object sender,
RepeaterItemEventArgs e)
{
    Department dpt = (Department)e.Item.DataItem;
    Label lblName = (Label)e.Item.FindControl("lblDepartmentName");

    lblName.Text = dpt.Name;
}
```

ItemDataBound-tapahtuma suoritetaan niin monta kertaa, kuin toistimessa on datailmentymiä. Tämän vuoksi ItemDataBound-metodissa ei saa suorittaa tietokantahakuja, koska silloin koko kontrollin suorituskyky heikkenee merkittävästi. Edellisen esimerkin mukainen toistin käsittelee 16 datailmentymää (kuvio 43.), jolloin sen ItemDataBound-tapahtuma käydään läpi 16 kertaa. (Walther 2008, 100.)

Kun datasidonta on tehty toistimeen, DataItem sisältää kontrollin riviin liitetyn olion kaikki ominaisuudet, jota ItemDataBound-metodissa sillä hetkellä käsitellään. DataItem-varaston sisältö voidaan siis muuntaa (*engl.* cast) olioksi, joka on kontrollin datalähteenä. Edellisessä esimerkissä DataItem muunnetaan osasto-olioksi. (Walther 2008, 100.)

Koska toistin ei sisällä sisäänrakennettuja toimintoja tai grafiikkamalleja, sen sisällön lisäksi myös sen graafinen ulkoasu täytyy tehdä itse. Tämä tarkoittaa käytännössä sitä, että ohjelmoijan on osattava HTML- ja ASP-koodin lisäksi myös CSS-tyylien hallintaa. (Walther 2008, 100.)

4.5.3 Listanäkymä

Listanäkymä on ASP.NET Framework 3.5:n ainoa uusi datakontrolli. Listanäkymän toiminnallisuudet ovat sekoitus vanhaa toistin-kontrollia ja uudempaa taulukkonäkymää.

Listanäkymä on toistaiseksi ainoa datakontrolli, joka on yhteensopiva uuden sivutuskontrollin kanssa. (MacDonald 2007, 423.)

Toistimesta poiketen listanäkymä sisältää valmiit mekanismit datanlähteen sisällön näyttämiselle ja muokkaamiselle. Lisäksi listanäkymä sisältää datan automaattiset esitysrutiinit ja tehtäväpaneelin muokkaustoiminnot, jotka tosin ovat käytettävissä vasta, kun kontrollin kanssa käytetään datalähdekontrolleja (esimerkiksi SQL-datalähdekontrollia). Näkökulmasta riippuen listausnäkymä on siis joko yksinkertaistettu toistin tai monimutkaistettu taulukkonäkymä. (MacDonald 2007, 423.)

Listanäkymän toiminnallisuus voidaan ohjelmoida lähes samoin tavoin kuin toistimen: Kooditiedostossa datasidonta tapahtuu identtisesti käytettäessä DataSource- ja DataBind -metodeja, mutta aspx-sivun koodi on monimutkaisempi. Seuraavassa esimerkissä on tehty toistimen ASP-koodiesimerkkiä vastaava toiminnallisuus. (MacDonald 2007, 425)

```
<asp:ListView ID="lwDepartments" runat="server">
    <LayoutTemplate>
        <span ID="itemPlaceHolder" runat="server" ></span>
        <br />
    </LayoutTemplate>
    <ItemTemplate>
        <span>
            <asp:Label ID="lblDepartmentName" Text='<%# Eval("Name")
           %>' runat="server"></asp:Label>

            <br />
        </span>
    </ItemTemplate>
</asp:ListView>
```

Seuraavassa esimerkissä tehdään datalähteen sidota, joka tapahtuu kooditiedostossa samoin tavoin kuin toistimessa (Walther 2008, 622).

```
using (AWDataContext _dc = new AWDataContext())
{
    var dep =  from d in _dc.Departments
               select d;

    lwDepartments.DataSource = dep;
    lwDepartments.DataBind();
}
```

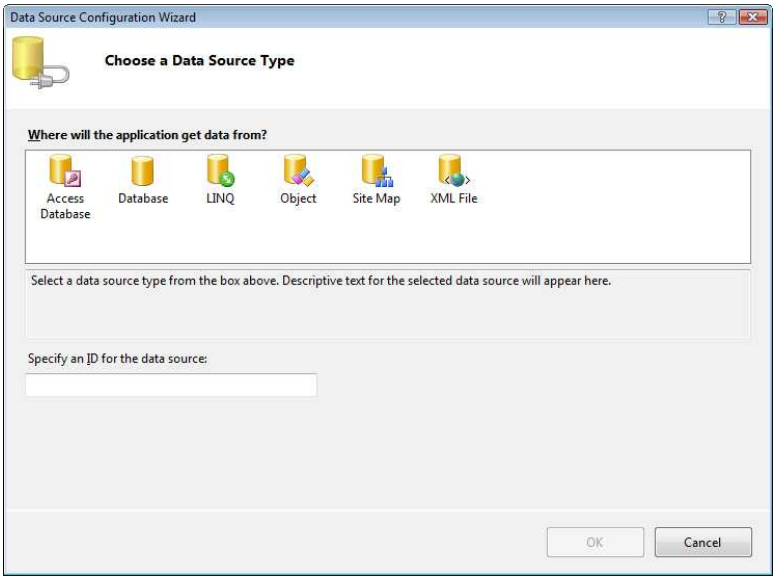
Merkittävin ero listanäkymän ASP-koodissa toistimen verrattuna on pakollinen **LayoutTemplate**-lohko. Sen sisälle on asetettava kontrolli, jonka tunniste asetetaan arvoksi itemPlaceHolder. **ItemTemplate**-lohkon sisälle tulee sivulla esitettävä sisältö, joka ajon

aikana sijoitetaan LayoutTemplate-lohkon itemPlaceHolder-kontrollin sisälle. ItemTemplate -lohkon sisällä oleva koodi tulee sijoittaa samantyyppisen kontrollilohkon sisälle mitä itemPlaceHolder-kontrolli on. Sekä HTML- että ASP-kontrolli käy tähän. Edellisessä ASP-esimerkissä itemPlaceHolder-kontrolli on **span**-tyyppinen. Itse tyypillä ei ole merkitystä, kun Template-lohkojen sisällä olevien kahden kontrollin tyypit ovat yhdenmukaisia. (MacDonald 2007, 424-425.)

4.5.4 LINQ-datalähdekontrolli ja listanäkymä

Datalähdekontrollit mahdollistavat tietokantojen käsittelyn ilman ohjelmakoodin kirjoittamista. Ennen LINQ-datalähdekontrollin käyttöä projektiin tulee rakentaa tarvittava datakonteksti esimerkiksi oliomallieditorilla. (Cameron 2008, 379.)

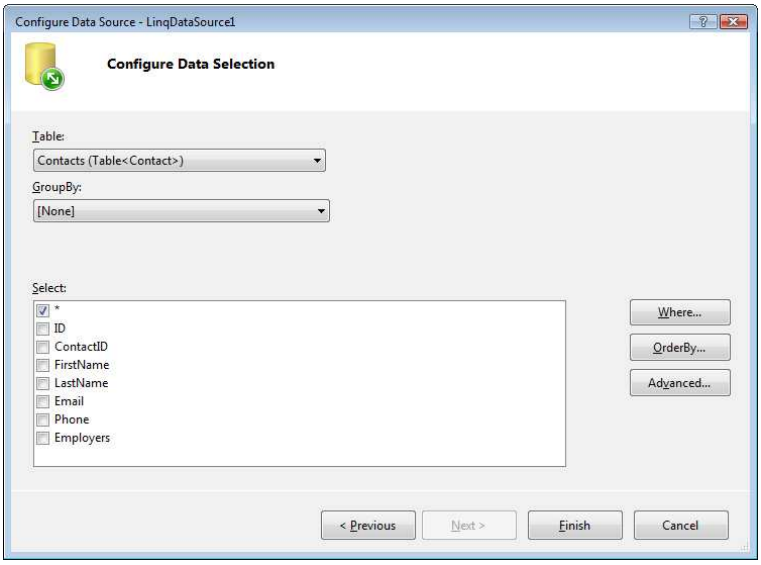
Kun datakonteksti on laadittu ja aspx-sivulle on tehty listanäkymä, voidaan sen tehtävälistan datalähteen valinta -alasvetovalikosta (*engl.* choose data source) valita kohta 'uusi datalähde' (*engl.* New Data Source). Valinta johtaa kuvion 44. mukaiseen dialogiin. (MacDonald 2007, 580.)



Kuvio 44. Uuden datalähteen valinta

Jos tarkoituksena on käsitellä tietokantoja LINQ-kyselyiden avulla, valitaan kohta LINQ ja painetaan OK. Seuraavassa dialogissa tulee valita haluttu datakonteksti. Jos projektiin on luotu vain yksi datakonteksti, tässä dialogissa tarvitsee vain painaa seuraava-painiketta (*engl.*

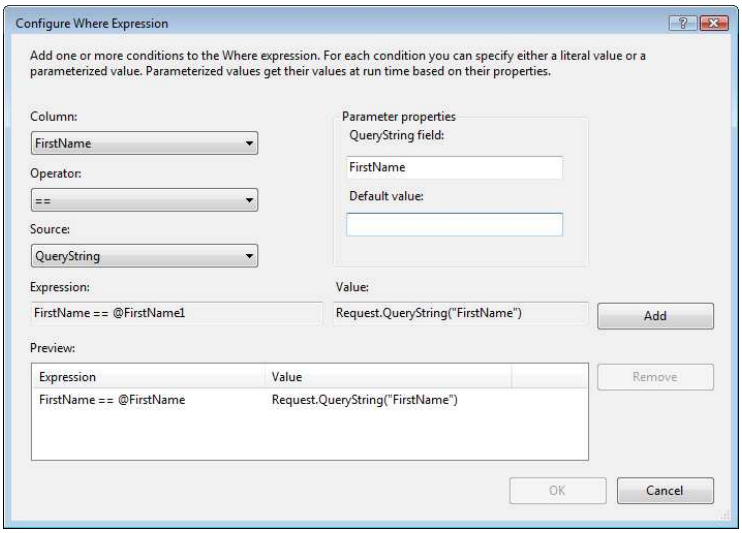
next). Näin päästään kolmanteen dialogiin, jossa määritetään tietokannasta palautettavien olioilmentymien muoto, eli mitä ominaisuuksia oliot sisältävät (kuvio 45.). (MacDonald 2007, 580.)



Kuvio 45. Datalähteen määrittäminen (engl. configure data source)

Datalähteen määritysdialogista (kuvio 45.) voidaan palautettavien olioiden ominaisuuksien lisäksi ryhmitellä (engl. groupby) hakutulokset jonkin olion ominaisuuden mukaan. Dialogi mahdollistaa myös where- ja orderby-kyselylausekkeiden muodostamisen erillisten dialogien avulla. (MacDonald 2007, 580.)

Painettaessa Where-painiketta, näkyviin aukeaa jälleen uusi dialogi (kuvio 46.), joka sisältää useita alasvetovalikoita ja tekstikenttiä, joiden avulla tietokantakutsun ehto-osa voidaan rakentaa. (MacDonald 2007, 580.)

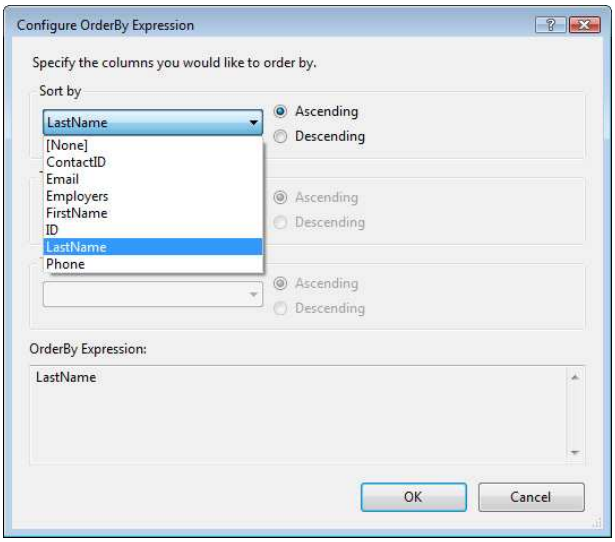


Kuvio 46. Where-ehdon määrittäminen dialogissa

Kuvion 46. dialogin sarake-alasvetovalikko (*engl.* column) määrää mihin tietokantataulun ominaisuuteen vertailu kohdistetaan. Operaattorivalikosta (*engl.* operator) valitaan tilanteeseen sopiva vertailutapa (yhtä suuri, pienempi kuin, suurempi kuin) ja lähdevalikosta valitaan se tietolähde, mistä tietokantataulun ominaisuudelle otetaan vertailudata. (MacDonald 2007, 580.)

Kuviossa 46. on rakennettu ehtolause jossa etsitään tietokantailmentymiä FirstName -ominaisuuden mukaan. Vertailudata siinä haetaan selaimen osoiteriviltä (*engl.* querystring) FirstName-kentästä. Jos tätä kenttää ei löydy, voidaan data hakea oletusarvokenttään (*engl.* default value) kirjoitetusta merkkijonossa, joka kuviossa 46. on tyhjä. (MacDonald 2007, 582.)

Add-painikeen painaminen luo ehdon, jonka jälkeen dialogi voidaan sulkea painamalla OK -painiketta. Datalähteen määritysdialogin (kuvio 45.) OrderBy-painikkeesta aukeavasta dialogista voidaan hakutuloksia järjestää jonkin tietyn tietokantataulun sarakkeen mukaan Kuviossa 47. järjestys on tehty sukunimi-sarakkeen mukaan. Järjestyksen suunnan voi määrätä Sort by -alasvetovalikon vieressä olevista valitsimista. Kun toiminnot on tehty, muutokset talletetaan ja dialogi suljetaan OK-painikkeesta. (MacDonald 2007, 580.)



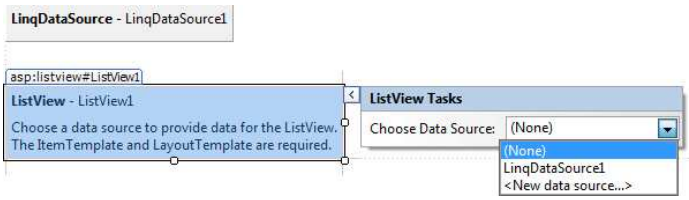
Kuvio 47. OrderBy-dialogi

Datalähteen määritysdialogissa rakennettu datalähdekontrolli hyväksytään lopetuspainikkeella (*engl.* finish) (kuvio 45.), jolloin aspx-sivulle seuraavan esimerkin mukainen LINQ-datalähde (MacDonald 2007, 581).

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="PersonDataContext" EnableDelete="True"
    EnableInsert="True" EnableUpdate="True" OrderBy="LastName"
    TableName="Contacts"
    Where="FirstName == @FirstName">

    <WhereParameters>
           <asp:QueryStringParameter Name="FirstName"
           QueryStringField="FirstName" Type="String" />
    </WhereParameters>
</asp:LinqDataSource>
```

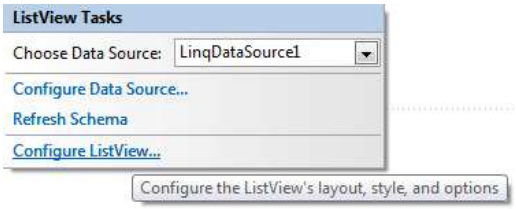
LINQ-datalähde sisältää samat osat kuin .cs-luokkaan koodatut LINQ- tai SQL-kyselyt. Erona näillä on se, että datalähdekontrollissa tietokantakyselyn osat ovat kontrollin ominaisuuksia. (MacDonald 2007, 581.)

Kun LinqDataSource-kontrolli on valmis, voidaan sivulle lisätä työkaluikkunasta listanäkymä haluttuun kohtaan aspx-sivua. Ensimmäiseksi tämän kontrollin tehtävälistasta tulee valita datalähteeksi edellä luotu LinqDataSource1 (kuvio 48.). (Cox 2008, 124.)



Kuvio 48. Listanäkymä ja sen tehtävälista

Datalähteen asettamisen jälkeen tehtävälistan muut toiminnallisuudet tulevat käyttöön (kuvio 49.). Nyt listanäkymän ulkoasua voi muokata erillisessä dialogissa (*engl.* configure listview). (Walther 2008, 666-667.)



Kuvio 49. ListViewin tehtävälista datalähteen asettamisen jälkeen

4.5.5 Sivutuskontrolli

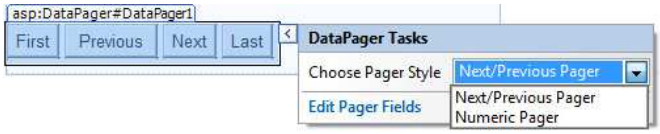
Sivutuskontrolli toimii ainoastaan niiden datakontrollien kanssa, jotka tukevat IPageableItemContainer-rajapintaa. Tällä hetkellä ainoastaan listanäkymä tukee tätä rajapintaa. (Walther 2008, 677.)

Useissa vanhemmissa ASP.NET-datakontrolleissa sivutustoiminnot ovat olleet osana kontrollia, jolloin sivutustoimintojen hallinta on ollut hankalaa. Uusi sivutuskontrolli tarjoaa mahdollisuuden sivutuspainikkeiden vapaaseen sijoitteluun ja toiminnallisuuksien monipuolisempaan määrittelyyn. (MacDonald 2007, 428)

Ennen sivutuskontrollin käyttämistä tulee sivulle lisätä listanäkymä, jolle on määritetty jokin datalähde. Datalähteen voi määrittää manuaalisesti ASP- ja C#-koodin avulla (kappale 4.5.3) tai erillisten velhojen avulla (kappale 4.5.4). (Walther 2008, 677.)

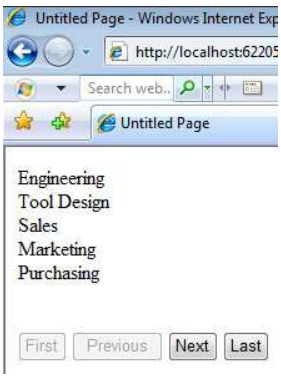
Kun listausnäkymä on valmis, siirretään sivutuskontrolli sivulle ja määritetään sen PageControlID- ja PageSize-ominaisuudet tarvittaviin arvoihin. PageControlID-ominaisuus määrää datakontrollin, johon sivutustoiminnot halutaan liittää ja PageSize-ominaisuus kuinka monta tietokantailmentymää yhdellä sivulla näytetään. (Walther 2008, 677.)

Näiden kahden ominaisuuden lisäksi kontrollin esitystapa tulee valita sen tehtävälistasta (kuvio 50.). Valmiiden sivutustyylien lisäksi kontrollin ulkoasua voi muokata vapaammin kontrollin editointilinkistä (*engl.* edit pager fields). (Walther 2008, 677.)



Kuvio 50. Datapager-kontrolli ja tehtävälista

Jos kontrolli rakennetaan kappaletta 4.9.3 noudattaen, selaimeen piirtyy kuvion 50. mukaisilla sivutuskontrollin ominaisuuksilla kuvion 51. kaltainen näkymä (PageControlID- ja PageSize-ominaisuudet on muutettu arvoon lwDepartments ja 5) (Walther 2008, 677).



Kuvio 51. Listanäkymä ja sen alapuolella olevat sivutuspainikkeet

4.5.6 Sessio ja osoiteriviparametrit

Sessio-muuttujien ja osoiteriviparametrien käyttö ovat menetelmiä, joiden avulla on helppo siirtää tietoa sovelluksen sivulta toiselle tai myös saman sivun sisällä, ilman että tiedot katoavat PostBack-prosessin aikana (Cameron 2008, 243; Walther 2008, 432).

Osoiteriviparametrien käyttö on näistä kahdesta menetelmästä suoraviivaisempi ja sen tarkoituksena onkin puhtaiden merkkijonojen siirtäminen osoiterivillä (*engl.* URL) sivulta toiselle. Session muuttujatyyppi, johon voi tallettaa esimerkiksi Luokkia, olioita tai tavallisia muuttujia. Osoiteriviparametrit ovat voimassa niin kauan, kuin ne ovat osoiterivillä ja vastaavasti sessiomuuttuja pysyy palvelimen muistissa kunnes käyttäjä sulkee selaimen, tai muuttuja ohjelmallisesti tyhjennetään. (Cameron 2008, 243-245; Walther 2008, 432-435.)

Osoiteriviparametrit lisätään selaimen osoiteriville, jolloin ne voidaan lukea toisella sivulla ja käyttää niitä siellä muuttujina. Parametri-osa erotetaan muusta URL-osoitteesta kysymysmerkin avulla. Osoitteen muut kysymysmerkit eivät haittaa, kunhan osoitteen osat eivät ole samannimisiä osoiteriviparametrien kanssa. Seuraavassa esimerkissä on muodostettu URL-osoite, jossa on kaksi parametria, PageId ja Role. Parametrit eroitetaan toisistaan &-merkeillä. (Walther 2008, 434.)

http://www.kajak.fi/testisivu.html**?**PageId=1**&**Role=student

Parametrit poimitaan osoiteriviltä kooditiedostossa seuraavan esimerkin mukaisesti (Walther 2008, 434).

```
string sPageId = Request.QueryString["PageId"];
string sRole   = Request.QueryString["Role"];
```

`Request.QueryString`-menetelmää käytettäessä isot ja pienet kirjaimet ovat merkitseviä. Koska parametrit ovat osana osoiteriviä, ne ovat aina merkkijono-tyyppisiä. (Walther 2008, 434.)

Sessio-muuttujan käyttötavat ovat monipuolisempia. Seuraavassa esimerkissä intArray -nimiseen session-muuttujaan on talletettu numeroita sisältävä taulukko. (Cameron 2008, 243.)

```
Session["intArray"] = new Array[1,4,2];
```

Kun data on talletettu sessioon, sitä voidaan käyttää kaikilla sivuston sivuilla. Seuraavassa esimerkissä sessiomuuttujan sisältö puretaan taulukkoon, Array-muunnoksen avulla. (Cameron 2008, 243.)

```
Array iArray = (Array)Session["intArray"];
```

Sessio-muuttuja on käytön jälkeen järkevää alustaa null-arvoon, jos sitä ei enää tarvita. Tällä vähennetään palvelinrasitusta. (Cameron 2008, 243.)

Sessio on erityisen käytännöllinen tilanteissa, joissa halutaan siirtää suurempi määrä dataa toisen sivun käyttöön. Seuraavassa esimerkissä tietokannasta LINQ-kyselyllä palautunut data sijoitetaan sessioon. (Walther 2008, 1240.)

```
List<Employer> _employers = new List<Employer>();

using (PersonDataContext _dc = new PersonDataContext())
{
    _employers = (from emp in _dc.Employers
                  where emp.Title == "ohjelmoija"
                  select emp).ToList();
}

Session["lstEmployers"] = _employers;
```

Seuraavassa esimerkissä lstEmployers-sessio puretaan takaisin työntekijälistaksi, missä tahansa sivuston kooditiedostossa (Walther 2008, 1240).

```
List<Employer> _employers = (List<Employer>)Session["lstEmployers"];
```

Sivutilamuuttuja (*engl.* viewstate) on samankaltainen sessiomuuttujan kanssa, mutta sivutila tyhjennetään aina, kun käyttäjä navigoi toiselle sivulle. Muuten sivutilamuuttujan käyttö ei eroa sessio-muuttujasta. (Walther 2008, 1241.)

4.6 .NET AJAX

Perinteisesti AJAX (*engl.* Asynchronous JavaScript And XML) tarkoittaa asiakaspään tekniikoita, joissa verkkosivuilla JavaScript-kielellä tehdyistä HTTP-pyynnöistä palautetaan XML-merkkauskielen mukainen tieto. Palvelin ja selain vaihtavat tietoa taustaprosessina, ilman että koko sivu ladataan uudelleen. (Walther 2008, 1598.)

ASP.NET on palvelinteknologia web-sovellusten laatimiseen, johon nimi ASP viittaa (*engl.* Active Server Pages). ASP-tekniikoissa lähes kaiken työn tekee web-palvelin eikä selain. Kun käyttäjä tekee sivulla jonkin toiminnon tai tapahtuman, esimerkiksi painaa nappia, koko sivu lähetetään palvelimelle. Muutokset palautetaan käyttäjälle postback-prosessissa. (Walther 2008, 1598.)

Usein pelkkien ASP.NET-tekniikoiden käyttäminen sivuilla voi johtaa huonoon käyttäjäkokemukseen. Toimintojen seurauksena sivun välkkyminen tai pahimmillaan koko sivu ei vastaa käyttäjän pyyntöihin. Esimerkiksi kuvat ja tekstit voivat osittain hävitä kokonaan ja jos odotusaika on pitkä, asiakas todennäköisesti poistuu sivustolta. Tämänkaltainen odottelu on nykyisissä Internet-sivuissa tavallista. (Walther 2008, 1598.)

Suuntaus on nyt kohti asiakaspäätä. Jos sovelluskehittäjät aikovat laatia aidosti käyttäjäystävällisiä sivustoja, on uskallettava siirtää palvelinpuolen toimintoja asiakkaan selaimeen. Hyvä esimerkki tästä on Google Docs (http://docs.google.com), joka todistaa, että Microsoft Office-tyyppisen ohjelmistokokonaisuuden voi siirtää Internet-selaimeen omaksi sovelluksekseen. Google Docs mahdollistaa asiakirjojen, taulukkolaskentatiedostojen ja dia-esitysten tallennuksen keskuspalvelimelle, joten ne eivät voi kadota ja niihin pääsee käsiksi lähes kaikkialta. Google Docsissa samaa työtä voi tehdä yhtä-aikaa useampi henkilö, toisin kuin Microsoftin Officessa. Google Docs -ohjelmiston perustana on AJAX. (Walther 2008, 1598.)

Microsoft kehitti perustan AJAX-tekniikoiden alle tehdessään tukea omalle Exchange -sähköpostisovellukselleen. Siksi on ironista että sovellus, joka nosti AJAX-tekniikat suureen suosioon, on Googlen Gmail-sähköposti. (Walther 2008, 1599.)

AJAX-sovellus on asiakaspään web-ohjelmisto, joka nojaa selaimen omaan natiivitekniikkaan kuten JavaScript-kieleen. Kaikki liikenne tällaisen sovelluksen ja palvelimen välillä hoidetaan webservice-kutsuilla, jolloin ASP-tyylistä sivun päivitysrutiinia ei tarvita. (Walther 2008, 1599.)

Palvelinpään web-ohjelmistossa painikkeen painaminen aiheuttaa siis sivun lähettämisen palvelimelle ja vasta kun sivu palautetaan muutoksineen takaisin asiakaspäähän, napin painaminen rekisteröityy käyttäjälle. Asiakaspään AJAX-ohjelmistossa painikkeen painamisesta aiheutuvat toiminnot tapahtuvat heti käyttäjän selaimessa eikä palvelimella. AJAX-menetelmissä käyttöliittymä sijaitsee selaimessa ja bisneslogiikka sekä web-kerrokset siitä eteenpäin web-palvelimella. (Walther 2008, 1599.)

Microsoftin omia AJAX-tekniikoita kutsutaan nimeltä ASP.NET AJAX ja Microsoft AJAX Framework. Visual Studio -ympäristössä on mahdollista tehdä sekä palvelinpään että asiakaspään AJAX-sovelluksia. Asiakaspään AJAX-toimintojen käyttö vaatii Microsoft AJAX Framework -kokonaisuuden käyttöä ja JavaScript-kielen osaamista. (Walther 2008, 1600.)

JavaScript on oliopohjainen ohjelmointikieli, kuten C#. JavaScript-kielessä ei kuitenkaan ole lainkaan luokkarakenteita eivätkä oliot ole sidoksissa toisiinsa olemalla instansseja samoista luokista. Toinen merkittävä ero JavaScript- ja C#-kielen välillä on JavaScript-kielen dynaamiset muuttujat. Kun JavaScript-koodi ajetaan voi esimerkiksi string-muuttuja muuttua useaan kertaan Interger-muuttujaksi ja päinvastoin. (Walther 2008, 1696.)

Palvelinpään ASP.NET AJAX ei vaadi JavaScript-koodia, vaan AJAX-toiminnallisuuksia käytetään kuten tavallisia Visual Studion ASP-kontrolleja. ASP.NET AJAX -tekniikoiden käyttö on helpompaa, mutta ei yhtä joustavaa ja muokattavaa kuin asiakaspään AJAX. ASP.NET AJAX-tekniikoissa suurin osa toiminnallisuudesta on automatisoitu, jolloin ohjelmoijan vastuulle jää lähinnä kontrollien ominaisuuksien määrittäminen ja niiden taustalogiikoiden laadinta. (Walther 2008, 1600.)

Visual Studio 2008 sisältää natiivisti muutamia AJAX-kontrolleja, joita ovat päivityspaneeli (*engl.* updatepanel), päivitysilmoitin (*engl.* updateprogress) ja ajastin (*engl.* timer). ASP.NET AJAX sisältää huomattavasti suuremman valikoiman eri kontrolleja, mutta Visual Studio -ympäristöön ne on yhä asennettava erikseen. Kaikki ASP.NET AJAX -kontrollit tarvitsevat skriptinhallintakontrollin (*engl.* scriptmanager) jokaiselle sivulle, joissa AJAX-kontrolleja käytetään. Vaihtona on määrittää skriptinhallintakontrolli pelkästään ylisivussa, jolloin skriptienhallinta on käytössä niillä sivuilla, jotka perivät ylisivun. (Walther 2008, 1600; Powers 2008, 26-30.)

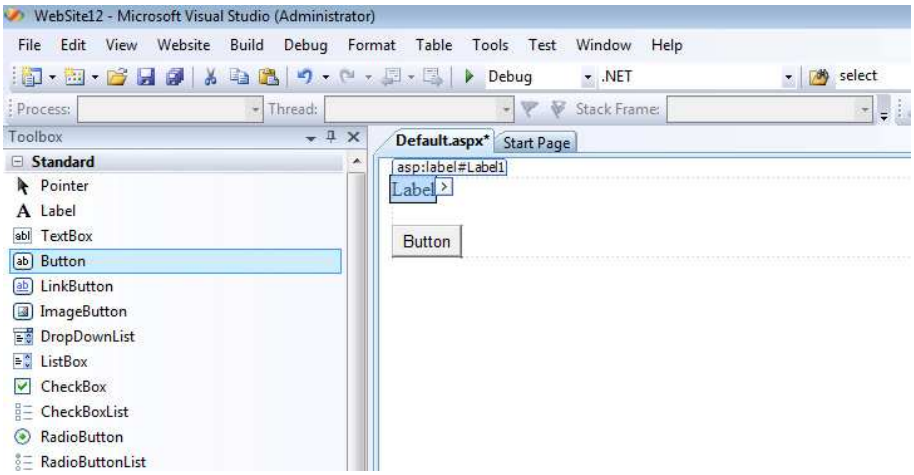
Visual Studion AJAX -kontrollit liittyvät kaikki osittaisiin sivunpäivityksiin. Kun www -sivujen käyttäjäkontrolleja aktivoidaan (esimerkiksi painetaan painiketta), koko sivu lähetetään ensin palvelimelle ja sitten muutoksineen takaisin käyttäjän selaimeen. Tämä aiheuttaa sivun piirtämisen uudelleen ja suuremmissa sivustoissa esimerkiksi kuvat voivat latautua hitaasti. Selainten välimuisti auttaa kuvien latauksessa, mutta silti käyttäjäkokemus ei aina ole sulava. (Powers 2008, 654.)

ASP.NET AJAX -kontrollien tehtävänä on nostaa käyttäjäkokemuksen tasoa. Kontrolleista päivityspaneeli mahdollistaa datan lähettämisen palvelimelle vain osasta sivua, eikä koko sivusta. Selaimessa päivitetään sivusta vain haluttu alue, jolloin muu osa säilyy muuttumattomana. Näin vältytään sivujen uudelleenpiirtämiseltä ja oikein käytettynä myös palvelinrasitus vähenee. (Powers 2008, 656.)

Päivityspaneeli

Päivityspaneeli mahdollistaa osittaiset sivunpäivitykset. Seuraavassa esimerkissä näytetään kontrollin käyttö käytännössä. (MacDonald 2007, 1344.)

Aluksi luodaan uusi Website-projekti (File - New - Website) ja projektin default.aspx -sivulle raahataan työkaluikkunasta tekstikenttä- ja painike-kontrollit (kuvio 52.). (MacDonald 2007, 1344.)

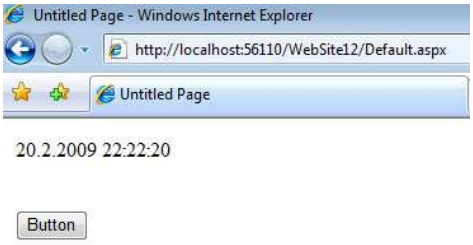


Kuvio 52. Default.aspx-sivu ja siihen tekstikenttä ja painike

Painiketta kaksoispainaltamalla avautuu default.cs-tiedoston `Button1_Click`-tapahtuma, jonka sisälle kirjoitetaan seuraavan esimerkin mukainen koodirivi. Koodi sijoittaa tekstikenttään laitteiston kellonajan, kun painiketta painetaan. (MacDonald 2007, 1344.)

```
Label1.Text = DateTime.Now.ToString();
```

Edellisen esimerkin koodirivi kannattaa sijoittaa myös `Page_Load`-lohkon sisälle, jos haluaa että kellonaika näkyy tekstikentässä, kun sivu ajetaan ensimmäisen kerran (kuvio 53.). (MacDonald 2007, 1346.)



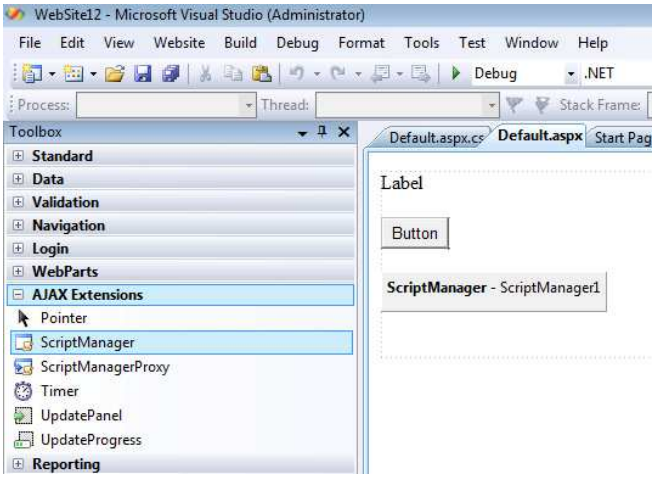
Kuvio 53. Ajastinpäivyri selaimessa

Kun painiketta painetaan, kellonaika tekstikentässä muuttuu. Esimerkin koodimäärä on niin pieni, että painikkeen painaminen tuskin aiheuttaa selaimessa huomattavaa muutosta. On mahdollista, että tekstikentän teksti häviää hetkeksi. Viivettä on keinotekoisesti mahdollista kasvattaa itse lisäämällä `Button1_Click`-tapahtumaan säie, joka asettaa tapahtuman käsittelylle viiveen. Viive ilmoitetaan millisekunteina. Säie tulee kirjoittaa tekstikentän tapahtumaan ennen kellonaikaa, kuten seuraavassa koodiesimerkissä. (MacDonald 2007, 1354.)

```
System.Threading.Thread.Sleep(4000);
Label1.Text = DateTime.Now.ToString();
```

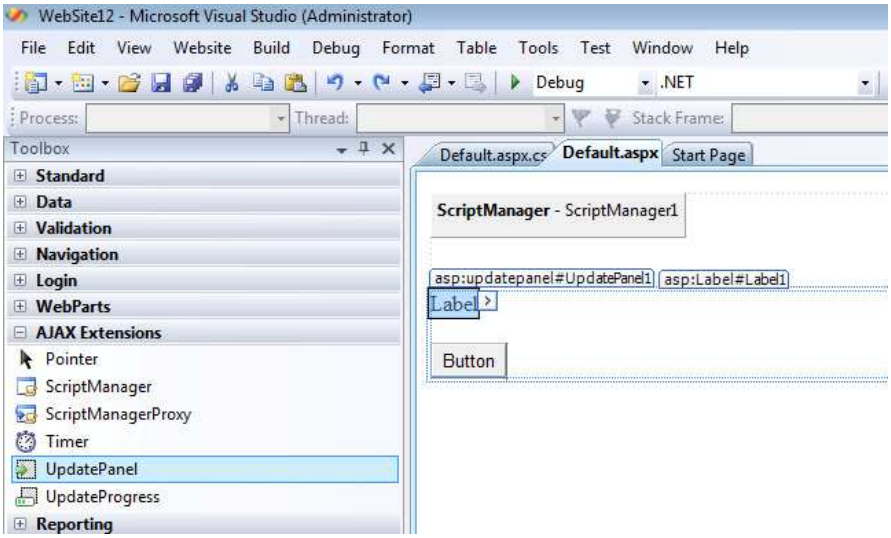
Säikeen ansiosta painikkeen painaminen pysäyttää sivun latautumisen, tässä tapauksessa neljäksi sekunniksi. Ennen päivityspaneelia sivulle sijoitetaan ASP.NET AJAX -kontrolleille pakollinen skriptinhallintakontrolli. Sen tehtäviä ovat esimerkiksi ASP.NET AJAX -kontrollien renderöinti, globalisointi ja lokalisointi. Sen voi ajatella olevan pieni CLR -komponentti ASP.NET AJAX -kontrolleille. (MacDonald 2007, 1354.)

Kun skriptienhallintakontrolli on asetettu sivulle, graafisen näkymän tulisi näyttää kuvion 54. mukaiselta (MacDonald 2007, 1354).



Kuvio 54. Skriptinhallintakontrolli

Skriptien hallintakontrollin sijainnilla ei ole merkitystä, sillä kontrolli ei näy Internet -selaimessa. Nyt sivulle lisätään päivityspaneeli ja sen sisälle sijoitetaan kaikki ne kontrollit, joiden halutaan kuuluvan osittaiseen sivunpäivitysrutiiniin. Päivityspaneeli on rajattu graafisessa näkymässä pisteviivoituksella. (Powers 2008, 656.)

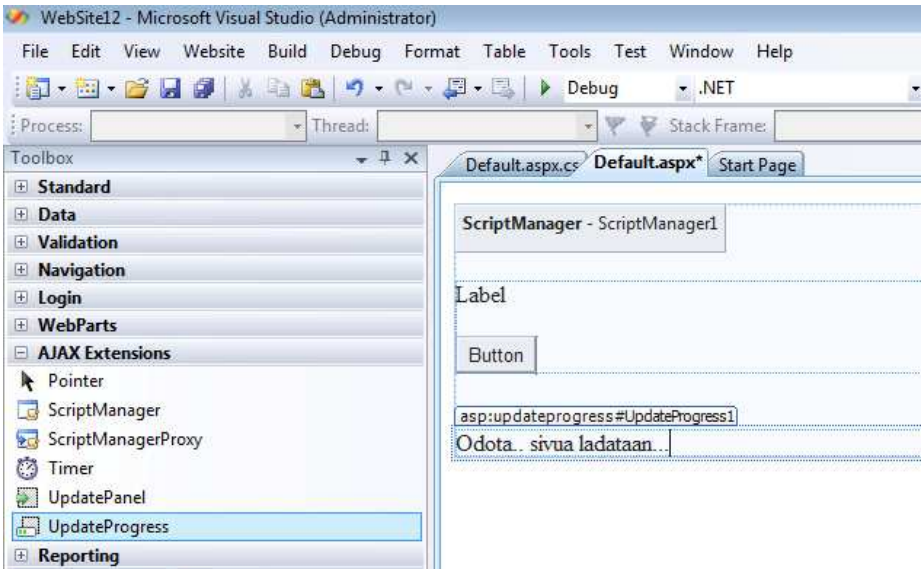


Kuvio 55. Graafinen näkymä, päivityspaneeli

Päivitysilmoitin

Sivun latautuessa kontrollit eivät vastaa käyttäjän pyyntöihin, riippumatta päivityspaneelin käytöstä. Jos lataus kestää useita sekunteja, on järkevää että latautumisesta ilmoitetaan käyttäjälle. Koska päivityspaneeli ei piirrä sivua uudelleen, voi muuttumaton ja toimintoihin vastaamaton sivu olla käyttäjälle hämmentävä. Siksi päivityspaneelin kanssa kannattaa käyttää aina päivitysilmoitinta, jolla voi ilmoittaa käyttäjälle viestein tai esimerkiksi anomoiduin gif -kuvin sivun latautumisesta. Päivitysilmoittimen voi liittää vain päivityspaneeliin. (Powers 2008, 656.)

Edelliseen esimerkkiin päivitysilmoitin on helppo liittää. Se täytyy sijoittaa päivityspaneelin vaikutusalueen **ulkopuolelle** ja skriptinhallintakontrollin **alapuolelle**. Kun päivitysilmoitin on sivulla, voi sen sisälle esimerkiksi siirtää uusia kontrolleja tai kirjoittaa tekstiä (Kuvio 56.). (Powers 2008, 657.)

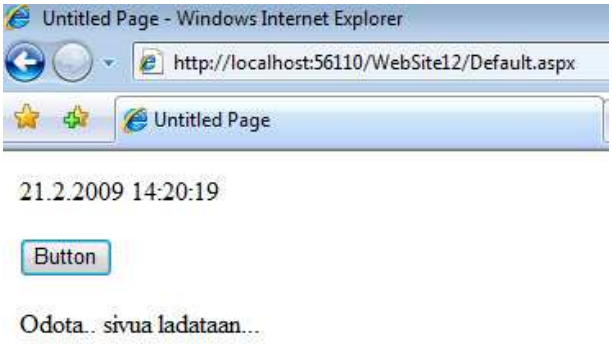


Kuvio 56. Graafinen näkymä sekä päivitysilmoitin ja siihen asetettu teksti

Päivitysilmoittimen sisältö näytetään, kun päivityspaneelin sisältöä päivitetään. Kun päivitys on suoritettu, ilmoitinkontrolli ja sen sisältö häviää. Päivitysilmoitin näkyy sivua ajetaessa siinä kohdassa, mihin se on asetettu. (Powers 2008, 659.)

Päivitysilmoitin on vielä liitettävä päivityspaneeliin. Se tehdään asettamalla ilmoitinkontrollin ominaisuus AssociatedUpdatePanelID samaan arvoon kuin päivityspanelin ID -ominaisuus. Tässä esimerkissä AssociatedUpdatePanelID on asetettava arvoon

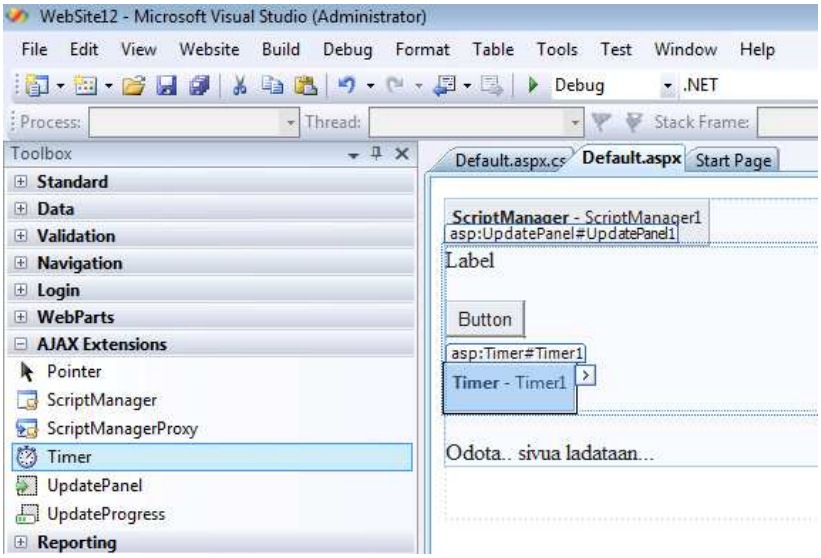
UpdatePanel1. Tämän ominaisuuden muuttamisen jälkeen sivu näyttää päivityspaneelin päivityksen aikana kuvion 57. mukaiselta. (Powers 2008, 660.)



Kuvio 57. UpdateProgress-kontrolli ja sen latausviesti

Ajastin

Ajastin mahdollistaa päivityspaneelin päivittämisen halutun ajanjakson välein käyttäjän toimista riippumatta. Ajastinta ei tarvitse liittää erikseen päivityspaneeliin, sillä ajastin päivittää automaattisesti sitä päivityspaneelia, jonka sisällä se on. (Powers 2008, 660.)



Kuvio 58. Timer-kontrolli UpdatePanelin sisällä

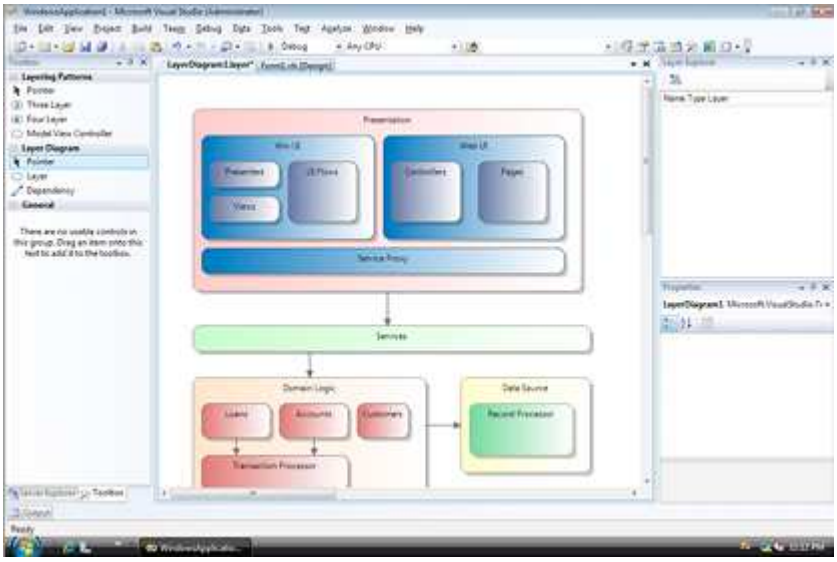
Ajastimen Interval-ominaisuus määrää päivityspaneelin päivitysvälin. Kuten sivun säie -esimerkissä, myös tämä ominaisuus ilmoitetaan millisekunteina. Jos ominaisuus asetetaan arvoon 2000, päivityspaneeli päivitetään kahden sekunnin välein. Jos kuvion 57. mukaista

sivua testataan selaimessa ajastimen Interval-arvolla 2000, päivitysilmoitin ilmestyy esiin automaattisesti kahden sekunnin välein. (Powers 2008, 660.)

4.7 Visual Studio 2010 ja .NET Framework 4.0

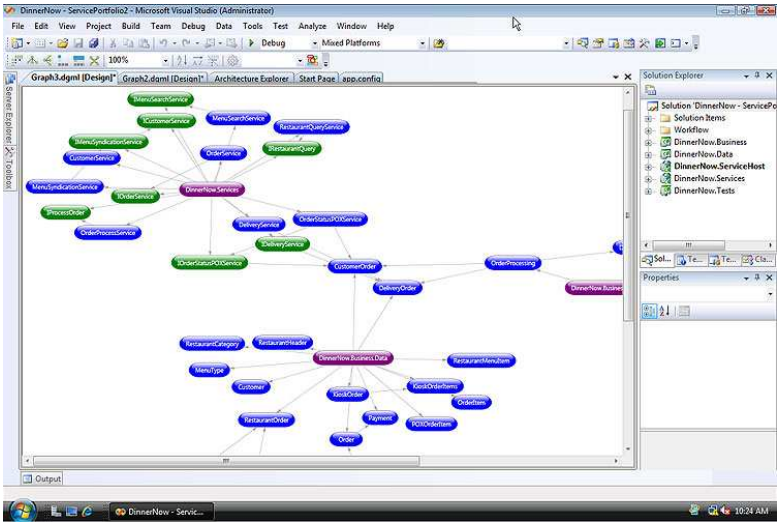
Uudesta Rosario-ohjelmointialustasta on annettu vasta vähän informaatiota. Vuoden 2008 talvella Visual Studio 2010:stä oli saatavilla lyhytaikainen lisenssi CTP-esittelyversioon (*engl.* community technology preview). Tämä versio oli kooltaan noin seitsemän gigatavua, ja myös sen laitteistovaatimukset olivat korkeat. CTP-versio oli käyttöjärjestelmän levykuva (*engl.* image), joka tuli ajaa Microsoftin omalla Virtual PC 2007 -ohjelmalla. (Redmond 2008.)

VS 2010:nen lehdistötilaisuuksissa on luvattu uusia työkaluja lähinnä projektinhallintaan ja -testaukseen, sekä Team Foundation Server -toimintoihin. Luvassa on lisäksi erilaisia mallinnuskieliä tukevia mallinnustyökaluja, joita voi käyttää suunnittelun lisäksi myös kuvaamaan jo olemassaolevia ohjelmistoarkkitehtuureja (kuvio 59.). (Redmond 2008.)



Kuvio 59. Projektirakenteen mallinnustyökalu

Näistä työkaluista mielenkiintoiseksi on osoittautunut myös rakennenäkymä jossa ohjelmiston luokat, loogiset lohkot ja niiden väliset riippuvuudet esitetään graafisesti (kuvio 60.) (Redmond 2008).



Kuvio 60. Sovelluksen rakenne graafina

Yksi ohjelmointiin liittyvistä uudistuksista tulee olemaan uusi virheentarkistustyökalu, joka toimii kuten lentokoneen musta laatikko, joka tallettaa ajon aikana laitteiston ja ohjelmoiston tilaa. Kun testaaja löytää virheen, siitä voidaan ottaa kuvankaappaus ja koko ”musta laatikko” voidaan palauttaa ohjelmoijalle. (Redmond 2008.)

Koska Visual Studio 2008 ja sen .NET Framework 3.5 (ja Windows Vistan .NET Framework 3.0) muodostivat yhdessä valtavan, uusia teknologioita runsaasti sisältävän kokonaisuuden, voi olettaa että seuraavat versiot tuovat mukanaan saman suuruusluokan muutoksia. .NET-ohjelmoijien kannattaakin seurata Microsoftin julkaisuja tasaisin väliajoin, sillä jos tulossa on saman kokoluokan tekniikoita kuten esimerkiksi WPF, WCF tai LINQ, odotus on tuskin turhaa. (Redmond 2008.)

5 STEP-BY-STEP -OHJE LINQ TO SQL -OHJELMISTOON

Tässä oppaassa laaditaan askel askeleelta LINQ to SQL -tekniikoita käyttäen pieni ASP.NET-sovellus, jonka tarkoituksena on opastaa ASP-kontrollien käyttöä sekä tietokantoihin kohdistuvia operaatioita LINQ-tekniikoita käyttäen. Tarvittavia työkaluja ovat Visual Studio 2008 RTM ja SQL Server 2005 Management Studio. Myös Management Studio -ohjelmistolla tehtävät toiminnot opastetaan vaihe vaiheelta.

Tässä esimerkissä ei kiinnitetä huomiota ohjelmoinnin tietoturvaan eikä syvällisemmin n-tier -projektimalleihin.

Käytettävistä ASP-kontrolleista tähän projektiin valittiin ASP 2.0:n taulukko- ja ilmentymänäkymä, koska katsottiin tarpeelliseksi esitellä tarkemmin vanhempien kontrollien käyttö LINQ-ympäristössä. Muita ASP-datakontrolleja tässä oppaassa ei käytetä.

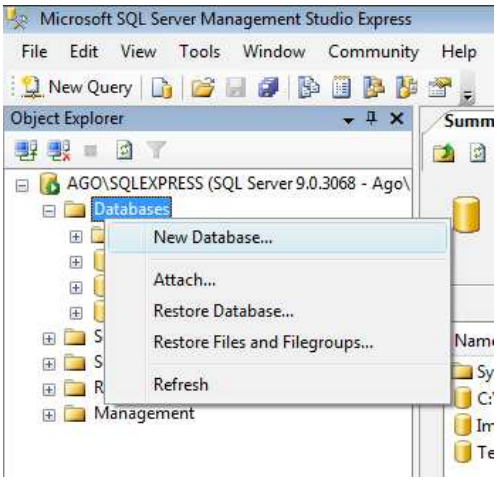
SQL Server Management Studio

SQL Server Management Studio -työkalulla luodaan työntekijä- ja kontaktitaulut (kuvio 1.). Työntekijätaulu toimii tietovarastona vain työntekijän välittömille tiedoille ja kontaktitaulu hänen tarkemmille yhteys- ja henkilötietoineen. Taulut yhdistetään toisiinsa liitoksella, joka tehdään työntekijätaulun kontaktitunnisteen (*engl.* contactid) ja kontaktitaulun tunnisteen välille.

Kuvio 1. Taulujen diagrammi

1) Uuden Tietokannan luominen

Käynnistä SQL Server Management Studio (myös express-versio käy) ja kirjaudu sisään. Luo uusi tietokanta painamalla hiiren oikealla painikkeella projekti-ikkunan (*engl.* object explorer) tietokantakansiota (*engl.* databases) ja valitsemalla kohta 'uusi tietokanta' (*engl.* new database) (kuvio 2.).

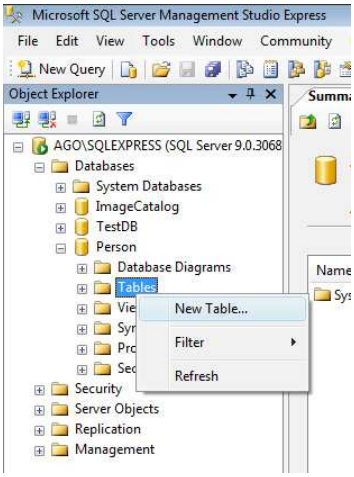


Kuvio 2. Uusi tietokanta

Anna uudelle tietokannalle nimeksi Person, kohtaan 'tietokannan nimi' (*engl.* database name) ja paina OK.

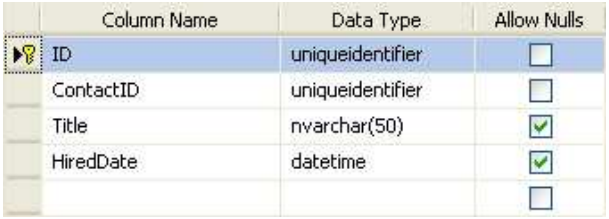
2) Työntekijätaulun luominen

Navigoi selausikkunasta auki edellä luotu henkilötietokanta. Luo siihen uusi taulu painamalla taulut-kansion (*engl.* tables) kohdalla hiiren oikeaa painiketta ja painamalla kohtaa 'uusi taulu' (*engl.* new table) (kuvio 3.). Työntekijätauluun talletetaan tittelin lisäksi esimerkiksi työhönoton päiväys (*engl.* hireddate).



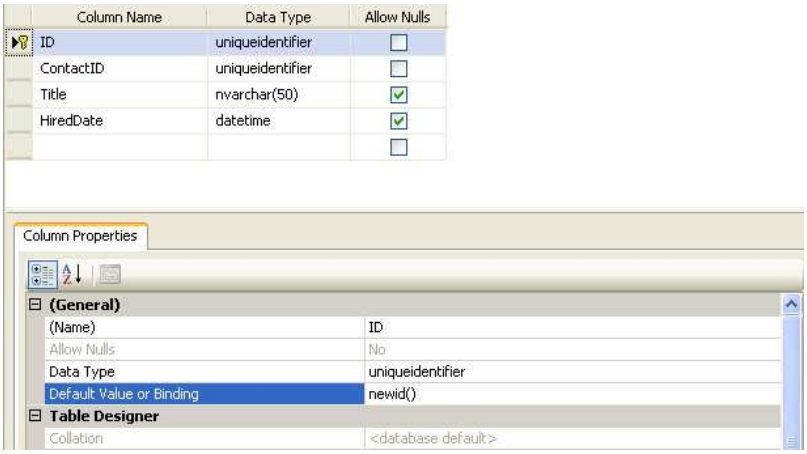
Kuvio 3. Uusi taulu

Laadi taululle kuvion 4. mukaiset rivit ja aseta sarakkeen ominaisuuksista tunnistesarakkeen (ID) oletusarvoksi newid() (kuvio 5.). Kun tauluun lisätään uusi rivi, kyseisten sarakkeiden arvot kirjoitetaan newid-käskyn ansiosta automaattisesti. Komento newid() luo sarakkeeseen satunnaisen GUID-tyyppisen arvon (*engl.* global unique identifier). Aseta myös sarakkeen 'salli null-muuttujat' -valintapainikkeet (*engl.* allow nulls) kuvion 4. osoittamalla tavalla.



Column Name	Data Type	Allow Nulls
ID	uniqueidentifier	☐
ContactID	uniqueidentifier	☐
Title	nvarchar(50)	☑
HiredDate	datetime	☑
		☐

Kuvio 4. Employer-taulun sarakkeet



Column Name	Data Type	Allow Nulls
ID	uniqueidentifier	☐
ContactID	uniqueidentifier	☐
Title	nvarchar(50)	☑
HiredDate	datetime	☑
		☐

Column Properties

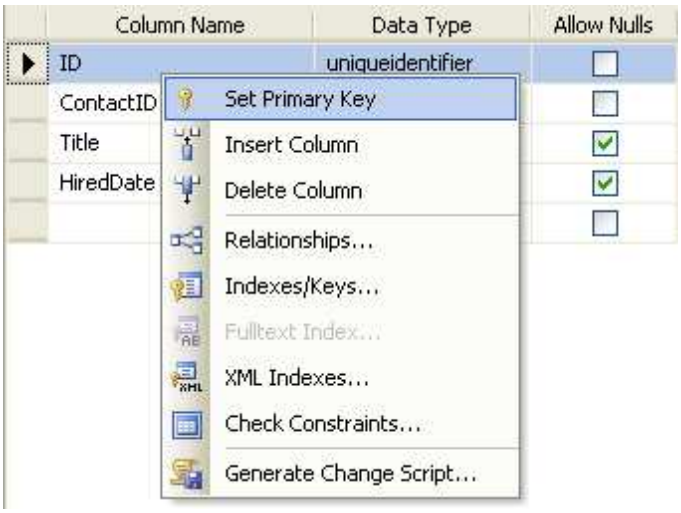
(General)	
(Name)	ID
Allow Nulls	No
Data Type	uniqueidentifier
Default Value or Binding	newid()
Table Designer	
Collation	<database default>

Kuvio 5. Default Value or Binding -arvo sarakkeen ID ominaisuuksissa

Visual Studion GUID-muuttuja (vastaa SQL-ympäristön uniqueidentifier-tietotyyppiä) on 128-bittiPersonen integer-arvo, joka usein esitetään heksadesimaalimuotoisena. Heksadesimaaliksi käännettynä Guid on muotoa xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (jossa x = 0-9 tai a-h), jossa yksiköiden määrä on 8-4-4-4-12. Heksadesimaaliarvona Guid voi olla esimerkiksi 3f0c92ec-416d-472e-bb48-9a6affe280b0. Guid-tyyppisiä arvoja on mahdollista olla niin suuri määrä, että toisen samanlaisen arvon generoinnin katsotaan olevan mahdotonta. Mahdollisia variaatioita on 2^{128}. Guid soveltuu hyvin tietokantojen tunnistekentiksi ja avainsarakkeiksi. (Microsoft 2008 g.)

3) Primääriavaimen määrittäminen

Aseta tunnistesarake taulun primääriavaimeksi painamalla sarakkeen kohdalla hiiren oikeaa painiketta ja valitsemalla kohta 'aseta primääriavaimeksi' (*engl.* set primary key) (kuvio 6.). Samalla sarakkeen vasempaan laitaan ilmestyy avaimen kuva. Jos avaimen laittaa väärään sarakkeeseen, avaimen saa poistettua kyseisestä sarakkeesta samasta valikosta (*engl.* remove primary key).



Kuvio 6. Perusavaimen asettaminen

4) Taulun tallettaminen

Talleta taulu Save Table_1 -kohdasta, joka löytyy esimerkiksi ohjelmiston File-valikosta. Kirjoita aukeavaan dialogiin taulun nimeksi Employer ja paina OK. Taulu on valmis.

5) Kontaktitaulun luominen

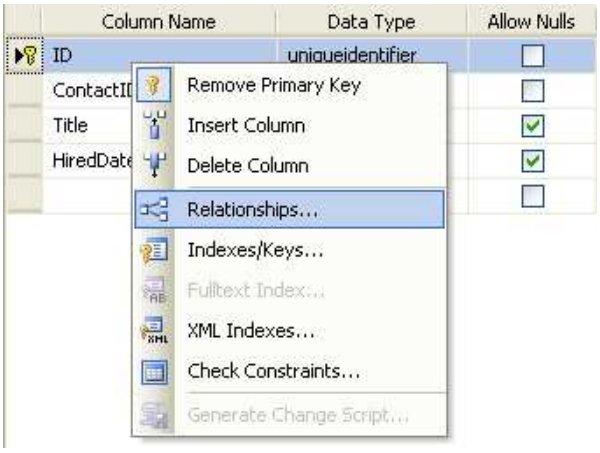
Luo kontaktitaulu samaan tietokantaan vaiheiden 2-4 mukaisesti kuvion 7. osoittamilla arvoilla. Määritä taulun tunnistekenttä taulun primääriavaimeksi, mutta **älä** aseta sen oletusarvoksi newid()-metodia. Tallenna lopuksi taulu nimellä Contact.

Column Name	Data Type	Allow Nulls
ID	uniqueidentifier	☐
FirstName	nvarchar(50)	☑
LastName	nvarchar(50)	☑
Email	nvarchar(50)	☑
Phone	nvarchar(50)	☑
		☐

Kuvio 7. Kontaktitaulu

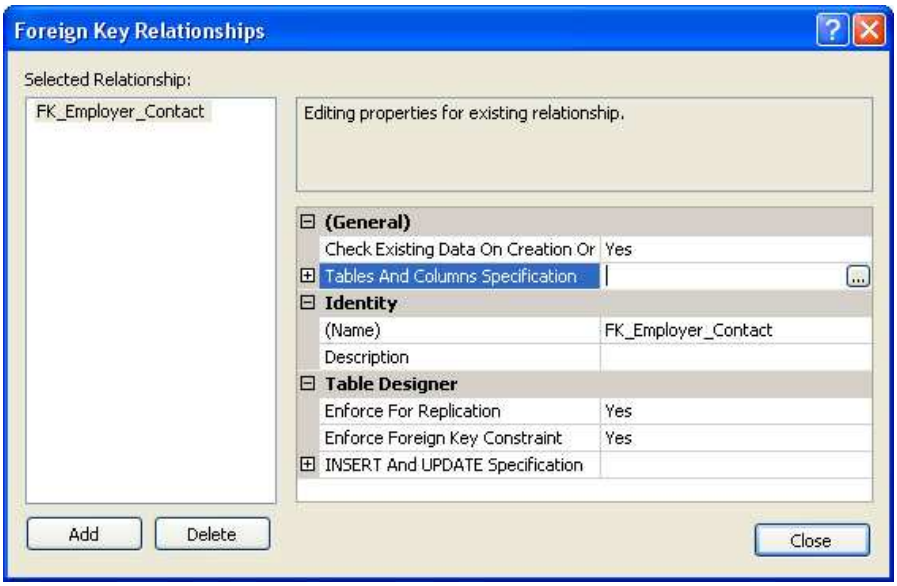
6) Liitoksen määrittäminen taulujen välille

Nyt kun tarvittavat taulut on tehty, tulee niiden välille luoda liitos. Avaa työntekijätaulun muokkausnäkymä (*engl.* modify), joka löytyy painamalla selausikkunassa taulua hiiren oikealla painikkeella. Valitse jokin taulun sarakkeista ja paina liitoslinkkiä (*engl.* relatioships) hiiren oikealla painikkeella aukeavasta valikosta (kuvio 8.).



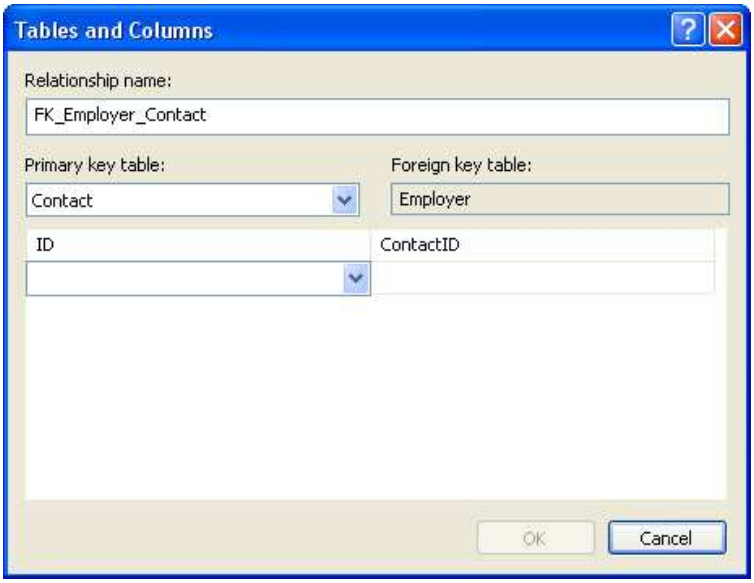
Kuvio 8. Relationships-valinta

Avautuvasta dialogista (kuvio 9.) paina lisäyspainiketta (*engl.* add). Klikkaa liitoksen määritysriviä (*engl.* tables and columns specification) ja paina tyhjän tekstialueen oikeassa laidassa olevaa painiketta, jossa on kolme pistettä.



Kuvio 9. Relationships-dialogi

Aseta avautuvan dialogin arvot kuvion 10. osoittamalla tavalla ja paina OK. Tallenna vielä molemmat taulut (tallentamattomat muutokset näkyvät tähtenä taulun välilehdellä). Nyt tietokanta on valmis ohjelmointia varten.



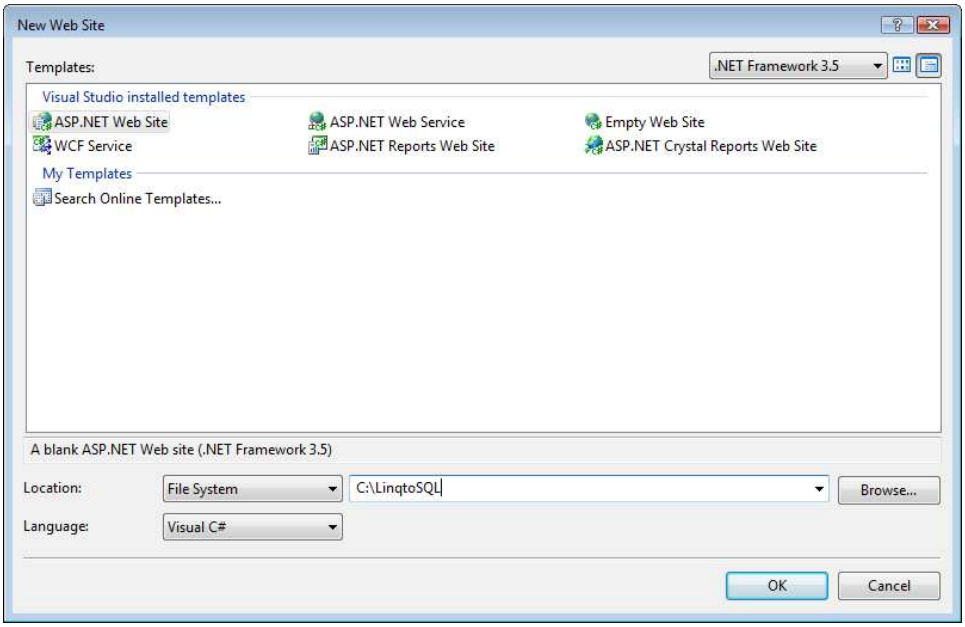
Kuvio 10. Liitoksen määrittäminen

Visual Studio 2008

Uuden projektin ja LINQ to SQL -luokkien luominen

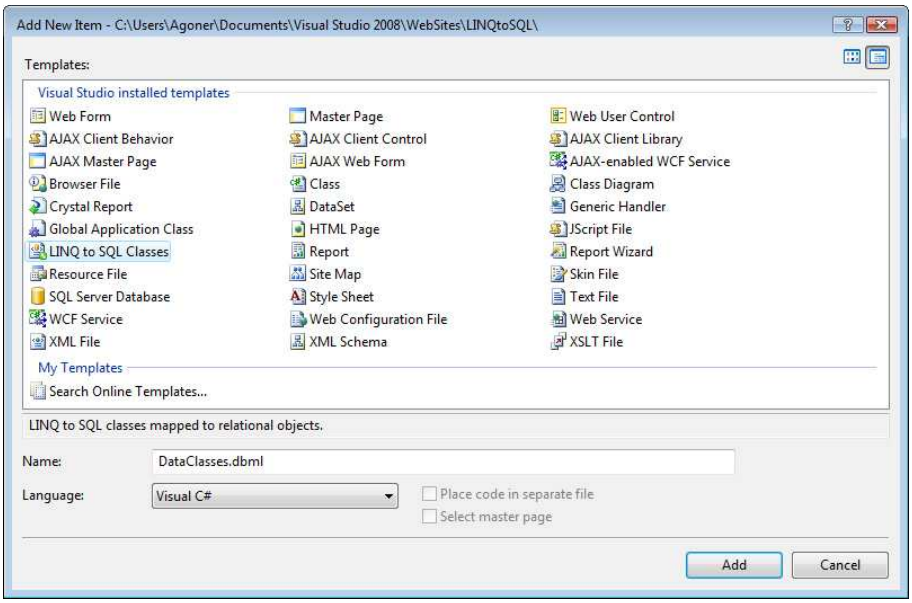
1) Uuden web-projektin luominen

Avaa Visual Studio 2008 ja avaa uuden web-sivuston luomisdialogi (File => New => Website) ja anna projektille nimeksi LINQtoSQL. Ennen OK-painikkeen painamista tarkista että dialogin oikeassa yläreunassa olevassa pudotusvalikossa lukee .NET Framework 3.5 (kuvio 11.).



Kuvio 11. Uuden web-projektin luominen

Paina projektia projektien selausikkunasta hiiren oikealla painikkeella ja valitse kohta 'uusi tiedosto' (*engl.* new item). Valitse avautuvasta dialogista LINQ to SQL -luokka (*engl.* LINQ to SQL classes) ja paina Add-lisäyspainiketta (luokan oletusnimi DataClasses.dbml käy hyvin) (kuvio 12.).



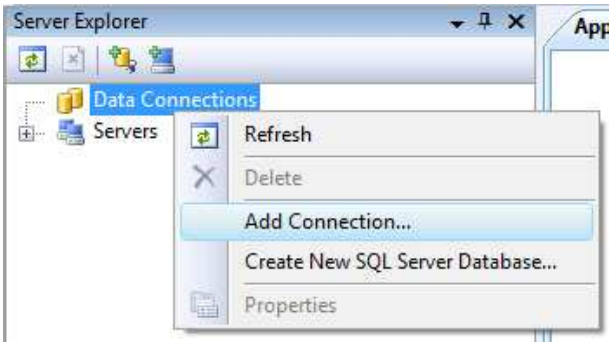
Kuvio 12. Uuden tiedoston luominen projektiin

Visual Studio kysyy dialogissa, talletetaanko uusi luokka App_Code-hakemistoon. Hyväksy dialogi ja avaa DataClasses.dbml-tiedosto tuplaklikkaamalla sitä projekti-ikkunasta.

2) Uuden tietokantayhteyden luominen Visual Studioon

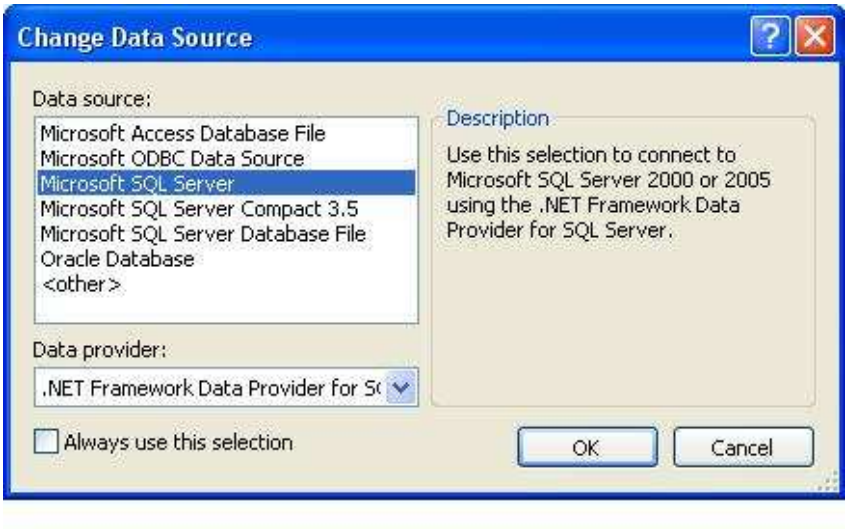
Tietokantojen käyttäminen Visual Studio -ympäristössä vaatii datayhteysten luomisen SQL Server -ympäristöön ja sen tietokantoihin. Tämä yhteys on luotava myös käytettäessä LINQ to SQL -tekniikoita.

Navigoi palvelinikkunassa kohtaan datayhteydet (*engl.* data connections) ja valitse valikosta kohta 'lisää yhteys' (*engl.* add connection) (kuvio 13.).



Kuvio 13. Yhteyden lisääminen

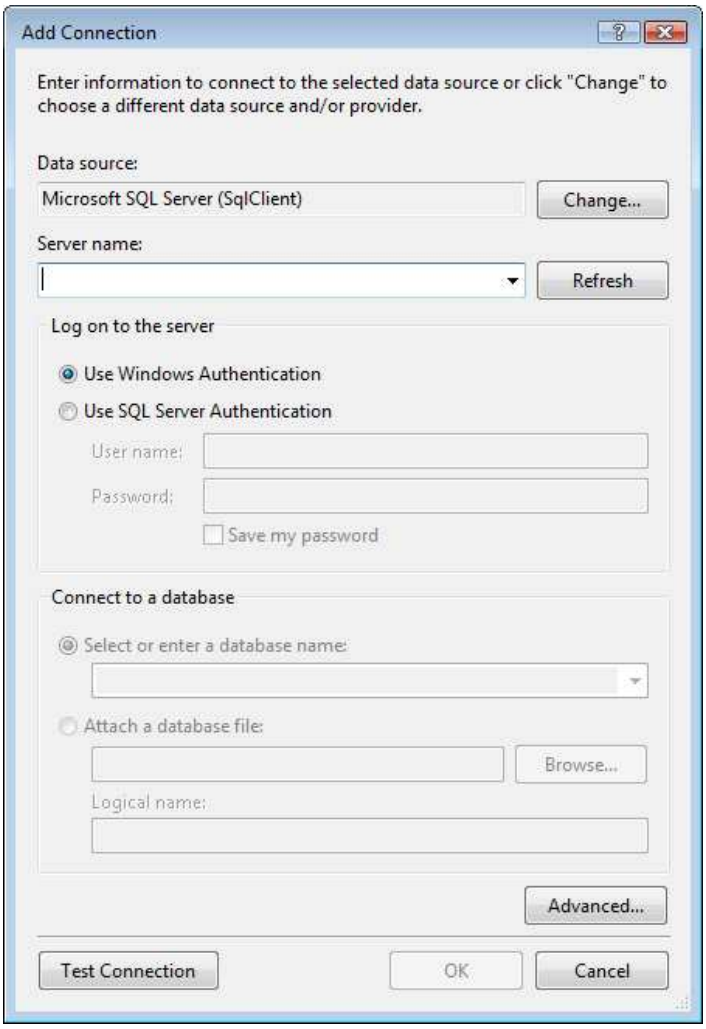
Valitse seuraavasta dialogista datalähteen tyyppi, joka tässä tapauksessa on Microsoft SQL Server ja paina OK (kuvio 14.).



Kuvio 14. Datalähteen valinta

Jos haluaa käyttää datalähteenä erillistä, aiemmin luotua tietokantatiedostoa, voi tästä valikosta valita kohdan Microsoft SQL Server Database File, ja painaa OK (kuvio 14.). Käytettäessä erillistä tiedostoa, datalähteen valintadialogia seuraavassa yhteydenluontidialogissa (*engl.* Add Connection) tulee navigoida oikea tiedosto Browse-painikkeella ja painaa OK-painiketta. Erillistä tietokantatiedostoa käytettäessä siirry suoraan kohtaan 3.

Valitse yhteyden lisäysdialogin alasvetovalikosta (kuvio 15.). palvelimen nimi (*engl.* server name) ja tämän jälkeen valitse tietokantaryhmän (*engl.* connect to a database) alasvetovalikosta aiemmin luotu Person-henkilötietokanta.



Kuvio 15. Add Connection

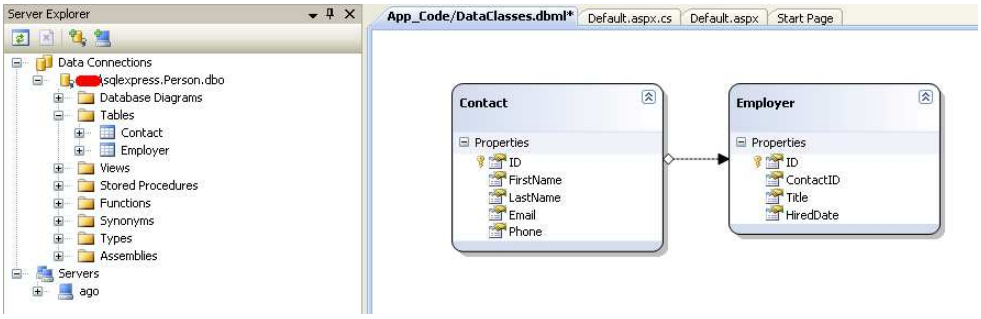
Jos valittu palvelin ei anna tietokantalistausta, kannattaa Management Studio -ohjelmistolla katsoa palvelimen tarkka nimi. Se on ohjelmiston Express-versiota käytettäessä muotoa <serverin nimi>\SQLEXPRESS. On mahdollista, että tämän joutuu kirjoittamaan manuaalisesti dialogin palvelin-alasvetovalikkoon (kuvio 15.).

Testaa vielä yhteys testauspainikkeella (*engl.* test connection) ja jos tulos on onnistunut, sulje ilmoitusikkuna OK-painikkeesta ja paina yhteydenluontidialogin OK-painiketta. Henkilötietokannan pitäisi nyt näkyä palvelinikkunassa.

3) Tietokantataulujen muuttaminen LINQ-entiteeteiksi

Avaa DataClasses.dbml-tiedosto projekti-ikkunasta ja henkilötietokanta palvelinikkunasta. Navigoi auki henkilötietokannan taulut-hakemisto. Raahaa sieltä molemmat taulut (kontakti, työntekijä) DataClasses.dbml -tiedoston tyhjään editorinäkymään (kuvio 16.) joko erikseen

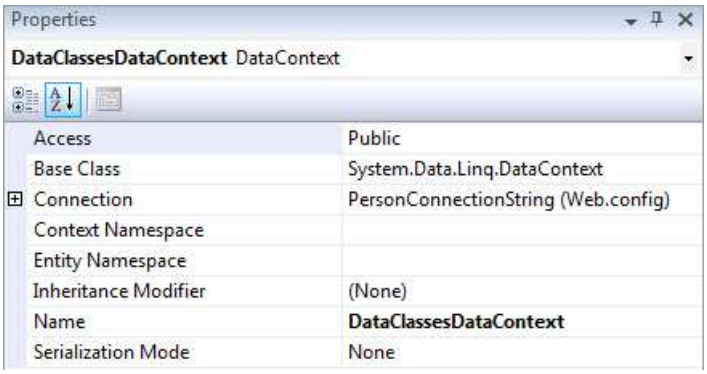
tai maalaamalla ne molemmat ja siirtämällä kerralla. SQL Serverissä luotu taulujen välinen liitos näkyy nyt oliomallieditorissa mustana nuolena.



Kuvio 16. Oliomallieditori ja palvelinselaimesta siihen siirretyt taulut

4) Entiteettien ominaisuudet

Klikkaa hiirellä jostakin oliomallieditorin valkoisesta taustasta, jolloin Visual Studion ominaisuusikkuna näyttää koko datakontekstin ominaisuudet (kuvio 17.). Tästä ominaisuusikkunasta kannattaa huomioida IDE:n luoma DataClassesDataContext -nimiominaisuus, koska sitä käytetään LINQ to SQL -kyselyissä. Vaihda nimeksi PersonDataContext.



Kuvio 17. DataClasses.dbml-tiedoston datakonteksti-ominaisuudet

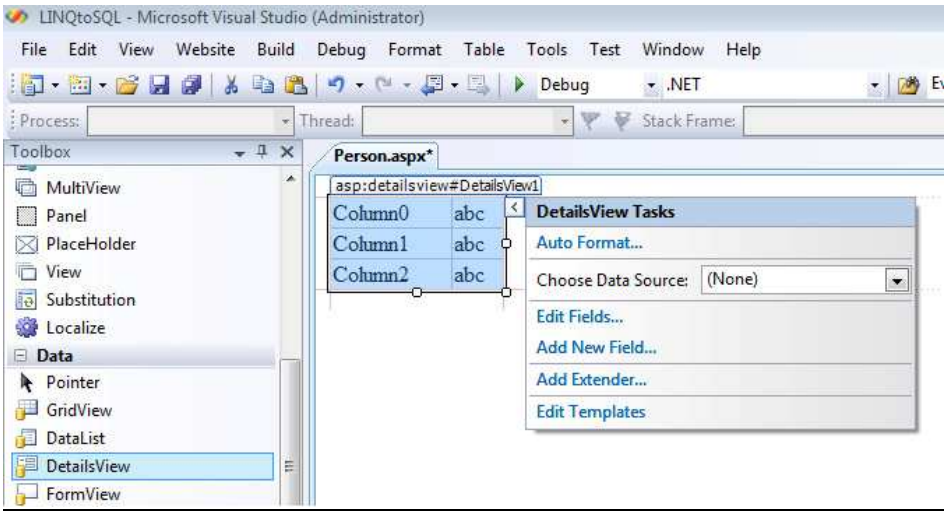
Klikkaa oliomallieditorin työntekijätaulun tunnistesaraketta (ID) ja muuta sen ominaisuuksista Auto Generated Value -ominaisuuden arvo muotoon tosi. Muuta myös ominaisuus Auto-Sync arvoon OnInsert. Tallenna dbml-tiedoston muutokset.

Käyttöliittymän laadinta ja sen liittäminen tietokantaan

1) Ilmentymänäkymän asentaminen sivulle

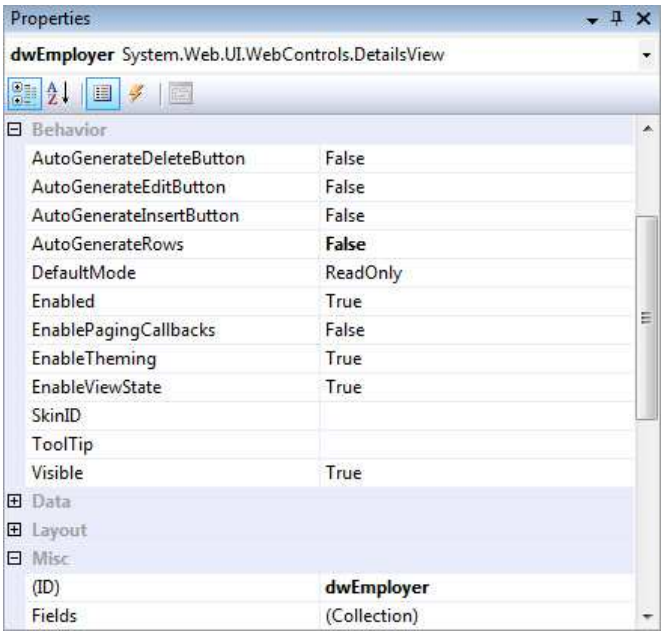
Luo projektiin **Person.aspx**-sivu uuden tiedoston lisäystoiminnon kautta. Tästä sivusta tulee pohja, jolla näytetään yksittäisen työntekijän tiedot ja jonka avulla tietokantaan myös lisätään uusi työntekijäilmentymä tietokantaan.

Raahaa työkaluikkunasta sivun **graafiseen** näkymään (Design) ilmentymänäkymä (kuvio 18.).



Kuvio 18. Ilmentymänäkymä ja sen tehtävälista

Jos ilmentymänäkymä ei ole aktiivisena, aktivoi se ja muuta sen tunniste ominaisuusikkunasta arvoon dwEmployer ja ominaisuus AutoGenerateRows arvoon epätosi (kuvio 19.).

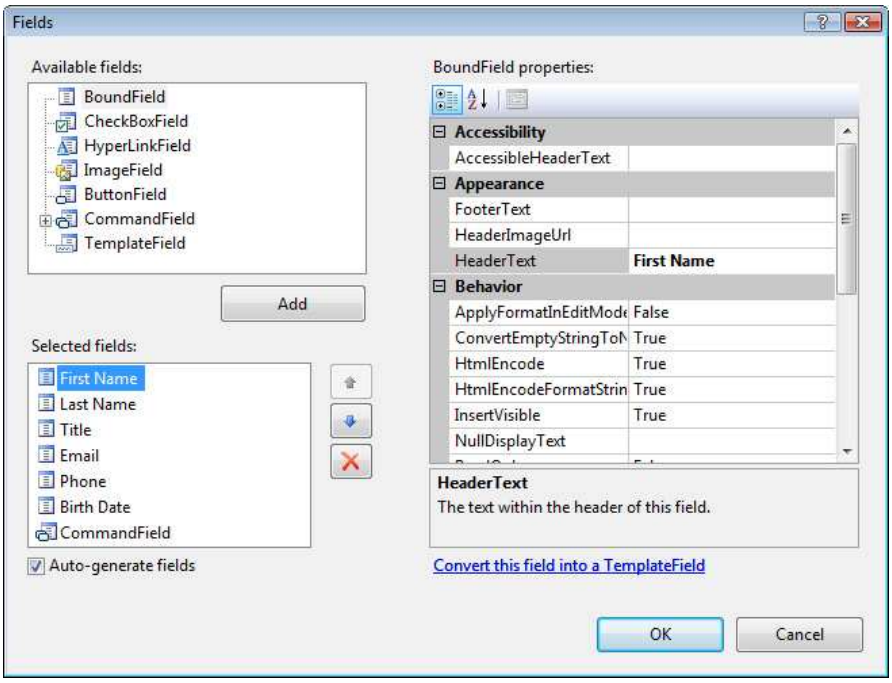


Kuvio 19. Ilmentymänäkymän ominaisuudet

Tämän jälkeen voit muuttaa kontrollin ulkoasua tehtäväpaneelin automatisoitujen grafiikkamallien avulla (Auto Format) (kuvio 18.). Jos Person.aspx-sivu ajetaan nyt selaimessa (projekti-ikkunassa paina hiiren oikeaa painiketta Person.aspx-tiedoston kohdalla ja valitse kohta 'katso selaimessa' (*engl.* view in browser)), huomataan että sivulla ei näy mitään, koska kontrollin datakenttiä ei ole määritetty.

2) Ilmentymänäkymän datakenttien määritys

Paina Ilmentymänäkymän tehtävälistan kohtaa 'kenttien muokkaus' (*engl.* edit fields) (kuvio 18.). Valitse dialogin kenttienvalintaikkunasta (*engl.* available fields) BoundField-tyyppinen kenttä ja paina kuusi kertaa lisäyspainiketta (kuvio 20.). Nyt kenttäikkunassa (*engl.* selected fields) on kuusi BoundField-nimistä datakenttää. Valitse kenttäikkunasta ylin datakenttä ja aseta dialogin oikeanpuoleisesta ominaisuus-ikkunasta HeaderText-ominaisuus arvoon 'First name' ja DataField-ominaisuus arvoon FirstName. Huomaa että kentän nimi muuttuu kenttäikkunassa.



Kuvio 20. Kenttäeditori

HeaderText-ominaisuus on kyseisen datakentän otsikko, joka näytetään sivulla ja DataField -ominaisuus on datalähteen nimi, joka sidotaan tähän kenttään myöhemmin tietokannasta. DataField-ominaisuuksien tulee nimetty identtisiksi niiden tietokantataulujen sarakkeiden kanssa, joita käytetään tässä kontrollissa tietolähteenä.

Jäljellä oleviin viiteen BoundField-kenttään asetetaan samat ominaisuudet seuraavilla arvoilla:

2. BoundField: Last name - LastName
3. BoundField: Title - Title
4. BoundField: Email - Email
5. BoundField: Phone - Phone
6. BoundField: Hired date - HiredDate

Lisää mukaan vielä yksi CommandField-tyyppinen kenttä. Muuta tai aseta sen ominaisuuksia seuraavasti:

1. ButtonType - Button
2. ShowDeleteButton - true
3. ShowEditButton - true

Hyväksy muutokset painamalla OK ja tallenna Person.aspx-tiedoston muutokset.

3) Ilmentymänäkymän toimintamoodin vaihtaminen

Kyseinen aspx-sivu näyttää selaimessa vieläkin tyhjältä, joten ennen kuin tietoja pääsee syöttämään tietokantaan, ilmentymänäkymän toimintamoodi tulee vaihtaa syöte -tyyppiseksi.

Avaa Person.aspx-tiedoston koodiluokka Person.aspx.cs. Kirjoita Page_Load-lohkon sisälle seuraava koodi:

```
dwEmployer.ChangeMode(DetailsViewMode.Insert);
dwEmployer.AutoGenerateInsertButton = true;
```

Tämä koodi muuttaa ilmentymänäkymän insert-tilaan, jolloin sitä voi käyttää datan syöttämiseen tietokantaan. Ilmentymänäkymä näkyy nyt selaimessa oikein, mutta syötä -painikkeen painaminen aiheuttaa virheen.

Lisää luokan alkuun lisäksi seuraavan esimerkin mukainen nimiavaruusesittely, koska luokassa tarvitaan myöhemmin listamuuttujia.

```
using System.Collections.Generic;
```

4) Syötemetodin määrittäminen

Avaa Person.aspx-sivu, aktivoi ilmentymänäkymä ja paina ominaisuusikkunasta tapahtumat -painiketta. Etsi tapahtumalistasta ItemInserting-rivi ja kaksoispainalla sitä, jolloin tiedosto Person.aspx.cs aktivoituu ja siihen on generoitunut seuraava koodilohko:

```
protected void dwEmployer_ItemInserting(object sender,
DetailsViewInsertEventArgs e)
{

}
```

Koodilohkon sisälle tulee laatia ohjelmakoodi, joka kerää käyttäjän syöttämät tiedon ilmentymänäkymästä ja tallettaa ne tietokantaan työntekijä- ja kontaktitauluihin. Kirjoita seuraava koodi `dwEmployer_ItemInserting` -tapahtuman sisälle:

```
using (PersonDataContext _dc = new PersonDataContext())
{
    //luodaan uudet kontakti- ja työntekijäoliot
    Contact _contact = new Contact();
    Employer _employer = new Employer();

//Ilmentymänäkymäm textbox-kontrollit muunnetaan tekstikentiksi ja
//sijoitetaan muuttujiin
TextBox txtFirstName =(TextBox)dwEmployer.Rows[0].Cells[1].Controls[0];
TextBox txtLastName = (TextBox)dwEmployer.Rows[1].Cells[1].Controls[0];
TextBox txtTitle = (TextBox)dwEmployer.Rows[2].Cells[1].Controls[0];
TextBox txtEmail = (TextBox)dwEmployer.Rows[3].Cells[1].Controls[0];
TextBox txtPhone = (TextBox)dwEmployer.Rows[4].Cells[1].Controls[0];
TextBox txtHiredDate =(TextBox)dwEmployer.Rows[5].Cells[1].Controls[0];

    //tekstikenttiin kirjoitettu sisältö talletetaan olioihin
    _contact.FirstName = txtFirstName.Text;
    _contact.LastName = txtLastName.Text;
    _contact.Email = txtEmail.Text;
    _contact.Phone = txtPhone.Text;
    _contact.ID = Guid.NewGuid();

    _employer.ContactID = _contact.ID;
    _employer.Title = txtTitle.Text;

    //tarkistetaan onko kirjoitettu Hired date -arvo oikeaa muotoa
    //ja talletetaan päivämäärä työntekijäolioon
    DateTime outValue;

    if (DateTime.TryParse(txtHiredDate.Text, out outValue))
    { _employer.HiredDate = outValue; }

    //talletetaan tiedot tietokantaan
    _dc.Contacts.InsertOnSubmit(_contact);
    _dc.Employers.InsertOnSubmit(_employer);

    _dc.SubmitChanges();
}

//ohjaa samalle sivulle uudelleen, jolloin tekstikentät tyhjenevät
//-> ei tarvitse muuttaa kaikkien kenttien arvoja manuaalisesti
//tyhjiksi
Response.Redirect("Person.aspx");
```

5) Testidatan syöttäminen tietokantaan

Käynnistä Person.aspx-sivu selaimessa ja kirjoita tekstikenttiin haluamasi testitiedot ja paina syötä-linkkiä. Tee muutamia koehenkilöitä, kuusi riittää. Tätä sivua ja sen ilmentymänäkymää käytetään myöhemmin yksittäisen henkilön tietojen tarkastelussa ja muokkaamisessa.

6) Henkilöiden listaaminen taulukkonäkymässä

Avaa default.aspx-sivu ja siirrä graafiseen suunnittelutilaan taulukkonäkymä. Muuta kontollin tunniste arvoon gwEmployee ja muuta kontrollin ulkoasua haluamaksesi, kuten ilmentymänäkymässä. Muuta ominaisuusikkunasta ominaisuus AllowPaging arvoon tosi ja ominaisuus PageSize arvoon 5. AllowPaging-ominaisuus luo kontrolliin automaattisesti sivutustoiminnon.

Avaa Default.aspx.cs-tiedosto ja uusi Using-määrite listamuuttujaa varten:

```
using System.Collections.Generic;
```

Seuraavaksi kirjoita `Page_Load`-lohkoon koodi, jossa haetaan kaikki kontaktitaulun ilmentymät ja sidotaan ne taulukkonäkymään.

```
using (PersonDataContext _dc = new PersonDataContext())
{
        var _contact = (from con in _dc.Contacts
                        select con).ToList();

        gwEmployee.DataSource = _contact;
        gwEmployee.DataBind();
}
```

Kun sivu ajetaan selaimessa, tuloksen pitäisi olla kuvion 21. kaltainen.

ID	ContactID	FirstName	LastName	Email	Phone
73e8fe6c-f9eb-452c-8acd-cd2014de6ca8	5592d564-f2be-43b3-9958-0c26983f151c	Paavo	Kinnunen	pk@jippii.fi	35623424376
bacbd5dd-f004-465c-bd6f-b50a97b626e9	b8457ff4-4c18-435d-ad27-3053e592a761	Pirjo	Hämäläinen	ert@hotmail.com	123123123
352705eb-aae0-438d-9c07-79a28cd336fd	67abadec-7867-4df8-95f3-36e12aa4d2b5	Mervi	Pekkarinen	mp@mbnet.fi	456456234
5da8b05b-5407-40d5-bfbe-061a88ef56dd	2991a510-65a7-4e48-905a-af05459a637b	Teppo	Heikkinen	test@hotmail.com	123234557
98d67f14-d90c-4799-b1cc-ad1c4cc0f3ff	b5fff90b-5031-4084-845a-b8b49c8e8f83	Tuomas	Kilponen	tk@hotmail.com	45612578
1 2					

Kuvio 21. Kontaktiolioiden listaus

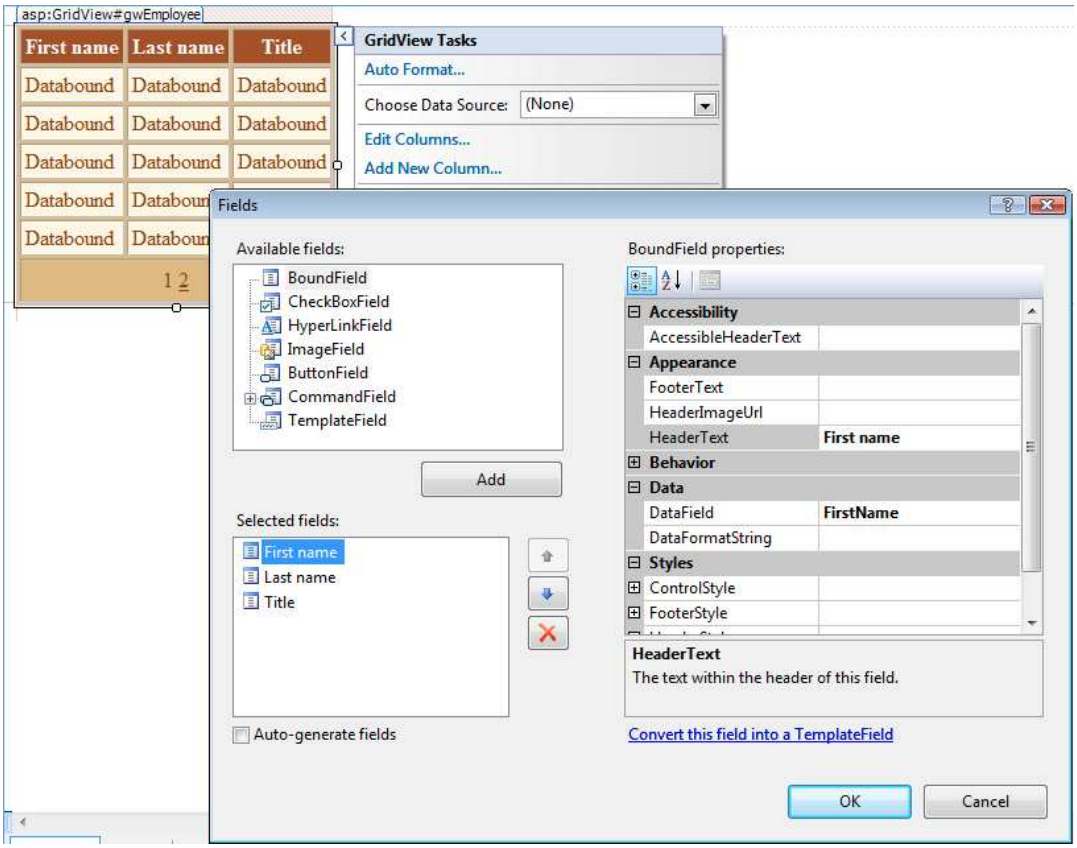
Tulos on luettavuudeltaan epäselvä, sillä tunnistekenttä (ID) on turha, eikä työntekijätietoja ole lainkaan. Muuta sivun koodiluokan Page_Load -lohkoa siten, että LINQ-kyselystä palautetaan oma tynkäolio, jolla on etunimi ja sukunimiominaisuudet kontaktitaulusta sekä titteliominaisuus työntekijätaulusta:

```
var _employee = (from emp in _dc.Employers
                select new { emp.Contact.FirstName,
                emp.Contact.LastName, emp.Title }
                ).ToList();

gwEmployee.DataSource = _employee;
gwEmployee.DataBind();
```

Nyt tuloste on luettavampi, mutta kuviosta 19. voidaan huomata, että taulukkonäkymän sarakkeiden otsikoiksi tulee automaattisesti nimet tietokantataulujen sarakkeista.

Tässä esimerkissä kontrollin otsikoiden kieliasu tuskin on olennaista, mutta kontrolli voidaan haluttaessa muuttaa siten, että omien otsikoiden käyttö on mahdollista. Tämä tapahtuu samalla tavalla kuten ilmentymänäkymässä, eli graafisessa suunnittelutilassa siirrytään kontrollin tehtäväpaneeliin ja lisätään sille sarakkeenmuokkausdialogissa (*engl.* edit columns) kolme BoundField-kenttää. Tämän jälkeen muutetaan samaan tapaan näiden kenttien otsikko- ja datakenttä-arvoja (HeaderText ja DataField) (kuvio 22.). Kun taulukkonäkymän ominaisuusikkunasta muutetaan ominaisuus AutoGenerateColumns arvoon epätosi, muutos kontrollille on valmis.

Huomaa että määritettäessä itse kontrollin käytössä olevat datakentät, niihin määritetyt tiedot on pakko olla kontrollin käytössä. Oletetaan, että edellisen esimerkin mukaisesti kontrollin datalähteeksi asetetaan joukko olioita, joilla on ominaisuudet FirstName, LastName ja Title. Jos kontrollille määritetään datakenttiä, joiden DataField-ominaisuus ei vastaa olion ominaisuuksia, on seurauksena virhe sivulla.



Kuvio 22. Taulukkonäkymä ja kenttäeditori

7) Sivutustapahtuman ohjelmointi

Kun sivua Default.aspx-ajetaan selaimessa, ja jos henkilöilmentymiä luotiin aikaisemmin tarpeeksi, sivutuslinkit ilmestyvät kontrollin alalaitaan. Sivutuslinkkien käyttö kuitenkin kaataa sivun.

Etsi kontrollin tapahtumalistasta rivi PageIndexChanging ja kaksoispainalla sitä. Kirjoita avautuvaan koodilohkoon seuraavan esimerkin koodi ja talleta muutokset.

```
gwEmployee.PageIndex = e.NewPageIndex;
gwEmployee.DataBind();
```

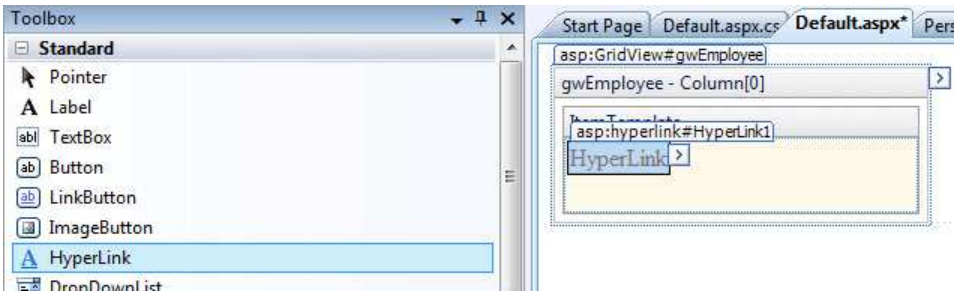
8) Yksittäisen henkilön selaus (Default.aspx)

Seuraavaksi taulukkonäkymään luodaan uusi sarake, joka sisältää hyperlinkin (kuvio 23.). Tämä hyperlinkki johtaa Person.aspx-sivulle, jossa näytetään henkilön tarkemmat tiedot.

	First name	Last name	Title
Info	Paavo	Kinnunen	ohjelmoija
Info	Pirjo	Hämäläinen	projektipäällikkö
Info	Tuomas	Kilponen	Toimitusjohtaja
Info	Pekka	Petelius	ohjelmoija
Info	Teppo	Heikkinen	ohjelmoija
1 2			

Kuvio 23. Listaus ja Info-kenttä

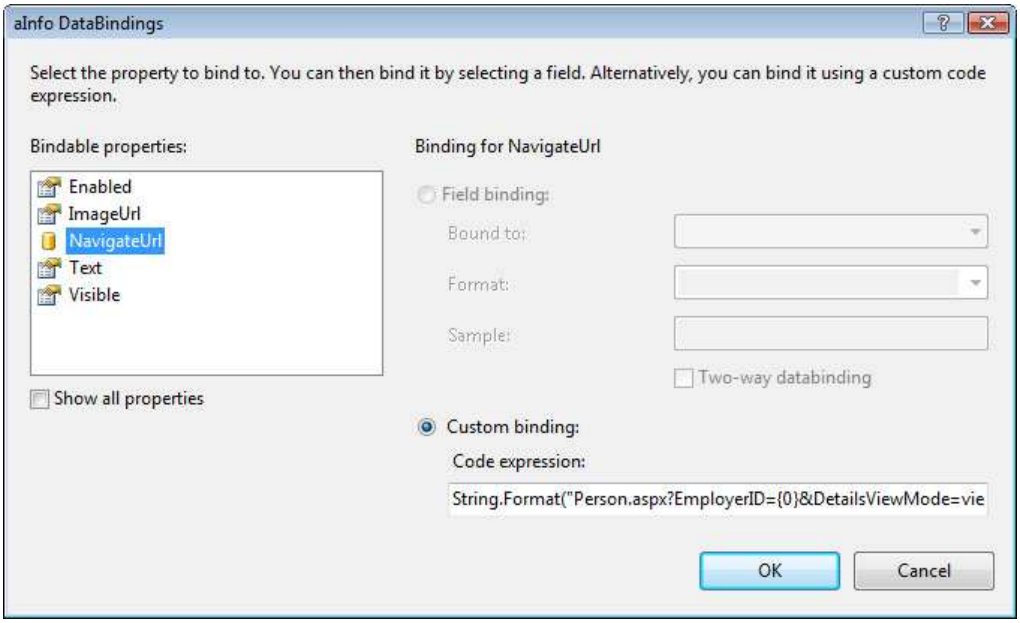
Avaa Default.aspx-sivu ja lisää taulukkonäkymän tehtävälistan sarake-editorin kautta (*engl.* edit columns) kontrollille uusi TemplateField-kenttä. Siirrä luotu kenttä pinon päällimmäiseksi nuoli-painikkeita käyttäen. Hyväksy kenttämuutokset painamalla OK. Navigoi uudelleen kontrollin tehtävälistaan ja paina siitä mallieditori-linkkiä (*engl.* edit templates). Siirrä graafisessa näkymässä hyperlinkki ItemTemplate-ikkunaan (kuvio 24.).



Kuvio 24. Itemtemplate-ikkuna ja hyperlinkki

Kun hyperlinkki on aktiivisena, muuta sen ominaisuuksista tunniste-ominaisuus arvoon aInfo ja linkin teksti (*engl.* text) arvoon Info. Avaa hyperlinkin tehtävälista ja paina datansidonta-linkkiä (*engl.* edit databindings). Valitse dialogin (kuvio 25.) vasemmasta ikkunasta ominaisuus NavigateUrl ja kirjoita koodinmäärityskenttään (*engl.* code expression) seuraava määritys:

String.Format("Person.aspx?EmployerID={0}&DetailsViewMode=view", Eval("ID"))



Kuvio 25. Datansidontaeditori

Code expression -määrityksessä hyperlinkin osoitteeksi määritetään Person.aspx-sivu, mutta sille annetaan myös osoiteriviparametrejä (EmployerID ja DetailsViewMode). Nämä parametrit kertovat Person.aspx-sivulle, mikä työntekijä sen tulee tietokannasta näyttää ja mitä moodia ilmentymänäkymässä käytetään (selaus, muokkaus vai syöttö).

Paina OK, ota esille GridView-kontrollin tehtävälista ja paina linkkiä End Template Editing. Tallenna muutokset.

Koska hyperlinkki tarvitsee nyt työntekijäolion ominaisuutta ID, täytyy LINQ-kyselyssä luotavaan tynkäolioon lisätä kyseinen ominaisuus. Muokkaa Default.aspx.cs-tiedoston Page_Load-lohkossa oleva LINQ-kysely seuraavanlaiseksi:

```
var _employee = (from emp in _dc.Employers
                 select new { emp.Contact.FirstName,
                 emp.Contact.LastName, emp.Title, emp.ID }).ToList();
```

Jos sivu default.aspx-sivu ajetaan selaimessa, painettaessa Info-linkkiä, siirrytään henkilö -sivulle, jonka osoiterivillä oleva osoite on muotoa (sivulla on tyhjä datan syöttönäkymä):

Person.aspx?EmployerID=b072d47a-3ede-4a44-80d8-329ed1f26156&DetailsViewMode=view

9) Henkilö-luokan luominen

Tästä eteenpäin on järkevää käyttää LINQ-kyselyiden yhteydessä var-muuttujien sijasta listamuuttujia, joka sisältää vahvasti tyypitettyjä olioita. Listakokoelmaa voi muiden perinteisten muuttujatyyppien lisäksi palauttaa metodeista ja luokista, toisin kuin var -muuttujaa.

Luo projektiin uusi .cs-tiedosto nimeltä Persons.cs. Kirjoita tiedoston juureen uusi luokkamääritys get- ja set-metodeineen ja `System.Collections.Generic` -kirjastoesityksineen, jolloin koko luokka näyttää seuraavalta:

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Collections.Generic;

public class CustomPersonItem
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Title { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public DateTime? HiredDate { get; set; }
}

public class Persons
{
      public Persons()
      {
      }
}
```

Pyyhi konstruktori **Persons** pois ja kirjoita sen tilalle seuraava metodi, joka annetun työntekijätunnisteen perusteella hakee kaikki työntekijäilmentymät ja muodostaa niistä CustomPersonItem-tyyppisen oliolistan. Seuraavassa esimerkissä esitetään Person-luokka kokonaisuudessaan tämän metodin kanssa.

```
public class Persons
{
    /// <summary>
    /// Palauttaa listan henkilö-olioita annetun EmployerID:n
    /// perusteella
    /// </summary>
    public static List<CustomPersonItem> GetPerson(string sGuid)
    {
      List<CustomPersonItem> PersonItem = new List<CustomPersonItem>();

        using (PersonDataContext _dc = new PersonDataContext())
        {
            PersonItem = (from emp in _dc.Employers
                          where emp.ID == new Guid(sGuid)
                          select new CustomPersonItem
                          {
                              FirstName = emp.Contact.FirstName,
                              LastName = emp.Contact.LastName,
                              Email = emp.Contact.Email,
                              Phone = emp.Contact.Phone,
                              Title = emp.Title,
                              HiredDate = emp.HiredDate
                          }).ToList();
        }

        return PersonItem;
    }
}
```

10) Yksittäisen henkilön selaus (Person.aspx)

Tallenna muutokset, avaa Person.aspx.cs-sivu ja kirjoita Page-Load -lohkon alkuun seuraavat muuttujamääritteet:

```
string sMode = Request.QueryString["DetailsViewMode"];
string sEmployerID = Request.QueryString["EmployerID"];
```

Nämä rivit poimivat osoiteriviltä DetailsViewMode- ja EmployerID-parametrit, jotka talletetaan muuttujiin.

Tee sivulle uusi metodi GetSource (`dwEmployer_ItemInserting`-metodin alapuolelle), joka ottaa vastaan osoiterivin työntekijätunnisteen ja palauttaa uuden Persons-luokan GetPerson-metodia käyttäen listan CustomPersonItem-olioita:

```
protected List<CustomPersonItem> GetSource(string sEmployerID)
{
    if (!String.IsNullOrEmpty(sEmployerID))
    {
        return Persons.GetPerson(sEmployerID);
    }

    return null;
}
```

Poista Page_Load-lohkon kontrollin esitystilan vahtavat rivit, ja lisää niiden tilalle (`Request.QueryString`-rivien alapuolelle) seuraava switch-case -ehtorakenne:

```
switch (sMode)
{
    //jos DetailsViewMode on ”view”, näytetään käyttäjälle selausnäkymä
    //jos mode on ”insert”, näytetään datansyöttönäkymä
    //jos mode ei ole kumpikaan, ohjataan käyttäjä default.aspx-sivulle
    case "view":
        if (!IsPostBack)
        {
            dwEmployer.DataSource = GetSource(sEmployerID);
            dwEmployer.DataBind();
        }
        break;
    case "insert":
        if (!IsPostBack)
        {
            dwEmployer.ChangeMode(DetailsViewMode.Insert);
            dwEmployer.AutoGenerateInsertButton = true;
        }
        break;
    default:
        if (!IsPostBack)
        {
            Response.Redirect("default.aspx");
        }
        break;
}
```

11) Yksittäisen henkilön poistaminen

Avaa **Persons.cs**-tiedosto ja tee sinne uusi Delete-niminen metodi, joka ottaa vastaan työntekijätunnisteen ja tuhoaa sitä vastaavan kontakti- ja työntekijärivin:

```
public static void Delete(string sEmployerID)
{
    Employer _employer = new Employer();
    Contact _contact = new Contact();

    using (PersonDataContext _dc = new PersonDataContext())
    {
        _employer = (from emp in _dc.Employers
                     where emp.ID == new Guid(sEmployerID)
                     select emp).FirstOrDefault();

        // asetetaan datakonteksteille delete-käsky, joka toteutetaan
        // kun SubmitChanges()-metodi ajetaan
        _dc.Contacts.DeleteOnSubmit(_employer.Contact);
        _dc.Employers.DeleteOnSubmit(_employer);

        _dc.SubmitChanges();
    }
}
```

Avaa Person.aspx, valitse taulukkonäkymä aktiiviseksi ja kaksoispainalla sen ominaisuusikkunan tapahtumalistasta riviä ItemDeleting. Kirjoita avautuvaan koodilohkoon seuraavaa:

```
string sEmployerID = Request.QueryString["EmployerID"];

if (!String.IsNullOrEmpty(sEmployerID))
{
    Persons.Delete(sEmployerID);
    Response.Redirect("default.aspx");
}
```

Nyt delete-painikkeen pitäisi toimia katseltaessa yksittäisen henkilön tietoja.

12) Yksittäisen henkilön tietojen muokkaus (Persons.cs)

Avaa Persons.cs-tiedosto ja tee siinne uusi Update-niminen metodi, joka ottaa vastaan useita muuttujia:

```
public static void Update(string sEmployerID, string sFirstName,
                          string sLastName, string sTitle,
                          string sEmail, string sPhone, string
                          sHiredDate )
{
    DateTime dt;

    using (PersonDataContext _dc = new PersonDataContext())
    {
        Employer _employer = (from emp in _dc.Employers
                              where emp.ID == new Guid(sEmployerID)
                              select emp).SingleOrDefault();

        _employer.Contact.FirstName = sFirstName;
        _employer.Contact.LastName = sLastName;
        _employer.Contact.Phone = sPhone;
        _employer.Contact.Email = sEmail;
        _employer.Title = sTitle;

        //käytetään datetime-tyypin TryParse-metodia, joka tarkistaa
        //onko sHiredDate-string oikeaa muotoa ja tallettaa sen dt-
        //nimiseen datetime-muuttujaan
        //jos syötetty aika oli oikeaa muotoa, se talletetaan olioon
        if (DateTime.TryParse(sHiredDate, out dt))
            _employer.HiredDate = dt;

        _dc.SubmitChanges();
    }
}
```

Tallenna muutokset, avaa Persons.aspx, valitse ilmentymänäkymä ja kaksoispainalla tapahtumalistan ModeChanging-riviä. Kirjoita koodilohkoon seuraava ohjelmakoodi, ja tallenna muutokset:

```
string sEmployerID = Request.QueryString["EmployerID"];

dwEmployer.ChangeMode(e.NewMode);
dwEmployer.DataSource = GetSource(sEmployerID);
dwEmployer.DataBind();
```

13) Yksittäisen henkilön tietojen muokkaus (Person.aspx.cs)

Avaa Person.aspx ja tuplapainalla ilmentymänäkymän ominaisuusikkunasta ItemUpdating -tapahtumaa. Kirjoita koodilohkoon:

```
string sEmployerID = Request.QueryString["EmployerID"];

if (!String.IsNullOrEmpty(sEmployerID))
{
    // olioille syötetyt tiedot haetaan manuaalisesti
    // ilmentymänäkymästä (joka on taulukko) tekstikenttä-muuttujiin
    TextBox txtFirstName =
           (TextBox)dwEmployer.Rows[0].Cells[1].Controls[0];
    TextBox txtLastName =
           (TextBox)dwEmployer.Rows[1].Cells[1].Controls[0];
    TextBox txtTitle =
           (TextBox)dwEmployer.Rows[2].Cells[1].Controls[0];
    TextBox txtEmail =
           (TextBox)dwEmployer.Rows[3].Cells[1].Controls[0];
    TextBox txtPhone =
           (TextBox)dwEmployer.Rows[4].Cells[1].Controls[0];
    TextBox txtHiredDate =
           (TextBox)dwEmployer.Rows[5].Cells[1].Controls[0];

    Persons.Update(sEmployerID, txtFirstName.Text, txtLastName.Text,
                   txtTitle.Text, txtEmail.Text, txtPhone.Text,
                   txtHiredDate.Text);
}

Response.Redirect("default.aspx");
```

14) Hyperlinkki henkilön lisäämiseen

Nyt tässä ohjelmassa tulisi olla kunnossa henkilöiden selaus, tarkastelu, lisäys, muokkaus ja poisto. Ainoa ominaisuus joka puuttuu, on linkki default.aspx-sivulla henkilön lisäykseen Person.aspx-sivulla.

Avaa default.aspx ja raahaa taulukkonäkymän alapuolelle hyperlinkki. Muuta hyperlinkin teksti-ominaisuus arvoon Add Employer ja aseta NavigateUrl-ominaisuuden sisällöksi seuraava rivi:

Person.aspx?DetailsViewMode=insert

15) Henkilöiden etsimistoiminto

Lisää default.aspx-sivulle lisäyslinkin alapuolelle tekstikenttä- ja painike-kontrollit (kuvio 26.). Muuta niiden tunnisteet esimerkiksi arvoiksi txtSearch ja btnSearch. Kaksoispainalla painiketta ja kirjoita avautuvaan koodilohkoon seuraava LINQ-kysely, jossa txtSearch -tekstikentän merkkijonoja etsitään tietokannan kontaktitaulun FirstName- ja LastName -sarakkeista sekä työntekijätaulun Title-sarakkeesta:

```
string sSearch = txtSearch.Text;

using (PersonDataContext _dc = new PersonDataContext())
{
    var _employee = (from emp in _dc.Employers

                     where
                     emp.Contact.FirstName.Contains(sSearch) ||
                     emp.Contact.LastName.Contains(sSearch) ||
                     emp.Title.Contains(sSearch)

                     select new { emp.Contact.FirstName,
                     emp.Contact.LastName, emp.Title, emp.ID }).ToList();

    gwEmployee.DataSource = _employee;
    gwEmployee.DataBind();

}
```

	First name	Last name	Title
Info	Databound	Databound	Databound
Info	Databound	Databound	Databound
Info	Databound	Databound	Databound
Info	Databound	Databound	Databound
Info	Databound	Databound	Databound
1 2			

Add Employer

Search

Kuvio 26. TextBox- ja Button-kontrollit etsimistoimintoa varten

6 ASP.NET JA OHJELMOIJAN TIETOTURVA

Nykyisin sivustojen tietoturvallisuudesta huolehtiminen kuuluu myös ensimmäisen tason arkkitehtuurisuunnittelijoille, ei pelkästään ohjelmoijille. Suurin osa web-sovelluksista käsittelee arkaluonteisia tietoja, joita pitää luojata suorilta hyökkäyksiltä. Kun suunnitellaan sovelluksen pakollista tietoturvamallia, täytyy olla tietoinen mitä tietoturvavaatimuksia sovelluksen täytyy täyttää ja mikä vaikutus valitulla mallilla on sovelluksen suorituskykyyn, skaalautuvuuteen ja käyttöönottoon. (Microsoft 2008 f.)

.NET Framework ja sen tietoturvakontrollit antavat työkalut niin käyttäjien tunnistamiseen kuin heidän käyttöoikeuksiensa määrittämiseen ja toimintojen tarkkailuun. Lisäksi käytössä on luokat tietoturvan ja tiedon eheyden varmistamiseen kryptausmenetelmillä sekä digitaalisilla allekirjoituksilla. (MacDonald 2007, 827)

ASP.NET 2.0, joka toimii edelleen pohjana ASP.NET 3.5:lle, lisäsi ASP-tekniikoiden tietoturvaa merkittävästi lisäämällä siihen korkeatasoisemmat työkalut käyttäjienhallintaan ja roolien määrittämiseen, sekä ohjelmallisesti että valmiiden työkalujen avulla. Nämä toiminnallisuudet rakentavat yhdessä turvamallin, jonka runko on ollut olemassa jo ASP 1.0 -versiosta saakka. ASP.NET 3.5 ulottaa tämän tietoturvamallin viimein AJAX-tekniikoihin. (MacDonald 2007, 827)

Vaikka potentiaalisten uhkien kartoittaminen ja mallintaminen on ohjelmistosuunnittelussa tärkeää, siihen ei varsinaisesti tässä dokumentissa puututa. Tämä kappale keskittyy siihen, mitä tietoturvallista vastuuta ohjelmoijalla on omasta koodistaan. (MacDonald 2007, 827.)

Ohjelmoijan tulee omatoimisesti täyttää vähintään seuraavat ohjelmointikriteerit (MacDonald 2007, 829):

1) Käyttäjien syötteisiin ei tule luottaa

Suositus on vapaasti olettaa että kaikki käyttäjät selaavat sivuja tehdäkseen rikoksia. Kaikki syöte on tarkistettava esimerkiksi validointikontrollien avulla ja jos mahdollista, sallittava käytettäväksi vain tiettyjä merkkejä.

2) Tietokantakutsuja ei tule laatia string-muodossa

Kaikki tietokantakutsun osat on parametroitava SQL-skriptihyökkäyksen varalta. LINQ -tekniikoita käytettäessä parametrointi tapahtuu automaattisesti.

Suojaamaton tietokantakutsu toteutetaan ADO.NET-ympäristössä seuraavan esimerkin mukaisesti. Muuttuja inputTitle on merkkijono-tyyppinen ja sen sisältö otetaan jostakin sivun tekstikentästä.

```
SqlCommand cmd = new SqlCommand(
"select * from Customers where Title = '" + inputTitle + "'");
```

Jos käyttäjä antaa syötteenä esimerkiksi seuraavan esimerkin mukaisen skriptin, poistetaan koko Employers-taulu tietokannasta.

```
a';DROP TABLE Employers;
```

SQL-Serveriin välittyvä SQL-kysely olisi silloin muotoa:

```
Select * From Employers Where Title = 'a';DROP TABLE Employers;
```

Parametrointi suoritetaan samaan esimerkkiin seuraavasti:

```
// määritetään command-objekti parametreineen
SqlCommand cmd = new SqlCommand(
                "select * from Employers where Title = @Title", conn);

// määritetään command-objektissa käytettävät parametrit
SqlParam param      = new SqlParameter();
param.ParameterName = "@Title";
param.Value         = inputTitle;
```

Nyt SQL-kyselyyn voi huoletta yrittää laittaa scriptikoodia, sillä parametroinnin ansiosta niitä ei koskaan ajeta SQL-Serverissä.

3) Käyttäjän syötteitä ei sijoiteta koskaan suoraan sivulle

Syötteet tulee validoida ja enkoodata aina ennen niiden sijoittamista sivulle. Käyttäjä voi syöttää esimerkiksi palasia HTML-koodia tai skriptejä, jotka selain tulkkaa esitettäväksi ja ajettavaksi koodiksi. Siksi syötteitä käsitellessä kannataa käyttää HttpUtility-nimiavaruuden HtmlEncode()-metodia, jotka muuttaa esimerkiksi skripteissä tarvittavat < ja > -merkit < ja > -merkkijonoiksi. HtmlDecode()-metodi tekee saman käänteisessä järjestyksessä:

```
string sInput = "<script type='text/javascript'>";
string sEncodedInput = HttpUtility.HtmlEncode(sInput);
```

Edellisen esimerkin mukaisessa koodissa `sEncodedInput`-muuttujaan tallettuu seuraava merkkijono:

<script type='text/javascript'>

Tästä voidaan nähdä, että HtmlEncode ei käännä esimerkiksi '-merkkejä, jolloin ne kannattaa esimerkiksi itse poistaa syötteestä kokonaan, varsinkin jos tätä syötettä ollaan tallettamassa tietokantaan.

4) Arkaluontoista dataa ei talleteta väliaikaisesti sivun kontrolleihin

Hidden field ja esimerkiksi tavallisen tekstikontrollien sisältö näkyy lähdekoodissa vaikka ne olisivatkin piilotettuina. Kontrollin tyyppiä on helppo muokata tallettamalla ensin sivu omalle koneelle. Kun kontrollin sisältöä ja tyyppiä on muokattu, sivu voidaan ajaa paikallisesti. Jos sivuston tietoturvallisuus on laadittu huonosti, muokattu sivu ajetaan palvelimella kuten mikä tahansa sivuston sivu, ja tietoturvallisuus on riskeerattu.

5) Sivutila-tyyppisiin muuttujiin ei talleteta arkaluontoista tietoa

Sivutila on käyttäjän laitteistoon talletettava tieto, jonka heikko salaus voidaan purkaa helposti. Jos sivuilla käytetään asetusta EnableViewStateMAC=true, ViewState suojataan avaimella, joka talletetaan vain palvelimelle ja jonka periaatteessa vain se voi avata.

Näiden ehdottomien ohjelmointilinjojen lisäksi taustalle tarvitaan suurempia kokonaisuuksia, joiden toteutus tulee tiedostaa koko yrityksessä. Näitä ovat jo edellä mainitut käyttäjien tunnistaminen ja hallinta, heidän käyttöoikeuksiensa rajaaminen, jäsenyyksien (*engl.* membership), roolien ja profiilien käyttö sekä SSL-salaustekniikoiden ja -sertifikaattien käyttö.

ASP.NET -pohjaisten Web-sivustojen tietoturvaan kuuluu tärkeänä osana myös palvelinten tietoturva- ja IIS-asetusten tarkka määritys.

7 POHDINTA

Microsoft on viime vuosina ison ohjelmistojätin elein puskenut markkinoille suuren määrän uutta ja monen mielestä mullistavaa ohjelmointiteknologiaa. Visual Studio 2008 ja .NET Framework 3.0 ja 3.5 ovat asettaneet niin suuret odotukset, että tuntuu mahdottomalta että niitä voidaan seuraavilla versioilla täyttää. Enää eivät pelkät parannetut ominaisuudet ja poistetut virheet riitä, vaan tarjolle on saatava sellaisia uusia kokonaisuuksia, jotka houkuttelisivat ohjelmoijia ja ohjelmointiyrityksiä toisilta ohjelmointialustoilta. Tämä ei kustannussyistä ja usein vahvojen ennakkoluulojen takia ole koskaan helppoa, koska web -kehitykseen on tarjolla useita, käytännössä samoja palveluja tarjoavia ohjelmistoalustoja. Visual Studion vahvuutena on sen todella laajat ohjelmointimahdollisuudet, kuten web-, mobiili- ja client-server -sovellukset sekä Windows- ja Office-ohjelmointi. Silti, kun tarkastellaan puhtaasti www-sivustojen kehittämistä, kilpailu on vilkasta.

Microsoft on Visual Studiossa ja sen kehityksessä pyrkinyt pääasiassa vahvistamaan omia teknologioitaan ja luomaan uusia jo olemassa olevien ympärille, lähinnä niiden tueksi. Yksi Microsoftin yrityksistä tunkeutua toisten hallitsemille markkinoille oli vuonna 2007 julkistettu SilverLight, jonka luvattiin olevan 'Flash-killer'. SilverLight on erinomaisen joustava ja .NET -ohjelmointiympäristössä nopea tapa toteuttaa Flash-tyyppistä toiminnallisuutta. Silti normaalissa Internet-selailussa SilverLight-tekniikoihin törmää vain harvoin. Suurin syy tähän ovat sekä kilpailun vähäisyys, että Flashin voimakas etulyöntiasema sen jo ollessa standardoitu osa lähes kaikkia Internet-selaimia. Lisäksi esimerkiksi FireFox -selaimessa SilverLightin asennus on itselläni tuottanut suuria ongelmia.

Jos ohjelmointialustan vaihtaminen Visual Studioon on kyseenailainen prosessi, Visual Studion vaihtaminen uudempaan ei ole. Vaikka siinäkin on varauduttava kustannuksiin, on siihen siirtymättömyys suurempi haitta kuin pakollinen muutos. VS 2008 sisältää niin paljon kustannustehokkuutta nostavia työkaluja, että tuskin on monia .NET-ympäristöön liittyviä ohjelmointialoja, joille näistä päivityksistä ei olisi apua. Siksi on mielestäni tärkeää, että kouluissa Visual Studion uusien versioiden opettamiseen panostetaan myös tulevaisuudessa.

Varsinainen opinnäytetyöprosessi eteni mielestäni huomattavasti hitaammin kuin olin kuvitellut. Suurin syy tähän oli se, että en työn aikataulutuksessa huomioinut lainkaan varsinaisen kirjoitustyön ulkopuolisia toimenpiteitä, kuten työn kirjoitusasun tarkistamista ja korjaamista. Työn ohjelmointioppaan ohjelmointiyö valmistui odotetusti parissa tunnissa,

mutta itse oppaan kirjoittamiseen kului melkein kaksi viikkoa, johon piti alunperin kulua korkeintaan muutama päivä. Aikatauluongelmia selittää osaksi päivätyön kireä tahti, jolloin energiaa työn kirjoittamiseen ei aina tuntunut riittävän. Työn laajuudesta huolimatta en missään vaiheessa ajatellut työn olevan liian iso tai haastava. Halusin tästä työstä jäävän jotain konkreettista hyötyä myös muille kuin itselleni.

Työn tekemistä helpotti suuresti se, että työn aihe ja siinä käytetyt ohjelmointitekniikat olivat itselleni tuttuja. Siksi tästä syystä työstä tuli ehkä laajempi, kuin aluksi suunniteltiin. Nyt työn valmistuttua, on tuntunut, että aiheet SilverLight tai WPF olisivat ehkä olleet mielenkiintoisempia, mutta niistä saatava hyöty muille olisi todennäköisesti ollut vähäisempi. ASP.NET 3.5:n merkitys web-kehityksessä on varmasti suurempi kuin SilverLightin, joka ei ole juuri saanut nostetta alleen. Toisaalta, oppimisprosessi olisi ollut näissä aiheissa syvällisempi, erityisesti WPF:n tapauksessa, johon olen perehtynyt vain C#-kurssin harjoitustyön kautta. Opin kuitenkin ASP.NET-ympäristöstä uusia asioita, erityisesti LINQ -tekniikoista, ja asiakirjoittamisen tasoni nousi merkittävästi, kuten varmasti kaikilla opinnäytetyötä tekevillä.

Työn tavoitteet, ASP.NET-tekniikoiden ja Visual Studio -ohjelmistoalustan esittely, toteutui työssä odotetulla tavalla, vaikka Visual Studion osuus jäikin oletettua pienemmäksi. Lisäksi, koska ohjelmointiopas lisättiin työhön jossain määrin jälkikäteen, on käytännön osuuden määrä työssä huomattava. Silti mielestäni ohjaavan opettajan ideoima ohjelmointiopas täydensi työtä merkittävästi.

Aikatauluongelmista huolimatta työn aihe jaksoi kiinnostaa loppuun asti ja toivon, että tästä työstä olisi apua mahdollisimman monille opiskelijoille ja että mahdollisimman moni saisi innostuksen web-ohjelmointiin.

LÄHDELUETTELO

Baker, Bertocci, Serack. 2007. Understanding Windows Cardspace. First Printing. ISBN: 978-0-321-49684-3.

Burnett Mark, Foster James. 2004. Hacking the Code: ASP.NET Web Application Security. Syngress Publishing. ISBN 1932266658.

Cameron Rob, Michalk Dale. 2008. Pro ASP.NET 3.5 Server Controls and AJAX Components. Apress. ISBN 978-1-59059-865-8.

Cox Ken. 2008. ASP.NET 3.5 For Dummies. 2008. Wiley Publishing. ISBN 978-0-470-19592-5.

Eichert Steve, Marguerie Fabrice, Wooley Jim. 2008. LINQ in Action. Manning Publications Co. ISBN: 1-933988-16-9.

Exforsys Inc. 2006. Building Web Based N-Tier Applications using C#. Web-dokumentti. Saatavilla: http://www.exforsys.com/tutorials/csharp/building-web-based-n-tier-applications-using-csharp.html (Luettu 5.12.2008).

Ferracchiati Fabio Claudio. 2008. LINQ for Visual C# 2008. Apress. ISBN 978-1-4302-1580-6.

Freeman Adams, Jones Allen. 2004. Programming .NET Security. O'Reilly Media. ISBN: 0-596-00442-7.

Guthrie Scott. 2005. Tips for Nested Master Pages and VS 2005 Design-Time. Web-dokumentti. Saatavilla: http://weblogs.asp.net/scottgu/archive/2005/11/11/430382.aspx (Luettu 20.11.2008).

Guthrie Scott 2008 a. VS 2008 JavaScript Debugging. Web-dokumentti. Saatavilla: http://weblogs.asp.net/scottgu/archive/2007/07/19/vs-2008-javascript-debugging.aspx (Luettu 20.11.2008).

Guthrie Scott 2008 b. Using LINQ with ASP.NET in VS "Orcas" (Part 1). Video. Saatavilla: http://www.scottgu.com/blogposts/video/linqtalk1.wmv (Katsottu 9.1.2009).

MacDonald Matthew, Szpuszta Mario. 2007. Pro ASP.NET 3.5 in C# 2008. Apress. ISBN: 978-1-59059-893-1.

Mahmoodi Rahman. 2005. 3-Tier architecture in C#. Web-dokumentti. Saatavilla: http://www.codeproject.com/KB/architecture/three_tier_architecture.aspx (Luettu: 3.12.2008).

Microsoft. 2003. An Overview of Managed/Unmanaged Code Interoperability. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/enus/library/ms973872.aspx (Luettu: 21.11.2008).

Microsoft 2008 a. .NET Framework Conceptual Overview. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx (Luettu: 21.11.2008)

Microsoft 2008 b. XNA Frequently Asked Questions. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/xna/aa937793.aspx (Luettu: 22.11.2008).

Microsoft 2008 d. What's New int the Visual Studio Debugger. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/01xdt7cs.aspx (Luettu: 23.11.2008).

Microsoft 2008 e. Debugging a Multithreaded Application. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/bb157784.aspx (Luettu: 26.11.2008).

Microsoft 2008 f. Autenthication in ASP.NET: .NET Security Guidance. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/ms978378.aspx (Luettu 9.12.2008).

Microsoft 2008 g. Guid Structure. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/system.guid.aspx (Luettu: 15.12.2008).

Microsoft 2008 h. How to: Set Up Remote Debugging. Web-dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/bt727f1t.aspx (Luettu: 25.11.2008).

Microsoft 2009 i. Building ASP.NET 2.0 Web Sites Using Web Standards. Web -dokumentti. Saatavilla: http://msdn.microsoft.com/en-us/library/aa479043.aspx (Luettu 10.3.2009).

Petricek Tomas. 2008. Can't return anonymous type from method? Really? Web -dokumentti. Saatavilla: http://tomasp.net/blog/cannot-return-anonymous-type-from-method.aspx (Luettu 26.12.2008).

Taulty Mike 2007 a. When Do Queries Execute? Video. Saatavilla: http://mtaulty.com/videos/nuggets/l2s/11_mt_l2s_whenqueriesexecute.wmv (Katsottu 9.1.2009).

Taulty Mike 2007 b. Introduction to LINQ to SQL. Video. Saatavilla: http://mtaulty.com/videos/nuggets/l2s/01_mt_l2s_introduction.wmv (Katsottu 9.1.2009).

Pialorsi Paolo, Russo Marco. 2008. Introducing Microsoft LINQ. Microsoft. ISBN: 978-0-7356-2391-0

Powers Lars, Snell Mike. 2008. Visual Studio 2008 Unleashed. Sams Publishing. ISBN: 978-0-672-32972-2.

Rattz Joseph. 2008. Pro LINQ: Language Integrated Query in C# 2008. Apress. ISBN: 978-1-59059-789-7.

Redmond Wash. 2008. Microsoft Unveils Next Version of Visual Studio and .NET Framework. Web-dokumentti. Saatavilla: http://www.microsoft.com/presspass/press/2008/sep08/09-29VS10PR.mspx (Luettu 15.2.2009).

Stephen Walther. 2008. ASP.NET 3.5 Unleashed. Sams Publishing.
ISBN: 987-0-672-33011-7.

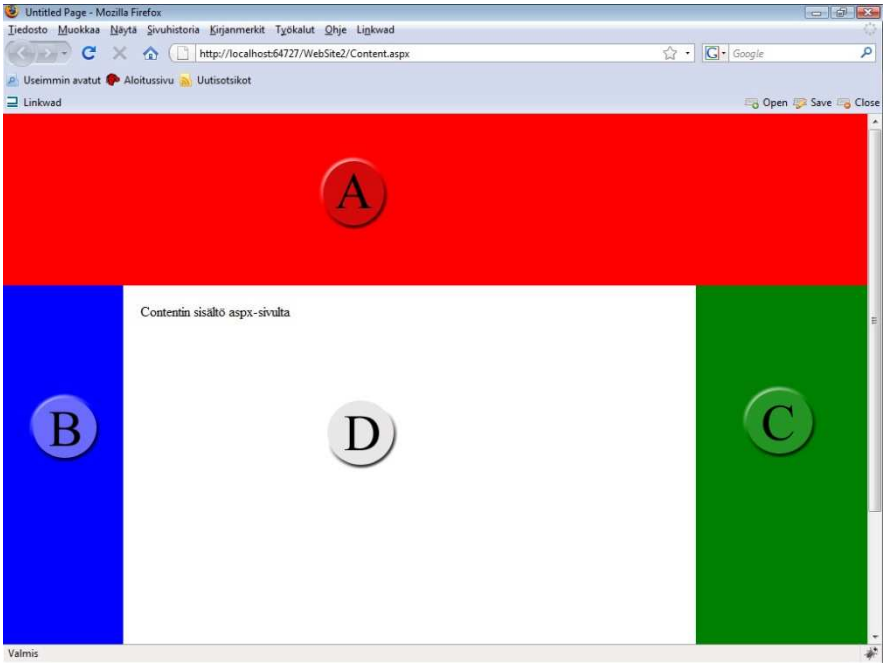
W3. 2008. Introduction to CSS2. Web-dokumentti.
Saatavilla: http://www.w3.org/TR/CSS2/intro.html (Luettu: 22.11.2008).

LIITTEET

Liite 1: Ylisivujen käyttö

Liite 2: Games.xml -data

Yleensä web-sivuston rakenne on seuraavan kuvion 1a. kaltainen.



Kuvio 1a. Kuvitteellisen web-sivuston rakenne

Kuviossa esitettävän web-sivun osat A, B ja C kuvaavat alueita, joita käytetään esimerkiksi mainoksien ja valikoiden esittämiseen. Usein kohdat A, B ja C ovat graafiselta rakenteeltaan muuttumattomia riippumatta siitä, millä sivulla sivustolla liikutaan. Web-sivun osa D kuvaa aluetta, joka yleensä muuttuu eri sivuilla sekä graafisesti että sisällöllisesti. Jos sivustolla käytetään ASP.NET-työkaluja, on järkevää ohjelmoida sivuston alueet A, B ja C ylisivujen avulla. Sivuston muuttuvat osat, kuten alue D, ohjelmoidaan sivukohtaisesti.

Seuraavassa esimerkissä esitellään miten kuvion (kuvio 1a.) mukainen grafiikka tehdään tyylitiedoston ja ylisivun avulla. Esimerkissä käytetään kolmea tiedostoa, jotka ovat Layout.css, Content.aspx ja Masterpage.master.

Layout.css (tyylitiedosto, sijoitettu kansioon Styles):

```
body
{
		margin:0px;
}
#TopBanner
{
		height:200px;
		width:100%;
		background-color:red;
}
#LeftColumn
{
		width:140px;
		height:600px;
		background-color:Blue;
		float:left;
}
#RightColumn
{
		width:200px;
		height:600px;
		background-color:green;
		float:right;
}
#Content
{
		padding:20px 20px 20px 20px;
		float:left;
}
```

Content.aspx (projektin juuritasolla):

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Content.aspx.cs" Inherits="Content" Title="Untitled Page" %>

<asp:Content ID="Content2" ContentPlaceHolderID="head" Runat="Server">
    <link href="Styles/Layout.css" rel="Stylesheet" type="text/css" />
</asp:Content>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    Contentin sisältö aspx-sivulta
</asp:Content>
```

Masterpage.master (projektin juuritasolla):

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Testisivu</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">

    <div>
        <div id="TopBanner"></div>

        <div id="LeftColumn"></div>

        <div id="Content">
            <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">

            </asp:contentplaceholder>
        </div>

        <div id="RightColumn"></div>
    </div>
    </form>
</body>
</html>
```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<games>

  <!--game section-->
  <game>
    <id>1</id>
    <name>Doom</name>
    <price>12,95</price>
    <idgenre>1</idgenre>
  </game>
  <game>
    <id>2</id>
    <name>Age of Conan</name>
    <price>39,95</price>
    <idgenre>3</idgenre>
  </game>
  <game>
    <id>3</id>
    <name>Secret of Monkey Island</name>
    <price>7,95</price>
    <idgenre>1</idgenre>
  </game>
  <game>
    <id>4</id>
    <name>Call of Duty 4</name>
    <price>49,95</price>
    <idgenre>1</idgenre>
  </game>

  <!--genre section-->
  <genre>
    <id>1</id>
    <description>FPS</description>
  </genre>
  <genre>
    <id>2</id>
    <description>Adventure</description>
  </genre>
  <genre>
    <id>3</id>
    <description>RPG</description>
  </genre>

  <!--sales section-->
  <sales>
    <idgame id="1" year="2004" sold="168" />
    <idgame id="1" year="2005" sold="55" />
    <idgame id="2" year="2007" sold="223" />
    <idgame id="2" year="2008" sold="209" />
    <idgame id="3" year="2004" sold="168" />

  </sales>
</games>
```