Andmebaasipõhiste veebirakenduste arendamine Microsoft Visual Studio 2008 ja SQL Server 2008 baasil

Tallinn
2008

Sisukord

Eessõna

Hea õpilane!

Microsofti arenduspartnerid ja kliendid otsivad pidevalt noori ja andekaid koodimeistreid, kes oskavad arendada tarkvara laialt levinud .NET platvormil. Kui Sulle meeldib programmeerida, siis usun, et saame Sulle pakkuda vajalikku ja huvitavat õppematerjali.

Järgneva praktilise ja kasuliku õppematerjali on loonud tunnustatud professionaalid. Siit leid uusimat infot nii .NET aluste kohta kui ka juhiseid veebirakenduste loomiseks. Teadmiste paremaks omandamiseks on allpool palju praktilisi näiteid ja ülesandeid. Ühtlasi on sellest aastast kõigile kättesaadavad ka videojuhendid, mis teevad õppetöö palju põnevamaks.

Oleme kogu õppe välja töötanud vabavaraliste Microsoft Visual Studio ja SQL Server Express versioonide baasil. Need tööriistad on mõeldud spetsiaalselt õpilastele ja asjaarmastajatele Microsofti platvormiga tutvumiseks. Kellel on huvi professionaalsete tööriistade proovimiseks, siis tasub lähemalt tutvuda õppuritele mõeldud DreamSpark programmiga (http://www.dreamspark.com), mille kaudu saab alla laadida tehnilist tarkvara täiesti tasuta.

Microsofti eesmärk selliste õppematerjalide koostamisel on lihtne: tahame tuua kokku uusimat tehnoloogiat tundva põlvkonna, kes ületaks meie partnerite senised ootused ja tõstaks programmeerimise lati tasemele, kuhu me ise ulatunud pole. Sellesama materjali põhjal on edukalt testinud oma teadmisi Eesti kõige paremad arvutiõpetajad, sh ka Sinu kooli õpetaja!

Tulevased maailma parimad koodimeistrid, edu teile!

Rain Laane
Microsofti Eesti esinduse juht

Sissejuhatus

Käesolev juhend on mõeldud kasutamiseks õppematerjalina Veebistuudiumis. Juhendis antakse edasi põhiteadmised, mis on vajalikud andmebaasipõhiste ASP.NET 3.5 veebirakenduste loomiseks.

Koostades alustasime põhitõdedest ning väga keerulisi konstruktsioone ei käsitle. Selle juhendiga töötamiseks piisab, kui on olemas huvi programmeerimise vastu.

Kuigi .NET raamistik võimaldab koodi kirjutamist kümnetes erinevates keeltes, piirdume siin juhendis C# keelega, kui keelega, mis on spetsiaalselt loodud .NET raamistiku tarbeks. Andmebaaside osas vaatleme SQL Server 2008 võimalusi ning XML failide kasutamist.

Õppematerjali väljatöötamist toetasid Microsoft Eesti, BCS Koolitus ja Tiigrihüppe Sihtasutus.

Avastamisrohkeid õpinguid!

Erki Savisaar ja Jaagup Kippar
BCS Koolituse lektorid

Õppematerjali kasutamise juhis

Siinses materjalis on hulgem lehekülgi ja peatükke. Esimese hooga võib see tekst väljatrükituna kätte võttes suisa ära ehmatada. See pole aga sugugi autorite eesmärk. Kirjutist koostades on mõeldud nii algajate kui edasijõudnute peale. Et eesti keeles pole vähemasti siiani (2008ks aastaks) .NETi kohta midagi põhjalikumat kirjutatud, siis püüab see õppematerjal sobida võimalikult paljudele, kel teema kohta huvi või vajadus. Eks edasijõudnud suudavad juba ise sobivaid teemasid leida ning ka veebist ja suurematest targematest raamatutest juurde otsida. Siin aga peaksid siiski nii veebirakenduste, andmebaaside kui ka „puhta" programmeerimise kohta olema sees põhitõed, mille abil on enamik ettetulevatest olukordadest võimalik ära lahendada ning nende oskuste põhjal olla piisavalt tasemel, et suuta soovi korral arvutifirmasse praktikale minna ja mõne ajaga sealsesse töösse sulanduda.

Päris algajatele on valida kaks sisendpunkti, kust peaks saama alustada „tavainimese" arvutialaste teadmiste – ehk siis teksti kirjutamise ja failide salvestamise oskusega. Lihtsam, ilusam ja värvilisem on ASP.NETi peatüki algusosa, kus saab enesele veebilehe kokku panna ja seda soovide järgi kujundada. Edasi sealt juba oskused andmete lugemiseks ja salvestamiseks.

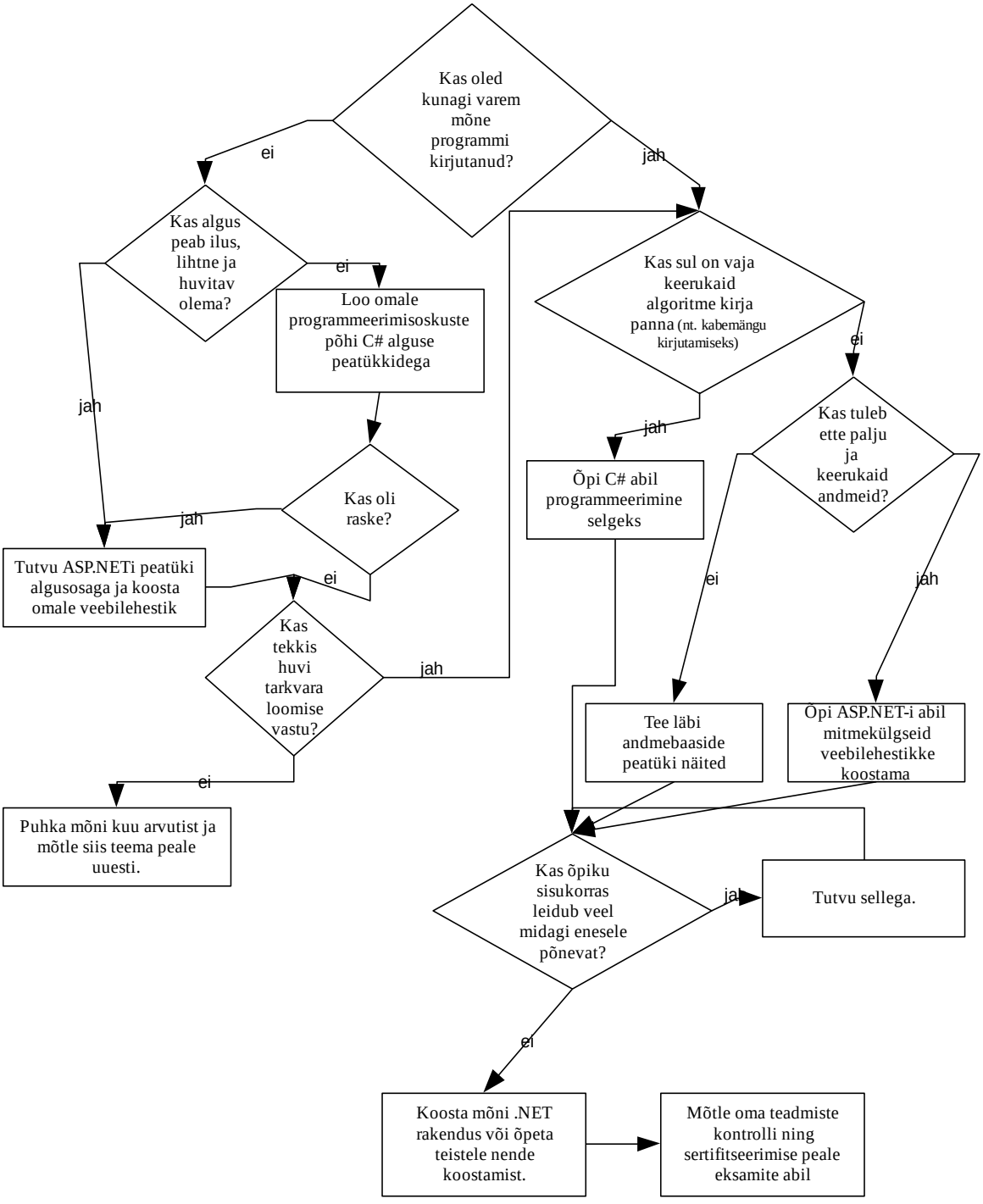
Kel aga tahtmist kohe tõsisemalt programmeerimismaailma siseneda, nende algajate jaoks peaks ka C# peatükk täiesti õpitav olema. Võrreldes veebimaailmaga on sealne kasutajaliides tunduvalt mustvalgem, aga selle eest on jälle tunduvalt vähem kohti, kuhu programmeerimise käigus eksida võiks. Esimese mõnekümne leheküljega saab ülevaate programmeerimise põhimõistetest ja –võimalustest. Edasi suuremaks väljakutseks on objektorienteeritusega seonduv, millest tänapäeval vähegi suurema programmi kirjutamisel ei saa üle ega ümber. Viimased punktid C# juures aga on pigem edasijõudnutele erilisematel juhtudel hakkama saamise või siis sertifikaadieksamiks valmistumise tarbeks. Ka on kõrgkoolidel võimalus kasutada C# materjali oma programmeerimise põhikursuse alusena – senised katsed on olnud edukad.

Veebistuudiumi materjali muud peatükid on ka üles ehitatud põhimõttel, et lihtsamast ja tuttavamast keerukama ja võõrama poole. Andmebaaside juures alustamiseks on baasi ja tabeli loomine hädavajalik tegevus. Lihtsama ülesehitusega päringud kuuluvad iga rakenduse juurde ning sageli saab nendega küllalt palju ära teha. Kui aga andmed keerukamad, siis paratamatult tuleb tabeleid siduda ning sobivate väljavõtete saamiseks ka ridu grupeerida ning nende pealt tulemusi kokku arvutada. Praktiliste andmebaasirakenduste koostamiseks võiks siinne materjal päris hea aluse anda. Kui aga peetava kursuse eesmärgiks on anda ka andmebaaside projekteerimise põhialused (nagu kõrgkoolides kombeks), siis on vaja lisaks tutvuda veel andmemudelite, normaalkujude jm. andmebaasiteooria alla kuuluvate teemadega.

Iga osa lõpus on ülesanded. Neid on püütud sättida nõnda, et keskmisel õppuril oleks paras jutt läbi lugeda, läbi mõelda, mõni näide ka järele proovida. Ning siis ülesanded ette võtta ja nende abil kogu lugu otsast peale uuesti läbi teha. Kui aga mõni ülesanne tundub juba tõsiselt tuttav ja liialt lihtne, eks selle või siis vahele jätta. Või kiiresti läbi proovida, et kas ikka on nii

lihtne kui pealtnäha tundub. Samuti, kui pärast ülesannete läbitegemist kipub vastav peatükk ikka segaseks jääma, tuleb kindlasti kasuks, kui enesele või oma õpilastele mõned teemakohased toimetused juurde mõelda ning sealtkaudu kogu lugu veel korra läbi katsetada. Kordamine on tarkuse ema. Ning eriti hiljem aluseks olevate teemade puhul tuleb ikka kasuks, kui see veelkord pulkadeni läbi näritud ja enesele selgeks tehtud on.

Soovitavat õppematerjali kasutusvõimalust saab vaadata ka järgneva skeemi pealt:



Microsoft .NET platvorm

Microsoft .NET raamistik (Framework) on platvorm programmide loomiseks. Enne .NETi oli võimalik programme luua mitmetele erinevatele platvormidele nt DOS, Win32, Linux jne. .NET platvormi loomise eesmärk on võtta programmeerija õlult kohustus tagada programmi ühilduvus erinevate protsessorite ja operatsioonisüsteemidega, andes rohkem aega programmiga põhifunktsionaalsusega tegelemiseks. Riistvara ja operatsioonisüsteemidega ühilduvuse tagamine on jäetud raamistiku loojate hooleks.

Lisaks Microsoftile, kes pakub raamistiku Windowsi operatsioonisüsteemidele, arendatakse Novelli toetusel analoogset platvormi ka Linux, Solaris, Max OS X ja Unix operatsioonisüsteemile (lisaks on toetatud ka Windowsi operatsioonisüsteem). Platvormi nimeks on Mono ning selle kohta saab rohkem lugeda aadressilt www.mono-project.com. Mõlemad lahendused baseeruvad ühtedel ja samadel ECMA/ISO poolt kinnitatud avatud standarditel.

Kõige aluseks nende platvormide loomisel on CLI e. *Common Language Infrastructure*, mis kirjeldab ära, millised peavad olema keeled, mida platvormil saab kasutada ning millisele kujule peavad nendes keeltes kirjutatud koodid kompileeruma. CLI üks huvitav omadus on see, et enam pole vahet, millises keeles programmeerid, sest kompileeritud kood on lõpuks ikkagi sama. Seega saab iga programmeerija valida endale just selle keele, mis kõige hingelähedasem.

Järgmine tase on CLR e. *Common Language Runtime* (Mono puhul CLI *virtual machine*), mis pakub hallatud (turvalist) keskkonda programmeerimiseks. Selles hallatud keskkonnas on keskse kontrolli all klasside laadimise e. mälu hõivamine, mälu vabastamine (*Garbage collector*) ning vahekoodi kompilaator masinkoodiks. Lisaks pakub CLR programmi loomiseks vajalikke klasse. Lisaks klasside kasutamise võimalusele on paljud neist ka päritavad e. nendest on võimalik pärimise teel luua oma klasse.

Kõrval asuvalt diagrammilt on näha, et raamistik lisab uue kihi programmi ja riistvara vahele, mis loomulikult ei mõju hästi programmi jõudlusele. Hea uudis on see, et Microsofti .NET raamistik integreerub üsna ilusti Windowsi sisse muutudes osaks Windowsist. Enamgi veel, kui võtta Microsofti viimane Windows e. Vista siis raamistik ongi Windows! See tähendab seda, et kõik klassid ja meetodid, mida Microsoft kasutab oma operatsioonisüsteemi loomisel, on kasutamiseks ka kõigile teistele arendajatele, mis omakorda annab kätte väga võimalusterohked vahendid kiiresti funktsionaalsete programmide ehitamiseks.

CLR rakenduste kompileerimine käib kahes faasis:

- Esimese sammuna programmeerija kompileerib oma lähtekoodi vahekeelde. Microsofti raamistiku puhul on selleks MSIL (Microsoft Intermediate Language).

- Teise sammuga käivitamise hetkel CLRi kooseisus olev kompilaator kompileerib MSILi masinkoodi, mida protsessor hakkab täitma. Kompileeritakse vaid need osad programmist, mida kasutatakse e. kompileerimine on kiire, kuigi esimene käivitamine võib olla aeglasem kui kohe binaarsel kujul oleval programmil.

Viimasel hetkel kompileerimise eelis seisneb selles, et protsessorile käivitamiseks mõeldud binaarset koodi on võimalik optimeerida täpselt selle protsessori jaoks, mis hakkab programmi jooksutama. Ei ole vahet, kas protsessor on 32 või 64 bitine jne.

Tulles nüüd tagasi Microsofti .NET platvorm juurde siis see pole mitte üksnes CLR, vaid toodete kogumik, mis sisaldab kõiki vajalikke vahendeid jagatud rakenduste ehitamiseks, pakkudes keelest sõltumatut, ühtset programmeerimise mudelit programmi kõigi kihtide jaoks. .NET platvorm toetab täielikult Interneti platvormist sõltumatuid ja standardseid tehnoloogiaid nagu HTML, XML ja SOAP.

.NET platvormiga seotud tooted võib jagada 4 kategooriasse:

- .NET raamistik – baseerub Common Language Runtime'l (CLR). CLR pakub baasteenuseid ja klasse programmide loomiseks, sõltumata keelest ja programmi kihist.

- .NET My Services – on kogumik kasutajatele suunatud XML veebiteenuseid. Nende teenuste hulka kuuluvad: .NET Passport autentimine, meeldetuletuste saatmine ja vastuvõtmine, personaalse info ja andmete salvestamise võimalus

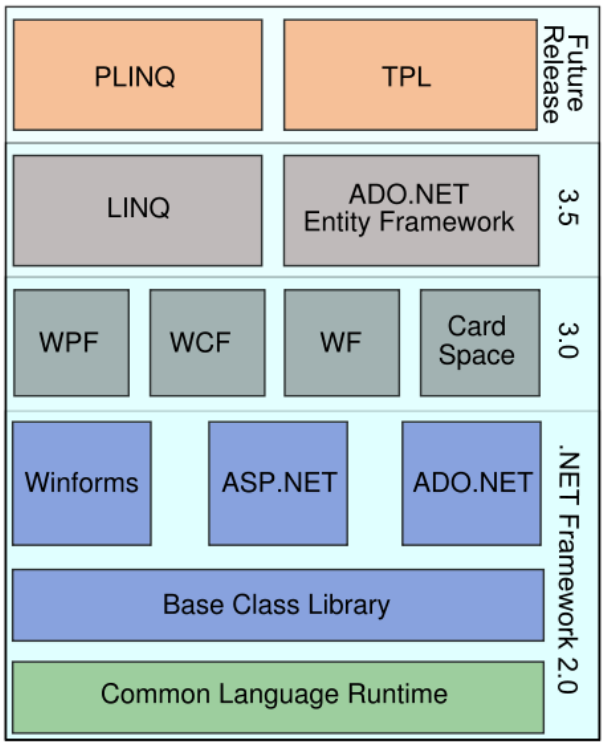
- .NET Enterprise Servers – pakuvad lisafunktsioone ärirakenduste tarbeks. Enterprise serverid on nt SQL Server (andmebaaside tarbeks), Exchange (kirjavahetuse jaoks) jne

- Visual Studio – Vahend .NET rakenduste loomiseks. Lisainfot Visual Studio kohta saab Microsoft veebist http://msdn.microsoft.com/vstudio. Visual Studio 2008st on olemas mitmeid versioone. Programmeerimistee alustamiseks ja katsetamiseks on loodud spetsiaalsed tasuta versioonid Visual Studio Express näol. Express versioone on võimalik tõmmata aadressilt http://msdn.microsoft.com/express. Selle kursuse raames on kasu kahest versioonist Visual Web Developer 2008 Express – ASP.NET 3.5 veebirakenduste loomiseks ja Visual C# Express Edition lihtsalt C# keeles programmeerimise jaoks.

Selleks, et .NET platvormile loodud programme käivitada, on vaja .NET raamistikku. Juhul, kui Teil seda arvutis veel ei ole, on seda võimalik tõmmata mitmelt veebilehelt, sh Windows Update'st või http://www.microsoft.com/net

Programmeerimiskeeltena on vaikimisi võimalik kasutada (kompilaatorid sisalduvad .NET raamistikus) C#, hallatud C++, VB.NET ja J# keeli. Lisaks nendele pakutakse erinevate tootjate poolt veel ca 40 .NET keelt.

Microsoft on .NET raamistikust loonud juba mitmeid versioone. Selle materjali loomise hetkeks on väljas versioon 3.5 koos SP1 teenusepaketiga. Hea uudis nende paljude versioonide juures on see, et igas uues versioonis sisaldub ka vana e. midagi tuleb juurde, kuid kõik see, mis oli, töötab edasi!



The .NET Framework Stack

Kui vaadelda erinevaid .NET raamistiku versioone siis selgub, et .NET põhifunktsionaalsus pärineb hetkel versioonist 2.0, millele on siis lisatud funktsionaalsust järgnevate versiooniuuendustega.

Versiooniga 3.0 lisati juurde:

WPF – Windows Presentation Foundation so uus kasutajaliidese ehitamise liides, mis asendab Winform'id. WPF baseerub XMLil ning vektrograafikal, mis oskab kasutada 3D arvutigraafika riistvara ning Direct 3D tehnoloogiat.

WCF – Windows Communication Foundation so teenusepõhine teadetevahetussüsteem, mis võidab programmidel suhelda nii lokaalselt kui ka üle võrgu. Asendab Web Serviced

WF – windows Workflow Foundation võimaldab automatiseerida tegevuste sooritamist ning luua erinevatest tegevustest transaktsioone kasutades töövooge.

Windows CardSpace – tarkvarakomponent, mis hoiustab turvaliselt kasutaja digitaalseid identiteetiteete ning pakub ühtset liidest identiteedi valimiseks erivatele transaktsioonidele nagu nt veebilehele sisselogimine.

Versiooniga 3.5 lisandus LINQ – Language INtegrated Query, mis pakub ühtset päringute kirjutamise liidest erinevat tüüpi andmeallikatele. Päringute loomise käsustik on integreeritud keele sisse, sellest tulenevalt kaasnevad .NET 3.5ga ka uued versioonid keeltest C# ja VB.NET. Esialgu on koos raamistikuga kaasas LINQ teenusepakkujad päringute tegemiseks objektidest, XMList ja SQL Serverist.

Lisainfot .NET raamistiku erinevate versioonide kohta leiate aadressilt
http://en.wikipedia.org/wiki/.net_framework#versions/

C#

Mõnigi võib ohata, et jälle üks uus programmeerimiskeel siia ilma välja mõeldud. Teine jälle rõõmustab, et midagi uut ja huvitavat sünnib. Kolmas aga hakkas äsja veebilahendusi kirjutama ja sai mõnegi ilusa näite lihtsasti kokku. Oma soovide arvutile selgemaks tegemise juures läheb varsti vaja teada, "mis karul kõhus on", et oleks võimalik täpsemalt öelda, mida ja kuidas masin tegema peaks. Loodetavasti on järgnevatel lehekülgedel kõigile siia sattunute jaoks midagi sobivat. Mis liialt lihtne ja igav tundub, sellest saab kiiresti üle lapata. Mis esimesel pilgul paistab arusaamatu, kuid siiski vajalik, seda tasub teist korda lugeda. Ning polegi loota, et kõik kohe lennult külge jääks!?

Selle jaoks on teksti sees koodinäited, mida saab kopeerida ja arvutis tööle panna. Ning mõningase muutmise ja katsetamise peale avastada, mis mille jaoks on ning kuidas seda oma kasuks rakendada saab. Töötav näide on üks hea kindel tugipunkt nagu üks suur puu lagendikul, kuhu oskab alati tagasi minna. Juhul, kui muutmistega on õnnestunud oma koodilõik nii sõlme keerata, et see sugugi enam töötada ei taha, saab alati võtta materjalist taas algse töötava näite ning sealt juurest katsetama hakata.

Kes pikemalt mitmesuguseid rakendusi kirjutab, avastab mõne aja pärast, et samas keeles kirjutatud programm võib vähemalt esmapilgul mõnevõrra erinev välja näha sõltuvalt sellest, kas programm käivitatakse veebist, tegutsetakse nuppudega ja tekstiväljadega aknas, väljundiks on mobiiltelefon või piirdub kogu tegevus tekstiaknaga. Esimesel korral võib tunduda, et oleks nagu täiesti eri keeltes ja eri moodi kirjutamine. Ühes kohas on alati koodi juures salapärane `button1_click`, teises `public static void Main` ning kolmandas veel midagi muud. Aga sellest ei tasu ennast väga häirida lasta. Ehkki .NETi ja C# juures on püütud eri kohtades käivituvate rakenduste loomist sarnasemaks muuta, tuleb siiski kirjutamisel arvestada käivitumiskoha võimalustega. Siin materjalis keskendume C# keele ülesehitusega seotud teemadele, mis on ühised kõigi käivitumiskohtade puhul. Ning kasutajaliidesena pruugime programmeerimisõpikute traditsioonilist lihtsat ning väheste (eksimis)võimalustega tekstiakent - nii jääb rohkem aega tähelepanu pühendada keele enese konstruktsioonidele, mida siis edaspidi julgesti veebirakenduste juures ja soovi korral mujalgi pruukida. Kes aga tahab omale koodi kirjutamiseks rohkem abivalmis ning lisavõimalustega keskkonda, sellele soovitame lugeda peatükki nimega Visual Studio C# Expressi install – samm sammult juhis vastava keskkonna paigaldamiseks ning esimese rakenduse käivitamiseks. Edasine kirjutamine sarnaselt konspektis olevale.

Põhivõimalused

Kui rakendus juba mingitki elumärki annab, on see tunduvalt rohkem, kui lihtsalt hulk koodi, mis peaks "midagi arukat" tegema. Tunne, et suutsin programmi ise, omade roosade kätega käima panna, on hea. Ja annab kindlustunde, et järgmisel korral saab asi ainult paremaks minna. Kui käima on lükatud, siis edasi võib mõtelda juba juurde panemise peale. Nii nagu talumees, kes omale krati oli ehitanud, sai hakata talle ülesandeid andma alles siis, kui kratt hinge sisse võttis. Muul juhul on tegemist palja põhuhunnikuga, millele teivas sisse löödud ja vanad kartulikorvid külge riputatud - olgu need nii suured ja vägevad tahes. Tööle hakkamiseks on hinge vaja. Aga hinge ei saa enne sisse puhuda, kui väikegi tervik olemas. Ning C# puhul näeb lühim tervikprogramm välja ligikaudu järgmine:

```
using System;
class Tervitus{
   public static void Main(string[] arg){
      Console.WriteLine("Tere");
   }
}
```

Esmapilgul võib tunduda, et suur hulk tundmatuid sõnu on ekraanile ritta seatud. Ning veel mõned sulud ka. Ning kui teada saada, et kõik see pikk jutt on vaid väikese programmi loomiseks, mis oma töö tulemusena ütleb "Tere", siis võib paista, et tegemist on ilmse raiskamisega. Eriti, kui mõni on varem tutvunud Basicu või Pythoniga, kus sama tulemuse saamiseks tuleb lihtsalt kirjutada

```
print "Tere"
```

Seetõttu veel 2000ndate aastate alguses tutvustati inimesi programmeerimisega mitmeski koolis just Basicu kaudu, sest algus on "nii lihtne". Ning kui programmid "väga suureks" - ehk siis tuhandete ridade pikkuseks - ei kasva, võib julgesti lihtsa algusega keele juurde jäädagi. Kõik vajalikud matemaatikaülesanded, kirjutamised ja arvutamised saavad tehtud.

Aga millegipärast on programmidel kombeks paisuda. Ning et suureks paisunud rakenduse seest sobiv toiming üles otsida, peab lisama vajalikule käsule üha uusi ja uusi kesti. Nii nagu üheainukese vajaliku sidekanali puhul võib korraldada nii, et telefonitoru tõstes ühendatakse rääkija kohe õigesse kohta. Kümnekonna puhul piisab telefoninumbritest, mis telefoni külge kleebitud või kiirklahvidele salvestatud. Kui aga raamatupidaja peab vajalikul hetkel kätte saama kõik temaga viimastel aastatel suhelnud kliendid, siis läheb juba tarvis korralikumat kataloogi.

Nõnda on C# ja ka mitme muu keele loojad otsustanud juba alguses programmikoodi osad korralikult süstematiseerida, et poleks vaja piltlikult öeldes pärast kataloogi tehes nuputada, millist värvi paberilipikule kirjutatud Mati telefoninumber just selle õige Mati oma on. Mõnevõrra läheb tekst selle peale küll pikemaks, aga loodetavasti piisavalt vähe, et ikka julgete C# võlusid tundma õppida.

C# kood jaotatakse üksteise sees olla võivatesse plokkidesse. Iga ploki ümber on looksulud. Siinses näites on välimiseks plokiks klass nimega Tervitus ning tema sees alamprogramm nimega Main. Plokke võib vahel tunduvalt rohkem olla. Omaette terviklikud toimingud paigutatakse üldjuhul alamprogrammidesse. Nende sees võivad olla plokid tingimuste ja korduste tarbeks. Klassi moodustab üheskoos toimivate või sarnaste alamprogrammide komplekt (näiteks matemaatikafunktsioonid) - sellest pikemalt aga hiljem objektinduse juures. Suuremate rakenduste juures jagatakse klassid veel omakorda nimeruumidesse. Nii on lootust ka pikemate rakenduste puhul midagi hiljem koodist üles leida.

Mõned salapärased kohad on veel jäänud. Esimene rida

```
using System;
```

teatab, et nimeruumist System pärinevaid klasse saab kergesti kasutada - piisab vaid klassi nimetamisest. C# standardpaketis leidub tuhandeid kasutamisvalmis klasse. Lisaks veel loendamatu hulk lisapakette, mis muud programmeerijad on valmis kirjutanud. Kindlasti leidub nende hulgas korduvaid klassinimesid - sest mõttekaid ja kõigile arusaadavaid klassinimetusi lihtsalt ei saa olla nii palju. Kui aga samanimelised klassid on eri nimeruumides, siis nad üksteist üldjuhul ei sega. Nii võib igaüks oma nimeruumis nimetada klasse kuidas tahab. Ja using-lausega märgitakse, millist klasside komplekti parajasti kasutatakse. Edasi siis

```
class Tervitus{
```

mis praegusel juhul annab programmile ka nime ning

```
   public static void Main(string[] arg){
```

näitab alamprogrammi `Main`, kust käsurearakendus oma tööd alustab. Muudest sõnadest siin rea peal ei tasu end veel liialt häirida lasta. Kiirülevaatena: `public` näitab, et pole seatud käivitusõiguse tõkkeid, `static` - et alamprogramm Main on olemas ja käivitusvalmis kohe rakenduse töö alguses. `void` näitab, et ei raporteerita operatsioonisüsteemile programmi töö edukuse kohta. Ja ümarsulgudes oleva abil saab mõnes rakenduses kasutaja omi andmeid ette anda - siin seda võimalust aga ei kasutata.

Järgneb kasutajale nähtav toiming, ehk

```
      Console.WriteLine("Tere");
```

`Console` klass asub nimeruumis `System` ja on üleval märgitud using lause tõttu kasutatav. Klassi käsklus `WriteLine` lubab kirjutada konsoolile ehk tekstiekraanile. Praegu piirdutakse ühe väikese teretusega. Jutumärgid on ümber selleks, et arvuti saaks aru, et tegemist on tekstiga - mitte näiteks käskluse või muutuja (märksõna) alla salvestatud andmetega.

```
   }
}
```

Kaks sulgu lõpus lõpetamas eespool avatud sulgusid. Iga sulg, mis programmikoodi sees avaneb, peab ka kusagil lõppema - muidu ei saa arvuti asjast aru, hing ei tule sisse ja programm ei hakka tööle. Tühikud ja reavahetused on üldjuhul vaid oma silmailu ja pildi korrastuse pärast. Kompilaatori jaoks võiks kõik teksti rahumeeli ühte ritta jutti kirjutada, enesele kasvaks aga selline programm varsti üle pea. Siin näites paistab, et alamprogramm `Main`' i sulg on sama kaugel taandes kui alamprogrammi alustav rida ise. Ning klassi sulg on sama kaugel kui klassi alustava rea sulg. Nõnda saab programmi pikemaks kasvamisel kergemini järge pidada, millises plokis või millistes plokkides vaadatav käsk asub.

Nõnda on esimene väike programm üle vaadatud ja loodetavasti tekib natuke tuttav tunne, kui vaadata järgnevaid ridu:

```
using System;
class Tervitus{
   public static void Main(string[] arg){
      Console.WriteLine("Tere");
   }
}
```

Käivitamine

Edasi tuleb tervitusrakendus käima saada. Töö tulemusel tekkinud pildi võib arenduskeskkonnas (näiteks Visual Studio 2008) ette saada kergesti - vajutad lihtsalt käivitusnuppu ja midagi ilmubki. Kuidas rakendus paigaldada, seda saab lugeda C# osa lõpust. Et aga "karu kõhust" tekiks parem ülevaade, püüame tervitaja käsurealt tööle saada. Sealt paistab paremini välja, millised etapid enne läbi käiakse, kui töötava programmini jõutakse. See on kasulik näiteks vigade otsimisel ja neist aru saamisel.

C# programmi kood salvestatakse laiendiga cs. Faili nime võib panna loodud programmi klassiga samasuguseks, kuigi otsest kohustust ei ole. Aga nõnda on kataloogist omi programme ehk kergem üles leida, kui klass ja fail on sama nimega.

Käsureale pääsemiseks on tarvilik see avada. Tavaliseks mooduseks on Start -> run ja sinna sisse käsklus cmd. Et sealtkaudu kompilaatorile ligi pääseda, on lisaks vaja panna otsinguteesse kompilaatori csc.exe kataloogi aadress, mis ühe konkreetse installatsiooni puhul on näiteks C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727.

Lihtsam moodus on kasutada Microsoft .NET Framework SDK-ga kaasa tulevat SDK Command Prompt' i, kus vastav seadistus juba paigas.

Edasi tuleb käsuaknaga liikuda kataloogi, kuhu programmikoodifail salvestatud sai. Siinpuhul on selleks C ketta Projects kataloogi alamkataloogi oma alamkataloog näited. Käsureal liikumiseks saab ketast valida käsuga, mis koosneb kettatähest ja temale järgnevast koolonist. Kataloogides liikumiseks sobib käsklus cd. Nii et siin puhul tuleb ketta valimiseks kirjutada

```
C:
```

ning sealt seest kataloogi valimiseks

```
cd \TEMP\naited
```

Nõnda jõudsingi sobivasse asukohta. Kataloogi sisu nägemiseks sobib käsklus `dir`. Tulemus:

```
C:\TEMP\naited>dir
 Volume in drive C has no label.
 Volume Serial Number is 0C0F-6FEC


 Directory of C:\TEMP\naited
```

```
08.10.2008  16:46    <DIR>          .

08.10.2008  16:46    <DIR>          ..

08.10.2008  16:46               121 Tervitus.cs

               1 File(s)            121 bytes

               2 Dir(s)  22 939 922 432 bytes free
```

Siit paistab, et kataloogis on üks fail. Selle nimeks on Tervitus.cs ning pikkuseks 116 baiti.

Käivitamiseks tuleb meie loodud programmikoodiga tekstifail kõigepealt ära kompileerida. Kompileerimise käigus tehakse käsklused masinale kergemini loetavamaks. C# käsurearakenduse puhul on kompileerimise tulemuseks .exe - laiendiga fail, mis oma käivitumiseks vajab, et .NET keskkond oleks masinas juba olemas. Kompileerimiseks kirjutatakse kompilaatorprogrammi nimi (csc) ning programmikoodifaili nimi (praegu Tervitus.cs). Kui programmikood on masinale arusaadavalt kirja pandud, veateateid ei anta ning kataloogi tekib juurde käivitamisvalmis Tervitus.exe

```
C:\TEMP\naited>csc Tervitus.cs
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.1
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.
```

Käima panekuks tuleb vaid selle .exe faili nimi kirjutada ning võimegi tulemust näha.

```
C:\TEMP\naited>Tervitus
Tere
```

Ülesandeid

* Muuda väljatrükitavat teksti

* Kirjuta ekraanile kaks rida (kaks järjestikust Console.WriteLine käsklust, kumbki omal real)

* Tekita programmi sisse väikeseid vigu, ja vaata, mis kompilaator selle peale teatab. Paranda tagasi ja veendu, et programm töötab jälle. Näiteks kaota ära n-täht sõnast Console, üks jutumärk "Tere" ümbert, üks sulg, semikoolon, muuda klassi nime. Ja igal korral jäta meelde või soovi korral suisa märgi üles, kas ja milline viga oli tegelikult ning millise veateate andis kompilaator. Sellised on kõige tüüpilisemad kompileerimisel tekkivad vead. Kord väikese programmi juures läbi katsetatuna on kergem sellistest veateadetest ka suures rakenduses aru saada.

Suhtlus arvutiga

Enamik programme vajavad andmeid kasutajalt - muul juhul ei teaks ju arvuti, mida meil vaja on. Kui just programmi ainsaks ülesandeks polegi kellaaja teatamine, sest sellisel juhul tõesti piisab vaid programmi enese käivitamisest.

Sisendite võimalused sõltuvad programmi kasutuskohast. Veebis näiteks saame pruukida veebilehel töötavaid graafikakomponente - nuppe, tekstivälju, rippmenüüsid jne. Windows Formide ehk arvutis iseseisvalt (mitte veebi kaudu) töötavate graafiliste rakenduste puhul on tavalised graafikakomponendid suhteliselt sarnased veebis nähtavatega. Vaid mõnevõrra vabamalt pääseb oma komponente juurde tegema ning mitmekülgsemaid võimalusi kasutama. Tekstiakna rakenduste juures piirdub suhtlus arvutiga loetava ja trükitava tekstiga. Lihtsaim dialoogi pidav programm näeb välja järgmine:

```
using System;
class Sisend{
   public static void Main(string[] arg){
      Console.WriteLine("Palun eesnimi:");
      string eesnimi=Console.ReadLine();
      Console.WriteLine("Tere, "+eesnimi);
   }
}
```

Ning töö paistab välja nii:

```
C:\Projects\oma\naited>Sisend
Palun eesnimi:
Juku
Tere, Juku
```

Esmane "Palun eesnimi" trükitakse välja sarnaselt nagu lihtsaimaski tervitavas programmis. Edasine `Console.ReadLine()` jääb kasutajalt sisestust ootama. Kõik, mis kasutaja kuni reavahetuseni kirjutab, püütakse kokku üheks tekstiks ning selle saab arvutisse meelde jätta. Märksõnaks ehk muutuja nimeks sai "eesnimi" ning andmetüübiks "string", mis inimkeeli tähendab teksti. Järgmisel real trükitakse tulemus välja. Nõnda sõltub programmi vastus küsimise peale sisestatavast nimest.

Arvutamine

Arvutamine teadupärast arvuti põhitöö - vähemalt arvutustehnika algaastatel. Et siin lahkesti kasutaja antud arve liita/lahutada saaks, tuleb kõigepealt hoolitseda, et need ka arvuti jaoks arvud ja mitte sümbolite jadad oleksid. Kõigepealt annab `ReadLine` kätte numbriliste sümbolitega teksti. Ning käsklus `int.Parse` muudab selle arvutuste jaoks kõlbulikuks. Tüüp `int` (sõnast integer) tähistab täisarvu. Kui on vaja komakohtadega ümber käia, siis sobib selleks tüüp `double`. Teise arvu puhul on andmete lugemine ning arvuks muundamine ühte käsklusesse kokku pandud. Nii võib ka.

Väljatrüki juures näete kolme looksulgudesse paigutatud arvu. Nõnda on võimalik andmeid trükkides algul määrata ära trükkimise kohad ning alles pärast loetellu kirjutada tegelikud väärtused. Juhul, kui väärtuste arvutamine on pikk (näiteks `arv1*arv2`), aitab see programmikoodi pilti selgemana hoida. Muul juhul tuleks hulk pluss- ja jutumärke väljatrüki juurde. Jutumärgid tekstide eristamiseks ning plussmärgid üksikute osade kokku liitmiseks. Samuti on sellisest asukohanumbritega paigutamisest kasu juhul, kui rakendust tõlgitakse. Keele lauseehituste tõttu võib sõnade järjestus lauses muutuda. Selliselt looksulgude vahel olevate arvudega mängides aga saab lihtsamalt tõlkida ilma, et peaks selleks programmikoodis märgatavaid muutusi tegema.

```
using System;
class Arvutus{
   public static void Main(string[] arg){
      Console.WriteLine("Esimene arv:");
      string tekst1=Console.ReadLine();
      int arv1=int.Parse(tekst1);
      Console.WriteLine("Teine arv:");
      int arv2=int.Parse(Console.ReadLine());
      Console.WriteLine("Arvude {0} ja {1} korrutis on {2}",
              arv1, arv2, arv1*arv2);
   }
}
C:\Projects\oma\naited>Arvutus
Esimene arv:
3
Teine arv:
5
Arvude 3 ja 5 korrutis on 15
```

Ülesandeid

* Küsi kahe inimese nimed ning teata, et täna on nad pinginaabrid

* Küsi ristkülikukujulise toa seinte pikkused ning arvuta põranda pindala

* Leia 30% hinnasoodustusega hinna põhjal alghind

Valikud

Ehk võimalus otsustamiseks, kui on vaja, et programm käituks kord üht-, kord teistmoodi. Allpoololev näide koos väljundiga võiks näidata, kuidas tingimuslause abil tehtud valik toimib.

```
using System;
public class Valik1{
   public static void Main(string[] arg){
      Console.WriteLine("Palun nimi:");
      string eesnimi=Console.ReadLine();
      if(eesnimi=="Mari"){
         Console.WriteLine("Tule homme minu juurde!");
      } else {
```

```
            Console.WriteLine("Mind pole homme kodus.");
        }
    }
}
```

Väljund:

```
D:\kodu\0606\opikc#>Valik1
Palun nimi:
Juku
Mind pole homme kodus.
```

Nagu näha - Jukut külla ei kutsutud. C# juures, nii nagu selle aluseks oleva C-keele puhul kasutatakse võrdlemise juures kahte võrdusmärki. Üks võrdusmärk on omistamine ehk kopeerimine. Arvude puhul saab kasutada ka võrdlusi < ja > ehk suurem kui ja väiksem kui. Näiteks

```
if(vanus>14){
   Console.WriteLine("Tuleb osta täispilet");
}
```

Samuti kehtivad võrdlused >= ja <= ehk suurem või võrdne ning väiksem või võrdne. Kui uurida, kas arv jääb soovitud vahemikku, tuleb järjest panna kaks võrdlust. Näiteks:

```
if(vanus>6 && vanus <=14){
   Console.WriteLine("Sinu jaoks on lapsepilet");
}
```

Kaks &-märki tähendab, et kogu tingimus on tõene ainult siis, kui mõlemad võrdlused on tõesed. Teistpidi saab ka.

```
if(vanus<7 || vanus>14){
   Console.WriteLine("Sulle lapsepilet ei sobi");
}
```

&&märke saab lugeda sõnaga "ja", || märke sõnaga "või". Ehk siis: kui vanus on alla seitsme või suurem neljateistkümnest, sel juhul lapsepilet ei sobi.

Kommentaarid

Vahel on kasulik enesele ja teistele selgituseks programmi sisse kommentaare lisada. Teksti sisse kirjutades pannakse kommentaari ette kaks kaldkriipsu. Selliselt kirjutatud teksti pole kompilaatori jaoks olemas. Enesel on aga vahel päris hea meenutada, mida mõne lausega kirja panna taheti.

```
      double summa=kogus*pirnihind; //korrutatakse ühe pirni hinnaga
```

Kommentaar võib olla ka üle mitme rea. Sellisel juhul pannakse kommentaari algusesse `/*` ning lõppu `*/` Siin on näiteks programmi väljund sarnaselt kaldkriips-tärni ning tärn-kaldkriipsu vahele paigutatud - et ka ilma käivitamata võib juba näha, mis konkreetsel juhul tehti.

Lisaks märkmete jätmisele on kommenteerimine ka heaks võimaluseks programmi testimisel ja vigade otsimisel. Kui kuidagi viga üles ei leia, siis saab algul kahtlase lause või lausete ploki vahel tervikuna välja kommenteerida. Seejärel veenduda, et programm ilma selle osata ilusti töötab. Ning edasi võib asuda juba kindlama tundega sellest väljakommenteeritud plokist viga otsima teades, et ta kindlasti seal peab olema. Omakorda saab lauseid ükshaaval võtta kommenteeritud osast välja ja lõpuks nõnda kõik ilusti tööle saada. Aga nüüd näide ise.

```
using System;
public class Valik1{
   public static void Main(string[] arg){
      double pirnihind=1.70;
      Console.WriteLine("Mitu pirni ostad?");
      double kogus=double.Parse(Console.ReadLine());
      double summa=kogus*pirnihind; //korrutatakse ühe pirni hinnaga
      Console.WriteLine("Kas kilekotti ka soovid? (jah/ei)");
      string vastus=Console.ReadLine();
      if(vastus=="jah"){
         summa=summa+0.70;
      }
      Console.WriteLine("Kogusumma: "+summa);
   }
}
/*
D:\kodu\0606\opikc#>Valik2
Mitu pirni ostad?
3
Kas kilekotti ka soovid? (jah/ei)
jah
Kogusumma: 5,8
*/
```

Kui soovida üksikutest osadest summat kokku arvutada nagu ülal näites, siis on küllalt tavaline, et iga küsimuse peal otsustatakse, kas ja mida selle juures teha. Siin nagu näha - juhul kui kilekott ostetakse lisaks, siis pannakse hinnale 70 senti juurde.

```
summa=summa+0.70
```

tähendab just seda.

Ülesandeid

* Küsi temperatuur ning teata, kas see on üle kaheksateistkümne kraadi (soovitav toasoojus talvel).

* Küsi inimese pikkus ning teata, kas ta on lühike, keskmine või pikk (piirid pane ise paika)

* Küsi inimeselt pikkus ja sugu ning teata, kas ta on lühike, keskmine või pikk (mitu tingimusplokki võib olla üksteise sees).

* Küsi inimeselt poes eraldi kas ta soovib osta piima, saia, leiba. Löö hinnad kokku ning teata, mis kõik ostetud kraam maksma läheb.

Kordused

Arvutist on enamjaolt kasu siis, kui ta meie eest mõned sellised tööd ära teeb, kus enesel tarvis ükshaaval ja üksluiselt sama toimetust ette võtta. Ühest hinnast paarikümne protsendi maha arvutamisega saab ka käsitsi mõne ajaga hakkama. Kui aga hindu on mitukümmend, siis on päris hea meel, kui arvuti selle töö meie eest ära teeb. Järgnevalt näide, kuidas arvuti vastu tulevale viiele matkajale tere ütleb. Täisarvuline muutuja nimega nr näitab, mitmenda matkaja juures parajasti ollakse. Käskluse while juurde kuuluvat plokki saab korrata. Plokk läbitakse üha uuesti juhul, kui ümarsulgudes olev tingimus on tõene. Et kordusi soovitud arv saaks, on juba programmeerija hoolitseda. Selleks on siin igal korral pärast tervitust käsklus `nr=nr+1`, ehk siis suurendatakse matkaja järjekorranumbrit. Kui see suurendamine kogemata tegemata jääks, siis jääkski arvuti igavesti esimest matkajat teretama (proovi järele). Nüüd aga töötav korduste näide:

```
using System;
public class Kordus1{
   public static void Main(string[] arg){
      int nr=1;
      while(nr<=5){
         Console.WriteLine("Tere, {0}. matkaja!", nr);
         nr=nr+1;
      }
   }
}
```

Ning tema väljund:

```
D:\kodu\0606\opikc#>Kordus1
Tere, 1. matkaja!
Tere, 2. matkaja!
Tere, 3. matkaja!
Tere, 4. matkaja!
Tere, 5. matkaja!
```

Heal lapsel mitu nime. Ehk ka korduste kirja panekuks on mitu moodust välja mõeldud. Algul näidatud while-tsükkel on kõige universaalsem, selle abil on võimalik iga liiki kordusi kokku panna. Sama tulemuse aga saab mõnevõrra lühemalt kirja panna `for`-i abil. Levinud kordusskeemi jaoks on välja mõeldud omaette käsklus, kus algul luuakse muutuja ja antakse talle algväärtus; seejärel kontrollitakse, kas järgnevat plokki on vaja täita; lõpuks võetakse ette toiming andmete ettevalmistamiseks uue ploki jaoks. `nr++` on sama, mis `nr=nr+1` - ehk siis suurendatakse muutuja väärtust ühe võrra. Näha programminäide:

```
using System;
public class Kordus2{
   public static void Main(string[] arg){
      for(int nr=1; nr<=5; nr++){
         Console.WriteLine("Tere, {0}. matkaja!", nr);
      }
   }
}
```

ning tema väljund:

```
D:\kodu\0606\opikc#>Kordus2
Tere, 1. matkaja!
Tere, 2. matkaja!
Tere, 3. matkaja!
Tere, 4. matkaja!
Tere, 5. matkaja!
```

Järelkontroll

Nii `for`-i kui `while` puhul kontrollitakse alati ploki algul, kas seda on vaja täita. Ning kui matkajate arv poleks mitte 5, vaid hoopis 0, siis sellisel juhul ei täideta plokki ainsatki korda, ehk kõik tered jäävad ütlemata. Mõnikord on aga teada, et plokk tuleb kindlasti läbida. Lihtsalt pole teada, kas sama teed tuleb ka teist või kolmandat korda käia. Tüüpiline näide selle juures on sisestuskontroll. Kui esimene kord toiming õnnestus, pole vaja kasutajalt andmeid uuesti pärida. Juhtus aga äpardus, siis tuleb senikaua jätkata, kuni kõik korras on. Siin küsitakse jälle tunninäitu. Sattus see arusaadavase vahemikku, minnakse rahus uue väärtusega edasi. Kui aga midagi ebaõnnestus, siis on arvuti väga järjekindel uuesti ja uuesti küsima lootuses, et kunagi ka midagi temale sobilikku jagatakse.

```
using System;
public class Kordus3{
   public static void Main(string[] arg){
      int tund;
      do{
         Console.WriteLine("Sisesta tund vahemikus 0-23");
         tund=int.Parse(Console.ReadLine());
      } while(tund<0 || tund>23);
      Console.WriteLine("Tubli, sisestasid {0}.", tund);
   }
}
/*
D:\kodu\0606\opikc#>Kordus3
Sisesta tund vahemikus 0-23
32
Sisesta tund vahemikus 0-23
11
Tubli, sisestasid 11.
*/
```

Ülesandeid

* Trüki arvude ruudud ühest kahekümneni

* Küsi kasutajalt viis arvu ning väljasta nende summa

* Ütle kasutajale "Osta elevant ära!". Senikaua korda küsimust, kuni kasutaja lõpuks ise kirjutab "elevant".

Korrutustabel

... ehk näide, kuidas eelnevalt vaadatud tarkused ühe programmi sisse kokku panna ning mis selle peale ka midagi tarvilikku teeb.

Algul on näha, kuidas otse programmi käivitamise juures ka mõned andmed sinna kätte anda. Et kui kirjutan

```
Korrutustabel 4 5
```

siis saadakse sellest aru, et soovin korrutustabelit nelja rea ja viie veeruga. Nende käsurea parameetrite püüdmiseks on alamprogramm `Main`-i ümarsulgudes koht `string[] argumendid`. Kõik käsureale kirjutatud sõnad (ka üksik number on arvuti jaoks sõna) pannakse sinna argumentide massiivi ehk jadasse, kust neid järjekorranumbri järgi kätte saab. Andmetüüp `string[]` tähendabki, et tegemist on stringide ehk sõnede ehk tekstide massiiviga. Kirjutades massiivi järgi `.Length`, saab teada, mitu elementi selles massiivis on - mis praegusel juhul on võrdne lisatud sõnade arvuga käsureal. Kõik sõnad saab ka ükshaaval järjekorranumbri järgi kätte. Arvestama peab ainult, et sõnu hakatakse lugema numbrist 0. Nii et kui eeldatakse, et tegemist on kahe parameetriga, siis nende kättesaamiseks peame ette andma numbrid null ja üks.

Nagu tingimusest on näha: juhul kui argumente pole täpselt kaks, siis kasutatakse vaikimisi ridade ja veergude arvu ning joonistatakse korrutustabel suurusega 10 korda 10.

Tabeli trükkimiseks on kaks `for`-tsüklit paigutatud üksteise sisse. Selles pole midagi imelikku - iga rea juures trükitakse kõik veerud esimesest kuni viimaseni. Ning selleks, et erinevate numbrite arvuga arvud meie tabelit sassi ei lööks, on väljatrüki juurde vorminguks kirjutatud `{0, 5}`. Ainsat `Console.Write` argumenti (järjekorranumbriga 0) trükitakse nõnda, et ta võtaks alati viis kohta.

```
using System;
class Korrutustabel{
   public static void Main(string[] argumendid){
      int ridadearv=10, veergudearv=10;
      if(argumendid.Length==2){
         ridadearv=int.Parse(argumendid[0]);
         veergudearv=int.Parse(argumendid[1]);
      }
```

```
      for(int rida=1; rida<=ridadearv; rida++){
         for(int veerg=1; veerg<=veergudearv; veerg++){
            Console.Write("{0, 5}", rida*veerg); //5 kohta
         }
         Console.WriteLine();
      }
   }
}
/*
C:\Projects\oma\naited>Korrutustabel
    1    2    3    4    5    6    7    8    9   10
    2    4    6    8   10   12   14   16   18   20
    3    6    9   12   15   18   21   24   27   30
    4    8   12   16   20   24   28   32   36   40
    5   10   15   20   25   30   35   40   45   50
    6   12   18   24   30   36   42   48   54   60
    7   14   21   28   35   42   49   56   63   70
    8   16   24   32   40   48   56   64   72   80
    9   18   27   36   45   54   63   72   81   90
   10   20   30   40   50   60   70   80   90  100
C:\Projects\oma\naited>Korrutustabel 4 5
    1    2    3    4    5
    2    4    6    8   10
    3    6    9   12   15
    4    8   12   16   20
*/
```

Alamprogramm

Nii nagu algul kirjas, nii ka siin tasub meeles pidada, et programmid armastavad paisuda ja paisuda. Seepärast tuleb leida mooduseid, kuidas üha suuremaks kasvavas koodihulgas orienteeruda. Alamprogramm on esmane ja hea vahend koodi sisse uppumise vältimiseks. Lisaks võimaldab ta terviklikke tegevusi eraldi ning mitu korda välja kutsuda. Samuti on ühe alamprogrammi tööd küllalt hea testida. Järgnevalt võimalikult lihtne näide, kuidas omaette tegevuse saab alamprogrammiks välja tuua. Siin on selliseks tegevuseks korrutamine. Luuakse käsklus nimega Korruta, talle antakse ette kaks täisarvu nimedega arv1 ja arv2 ning välja oodatakse sealt ka tulema täisarv.

```
using System;
class Alamprogramm{
   static int Korruta(int arv1, int arv2){
      return arv1*arv2;
   }
   public static void Main(string[] arg){
      int a=4;
      int b=6;
      Console.WriteLine("{0} korda {1} on {2}", a, b, Korruta(a, b));
      Console.WriteLine(Korruta(3, 5));
   }
}
/*
C:\Projects\oma\naited>Alamprogramm
4 korda 6 on 24
15
*/
```

Ülesandeid

* Koosta alamprogramm kahe arvu keskmise leidmiseks

* Koosta alamprogramm etteantud arvu tärnide väljatrükiks. Katseta.

* Küsi inimeselt kolm arvu. Iga arvu puhul joonista vastav kogus tärne ekraanile

Massiivid

Kuna arvuti on mõeldud suure hulga andmetega ümber käimiseks, siis on programmeerimiskeelte juurde mõeldud ka vahendid nende andmehulkadega toimetamiseks. Kõige lihtsam ja levinum neist on massiiv. Iga elemendi poole saab tema järjekorranumbri abil pöörduda. Algul tuleb määrata, millisest tüübist andmeid massiivi pannakse ning mitu kohta elementide jaoks massiivis on. Järgnevas näites tehakse massiiv kolme täisarvu hoidmiseks. Kusjuures nagu C-programmeerimiskeele sugulastele kombeks on, hakatakse elemente lugema nullist. Nii et kolme massiivielemendi puhul on nende järjekorranumbrid 0, 1 ja 2. Tahtes väärtusi sisse kirjutada või massiivist lugeda, tuleb selleks kirja panna massiivi nimi (praeguse juhul m) ning selle taha kandiliste sulgude sisse järjekorranumber, millise elemendiga suhelda tahetakse.

```
using System;
class Massiiv1{
   public static void Main(string[] arg){
      int[] m=new int[3];
      m[0]=40;
      m[1]=48;
      m[2]=33;
      Console.WriteLine(m[1]);
   }
}
/*
C:\Projects\oma\naited>Massiiv1
48
*/
```

Tsükkel andmete kasutamiseks

Massiivi kõikide elementidega kiiresti suhtlemisel aitab tsükkel. Siin näide, kuidas arvutatakse massiivi elementidest summa. Algul võetakse üks abimuutuja nulliks ning siis liidetakse kõikide massiivi elementide väärtused sellele muutujale juurde. Avaldis summa+=m[i] on pikalt lahti kirjutatuna summa=summa+m[i] ning tähendab just olemasolevale väärtusele otsa liitmist. for-tsükli juures kõigepealt võetakse loendur (sageli kasutatakse tähte i) algul nulliks, sest nullist hakatakse massiivi elemente lugema. Jätkamistingimuses kontrollitakse, et on veel läbi käimata elemente ehk loendur on väiksem kui massiivi elementide arv (massiivinimi.Length). Pärast iga sammu suurendatakse loendurit (i++). Nõnda ongi summa käes.

```
using System;
```

```
class Massiiv2{
   public static void Main(string[] arg){
      int[] m=new int[3];
      m[0]=40;
      m[1]=48;
      m[2]=33;
      int summa=0;
      for(int i=0; i<m.Length; i++){
         summa+=m[i];
      }
      Console.WriteLine(summa);
   }
}
/*
C:\Projects\oma\naited>Massiiv2
121
*/
```

Massiiv ja alamprogramm

Nagu ennist kirjutatud, saab eraldiseisva toimingu kergesti omaette alamprogrammi tuua. Siin on nõnda eraldatud summa leidmine. Massiive saab alamprogrammile samuti ette anda nagu tavalisi muutujaid. Lihtsalt andmetüübi taga on kirjas massiivi tunnusena kandilised sulud.

```
using System;
class Massiiv3{
   static int LeiaSumma(int[] mas){
      int summa=0;
      for(int i=0; i<mas.Length; i++){
         summa+=mas[i];
      }
      return summa;
   }
   public static void Main(string[] arg){
      int[] m=new int[3]{40, 48, 33};
      int vastus=LeiaSumma(m);
      Console.WriteLine(vastus);
   }
}


/*
C:\Projects\oma\naited>Massiiv3
121
*/
```

Algväärtustamine, järjestamine

Kui massiivi elementide väärtused on kohe massiivi loomise ajal teada, siis saab nad loogelistes sulgudes komadega eraldatult kohe sisse kirjutada. Nii saab andmed lihtsalt lühemalt kirja panna.

Kui elemendid on lihtsa võrdlemise teel järjestatavad nagu näiteks täisarvud, siis piisab nende rittaseadmiseks klassi Array ainsast käsust Sort.

```
using System;
class Massiiv4{
   public static void Main(string[] arg){
      int[] m=new int[3]{40, 48, 33};
      Array.Sort(m);
      for(int i=0; i<m.Length; i++){
         Console.WriteLine(m[i]);
      }
   }
}


/*
C:\Projects\oma\naited>Massiiv4
33
40
48
*/
```

Osutid ja koopiad

Kui ühe hariliku täisarvulise muutuja väärtus omistada teisele, siis mõlemas muutujas on koopia samast väärtusest ning toimingud ühe muutujaga teise väärtust ei mõjuta. Massiividega ning tulevikus ka muude objektidega tuleb tähelepanelikum olla. Kui üks massiiv omistada teisele, siis tegelikult kopeeritakse vaid massiivi osuti, mõlema muutuja kaudu pääsetakse ligi tegelikult samadele andmetele. Nagu järgnevas näites: massiivid `m2` ja `m` näitavad samadele andmetele. Kui ühe muutuja kaudu andmeid muuta, siis muutuvad ka teise muutuja kaudu nähtavad andmed nagu väljatrüki juures paistab. Algselt on massiivi `m` ja `m2` elemendid 40, 48, 33. Pärast massiivi m elemendi number 1 muutmist 32ks, on ka massiivi `m2` elemendid muutunud - väärtusteks 40, 32, 33. Nõnda on suurte andmemassiivide juures teise muutuja tegemine andmete juurde pääsemiseks arvuti jaoks kerge ülesanne. Samas aga peab vaatama, et vajalikke andmeid kogemata ettevaatamatult ei muudaks.

```
int[] m=new int[3]{40, 48, 33};
int[] m2=m; //Viide samale massiivile
Tryki(m2);
m[1]=32;
Tryki(m2);
```

Kui soovida, et kaks algsetest andmetest pärit massiivi on üksteisest sõltumatud, siis tuleb teha algsest massiivist koopia (kloon).

```
int[] m3=(int[])m.Clone(); //Andmete koopia
m[1]=20;
Tryki(m3);
```

Pärast kloonimist muutused massiiviga m enam massiivi m3 väärtusi ei mõjuta.

Soovides massiivi tühjendada, aitab klassi Array käsklus Clear, mis täisarvude puhul kirjutab etteantud vahemikus (ehk praegusel juhul kogupikkuses, 0 ja Length-i vahel) täisarvude puhul väärtusteks nullid.

```
      Array.Clear(m3, 0, m3.Length); //Tühjendus
```

Massiivist andmete otsimiseks sobib käsklus `IndexOf`. Soovitud elemendi leidumise korral väljastatakse selle elemendi järjekorranumber. Otsitava puudumisel aga -1.

```
      Console.WriteLine(Array.IndexOf(m,33));
      Console.WriteLine(Array.IndexOf(m,17)); //puuduv element
using System;
class Massiiv5{
   static void Tryki(int[] mas){
      for(int i=0; i<mas.Length; i++){
         Console.WriteLine(mas[i]);
      }
      Console.WriteLine();
   }
   public static void Main(string[] arg){
      int[] m=new int[3]{40, 48, 33};
      int[] m2=m; //Viide samale massiivile
      Tryki(m2);
      m[1]=32;
      Tryki(m2);
      int[] m3=(int[])m.Clone(); //Andmete koopia
      m[1]=20;
      Tryki(m3);
      Array.Clear(m3, 0, m3.Length); //Tühjendus
      Tryki(m3);
      Console.WriteLine(Array.IndexOf(m,33));
      Console.WriteLine(Array.IndexOf(m,17)); //puuduv element
   }
}

/*
C:\Projects\oma\naited>Massiiv5
40
48
33
40
32
33
40
32
33
0
0
0
2
-1
*/
```

Massiiv alamprogrammi parameetrina

Massiivimuutuja omistamisel tekib võimalus kahe muutuja kaudu samadele andmetele ligi pääseda. See võimaldab luua alamprogramme, mis massiivi elementidega midagi peale hakkavad. Eelnevalt vaadeldud käsklus `Sort` tõstab massiivis elemendid kasvavasse järjekorda. Siin on näha omatehtud alamprogramm `KorrutaKahega`, mis massiivi kõikide elementide väärtused kahekordseks suurendab.

```
using System;
class Massiiv6{
   static void KorrutaKahega(int[] mas){
      for(int i=0; i<mas.Length; i++){
         mas[i]=mas[i]*2;
      }
   }
   static void Tryki(int[] mas){
      for(int i=0; i<mas.Length; i++){
         Console.WriteLine(mas[i]);
      }
      Console.WriteLine();
   }
   public static void Main(string[] arg){
      int[] m=new int[3]{40, 48, 33};
      KorrutaKahega(m);
      Tryki(m);
   }
}

/*
C:\Projects\oma\naited>Massiiv6
80
96
66
*/
```

foreach - tsükkel

Kui on vaja kogumi kõik elemendid läbi käia, kuid samas pole tähtis läbikäigu järjekord, siis sellisel puhul aitab all näites paistev `foreach`-tsükkel. Selle abi saab näiteks summa arvutamise juures pruukida.

```
using System;
class Massiiv7{
   public static void Main(string[] arg){
      int[] m=new int[3]{40, 48, 33};
      foreach(int arv in m){
        Console.WriteLine(arv);
      }
   }
}

/*
C:\Projects\oma\naited>Massiiv7
40
```

```
48
33
*/
```

Mitmemõõtmeline massiiv

Massiivis võib mõõtmeid olla märgatavalt rohkem kui üks. Kahemõõtmelist massiivi saab ette kujutada tabelina, milles on read ja veerud. Kolmemõõtmelise massiivi elemendid oleksid nagu tükid kuubis, mille asukoha saab määrata pikkuse, laiuse ja kõrguse kaudu. Harva läheb vaja enam kui kolme mõõdet - siis on sageli juba otstarbekam ja arusaadavam oma andmestruktuur ehitada. Kasutamine toimub aga nii, nagu allpool näites näha. Massiivi elementidega saab ümber käia nagu tavaliste muutujatega. `foreach`-tsükliga saab soovi korral läbi käia ka mitmemõõtmelise massiivi kõik elemendid.

```
using System;
class Massiiv8{
   public static void Main(string[] arg){
      int[,] m=new int[2,3]{
         {40, 48, 33},
         {17, 23, 36}
       };
      Console.WriteLine(m[0, 1]); //48
      Console.WriteLine("M66dete arv: "+m.Rank);
      Console.WriteLine("Ridade arv: "+m.GetLength(0));
      Console.WriteLine("Veergude arv: "+m.GetLength(1));
                            //elemente mõõtmes nr. 1
      int summa=0;
      foreach(int arv in m){
        summa+=arv;
      }
      Console.WriteLine("Summa: "+summa);
   }
}
/*
C:\Projects\oma\naited>Massiiv8
48
M66dete arv: 2
Ridade arv: 2
Veergude arv: 3
Summa: 197
*/
```

Ülesandeid

***** Küsi kasutaja käest viis arvu ning väljasta need tagurpidises järjekorras.

***** Loo alamprogramm massiivi väärtuste aritmeetilise keskmise leidmiseks. Katseta.

* Loo alamprogramm, mis suurendab kõiki massiivi elemente ühe võrra. Katseta.

***** Sorteeri massiiv ning väljasta selle keskmine element.

* Koosta kahemõõtmeline massiiv ning täida korrutustabeli väärtustega. Küsi massiivist kontrollimiseks väärtusi.

Käsud mitmes failis

Suuremate programmide puhul on täiesti loomulik, et kood jagatakse mitme faili vahel. Nii on hea jaotuse puhul kergem orienteeruda. Samuti on mugav terviklike tegevuste plokke muudesse programmidesse ümber tõsta, kui see peaks vajalikuks osutuma. Siin näites on kaks lihtsat arvutustehet omaette abivahendite klassis välja toodud.

```
class Abivahendid{
   public static int korruta(int a, int b){
      return a*b;
   }

   public static int liida(int a, int b){
      return a+b;
   }
}
```

Abivahendeid kasutav alamprogramm asub aga hoopis eraldi klassis ja failis.

```
using System;
class Abivahendiproov{
   public static void Main(string[] arg){
      Console.WriteLine(Abivahendid.korruta(3, 6));
   }
}
```

Kui tahta, et kompilaator vajalikud osad üles leiaks, tuleb kompileerimisel ette anda kõikide vajaminevate failide nimed.

```
C:\Projects\oma\naited>csc Abivahendid.cs Abivahendiproov.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
Lõpliku, exe-faili nimi määratakse vaikimisi faili järgi, kus oli
kirjas Main-meetod.
C:\Projects\oma\naited>Abivahendiproov
18
```

Ülesandeid

* Lisa käsklus täisarvude astendamiseks tsükli abil. Katseta

* Lisa kolmas fail paari tärnidest kujundeid joonistava funktsiooniga. Katseta peaprogrammis mõlema abifaili funktsioonide väljakutseid.

Tekst

Teksti koostamise, analüüsimise ja muutmisega puutuvad kokku enamik arvutiprogramme. Järgnevalt mõned näited, mille põhjal saab enamiku vajalikke tekstiga seotud toimetusi korda.

Teksti pikkuse saab kätte muutujast `Length`. Loetakse kokku kõik tekstis leiduvad sümbolid, kaasaarvatud tühikud.

Tekstist lõigu eraldamiseks sobib `Substring`. Esimese parameetrina olev arv näitab, mitmendast tähest hakatakse andmeid võtma, teine näitab, mitu tähte võetakse. `String` ehk sõne algab tähega number 0. Nii lugedes ongi sõna "tuli" algustäht järjekorranumbriga 5.

`IndexOf` võimaldab tekstis tähte või sõna leida. Leidmise korral väljastatakse leitud jupi algustähe järjekorranumber. Otsitava puudumise tulemusena väljastatakse -1.

```
using System;
class Tekst1{
  public static void Main(string[] arg){
    string s="Juku tuli kooli";
    Console.WriteLine("Pikkus: "+s.Length);
    Console.WriteLine(s.Substring(5, 4));
    Console.WriteLine("'tuli' kohal "+s.IndexOf("tuli"));
  }
}
/*
C:\Projects\oma\naited>Tekst1
Pikkus: 15
tuli
'tuli' kohal 5
*/
```

Muutmine

Teksti muutmiseks sobivad käsud `Insert` ja `Remove`. Esimene lisab soovitud kohale juurde etteantud teksti. Lausest "Juku tuli kooli" sai üheksandale kohale sõna " vara" lisamisel lause "Juku tuli vara kooli". `Remove` võimaldab sobivast kohast tähti välja võtta. Tehniliselt vaadates käsud `Insert` ja `Remove` ei muuda algses muutujas olevat teksti, vaid luuakse uus tekstiplokk mälus, mille poole on võimalik muutuja kaudu pöörduda. Muutujas `s` on endiselt "Juku tuli kooli". Vaid `s2` ja `s3` on uute väärtustega.

Tekstiga ümber käimiseks on ka mitukümmend muud käsklust, millega lähemat tutvust teha saab veebiaadressilt http://msdn2.microsoft.com/en-us/library/system.string_methods.aspx

Levinumatest on siin näha `StartsWith` alguse kontrolliks, `IndexOf` teksti otsinguks ja `Replace` asendamiseks. Aja jooksul läheb aga vaja tõenäoliselt meetodit `Compare` järjestuse kindlaks tegemisel, `EndsWith` lõpu kontrollimisel, `Trim` tühikute eemaldamisel, `ToUpper` ja `ToLower` suur- ja väiketähtedeks muutmisel ning `ToCharArray` juhul, kui soovitakse tervikliku teksti asemel tähemassiivi kasutada (näiteks tähtede massiliseks ümbertõstmiseks).

```
using System;
class Tekst2{
  public static void Main(string[] arg){
    string s="Juku tuli kooli";
    Console.WriteLine("Pikkus: "+s.Length);
    Console.WriteLine(s.Substring(5, 4));
    Console.WriteLine("'tuli' kohal "+s.IndexOf("tuli"));
    string s2=s.Insert(9, " vara");
    Console.WriteLine(s2);
    string s3=s.Remove(5, 5); //Kuuendast alates viis tähte
    Console.WriteLine(s3);
    if(s.StartsWith("Juku")){
       Console.WriteLine("Algab Jukuga");
    }
    if(s.IndexOf("kala")==-1){
       Console.WriteLine("Siin ei ole kala");
    }
    Console.WriteLine(s.Replace("tuli", "jooksis"));
  }
}
/*
C:\Projects\oma\naited>Tekst2
Pikkus: 15
tuli
'tuli' kohal 5
Juku tuli vara kooli
Juku kooli
Algab Jukuga
Siin ei ole kala
Juku jooksis kooli
*/
```

Tükeldamine

Pika teksti osadeks jaotamiseks on mitmetes keeltes olemas vastavad käsud ja objektid. Nii ka siin. Käsuga `Split` võib olemasoleva teksti määratud sümbolite koha pealt juppideks lõigata. Kõikidest üksikutest tükkidest moodustatakse massiiv. Siin näiteks on eraldajaks koma, mis tähemassiivi üksiku elemendina ette antakse. Aga nagu aimata võib, saab massiivi lisada ka rohkem eraldajaid.

Kui on põhjust massiiv uuesti üheks pikaks tekstiks kokku panna, siis selle tarbeks leiab käskluse `Join`.

```
using System;
class Tekst3{
  public static void Main(string[] arg){
    string s="Tallinn,Tartu,Narva";
    string[] linnad=s.Split(new char[]{','});
    foreach(string linn in linnad){
       Console.WriteLine(linn);
    }
    Console.WriteLine(String.Join("; ", linnad));
  }
}
/*
D:\kodu\0606\opikc#>Tekst3
```

```
Tallinn
Tartu
Narva
Tallinn; Tartu; Narva
*/
Ülesandeid
```

* Trüki inimese nime eelviimane täht

* Teata, kas sisestatud nimi algab A-ga

* Trüki sisestatud nimi välja suurtähtedega

* Teata, kas lauses leidub sõna "ja"

* Asenda olemasolu korral "ja" sõnaga "ning" ja teata asendusest

* Trüki välja lause kõige pikem sõna

Tekstifailid

Kui samu lähteandmeid on vaja korduvalt kasutada, siis on neid igakordse sissetoksimise asemel mugavam sisse lugeda failist. Samuti on suuremate andmemahtude korral hea, kui sisendi ja väljundi andmed jäävad alles ka masina väljalülitamise järel. Keerukama struktuuriga andmete ning mitme üheaegse kasutaja korral (näiteks veebirakendustes) kasutatakse enamasti andmebaasi. Käsurea- või iseseisva graafilise rakenduse puhul on tekstifail aga lihtne ja mugav moodus andmete hoidmiseks. Samuti on ta tarvilik juhul, kui juba pruugitakse eelnevalt tekstifaili kujul olevaid andmeid.

Kirjutamine

Andmete faili saatmiseks tuleb kõigepealt moodustada voog etteantud nimega faili kirjutamiseks. Edasine trükk toimub juba sarnaselt ekraaniväljundiga `Write` ning `WriteLine` käskude abil. Lõppu tuleb kindlasti lisada käsklus `Close`, et teataks hoolitseda andmete füüsilise kettale jõudmise eest ning antakse fail vabalt kasutatavaks ka ülejäänud programmide jaoks. Siinse näite tulemusena tekib käivituskataloogi lihtne fail nimega "inimesed.txt", kuhu kirjutatakse kaks nime. Tekstiekraanil enesel pole käivitamise järel midagi näha - aga see veel ei tähenda, nagu programm midagi ei teeks. Lihtsalt tema töö tulemus jõuab loodavasse faili.

```
using System;
using System.IO;
class Failikirjutus{
  public static void Main(string[] arg){
            FileStream f = new FileStream("inimesed.txt",
                           FileMode.Create, FileAccess.Write);
            StreamWriter valja = new StreamWriter(f);
            valja.WriteLine("Juku");
            valja.WriteLine("Kati");
```

```
            valja.Close();
  }
}
```

Lisamine

Kui FileMode.Create asendada seadega FileMode.Append, siis jäävad kirjutades vanad andmed alles. Uued read lihtsalt lisatakse olemasoleva teksti lõppu. Sellist lisamist läheb tarvis näiteks sündmuste logi kirjutamise juures.

```
using System;
using System.IO;
class Faililisamine{
  public static void Main(string[] arg){
            FileStream f = new FileStream("inimesed.txt",
                              FileMode.Append, FileAccess.Write);
            StreamWriter valja = new StreamWriter(f);
            valja.WriteLine("Siim");
            valja.WriteLine("Sass");
            valja.Close();
  }
}
```

Lugemine

Faili lugemisel on vood teistpidi. `Create` ja `Write` asemel on `Open` ja `Read`. Ning `StreamWriter`i asemel `StreamReader`. Voost tuleva iga `ReadLine` tulemusena antakse üks tekstirida failist. Kui faili andmed lõppesid, saabub `ReadLine` käsu tulemusena tühiväärtus null. Selle järgi saab programmeerija otsustada, et fail sai läbi. Tahtes faili keerukamalt töödelda - üksikute tähtede poole pöörduda või tervikuna ette võtta - selleks tuleb juba failiga seotud õpetusi ise lähemalt uurida.

```
using System;
using System.IO;
class Faililugemine{
   public static void Main(string[] arg){
         FileStream f = new FileStream("inimesed.txt",
                        FileMode.Open, FileAccess.Read);
         StreamReader sisse=new StreamReader(f);
         string rida=sisse.ReadLine();
         while(rida!=null){
            Console.WriteLine(rida);
            rida=sisse.ReadLine();
         }
   }
}
/*
C:\Projects\oma\naited>Faililugemine.exe
Juku
Kati
*/
```

Ülesandeid

* Tekita programmi abil fail, milles oleksid arvud ja nende ruudud ühest kahekümneni

* Tekstifaili igal real on müüdud kauba hind. Arvuta programmi abil nende hindade summa.

* Iga hinna kõrval on ka selle hinnaga müüdud kauba kogus. Korruta igal real hind kogusega ning liida lõpuks summad kokku.

* Võrreldes eelmise ülesandega kirjuta teise faili igale reale esimese faili vastaval real oleva hinna ja koguse korrutis.

Juhuarv

Kui soovida, et arvuti samade algandmete abil erinevalt käituks, tulevad appi juhuarvud. Nende abil saab kasutajale ette anda juhusliku tervituse, muuta soovi järgi pildi värvi, või näiteks kontrollida loodud funktsiooni toimimist mitmesuguste väärtuste juures. Kõigi nende erinevate väljundite aluseks on arvuti poolt loodud juhuarvud. Neid aitab saada nimeruumi `System` klassi `Random` eksemplar. Reaalarvu saamiseks on käsklus `NextDouble`. Kui soovida mõnda muud vahemikku kui nullist üheni, tuleb saadud arv lihtsalt soovitud suurusega läbi korrutada. Ühtlase jaotuse asemele normaal- või mõne muu jaotuse saamiseks tuleb mõnevõrra enam vaeva näha - juhul kui see peaks tarvilikuks osutuma.

Täisarv luuakse käsuga `Next`, andes ette ülempiiri, soovi korral ka alampiiri. Ning sobiva nime, anekdoodi või terviku saamiseks tuleb olemasolevad valitavad objektid paigutada massiivi, leida juhuarv massiivi elementide arvu piires ning võibki sobiva väärtuse välja võtta.

```
using System;
public class Juhuarv1{
   public static void Main(string[] arg){
      Random r=new Random();
      Console.WriteLine(r.NextDouble()); //Nullist üheni
      Console.WriteLine(r.Next(20)); //Täisarv alla 20
      Console.WriteLine(r.Next(50, 100)); //Viiekümnest sajani
      string[] nimed={"Juku", "Kati", "Mati"};
      Console.WriteLine(nimed[r.Next(nimed.Length)]); //Juhuslik nimi
   }
}
/*
D:\kodu\0606\opikc#>Juhuarv1
0,74339002358885
11
95
Kati
*/
```

Ülesandeid

* Trüki juhuslike teguritega korrutamisülesanne

* Kontrolli, kas kasutaja pakutud vastus oli õige

* Sõltuvalt vastuse õigsusest lase arvutil pakkuda olemasolevate hulgast valitud kiitev või julgustav kommentaar.

Omaloodud andmestruktuur

Standardandmetüüpe on .NET raamistikus kätte saada palju. Klasside arvu loetakse tuhandetes. Sellegipoolest juhtub oma rakenduste puhul olukordi, kus tuleb toimetada andmetega, mille hoidmiseks mugavat moodust pole olemas. Või siis on keegi kusagil selle küll loonud, aga lihtsalt ei leia üles. Harilike muutujate ja massiivide abil saab küll kõike arvutis ettekujutatavat hoida. Vahel aga on mugavam, kui pidevalt korduvate sarnaste andmete hoidmiseks luuakse eraldi andmetüüp. Siis on teada, et kokku kuuluvad andmed püsivad kindlalt ühes kohas koos ning pole nii suurt muret, et näiteks kahe firma andmed omavahel segamini võiksid minna.

Järgnevas näites kirjeldatakse selliseks omaette andmestruktuuriks punkt tasandil, kaks täisarvulist muutujat asukohti määramas.

```
struct Punkt{
    public int x;
    public int y;
}
```

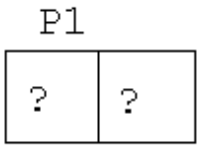
Kui edaspidi programmis kirjutatakse

```
Punkt p1
```

siis on teada, et `p1`-nimelisel eksemplaril on olemas `x` ja `y` väärtused ning neid on võimalik vaadata ja muuta. `struct`-iga kirjeldatud andmestruktuuri põhjal loodud muutuja omistamisel teisele sama tüüpi muutujale kopeeritakse väärtused ilusti ära. Nii nagu ühe täisarvulise muutuja väärtuse omistamisel teisele tekib sellest arvust koopia uude mälupiirkonda, nii ka struktuurina loodud `Punkt`i omistamisel teisele `Punkt`ile on mälus kaks eraldi ja sõltumatut piirkonda, mõlemal x-i ja y-i teise x-i ja y-iga samasugused. Õieti ei peakski sellist kopeerimist eraldi rõhutama - tundub ju nõnda omistamisel kopeerimine täiesti loomulik. Märkus on toodud lihtsalt tähelepanu äratamiseks. Kui tulevikus ei kasutata andmetüübi loomisel mitte sõna struct, vaid sõna class, siis käitutakse omistamisel mõnevõrra teisiti.

Nüüd aga programmi töö kohta seletus.

Punkt p1;



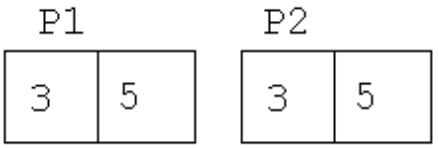
Luuakse muutuja nimega p1. Tal on mäluväljad nii x-i kui y-i jaoks.

p1.x=3;

p1.y=5;

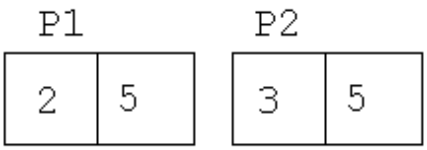
Väljadele antakse väärtused

Punkt p2=p1;



Luuakse muutuja nimega p2. Ka temal on mäluväljad nii x-i kui y-i jaoks. Muutuja p1 x-i väärtus kopeeritakse muutuja p2 x-i väärtuseks ning samuti ka y-iga. Praeguseks on siis mälus kahte kohta kirjutatud kolm ning kahte kohta viis. Edasi muudetakse esimese punkti x-koordinaat kaheks. Teise oma aga jääb endiselt kolmeks.

p1.x=2;



Et kui nüüd teise punkti koordinaadid välja trükkida, siis tulevad sealt rõõmsasti 3 ja 5.

```
          Console.WriteLine(p2.x+" "+p2.y);
```

Võrreldes varasemate näidetega on koodi juurde tekkinud sõna `namespace`. Varasemate väiksemate programmide puhul mõtles kompilaator neile juurde nimetu ja nähtamatu vaikimisi nimeruumi. Kui aga klasse või struktuure on rohkem, on viisakas koos kasutatavad klassid teise nimeruumi eraldada. Sel juhul pole nii palju karta, et keegi teine oma klassid meie omadega sarnaselt nimetaks ning nad meie omadega tülli läheksid. Kui ka klassi nimi juhtub sama olema, aga nimeruum on erinev, siis saab kompilaator aru, et need on erinevad ning ei sega üksteist.

Töötav kood tervikuna.

```
using System;
namespace Punktid{
   struct Punkt{
       public int x;
       public int y;
   }
   class Punktiproov{
     public static void Main(string[] arg){
         Punkt p1;
         p1.x=3;
         p1.y=5;
         Punkt p2=p1;  //Väärtused kopeeritakse
         p1.x=2;
         Console.WriteLine(p2.x+" "+p2.y);
     }
   }
}
/*
C:\Projects\oma\naited>Punktid
3 5
*/
```

Punktimassiiv

Oma andmetüüpi on enamasti põhjust luua juhul, kui seda tüüpi andmeid on rohkem kui paar-kolm eksemplari. Suurema hulga samatüübiliste andmete jaoks kasutatakse sageli massiivi. Nii ka omaloodud andmetüübi puhul.

```
          Punkt[] pd=new Punkt[10];
```

teeb punktidest kümme eksemplari järjenumbritega nullist üheksani. Ning ilusti järjenumbri järgi elemendi poole pöördudes saab sinna nii väärtusi panna kui küsida. Tsükli abil pannakse iga elemendi y väärtuseks tema järjekorranumbri ruut. Ning väljatrükil

```
          Console.WriteLine(pd[4].y);
```

trükitakse rõõmsasti ekraanile 16.

```
using System;
namespace Punktid2{
   struct Punkt{
       public int x;
       public int y;
   }
   class Punktimassiiv{
     public static void Main(string[] arg){
         Punkt[] pd=new Punkt[10]; //mälu kohe olemas
         for(int i=0; i<10; i++){
           pd[i].x=i;
           pd[i].y=i*i;
         }
         Console.WriteLine(pd[4].y);
     }
   }
}
/*
C:\Projects\oma\naited>Punktid2
16
*/
```

Ülesandeid

* Koosta struktuur riidelappide andmete hoidmiseks: pikkus, laius, toon

* Katseta loodud andmetüüpi paari eksemplariga.

* Loo lappidest väike massiiv, algväärtusta juhuarvude abil.

* Trüki välja lappide andmed, mille mõlemad küljepikkused on vähemalt 10 cm.

Edasijõudnute osa: Objektorienteeritud programmeerimine

Tutvustus

struct-lausega loodud kirjed on mõeldud põhiliselt andmete hoidmiseks ning vajadusel üksikute andmete (nt. sünniaasta abil vanuse) välja arvutamiseks. Toimingud andmetega jäävad enamjaolt kirjest väljapool paikneva programmi ülesandeks. Objektide puhul aga püütakse enamik konkreetse objektitüübiga seotud toiminguid ühise kesta sisse koondada. Piir kirjete ja objektide vahel on mõnevõrra hägune ning mõnes keeles (nt. Java) polegi kirjete jaoks eraldi keelekäsklust olemas. Samas on siiski hea eri lähenemised lahus hoida.

Traditsioonilise objektorienteeritud programmeerimise juures pole eksemplari muutujatele sugugi võimalik otse väljastpoolt ligi saada. Samuti on vaid mõned alamprogrammid teistele objektidele vabalt kasutavad, küllalt palju vahendeid võib olla loodud vaid objekti enese toimimise tarbeks. Selline kapseldamine aitab suuremate programmide puhul järge pidada, et vaid ühe teemaga seotud ning teemadevahelised lõigud üksteist segama ei hakkaks. Lisaks väliste käskude vähemast arvust tulevale lihtsusele lubab muutujate ja alamprogrammide varjamine teiste objektide eest hiljem oma objekti sisemist ülesehitust muuta ilma, et muu programmi osa sellest häiritud saaks. Selline muutmisvõimalus on aga hästi tänuväärne olukorras, kus tegijaid on palju, ja samu objekte kasutatakse tulevikus teisteski programmides.

Järgnevas näites on punkt loodud klassina ehk objektitüübina. Koordinaadid `x` ja `y` on privaatse ligipääsuga, neid saab kasutada vaid sama klassi meetodites. Väärtuste küsimiseks on eraldi käsklused `GetX` ja `GetY`. Esimese hooga võib tunduda selline väärtuse küsimise peitmine imelik. Aga kui tulevikus näiteks soovitakse teha statistikat, et mitu korda on küsitud `x`-i ja mitu korda `y`-i väärtust, siis alamprogrammiga andmete küsimise puhul saab selle loenduse kergesti paika panna. Muutuja puhul aga ei ole nõnda tavaprogrammeerimise vahenditega võimalik. Kuna siin on lubatud punkti asukoha määramine vaid eksemplari loomisel, siis kord antud väärtusi enam muuta ei saa, sest `Punkt`i juures pole lihtsalt vastavat käsklust. Klassiga samanimelist väljastustüübita käsklust nimetatakse konstruktoriks. See käivitatakse vaid üks kord - eksemplari loomisel - ning sinna sisse pannakse tavaliselt algväärtustamisega seotud toimingud. Kui näiteks algväärtustamisel seada sisse piirang, et koordinaatide väärtused peavad jääma nulli ja saja vahele, siis pole võimalik muus piirkonnas punkti luua. Sedasi õnnestub objektina luua küllalt keerukaid süsteeme, mis oskavad "iseenese eest hoolitseda" ning millel saab lihtsalt käsklusi ehk meetodeid välja kutsuda.

```
using System;
namespace Punktid3{
   class Punkt{
       private int x;
       private int y;
       public Punkt(int ux, int uy){
         x=ux; y=uy;
```

```
        }
        public int GetX(){
          return x;
        }
        public int GetY(){
          return y;
        }
        public double KaugusNullist(){
          return Math.Sqrt(x*x+y*y);
        }
    }
    class Punktiproov{
      public static void Main(string[] arg){
          Punkt p1=new Punkt(3, 4);
          Console.WriteLine(p1.GetX());
          Console.WriteLine(p1.KaugusNullist());
      }
    }
}
/*
D:\kodu\0606\opikc#>Punktid3
3
5
*/
```

Ülesandeid

* Koosta klass riidelapi andmete hoidmiseks: pikkus, laius, toon

* Lisa käsklus lapi andmete väljatrükiks

* Lisa käsklus lapi pindala arvutamiseks

* Lisa meetod (alamprogramm) lapi poolitamiseks: pikem külg tehakse poole lühemaks.

* Poolitamise meetod lisaks algse lapi poolitamisele väljastab ka uue samasuguse algsest poole väiksema eksemplari.

* Lisa teine poolitusmeetod, kus saab määrata, mitme protsendi peale lõigatakse pikem külg

Klassimuutuja

Klassi juurde korjatakse võimalusel kokku kõik vastavat tüüpi objektidega tehtavad toimingud ja andmed. Enamasti kuuluvad andmed isendite ehk eksemplaride juurde, kuid mitte alati. Näiteks tüüpiliselt - loodud punktide arvu loendur on küll punktidega sisuliselt seotud, kuid pole sugugi ühe konkreetse punkti omadus. Punktide arv on olemas ka juhul, kui ühtegi reaalset punkti eksemplari pole veel loodud. Sellisel juhul on punktide arv lihtsalt null. Samuti on punktide arv olemas juhul, kui punkte on loodud palju. Lihtsalt sel juhul on loenduri väärtus suurem. Sellise klassi ja mitte isendiga seotud muutuja ette pannakse sõna `static`.

```
static int loendur=0;
```

Teine asi on punkti järjekorranumber - see on iga punkti juures kindel arv, mis antakse punktile tema loomise hetkel ning mida siinse programmi puhul enam hiljem muuta ei saa.

```
private int nr;
nr=++loendur;
```

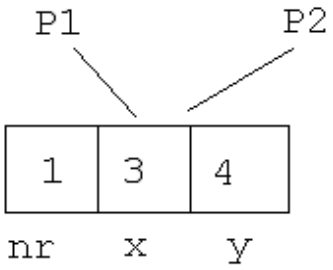
tähendab, et kõigepealt suurendatakse loendurit (olemasolevate punktide arvu) ühe võrra ning saadud arv pannakse uue loodud punkti järjekorranumbriks.

Osuti, omistamine

Klassi eksemplaride puhul toimub omistamine teisiti kui struktuuri eksemplaride puhul. Struktuuri juures tehti lihtsalt väljade väärtustest koopia ja edasi elas kumbki eksemplar oma elu teisest sõltumata. Kui nüüd aga teha omistamised

```
Punkt p1=new Punkt(3, 4);
Punkt p2=p1; //Kasutab sama mälupiirkonda
```

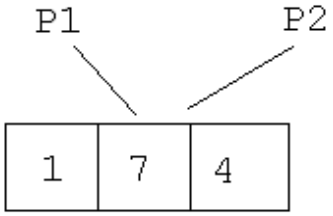
siis on mälus tegelikult koht vaid ühe punkti andmete jaoks, muutujate p1 ning p2 kaudu pääseb lihtsalt ligi samadele andmetele.



Nii et kui ühe muutuja kaudu andmeid muuta:

```
p1.SetX(7);
```

siis muutuvad andmed ka teise märksõna alt paistvas vaates.



Ehk siis ükskõik, kas andmete poole pöörduda `p1` või `p2` kaudu – ikka saan tegelikult samad väärtused.

```
Console.WriteLine(p2.GetNr()+" "+p2.GetX()+" "+p2.GetY());
```

trükib välja

```
1 7 4
```

ehkki esialgu olid koordinaatideks 3 ja 4 ning `p2` kohe deklareerimisel polnud sugugi algne koht esimese punkti andmetele ligi pääsemiseks.

Punkt järjekorranumbriga 2 on alles `p3`, sest tema loomisel käivitati uuesti Punkti konstruktor – loodi uus eksemplar, mille käigus suurendati loendurit ning anti uued kohad andmete mälus hoidmiseks.

Nüüd siis eelnevate seletuste kood tervikuna.

```
using System;
namespace Punktid4{
   class Punkt{
       static int loendur=0;
       private int nr;
       private int x;
       private int y;
       public Punkt(int ux, int uy){
         x=ux; y=uy;
         nr=++loendur;
       }
       public int GetX(){
         return x;
       }
       public int GetY(){
         return y;
       }
       public int GetNr(){
         return nr;
       }
       public void SetX(int ux){
         x=ux;
       }
```

```
        public void SetY(int uy){
          y=uy;
        }
        public double KaugusNullist(){
          return Math.Sqrt(x*x+y*y);
        }
    }
    class Punktiproov{
      public static void Main(string[] arg){
          Punkt p1=new Punkt(3, 4);
          Punkt p2=p1; //Kasutab sama mälupiirkonda
          p1.SetX(7);
          Console.WriteLine(p2.GetNr()+" "+p2.GetX()+" "+p2.GetY());
          Punkt p3=new Punkt(77, 32); //Punkt järgmise järjekorranumbriga
          Console.WriteLine(p3.GetNr());
      }
    }
}

/*
C:\Projects\oma\naited>Punktid4
1 7 4
2
*/
```

Punktimassiiv

Nii nagu üksiku muutuja juures, nii ka massiivi puhul tuleb (erinevalt structist) igale uuele klassi eksemplarile new-käsuga mälu anda.

```
          Punkt[] pd=new Punkt[10];
```

loob vaid kümme mälupesa, mis on võimelised näitama Punkt-tüüpi objektile. Samas punktiobjekte pole selle käsu peale veel ühtegi. Ehk kui keegi sooviks näiteks pd[3] olematu eksemplariga midagi teha, siis antaks veateade.

Alles siis, kui tsüklis luuakse iga ringi juures uus Punkti eksemplar ning pannakse massiivi element sellele näitama, on võimalik iga massiivi elemendiga kui Punkti eksemplariga ringi käia.

```
          for(int i=0; i<pd.Length; i++){
             pd[i]=new Punkt(i, i*i);
          }
```

Näiteks küsida konkreetse elemendi väärtust.

```
          Console.WriteLine(pd[4].GetY());
using System;
namespace Punktid5{
   class Punkt{
       static int loendur=0;
```

```
        private int nr;
        private int x;
        private int y;
        public Punkt(int ux, int uy){
          x=ux; y=uy;
          nr=++loendur;
        }
        public int GetX(){
          return x;
        }
        public int GetY(){
          return y;
        }
        public int GetNr(){
          return nr;
        }
    }
    class Punktiproov{
      public static void Main(string[] arg){
          Punkt[] pd=new Punkt[10];
          for(int i=0; i<pd.Length; i++){
             pd[i]=new Punkt(i, i*i);
          }
          Console.WriteLine(pd[4].GetY());
      }
    }
}
/*
C:\Projects\oma\naited>Punktid5
16
*/
```

Ülesandeid

* Koosta klass riidelapi andmete hoidmiseks: pikkus, laius, toon

* Koosta riidelappidest massiiv.

* Koosta uus lapimassiiv, kuhu pannakse osa eelmisest massiivist pärit lappe ja osa muid lappe (kaltsukott).

* Arvuta mõlema massiivi puhul välja seal leiduvate lappide pindalade summa.

* Koosta riidelapi klassile staatiline käsklus näitamaks loodud riidelappide keskmist pindala. Lappide puudumisel väärtuseks -1. Lisa vajalikud staatilised muutujad (lappide arv, kogupindala)

* Koosta klass isikukoodi andmete hoidmiseks

* Lisa käsklus sünnikuupäeva küsimiseks

* Lisa käsklus sünnikuu küsimiseks sõnana

* Lisa käsklus sünniaasta väljastamiseks ka sajandit arvestades

* Kontrolli isikukoodi objekti tegemisel koodi pikkust

* Kontrolli isikukoodi kontrollisumma õigsust vastavalt järgnevale algoritmile:

Isikukoodi kontrolli algoritm

Isikukoodis peavad kõik sugu ning kuupäeva tähistavad väärtused olema

võimalikud ning viimane kontrollnumber arvutatakse järgneva algoritmi järgi:

liidetakse kokku esimese üheksa numbri korrutised igale arvule vastava

järjekorranumbriga, kümnenda numbri puhul ühega ning leitakse saadud summa

jääk jagamisel 11-ga. Kui jääk on võrdne kümnega, siis tehakse arvutus

uuesti ning võetakse teguriteks vastavalt 3, 4, 5, 6, 7, 8, 9, 1, 2, 3.

Näide: isikukoodi 37605030299 kontroll. Summa =

1*3 + 2*7 + 3*6 + 4*0 + 5*5 + 6*0 + 7*3 + 8*0 + 9*2 + 1*9 = 108

108 jääk jagamisel 11-ga on 9 => isikukoodi viimane number peab olema

üheksa.

Dokumenteerivad kommentaarid

Dokumentatsiooni loomiseks on üks võimalus panna kolme kaldkriipsuga märgistatud kommentaarid koodi sisse, kommenteeritava ploki või muutuja ette. Sellised kommentaarid oskab kompilaator sobiva võtme järgi välja, eraldi kommentaaride faili korjata. Kommentaariks sobib xml-märgenditega piiratud tekst. Tavalisimaks elemendiks on <summary>, kuid eraldi on olemas ka kokkuleppelised käsklused parameetrite, versioonide jm. jaoks.

Readonly

Võtmesõna `readonly` muutuja ees tähendab, et sinna võib väärtuse omistada vaid ühe korra. Olgu siis muutuja kirjeldamisel või konstruktoris. Selliste muutujate puhul hoolitseb juba kompilaator, et poleks võimalik kirjutada käsklusi, mis readonlyga kaitstud mäluväljade väärtusi muudavad.

Näide

```
using System;
namespace Punktid6{
   /// <summary>
   ///   Tasandi punkti andmete hoidmine
   /// </summary>
   class Punkt{
       /// <summary>
       ///    Muutuja ainult lugemiseks.
       ///    Andmed sisestatavad vaid konstruktoris.
       /// </summary>
       private readonly int x;
       private readonly int y;
       /// <summary>
       ///    Algandmed punkti loomisel kindlasti vajalikud
       /// </summary>
       public Punkt(int ux, int uy){
         x=ux; y=uy;
       }
       public int GetX(){
         return x;
       }
       public int GetY(){
         return y;
       }
       /// <summary>
       ///   Kaugus arvutatakse Pythagorase teoreemi järgi.
       /// </summary>
       public double KaugusNullist(){
         return Math.Sqrt(x*x+y*y);
       }
   }
   /// <summary>
   ///    Eraldi klass punkti katsetamiseks.
   /// </summary>
   class Punktiproov{
     public static void Main(string[] arg){
         Punkt p1=new Punkt(3, 4);
         Console.WriteLine(p1.KaugusNullist());
     }
   }
}
```

Kompileerimine

Kompileerides saab siis /doc võtmega anda ette failinime, kuhu kirjutatakse kommentaarid.

```
/*
D:\kodu\0606\opikc#>csc Punktid6.cs /doc:Punktid6.xml
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
D:\kodu\0606\opikc#>Punktid6
5
*/
```

Kommentaarifail

Edasi on juba vastavalt kasutaja soovile - kas ta loeb tekkinud XML-faili lihtsa tekstiredaktoriga, mõne eraldi XML-lugejaga (nt. vastavate võimetega veebibrauser) või siis kirjutab juurde XSL-lehe andmete omale sobivale kujule muundamiseks.

Väike ülevaade tekkinud XML-failist. Esimene rida on ühine kõigile XML-vormingus dokumentidele.

```
<?xml version="1.0"?>
```

Järgnevaks juurelemendiks on doc, mille sisse kõik ülejäänud andmed mahuvad.

```
<doc>
```

Assembly tähendab kokkukompileeritud üksust. Siin on ta laiendiks .exe, mõnel juhul võib olla aga ka .dll

```
    <assembly>
        <name>Punktid6</name>
    </assembly>
```

Ning edasi juba tulevadki klassi, muutuja, konstruktori ja meetodi töö kirjeldused.

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>Punktid6</name>
    </assembly>
    <members>
        <member name="T:Punktid6.Punkt">
            <summary>
              Tasandi punkti andmete hoidmine
            </summary>
        </member>
        <member name="F:Punktid6.Punkt.x">
            <summary>
               Muutuja ainult lugemiseks.
               Andmed sisestatavad vaid konstruktoris.
            </summary>
        </member>
        <member name="M:Punktid6.Punkt.#ctor(System.Int32,System.Int32)">
            <summary>
               Algandmed punkti loomisel kindlasti vajalikud
            </summary>
        </member>
        <member name="M:Punktid6.Punkt.KaugusNullist">
            <summary>
              Kaugus arvutatakse Pythagorase teoreemi jC¤rgi.
            </summary>
```

```
        </member>
        <member name="T:Punktid6.Punktiproov">
            <summary>
               Eraldi klass punkti katsetamiseks.
            </summary>
        </member>
    </members>
</doc>
```

Ülesandeid

* Koosta klass riidelapi andmete hoidmiseks, lisa kommentaarid.

* Tutvu kompileerimisel loodud kommentaaride failiga.

Pärilus

Ajapikku on objektorienteeritud programmeerimiskeelte juurde lisatud mitmesuguseid täiustusi ja võimalusi. Pärilus (inheritance) on mingil moel enamike objektorienteeritud keelte küljes olemas. Nii ka end suhteliselt arenenuks keeleks pidava C# juures.

Objektidega toimetamisel ning pärilusel sealhulgas on vähemalt kaks eesmärki. Püütakse lihtsustada koodi kohandamist. Samuti võimaldab pärilus vältida sama tööd tegeva koodi kopeerimist, lubades samu käsklusi kasutada mitmel pool.

Päriluse abil püütakse programmeerimiskeele objektitüüpide omavahelisi suhteid lähendada inimeste igapäevaselt tajutavale. Üheksakümnendate aastate algul ilmunud objektorienteeritud programmeerimist tutvustavas õpikus oli ilus näide: "sebra on nagu hobune, kellel on triibud". Sarnane tuttava kaudu uue tüübi kokkupanek ongi päriluse põhisisu. Midagi lisatakse, midagi muudetakse, vahel harva ka kaotatakse. Ning märkamatult muutubki hobune sebraks või tühi paneel tekstiväljaks.

Nõnda õnnestub olemasolevate (pool)valmis tükkide abil omale sobiv komplekt kokku panna. Teiselt poolt võidakse objektide päriluspuu ehitada ka oludes, kus kõik programmi osad on enese määrata. Sellisel juhul pole küll eesmärk omale üks ja parim tüüp kokku panna, vaid otsitakse ühisosi, mille puhul on võimalik valminud tüübid gruppidesse jaotada ning nende gruppidega vajadusel koos midagi ette võtta. Tüüpilise näitena pole teatripiletit müües vaja teada inimese sugu ja ametit. Küll aga on balletirolli juures mõlemad andmed tähtsad. Tavaelus tundub loomulikuna, et eriomadusi arvestatakse vaid kohtades, kus need on vajalikud ning muul juhul aetakse vaid üldiste tunnustega läbi. Haruharva tuleb eraldi rõhutada, et "meie teatrisse tohivad vaatajaks tulla inimesed sõltumata usutunnistusest ja nahavärvist". Arvuti juures tuleb aga tasemed selgelt välja tuua. Igal keerukuse astmel tuleb määrata, millised oskused ja omadused sinna juurde kuuluvad ning millise rolli jaoks millisel tasemel oskuste komplekti vaja on. Keerukaks kiskunud seletuste kõrvale selgitused näidete abil.

Päriluseta näide

Alustame inimese klassiga, kus igasugune pärilus puudub. Üks klass oma muutuja, konstruktori ja meetodiga ning testprogrammis saab tema võimalusi katsetada.

```
using System;
namespace Parilus1{
 class Inimene{
   protected int vanus;
   public Inimene(int uvanus){
      vanus=uvanus;
   }
   public void YtleVanus(){
      Console.WriteLine("Minu vanus on "+vanus+" aastat");
   }
 }
 class InimTest{
  public static void Main(string[] arg){
     Inimene inim1=new Inimene(13);
     inim1.YtleVanus();
  }
 }
}
/*
C:\Projects\oma\naited>Parilus1
Minu vanus on 13 aastat
*/
```

Alamklass

Edasi juba juures inimese alamklass (subclass) `Modell`, kel on olemas kõik inimese omadused (ehk siis selle programmi puhul võime oma vanust öelda), kuid juures käsklus enese esitlemiseks. Et esitlemine koosneb vanuse ja ümbermõõdu teatamisest, on see juba esitlemiskäskluse sisse kirjutatud.

Klass `Inimene` on `Modelli` suhtes ülemklassiks (superclass, base class, parent class). C# puhul on igal klassil alati täpselt üks ülemklass. Kui muud pole määratud, siis kasutatakse selleks vaikimisi nimeruumi `System` klassi `Object`. Muul juhul aga on ülemklass kirjas klassikirjelduse päriluse osas. Nii tekibki klasside puu, mis algab klassist `Object`.

Kui klassil konstruktor puudub, siis loob kompilaator vaikimisi tühja konstruktori, millel pole parameetreid ja mis ka midagi ei tee. Inimese puhul siis näiteks

```
   public Inimene(){}
```

Kui aga vähemalt üks programmeerija loodud konstruktor on olemas, siis seda nähtamatut konstruktorit ei tehta. Päriluse puhul kutsutakse alamklassi eksemplari loomisel alati välja ülemklassi konstruktor. Vaikimisi võtab kompilaator selleks ülemklassi parameetritega konstruktori. Kui see aga puudub või soovitakse käivitada mõnda muud, siis tuleb sobiva konstruktori väljakutse alamklassi juures ära märkida. Siin märgitakse näiteks `Modelli`

loomise juures, et `Modelli` isendi loomise juures tehakse kõigepealt valmis baasklassi (inimese) isend, kellele siis `Modelli` enese konstruktoris vajalikud lisandused juurde pannakse. Ülemklassi konstruktori määramine on kohe `Modelli` konstruktori juures. Pärast koolonit olev `base(vanus)` ütleb, et kasutatagu inimese puhul seda konstruktorit, kus tuleb täisarvuline vanus kohe ette öelda.

```
   public Modell(int vanus, int uymberm66t):base(vanus){
      ymberm66t=uymberm66t;
   }
```

Ehkki praegu tegelikult muud võimalust polnudki, tuleb see ikkagi arvutile ette öelda.

```
using System;
namespace Parilus2{
 class Inimene{
   protected int vanus;
   public Inimene(int uvanus){
      vanus=uvanus;
   }
   public void YtleVanus(){
      Console.WriteLine("Minu vanus on "+vanus+" aastat");
   }
 }
 class Modell:Inimene {
   protected int ymberm66t;
   public Modell(int vanus, int uymberm66t):base(vanus){
      ymberm66t=uymberm66t;
   }
   public void Esitle(){
      YtleVanus();
      Console.WriteLine("Mu ymberm66duks on "+ymberm66t+" sentimeetrit");
   }
 }
 class InimTest{
  public static void Main(string[] arg){
     Modell m=new Modell(20, 90);
     m.Esitle();
  }
 }
}
/*
C:\Projects\oma\naited>Parilus2
Minu vanus on 20 aastat
Mu ymberm66duks on 90 sentimeetrit
*/
```

Ülesandeid

* Lisa Inimesele pikkuse väli.

* Pikkus tuleb sisestada konstruktoris sarnaselt vanusele.

* Modelli käsklus Esitle teatab eraldi lausena ka pikkuse.

* Omista Modell Inimese tüüpi muutujale. Küsi sealtkaudu vanus.

* Koosta inimeste massiiv. Lisa sinna nii Modelle kui tavalisi inimesi. Küsi kõikide vanused.

* Lisa Inimesele int-tagastusväärtusega käsklus pikkuse väljastamiseks.

* Lisa testprogrammi käsklus static bool KasMahubAllveelaeva(Inimene isik), mis väljastab tõese väärtuse juhul, kui pikkust on alla 165 sentimeetri. Katseta käsku nii Inimese kui Modelli puhul, samuti Modellidest ja Inimestest koosneva massiivi juures.

* Väljasta false-vastus ka null-sisendi korral. Allveelaeva luba ka kuni 170 sentimeetri pikkused modellid (painduvad hästi, kontrolliks isik is Modell).

* Loo mõlemale klassile taas ka ilma pikkuseta konstruktor. Sellisel juhul pannakse pikkuse väärtuseks -1 ning esitluse juures teatatakse, et pikkuse andmed puuduvad.

Ülekate

Mõnikord ei piirduta omaduste lisamisega - tahetakse olemasolevaid võimalusi ka muuta. Tavanäiteks on kujundid või graafikakomponendid, mis igaüks ennast vastavalt oma omadustele joonistavad. Aga sarnaselt üle kaetavad võivad olla voogudesse andmete kirjutamise käsklused või andmestruktuuridesse andmeid lisavad või sealt eemaldavad käsklused nii, et neid käsklusi kasutav programm võib lihtsalt kasutatavat tükki usaldada, et tal on vastava nimega käsklus olemas ning ta selle soovitud ajal sooritab.

Siin näites on `Inimese` alamklassiks tehtud `Daam`, kel kõik muud kirjeldatud omadused hariliku Inimesega sarnased. Kuid vanuse ütlemine käib mõnevõrra teisiti. Et käske saaks rahus üle katta, selleks on `Inimese` juurde käsu ehk meetodi `YtleVanus` ette lisatud sõna `virtual`, `Daami` vastava meetodi ette aga `override`. Selliselt on mõlemat tüüpi objektil vanuse ütlemine selge, need lihtsalt käituvad erinevalt.

```
using System;
namespace Parilus3{
 class Inimene{
   protected int vanus;
   public Inimene(int uvanus){
      vanus=uvanus;
   }
   public virtual void YtleVanus(){
      Console.WriteLine("Minu vanus on "+vanus+" aastat");
   }
 }
 class Daam:Inimene {
   public Daam(int vanus):base(vanus){}
   public override void YtleVanus(){
      Console.WriteLine("Minu vanus on "+(vanus-5)+" aastat");
   }
 }
 class InimTest{
  public static void Main(string[] arg){
```

```
      Inimene inim1=new Inimene(40);
      Daam inim2=new Daam(40);
      inim1.YtleVanus();
      inim2.YtleVanus();
   }
  }
}

/*
C:\Projects\oma\naited>Parilus3
Minu vanus on 40 aastat
Minu vanus on 35 aastat
*/
```

Ülesandeid

* Lisa Inimesele käsklus "KutsuEttekandja", katseta seda eksemplari juures.

* Katseta sama käsklust ka Daami eksemplari juures.

* Kata nimetatud käsklus Daami juures üle, pannes sisse pidulikum ja peenem tekst. Katseta.

* Loo Inimeste massiiv, kus on nii Inimesed kui Daamid. Palu igaühel teatada oma vanus ning kutsuda ettekandja.

Liidesed

Nagu eespool kirjeldatud, on C# puhul üks ja kindel objektitüüpide pärinemise puu. Igal klassil on oma ülemklass ning juureks on klass `Object` nimeruumist `System`. Samas aga mõnegi tüübi puhul on sellest klassist objekte mugav kasutada tunduvalt rohkemates kohtades, kui otsene päriluspuu ette näeb. Nii nagu Ambla kihelkonna Lehtse valla Läpi küla Troska talu perepoeg Karl oli lisaks nendele ametlikele tiitlitele veel küla sepp, puutöömees ning korralik vanapoiss, nii on hea mõnegi klassi puhul programmi ülesehituse lihtsuse huvides pakkuda sinna rohkem rolle kui otsese päriluse järgi neid kokku tuleks. Näiteks Karlal oli leivateenimise mõttes sepatööst kindlasti rohkem kasu kui teatest, et ta Ambla kirikuraamatus kirjas on. Niisamuti on klassidele võimalik juurde panna liideseid, mis annavad õiguse vastava klassi eksemplare kloonida, järjestada või muid kasulikke omadusi lisada. Liideste arv ühe klassi juures ei ole piiratud. Liidesed mõeldi programmeerimiskeelte juurde välja, kuna selgus, et nõnda kirjeldusi määrates ja kokku leppides õnnestub programmeerimisel vigu vähendada.

Järgnevas näites loodi liides `IViisakas`. I on liidesel ees sõnast Interface. Liidese juurde käib enamasti sisuline seletus selle kohta, millised seda liidest realiseerivad objektid on. Ning lisaks võivad olla mõned käsklused, millele vastavat liidest realiseerivad objektid on võimelised reageerima. Liidese `IViisakas` puhul valiti selliseks käskluseks `Tervita`, mis saab omale tekstilise parameetri. Ehk siis eeldatakse, et iga viisakas tegelane mõistab tervitusele vastata juhul, kui talle öeldakse, kellega on tegemist. Nagu näha - `Lapse` ja `Koera` puhul need tervitused on erinevad. Kuid nii nagu liides nõuab - nad on olemas. Sinnamaani

suudab kompilaator kontrollida. Ülejäänu on juba programmeerija hoolitseda, et kindlaksmääratud nimega käskluse juurde ka vastavale klassile sobiv sisu saaks.

```
  static void TuleSynnipaevale(IViisakas v){
     v.Tervita("vanaema");
  }
```

Loodud meetodi juures on näha, et parameetrina võetakse vastu vaid nende klasside eksemplare, kelle puhul liides `IViisakas` on realiseeritud. Ning igaühel neist palutakse tervitada vanaema. Kuidas keegi sellega hakkama saab, on juba klassi enese sees hoolitsetud.

```
using System;
namespace Liides1{
 class Inimene{
   protected int vanus;
   public Inimene(int uvanus){
      vanus=uvanus;
   }
   public virtual void YtleVanus(){
      Console.WriteLine("Minu vanus on "+vanus+" aastat");
   }
 }
 interface IViisakas{
   void Tervita(String tuttav);
 }
 class Laps:Inimene, IViisakas {
   public Laps(int vanus):base(vanus){}
   public void Tervita(String tuttav){
     Console.WriteLine("Tere, "+tuttav);
   }
 }
 class Koer: IViisakas{
   public void Tervita(String tuttav){
      Console.WriteLine("Auh!");
   }
 }
 class InimTest{
  static void TuleSynnipaevale(IViisakas v){
     v.Tervita("vanaema");
  }
  public static void Main(string[] arg){
     Laps juku=new Laps(6);
     juku.YtleVanus();
     Koer muki=new Koer();
     TuleSynnipaevale(juku);
     TuleSynnipaevale(muki);
  }
 }
}
/*
C:\Projects\oma\naited>Liides1
Minu vanus on 6 aastat
Tere, vanaema
Auh!
*/
```

Kui mingil põhjusel jäänuks Lapsele või Koerale käsklus Tervita lisamata, siis annaks kompilaator veateate. Sama tekiks ka juhul, kui käskluse tekstis trükiviga tehtaks. Selline kompilaatoripoolne kontroll aitab vead üles leida juba enne tegelike andmetega katsetamise alustamist.

Ülesandeid

* Katseta, mis juhtub, kui Lapse tervita kirjutada väikese tähega.

* Lisa liidesesse IViisakas käsklus KoputaUksele.

* Muuda liidest realiseerivaid klasse nii, et kood kompileeruks.

* Testi töö tulemust.

* Koosta liides ISummeerija käsklustega Alusta, Lisa ning KysiSumma. Esimene siis tühjendab andmed, teine lisab ühe väärtuse ning kolmas väljastab olemasolevate summa.

* Koosta liidest realiseeriv klass, kus on muutuja summa hoidmiseks. Alustamise peale pannakse see nulli, lisamise puhul suurendatakse väärtust ning summa küsimisel väljastatakse meeles olev summa. Omista klassi eksemplar liidese tüüpi muutujale. Katseta.

* Koosta sama liidest realiseerima teine klass, kus lisatavate andmete jaoks on olemas massiiv. Kogu aeg peetakse meeles lisatud väärtuste arv ning väärtused ise. Summa küsimisel arvutatakse kokku elementide väärtuste summa ning väljastatakse see.

* Koosta eraldi peaprogramm, mis eeldab, et seal sees on kasutada ISummeerija liidest realiseeriv objekt – tegelikult aga algul on seal tühiväärtus-null. Peaprogramm koostatakse nõnda, et ta küsib kasutajalt arve ja palub ISummeerijal need kokku liita. Kontrolli, et kood kompileeruks.

* Koosta libasummeerija, mille Alusta ja Lisa-käsklused ei tee midagi. KysiSumma aga väljastab alati väärtuse -1. Katseta peaprogrammi loodud libaklassi objektiga.

* Loo eraldi klass SummeerijaVabrik summeerijaobjektide tootmiseks. Lisa sinna sisse staatiline käsklus LooSummeerija(int EeldatavKogus). Kui kogus on alla kümne, väljastatakse andmeid salvestav summeerija, muul juhul kohe andmeid kokku liitev summeerija. Katseta süsteemi toimimist.

Abstraktne klass

Liideseid pidi pärinevad vaid meetodite nimed. Klasside kaudu pärinevad nimed koos sisuga. Aga mõnikord soovitakse vahepealset varianti: et mõnedel meetoditel on olemas sisu, teistel aga veel mitte. Sellisel juhul aitab välja abstraktne klass. Tüüpiline näide on toodud allpool. Sirgete ja püstiste seintega kujundi puhul saab ruumala arvutada juhul, kui on teada põhja pindala ja kõrgus - valemi järgi piisab vaid nende omavahelisest korrutamisest. Kui aga

kujundeid on palju, siis on niru seda valemit igale poole uuesti kirjutada. Rääkimata juhtudest, kus valem tunduvalt keerukam on, või neid iga kujundi kohta mitu.

Siin on kujundi näidetena toodud `Tikutops` ja `Vorstijupp`. Esimesel neist on standardmõõtude puhul suurus kohe teada, vastavad meetodid väljastavad kohe konkreetsed arvud. Teisel juhul antakse mõõtmed objekti loomisel ette, meetodid peavad küsimise peale need salvestatud väärtused välja andma. Ning kui vorst ilusti matemaatiliselt omale silindrina ette kujutada, siis annab pii korda raadiuse ruut ilusti põhja ehk ristlõike pindala välja.

Klassist `Kujund` enesest ei saa eksemplare luua. Kui klass sisaldab abstraktseid ehk defineerimata meetodeid, siis peab klass ka ise tervikuna olema abstraktne ning siis ei lubata temast eksemplare teha. Muidu ju tekiks probleem, kui soovitaks käivitada kujundi olematu koodiga käsklust `KysiKorgus`().

Küll aga tohib muutujale tüübist `Kujund` tegelikke objekte omistada - olgu nad siis `Tikutopsid`, `Vorstijupid` või pärit mõnest muust `Kujundi` alamklassist. Sarnaselt nagu võis `Daami` omistada `Inimese` tüüpi muutujale või `Lapse` muutujale tüübist `IViisakas`. Kui üle kaetud klassis on eelnevalt abstraktsetele meetoditele sisu antud, siis võib sellest klassist julgesti isendeid luua ning neid ka kõikidest ülemklassidest pärit muutujatele omistada.

```
using System;
namespace AbstraktseKlassiUuring{
 abstract class Kujund{
   public abstract double KysiPohjaPindala();
   public abstract double KysiKorgus();
   public double KysiRuumala(){
      return KysiPohjaPindala()*KysiKorgus();
   }
 }
 class Tikutops:Kujund{
   public override double KysiPohjaPindala(){return 8;}
   public override double KysiKorgus(){return 1.5;}
 }


 class Vorstijupp: Kujund{
   int pikkus, raadius;
   public Vorstijupp(int upikkus, int uraadius){
      pikkus=upikkus;
      raadius=uraadius;
   }
   public override double KysiPohjaPindala(){
     return Math.PI*raadius*raadius;
   }
   public override double KysiKorgus(){
     return pikkus;
   }
 }
 class Test{
   public static void Main(string[] arg){
      Tikutops t=new Tikutops();
      Vorstijupp v=new Vorstijupp(10, 3);
      Console.WriteLine("Ruumalad {0} ja {1}",
                t.KysiRuumala(), v.KysiRuumala());
   }
```

```
 }
}
/*
D:\kodu\0606\opikc#>AbstraktseKlassiUuring
Ruumalad 12 ja 282,743338823081
*/
```

Ülesandeid

* Lisa klassile Kujund abstraktne meetod PohjaYmbermoot ning meetod KyljePindala. Katseta - lisades vajalikud meetodid ka alamklassidesse.

* Loo Kujundi alamklass Risttahukas lisades talle vajalikud väljad kolme mõõtme hoidmiseks ja kattes üles Kujundi abstraktsed meetodid. Katseta mitmesuguste Risttahuka eksemplaridega.

* Koosta mitmesuguste Kujundite massiiv. Loo alamprogramm leidmaks massiivis olevate kujundite ruumalade summa. Loo eraldi alamprogramm leidmaks massiivis olevate kujundite pindalade summa.

Meetodite asendus

Harilikult kirjutatakse meetodite üle katmise juures ülemklassi meetodi ette virtual ning alamklassi juurde override. Sellisel juhul alamklassi (siinses näites `Daami`) objekti puhul kasutatakse alati seda meetodit, mis tema juurde käib - sõltumata, millisest tüübist on muutuja, mille kaudu eksemplari poole pöördutakse. C++ võimalusi säilitades aga on jäetud ka teine võimalus. Meetodi võib asendada nõnda, et tema kirjeldamise ette kirjutatakse sõna `new`. Sel juhul peidetakse vana meetod samuti ära. Vana meetodi saab aga kätte juhul, kui objekti poole pöörduda ülemklassi muutujast, kus vastav meetod vanal kujul kasutusel oli. Kui `virtual`/`override` puhul pidid parameetrid ja väljastustüüp samaks jääma, siis `new` loob täiesti uue ja eelmisest sõltumatu meetodi.

Järgnevas näites on ehitatud kunstlik pärilusahel. Ülemklassiks `Inimene`, kes ütleb oma vanuse nõnda nagu see on. `Inimesest` pärinenud `Daam` võtab ilma pikemalt mõtlemata 5 aastat maha. `Daami` alamklassiks olev `Beib` keeldub üldse vanuse teatamisest ning eriti kaugele arenenud `KavalBeib` palub kasutajal ise tema vanust pakkuda. Sõna `sealed` klassi juures näitab, et sellest klassist ei lubata enam edasi pärida. Selline määrang aitab kompilaatoril koodi optimeerida.

Alljärgnevalt katsetatakse, millist tüüpi muutuja kaudu millise tegeliku objekti poole pöördumisel milline tulemus saadakse. Et omistamine on võimalik ainult ülemklassi suunas, siis igaühe neist saab omistada `Inimese` tüüpi muutujale. Mida tase edasi, seda vähem on omistusvõimalusi.

Katsetamise käigus antakse `Beiblastele` vanuseks 17 aastat, teistele 40. Ning jälgitakse, milline meetod millise muutuja kaudu väljakutsel käima läheb. Kõige pikem ja keerukam lugu on Kavala Beibega. Et ta on pärimispuus kõige kaugemal, siis teda on võimalik omistada kõikide selles puus olevate tüüpidega muutujatele. Enese tüübi puhul küsib ta vanuseks 19,

nagu käskluses öeldud. Ka lihtsalt Beib-tüüpi muutuja kaudu küsib ta enesele 19, sest klassi `Beib` meetod `YtleVanus` on virtual ning tegelikult käima läheb objekti enese ehk klassis `KavalBeib` loodud meetod.

Edasi muutub lugu keerulisemaks. `Daami`-muutujast välja kutsutav Kavala Beibe vanus tuleb 12, sest ta lahutab aastad maha nagu Daamile kohane. Ning ka hariliku inimese kaudu tuleb 12, sest `virtual`-piiritleja kaudu võetakse käsklus võimalikult objekti enese lähedalt.

```
using System;
namespace Asendus{
 class Inimene{
   protected int vanus;
   public Inimene(int uvanus){
      vanus=uvanus;
   }
   public virtual void YtleVanus(){
      Console.WriteLine("Minu vanus on "+vanus+" aastat");
   }
 }
 class Daam:Inimene {
   public Daam(int vanus):base(vanus){}
   public override void YtleVanus(){
      Console.WriteLine("Minu vanus on "+(vanus-5)+" aastat");
   }
 }
 class Beib:Daam{
   public Beib(int vanus):base(vanus){}
   new public virtual void YtleVanus(){
       Console.WriteLine("Minu vanus pole sinu asi, vot!");
   }
 }
 sealed class KavalBeib:Beib{ //siit ei saa enam edasi areneda
   public KavalBeib(int vanus):base(vanus){}
   public override void YtleVanus(){
      Console.WriteLine("Arva, kas olen {0}?", vanus+2);
   }
 }
 class InimTest{
  public static void Main(string[] arg){
     KavalBeib kb=new KavalBeib(17);
     Beib b=new Beib(17), bkb=kb;
     Daam d=new Daam(40), db=b, dkb=kb;
     Inimene i=new Inimene(40), id=d, ib=b,ikb=kb;
     kb.YtleVanus();
     b.YtleVanus();
     bkb.YtleVanus();
     d.YtleVanus();
     db.YtleVanus();
     dkb.YtleVanus();
     i.YtleVanus();
     id.YtleVanus();
     ib.YtleVanus();
     ikb.YtleVanus();
  }
 }
}

/*
```

```
D:\kodu\0606\opikc#>Asendus
Arva, kas olen 19?
Minu vanus pole sinu asi, vot!
Arva, kas olen 19?
Minu vanus on 35 aastat
Minu vanus on 12 aastat
Minu vanus on 12 aastat
Minu vanus on 40 aastat
Minu vanus on 35 aastat
Minu vanus on 12 aastat
Minu vanus on 12 aastat
*/
```

Ülesandeid

* Loo klass Punkt väljadega x ja y ning meetoditega KaugusNullist ja TeataAndmed. Esimene väljastab reaalarvuna kauguse koordinaatide alguspunktist. Teine tagastab tekstina koordinaatide väärtused.

* Loo Punktile alamklass RuumiPunkt. Lisa väli z, kata üle KaugusNullist ning asenda TeataAndmed. Esimene väljastab kauguse nullist kolme koordinaadi korral, teine aga kirjutab RuumiPunkti andmed ekraanile, meetodid tagastustüübiks on void.

* Katseta loodud objekte ja nende käsklusi igal võimalikul moel. Punkt Punkti muutujast, RuumiPunkt RuumiPunkti muutujast ning RuumiPunkt Punkti muutujast.

Omadused

Objektide juures tehakse enamasti selget vahet: väljad ehk muutujad on andmete hoidmiseks, käsud ehk funktsioonid ehk meetodid toimingute sooritamiseks, muuhulgas ka andmete poole pöördumiseks. Ning korralikult kapseldatud objektorienteeritud programmis pääseb väljadele otse ligi vaid objekti seest, kõik muud välised toimetused käivad meetodite kaudu.

Kahjuks või õnneks on enamik programmeerijaid kirjutanud ka objektikaugemat koodi. Kes sellepärast, et tema kooliajal polnud veel objektorienteeritus kuigi laialt levinud või mõni teine põhjusel, et piisavalt väikeste programmide puhul võib olla objektindusest rohkem tüli kui tulu. Nõnda pakuvad programmeerimiskeeled mitmesuguseid "vahevorme", kus püütakse kasu lõigata objektide süstemaatilisusest ning samal ajal jätta alles protseduuridel põhineva programmeerimise lihtsuse.

Eelpool oli selliseks heaks mooduseks kirje (`struct`). Kirje puhul eeldatakse, et ta on loodud põhiliselt andmete hoidmiseks, andmete õigsuse ja kokkusobivuse eest hoolitseb väline programm, et tegemist on lihtsalt muutujate komplektiga nagu näiteks punkti koordinaadid. Kirje vaid hoolitseb, et `x` ja `y` alati kokku kuuluksid.

Siiski on erinevalt mõnest muust keelest C# puhul lubatud kirje juurde ka käsklusi lisada. Enamasti kasutatakse neid arvutatavate omaduste - nagu näiteks sünniaja järgi vanuse - leidmiseks. Kirje kasutamise teeb objektist mugavamaks käskude mõningane lühidus. Seal võib rahulikult kirjutada

```
p1.X=17;
int a=p1.X;
```

Objektide puhul peetakse sellist otse muutujate poole pöördumist ebaviisakaks. Paremaks lahenduseks soovitatakse

```
p1.paneX(17);
int a=p1.KysiX();
```

On küll vaid paar tähte juures, aga piisavalt selle jaoks, et laisad programmeerijad vahel pigem muudavad objekti muutujad otse ja avalikult kättesaadavaks, kui et andmeid viisakalt meetodite kaudu vahetavad. Et lühidat kirjastiili säilitada ning samas jätta alles meetoditele iseloomulik kontrollitavus ja muudetavus, selleks ongi loodud võimalus objektile luua omadusi, mis näevad välja nagu muutujad, kuid käituvad nagu meetodid.

Järgnevas näites on klassi `Ilmaandmed` eksemplaridele lisatud omadus `Temperatuur`, millele saab väljapoolt väärtust omistada ning küsida nagu tavaliselt muutujalt ehk väljalt. Küsimise peale pannakse lihtsalt tööle `get`-koodiosa ning omistamise peale `set`-koodiosa. Sõna "`value`" tähistabki omistamisel antud väärtust. Koodi ülesandeks on vastava märksõna all saabunud andmed vajalikku kohta paigutada. Tegelikke andmeid hoitakse privaatses ehk väljapoolt nähtamatus muutujas nimega `temper`. Vajadusel saab nõnda omaduse või meetodi kaudu andmete poole pöördudes peale panna näiteks kontrolli võimalike väärtuste üle omistamisel. Või siis saab programmeerija otsustada hoopis, et temperatuure endid ei hoita mingil hetkel enam muutujas, vaid hoopis andmebaasis. Et kogu andmetega toimetamine on jäetud `get`- ja `set`-meetodite hooleks, siis on selline asendus täiesti võimalik.

```
using System;
namespace Omadused1{
 class Ilmaandmed{
   private int temper;
   public int Temperatuur{
      get{return temper;}
      set{temper=value;} //value on sisendväärtuse nimi
   }
 }
 class Test{
   public static void Main(string[] arg){
      Ilmaandmed jaam1=new Ilmaandmed();
      jaam1.Temperatuur=15;
      Console.WriteLine(jaam1.Temperatuur);
   }
 }
}

/*
D:\kodu\0606\dotnet>Omadused1
15
*/
```

Pöördumisstatistika

Siin ongi toodud võimalikult lihtne näide, kuidas peale omistamise/küsimise veel meelde jätta ja teada saada nende operatsioonide arv. Lihtsalt loendamise jaoks vastavad muutujad ja käsklused juures.

```
using System;
namespace Omadused2{
 class Ilmaandmed{
   private int temper;
   private int muudetud=0;
   private int loetud=0;
   public int Temperatuur{
      get{
        loetud++;
        return temper;
      }
      set{
        muudetud++;
        temper=value;
      }
   }
   public override string ToString(){
      return "Muudetud: "+muudetud+", loetud:"+ loetud+
                                     ", temperatuur:"+temper;
   }
 }
 class Test{
   public static void Main(string[] arg){
      Ilmaandmed jaam1=new Ilmaandmed();
      jaam1.Temperatuur=15;
      Console.WriteLine(jaam1.Temperatuur);
      Console.WriteLine(jaam1.Temperatuur);
      Console.WriteLine(jaam1);
   }
 }
}
/*
C:\Projects\oma\naited>Omadused2
15
15
Muudetud: 1, loetud:2, temperatuur:15
*/
```

Ülesandeid

***** Kui temperatuuriks märgitakse üle 35, trüki hoiatusteade kahtlase väärtuse kohta

***** Loo klass Kasutaja. Kasutajanimi määratakse konstruktoris, ning seda on hiljem võimalik ainult omadusena küsida, mitte muuta (`set`-osa puudub). Parooli saab ainult määrata, aga küsida pole võimalik (`get`-osa puudub). Lisa käsklus parooli kontrolliks. Lisa omadusena kasutaja telefoninumber, mida on võimalik küsida ja muuta. Kontrolli, et töötaksid vaid määratud tehted (st. et kord pandud parooli ei saaks küsida).

Indekseering

Massiivide puhul oleme harjunud, et kandiliste sulgude ja järjekorranumbri järgi küsime või seame omale väärtuse. C# süntaks lubab aga ka ise kirjeldatud objektidel omapoolsed käsud sellise pöördumise peale tööle panna. Iseenesest oleks võimalik kõik sellised toimetused mõne hariliku funktsiooni ehk meetodiga ära ajada, aga kandiliste sulgudega kirja panduna tehe näeb lühem välja. Ning mõnigi kord, kui andmed paistavad sarnased nagu massiivides hoitakse, võimaldab selline kantsulgudega tehe koodi ka harjunumalt kirja panna.

Esimeses, võimalikult lihtsas näites väljastatakse indekseerimistehte tulemusena etteantud arvu ruut. Kirjaviis on mõnevõrra sarnane omaduse kirjeldamisele: `get`-osas tuleb soovitud väärtus `return`i abil tagasi anda.

```
using System;
namespace Indekseering1{
 class Ruuduarvutus{
   public int this[int nr]{
      get{return nr*nr;}
   }
 }
 class Test{
  public static void Main(string[] arg){
     Ruuduarvutus r=new Ruuduarvutus();
     Console.WriteLine(r[3]);
  }
 }
}
/*
C:\Projects\oma\naited>Indekseering1
9
*/
```

Vahendus

Mõnelgi korral võib omaloodud indekseerimisel kasutada juba olemasolevat massiivi või muud andmekogumit. Ning indekseerimise kaudu saab lisada selle elementide kasutamisele mõne piirangu või ümberarvutusfunktsiooni. Siinses näites ehitati indekseerimise abil kest ümber paisktabelile - paaride kogumile, kus igas paaris on võti ja väärtus. Võtmeks on kuupäev, väärtuseks asukoht, kus vastaval päeval ansambel esineb. Andmete küsimisel vastatakse veel vaba päeva kohta küsimisel "Vaba", kinnipandud päeva puhul teatatakse, kus vastaval päeval esinemine on. Nimeruumis `System.Collections` asuval `Hashtable` klassist objektil on andmete salvestamise ja küsimise käsklused juba sisse ehitatud. Lihtsalt `Hashtable` annab vastava võtme puudumisel vastuseks tühiväärtuse `null`, meie aga vastame selle peale inimkeelse "Vaba" ning andmete salvestamise juures juhul kui vastav kuupäev kinni on heidetakse erind veateatega, miks vastav päev ei sobi. Kui aga soovitud kuupäev on veel vaba, siis pannakse sinna juurde ilusti sobiv väärtus kirja.

```
using System;
using System.Collections;
namespace Indekseering2{
 class Ringreis{
```

```
   Hashtable esinemised=new Hashtable();
   public string this[int kuupaev]{
      get{
         if(esinemised[kuupaev]==null){return "Vaba";}
         return (string)esinemised[kuupaev];
      }
      set{
         if(esinemised[kuupaev]!=null){
            throw new Exception("Juba kinni, esinemine linnas "+
                                                esinemised[kuupaev]);
         }
         esinemised[kuupaev]=value;
      }
   }
 }
 class Test{
  public static void Main(string[] arg){
     Ringreis r=new Ringreis();
     r[3]="Narva";
     r[4]="Tartu";
     Console.WriteLine(r[3]);
     Console.WriteLine(r[5]);
     r[3]="Viljandi";
  }
 }
}
/*
C:\Projects\oma\naited>Indekseering2
Narva
Vaba
Unhandled Exception: System.Exception: Juba kinni, esinemine linnas Narva
   at Ringreis.set_Item(Int32 kuupaev, String value)
   at Test.Main(String[] arg)
*/
```

Ülesandeid

* Muuda arvu ruutu väljastavat indekseerimise näidet nii, et see väljastaks etteantud arvu kuubi.

* Loo objekt, mis võtaks konstruktoris vastu sõna. Väljasta nii mitmes täht, kui indeksiga näidatakse. Kui arv ületab sõna pikkuse, siis antakse teada, et sellise järjekorranumbriga tähte ei leidu. Kui indeksiks pandud arv on negatiivne, siis väljastatakse niimitmes täht sõna lõpust arvates.

* Loo indekseeringu kaudu kasutatav seitsmeelemendiline massiiv vastaval nädalapäeval tehtud töötundide arvu summeerimiseks. Igal omistamisel liidetakse töötunnid vastava päeva arvule otsa. Igal küsimisel antakse välja sellele nädalapäevale liidetud töötundide summa. Negatiivse arvu sisestamisel tuleb veateade.

Struktuurne andmestik

Mitmedki toimetused saab tehtud ühe klassiga. Vajalikud andmed algul konstruktoris sisse ning käskudega kannatab neid küsida, lisada ja muuta nagu parajasti sobiv ja võimalik on. Kui

on tegemist korraga mitme sarnase ülesehituse, kuid erinevate väärtustega andmetega (näiteks mitme isikukoodiga), siis võib loodud klassist luua iga väärtuse jaoks omaette objekti ja sealtkaudu siis mõningast analüüsi vajavate andmetega ümber käia. Kui aga andmeid on palju, mitut tüüpi ja omavahel seotud, siis tasub mõelda nende omavahelise struktuuri peale.

Programmeerimist saab üldjuhul õppida „pastakast välja imetud" lihtsate näidete peal. Kui aga mõnda teemasse sisse minna, siis tuleb erialateadmised ja programmeerimisoskused ühendada. Mõlemad võivad eraldi võttes olla küllaltki lihtsad, kuid nende kokkupanek võib veidi peamurdmist nõuda. Samas aga teadmiste ja tehnika ühendamine võib anda eraldi toimetamisest märksa kasulikumaid tulemusi.

Järgnevas näites mängitakse läbi elektriskeemides loodetavasti põhikooliajast tuttavate takistite ja nende ühendamisega ette tulevad arvutused. Elektroonikule on seostub takistiga tõenäoliselt sageli punast värvi väike silinder, mille mõlemast otsast traat välja tuleb. Iseenesest aga käituvad takistina ka tavalised lambipirnid ja mõned muudki elektroonikaseadmed. Lihtsal takistil on keskkonnaoludest suhteliselt sõltumatu väärtusega takistus. Samuti on tal lubatud suurim eralduv võimsus, mille ületamisel võib takistist välja tulla „hall mull" ehk tossupilv ning komponent pole pärast seda enam kasutatav. Takistuseks nimetatakse juhtmeotstel ehk klemmidel oleva pinge (surve, voltides) ning takistit läbiva voolu (elektronide voog, amprites) suhet. Mida suurem takistus, seda vähem läheb sama pinge juures takistist voolu läbi. Takistil soojusena eralduv võimsus (wattides) leitakse takisti klemmidele pandud pinge ning takistit läbiva voolu korrutisena. Põhivalemid siis:

U/I=R

U*I=N

Kui veidi avaldada, siis leiab sealt, et $I = \sqrt{\frac{N}{R}}$. Ehk siis teadaoleva lubatud maksimumvõimsuse ja takistuse põhjal on võimalik leida takistit läbi suurim lubatud vool.

Takisteid saab omavahel kombineerida. Tüüpilised ühendused on järjestikku (jadamisi) ja rööbiti (paralleelselt). Jadamisi ühendades on arvutuskäik lihtne – takistuste komplekti kogutakistus on võrdne ühendatud takistite takistuste summaga. R=R1+R2 või ka rohkem komponente üksteisele järele liidetult. Rööbiti on arvutuskäik veidi keerulisem, kuid ka mitte lootusetu. Kahe rööbiti takisti kogutakistus on väiksem kui kummalgi eraldi, kogutakistuse pöördväärtus on võrdne üksikute takistite takistuste pöördväärtuste summaga $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$-esialgses programmis aga rööbiti ühendusega lihtsuse mõttes ei tegele.

Alustame kõige lihtsamast – loome klassi takisti andmete hoidmiseks ja sealt vajalike väärtuste küsimiseks või arvutamiseks vastavalt lisaandmetele. Takistil omal muutujateks takistus R ja maksimumvõimsus MaxN. Need antakse sisse ka konstruktoris. Vastavate andmete kättesaamiseks eraldi käsklused LeiaTakistus ja LeiaMaksimumV6imsus – kuna tegemist piiratud ligipääsuga (protected) muutujatega, siis muidu ei pruugi loodud objektist enam väärtusi näha. Otse muutujate poole pöördumine pole viisakas – ei võimalda programmeerijal hiljem enam loodud objektiga toimuvat kontrollida. Siin aga kui andmed

antakse sisse konstruktoris ning hiljem on võimalik neid ainult küsida, siis pole muret, et takisti andmed seletamatul põhjusel muutuma hakkaksid. Juurde ka mõned käsklused voolu ja võimsuse arvutamiseks ning kontrollimiseks, et kas soovitud pinge või vool ka konkreetsele takistile lubatud on.

```
using System;
using System.Text;
namespace Takistid {
    class Takisti {
        /// <summary>
        /// Takistus
        /// </summary>
        protected double R;
        /// <summary>
        /// Maksimumvõimsus
        /// </summary>
        protected double MaxN;
        public Takisti(double Takistus, double Maksimumv6imsus) {
            this.R = Takistus;
            this.MaxN = Maksimumv6imsus;
        }
        public double LeiaVool(double Pinge) {
            return Pinge / R;
        }
        public double LeiaV6imsus(double Pinge, bool
SobivusKontroll) {
            double V6imsus=Pinge * LeiaVool(Pinge);
            if (SobivusKontroll) {
                if (V6imsus > MaxN) {
                    throw new Exception("Pingel " + Pinge + " ületab
võimsus " +
                       V6imsus + " lubatud maksimumvõimsust " +
MaxN);
                }
            }
            return V6imsus;
        }
        public double LeiaV6imsus(double Pinge) {
            return LeiaV6imsus(Pinge, true);
        }
        public bool KasLubatudVõimsusVastavaltPingele(double Pinge)
{
            return LeiaV6imsus(Pinge, false) <= MaxN;
        }
        public double LeiaMaksimumVool() {
            return Math.Sqrt(MaxN / R);
        }
        public double LeiaTakistus() {
            return R;
        }
        public double LeiaMaksimumV6imsus() {
            return MaxN;
        }
    }
```

```
}
```

Kui takisti klass valmis, siis on hea seda katsetada. Loome konkreetsete omadustega takisti – näiteks 5Ω takisti maksimumvõimsusega 2W – ehk siis ettekujutatuna ühe pisikese lapse näpuotsasuuruse jupikese, millest kaks juhet välja tulevad. Kontrollime, kas sellise takisti kannataks ühendada 1,5-voldise patarei taha. Programm arvutab ja teatab, et kannatab küll, väljundvõimsuseks 0,45 Watti. Ise järgi arvutades võime tulemust kontrollida. 1,5 volti jagatud 5 oomiga annab 0,3 amprit. 0,3 amprit korrutatuna 1,5 voldiga teebki 0,45 watti. Mis siis teeb küll takisti õrnalt soojaks, aga ei lõhu seda veel ära. Kui küsitaks peale tunduvalt suurem pinge, siis meie programm peaks teatama, et sellise pingega tekkiv võimsus pole lubatud.

```
using System;
using System.Text;
namespace Takistid {
    class TakistiProov {
        public static void Main(string[] arg) {
            Takisti t1 = new Takisti(5, 2); //5 oomi, 2 watti
            double PatareiPinge = 1.5; //volti
            if (t1.KasLubatudVõimsusVastavaltPingele(PatareiPinge))
{
                Console.WriteLine("Patareilt " + PatareiPinge + "V
saadakse " +
                    " takistiga " + t1.LeiaTakistus() + " oomi vool
" +
                      t1.LeiaVool(PatareiPinge) + "A ja " +
                    " võimsus " + t1.LeiaV6imsus(PatareiPinge) +
"W");
            } else {
                Console.WriteLine("Pingel "+PatareiPinge+"V tekkiv
võimsus "+
                   t1.LeiaV6imsus(PatareiPinge, false)+"W ületab
lubatud "+
                   "võimsust "+t1.LeiaMaksimumV6imsus()+"W");
            }
        }
    }
}

/*
Patareilt 1,5V  saadakse  takistiga 5 oomi vool 0,3A ja  võimsus
0,45W
*/
```

Ühe või kahe takistiga saab rahumeeli toimetada. Nendega ümber käimiseks pole isegi programmi vaja. Kui aga andmeid rohkem, siis tasub mõelda nende haldamise peale. Elektroonikutel on suuremate takistikogustega ümber käimiseks kasutada takistussalved. Sarnase vahendi kannatab ka programmi poole pealt valmis teha. Nii nagu programmiobjektid ühel või teisel moel jäljendavad reaalse maailma objektid omadusi, nii ka siinpool. Koostame takistite jadamisi ühendamiseks eraldi takistussalve. Nii nagu pärissalves on kindel arv kohti takistite jaoks, nii siin on massiivil ka kindel hulk mälupesi takistiobjektide jaoks. Pesade arv

määratakse kindlaks salve konstruktoris. Salvest voolu läbilaskmisel on piirajaks kõige nõrgemat voolu kannatava takisti maksimumvool. Kui aga salv on tühi, ka siis ei või sellest lõpmatult suurt voolu läbi lasta. Siin näites on määratud tühja salve maksimumvooluks 16 amprit – selline korraliku koduse pikendusjuhtme läbilaskevõime. Salvel on käsklus takisti lisamiseks – esialgu aga veel mitte eemaldamiseks või väljavahetamiseks – seda lihtsalt lühiduse ja lihtsuse mõttes. Lisamisel jäetakse meelde lisatud takistite arv – siis teab, millisesse pessa järgmine lisatav takisti panna. Samuti on võimalik hoiatusteade anda juhul, kui salve enam takisteid ei mahu. Kogutakistuse leidmiseks liidetakse salves olevate takistite takistuste summad kokku. Maksimumvoolu leidmiseks leitakse vool, mida kannatab ka kõige nõrgem takisti ahelas. Lubatud pinge kontrollimiseks leitakse salve kogutakistuse järgi arvutatud vool vastavalt pingele ning kontrollitakse, kas see on piisavalt väike, et kõik salves olevad takistid selle välja kannataksid.

```
using System;
using System.Text;
namespace Takistid {
    class JadamisiTakistusSalv {
        Takisti[] Andmed;
        int TakistiteArv = 0;
        const int TyhjaSalveMaksimumvool = 16;
        public JadamisiTakistusSalv(int TakistiteMaksimumArv) {
            Andmed = new Takisti[TakistiteMaksimumArv];
        }
        public void LisaTakisti(Takisti t) {
            if (TakistiteArv < Andmed.Length) {
                Andmed[TakistiteArv] = t;
                TakistiteArv++;
            } else {
                throw new Exception("Takistussalv täis!");
            }
        }
        public double LeiaKoguTakistus() {
            double summa = 0;
            for (int i = 0; i < TakistiteArv; i++) {
                summa = summa + Andmed[i].LeiaTakistus();
            }
            return summa;
        }
        public double LeiaMaksimumVool() {
            double MaxVool = TyhjaSalveMaksimumvool;
            for (int i = 0; i < TakistiteArv; i++) {
                if (Andmed[i].LeiaMaksimumVool() < MaxVool) {
                    MaxVool = Andmed[i].LeiaMaksimumVool();
                }
            }
            return MaxVool;
        }
        public bool KasLubatudV6imsusVastavaltPingele(double Pinge)
{
            double TekkivVool = Pinge / LeiaKoguTakistus();
            return TekkivVool <= LeiaMaksimumVool();
        }
    }
```

```
}
```

Iga loodud klassi on ka hea katsetada. Siis julgem tunne, et tulevikus klassi tööd usaldada võib. Vastutusrikkamatel kohtadel tuleb läbi proovida igasugu kombinatsioonid ja võimalikud erijuhud. Ning seal võib testprogrammide kirjutamine olla paar-kolm korda keerukam kui rakenduse enese loomine. Siin aga lihtsalt proovime, kas salve loomine õnnestub. Paar takistit sisse ja väike arvutus kogutakistuse, maksimumvoolu ja lubatud pinge kohta. Nagu näha, salv loodi viiele takistile, sinna sisse paneme praegu ainult kaks. Üks kümneoomine ühevatine ning teine kümneoomine kahevatine. Siis aimatavalt on kogutakistus kakskümmend oomi ehk kahe takisti takistuste summa.

```
using System;
namespace Takistid {
    class TakistusSalveProov {
        public static void Main(string[] arg) {
            JadamisiTakistusSalv Salv = new JadamisiTakistusSalv(5);
            Salv.LisaTakisti(new Takisti(10, 1));
            Salv.LisaTakisti(new Takisti(10, 2));
            Console.WriteLine("Kogutakistus:
"+Salv.LeiaKoguTakistus());
            Console.WriteLine("Maksimumvool: " +
Salv.LeiaMaksimumVool());
            Console.WriteLine("Kas 12 Volti salve klemmidel lubatud:
" +

Salv.KasLubatudV6imsusVastavaltPingele(12));
        }
    }
}
/*
Kogutakistus: 20
Maksimumvool: 0,316227766016838
Kas 12 Volti salve klemmidel lubatud: False
*/
```

Ülesandeid

* Tutvu näidetega, vaheta väärtusi, kontrolli vastuste usaldusväärsust.

* Lisa takistiklassile käsklus, kus leitakse etteantud voolu tekitamiseks vajalik pinge. Katseta

* Lisa sama käsklus takistussalvele

* Kontrolli takisti lisamisel salve, et sama takisti ei oleks seal juba olemas.

Ühine ülemklass

Kui eelmist näidet tähelepanelikult jälgida, siis paistab, et nii takistussalve kui takisti juures on sarnaseid käsklusi. Voolu järgi pinge või pinge järgi voolu leidmise valem on ikka sarnane

– tuleb ainult omale selgeks teha, mida see takistus parajasti tähendab. Samuti ei tohi ei üksikule takistile ega salvele panna peale suuremat pinget, kui suurima lubatud voolu jaoks kõlbulik pinge on. Ning mis veel peenem – kui kord on kokku pandud takistussalv sobiva suurusega takistusega, siis võib selle salve juhtmeotsad hea tahtmise korral ühendada teise salve ühe takisti kohale. Nõnda kombineerides saab salvedest ja takistitest kokku ühendada päris paindliku süsteemi. Ja et see päriselus võimalik on, siis on programmeerimiskeelte loojad teinud kõik, et ka programmiklassidega maailma järele tehes sellise paindlikkuse kokku saaks. Nagu alljärgnevalt näha, see ka õnnestub.

Takistile ja takistussalvele (ja võibolla veel mõnele muule elektronide voolamist pidurdavale seadmele, näiteks lambipirnile) loodi ühine ülemklass TakistusKomponent. Käsklused LeiaTakistus ja LeiaMaksimumVool on määratud abstraktseteks. Sest igal seadmel on selle leidmise moodus isesugune, vastavad omadused on aga igal elektriseadmel ehk takistuskomponendil olemas. Ülejäänud abstraktse klassi käsklustes võime takistuse ja maksimumvoolu lugeda juba teatuks ning konstrueerida muid käsklusi neid käske kasutades. Kui takistus teada, siis voolu saab pinge jagades takistusega. Kui maksimumvool teada ja pingele vastav vool arvutatav, siis saab kergesti järele kontrollida, kas komponendi klemmidele tohib olemasolevat pinget rakendada või mitte.

```
using System;
namespace TakistusKomponendid {
    abstract class TakistusKomponent {
        public abstract double LeiaTakistus();
        public abstract double LeiaMaksimumVool();
        public double LeiaVoolVastavaltPingele(double Pinge) {
            return Pinge / LeiaTakistus();
        }
        public bool KasLubatudPinge(double Pinge) {
            return LeiaVoolVastavaltPingele(Pinge) <
LeiaMaksimumVool();
        }
    }
}
```

Takisti ise näeb pärast seda juba suhteliselt lihtne välja. Poest ostetud vidinale omased sisetakistus ja maksimumvõimsus tuleb ikka meelde jätta. TakistusKomponent kohustab üle katma käsklused LeiaTakistus ja LeiaMaksimumVool. Esimese saab kätte otse muutujast. Teise leidmise jaoks tuleb üleval tuletatud valemit rakendada.

```
using System;
namespace TakistusKomponendid {
    class Takisti:TakistusKomponent {
        protected double R, MaxN;
        public Takisti(double Takistus, double MaksimumV6imsus) {
            this.R = Takistus;
            this.MaxN = MaksimumV6imsus;
        }
        public override double LeiaTakistus() {
            return R;
        }
```

```
        public override double LeiaMaksimumVool() {
            return Math.Sqrt(MaxN / R);
        }
    }
}
```

Takistussalves tuleb ikka luua koht sinna sisse pandavate komponentide andmete hoidmiseks. Erinevalt eelnevast näitest aga nüüd on andmepesa tüübiks TakistusKomponent. See tähendab, et programmis lubatakse salve sisse panna ka teisi salvesid. Miski ei takista praegu salvel ka iseenese väljuvaid juhtmeid ühe oma takistikomponendi klemmidele ühendada – selline suhteline mõttetus jääks aga praegu lihtsalt programmeerija südametunnistusele. Kusjuures mõne programmi puhul pole iseenese andmete hoidmine sugugi mõttetu nähtus. Näiteks, kui grupijuht peab hoolitsema inimeste toiduportsude eest, siis peab ta kindlasti hoolitsema, et ta ka enese tarbeks supikausi muretseks. Kogutakistuse leidmisel leitakse üksikute pesade takistuste summad. Kui seal juhtuvad olema tavalised takistid, siis saadakse väärtused ühe käsuga muutujast kätte. Kui aga salve pesas juhtub olema teine salv, siis käsklus LeiaTakistus käivitab selle salve kogutakistuse leidmise käskluse. Samuti maksimumvoolu puhul leitakse eraldi iga komponendi maksimaalne lubatud vool. Ning kui mõneks komponendiks on salv, siis uuritakse läbi eraldi kõik selle salve pesad ja antakse tagasi sealse kõige õrnemat voolu kannatava komponendi vooluandmed.

```
using System;
namespace TakistusKomponendid {
    class JadamisiTakistiteSalv:TakistusKomponent {
        TakistusKomponent[] Andmed;
        int KomponentideArv = 0;
        const double TyhjaSalveMaksimumVool=16;
        public JadamisiTakistiteSalv(int MaxKogus) {
            Andmed = new TakistusKomponent[MaxKogus];
        }
        public void LisaTakistusKomponent(TakistusKomponent t) {
            Andmed[KomponentideArv] = t;
            KomponentideArv++;
        }
        public override double LeiaTakistus() {
            double abi = 0;
            for (int i = 0; i < KomponentideArv; i++) {
                abi = abi + Andmed[i].LeiaTakistus();
            }
            return abi;
        }
        public override double LeiaMaksimumVool() {
            double MaxVool = TyhjaSalveMaksimumVool;
            for (int i = 0; i < KomponentideArv; i++) {
                if (Andmed[i].LeiaMaksimumVool() < MaxVool) {
                    MaxVool = Andmed[i].LeiaMaksimumVool();
                }
            }
            return MaxVool;
        }
    }
}
```

Edasine on taas katsetus. Luuakse kaks salve. Ühele sisse kaks kümneoomist takistit, kahe- ja ühevatine. Teise salve üks 15-oomine kümnevatine takisti. Ning kolmanda salve sisse sootuks kaks esimest salve. Edasi võib juba rahus päringuid tegema hakata ja vaadata, milliseid andmeid kust tagastatakse.

```
using System;
namespace TakistusKomponendid {
    class TakistusKomponentideProov {
        public static void Main(string[] arg) {
            JadamisiTakistiteSalv ts1 =
                new JadamisiTakistiteSalv(5);
            ts1.LisaTakistusKomponent(new Takisti(10, 2));
            ts1.LisaTakistusKomponent(new Takisti(10, 1));
            JadamisiTakistiteSalv ts2 =
                new JadamisiTakistiteSalv(5);
            ts2.LisaTakistusKomponent(new Takisti(15, 10));
            JadamisiTakistiteSalv ts3 =
                new JadamisiTakistiteSalv(5);
            ts3.LisaTakistusKomponent(ts1);
            ts3.LisaTakistusKomponent(ts2);
            Console.WriteLine("Esimese salve maksimumvool: " +
                  ts1.LeiaMaksimumVool());
            Console.WriteLine("Teise salve maksimumvool: " +
                  ts2.LeiaMaksimumVool());
            Console.WriteLine("Kolmanda salve maksimumvool: " +
                  ts3.LeiaMaksimumVool());
        }
    }
}

/*
Esimese salve maksimumvool: 0,316227766016838
Teise salve maksimumvool: 0,816496580927726
Kolmanda salve maksimumvool: 0,316227766016838
*/
```

Ülesandeid

***** Koosta salv nelja viievatise kümneoomise takistiga. Leia kogutakistus, samuti suurim lubatud vool.

***** Lisa abstraktsele takistuskomponendile käsklus LeiaPingeVastavaltVoolule. Selle ning LeiaMaksimumVool-u abil koosta käsklus LeiaMaksimumPinge. Katseta seda käsklust nii üksiku takisti kui takistussalve juures. Võrdele leitud pinget eelmises ülesandes leitud voolule vastava pingega.

***** Koosta TakistusKomponendi alamklass rööpühenduses oleva kahe takistuskomponendi kogutakistuse leidmiseks. $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$. Maksimumvoolu leidmisel tuleb arvestada, et mõlemale takistile mõjub sama pinge, takisteid läbiv vool aga jaotub pöördvõrdeliselt

vastavalt takistusele – mida suurem takistus, seda väiksem vool. Võimalik on ka hakata pingeid katsetama väikese sammuga ja sealtkaudu leida sobiv maksimumvool. Katseta selle rööpühenduse klassi põhjal loodud objekti töö õigsust mitmesuguste takistite ja takistussalvede puhul – kaks ühesugust takistit rööpühenduses, kaks erinevat takistit rööpühenduses. Takisti ja jadamisi salv rööpühenduses. Kaks rööpühendust veel omakorda rööpühenduses.

Operaatorite üledefineerimine

Kui kõik saadud kiidusõnad kokku liita, siis võib end küll inglina tunda ...

Ehk siis liitmine ei pruugi sugugi tähendada arvude aritmeetilist summeerimist, vaid võib vastavalt taustale ja teemale hoopis isesuguse tähenduse saada. Samuti võib omaloodud klassist objektide puhul määrata, mida üks või teine tehtemärk tegelikult nende objektidega teeb. Mõnes keeles (nt. Java) on tehtemärkide programmeerijapoolne määramine veaohu tõttu ära keelatud. Aga siia keelde on võimalus alles jäetud ning aitab loodud objekte tavaelus toimuvaga sarnasemalt käituma panna.

Eelpoolkirjeldatud indekseerimistehte juures määras programmeerija samuti, kuidas omaloodud objekti puhul kantsulgude operaatori juures toimida. Järgnevalt paistab aga, et omapoolse tõlgenduse võib lisada mitmetele kasutatavatele tehtemärkidele. Nii nagu kantsulgude puhul, nii ka igasugu muude tehtemärkide asenduste puhul saab sama töö ära teha tegelikult vaid omaloodud funktsioone kasutades. Ja seetõttu pole koodi kirjutamise juures siinne teema sugugi hädavajalik. Aga võõra koodi mõistmiseks või oma objektidega lihtsaks ja elegantseks toimetamiseks sobivad omapoolselt käituma määratud tehtemärgid hästi.

Näiteks on võetud inimestele hästi tuttav kellaaeg. Tunde ja minuteid saab sarnaselt liita nagu kõiksugu muid suurusi. Kui aga näiteks 14:45le liita kaks tundi ja 30 minutit, siis tulemuseks pole mitte 16:75, vaid 17:15 ja selline ümberarvutus tuleb ka programmile selgeks teha, et välja arvutatud tulemustega ka inimeste keskkonnas midagi peale hakata oleks.

Plussmärgi üledefineerimiseks luuakse funktsioon nimega `operator+`, parameetriteks antakse kaks kellaaega - ehk siis vasakul ning paremal pool tehtemärki olev. Ja funktsiooni tagastustüübiks on samuti `Kellaaeg`. See tähendab, et kui kokku liidetakse kaks `Kellaaega`, siis tulemuseks on ka `Kellaaeg`.

Funktsiooni sisu jääb praegu lühikeseks. Käsu `new` abil luuakse uue `Kellaaja` eksemplar ning mõlema välja jaoks antakse lihtsalt ette plussmärgi mõlemal pool olnud vastava välja summad. Et funktsiooni esimese parameetrina olev `k1` on tehte väljakutsuja, siis siin on võetud tund ja minut ta väljade seest. Objekti `k2` puhul on andmete poole pöördumiseks viisakalt meetodit kasutatud. Aga eks oleks võimalik ka mõlemal juhul meetodiga seda küsimist toimetada.

```
public static Kellaaeg operator+(Kellaaeg k1, Kellaaeg k2){
   return new Kellaaeg(k1.tund+k2.Tund(), k1.minut+k2.Minut());
}
```

Et uue eksemplari loomisel andmed ilusti salvestatud saaksid, selle eest hoolitseb konstruktor. Samuti kutsub viimane välja käskluse `aegKorda`, mille ülesandeks on liigsed minutid tundideks muundada.

```
   public Kellaaeg(int utund, int uminut){
      tund=utund;
      minut=uminut;
      aegKorda();
   }
```

Senikaua, kui minuteid juhtub etteantud olukorras olema üle kuuekümne, minnakse järgmise tunni juurde ning võetakse minutite alt neid tunni jagu vähemaks.

```
   void aegKorda(){
      while(minut>60){
         tund++;
         minut-=60;
      }
   }
```

Kõige tavalisemal juhul, ehk siis, kui minuteid ongi alla kuuekümne, on tsükli tingimus kohe algul väär, tsüklit ei täideta ühtegi korda ning funktsioon lõpetab töö ilma midagi tegemata. Aga vähemasti võime kindlad olla, et ajaga on kõik korras.

```
using System;
class Kellaaeg{
   int tund, minut;
   public Kellaaeg(int utund, int uminut){
      tund=utund;
      minut=uminut;
      aegKorda();
   }
   void aegKorda(){
      while(minut>60){
         tund++;
         minut-=60;
      }
   }
   public int Tund(){return tund;}
   public int Minut(){return minut;}
   public void tryki(){
      Console.WriteLine("{0}:{1}", tund, minut);
   }
   public static Kellaaeg operator+(Kellaaeg k1, Kellaaeg k2){
      return new Kellaaeg(k1.tund+k2.Tund(), k1.minut+k2.Minut());
   }
}
class Test{
   public static void Main(string[] arg){
      Kellaaeg k1=new Kellaaeg(12, 10);
      Kellaaeg k2=new Kellaaeg( 1,  4);
      Kellaaeg k3=k1+k2;
      k3.tryki();
   }
```

```
}
/*
C:\Projects\oma\naited>Operaatorid1
13:14
*/
```

Tüübimuundusoperaatorid

Tahtes reaalarvu täisarvuks muundada nõnda, et komakohad kaduma lähevad, piisab arvu ette sulgudes sõna `(int)` kirjutamisest. Näiteks

```
int a=(int)6.34;
```

Kui uut tüüpi ette ei kirjutaks, siis annaks kompilaator veateate, sest reaalarvu ei pruugi olla võimalik kadudeta muundada täisarvuks. Sama lugu kehtib ka erineva suurusvaruga täisarvude või erineva täpsusega reaalarvude puhul: kui muundusel võib andmeid kaduma minna, siis tuleb muunduskäsk selgelt välja kirjutada. Vastupidi võib lasta ka arvutil tüübimuunduse automaatselt ära teha. Näiteks

```
double b=3;
```

Ehkki kirjutatud kolm on arvuti jaoks algselt tüübist int, lubatakse see rahus reaalarvule omistada.

Tüübimuundusi võib aga ka omaloodud tüüpide juures ette võtta.

Järgnevaga teatatakse, mis tuleb ette võtta juhul, kui kellaaeg omistatakse täisarvule. Sõna `implicit` ütleb, et omistada võib eraldi tüübiteisenduskäsku `(int)` näitamata.

```
   public static implicit operator int(Kellaaeg k){
      return k.Tund()*60+k.Minut();
   }
```

Ehk kui algul kirjutatakse

```
      Kellaaeg k1=new Kellaaeg(12, 10);
```

ja pärast

```
      int minutidPaevaAlgusest=k1;
```

siis kellaaja muutmine arvuks käib ilma, et programmeerija peaks sellele eraldi tähelepanu juhtima.

Kui operaatorite kirjeldajal aga tekib kahtlus, et teisenduste käigus võivad andmed ebatäpsemateks muutuda, või lihtsalt soovitakse, et kogemata ei tehtaks kirjeldatavat teisendust, siis tuleb lisada piiritlejaks sõna `explicit`.

```
   public static explicit operator double(Kellaaeg k){
      //kohustuslik muunduse näitamine
      return k.Tund()+k.Minut()/60.0;
   }
```

Sellisel juhul tuleb omistamisel sobivasse kohta kirjutada `(double)`, et teisendusest asja saaks. Muidu annab kompilaator lihtsalt veateate.

```
      double tunnidPaevaAlgusest=(double)k1;
```

Operaatorid võivad töötada ka teises suunas - ehk siis olemasolevast tüübist uue loodava tüübi poole. Siin näites tehakse minutite hulgast taas `Kellaaeg`. Niipalju, kui jagub täistunde, pannakse tundide alla. Mis aga tundideks jagamisest jäägina üle jääb, see muutub minutiteks.

```
   public static explicit operator Kellaaeg(int minutid){
       return new Kellaaeg(minutid/60, minutid%60);
   }
```

Ning omistamisel saabki minutid taas `Kellaajaks`.

```
      Kellaaeg k4=(Kellaaeg)minutidPaevaAlgusest;
```

Kui koodi töö tulemust vaadata, siis algul oli kellaaeg `k1` 12:10. Vahepeal muundati see muutujasse `minutidPaevaAlgusest` ja saadi täisarvuna 730. Lõpuks minutid taas tundide ja minutitena kellaaja sisse jaotatuna andsid jälle 12 tundi ja 10 minutit.

```
using System;
class Kellaaeg{
   int tund, minut;
   public Kellaaeg(int utund, int uminut){
      tund=utund;
      minut=uminut;
      aegKorda();
   }
   void aegKorda(){
      while(minut>60){
         tund++;
         minut-=60;
      }
   }
   public int Tund(){return tund;}
   public int Minut(){return minut;}
   public void tryki(){
      Console.WriteLine("{0}:{1}", tund, minut);
```

```
    }
    public static  Kellaaeg operator+(Kellaaeg k1, Kellaaeg k2){
       return new Kellaaeg(k1.Tund()+k2.Tund(), k1.Minut()+k2.Minut());
    }
    public static implicit operator int(Kellaaeg k){
       return k.Tund()*60+k.Minut();
    }
    public static explicit operator double(Kellaaeg k){
       //kohustuslik muunduse näitamine
       return k.Tund()+k.Minut()/60.0;
    }
    public static explicit operator Kellaaeg(int minutid){
        return new Kellaaeg(minutid/60, minutid%60);
    }
}
class Test{
    public static void Main(string[] arg){
       Kellaaeg k1=new Kellaaeg(12, 10);
       Kellaaeg k2=new Kellaaeg( 1,  4);
       Kellaaeg k3=k1+k2;
       k3.tryki();
       int minutidPaevaAlgusest=k1;
       double tunnidPaevaAlgusest=(double)k1;
       Console.WriteLine(minutidPaevaAlgusest);
       Console.WriteLine(tunnidPaevaAlgusest);
       Kellaaeg k4=(Kellaaeg)minutidPaevaAlgusest;
       k4.tryki();
    }
}

/*
C:\Projects\oma\naited>Operaatorid2
13:8
730
12,1666666666667
12:10
*/
```

Võrdlusoperaatorid

Ikka tahetakse võrrelda, kas midagi toimus enne või pärast; uus on vanast suurem või väiksem; keegi kellestki targem või rumalam. Mõningate objektide puhul ei pruugi selline võrdlus anda selgepiirilist jah/ei vastust ning sellisel juhul on mõistlik arvuti jaoks võrdlusoperaatori defineerimine ära jätta ning piirduda pigem mõnevõrra hägusema ja paindlikuma skaalaga. Kui aga enamikel juhtudel on objektid omavahel kindlalt võrreldavad, siis on vastav operaator omal kohal.

Võrdluse jaoks on tehteid palju: >, <, <=, <=, ==, !=. Enamasti pole aga otstarbekas kõiki neid kohe ja eraldi defineerima hakata. Pigem teha täisarvu väljastav levinud funktsioon `Compare`. Selles võrreldakse aktiivset objekti funktsiooni parameetrina antud objektiga. Kui programmeerija arvates on aktiivne objekt parameetrina antud objektist järjestusreas eespool, siis peaks funktsioon väljastama negatiivse arvu. Kui tagapool, siis positiivse. Ning kui objektide võrreldavad tunnused peaksid programmeerija arvates olema võrdsed, siis tuleb väljastada 0. Edasised võrdlustehted saab juba sedasama funktsiooni kasutades korda ajada.

Siin näites tehakse mõlema kellaaja sisu kõigepealt minutiteks alates päeva algusest, et oleks kergem võrrelda ning siis juba saab ühe lahutustehtega sobiva vastuse kätte.

```
public int Compare(Kellaaeg k){
   int omaminutid=(int)this;
   int teiseminutid=(int)k;
   return omaminutid-teiseminutid;
}
```

Samuti on viisakas üle katta klassist `Object` kaasa tulev käsklus `Equals`, mille ülesandeks on teatada, kas aktiivne objekt on etteantud objektiga sisu poolest võrdne. Kui oma `Compare` juba defineeritud, siis läheb edasine juba küllalt sarnaselt iga objekti puhul. Tingimuse esimese poolega tasub kontrollida, et kas võrreldav objekt üldse on meie omaga sama tüüpi. Ehk siis küsitakse, kas

```
ob is Kellaaeg
```

Vastus on tõene vaid juhul, kui etteantud objekt tõesti oli `Kellaaeg`. Ainult sel juhul minnakse `&&` abil kirjutatud võrdluse kontrollimisega edasi. Muul juhul on tingimus kohe vale, tingimusblokist hüpatakse üle ning väljastatakse funktsiooni lõpus tagastatav vastus teatamaks, et kindlasti ei ole meie `Kellaaeg` sama väärtusega kui võrdlemiseks etteantud objekt, sest see teine objekt lihtsalt ei ole `Kellaaeg`.

Et saadud objekti saaks `Kellaajana Compare`-meetodisse panna, peab eraldi teatama, et me teda ikka `Kellaajana` kasutame. Ehkki `is`-kontrolli abil tegime juba kindlaks, et tegemist on `Kellaajaga`, tuleb funktsioonile ette andmiseks see uuesti muundada. Üheks võimaluseks oleks

```
(Kellaaeg)ob
```

siin aga näitame teist sarnast võimalust

```
ob as Kellaaeg
```

mis käitub üldjoontes samamoodi. Ainsaks erinevuseks on, et kui peaks siiski tüübiprobleem tekkima, siis esimesel juhul heidetakse erind, teisel juhul väljastatakse aga lihtsalt tühiväärtus. Kuna siin on juba kontroll tehtud, siis veateadet nagunii ei saa tekkida ning kood töötaks mõlemal juhul sarnaselt. Käsule `Compare` antakse tulemus kontrollida. Kui väärtused loeti võrdseteks ning väljastatakse 0, siis sel juhul kannatab `Equals` välja anda `true`, muul juhul `false`.

```
public override bool Equals(Object ob){
   if (ob is Kellaaeg && (this.Compare(ob as Kellaaeg)==0)){
     return true;
   }
   return false;
```

```
    }
```

Oma võrdlusoperaatorite puhul on viisakas üle katta veel ka käsklus `GetHashCode`. Selle salapärase käsu ülesandeks on anda arvutile märku, kas objektid võiksid olla võrdse väärtusega. Selleks tuleb objekti andmete põhjal kokku panna üks täisarv. Kui kahe objekti andmed kattuvad, peab see arv olema ühesugune, muul juhul võimaluse korral erinev. Kasutatakse näiteks paisktabelites, andmete otsimise jms. korral. Et meil on juba olemas andmete minutiteks tegemise operaator, siis see sobib räsikoodiks imehästi. Piisab vaid andmed täisarvuks teha, kui juba ongi eri kellaaegade puhul erinev arv käes, sest sama arv minuteid päeva algusest saab olla vaid ühesuguse kellaaja korral.

```
    public override int GetHashCode(){
        return (int)this;
    }
```

Kui näiteks hoitaks ajahetke objektis nii kellaaegu kui kuupäevi, siis võiks olukord veidi raskemaks minna, sest neljabaidine int ei pruugi suuta näidata kõiki erinevaid väärtusi, mis aastatuhandete jooksul ette tulevad. Sel juhul tuleks juba keerukamaid arvutusi rakendada, et kõik objekti väljad oleksid räsikoodi arvutamisel rakendatud ja muudaksid tulemust. Samas aga oleks eri väärtusega objektidel kahe sarnase räsikoodi kokku juhtumine võimalikult haruldane. Ehk siis ligikaudu üks paari miljardi kohta - nii nagu neid int-muutuja erinevaid väärtusi on.

Kui funktsioonidega eeltöö tehtud, siis edasine operaatorite defineerimine on juba käkitegu: tuleb lihtsalt õige funktsioon välja kutsuda.

```
    public static bool operator!=(Kellaaeg k1, Kellaaeg k2){
      return !k1.Equals(k2);
    }
```

Ning teiste võrdlustega sarnaselt. Nüüd aga kood tervikuna.

```
using System;
class Kellaaeg{
    int tund, minut;
    public Kellaaeg(int utund, int uminut){
        tund=utund;
        minut=uminut;
        aegKorda();
    }
    void aegKorda(){
        while(minut>60){
            tund++;
            minut-=60;
        }
    }
    public int Tund(){return tund;}
    public int Minut(){return minut;}
    public void tryki(){
        Console.WriteLine("{0}:{1}", tund, minut);
```

```
   }
   public static  Kellaaeg operator+(Kellaaeg k1, Kellaaeg k2){
      return new Kellaaeg(k1.Tund()+k2.Tund(), k1.Minut()+k2.Minut());
   }
   public static implicit operator int(Kellaaeg k){
      return k.Tund()*60+k.Minut();
   }
   public static explicit operator double(Kellaaeg k){
      //kohustuslik muunduse näitamine
      return k.Tund()+k.Minut()/60.0;
   }
   public static explicit operator Kellaaeg(int minutid){
       return new Kellaaeg(minutid/60, minutid%60);
   }
   public int Compare(Kellaaeg k){
            //Võrdlusoperaatorite puhul soovitav defineerida
      int omaminutid=(int)this;
      int teiseminutid=(int)k;
      return omaminutid-teiseminutid;
   }
   public override bool Equals(Object ob){
      if (ob is Kellaaeg && (this.Compare(ob as Kellaaeg)==0)){
        return true;
      }
      return false;
   }
   public override int GetHashCode(){
      return (int)this;
   }
   public static bool operator==(Kellaaeg k1, Kellaaeg k2){
     return k1.Equals(k2);
   }
   public static bool operator!=(Kellaaeg k1, Kellaaeg k2){
     return !k1.Equals(k2);
   }
   public static bool operator<(Kellaaeg k1, Kellaaeg k2){
     return k1.Compare(k2)<0;
   }
   public static bool operator>(Kellaaeg k1, Kellaaeg k2){
     return k1.Compare(k2)>0;
   }
}

class Test{
   public static void Main(string[] arg){
      Kellaaeg k1=new Kellaaeg(12, 10);
      Kellaaeg k2=new Kellaaeg( 1,  4);
      Kellaaeg k3=new Kellaaeg( 1,  4);
      if(k2==k3){
         Console.WriteLine("Samad");
      }
      if(k2<k1){
         Console.WriteLine("Enne");
      }
   }
}

/*
C:\Projects\oma\naited>Operaatorid3
Samad
Enne
```

*/

Ülesandeid

* Alusta esimesest lühikesest näitest ning lisa kellaajale ka sekundid.

* Arvesta sekundeid ka kellaaegade liitmise juures.

* Juhul, kui Kellaaeg teisendatakse tüübiks int, anna väärtus endise minutite asemel sekundites.

* Loo klass Nihe, mille väljadeks on pikkus x- ja y-suunal.

* Defineeri operaator nihete liitmiseks.

* Võimalda nihkeid ka lahutada.

* Võrdlusoperaator loeb nihked võrdseks vaid juhul, kui mõlema koordinaattelje suunalised pikkused on võrdsed.

* Suurema ja väiksema võrdlemisel võrreldakse vaid nihete pikkusi.

* Hoolitse ka omaloodud räsikoodi eest.

Abivahendid

Erindid

Kui arvutil palutakse teha tema jaoks võimatu käsk, siis enamasti lõpeb programmi töö veateatega. Nii nagu järgnevas näites, kus tekst "Tere" püütakse muundada täisarvuks.

```
using System;
class Erind1{
   public static void Main(string[] arg){
      string tekst1="Tere";
      int arv1=int.Parse(tekst1);
      Console.WriteLine(arv1);
   }
}
```

Veateatest võib välja lugeda, et klassi `Erind1` käsklusest `Main` kutsuti välja `System.Number.ParseInt32`, mis omakorda kutsus välja käsu `StringToNumber`. Viimane aga jäi teksti arvuks muutmisega hätta. Anti välja veateade tüübist `System.FormatException` koos selgitusega, et etteantud sõna pole sobival kujul.

```
/*
C:\Projects\oma\naited>Erind1
Unhandled Exception: System.FormatException: Input string was not in a
correct format.
   at System.Number.StringToNumber(String str, NumberStyles options,
   NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style,
NumberFormatInfo info)
   at Erind1.Main(String[] arg)
*/
```

Vähemasti saime teada, mis juhtus, aga kasutaja ei pruugi sellise teatega kuigivõrd rahul olla. Eriti kui tuleb ette süsteemne veateateaken, mis püüab andmeid kuhugi saata või programmi siluma hakata ning keeldub eest ära minemast.

Püüdmine

Arvuti veateate saab asendada enese omaga. Või siis hoopis paluda veateate andmise asemel arv uuesti sisestada või sisestusest loobuda. Siin antakse lihtsalt omapoolne väike seletus toimunu kohta.

Veateate püüdmiseks on C# keeles olemas `try{}catch` plokk. Kõik looksulgude vahel juhtunud probleemid saadetakse lahendamisele `catch`-ossa, kus on juba programmeerija otsustada, mida tekkinud olukorras peale hakata.

```
using System;
class Erind2{
   public static void Main(string[] arg){
     try{
      string tekst1="Tere";
      int arv1=int.Parse(tekst1);
      Console.WriteLine(arv1);
     }catch(FormatException probleem){
        Console.WriteLine("Viga teisendusel: "+probleem.Message);
     }
   }
}
/*
C:\Projects\oma\naited>Erind2
Viga teisendusel: Input string was not in a correct format.
*/
```

Reageering tüübi põhjal

Sama koodilõigu juures võib ette tulla mitmesuguseid probleeme. Kord ei leita sobivat andmefaili, teinekord ei saa teksti arvuks muundada ning mõnikord võib hoopis ette tulla jagamine nulliga. Vanemate programmeerimiskeelte juures oli tavaks iga käsu juures kontrollida, kas see õnnestus, ning siis püüda koheselt reageerida. Kui kohene parandamine on võimalik, on selline lähenemine hea. Kui aga peab parandamiseks palju asju ära muutma, siis kulub palju tööd. Selle lihtsustamiseks erindid ja veahaldus välja mõeldigi.

Ploki lõpus oleva `catch`i sulgudesse kirjutatakse selline erinditüüp, millele ollakse valmis reageerima. Nagu eespool oli - `FormatException` tekkis sisendandmete vormingu vea tõttu ning sellele probleemile ka reageeriti. Võib tekkida aga olukord, kus sisendiks on küll kõik numbrid, aga kokku tuleb int-vormingu jaoks liiga suur arv. Sellisel juhul heidetakse hoopis `OwerflowException`. Eraldi `catch`idega püüdes saab nendele vigadele sobivalt reageerida.

Vigade klassid moodustavad isekeskis hierarhia. Selle puu juureks on klass nimega `Exception`. Tema alamklassideks on hulk .NET runtime käivitamisega seotud probleemiklasse, aga muuhulgas ka `SystemException`, mille alt siis omakorda kättesaadavad enamik meil programmides ettetulevaid System-nimeruumi objektidega seotud erindeid.

`SystemException`i enese alt leiab omakorda näiteks `ArithmeticException`i, mille juurest omakorda `OverflowException`i ja `DivideByZeroException`i. Kui me tahame liialt suurt arvu (ületäitumine) ning nulliga jagamist kontrollida sama `catch`i sees, siis võib piirduda `ArithmeticException`i püüdmisega. Kui aga kummagi olukorra jaoks on soov käivitada eri kood, siis tasub need eraldi kinni püüda.

Viga jääb kinni ainult ühes `catch`-plokis. Seepärast pannakse detailsemad erindid püüdmisel ettepoole ning üldisemad tahapoole. Muidu juhtuks, et teade jääb üldisematele tingimustele vastavasse plokki kinni ja väljavalitud lõiku kunagi ei pruugitagi. Tahtes kõik teated kindlasti

kätte saada, võib lõppu panna `catch(Exception)`. Sellele tüübile vastavad kõik veateated - ka need, mis on kõigist muudest püünistest juba mööda tulnud.

Juhul, kui soovitakse saabunud veateate andmetega midagi lähemat ette võtta, saab selle püüda omaette muutujasse ning sealtkaudu erindiobjektiga suhelda. Nagu näiteks

```
catch(FormatException probleem){
   Console.WriteLine("Viga sisendandmetega: "+probleem.Message);
}
```

Kui aga piisab vaid teadmisest, et juhtus vastavat tüüpi olukord ning sellest teadmisest on meile reageerimiseks küllalt, siis võib oma muutuja loomata jätta nagu näiteks

```
catch(OverflowException){
   Console.WriteLine("Liiga suur arv.");
}
```

`finally`-plokki püüniste lõpus kasutatakse käskluste jaoks, mis tulevad igal juhul ära teha. Näiteks faili sulgemine andmete lugemisel: isegi siis, kui lugemine ebaõnnestus, tuleb fail teistele kasutajatele kättesaadavaks teha. Aga mainimist väärib, et `finally`-plokki jõutakse siiski ainult juhul, kui viga polnud või sai sellele reageeritud. Nii et kindlaks lõpuni jõudmiseks on mõistlik panna viimaseks veapüüniseks ikkagi `catch(Exception)`. Kuigi - vahel soovitatakse, et pigem jäta erind püüdmata, kui et püüad midagi, millega sa mõistlikku peale hakata ei oska. Et kui tuleb ametlik veateade, on see vahel parem, kui omalt poolt vea peitmine, mis võib hiljem vigaste andmete näol kusagil kätte maksta.

Nüüd aga näide tervikuna. Käsurea parameetrina oodatakse numbreid, mille programm arvuks teisendab ning välja trükib. Juhtub aga midagi sobimatut, siis teatatakse vastav veateade.

```
using System;
class Erind3{
   public static void Main(string[] arg){
     try{
        if(arg.Length!=1){
          Console.WriteLine("Kasuta kujul: Erind3 sisendarv");
          return;
        }
        string tekst1=arg[0];
        int arv1=int.Parse(tekst1);
        Console.WriteLine("Sisestati edukalt "+arv1);
     } catch(FormatException probleem){
        Console.WriteLine("Viga sisendandmetega: "+probleem.Message);
     } catch(OverflowException){
        Console.WriteLine("Liiga suur arv.");
     } catch(Exception){
        Console.WriteLine("Tundmatu probleem");
     } finally{
        Console.WriteLine("Plokk otsas");
     }
   }
}
```

```
/*
C:\Projects\oma\naited>Erind3
Kasuta kujul: Erind3 sisendarv
Plokk otsas
C:\Projects\oma\naited>Erind3 tere
Viga sisendandmetega: Input string was not in a correct format.
Plokk otsas
C:\Projects\oma\naited>Erind3 1234567890123456
Liiga suur arv.
Plokk otsas
C:\Projects\oma\naited>Erind3 78
Sisestati edukalt 78
Plokk otsas
*/
```

Püüdmine alamprogrammist

Veapüüniste tähtsaim eelis varasema veakoodinduse ees ongi kogu rakenduse alamprogrammide rägastikus tekkinud probleemide transport konkreetsetesse kohtadesse kokku, kus nendega üheskoos on vahel mõnevõrra kergem hakkama saada.

Järgnevas näites tekibki tõenäoline probleem alamprogrammis nimega `LoeArv` juhul, kui sisendiks pole arv. Veateade aga trükitakse alles `Main`-meetodi juures. Nõnda võib näiteks paluda kasutajal arvutamise jaoks anda mitu arvu. Kui aga kasvõi ühel korral sisestusel eksiti, on tulemus ikka sama - tulemust pole võimalik kokku saada. Ning sellest antakse veapüünises ka teada.

```
using System;
class Erind4{
   public static int LoeArv(){
      Console.WriteLine("Palun arv:");
      string s=Console.ReadLine();
      int a=int.Parse(s);
      return a;
   }
   public static void Main(string[] arg){
     try{
      int arv1=LoeArv();
      Console.WriteLine("Kirjutati: "+arv1);
     }catch(FormatException probleem){
        Console.WriteLine("Viga teisendusel: "+probleem.Message);
     }
   }
}
/*
D:\kodu\0606\opikc#>Erind4
Palun arv:
5
Kirjutati: 5
*/
```

Erindi heitmine

Sugugi ei pea leppima vaid arvuti enese antud veateadetega. Kui ikka oma programmis paistab, et midagi läheb väga käest ära, siis on vahel kasulik ise märku anda, et sarnaselt edasi toimida pole enam mõtet. Näiteks, kui arvutuse algandmed on ilmselgelt valed (kolmnurga üks külg pikem kui teised kaks kokku), siis võib julgesti enne arvutamist teada anda, milles asi ning heita selleteemalise erindi. Edasi on juba vastavat koodilõiku väljakutsuva programmeerija ülesandeks silumise käigus kindlaks teha, millest probleem tekkis ning vastavalt edasi toimida.

Siin näites lihtsalt keelati sajast suuremate arvude sisestus. Kui arv juhtub liiga suur olema, heidetakse erind. Lihtsuse mõttes pole oma tüüpi loodud, kasutatakse `SystemException`it. Kuigi - vähegi pikema programmi selguse huvides oleks oma tüübi loomine kasulik. Et peaprogrammis pole `SystemException`i jaoks veapüünist, siis tuleb ette süsteemne veateade, mille järele programmeerija peab juba ise edasi mõtlema, mida edasi teha.

```
using System;
class Erind5{
   public static int LoeArv(){
      Console.WriteLine("Palun arv:");
      string s=Console.ReadLine();
      int a=int.Parse(s);
      if(a>100){
        throw new SystemException("Liiga suur arv");
      }
      return a;
   }
   public static void Main(string[] arg){
     try{
      int arv1=LoeArv();
      Console.WriteLine("Kirjutati: "+arv1);
     }catch(FormatException probleem){
        Console.WriteLine("Viga teisendusel: "+probleem.Message);
     }
   }
}
/*
D:\kodu\0606\opikc#>Erind5
Palun arv:
789
Unhandled Exception: System.SystemException: Liiga suur arv
   at Erind5.LoeArv()
   at Erind5.Main(String[] arg)
*/
```

Ülesandeid

* Katseta näite "Erind2" juures, kuidas käitub programm juhul, kui ette anda veatu arv.

* Muuda täisarvu käsklused reaalarvu omadeks ja leia, mis kasutamisel muutus.

* Loo tsükkel, mille abil küsitakse arvu senikaua, kuni saadakse sobiv sisend.

* Muuda näidet "Erind5" nõnda, et see annaks peaprogrammis viisaka seletuse ka omaheidetud erindi korral.

* Loo erindeid kasutades programm, mis suurendaks faili arv.txt sisu ühe võrra. Kui fail puudub, või failis pole arv, siis antakse selgitusega veateade.

* Kui failis olev arv ületab 365, siis anna välja omapoolne erind ning püüa sellele reageerida.

Enum

Ikka leidub kohti, kus on võimalik teha piiratud arv valikuid. Asukoht Eestis on ühes maakondadest. Ühissõiduk on üldjuhul rong, tramm, troll või buss jne. Kui programmikoodis tuleb leida käitumisjuhis ühele etteantud valikutest, siis on enum hea abivahend. Maakonna nime saab kirjutada mitmeti. Olgu siis "Harjumaa", või "Harju maakond", rääkimata suurte ja väikeste tähtede ning tühikute erisusest. Andmebaaside puhul kasutatakse üldjuhul võimalust, et ei kirjutata inimese andmete juurde maakonna nime, vaid pannakse selle maakonna kood. Ning selle järgi on vajadusel võimalik teisest andmetabelist järele vaadata, millise maakonnaga siis päriselt tegu. Midagi sarnast toimub ka enumi puhul. Ehk siis vastavas loetelus kirjeldatakse ära kõik võimalikud väärtused. Ning hiljem programmi sees pole võimalik enam vastavat nimetust valesti kirjutada ilma, et kood kompileerimata jääks. Sedasi on võimalik vältida vigu, mis muidu üllatavatel hetkedel võiksid avalduda. Tüüpiliseks kasutuskohaks on näiteks alamprogrammi parameetrid, kus enumi abil määratakse, kuidas just sel korral vastavate andmetega tuleb käituda. Järgnevas näites siis tuuakse tugevuse kohta kolm konstanti: tumm, ühekordne ja mitmekordne. Ning praegusel juhul alamprogrammis esimese variandi puhul jäetakse etteantud tekst sootuks trükkimata. Teisel juhul trükitakse ühe korra ning viimasel juhul mitu korda. Kui aga ühekordne kirjutatuks nõrga g-ga, siis jäänuks kood kompileerimata. Pealtnäha iseenesestmõistetav. Aga kui enumi asemel olnuks kasutatud stringi, siis just sellised vead on kerged tulema.

```
using System;
namespace Enumeratsioon1{
   enum tugevus{tumm, yhekordne, mitmekordne};
   class Trykkimine{
      static void Tryki(string tekst, tugevus t){
           if(t==tugevus.yhekordne){
              Console.WriteLine(tekst);
           }
           if(t==tugevus.mitmekordne){
              Console.WriteLine(tekst);
              Console.WriteLine(tekst);
           }
      }
      public static void Main(string[] arg){
         Tryki("Tere", tugevus.yhekordne);
      }
   }
}
/*
E:\jaagup\07\12>Enumeratsioon1
Tere
*/
```

Ülesandeid

* Loo klass vooluallika andmete hoidmiseks (pingevahemik, enum näitamaks kas tegemist alalis- või vahelduvvooluga)

* Koosta sellistest vooluallikatest mitmesuguste väärtustega massiiv.

* Koosta alamprogramm, mis saab parameetriks soovitud pinge, voolutüübi ja vooluallikate massiivi ning trükib välja soovitule vastavate vooluallikate andmed.

Andmekollektsioonid

Andmetega ümberkäimisele kulub märgatav osa arvutite ja programmeerija ajast. 2000 aasta paiku arvati selleks osaks olema ligikaudu kolmandik. Nüüd ehk veidi vähem, kuid tähtsus on ikka alles jäänud. Et põhioperatsioonidele ei kuluks liialt palju tähelepanu, selleks on programmeerimiskeeltes välja mõeldud valmis vahendid andmeoperatsioonideks. Nii ka C# puhul.

ArrayList

Hea lihtne koht andmete hoidmiseks ja kätte saamiseks. Võrreldes tavalise massiiviga pole vaja elementide arvu kohe ette määrata. `ArrayListi` objekt hoolitseb ise selle eest, et oleks parajalt ruumi sissepandud andmete hoidmiseks. Iga `Add`-käsklusega lisatakse sissepandud väärtus olemasolevate lõppu. Käsuga `Contains` võib kontrollida otsitava elemendi olemasolu. `Count` näitab elementide arvu. `Insert`-käsklus lisab uue elemendi soovitud järjekorranumbriga kohale, lükates ülejäänud ühe koha võrra edasi. `IndexOf` aitab soovitud väärtust otsida. Viimase puudumisel tagastatakse järjekorranumbrina -1. Ning `foreach`-tsükkel sobib kõigi elementide läbi käimiseks.

```
using System;
using System.Collections;
class Kollektsioon1{
   public static void Main(string[] arg){
      ArrayList nimed=new ArrayList();
      nimed.Add("Kati");
      nimed.Add("Mati");
      nimed.Add("Juku");
      if(nimed.Contains("Mati")){
         Console.WriteLine("Mati olemas");
      }
      Console.WriteLine("Nimesid kokku "+nimed.Count);
      nimed.Insert(1, "Sass");
      Console.WriteLine("Mati asub kohal "+nimed.IndexOf("Mati"));
      Console.WriteLine("Mari asub kohal "+nimed.IndexOf("Mari"));
      foreach(string eesnimi in nimed){
         Console.WriteLine(eesnimi);
      }
   }
}

/*
```

```
D:\kodu\0606\dotnet>Kollektsioon1
Mati olemas
Nimesid kokku 3
Mati asub kohal 2
Mari asub kohal -1
Kati
Sass
Mati
Juku
*/
```

Sortimine

Andmete järjestamiseks on välja mõeldud hulk algoritme, millel enamikul mõni eriline koht, kus ta teistest kiiremini töötab. Kui aga meid rahuldab korralik "Harju keskmine" tulemus, siis võib kasutada `ArrayList`ile sisseehitatud käsku `Sort`, mis elemendid kasvavasse järjekorda sätib.

Tahtes andmeid trükkides teada, mitmenda elemendi juures ollakse, tuleb ükshaaval neid järjekorranumbri abil küsida. `ArrayList`i elemendi poole saab pöörduda sarnaselt nagu massiivigi elemendi poole kantsulgude abil.

```
using System;
using System.Collections;
class Kollektsioon1a{
   public static void Main(string[] arg){
      ArrayList nimed=new ArrayList();
      nimed.Add("Kati");
      nimed.Add("Mati");
      nimed.Add("Juku");
      nimed.Sort();
      for(int i=0; i<nimed.Count; i++){
          Console.WriteLine(nimed[i]);
      }
   }
}

/*
D:\kodu\0606\opikc#>Kollektsioon1a
Juku
Kati
Mati
*/
```

Tüübimäärang

Eelkirjeldatud `ArrayList` on lahke - lubab enesesse panna ja sealt võtta igasugu andmetüüpe. Mõnikord on see mugav, kuid vähegi pikemate programmide juures võivad kogemata nimistud sassi minna ja näiteks sünniaasta andmed sattuda näiteks hoopis perekonnanime kohale. Et programmeerimiskeeltes püütakse vea võimalusi vältida, siis on alates .NET 2.0st lisatud uus nimeruum System.Collections.Generic, kus kasutatavate andmestruktuuride juures

tuleb kohe algul ära määrata, millist tüüpi andmeid kollektsiooni panna tohib. Ehk siis tekstiliste andmete hoidmiseks sobib

```
LinkedList<string> nimed=new LinkedList<string>();
```

Kui tegemist oleks arvudega, siis peaks <> märkide vahel olema sõna `int`, mõne muu andmetüübi puhul selle nimi. Lisamiseks ja küsimiseks mõnevõrra teistsugused käsud, aga kõik vajaliku saab tehtud. Kuna `ArrayList`i puhul hoitakse andmeid mälus massiivina, siis on arvuti jaoks lihtne ülesanne anda vastavalt järjekorranumbrile element. Samas aga jada algusesse lisamine võib suurema andmehulga puhul ootamatult palju ressursse nõuda. `LinkedList`iga on vastupidi: konkreetse elemendi poole pöördumine võib raske olla. Mööda ahelat edasi-tagasi liikumine ning elementide lisamine või eemaldamine käib kiiresti ka pika ahela juures.

```
using System;
using System.Collections.Generic;
class Kollektsioon2{
   public static void Main(string[] arg){
      LinkedList<string> nimed=new LinkedList<string>();
      //lubab ainult stringe
      nimed.AddLast("Kati");
      nimed.AddLast("Mati");
      nimed.AddLast("Juku");
      if(nimed.Contains("Mati")){
         Console.WriteLine("Mati olemas");
      }
      Console.WriteLine("Nimesid kokku "+nimed.Count);
      nimed.AddAfter(nimed.Find("Kati"), "Sass");
      LinkedList<string>.Enumerator enumr=nimed.GetEnumerator();
      while(enumr.MoveNext()){
         string eesnimi=enumr.Current;
         Console.WriteLine(eesnimi);
      }
   }
}

/*
D:\kodu\0606\dotnet>Kollektsioon2
Mati olemas
Nimesid kokku 3
Kati
Sass
Mati
Juku
*/
```

Järjekord

Näiteks graafikaülesannete juures on vajalik andmeid panna järjekorda ootele ning neid siis sealt sissepaneku järjekorras välja küsida. Iseenesest on sarnane toiming ka `LinkedList`i abil tehtav, aga juba .NET versioonis 1.0 oli selle tarvis omaette klass loodud, nimeks `Queue`.

Kasutamine lihtne: käsuga `Enqueue` lisatakse andmeid ning `Dequeue` võetakse neid teisest otsast ära.

```
using System;
using System.Collections;
class Kollektsioon3{
   public static void Main(string[] arg){
      Queue jarjekord=new Queue();
      jarjekord.Enqueue("Juku");
      jarjekord.Enqueue("Kati");
      jarjekord.Enqueue("Mati");
      while(jarjekord.Count>0){
         string eesnimi=jarjekord.Dequeue() as string;
         Console.WriteLine(eesnimi);
      }
   }
}

/*
D:\kodu\0606\dotnet>Kollektsioon3
Juku
Kati
Mati
*/
```

Paisktabel

Vahend andmepaaride hoidmiseks. Kord indekseerimise juures juba tutvusime selle vahendiga, siin nüüd vaatame talle veel korra otsa. Paisktabelis sobib hoida näiteks konfiguratsioonifailist loetud omaduste väärtusi, kasutajanimele vastavaid seadeid või tõlkefaili andmeid. Põhiliseks tingimuseks on, et võti (kasutajanimi või omaduse nimi) ei kordu ning võtme järgi saab küsida väärtuse. Siin näites hoitakse inimeste nimedele vastavaid hindeid.

```
      if(ht.ContainsKey("Kati")){
          Console.WriteLine("{0}", ht["Kati"]);
      }
```

Kontrollitakse, kas Kati on nimede hulgas olemas. Kui jah, siis trükitakse ta hinne.

```
      ht["Sass"]=((int)ht["Sass"])-1;
```

Sassi hinnet alandatakse ühe võrra.

```
      ht.Remove("Mati");
```

Mati eemaldatakse nimekirjast.

Tahtes kõik andmed kätte saada, aitab jälle enumeraator, ainult et igal enumeraatori elemendil on võti ja väärtus. Siin trükitakse nad lihtsalt välja, aga eks igaüks tea ise paremini, mida tal oma programmis nendega kõige mõistlikum teha on.

```
      IDictionaryEnumerator enumr=ht.GetEnumerator();
      while(enumr.MoveNext()){
         string eesnimi=enumr.Key as string;
         int hinne=(int)enumr.Value;
         Console.WriteLine("{0}: {1}", eesnimi, hinne);
      }
```

Ning kogu näide tervikuna.

```
using System;
using System.Collections;
class Kollektsioon4{
   public static void Main(string[] arg){
      Hashtable ht=new Hashtable();
      ht.Add("Juku", 3);
      ht.Add("Kati", 5);
      ht.Add("Mati", 4);
      ht.Add("Sass", 4);
      if(ht.ContainsKey("Kati")){
          Console.WriteLine("{0}", ht["Kati"]);
      }
      ht["Sass"]=((int)ht["Sass"])-1;
      ht.Remove("Mati");
      IDictionaryEnumerator enumr=ht.GetEnumerator();
      while(enumr.MoveNext()){
         string eesnimi=enumr.Key as string;
         int hinne=(int)enumr.Value;
         Console.WriteLine("{0}: {1}", eesnimi, hinne);
      }
   }
}

/*
D:\kodu\0606\dotnet>Kollektsioon4
5
Kati: 5
Juku: 3
Sass: 3
*/
```

Ülesandeid

* Küsi kasutajalt arve, kuni ta sisestab nulli. Salvesta ArrayListi. Väljasta need arvud tagurpidises järjekorras.

* Proovi eelmine ülesanne lahendada LinkedListi abil. Omadus Last annab loetelu viimase elemendi, RemoveLast() kustutab viimase.

* Loe tekstifailist arvud, väljasta nad sorteerituna teise tekstifaili.

* Loe tekstifailist arvud. Teise tekstifaili väljasta, mitu korda iga arv esines.

Mallid

Objektorienteeritus võimaldab alamklasside eksemplare omistada ülemklassi tüüpi muutujatele. Nõnda saab lahendada enamiku olukordi, kus koodilt nõutakse paindlikkust ning võimet veidi erinevaid objekte ühiselt hoida või käidelda. Kus pole võimalik objekte omistada muidu pärimispuu järgi, seal tuleb appi teadmine, et kõik pärineb ühisest ülemklassist System.Object. Või siis saab eri pärimispuudest tulnud klasside ühiseid käsklusi kasutada liideste abil. Nii et kõik vajalik peaks sellega olemas olema.

Ometigi on C# juurde kaasa võetud C++ist mallid ehk šabloonid ehk geneerilisus. Ehk siis võimalus kasutatavaid andmetüüpe määrata pärast kasutatava klassi koodi enese valmiskirjutamist. Sellega kaasneb vähemasti kaks head omadust:

* Kui andmetüüp on täpselt määratud, siis on karta vähem valest omistamisest tingitud vigu.

* Kompilaatoril on võimalik koodi optimeerida konkreetse andmetüübi omadustest lähtudes ning programmi töö käigus ei pea kulutama aega tegeliku andmetüübi kontrollimisele.

Võimalust kasutatakse tihti geneeriliste andmekollektsioonide juures. Kui muidu oli hoiustatud andmete kohta teada ainult, et need on klassi Object järglased (ehk siis nagu polnudki tüübi kohta suurt midagi teada), siis geneerilise Listi puhul saab määrata näiteks, et loetelus esinevad elemendid on täisarvud. Ning selle põhjal on edaspidises kasutuses teada, et vastavast loetelust välja võetavad andmed on ka sama tüüpi. Nagu järgnevast näitest näha, siis ka loetelu läbimiseks mõeldud enumeraatorile tuleb sama tüüp määrata.

Andmestruktuuri ülesehituse eripärast lähtudes on LinkedListi läbikäimine enumeraatori abil tunduvalt ökonoomsem kui järjestikuste andmete küsimine elemendi järjekorranumbri järgi. Konkreetse järjekorranumbriga kohani jõudmiseks tuleb järjekorranumbri puhul enne kõik elemendid listisiseselt läbi jalutada. Enumeraator aga mõistab andmeid ilusti järjest võtta.

```
using System;
using System.Collections.Generic;
class GeneerilineList{
        static void Main(string[] args)
        {
            LinkedList<int> loetelu = new LinkedList<int>();
            loetelu.AddLast(5);
            loetelu.AddLast(3);
            LinkedList<int>.Enumerator ahel = loetelu.GetEnumerator();
            while (ahel.MoveNext()) {
                Console.WriteLine(ahel.Current);
            }
        }
}
```

Geneerilisi klasse saab ka ise luua. Järgneva näitena pandi kokku lihtne väärtusehoidla. Kui klassi nime taga on kirjeldamise ajal <> märkide vahel täht, siis seda tähte saab klassi sees

kasutada andmetüübi kirjeldamiseks. Sarnaselt, nagu võin andmetüübiks märkida int või string, sarnaselt võin andmetüübiks kirjutada T. Ja alles pärast - siis kui klassist luuakse eksemplar. Alles siis määratakse täpsemalt, millist tüüpi seal andmete hoidmiseks tegelikult kasutatakse.

Kui katseprogrammi loomisel kirjutatakse, et

```
Hoidla<int> h=new Hoidla<int>();
```

siis sellega määratakse selles konkreetses hoidlas hoitavate väärtuste tüübiks int ning midagi muud sinna panna ei saa.

```
namespace Geneeriline1{
 public class Hoidla<T>{
  T sisu;
  public void Pane(T sisu){
     this.sisu=sisu;
  }
  public T Kysi(){
     return sisu;
  }
 }
 public class Katsetus{
    public static void Main(string[] arg){
       Hoidla<int> h=new Hoidla<int>();
       h.Pane(3);
       System.Console.WriteLine(h.Kysi());
    }
 }
}
```

Kasutatavatele andmetüüpidele saab ka mõningasi piiranguid seada - selleks, et nendega koodis mõnevõrra rohkem midagi hiljem teha oleks. Sest ilma määramata pole isegi teada, kas kasutatav tüüp on struct või class. Neil aga mäluhalduse poolest küllalt erinevad omadused. Kui aga siin teatan, et T on klass, siis on vastavat tüüpi muutujale võimalik anda algväärtuseks null näitamaks, et selle muutuja kaudu ühegi objekti juurde ligi ei pääse. Ning sealtkaudu samuti võimalik küsida, et kas me hoidlas on juba sisu olemas.

```
using System;
namespace Geneeriline2{
 public class Hoidla<T> where T:class{
  T sisu=null;
  public void Pane(T sisu){
     this.sisu=sisu;
  }
  public T Kysi(){
     return sisu;
  }
  public bool KasOlemas(){
    return sisu!=null;
  }
 }
```

```
 public class Katsetus{
    public static void Main(string[] arg){
       Hoidla<String> h=new Hoidla<String>();
       h.Pane("Kuku");
       if(h.KasOlemas()){
         System.Console.WriteLine(h.Kysi());
       }
    }
 }
}
```

Enese loodud geneerilistes klassides saab kasutada ka varemvalminud geneeriliste klasside võimalusi. Ehk siis kui on põhjust või tahtmist andmete hoidmist omale sobival moel täiustada või piirata, siis selleks on täiesti võimalus olemas. Omaloodud klassi külge antud geneerilise andmetüübi võib rahumeeles edasi kanda klassi sees loodud andmekollektsiooni eksemplarile ning sinna hiljem vastavat tüüpi andmeid edastada. Siin näites on loodud klass, mis hoiab oma andmeid Listi sees. Tuues muuhulgas aga juurde käskluse loetelust juhusliku elemendi tagastamiseks.

```
using System;
using System.Collections.Generic;
namespace Geneeriline3{
 public class Hoidla<T> where T:class{
  List<T> loetelu=new List<T>();
  Random r=new Random();
  public void Pane(T sisu){
     loetelu.Add(sisu);
  }
  public T Kysi(){
     return loetelu[r.Next(loetelu.Count)];
  }
  public bool KasOlemas(){
    return loetelu.Count>0;
  }
 }
 public class Katsetus{
    public static void Main(string[] arg){
       Hoidla<String> h=new Hoidla<String>();
       h.Pane("Kuku");
       h.Pane("Ahoi");
       h.Pane("Tere");
       if(h.KasOlemas()){
         System.Console.WriteLine(h.Kysi());
       }
    }
 }
}
```

Ülesandeid

* Katseta LinkedListi string-tüüpi andmetega

* Katseta LinkedListi omaloodud klassiga tüübist andmetega

* Loo klass kahe samast tüübist väärtuse hoidmiseks.

* Loo klass, mille puhul saab määrata elementide maksimummäära, mis klassi sees olevasse hoidlasse panna tohib.

Atribuudid

Atribuute kasutakse klassi ja seal sees leiduva omaduste kirjeldamiseks. Teisele programmeerijale võib kommentaaride teel kirja panna, et mida miski käsklus teeb või kuidas oleks seda hea kasutada. Teine rakendus aga üldjuhul inimkeeles kirjutatud kommentaare lugeda ei mõista. Samas on aga hea, kui saab käsule juurde panna seletuse, kuidas seda mõnes graafilises arendusvahendis näitama peaks. Või siis teate, et seda käsku pole vaja praegu käivitada. Atribuudid just sellisteks teadeteks mõeldud on. Mingi hulk on neid olemas .NETi enese poolt. Kui aga omal suurema eripärase raamistiku ehitamine käsil, siis võivad ka omaloodud atribuudid omal kohal olla. Algusnäiteks sobibki tingimuslik käivitamine. Alamprogrammile trüki on lisatud atribuut Conditional. See süsteemne atribuut teatab, et käsklus pannakse tööle vaid juhul, kui konstant nimega TESTSEISUND on defineeritud. Kui sellist konstanti pole, siis jääb käskluse töö lihtsalt tegemata.

Konstanti saab kompileerimisel defineerida näiteks järgnevalt

```
D:\kasutaja\jaagup\proov3>csc /define:TESTSEISUND Logimine1.cs
```

Sellisel juhul kompilaator leiab, et konstant on olemas ning käsk tuleb oma väljakutsel käima panna nagu tavaliselt.

Kui aga kompileerida ilma defineerimata,

```
D:\kasutaja\jaagup\proov3>csc Logimine1.cs
```

siis käsklust ei käivitata. Lihtne.

```
using System;
using System.Diagnostics;
class Logimine1{
   [Conditional("TESTSEISUND")]
   static void tryki(string teade){
      Console.WriteLine(teade);
   }
   public static void Main(string[] arg){
      tryki("algus");
   }
}

/*
D:\kasutaja\jaagup\proov3>csc Logimine1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
```

```
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
D:\kasutaja\jaagup\proov3>Logimine1
D:\kasutaja\jaagup\proov3>csc /define:TESTSEISUND Logimine1.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
D:\kasutaja\jaagup\proov3>Logimine1
algus
*/
```

Defineerida võib ka koodi sees. Et kui parajasti soovitakse abiteateid trükkida, siis kirjutatakse koodi algusesse

```
#define TESTSEISUND
```

ning kompilaatori jaoks ongi vastav konstant olemas ja käsklus käivitatakse. Kui mitte, siis mitte nagu ennegi.

```
#define TESTSEISUND
using System;
using System.Diagnostics;
class Logimine1a{
   [Conditional("TESTSEISUND")]
   static void tryki(string teade){
      Console.WriteLine(teade);
   }
   public static void Main(string[] arg){
      tryki("algus");
   }
}

/*
D:\kasutaja\jaagup\proov3>csc Logimine1a.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
D:\kasutaja\jaagup\proov3>Logimine1a
algus
*/
```

Omaloodud atribuut

Atribuute võib ka ise luua. Näiteks selleks, et mõningad käsklused vajadust mööda esile tuua. Selleks tuleb teha klassi System.Attribute alamklass ning sinna vajadust mööda konstruktorisse või omadustesse andmeid lisada. Käskluse lihtsalt ära märkimiseks pole aga muud vaja, kui selle ette atribuut panna. Hiljem ühe klassi koodi sees teist klassi uurides saab uudistada, et millised atribuudid viimase külge pandud on ning mis neist järeldada võib. Siin pannakse käima sellised käsklused, millele on lisatud HuvitavMeetodAttribute. Ja enne seda trükitakse sõna "huvitav", et oleks näha, millise käsklusega tegu on.

```
using System;
```

```
using System.Reflection;
namespace Atribuudid2{
   [AttributeUsage(AttributeTargets.Method)]
   class HuvitavMeetodAttribute: System.Attribute{
   }
   public class ValitudMeetodid{
      [HuvitavMeetodAttribute()]
      public void nuputa(){
         Console.WriteLine("Rakendus nuputab");
      }
      public void tervita(){
         Console.WriteLine("Tere");
      }
   }
   public class AtribuudiProov{
      public static void Main(string[] arg){
         ValitudMeetodid v=new ValitudMeetodid();
         MethodInfo[] m= typeof(ValitudMeetodid).GetMethods();
         foreach(MethodInfo mi in m){
            MethodAttributes ma=mi.Attributes;
            foreach(Attribute at in Attribute.GetCustomAttributes(mi)){
               if(at.GetType()==typeof(HuvitavMeetodAttribute)){
                  mi.Invoke(v, null);
                  Console.Write("huvitav ");
               }
            }
            Console.WriteLine(mi.Name);
         }
      }
   }
}

/*
D:\ctrell>Atribuudid2
Rakendus nuputab
huvitav nuputa
tervita
GetType
ToString
Equals
GetHashCode
*/
```

Atribuutide parameetrid

Atribuutidele kannatab ka andmeid ette anda. Olgu siis kohustuslikud väljad või vaikimisi väärtustega valikulised väljad. Esimesed antakse konstruktori parameetrina, teised omadustena. Ning omadustega atribuudi puhul peab andmete sisestamisel ette ütlema, millise nime alla väärtus läheb.

Järgnevas näites antakse soovitav käivituskordade arv kaasa konstruktorist ning selle väärtuse saab kätte omadusest Kogus. Valikuline atribuut Koostaja antakse sisse omaduse kaudu. Ning nagu näha käskluse Tervita juurest, võib selle ka andmata jätta.

Atribuudispetsiifiliste omaduste jaoks tuleb infokogumise juurest leitud atribuut kõigepealt sobivasse tüüpi muundada.

```
  HuvitavMeetodAttribute ha=at as HuvitavMeetodAttribute;
```

Edaspidi saab sealt andmeid kätte nagu tavalisest objektist. Küsitakse, kelle loodud käsklusega tegu, mitu korda käivitada ning ongi tegutsemisjuhised olemas.

```
using System;
using System.Reflection;
namespace Atribuudid3{
   [AttributeUsage(AttributeTargets.Method)]
   class HuvitavMeetodAttribute: System.Attribute{
      private int _kogus;
      private string _koostaja="tundmatu";
      public HuvitavMeetodAttribute(int Ukogus){
         _kogus=Ukogus;
      }
      public int Kogus{
        get{return _kogus;}
      }
      public string Koostaja{
        get{return _koostaja;}
        set{_koostaja=value;}
      }
   }
   public class ValitudMeetodid{
      [HuvitavMeetodAttribute(3, Koostaja="Jaagup")]
      public void nuputa(){
         Console.WriteLine("Rakendus nuputab");
      }
      [HuvitavMeetodAttribute(1)]
      public void tervita(){
         Console.WriteLine("Tere");
      }
   }
   public class AtribuudiProov{
      public static void Main(string[] arg){
         ValitudMeetodid v=new ValitudMeetodid();
         MethodInfo[] m= typeof(ValitudMeetodid).GetMethods();
         foreach(MethodInfo mi in m){
            MethodAttributes ma=mi.Attributes;
            foreach(Attribute at in Attribute.GetCustomAttributes(mi)){
               if(at.GetType()==typeof(HuvitavMeetodAttribute)){
                 HuvitavMeetodAttribute ha=at as HuvitavMeetodAttribute;
                 for(int i=0; i<ha.Kogus; i++){
                  mi.Invoke(v, null);
                 }
                 Console.WriteLine("Koostas "+ha.Koostaja);
                 Console.Write("huvitav ");
               }
            }
            Console.WriteLine(mi.Name);
         }
      }
   }
}

/*
E:\jaagup\07\12>Atribuudid3
Rakendus nuputab
Rakendus nuputab
```

```
Rakendus nuputab
Koostas Jaagup
huvitav nuputa
Tere
Koostas tundmatu
huvitav tervita
GetType
ToString
Equals
GetHashCode
*/
```

Ülesandeid

* Katseta omaloodud klassis tingimusliku käivitamise atribuuti

* Otsi abiinfost üles atribuut System.Obsolete ning märgi selle abil mõni omaloodud klassi meetod vananenuks. Tutvu teadetega kompileerimisel

* Loo omaloodud atribuut meetodi loomise kuupäeva ning meetodi kohta selgitusi sisaldava URLi hoidmiseks. Kuupäev on kohustuslik, selgituste jaoks pane vaikimisi aadressiks oma koduleht. Koosta koodi uuriv programm, mis atribuudi olemasolul näitaks välja käskluse loomise kuupäeva. Abiinfo URLilt saabuvad andmed salvesta võimaluse korral omaette faili.

Andmebaasiliides

Ühenduse loomine, päring

Kui juhtub, et arvutis või kättesaadavas võrgus on kasutada mõni andmebaas, siis suure tõenäosusega saab sealsete andmete poole pöörduda ja omakoostatud C#-programmi kaudu. Ühe-kaks väärtust võib olla lihtsam baasi enese juurde käiva halduskeskkonna abil paika sättida. Kui aga andmeid on pidevalt ja kümneid. Või siis pole andmetega tegelejaks mitte teie ise, vaid keegi muu. Või soovite, et teated toimingutest jõuaksid automaatselt andmebaasi – sellisel juhul on kasulik koostatud programmilõigu peale mõelda.

Esimene näide eeldab, et masinas nimega RINDE on üles seatud SQL-serveri eksemplar nimega SQLEXPRESS. Ning sinna sisse on loodud andmebaas nimega proovibaas. Ja selles baasis on tabel nimega "inimesed". Ning kõige esimeseks tulbaks inimeste tabelis on eesnimi. Sellisel juhul, kui kõik etapid õnnestuvad, võib loeteluna näha tabelis olevate inimeste eesnimesid. Eks igaüks saab masina, tabeli ja muud andmed enese omade vastu vahetada, ning ongi lihtne üldkasutatav näide, kuidas vajalikke andmeid baasist oma programmi sisse välja meelitada. Mõningad lähemad seletused toimuva kohta.

```
Muutujasse constr (connection string) salvestatakse kõigepealt
seletus programmi jaoks, kust andmebaas üles leida.
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=proovibaas; "+
       "Integrated Security=SSPI; Persist Security Info=False";
```

Eraldi muutujasse pannakse kirja SQL-lause, mille abil loodame andmeid küsida.

```
     string lause="SELECT eesnimi FROM inimesed";
```

Järgnevalt luuakse ja avatakse ühendus andmebaasiga. See toiming võib mõnikord märgatava hulga sekundeid aega võtta.

```
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
```

Et arvuti oskaks andmeid küsida, selleks luuakse SQL-käsklus, kus baasiühendusega seotakse SQL-lause. Siinse vaheetapi juures on keerulisematel juhtudel näiteks võimalus käsklusele parameetreid lisada. Siin aga piirduma lihtsama variandiga.

```
     SqlCommand cm = new SqlCommand(lause, cn);
```

Saabuvate andmete püüdmiseks on SQL-serveri puhu `SqlDataReader`. `SqlCommand`i käsklus `ExecuteReader` väljastab vastavat tüüpi objekti, mille kaudu programm omakorda saab andmeid küsima hakata. Selline vaheetapp on vajalik, et programm saaks vajadusel hakkama ka väga suure andmehulgaga. Kui andmete vahendamise jaoks on omaette objekt, kelle kaudu vaikselt andmeid küsima hakatakse, siis ei pea programm saabuvaid andmeid kõiki korraga enesele mällu laadima, vaid jätab selle töö `SqlDataReader`i hooleks.

```
SqlDataReader reader=cm.ExecuteReader();
```

Alles edaspidises tsüklis võetakse inimeste andmed ükshaaval ja toimetatakse nendega. Käsklus `Read` viib lugemiskursori ühe rea võrra edasi – esimesel korral siis esimese inimese juurde. Ning käsklus `GetString` annab etteantud järjekorranumbriga veerust andmed kätte. Nagu näha, hakkavad veerud lugema nullist.

```
while(reader.Read()){
    Console.WriteLine(reader.GetString(0));
}
```

Iga ühenduse kasutamise järel on viisakas see kinni panna. Mis juhtub, kui ühendus lahti jäetakse sõltub juba otsestest oludest. Aga kinni pandult on ressursside raiskamise mure programmeerija käest ära.

```
cn.Close();
```

Ning kood tervikuna.

```
using System;
using System.Data.SqlClient;
class Baasiproov1{
   public static void Main(string[] arg){
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=proovibaas; "+
       "Integrated Security=SSPI; Persist Security Info=False";
     string lause="SELECT eesnimi FROM inimesed";
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
     SqlCommand cm = new SqlCommand(lause, cn);
     SqlDataReader reader=cm.ExecuteReader();
     while(reader.Read()){
         Console.WriteLine(reader.GetString(0));
     }
     cn.Close();
   }
}

/*
D:\kodu\0606\dotnet>Baasiproov1
Juku
Mati
Sass
```

```
*/
```

Kui soovitakse baasiga rohkem inimkeeli suhelda ja küsida andmeid veeru pealkirja ja mitte järjekorranumbri järgi, siis sobib arvuti jaoks suupäraseks teisendamiseks `SqlDataReader`i käsklus `GetOrdinal`.

```
using System;
using System.Data.SqlClient;
class Baasiproov1a{
   public static void Main(string[] arg){
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=proovibaas; "+
       "Integrated Security=SSPI; Persist Security Info=False";
     string lause="SELECT eesnimi FROM inimesed";
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
     SqlCommand cm = new SqlCommand(lause, cn);
     SqlDataReader reader=cm.ExecuteReader();
     while(reader.Read()){

Console.WriteLine(reader.GetString(reader.GetOrdinal("eesnimi")));
     }
     cn.Close();
   }
}

/*
D:\kodu\0606\dotnet>Baasiproov1
Juku
Mati
Sass
*/
```

Sugugi alati ei pruugi kasutatavad andmed olla Microsofti SQL-serveris. Levinud veidi väiksemate andmetega ümber käimise programmiks on näiteks Access. Kui saab talle sobiva draiveriga sobiva versiooni faili külge minna, siis on täiesti lootust ka nõnda otse sidet pidada. Kusjuures nõnda programmi kaudu andmeid lugedes/kirjutades ei pea Accessi ennast üldse olema masinasse installeeritud. Piisab vaid sobivast draiverist, mis on üldjuhul juba operatsioonisüsteemiga kaasas.

```
using System;
using System.Data.Odbc;
class Baasiproov2a{
   public static void Main(string[] arg){
     string constr="Driver={Microsoft Access Driver (*.mdb)}; "+
       "DBQ=d:\\kodu\\0606\\dotnet\\proovibaas2.mdb; "+
       "Trusted_Connection=yes";
     string lause="SELECT mark FROM autod";
     OdbcConnection cn=new OdbcConnection(constr);
     cn.Open();
     OdbcCommand cm = new OdbcCommand(lause, cn);
     OdbcDataReader reader=cm.ExecuteReader();
```

```
      while(reader.Read()){
          Console.WriteLine(reader.GetString(0));
      }
      cn.Close();
   }
}
```

Kui aga tegemist pole Accessiga, vaid mõne muu andmebaasikeskkonnaga, mis aga sellegipoolest on Control Paneli kaudu ODBC alt kättesaadav, siis ka sealtkaudu saab oma andmetele ligi. Olgu näiteks olemas juba toimiv veebibaas PHP ja MySQLi abil – kuhu aga tahetakse ka kohalikus arvutis toimiva programmi kaudu pilku peale visata. Tehes see algne baas ODBC kaudu nähtavaks nime all näiteks proovibaas2, näeks sidepidamisprogramm välja nagu järgnevalt. Tasub tähele panna, et võrreldes SQL-serveriga on kasutatavateks objektiklassideks `OdbcConnection` ja `OdbcCommand`. Aga andmetega ümber käiakse ikka samamoodi.

```
using System;
using System.Data.Odbc;
class Baasiproov2{
   public static void Main(string[] arg){
      string constr="DSN=proovibaas2";
      string lause="SELECT mark FROM autod";
      OdbcConnection cn=new OdbcConnection(constr);
      cn.Open();
      OdbcCommand cm = new OdbcCommand(lause, cn);
      OdbcDataReader reader=cm.ExecuteReader();
      while(reader.Read()){
          Console.WriteLine(reader.GetString(0));
      }
      cn.Close();
   }
}
```

Kui baasist küsitakse vaid ühte väärtust ja mitte tervet tabelit, siis selle tarvis on .NET andmebaasi programmeerimise vahendite juurde loodud lihtne ja asjalik käsklus: `ExecuteScalar`. Küsitakse vaid inimeste arv, siis ühe käsuga saab selle kätte ilma, et peaks vahepeal andmepuhvrit looma.

```
using System;
using System.Data.SqlClient;
class Baasiproov1b{
   public static void Main(string[] arg){
      string constr="Data Source=RINDE\\SQLEXPRESS;"+
        "Initial Catalog=proovibaas; "+
        "Integrated Security=SSPI; Persist Security Info=False";
      string lause="SELECT COUNT(*) FROM inimesed";
      SqlConnection cn=new SqlConnection(constr);
      cn.Open();
      SqlCommand cm = new SqlCommand(lause, cn);
      Console.WriteLine("Inimeste arv: "+cm.ExecuteScalar());
      cn.Close();
```

```
   }
}

/*
D:\kodu\0606\dotnet>Baasiproov1b
Inimeste arv: 3
*/
```

Andmete lisamine

Eraldi moodus on käskluste jaoks, mis pole päringud: lisamine, muutmine, kustutamine. SQL-lause tuleb valmis teha nagu ikka, käivitamise jaoks aga käsklus ExecuteNonQuery.

```
using System;
using System.Data.SqlClient;
class Baasiproov1c{
   public static void Main(string[] arg){
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=proovibaas; "+
       "Integrated Security=SSPI; Persist Security Info=False";
     string lause="INSERT INTO inimesed (eesnimi) VALUES ('Siiri')";
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
     SqlCommand cm = new SqlCommand(lause, cn);
     cm.ExecuteNonQuery();
     Console.WriteLine("Andmed lisatud");
     cn.Close();
   }
}

/*
D:\kodu\0606\dotnet>Baasiproov1c
Andmed lisatud
*/
```

SQL-parameeter

Tahtes kasutaja andmeid SQL-lausesse lisada – olgu siis andmete küsimisel piirangu seadmiseks või andmete lisamisel salvestamiseks – on seda hea teha parameetri kaudu. Sellisel juhul ei pea SQL-lauset koostades muret tundma, et kasutaja sisestus kuidagi jutumärke või ülakomasid omavahel sõlme võiks ajada. Samuti püsib nõnda ka lause ülesehitus selgemana – lihtsalt @-märgiga tähistatud sõna asendatakse pärast sobiva väärtusega. Siin tähistatakse `@algaasta`ga arv, millisest aastast alates sündinud laste nimesid tahetakse näha.

```
     string lause="SELECT eesnimi, synniaasta FROM lapsed "+
        "WHERE synniaasta >= @algaasta";
```

Enne käskluse käivitamist tuleb siis parameetri kohta sobiv väärtus paigutada. Siin näites on kaks toimingut ühel real. Kõigepealt lisatakse käsklusele cm parameeter nimega @algaasta.

Seejärel omistatakse loodud objekti omadusele Value väärtus 1997. Viimatine aastaarv on praegu küll kirjutatud lühiduse mõttes arvuna koodi, aga sinna saab paigutada kasutajalt tulnud väärtuse.

```
     cm.Parameters.Add("@algaasta", SqlDbType.Int).Value=1997;
```

Vastuseks saame andmete loetelu nii nagu ikka.

```
using System;
using System.Data;
using System.Data.SqlClient;
class Baasiparameeter1{
   public static void Main(string[] arg){
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=baas1; "+
       "Integrated Security=SSPI; Persist Security Info=False";
     string lause="SELECT eesnimi, synniaasta FROM lapsed "+
        "WHERE synniaasta >= @algaasta";
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
     SqlCommand cm = new SqlCommand(lause, cn);
     cm.CommandType=CommandType.Text;
     cm.Parameters.Add("@algaasta", SqlDbType.Int).Value=1997;
     SqlDataReader reader=cm.ExecuteReader();
     while(reader.Read()){
         Console.WriteLine(reader.GetString(0)+": "+
             reader.GetInt32(1));
     }
     cn.Close();
   }
}

/*
D:\kodu\0606\dotnet>Baasiparameeter1
Juku: 1997
Kati: 1997
Siim: 1997
*/
```

Salvestatud protseduur

Tahtes programmi kaudu salvestatud protseduurile andmeid jagada on kasulik jällegi parameetrit pruukida. Parameetrile antakse nime järgi väärtus ning protseduur saabki selle kätte. Siinne protseduur loodi käsuga

```
CREATE PROCEDURE kysiLapsed(@algaasta decimal)
AS
SELECT eesnimi, synniaasta FROM lapsed
WHERE synniaasta>=@algaasta
```

Andmete lugemine tabeli väljastavast salvestatud protseduurist on sarnane hariliku päringu vastuste lugemisele. Ning tulemus on samuti eelmise näitega sarnane.

```
using System;
using System.Data;
using System.Data.SqlClient;
class Baasiparameeter1{
   public static void Main(string[] arg){
     string constr="Data Source=RINDE\\SQLEXPRESS;"+
       "Initial Catalog=baas1; "+
       "Integrated Security=SSPI; Persist Security Info=False";
     string lause="kysiLapsed";
     SqlConnection cn=new SqlConnection(constr);
     cn.Open();
     SqlCommand cm = new SqlCommand(lause, cn);
     cm.CommandType=CommandType.StoredProcedure;
     cm.Parameters.Add("@algaasta", SqlDbType.Int).Value=1997;
     SqlDataReader reader=cm.ExecuteReader();
     while(reader.Read()){
         Console.WriteLine(reader.GetString(0)+": "+
             reader.GetInt32(1));
     }
     cn.Close();
   }
}
/*
D:\kodu\0606\dotnet>Baasiparameeter2
Juku: 1997
Kati: 1997
Siim: 1997
*/
```

Ülesandeid

* Loo andmebaasitabelid: maakonnad (id, maakonnanimi), autod (id, mark, aasta, maakonna_id). Lisa mõned andmed.

* Küsi programmi abil ekraanile kõikide maakondade nimed

* Küsi ekraanile kõikide autode andmed koos maakondade nimedega, kus nad registreeritud.

* Loo käsklus uue auto lisamiseks. Sisendiks mark, aasta, maakonna_id.

* Olematu id-ga maakonna puhul näidatakse olemasolevaid maakondade nimesid ja id-sid ning palutakse viimane uuesti sisestada.

* Loo võimalus etteantud id-ga auto maakonna muutmiseks.

* Loo salvestatud protseduur näitamaks enne parameetrina antud aastat tehtud autosid. Käivita protseduur ja vaata tulemusi C# kaudu.

* Loo klassid auto ning maakonna andmete hoidmiseks. Katseta paari väärtusega.

* Loo klass AutoRegister toimetamaks andmebaasis paiknevate autodega. Järjekorranumbri abil peetakse meeles jooksvat autot, samuti on klassi eksemplaril meeles baasis leiduvate autode id-d (ArrayListina). Lisa klassile käsud jooksva auto küsimiseks, andmete muutmiseks, id järgi järgmise/eelmise auto juurde liikumiseks, auto lisamiseks lõppu. Andmete küsimisel väljastatakse auto klassi eksemplar, mis viitab maakonnaeksemplarile.

* Ehita AutoRegistri klassi ümber kasutajaliides, mille kaudu on võimalik käsurealt autode andmetega toimetada: loetelu vaadata, aktiivset autot üles/alla määrata, aktiivse auto andmeid muuta, autosid lisada/kustutada. Autosid soovitud tunnuse alusel sorteerida/filtreerida.

Funktsiooni delegaadid

Kui on programmi käitumise juures vaja valida mitme võimaluse vahel, siis tavalisimaks vahendiks on `if`/`else` plokk, millega tööjärge suunata.

Objektorienteeritud programmeerimise juures kui soovitakse objektid sarnastes kohtades erinevalt käituma panna, siis luuakse algsele klassile iga märgatavalt erineva käitumismooduse jaoks alamklass ning selles kaetakse vastav meetod üle nõnda nagu see parasjagu vajalik on. Hiljem saab soovitud käitumise väljakutsumise jaoks anda sobivasse kohta ette õige alamklassi sobivalt seadistatud eksemplari. Nõnda tuleb näiteks `ArrayList`i omapoolselt määratud järjestusse sorteerides ette anda liidest `IComparer` realiseeriva klassi eksemplar, mille juures tuleb üle katta funktsioon nimega `Compare`. Nimetatud meetodi ülesandeks on kahe etteantud objekti põhjal otsustada, kumb neist järjestuses ettepoole saab. Sedasi võib tekstid järjestada näiteks pikkuse ja mitte tähestiku järgi.

Kes on esimese keelena kirjutanud Cs või C++is, sel tekib nõnda alamklasse luues kohe küsimus, et miks nii keeruliselt peab tegema. Et eelnimetatud keeltes kasutatakse selliseks sortimisjärjekorra määramiseks funktsiooniviitu. Kuna C# püüab eneses ühendada rangemate keelte struktueeritust ning masinalähedasemate keelte võimalusi, siis on siia loodud funktsiooni delegaadid. Lahtiseletatult on tegemist eripärase muutujaga, mille kaudu saab selle külge omistatud funktsiooni välja kutsuda. Seletus järgmise näite põhjal.

Tervituseks on loodud kaks funktsiooni: `RahulikTervitus` ning `TragiTervitus`. Programmi mingis osas saab valida, millist funktsiooni neist kasutatakse. Et funktsiooni delegaadina meelde jätta, tuleb see programmi algul deklareerida.

```
public delegate void Tervitusfunktsioon();
```

Edasi võib sobival ajal omistada loodud delegaatmuutujale tegeliku funktsiooni eksemplari.

```
Tervitusfunktsioon tervitaja=new Tervitusfunktsioon(RahulikTervitus);
```

Delegaatmuutujale sulgude taha kirjutamisega pannaksegi soovitud funktsioon tööle.

```
tervitaja();
```

Ja tulemusena näeme sel korral lihtsat "Tere".

```
using System;
public delegate void Tervitusfunktsioon();
class Delegaat1{
```

```
   static void RahulikTervitus(){
      Console.WriteLine("Tere");
   }
   static void TragiTervitus(){
      Console.WriteLine("Ahoi!");
   }
   public static void Main(string[] arg){
      Tervitusfunktsioon tervitaja=new Tervitusfunktsioon(RahulikTervitus);
      tervitaja();
   }
}
/*
D:\kodu\0606\dotnet>Delegaat1
Tere
*/
```

Funktsioonide komplekt

Mõnikord tuleb talletada terve tegevuste jada. Et delegaatmuutujad käituvad küllalt tavaliste objektide sarnaselt, saab neid ka kogumis (siin `ArrayList`) talletada ja pärast välja kutsuda. Sedasi on võimalik ühes koodiosas teine käsujada kokku panna ja seda hiljem pruukida.

```
using System;
using System.Collections;
public delegate void Tervitusfunktsioon();
class Delegaat2{
   static ArrayList tervitused=new ArrayList();
   static void RahulikTervitus(){
      Console.WriteLine("Tere");
   }
   static void TragiTervitus(){
      Console.WriteLine("Ahoi!");
   }
   public static void Main(string[] arg){
      tervitused.Add(new Tervitusfunktsioon(RahulikTervitus));
      tervitused.Add(new Tervitusfunktsioon(TragiTervitus));
      tervitused.Add(new Tervitusfunktsioon(RahulikTervitus));
      foreach(Tervitusfunktsioon tervitaja in tervitused){
         tervitaja();
      }
   }
}

/*
D:\kodu\0606\dotnet>Delegaat2
Tere
Ahoi!
Tere
*/
```

Sündmused

Sarnane käskude jada salvestamine on mõnes olukorras nõnda tavaline, et selle jaoks on eraldi välja mõeldud tüüp event. Nii nagu võis `Add` käsuga lisada andmeid `ArrayList`i, nii saab += operaatoriga panna juurde funktsioonidelegaate sündmuste loendisse. Ja tulemus sarnane nagu

eelmisel korral. Ette rutates märkides kasutatakse sellist delegaatide sündmusteahelasse lisamist näiteks graafilistes programmides nupuvajutuse korral, kus iga nupu alla võib panna käivituma mitu funktsiooni. Siin aga lihtsalt taas tervitused.

```
using System;
public delegate void Sisenemine();
class Syndmused1{
   static event Sisenemine InimeneSisenes;
   static void RahulikTervitus(){
      Console.WriteLine("Tere");
   }
   static void TragiTervitus(){
      Console.WriteLine("Ahoi!");
   }
   public static void Main(string[] arg){
      InimeneSisenes+=new Sisenemine(RahulikTervitus);
      InimeneSisenes+=new Sisenemine(TragiTervitus);
      InimeneSisenes+=new Sisenemine(RahulikTervitus);
      InimeneSisenes();
   }
}

/*
D:\kodu\0606\dotnet>Syndmused1
Tere
Ahoi!
Tere
*/
```

Ilmajaamad

Järgnevalt eelneva põhjal kokku pandud näide, kus piirkonda hajusalt paigutatud 10 vaatlusjaama omi mõõteandmeid keskusesse saadavad ning kuidas saabunud andmetele reageeritakse.

Peaprogramm loob kõigepealt ilmajaamade objektid, siis määrab nende juures, milline funktsioon panna käima tavateate juures ning mida teha juhul, kui tegemist on mõne hoiatusega. += operaator lubab vajadusel panna vastava sündmuse külge ka mitu teadet või jätta sootuks ilma teateta.

Edasi palutakse kõigil jaamadel välja mõelda (mõõta) andmed ning keskusesse saata teated nende kohta. Ja siis küsitakse kasutajalt, kas ta soovib veel uuesti andmeid küsida. Et teabe trükkimise funktsioonid olid juba ilmajaamade teatesündmuste külge lisatud, siis lähevad nad sündmuste käivitamisel ise käima.

Jaamal on meeles enese number. Teate saatmisel küsitakse andurilt temperatuur ning pannakse kokku keskusesse minev vajalik andmeplokk. Temperatuuri küsimisel arvestab programm kuu järjekorranumbrit ning selle suhtes arvutab välja sellele aastaajale tõenäolise temperatuuri - lihtsalt üks viis kuigivõrd usutavaid vastuseid simuleerida.

```
   int kysiTemperatuur(){
```

```
      return 15-Math.Abs(6-System.DateTime.Now.Month)*5+
                                        arvugeneraator.Next(20);
   }
```

Et sündmuse käivitamisel saaks sinna andmeid kaasa saata, selleks peavad andmed olema koos ühes terviklikus objektis ning selle objekti klass peab olema klassi `EventArgs` alamklass - mis siis lubab neil andmetel teatega kaasa minna.

```
   public class IlmajaamaParameetrid: EventArgs{
     ...
   }
```

Kui jaam teateid saadab, siis kõigepealt pannakse kokku eelnäidatud tüübist andmeplokk. Edasi käivitatakse igal juhul tavateate saatmine (sündmus) ning kui põhjust, siis selle järel ka hoiatusteate saatmine. Mis teatesaatmissündmuse puhul tegelikult tehakse, on eespool ilmajaama loomise juures juba kirjeldatud.

```
   public void saadaTeated(){
      IlmajaamaParameetrid andmed=new IlmajaamaParameetrid(
                              jaamaNr, ++teateNr, kysiTemperatuur());
      tavaTeade(andmed);
      if(andmed.temperatuur<0 ||
                          andmed.temperatuur>20){hoiatusTeade(andmed);}
   }
```

Ja ongi kogu näide, mida võib edaspidi sarnaste rakenduste loomisel aluseks võtta.

```
using System;
namespace Ilmaandmed{
  public delegate void IlmajaamaTeade(IlmajaamaParameetrid p);
  class Syndmused2{
   static void HarilikTeave(IlmajaamaParameetrid p){
      Console.WriteLine(p);
   }
   static void HoiatusTeave(IlmajaamaParameetrid p){
      Console.WriteLine("     Hoiatus: "+p);
   }
   public static void Main(string[] arg){
      Ilmajaam[] jaamad=new Ilmajaam[10];
      for(int i=0; i<jaamad.Length; i++){
         jaamad[i]=new Ilmajaam();
         jaamad[i].tavaTeade+=new IlmajaamaTeade(HarilikTeave);
         jaamad[i].hoiatusTeade+=new IlmajaamaTeade(HoiatusTeave);
      }

      string vastus;
      do{
       for(int i=0; i<jaamad.Length; i++){
        jaamad[i].saadaTeated();
       }
       Console.WriteLine("Kas veel?");
       vastus=Console.ReadLine();
      } while(vastus.ToLower().StartsWith("j"));
```

```
   }
  }

  class Ilmajaam{
   static int jaamadeArv=0;
   int jaamaNr;
   int teateNr=0;
   public event IlmajaamaTeade tavaTeade;
   public event IlmajaamaTeade hoiatusTeade;
   private Random arvugeneraator;
   public Ilmajaam(){
      jaamaNr=++jaamadeArv;
      arvugeneraator=new Random(jaamaNr);
   }
   int kysiTemperatuur(){
      return 15-Math.Abs(6-System.DateTime.Now.Month)*5+
                                        arvugeneraator.Next(20);
   }
   public void saadaTeated(){
      IlmajaamaParameetrid andmed=new IlmajaamaParameetrid(
                               jaamaNr, ++teateNr, kysiTemperatuur());
      tavaTeade(andmed);
      if(andmed.temperatuur<0 || andmed.temperatuur>20){
                                    hoiatusTeade(andmed);}
   }
  }
  public class IlmajaamaParameetrid: EventArgs{
     public readonly int temperatuur;
     public readonly int jaamaNr;
     public readonly int teateNr;
     public IlmajaamaParameetrid(int uusJaamaNr, int uusTeateNr,
                                                  int uusTemperatuur){
        temperatuur=uusTemperatuur;
        jaamaNr=uusJaamaNr;
        teateNr=uusTeateNr;
     }
     public override string ToString(){
        return "Jaam "+jaamaNr+", teade "+teateNr+" temperatuur "+
                                                     temperatuur;
     }
  }
}
/*
D:\kodu\0606\dotnet>Syndmused2
Jaam 1, teade 1 temperatuur 14
Jaam 2, teade 1 temperatuur 25
     Hoiatus: Jaam 2, teade 1 temperatuur 25
Jaam 3, teade 1 temperatuur 15
Jaam 4, teade 1 temperatuur 26
     Hoiatus: Jaam 4, teade 1 temperatuur 26
Jaam 5, teade 1 temperatuur 16
Jaam 6, teade 1 temperatuur 27
     Hoiatus: Jaam 6, teade 1 temperatuur 27
Jaam 7, teade 1 temperatuur 17
Jaam 8, teade 1 temperatuur 28
     Hoiatus: Jaam 8, teade 1 temperatuur 28
Jaam 9, teade 1 temperatuur 18
Jaam 10, teade 1 temperatuur 29
     Hoiatus: Jaam 10, teade 1 temperatuur 29
Kas veel?
j
```

```
Jaam 1, teade 2 temperatuur 12
Jaam 2, teade 2 temperatuur 18
Jaam 3, teade 2 temperatuur 23
     Hoiatus: Jaam 3, teade 2 temperatuur 23
Jaam 4, teade 2 temperatuur 29
     Hoiatus: Jaam 4, teade 2 temperatuur 29
Jaam 5, teade 2 temperatuur 15
Jaam 6, teade 2 temperatuur 21
     Hoiatus: Jaam 6, teade 2 temperatuur 21
Jaam 7, teade 2 temperatuur 27
     Hoiatus: Jaam 7, teade 2 temperatuur 27
Jaam 8, teade 2 temperatuur 13
Jaam 9, teade 2 temperatuur 19
Jaam 10, teade 2 temperatuur 25
     Hoiatus: Jaam 10, teade 2 temperatuur 25
Kas veel?
e
*/
```

Graafiline liides

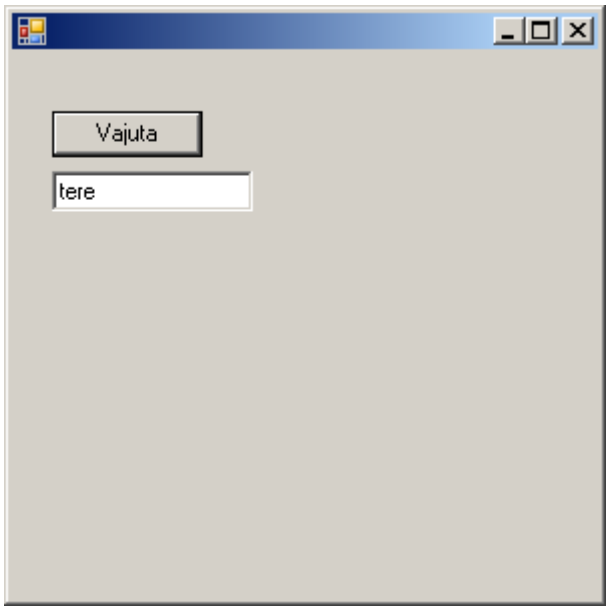
Lõppu veel näide, kuidas sündmusi graafilise keskkonna juures pruugitakse.

Windowsi aknarakenduse puhul on lihtsaks võimaluseks luua oma rakendus klassi Form alamklassina. Graafilised elemendid saab klassi algul ära kirjeldada ning konstruktoris ekraanile paigutada. Ekraanikoordinaatide järgi paigutus akna ülanurga suhtes on üks võimalusi. Kõigepealt määratakse graafikakomponendid asukohapunkti järgi paika ning `Controls.Add` käsu abil jõuavad nad sinna nähtavana ekraanile. Tahtes nupuvajutussündmuse peale miskit käivitada, saab eelpoolvaadatud += operaatori abil lisada omaloodud funktsiooni `Click`-nimelisele sündmusele.

`nupp1_Click` funktsioonile tulevad kaks parameetrit. Esimese kaudu saab eristada millist nuppu vajutati - juhul kui on oht, et sündmusi võib mitmelt poolt tulla. ning teisest võiks teavet saada näiteks hiire andmete kohta vajutushetkel. Aga kuna praegu meil on tegemist ainukese nupuga ning soovime käima panna lihtsa teretuse, siis pole meil neid parameetreid endid pruukida vaja.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class Tervitaja2:Form{
  Button nupp1=new Button();
  TextBox tekst1=new TextBox();
  public Tervitaja2(){
     nupp1.Location=new Point(20, 30);
     tekst1.Location=new Point(20, 60);
     nupp1.Text="Vajuta";
     Controls.Add(nupp1);
     Controls.Add(tekst1);
     nupp1.Click+=nupp1_Click;
  }
  void nupp1_Click(object saatja, EventArgs e){
     tekst1.Text="tere";
```

```
  }
  public static void Main(string[] arg){
     Application.Run(new Tervitaja2());
  }
}
```



Ülesandeid

* Uuri näidet `Delegaat1`, lisa sinna funktsioon `UinuvTervitus`, mis trükiks "brrr". Katseta.

* Muuda näidet `Syndmused1` nõnda, et lisanduks sündmus `InimeneUinus`, kuhu oleks tsükli abil 10 korda lisatud `UinuvTervitus`. Katseta.

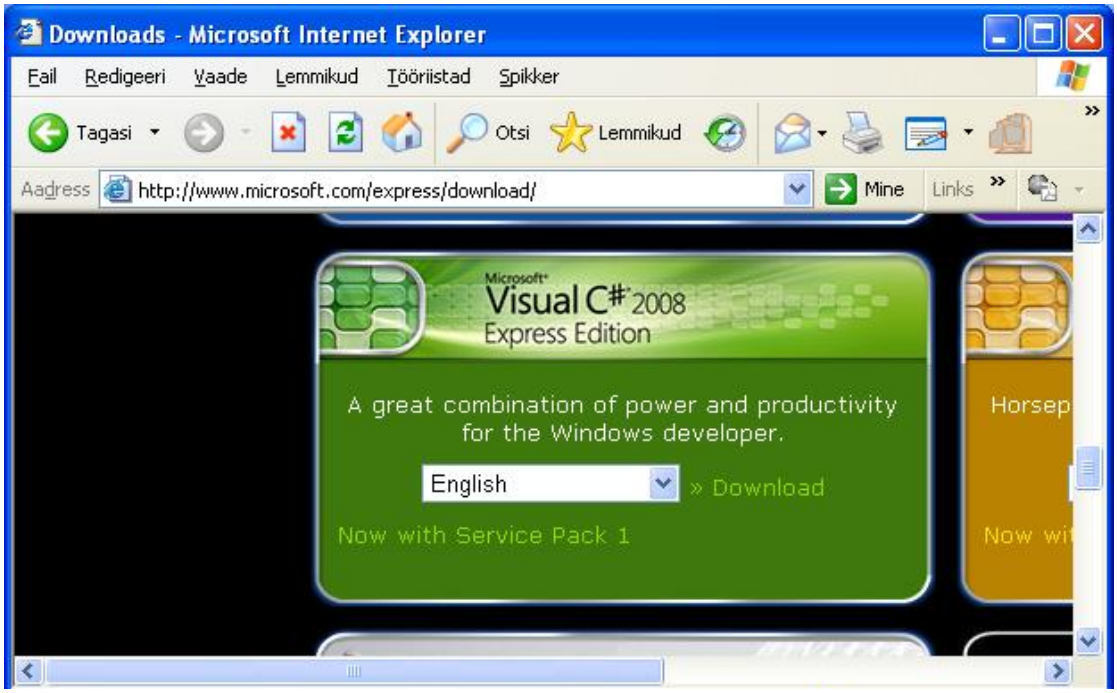
* Lisa ilmajaama näitele tuule kiirus.

* Koosta nupuvajutusnäite abil graafiline kalkulaator, mille abil saab sentimeetreid tollideks ja tagasi arvutada.

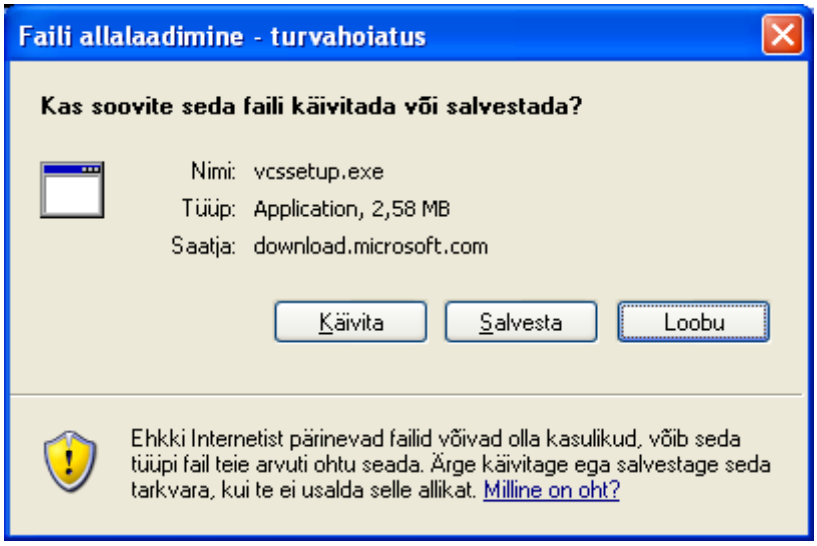
Visual Studio C# Expressi install

Programmikoodi kirjutamise ja käivitamise jaoks on loodud mitmesuguseid abivahendeid. Microsofti ametlikuks graafiliseks arenduskeskkonnaks on Visual Studio. Express-versioonid sellest on kõigile vabalt kasutatavad. Tutvumiseks ja allalaadimiseks sobib veebiaadress www.microsoft.com/express/

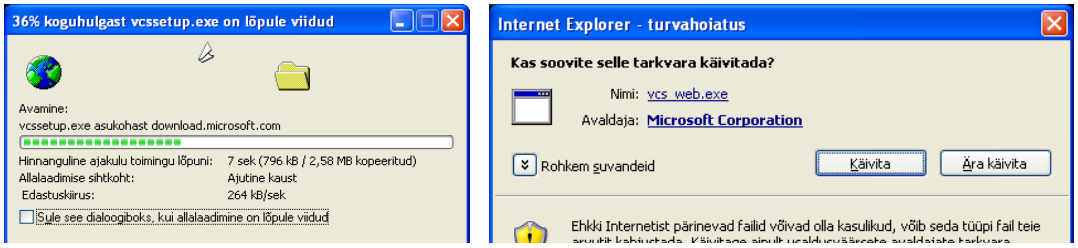
Sealt pääseb edasi allalaadimise juurde. Pakutakse kõikide Express-versioonide korraga alla laadimist või üksikute toodete installeerimist veebi kaudu. Ühe toote ühekordseks paigaldamiseks piisab veebikaudsest installeerimisest. Valime siit sobiva programmi.



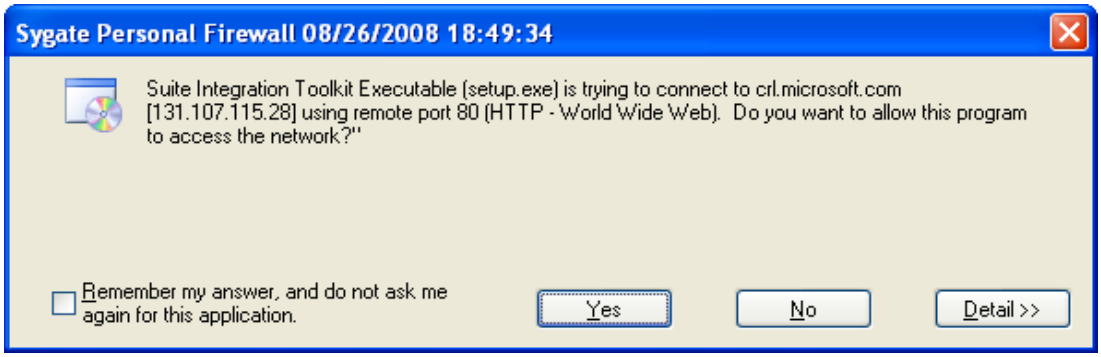
Kui ainult ühte masinasse panna, siis võib programmi kohe käivitada. Algul laetakse kohale vaid lühike, kahemegabaidine lõik kontrollimaks, mis masinas olemas ning mida vaja sinna juurde laadida.



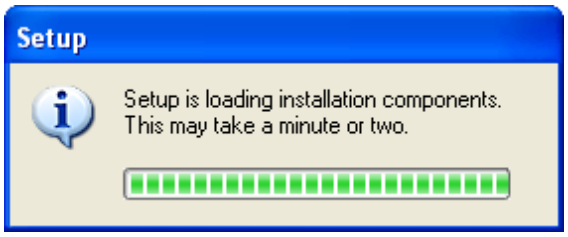
Pärast laadimist võib programmi käivitada.



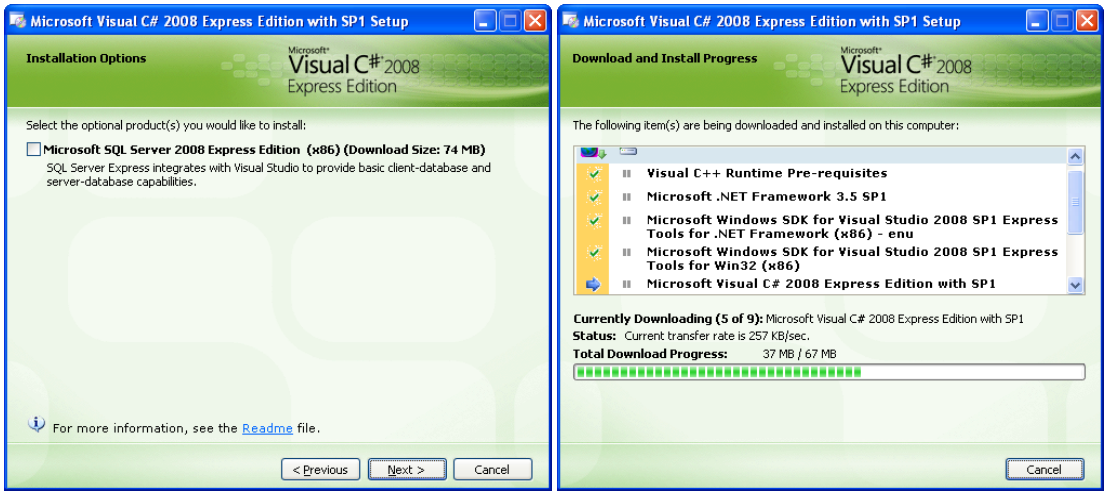
Sõltuvalt masina seadetest võidakse küsida, et kas tohib rakendus ise luua ühenduse välisvõrguga. Paigalduse jätkamiseks pole muud teha kui lubada.



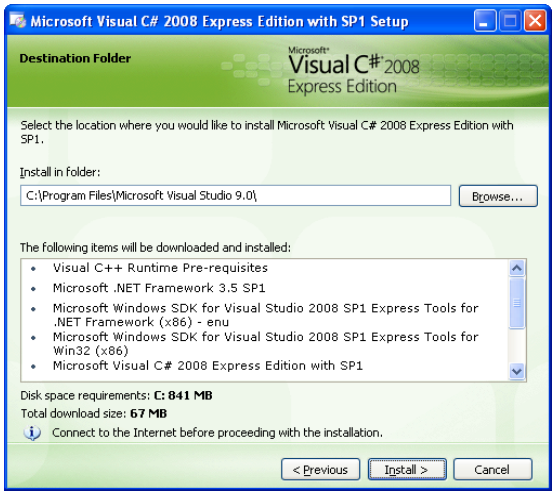
Edasi programmi käivitus ning masina ülevaatus

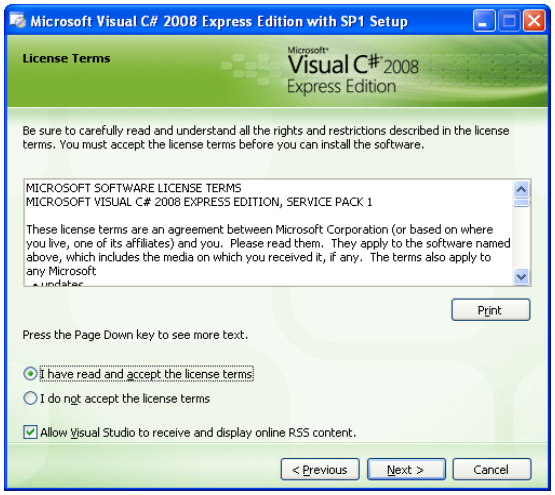


ning asutakse võrgust vajalikke lisatükke tõmbama. Praegusel juhul alla saja megabaidi. Aga kui lisavõimalusi rohkem valitud, siis võib ka see kogus mitmekordistuda ning aega rohkem minna.



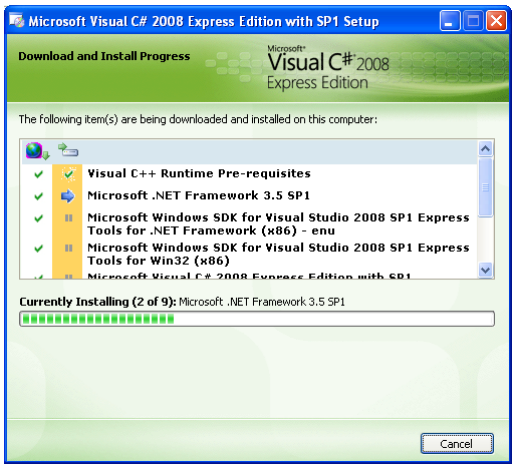
Küsitakse asukoht programmifailide jaoks. Kui põhikettal piisavalt ruumi, siis võib vaikimisi asukoht „Program files´i alla jääda. Kui aga näiteks C-ketas täis ja mõnes muus kohas ruumi, siis on lootust osa asju sinna peita - ehkki vahel kipub tekkima olukord, kus siis kõike kohe üles ei leita. Ja litsentsilepinguga nõustumisest ei pääse me ka seekordse installatsiooni käigus.



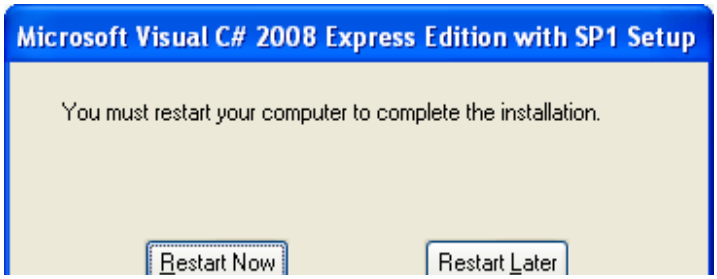


Kui kõik alla laetud, siis järgmine samm tegeliku installeerimise juurde. Paarkümmend minutit võib kõikide komponentide peale panekuks julgesti aega minna, vahel ka rohkem. Õnneks progressiribalt on näha, kaugel tegelikult ollakse.

Ja kui rakendus peal, siis palutakse see registreerida. Muidu hakatakse kuu aja pärast kasutajat kiusama, et see ikka märku annaks, kes ta selline on, kes siinset tasuta tarkvara kasutab.

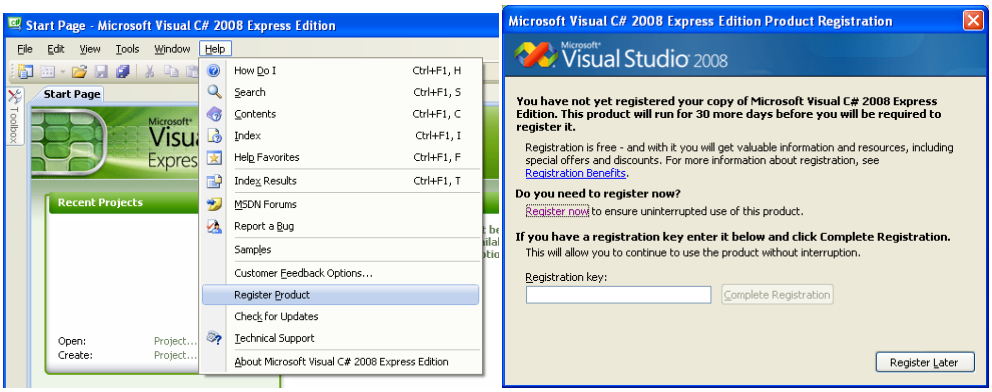


Suurema installeerimise juures ei pääse sageli ka restardist. Nii ka siin.

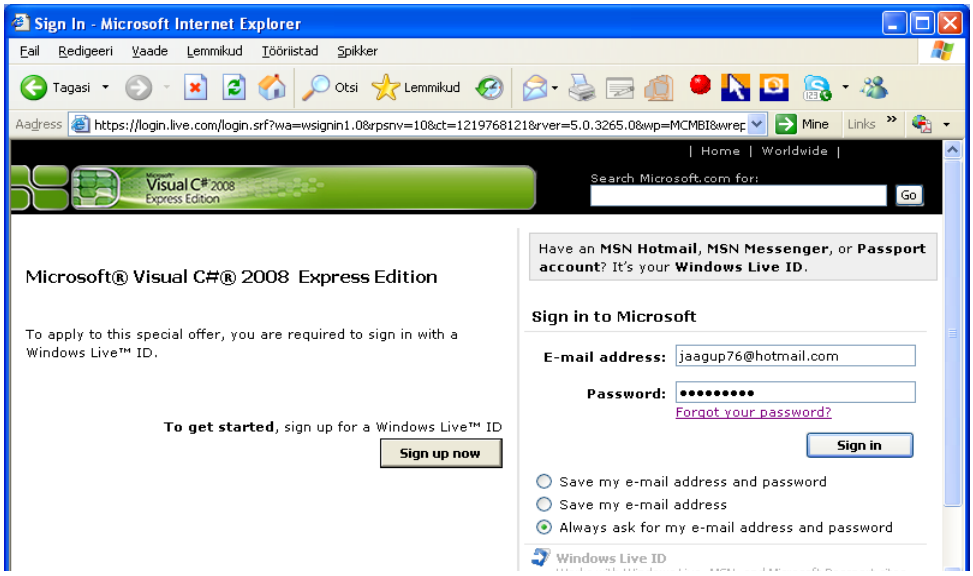


Esmasel käivitamisel kulub tavapärasest mõnevõrra pikem aeg, aga seda ka hoiatatakse.

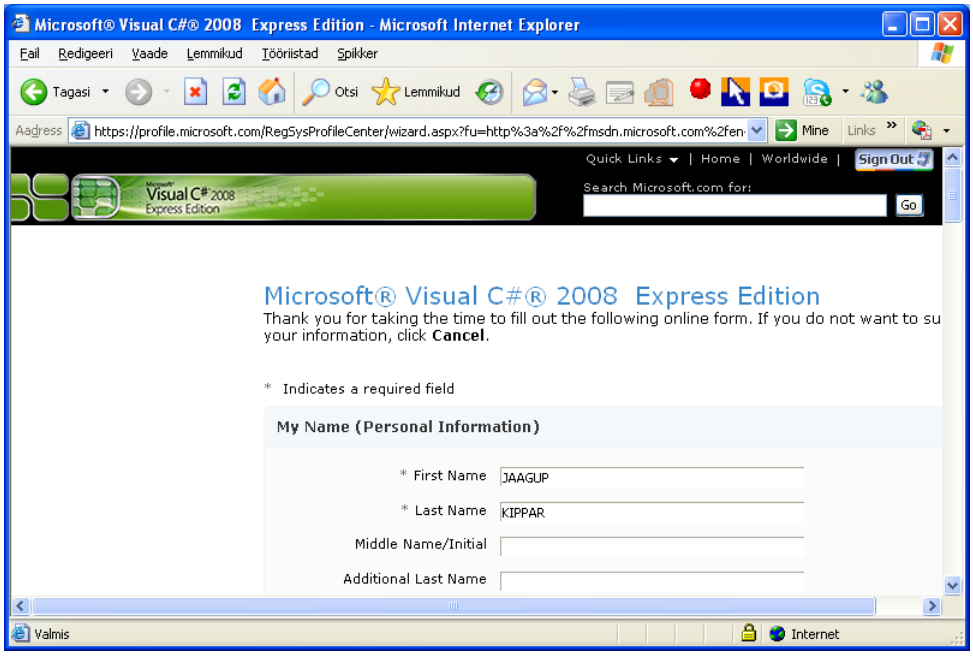
Et hiljem poleks registreerimisega muret, võib selle kohe ära teha. Help-menüüst tuleb alustuseks vastav rida valida. Saabuvasse aknasse palutakse siis registreerimisvõtit ning pakutakse viide selle saamiseks.



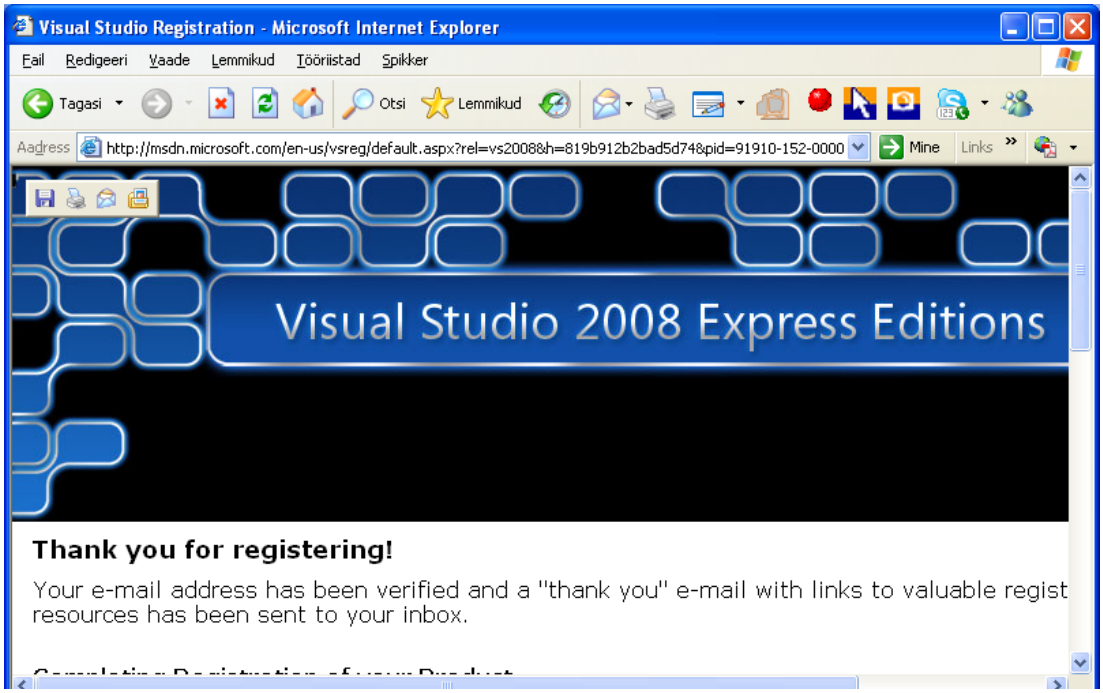
Kel Live ID ehk siis MSNi kasutajatunnus olemas, sel käib asi lihtsalt – suunatakse sobivale veebilehele. Kel mitte, peab registreerumiseks omale vastava tunnuse tegema.



Kontrollitakse üle, et ega mu eelistused vahepeal muutunud pole



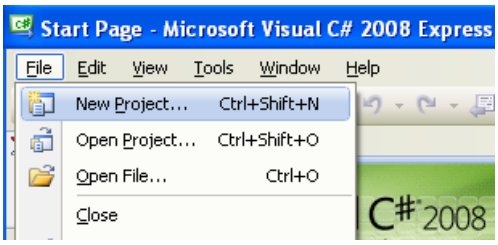
ning antaksegi pikk täherida vastuseks.



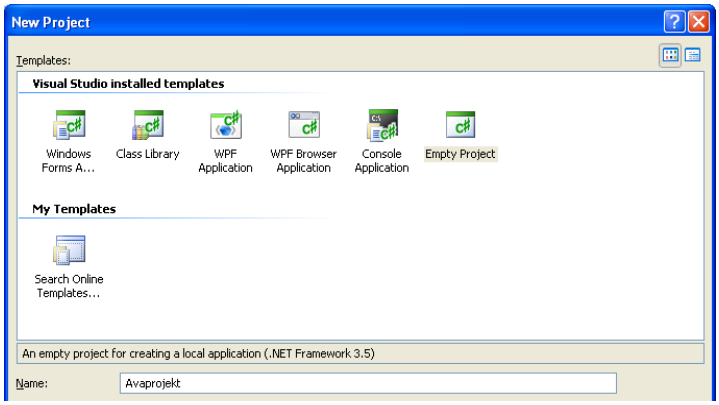
Viimane tuleb tollesse help-menüüst avanenud aknasse kopeerida ning võibki registreerumise lõppenuks lugeda.

Esimese rakenduse loomine

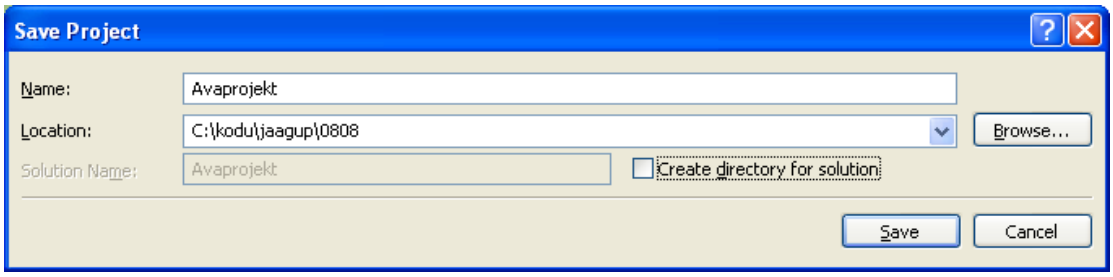
Keskkonna kaudu kompileerimise eeltingimuseks on projekti olemasolu. Eks ikka failimenüüst uus.



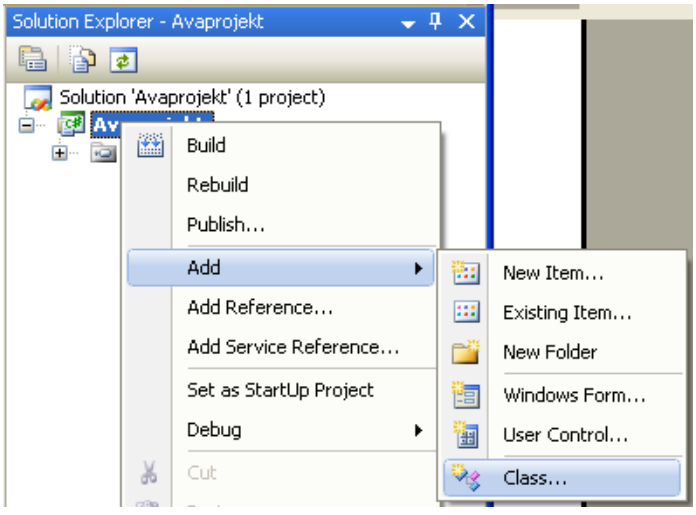
Saab küll valida sobivate valmisprojektide hulgast ning rakenduse kohe tööle panna. Aga kui tahtmist ise klasse ja nimeruume nimetada, siis tühja projektiga lihtsam.



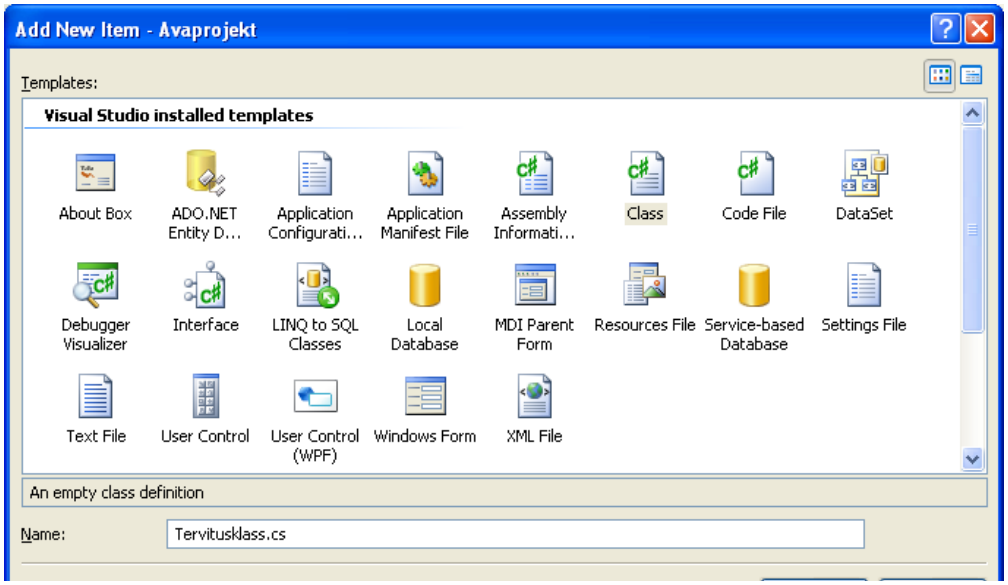
Nimeks näiteks Avaprojekt. Ning et pärast ei peaks faile mööda masinat taga otsima, selleks sobib projekt kohe ka omale sobivasse kausta salvestada. Ning kui liigselt alamkatalooge ei taha, siis tasub linnuke „Create directory for solution“ tühjaks jätta – üks kaust tehakse nagunii.



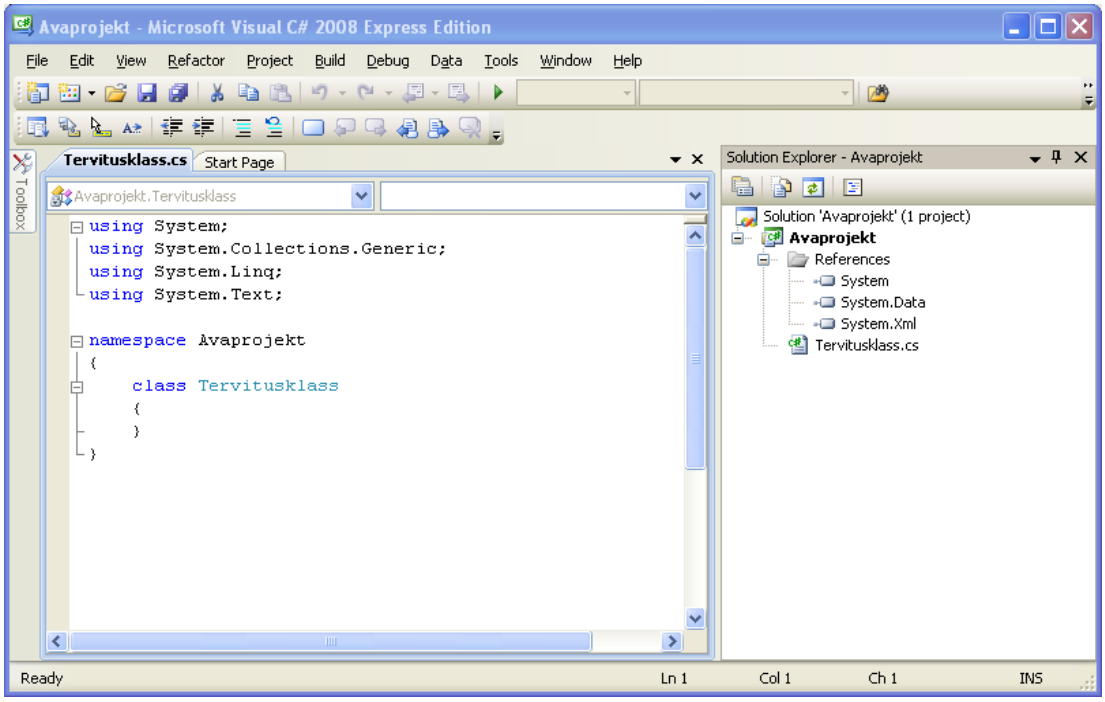
Projekti sisse siis uus klass, et oleks kuhugile kood kirjutada.



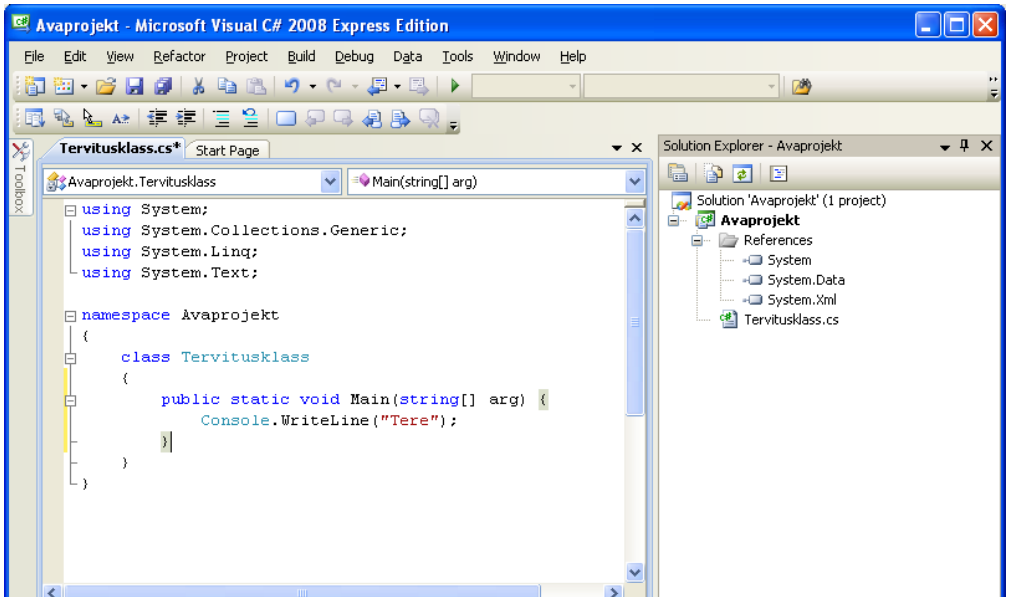
Klassile oma jaoks arusaadav nimi



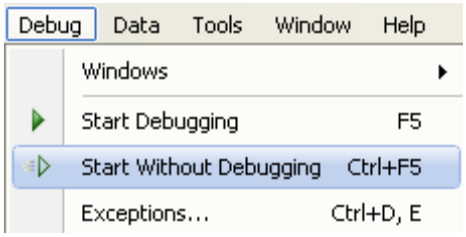
ning ongi koht kus koodi kirjutada.



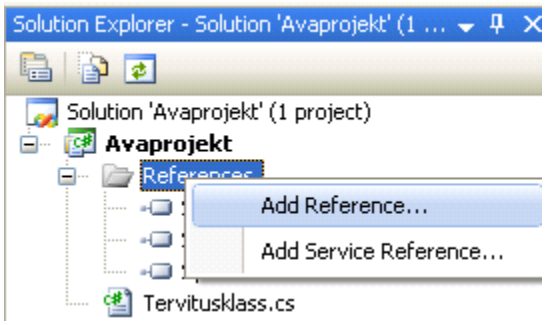
Main-meetod sisse, et kood mõistaks kusagilt tööd alustada



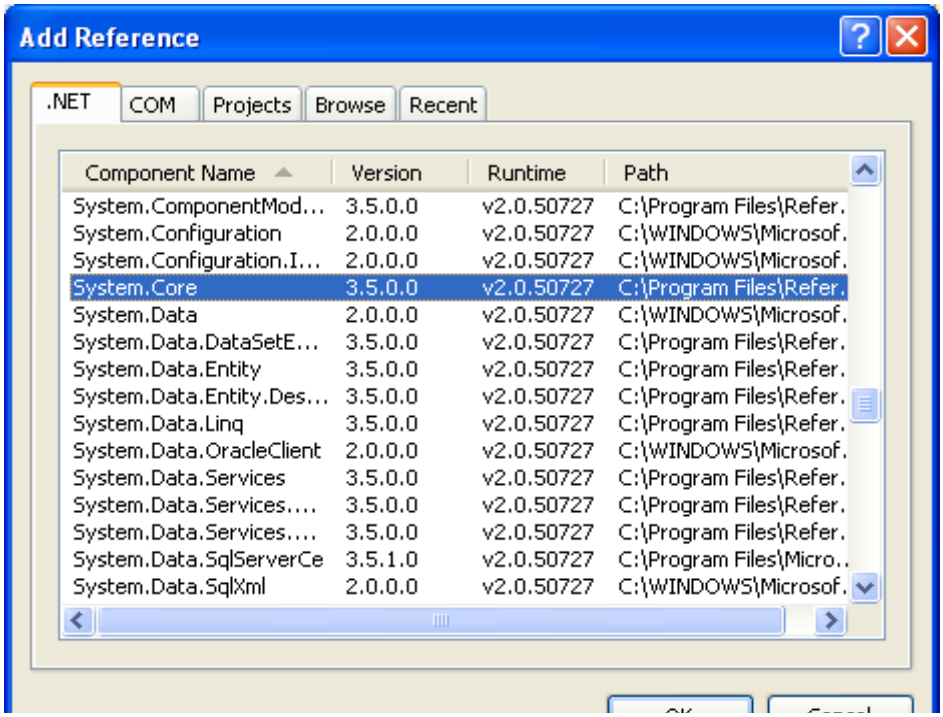
Ja juba võib proovida käivitada. Debug->Start Without Debugging peaks rakenduse lihtsalt käima panema ning pärast ka akna ette jätma, et võimalust töö tulemusi imetleda oleks.



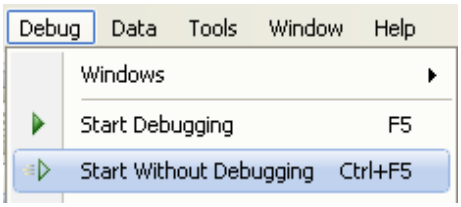
using System. LINQ rea peale aga kiputakse jorisema – et miski teek puudu. Praeguses näites võiks küll vastava rea rahus välja võtta, sest ega tegelikult siin LINQ-päringutega midagi ei tehta. Aga kui tuleviku peale mõelda ning arvata, et sellest laiendusest võiks kasu olla, siis tasub vastav teek projektile külge haakida. Viidete (References) juurest parema klahviga Add Reference



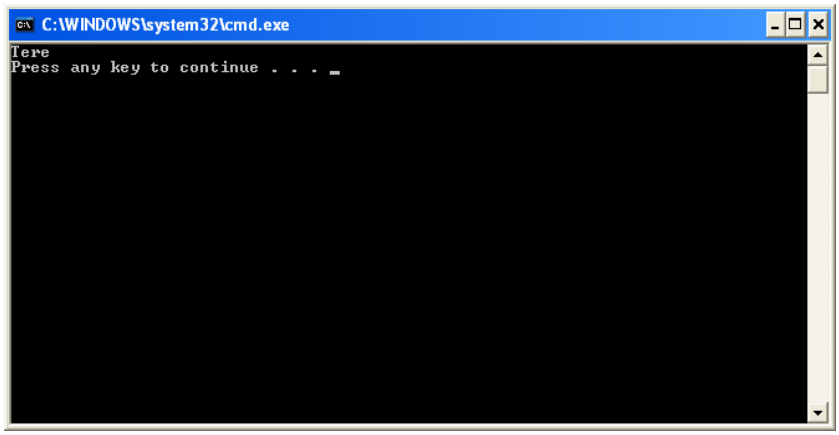
ning vastava abitüki leiab teegist System.Core.



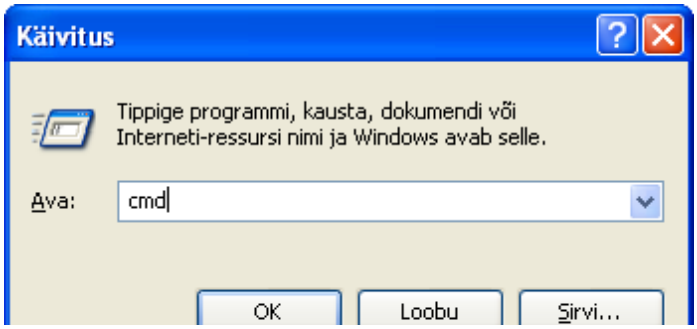
Edasi võiks käivitamine juba lihtsamalt minna.



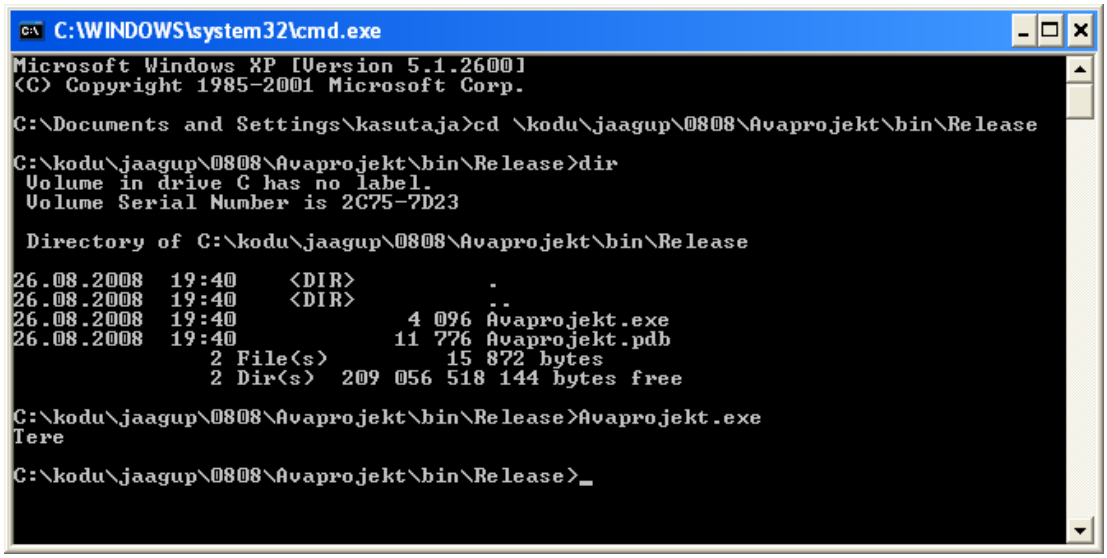
Ehk siis „Tere“ tuligi nähtavale.



Sellisest lähenemisest enamasti piisabki. Aga kui tahta end mõnikord mitte keskkonna trikkidest häirida lasta ning vaadata, kuidas programm „tavalise“ kasutaja juures käituda võiks, siis saab sedagi. Kõigepealt lahti käsureaaken



Edasi CD-ga sobivasse kataloogi ning võib tekkinud exe-faili ka sealt käima panna. Tulemuseks ikka sama „Tere“.



Kokkuvõte

Eelpoolsetel lehekülgedel alustati C# pisikese tervitava programmiga ning käidi läbi enamik tähtsamaid keele süntaksi ja ülesehituse juurde kuuluvaid teemasid. Hulk nüansse ja erijuhte jäi käsitlemata, kuid olemasolevate põhjal peaks õnnestuma enamik enesele tarvilikke programme kokku panna ja ette juhtunud valmisprogrammidest aru saada. Samuti võiks pärast siinse kirjutise läbitöötamist tekkida piisav karkass ja kogemus, mille külge on kergem mujalt leitud C# ja objektorienteeritusega seotud lõike haakida.

Siinsed programmid käivitati käsurealt. Kuid nagu lõpunäitest näha, võib ka käsurearakendustel olla täiesti graafiline liides. Kui edasi uurida System.Windows.Forms nimeruumi käske ja näiteid, siis sealtkaudu võib oma rakendusele üha uusi graafilisi vidinaid juurde lisada. Kui veel käivitada mitte mustast käsureaaknast vaid seada töölauale sobiv ikoon, siis ongi "harilikule" kasutajale tavaline rakendus valmis.

C# loodi veebiajastul. Selle tõttu on tal kasutada mitmesuguseid mooduseid võrgu kaudu suhtlemiseks. Nii otseühenduses üle TCP-protokolli näiteks jututoa või võrgumängu tarbeks kui ka kogu tehnikate komplekt veebirakenduste loomiseks. Sealne käivitus näeb tunduvalt teistsugune välja, kui siin loodetavasti tuttavaks saanud `public static void Main`. Aga sellegipoolest jääb keele põhisisu samaks. Nii muutujad, tsüklid, valikud, klassid ja objektid on endistviisi vajalikud. Lihtsalt tuleb neile osalt uus ja parasjagu vajalikule toimingule vastav sisu kokku panna.

SQLi keel

SQL on andmebaaside juhtimiseks kasutusel olnud juba mitu aastakümmet. Pole ta sugugi ainuke andmekirjelduskeel ega ka mitte päringukeel. Aga levinumates andmebaasides on ta siiski teinud jõudsa võidukäigu, nii et kel vaja andmetega tihedamalt tegelda, see SQList ei pääse.

Pea iga andmebaasitootja on keelele oma lisandusi pakkunud, mis rakendustele võimalusi ja keerukusi juurde toonud. SQL-92-ga standardiseeriti kõige üldisemad käsklused. Omi nippe ja andmetüüpe jagub aga tootjatel küllaga.

Siin kirjutises keskendutakse eripärade juures MS SQL Serveri võimalustele. Näited on tehtud SQL Server 2008 abil.

Alustatakse "puust ette ja punaseks" seletusest, kuidas oma andmebaas luua, sinna tabel lisada ja andmed sisse panna. Edasi liigutakse graafilistelt näidetelt koodi suunas, minnes mõnikord ka tasemeni, mida lihtrakenduste koostamisel hädasti vaja pole. Nii et kui lugedes/õppides tundub, et näited/seletused ka kolmandal lugemisel arusaamatud tunduvad, võib vähemasti peatükkide lõpus olevad osad selleks korraks laagerduma jätta ning nende juurde vajadusel uuesti tagasi tulla: siis, kui tööd tehes paistab, et oleks vaja keerukamaid päringuid kokku panna, aga lihtsate vahenditega ei taha välja tulla. Selleks ajaks on tõenäoliselt andmebaasidega ümber käimise juures ka piisav kogemus tekkinud, et on julgust üha uhkemaid päringuid ette võtta.

Esialgu koosnevad andmebaasipäringud tõenäoliselt kuni kümnekonnast sõnast ning nendega on võimalik enamik ettetulevaid muresid andmeotsingu vallas ära lahendada. Aga koos soovide ja tahtmiste ning süsteemide suurustega kipuvad ka päringud kasvama. Nii et pole ime, kui mõne firma andmemajanduse juurde sattudes võib mitmeleheküljelisi päringuid näha, mis esialgu silme eest kirjuks võtavad. Kui aga asuda rahulikult otsast vaatama, siis selgub, et selle suure keerukuse saab vähehaaval täiesti eraldatavateks tervikuteks jagada ning viimased omakorda juba nõnda mõistetavateks tükkideks, et kõigest on võimalik aru saada ja vajadusel oma tarbeks täiendada. Kõige rohkem on vaja tahtmist, kannatust ja pusimissoovi.

Ilusaid päringuid!

Microsoft SQL Server 2008

SQL Server 2008 on siinse materjali kirjutamise ajaks Microsofti viimane andmebaasimootor. Tegemist on millegi enamaga kui lihtsalt andmebaasimootoriga. SQL Server on midagi enamat kui lihtsalt andmebaasimootor. Lisaks relatsioonilisele (relational) andmebaasimootorile kaasneb SQL Serveriga mitmeid teisi komponente.

SQL Server võimaldab hoida andmebaasis nii struktureeritud, poolstruktureeritud (nt XML) kui ka struktureerimata andmeid (nt pildid, muusika). SQL Serveriga kaasnevad teenused, mis võimaldavad andmebaasis olevaid andmeid otsida, sünkroniseerida, analüüsida ning muuta aruanneteks. Andmebaas võib paikneda nii suures serveris, tööjaamas kui ka mobiilses seadmes.

SQL Serveriga kaasnevad lisavahendid, mis võimaldavad andmeid mugavalt kasutada nii oma programmide valmistamisel .NET platvormil kui ka Office paketti kuuluvates programmides.

SQL Server 2008 perekond

SQL Server 2008 perekonda kuulub päris palju liikmeid. Kõik liikmed omavad ühtemoodi head relatsioonilist andmebaasimootorit, kuid erinevad üksteisest kas riistvara kasutamise oskuste või lisavõimaluste poolest.

- Enterprise Edition – sisaldab kogu SQL Server funktsionaalsust ning maksimaalselt laiendamise võimalusi ning on optimeeritud 64-bit protsessorite jaoks.
- Standard Edition – sisaldab kõiki SQL Server funktsioone ning on mõeldud väikeste ja keskmiste ettevõtete tarbeks.
- Web Edition – soodne ja hästi laiendatava server veebirakenduste tarbeks.
- Workgroup Edition – SQL Serveri põhifunktsionaalsus väikestele ettevõtetele ja töörühmadele.

- Express Edition – SQL Serveri tasuta versioon - õppimiseks, arenduseks ning väikeste nõudmistega ärilahenduste teenindamiseks. Tasuta versiooni saate enda tõmmata SQL Serveri kodulehelt http://www.microsoft.com/sql

- Compact Edition – spetsiaalne SQL Serveri versioon mobiilsete seadmete tarbeks

Täpsem info erinevate SQL Serveri perekonnaliikmete kohta leiate aadressidelt http://www.microsoft.com/sql/editions ja http://www.microsoft.com/sql/prodinfo/features/compare-features.mspx

SQL Server Express Edition

SQL Server Express Edition on mõeldud SQL Serveriga tutvumiseks ning ka ärikasutuseks väikeste lahenduste puhul. Express versiooni saab tõmmata aadressilt http://www.microsoft.com/express/sql/download.

Express versioonile on seatud järgmised piirangud: Töötab kuni 1 protsessoriga, kasutab ära kuni 1 GB mälu on saadaval nii 32-bit kui ka 64-bit versioonis ja andmebaasi maksimaalne maht on 4 GB. Lisaks sellele on sealt veel ära võetud mõned teenused, mis on vajalikud suurte andmebaaside ja keerukate programmidega toimetamisel. Täpne funktsionaalsuse kirjeldus on saadaval aadressil http://www.microsoft.com/sqlserver/2008/en/us/editions.aspx

Ettevalmistused installeerimiseks

SQL Server 2008 Express Edition on toetatud Windows XP, Windows 2003, Windows Vista ja Windows 2008 operatsioonisüsteemidel.

Express Editioni 32-bit versioon vajab masinat, kus on vähemalt 256 MB mälu (soovituslik 1 GB või rohkem), ca 1 GB vaba kettaruumi, 1 GHz protsessorit (soovituslik üle 2 GHz).

Express Editioni 64-bit versioon vajab masinat, kus on vähemalt 256 MB mälu (soovituslik 1 GB või rohkem), ca 1 GB vaba kettaruumi, 1 GHz protsessorit (soovituslik üle 1,4 GHz).

Kui soovite lisaks SQL Serverile paigaldada ka haldusvahendid siis on vaja vaba kettaruumi vähemalt 1,9GB!

Paigaldamise protsess on protsess järgmine:

- Tuleb installeerida .NET raamistik 3.5 koos teeninduspaketiga SP1

- Veenduge, et teie masinas oleks Windows Installer 4.5. http://support.microsoft.com/kb/942288

- Kui paigaldate SQL Serveri koos haldusvahenditega siis tuleb installeerida ka Windows PowerShell 1.0. http://www.microsoft.com/windowsserver2003/technologies/management/powershell/download.mspx

- Tuleb eemaldada kõik SQL Server 2008 Beta ning CTP versioonid

- Installida SQL Server 2008 Express koos viimase teeninduspaketiga

- Võite installeerida lisakomponente nagu demoandmebaasid ja abiinfo e. Books Online

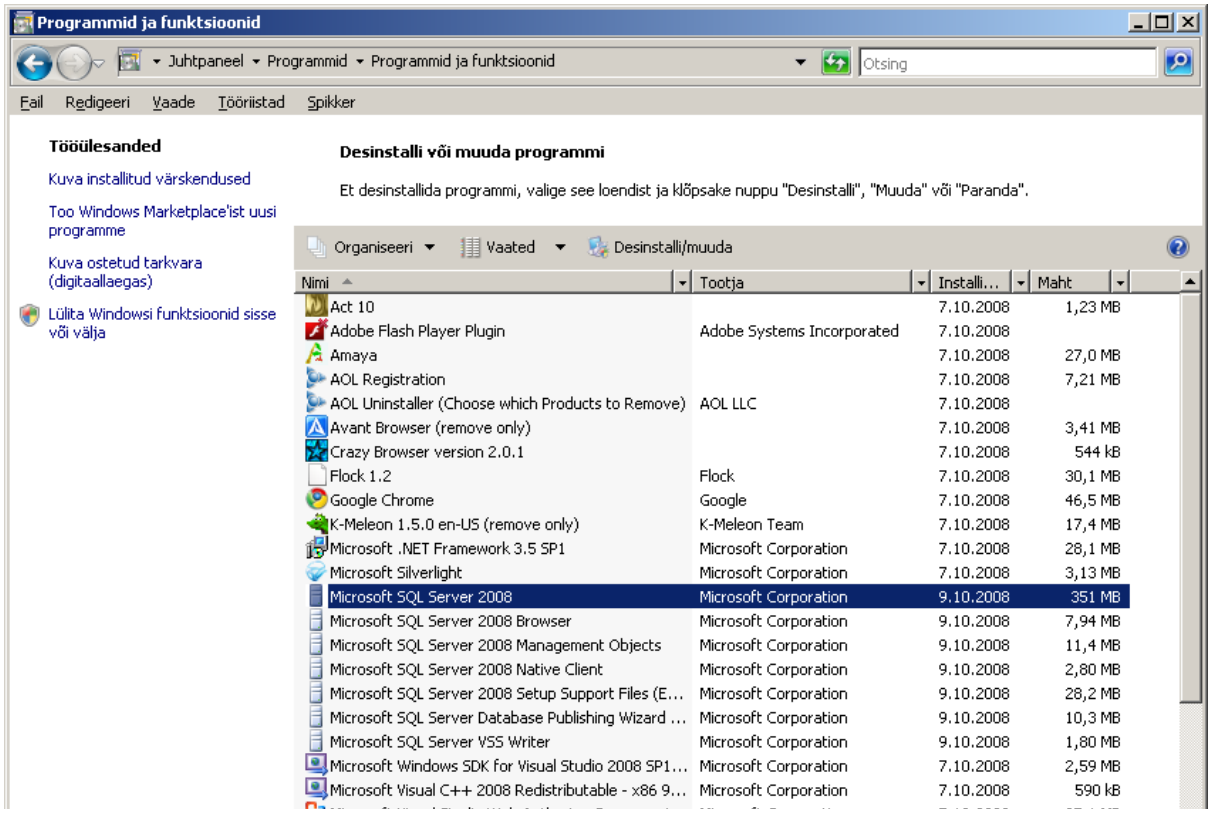
- Võite toote registreerida ja saate Microsoftilt kingitusi ☺

SQL Serveri haldamiseks on olemas käsurea liides, Visual Studio ja SQL Management Studio.

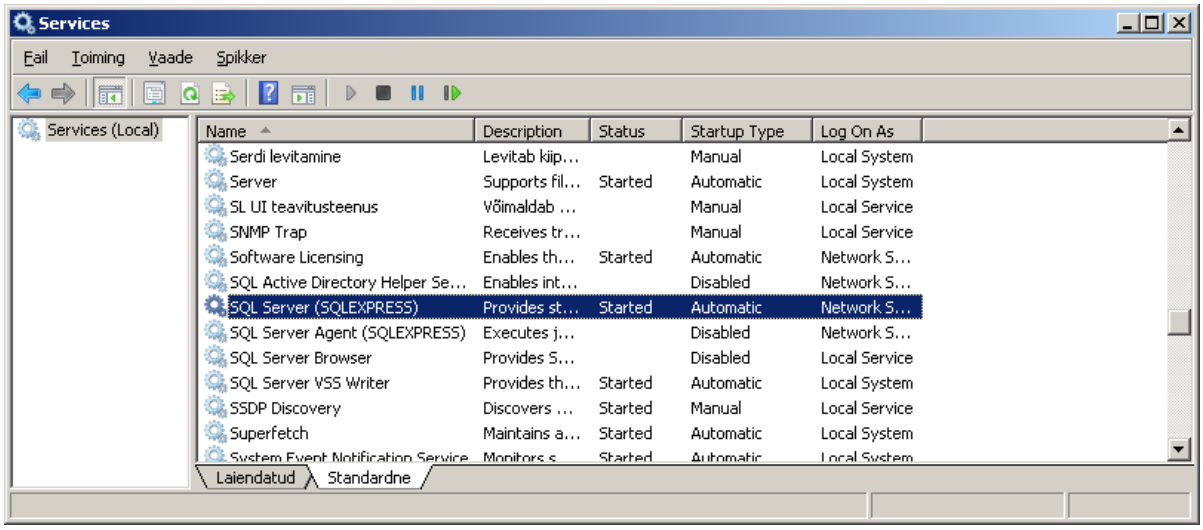
Installeerimine

SQL Serveri installeerimisel olge tähelepanelik, kuna ühte arvutisse võib installeerida mitu SQL Serverit. SQL Server Express Edition kaasneb mitmete Microsofti toodetega. Näiteks kui olete varem installeerinud Visual Studio siis on teil tõenäoliselt SQL Server juba installeeritud!

SQL Serveri olemasolu saate kontrollida läbi juhtpaneeli (*Control Panel*) programmide lisamise (*Add/Remove Programs*) vormi.



Ning lisaks sellele ka Windowsi teenuste alt.



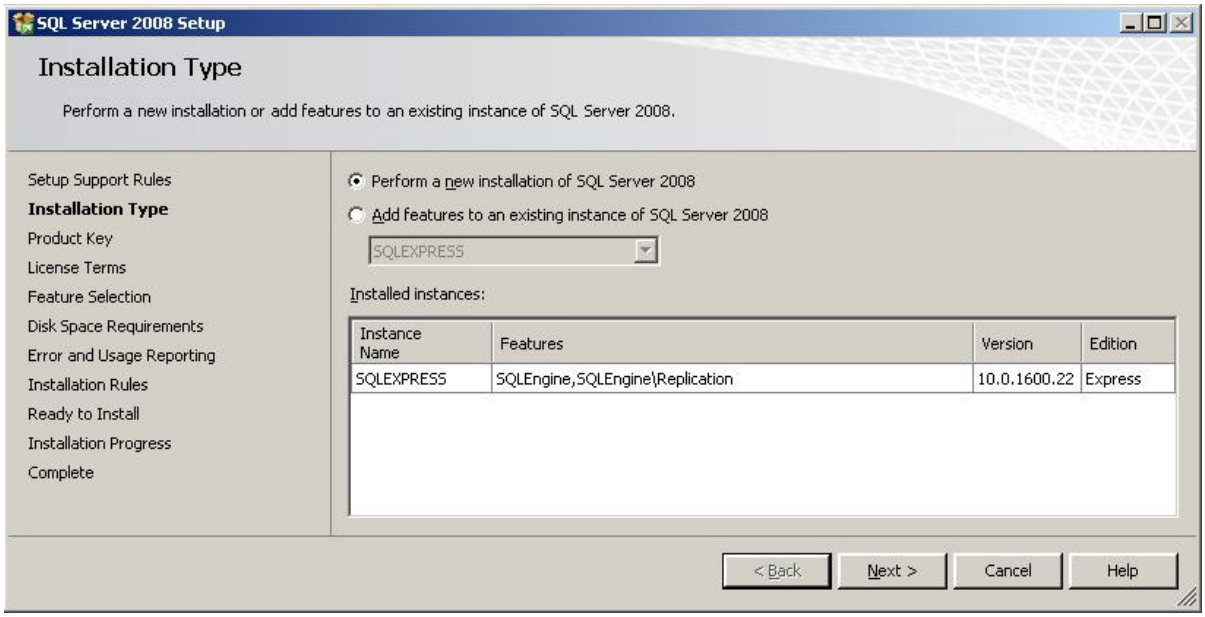
Kui teil on masinasse korra juba SQL Server installeeritud siis ei ole vaja seda uuesti teha! SQL Serveri paremaks haldamiseks on teil täiendavalt vaja paigaldada vaid SQL Serveri haldusvahendid e. Management Studio.

Enne installeerimist, kontrollib installiprogramm teie arvuti sobivust SQL Serveriks ning juhul, kui midagi avastatakse pakutakse ka lisainfot nende probleemide tausta ja võimalike lahenduste kohta.

Kui kõik vajalikud kontrollid tehtud ja planeeringu dokumendid läbi loetud ;) saategi asuda paigaldamise juurde. Paigaldamiseks tuleks valida kas `New SQL Server stand-alone installation or add features to an existing installation` või juhul kui olete eelnevalt juba mõne varasema SQL Serveri Expressi versiooni juba paigaldanud ning soovite versiooni uuendada siis `Upgrade from SQL Server 2000 or SQL Server 2005`.

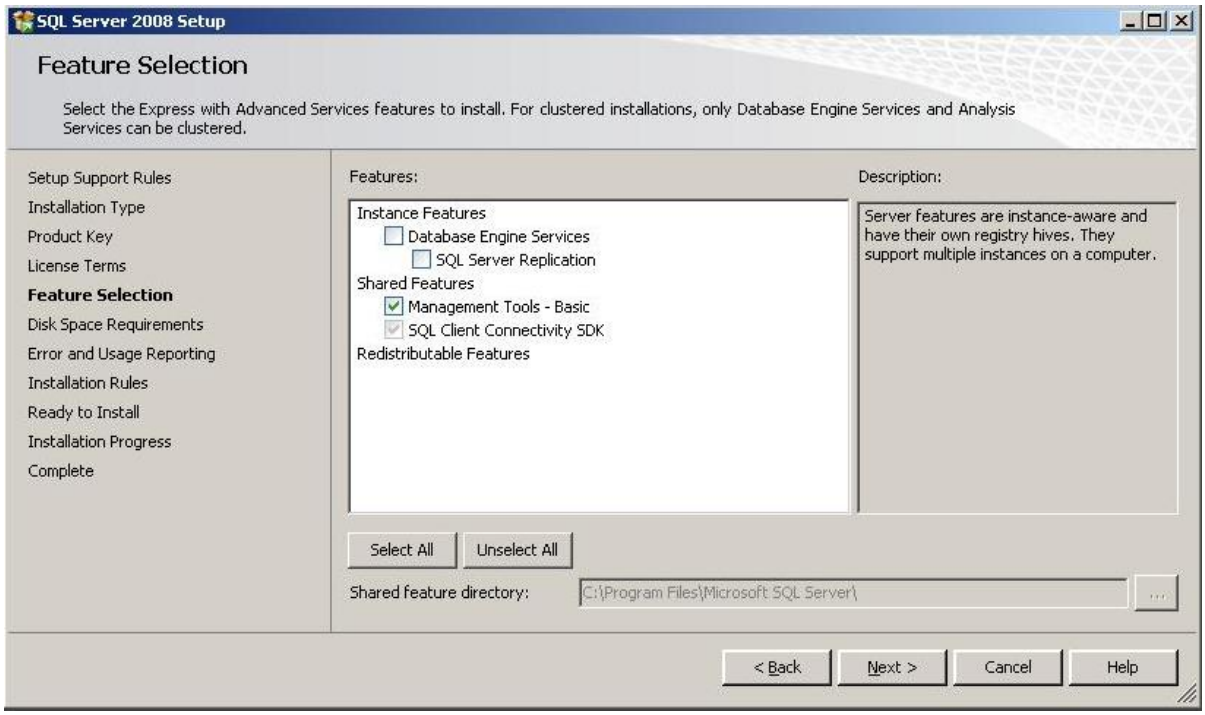
Peale seda, kui installiprogramm on endale vajalikud lisavidinad ära installeerinud saate valida kas paigaldada SQL Server või täiendada olemasolevaid SQL Servereid.

Installeerimiseks tuleks `Perform a new installation of SQL Server 2008`.



Ning peale litsentsilepingu läbilugemist saate valida paigaldatavad komponendid. KUI OLETE VAREM KOOS VISUAL STUDIOGA INSTALLEERINUD KA SQL SERVERI. SIIS EI OLE `DATABASE ENGINE SERVICE` KOMPONENTI INSTALLEERIDA. Kui teil ühtegi SQL Serverit veel paigaldatud ei ole siis installeerige ka `Database Engine Service`

KOMPONENT. Lisaks sellele saate paigaldada haldusvahendid `Management Tools - Basic`, mille abil saate serverit mugavamalt hallata ning kirjutada päringuid.



Peale komponentide valikut, kui teil on kõvakettal piisavalt ruumi, kulgeb install suhteliselt muretult lõpuni.

Seadistamine

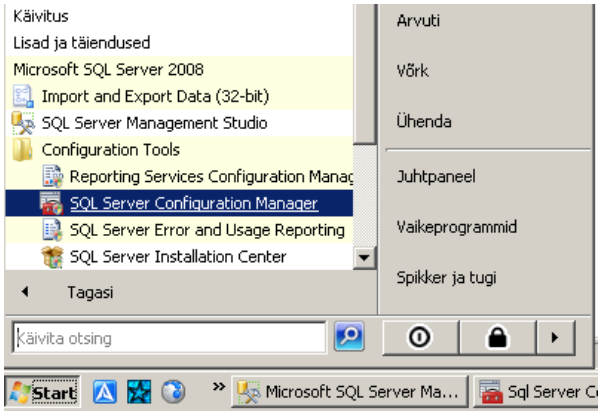
Kui te töötate oma masinas süsteemiülema (*administrator*) õigustes siis lokaalses masinas Visual Studio abil programmeerimiseks ja Management Studio abil päringute loomiseks SQL Server mingit seadistust ei vaja.

Kui tahate aga anda ligipääsu üle võrgu või võimaldada serveri kasutamist ka tavakasutajatele siis tuleb serverit pisut seadistada.

Ligipääsu jagamine serverile käib kolmel erineval tasemel:

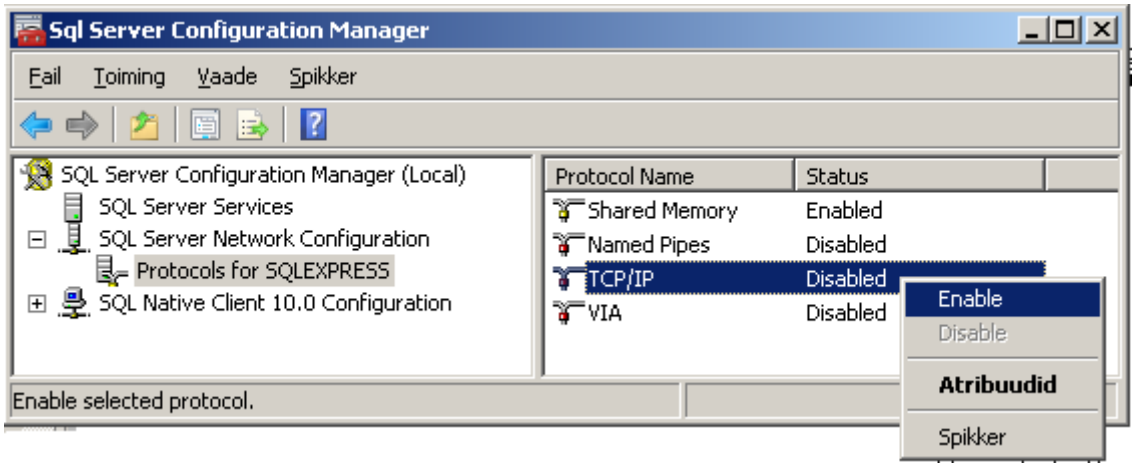
- Ligipääsukanal üle võrgu
- Ligipääs SQL Serveri teenusele
- Ligipääs andmebaasile

Ligipääsukanal üle võrgu

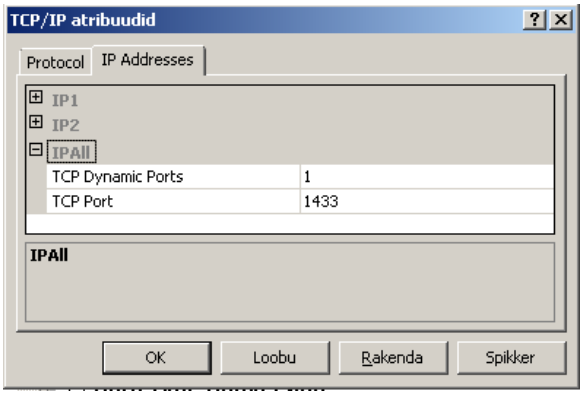


Ligipääsukanalite haldamiseks saate kasutada `SQL Server Configuration Manager`i koosesisus olevat `SQL Server Network Configuration` utiliiti.

Nagu näha all olevalt pildilt oskab SQL Server suhelda väga erinevate suhtluskanalite kaudu. Vaikimisi on lubatud vaid `Shared Memory` kanal e ligipääs üle võrgu on keelatud ning ligi saab ainult serverisse installeeritud vahenditega. Selleks, et anda ligipääs üle võrgu tuleb lubada `TCP/IP` kanal. Selleks klõpsake `TCP/IP` peal hiire parema nupuga ning valige `Enable`.

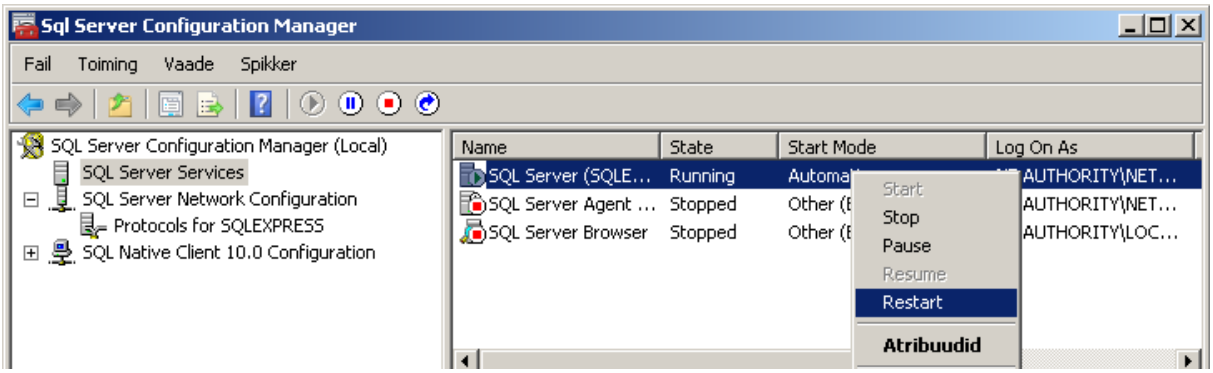


Lisaks kontrollige üle ka `TCP/IP` atribuudid. Vaikimisi soovib SQL Server Express kasutada dünaamilisi porte, mis tähendab seda, et kui teenus käivitub siis valitakse esimene vaba port. Parema töökindluse tagamiseks oleks soovitav määrata üks staatiline port. Enamasti paikneb SQL Server pordis 1433.



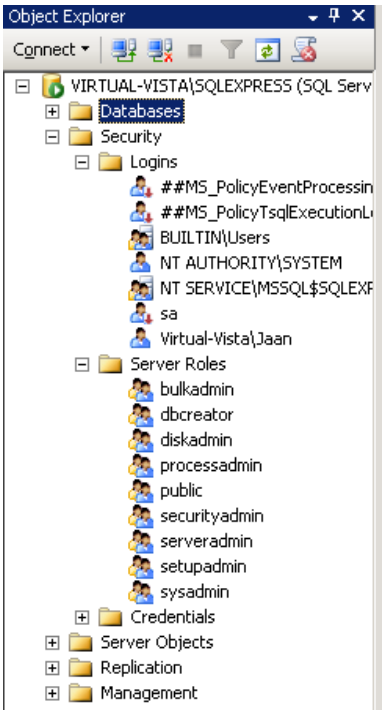
Kui te kasutate tulemüüri siis tuleb teha vastav auk ka tulemüüri!

Kui muudatused tehtud tuleb muudatuste jõustumiseks SQL Serveri teenus taaskäivitada. Selleks valige SQL Server Configuration Managerist SQL Server Services ning tehke SQL Serverile taaskäivitus.



Ligipääs SQL Serveri teenusele

Ligipääsu SQL Serveri teenusele saate hallata läbi Management Studio

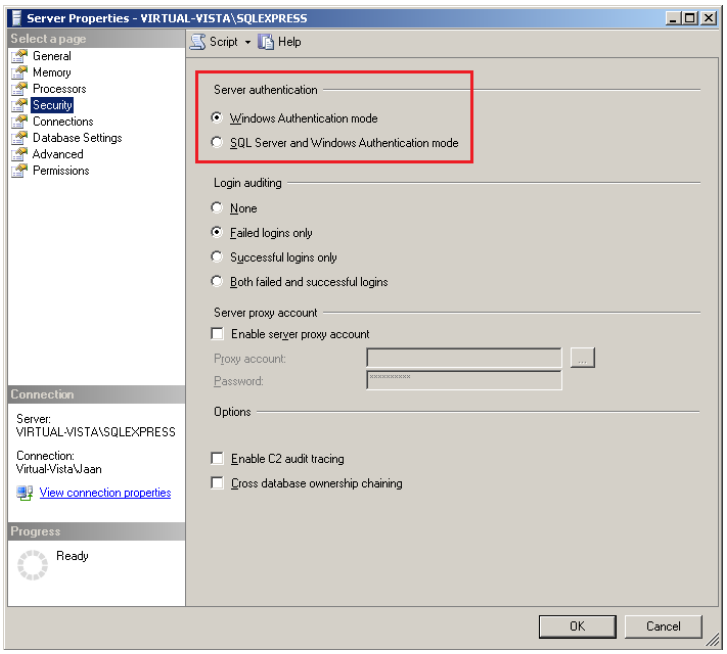


SQL Serverile ligipääsemiseks peate omama kasutajakontot, millele on antud ligipääs SQL Serverile.

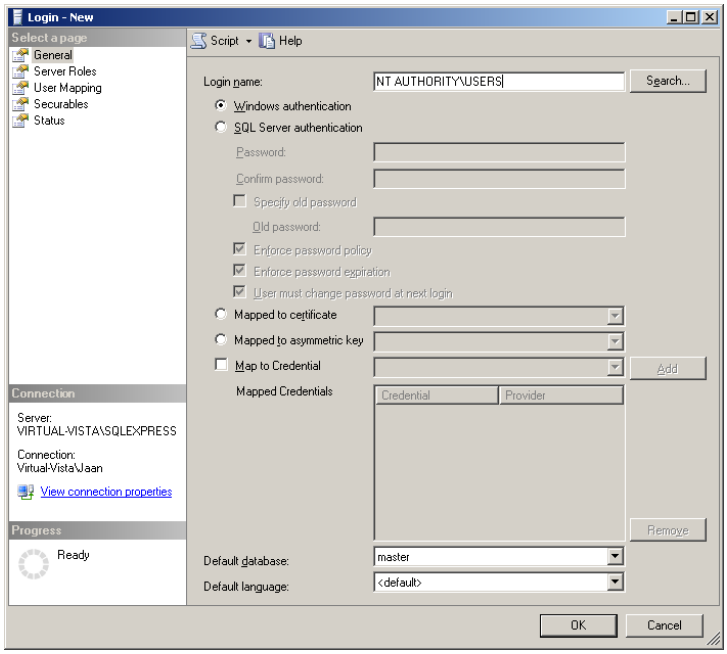
Kõik kasutajad on jagatud gruppidesse. Olulisemad grupid on:

- publik – kõik kasutajad
- sysadmin – süsteemihaldurid omavad kõiki SQL Serveri õigusi. Sellesse gruppi tavakasutajaid ei panda.
- dbcreator – omab õigust andmebaaside loomiseks ja kustutamiseks

Kasutajakontod on leiate Security\Logins alt. Kasutajakontod jagunevad kahte kategooriasse SQL kontod ja Windowsi kontod. SQL kontosid autendib SQL Server, Windowsi kontosid autendib Windows.



Turvakaalutlustel on vaikimisi lubatud vaid Windowsi autentimine. SQL autentimist läheb teil vaja kui mingil põhjusel ei ole Windowsi kasutajakontode kasutamine võimalik nt soovite anda ligipääsu kasutajatele, kes tulevad LINUXi masinatelt.



Kasutajakonto loomiseks klõpsate Management Studios Object Explorers Logins kaustal hiire paremat nuppu ning valige `New Login`. Otsige ülesse või kirjutage Windowsi kasutajatunnus, kellele soovite ligipääsu anda. Õigusi saab jagada ka Windowsi kasutajagruppidele ning enamasti on soovitav kasutada grupiviisilist lähenemist ja mitte anda õigusi igale kasutajale eraldi.

Teiseks määrake ära, millisesse andmebaasi kasutaja vaikimisi satub st kui kasutaja andmebaasi ei täpsusta siis kuhu lähevad tema päringud.

`master` andmebaas on süsteemne andmebaas, kus hoitakse SQL Serveri seadistust ning sinna on ligipääs kõigil kasutajatel.

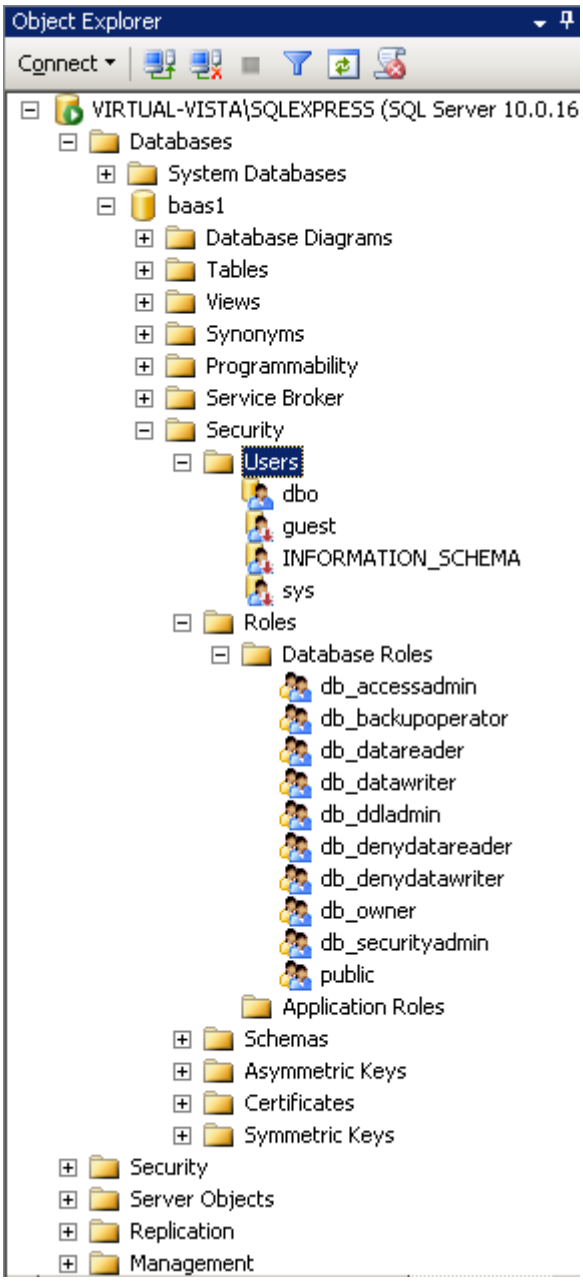
Seda kõike on võimalik teha ka lihtsa SQL käsuga:

```
CREATE LOGIN [DOMEEN\HUVILISED]
FROM WINDOWS
WITH DEFAULT_DATABASE=[master]
```

Kui soovite kasutajale anda eriõiguseid siis saate seda teha paigutades kasutaja sobivasse serveri rolli. Seda saate teha sama vormi `Server Roles` valikust või siis hiljem kasutaja või vastava serverirolli omadustest.

Ligipääs andmebaasile

Selleks, et serverisse pääsenud kasutaja saaks majandada ka teie andmebaasis on vaja anda serveri kasutajatele andmebaasi ligipääs.



Selleks tuleb avada oma andmebaas ning otsida sealt ülesse `Security\Users`

Seal on loetletud kõik kasutajad, kes omavad ligipääsu teie andmebaasile.

Lisaks sellele on vaja loomulikult anda ka ligipääs erinevate andmebaasiobjektide kasutamiseks. Sellega saate määrata ära, kes millisest tabelist tohib andmeid vaadata ning, kes kuhu tohib andmeid kirjutada. Loomulikult ka kõik eriõigused nagu andmebaasi objektide loomine.

Selle viimase tegevuse lihtsustamiseks on soovitus kasutada rolle. Vajadusel saate luua oma rolli, millele määrate unikaalse õigustekomplekti. Kuid on olemas ka terve hulk süsteemseid kasutajate rolle, mille abil saate luua arvestatava turvalisusega ligipääsu.

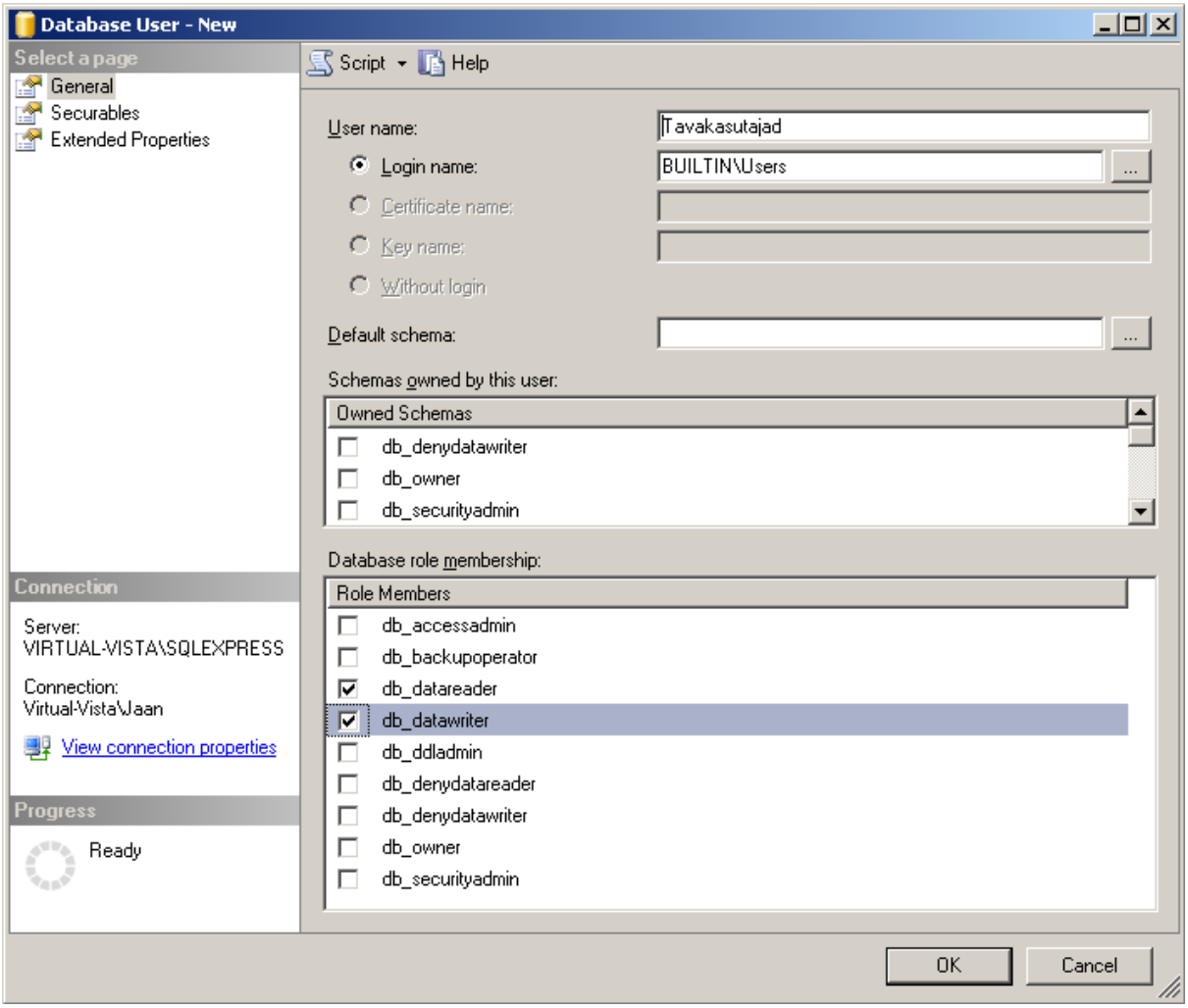
Olulisemad andmebaasirollid on:

- db_datareader – õigus lugeda andmeid kõigist tabelitest (table) ja vaadetest (view) ning käivitada kõiki funktsioone ja protseduure
- db_datawriter – õigus muuta andmeid kõigis tabelites
- db_owner – andmebaasi omanik e. õigus teha andmebaasiga kõike sh. muuta andmebaasi struktuuri. Vaikimisi on andmebaasi omanikuks andmebaasi looja

Kasutaja lisamiseks klõpsake `Security\Users` peal hiire paremat nuppu ning valige `New User …`

Andke serverikasutajale nimi, mille järgi soovite teda näha oma andmebaasis ning määrake ära, millistesse rollidesse see kasutaja kuulub.

Kui Teil on andmebaas valmis piisab enamasti kui kasutajal on `db_datareader` ja `db_datawriter` õigused. Kui vajate õigusi andmebaasi objektide (tabelid, vaated jne) loomiseks siis oleks vaja `db_owner` õiguseid.



Kõike seda saab teha ka väikse SQL skriptiga:

```
USE [baas1]
GO

CREATE USER [Tavakasutajad] FOR LOGIN [BUILTIN\Users]
GO
```

```
EXEC sp_addrolemember N'db_datawriter', N'Tavakasutajad'
EXEC sp_addrolemember N'db_datareader', N'Tavakasutajad'
GO
```

Põhivõimalused

Töö alustamine

SQL Server on väga hea andmete hoidaja ja töötleja, kuid tal puudub kasutajaliides käskude sisestamiseks. Selleks, et panna SQL Server enda pilli järgi tantsima, on Teil vaja mingit programmi, mille kaudu SQL Serverile korraldusi jagada. Selleks programmiks võib olla nii teie enda loodud programm nt mõnes .NET keeles, Microsoft Office paketist Excel ja Access või siis mõni Microsofti poolt pakutav töövahend. SQLi haldamiseks pakub Microsoft põhiliselt kolme vahendit: Visual Studio, SQL Server Management Studio ning SQL Server Command Line Tool e. sqlcmd (varem tuntud kui oSql utiliit).

Visual Studio on mõeldud professionaalsetele programmeerijatele, kes oma andmete hoidmiseks kasutavad SQL Serverit.

SQLi käsurea utiliit (`sqlcmd`) on mõeldud administraatoritele, kes ei soovi oma serverisse installeerida graafilisi vahendeid või soovivad mingeid kindlaid toimingid ajastada kasutades Windowsi ajastatud töid (*Scheduled Tasks*). Puhtalt SQL Serveriga tegelemiseks on kõige parem vahend SQL Management Studio, mis võimaldab kasutada SQL läbi graafilise liidese ning ühtlasi võimaldab ka edastada SQL käske kirjalikult.

SQL Management Studio on tasuta utiliit, mida on võimalik endale muretseda Microsofti kodulehelt.

Management Studio käivitamisel küsitakse, millist teenust soovite kasutada, millise serveri külge ühenduda ning millist autentimismoodust kasutada.

Teenustest on valida:

- Database Engine e. SQL andmetega manipuleerimine
- Analysis Services e andmete kaevamine
- Reporting Services e. aruannete koostamine
- SQL Server Mobile e. lubatakse SQL Serveri andmefaili peal kasutada vaid neid funktsioone, mida toetab SQL Serveri mobiilne versioon
- Integration Services e. andmete import, eksport ning teisendamine

Kui Serverite sirvimine on keelatud peate teadma SQL Serveri nime. SQL Serveri nime ütleb Teile SQLi administraator. Kuna ühte masinasse on võimalik installeerida mitu SQL Serverit siis võib juhtuda, et on vaja lisaks serveri nimele teada ka SQL nimelise instantsi (eksemplari)

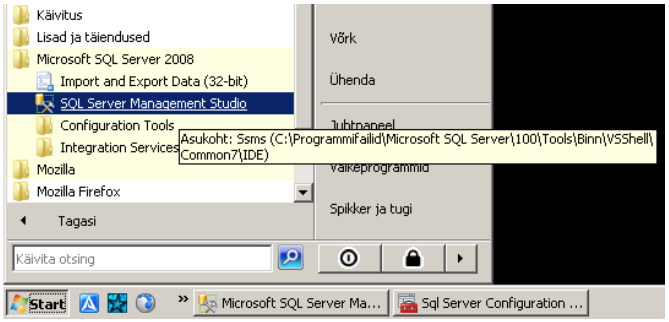
nime. Instantsi nime pole vaja kui soovite pöörduda Serveri vaikimisi instantsi poole. Serveri nimi tuleb kirjutada kujul `<server>\<instance>`. SQL Express installeeritakse tavaliselt `SQLEXPRESS` nimelise instantsi alla e. serveri nimeks võiksite panna `ARVUTINIMI\SQLEXPRESS`.

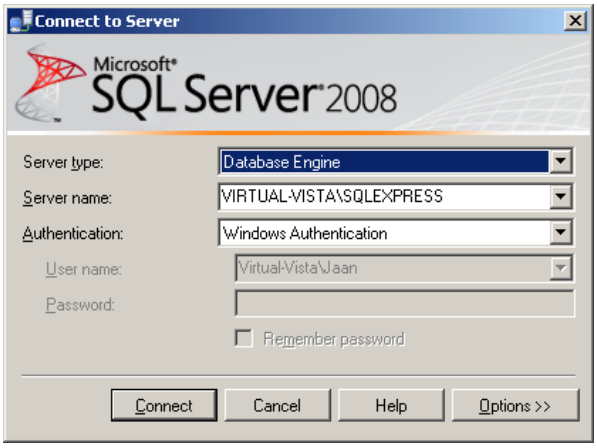
SQL Serveris peab teil iga konkreetse tegevuse jaoks olema õigus. Selleks, et õigusi kontrollida on vaja SQL Serverile öelda, kes Te olete. Ütlemise võimalusi on kaks:

- Windows Authentication – kasutatakse Windowsi autentimist. Ühendute SQL Serveri külge sama nimega, millega sisse logisite. See on kasutatav juhul, kui SQL Server on installeeritud Teie arvutisse või kui on ülesse seatud Windowsi domeen. Kasutades Windowsi autentimist kontrollib teie konto kehtivust ning parooli Windows ning SQL usaldab Windowsilt saadud infot. Tegemist on soovitusliku autentimise meetodiga kuna Windowsi turvapoliitikad ning kontrollmehhanismid on märksa intelligentsemad kui SQL Serveri enda omad.

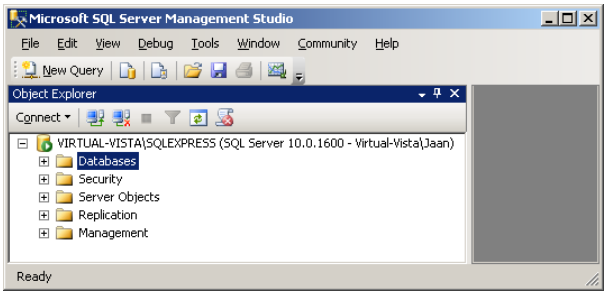
- SQL Server Authentication – SQL Serveris on kirjeldatud kasutajad ning SQL Server peab ise kontrollima kas Teie parool on õige. Sellist lahendust kasutatakse vaid juhul, kui Windowsi autentimise kasutamine pole võimalik e. olukorras kus on vaja serveriga suhelda erinevatel klientarvutitel, mis pole domeeni liikmed ning keda üldjuhul ei usaldata. Vaikimisi on selline suhtlus keelatud.

Autentimise meetod ning vajadusel kasutajanimi ning parool on taas väärtused, mille ütleb teile SQL Serveri administraator.



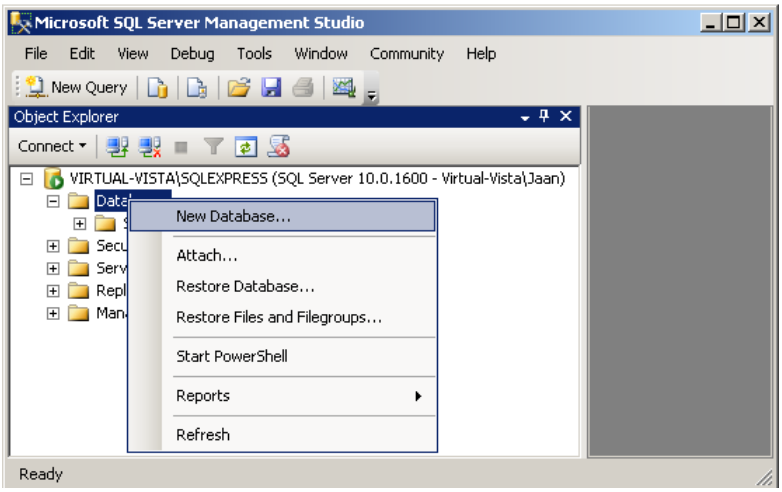


Kui ühendamine õnnestus, tuleb ette Object Exploreri aken, mille kaudu võib serveris paiknevat ja toimuvat näha ja muuta. Paremal pool asuvast Summary aknast saate vaadata detailsemat infot valitud objekti kohta

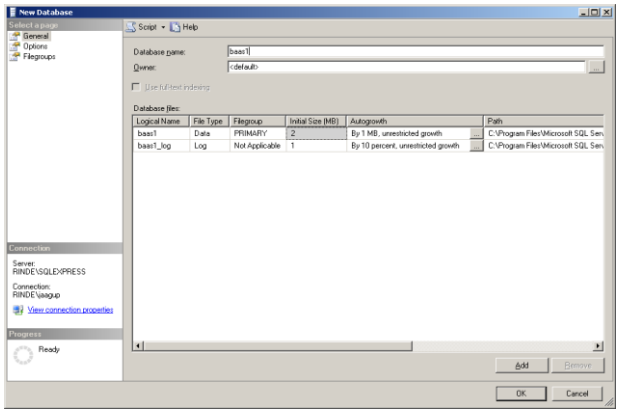


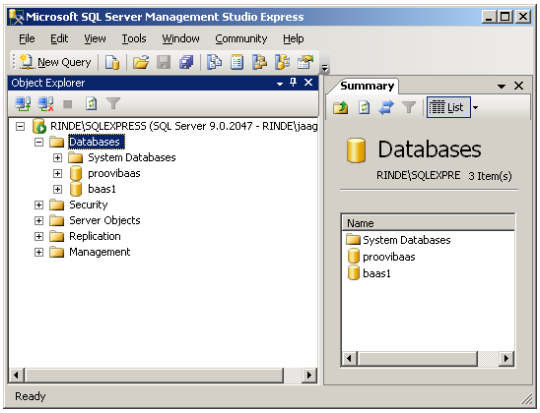
Andmebaasi loomine

Andmebaasi loomiseks klõpsake Databases kataloogi peal hiire paremat klahvi ning valige New Database…



Baasi loomise juures küsitakse kõigepealt andmebaasi nime. Saagu selleks baas1. Baasile saab hulga omadusi määrata, mis suuremate ja tihedasti kasutatavate baaside administreerimisel on küllalt tähtsad.. Path teatab, kuhu kettale ja kataloogi andmed salvestatakse. Initial Size kaudu öeldakse algne baasi jaoks reserveeritud kettamaht. Üldiselt oleks kasulik andmebaasi eeldatav suurus välja arvutada ning teha andmebaas kohe õige suurusega. Sellisel juhul fragmenteerub andmebaasifail ketta peal vähem ning üldine jõudlus on parem. Autogrowth ütleb, kui suurte sammudega ja kui suure mahuni on salvestatavatel andmetel lubatud arvutis kasvada. Nagu näha on eraldi failid andmete eneste ja logide jaoks. Kõik muudatused andmebaasis kirjutatakse esmalt logisse ning seejärel salvestatakse andmebaasi. Selline kahekäiguline täitmine võimaldab vajadusel teatud tegevusi e. transaktsioone tagasi kerida. Väikestel andmebaasidel on enamasti üks andmefail ning üks logifail. Suurematel ja keerukamatel andmebaasidel võib nii logi kui ka andmefaile olla mitmeid. Kasulik on see näiteks juhul, kui mõni ketas on kiirem kui teine – sinna saab panna sagedamini vajaminevaid andmeid. Samuti võib juhtuda, et mõningaid andmeid kasutatakse omavahel tihemini koos. Kõik see on peenema programmeerimise ning administreerimise rida, praeguse alguse juures piisab, kui vajutada OK ning baas ongi olemas. Pärast refresh-menüü valikut andmebaaside alt võib uue nimega baasi ka loetelus näha.

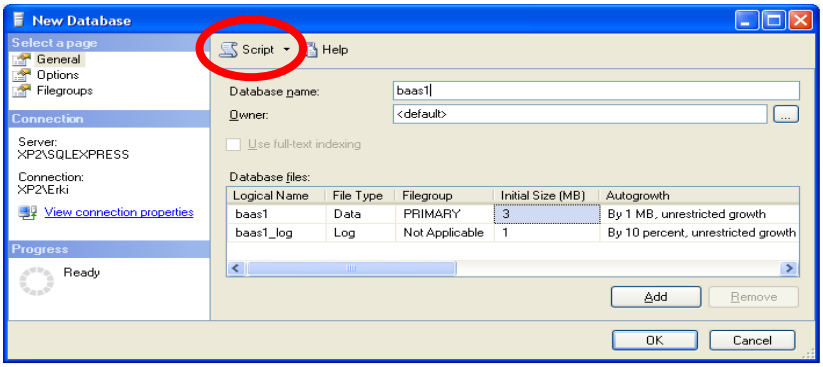




Graafiline liides on tore katsetamiseks ning uurimiseks, kuid tõsisemaks programmeerimiseks oleks vaja välja uurida kuidas SQL tegelikult asjadest aru saab ning kuidas teha nii, et peale programmeerimise lõppu oleks lihtne kogu loodud tarkus teise serverisse transportida.

Selgub, et andmebaasiserveriga suhtlemiseks on loodud omaette keel SQL-keel, milles on võimalik teha kõiki samu tegevusi, mis graafiliselt ja veel palju enamatki.

Kui soovite teada, milline on SQL käsk, mis teeb neid samu asju, mida püüdsite äsja graafiliselt teha võite vajutada akna ülaosas olevat Script nuppu.



Tulemuseks on järgmine SQL konstruktsioon

```
CREATE DATABASE baas1 ON PRIMARY
( NAME = N'baas1',
  FILENAME = N'D:\MSSQL\DATA\baas1.mdf' ,
  SIZE = 3072KB ,
  FILEGROWTH = 1024KB )
LOG ON
( NAME = N'baas1_log',
  FILENAME = N'D:\MSSQL\DATA\baas1_log.ldf',
  SIZE = 1024KB,
  FILEGROWTH = 10%)
GO
```

Saadud SQL lausest on näha, et andmebaasi saab tekitada kasutades CREATE DATABASE lauset

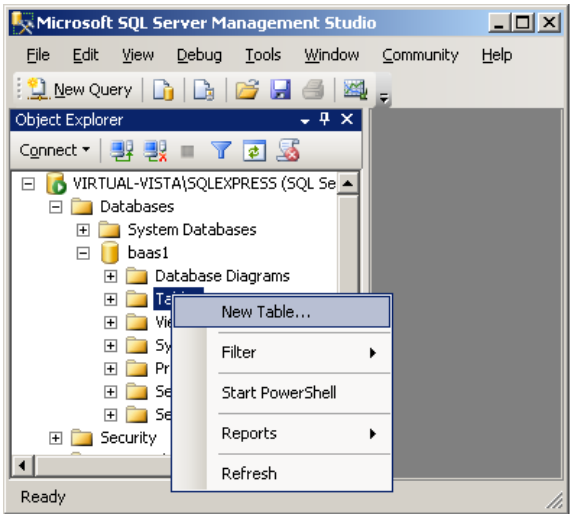
Siinsel juhul tehakse andmebaas nimega baas1, mille 3MB suurune andmefail baas1.mdf pannakse D: kettal kausta MSSQL\DATA kuhu läheb ka 1 MB suurune logifail baas1_log.ldf. Kui failimaht saab täis siis andmefail hakkab kasvama 1MB kaupa ning logifail 10% kaupa. Skriptist on näha ka see, et igal failil on kaks nime: üks määratud NAME ja teine FILENAME atribuudiga. Neist nimedest esimene on faili loogiline nimi, mida kasutab SQLi administraator ning teine on mõeldud operatsioonisüsteemile. Need nimed võivad olla erinevad kuid lihtsuse mõttes on kasulikum hoida need nimed ühesugused.

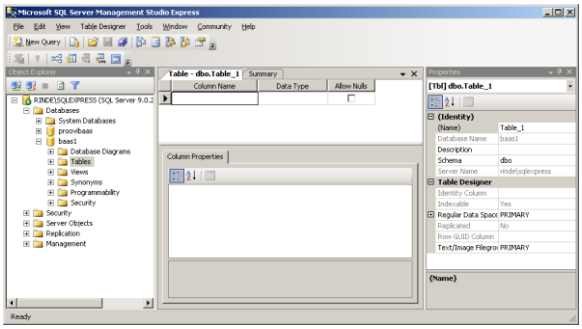
Tabeli loomine

Palja baasi olemasolust veel andmete hoidmiseks ei piisa. Relatsiooniliste andmebaaside puhul paiknevad andmed tabelites. Siin käime esialgu läbi tabeli loomise graafilise tee, hiljem vaatame ka programmikäskudega sättimise võimalusi.

Harilikku tabelit kujutab igaüks ette. Hulk ridasid ja veerge, andmeid täis. Eks andmebaasitabelid ole üsna sarnased – lihtsalt mõned reeglid ja piirangud on juures. Näiteks peab igal veerul olema andmetüüp: täisarv, reaalarv, kuupäev, tekst või midagi muud lubatut. Ja kõik vastavas veerus paiknevad andmed peavad vastavat tüüpi olema. Kui andmed erinevad üksteisest omaduste poolest – näiteks inimeste kontaktandmed ja autode tehnilised parameetrid – siis peavad need eri tüüpi kirjed olema eri tabelites. Katsetuseks aga üks lihtne linnanimede ja rahvaarvude tabel, mille puhul ei tohiks eksimist ja segadusi olla.

Avage oma andmebaas, vajutades hiirega andmebaasi ees olevat + märki ning `Tables` alt valik `New Table...` ja juba võibki hakata veergusid looma. Igale veerule nimi, andmetüüp ning linnuke selle kohta, kas väärtus võib puududa (`allow nulls`). Programmeerija elu on üldjuhul lihtsam, kui väärtus on alati olemas, st. NULLid pole lubatud. Aga kui päriselus sellegipoolest võib juhtuda, et mõnda lahtrisse pole väärtust kusagilt võtta, siis on tühjus siiski enamasti parem, kui mõni kokkuleppeline muu väärtus. Kuigi – vahel pannakse arvuliste andmete juures teadmata kohale nt. -1 või siis 999. Viimane näiteks inimese pikkuse juures, kus usutavad väärtused kuhugi paarisaja kanti ning üheksate riviga on kohe näha, et tegemist pole õige asjaga.





Esimesele veerule saab nimeks id või kood. Kui pole erilist põhjust selle veeru ära jätmiseks, siis üldjuhul tasub see `id`-veerg alati panna. Nõnda on igal real järjekorranumber, mille järgi saab rea poole pöörduda. Muidu võib kergesti juhtuda, et kahel asulal või kahel inimesel on sama nimi ning hilisemate päringute või muutmiste juures pole selge, millise reaga tegeldakse. Kui aga panna loenduriga tulp, millel traditsiooniliselt on nimeks id, siis sellist muret ei teki. Kuna tabeleid võib andmebaasis olla mitu ning nende vahel on enamasti seosed, on kasulik `ID`-le lisaks kirjutada ka mõni täpsustav märkus nt `LinnID`. Sellisel juhul saame kõikjal, kus vaja viidata linnale kasutada sama väljanime.

Kõik nimed peavad algama tähega, millele võivad järgneda nii tähed kui ka numbrid. Täheks loetakse ka alakriipsu. SQL Server 2005 lubab kasutada ka täpitähti ja tühikuid, kuid nende kasutamine pole soovitav kuna võib tekitada probleeme andmebaasi liigutamisega erinevate serverite vahel ning seab lisakohustusi päringute koostamisel. Nime maksimaalne pikkus on 128 märki.

Andmetüübid

Igale veerule tuleb valida andmetüüp. Esialgu tundub neid loetelus arutu hulk olema. Lähemal vaatlusel aga selgub, et tüüpe polegi liialt palju. Rohkem kasutatavad ehk täisarvud, reaalarvud, aeg, tekst, binaarvorming ja XML. Keerukamate andmebaaside tarbeks on võimalik andmetüüpe isegi juurde tekitada.

Täisarvud: `tinyint, smallint, int` ja `bigint` võtavad 1, 2, 4 ja 8 baiti mälus, ehk siis vastavalt on määratud suuruspiirid, milleni vastavas veerus andmeid salvestada saab. Tavalisim int on 4 baiti ja lubatud suurimad arvud seega ±2 miljardi kanti.

Reaalarvud: float, `decimal, money, numeric, real, smallmoney`. `Numeric` ja `decimal` sünonüümid ning põhjused, miks nii on juhtunud on ajaloolised. Reaalarvude hoidmiseks on kaks moodust:

- kasutada täisarve millel teatud arv kohti on reserveeritud murdosa tarbeks
- kasutada ligikaudseid numbreid e. salvestada arvust vaid esimesed x numbrit

Esimest meetodit kasutavad numeric ja `decimal` andmetüübid ning teist meetodit `float` ning `real` andmetüübid.

Erinevus seisneb selles, et kasutades täisarvu on alati teada, et nt kui kirjutan `decimal(15,5)` siis 10 kohta on enne koma ning 5 kohta peale koma. Kui üritan kirjutada enne koma 11 kohalise numbri, saan veateate ning kui kirjutan peale koma 6 kohta siis number ümardatakse 5 kohani peale koma. Kui kasutan aga `float(15)` andmetüüpi, salvestatakse arvust 15 esimest numbrit ning kümneaste ning lõpp ümardatakse. See võimaldab salvestada samale väljale nt 1000 kohalise arvu kuid selle arvu täpsus on määratud 15 esimese numbriga ning lõpp on ära lõigatud.

`Money` ja `SmallMoney` kasutavad salvestamiseks esimest metoodikat ning on mõeldud rahaliste väärtuste hoidmiseks.

Kuupäevade ja kellaaegade hoidmiseks on mitmeid erinevaid andmetüüpe. `date` – kuupäev 1 päevase täpsusega, `time` – kellaaeg 100 nanosekundilise täpsusega, `smalldatetime` – kuupäev ja kellaaeg täpsus 1 minut, `datetime` – kuupäev ja kellaaeg täpsus 3millisekundit, `datetime2` – kuupäev ja kellaaeg täpsus 100 nanosekundit, `datetimeoffset` kuupäev ja kellaaeg 100 nanosekundilise täpsusega koos ajavööndiga.

Tekst: `char, varchar` ja `text` tekstide jaoks. Esimene kindla pikkusega väljadele (nt. isikukood), teine muutuva pikkusega tekstide jaoks (nt. asutuse nimetus) ning kolmas

pikematele tekstidele. Sinna kõrvale käivad `nchar` (national char), `nvarchar` ja `ntext` Unicode kodeeringut kasutavate tekstide tarbeks.

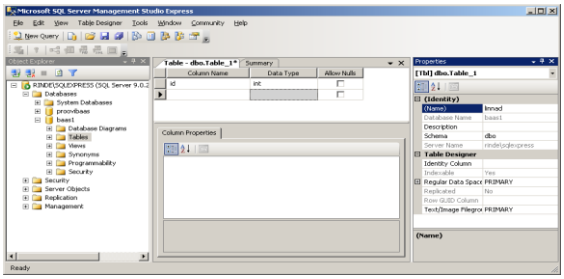
Lisaks sellele pakub SQL Server mitmeid spetsialiseeritud andmetüüpe: Geograafiliste andmete (pikkus- ja laiuskraadide) hoidmiseks on saadaval spetsiaalne andmetüüp `geography`. Tasapinnalises koordinaatsüstemis punktide hoidmiseks on andmetüüp `geometry`. Andmetüüp `hierarchyid` on mõeldud hierarhilise (puu tüüpi) struktuuri hoidmiseks.

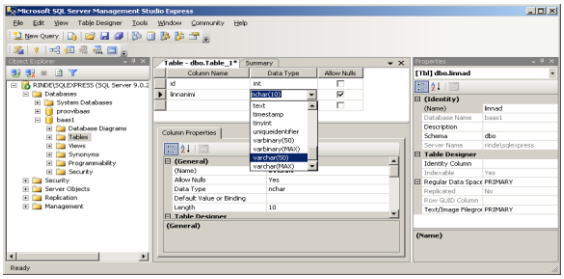
Enamiku lihtsamate väiksemate andmebaaside puhul saab tavaliselt hakkama nelja tüübiga: `int`, `float`, `datetime` ja `varchar`. Kui võib eeldada jõudlusprobleeme või muid olukordi, kus andmebaasiosa saab rakenduse pudelikaelaks, siis tasub lähemalt uurida andmebaasi poolt võimaldatavaid töö kiirendamise või andmemahtude kokkuhoiu mooduseid.

Tabeli nimi võiks tabeli sisuga seotud olla. Table_1 ei ütle sisu kohta suurt midagi. Pigem saab paremast tulbast tabeli nime ära muuta, andes talle siin nimeks "linn".

Tabeli nimed võiksid olla ainsuse nimetavas käändes.

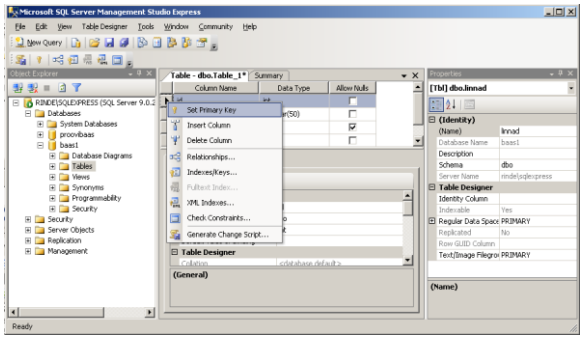
Kuna kõigi andmebaasiobjektide nimed peavad olema erinevad võib nime külge panna erinevaid ees ja järelliiteid. Nt Linn_tbl, mis ütleb, et tegemist on tabeliga, millest leiad linnade nimed.

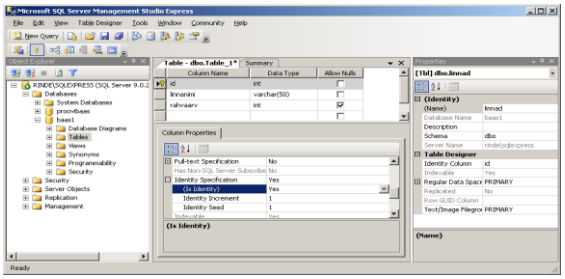




Primaarvõti

Tulpa, mille järgi tabeli ridadele viidatakse ning mille juures kindlasti on kõik väärtused erinevad, nimetatakse üldjuhul primaarvõtmeks. See on viisakas tabeli loomise juures ka ära määrata. Parema klahvi klõps tulba juures ning valik `Set Primary Key` ning tulba ette tekkiski primaarvõtit tähistav ikoon. Edasi tasub arvutile selgeks teha, et ridade numbreid automaatselt loendataks. Selleks võib id-tulba alt otsida sektsiooni `Identity Specification` ning sealt omaduse `Is Identity` väärtuseks panna Yes. Ülejäänud vaikeseaded võivad paika jääda juhul, kui oleme rahul olukorraga, kus loendama hakatakse numbrist 1 ning järgmine arv tuleb igal korral ühe võrra suurem.





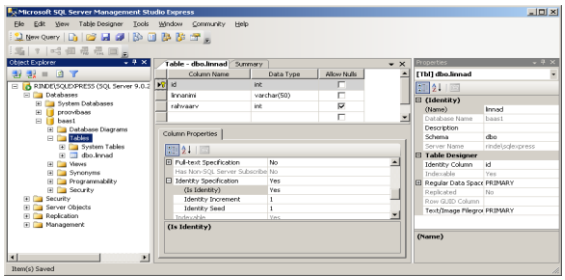
Vajutades salvestusnuppu, kantakse tehtud muudatused tegelikult andmebaasi ning vasakul andmemenüüs võib näha tabelite all uut tekkinud tabelid dbo.linnad. Andmete mugavamaks sisestamiseks hiire parema klahviga klõps tabeli nimele ja valik "Open Table".

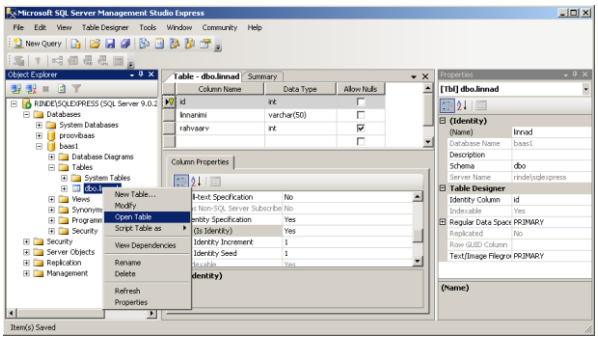
Taas on võimalik kõik kogu protsess teisendada SQL käskudeks. Selleks tuleb valida Table Designer\Generate Change Script ning tulemus on järgmine:

```
CREATE TABLE dbo.Linn_tbl
(
	LinnID int NOT NULL IDENTITY (1, 1) PRIMARY KEY
	, Nimi varchar(50) NOT NULL
	, Rahvaarv int NULL
)
GO
```

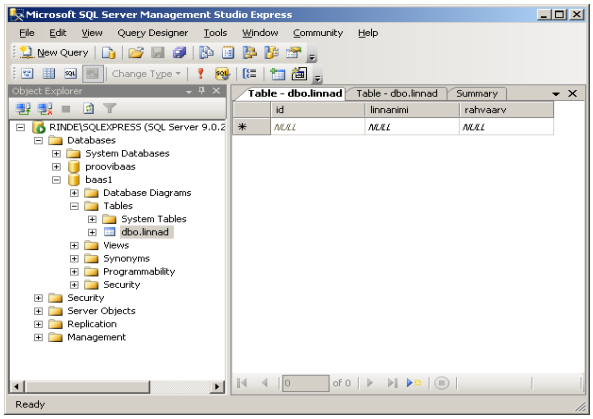
Nagu näha saab tabeleid teha CREATE TABLE lausega, mille järgi tuleb kirjutada tabeli nimi ning selle järele sulgudesse komadega eraldatud loetelu tabelis olevatest väljadest koos andmetüübi ning muude omadustega.

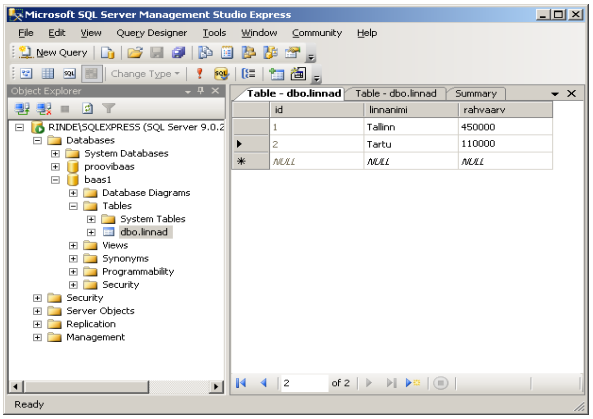
Andmete sisestus





Avanenud aknas saab rahumeeli andmeid juurde lisada. Identifitseerimistulba LinnID-numbrid kasvavad automaatselt, nimi ja rahvaarv tuleb ise sisse kirjutada.





Lisamislause tekitamine pole enam kahjuks nii lihtne kui andmebaasi või tabeli loomise/muutmise tegevus. Lisamise lause tekitamiseks on kasulik moodustada endale väike spikker valides Object Explorerist hiire parema klahviga Script Table as/INSERT to/New Query Editor Window.

Tulemuseks on järgnev SQL:

```
INSERT INTO [baas1].[dbo].[Linn_tbl]
      ( [Nimi], [Rahvaarv])
VALUES
      ( <Nimi, varchar(50),> ,<Rahvaarv, int,>)
```

Seega on andmete lisamiseks INSERT INTO lause, millele järgneb tabel, kuhu andmed lisatakse ning selle järgi sulgudes väljad, millele väärtused omistada. Peale VALUES märksõna tuleb samas järjekorras kirjutada ka lisatavad väärtused.

Nagu näha pole vaja väärtust omistada automaatselt nummerduvat välja LinnID ning lisaks sellele võib jätta väärtustamata kõik väljad millele on määratud vaikeväärtus või lubatud määramatus e. NULL.

Kui soovime lisada linnade loetellu nt Tartu 110000 elanikuga tuleks muuta saadud SQLi järgmiselt:

```
INSERT INTO [baas1].[dbo].[Linn_tbl]
      ([Nimi], [Rahvaarv])
VALUES
      ('Tartu', 110000)
```

Harjutus (tabeli loomine)

Tekitame tabeli Laps, milles kirjas lapse nimi, pikkus, sünniaasta ning sünnilinn.

Kes soovib võib teha tabeli kasutades graafilisi vahendeid kuid soovitan harjutada SQL keelt ning teha tabel kasutades SQL lauseid. SQL laused võimaldavad

- paremini kontrollida serverile antavaid käske
- laused on võimalik salvestada skriptidesse, mille käivitamisel saame kogu protsessi korrata

Management Studios SQL-lausete kirjutamiseks tuleb luua vastav tekstiaken. Selleks

File->New->Query with Current Connection

Igas tabelis peab olema primaarvõti e. lisaks eelpool loetletud väljadele võtame kasutusele ka välja LapsID, mis võiks olla taas automaatselt nummerdatav.

Lihtsuse mõttes salvestame tabelisse vaid eesnimed ning loodame, et nimi pole üle 40 tähe pikk (enamasti salvestatakse inimeste nimed kahel väljal ees- ja perekonnanimi, mis võimaldab andmeid paremini otsida ja sorteerida)

Pikkust hoiame sentimeetrites keskmise suurusega täisarvulisel väljal.

Kuna soovime hoida andmebaasis ainult sünniaastat, mitte sünnipäeva siis kasutame ka sünniaasta tarbeks täisarvulist numbrivälja, mitte kuupäeva välja. Arvutatavaid väärtusi (näiteks vanus) üldjuhul tabelis ei hoita vaid arvutatakse vajalikult hetkel. Kui arvutatavat välja on vaja väga tihti võib selleks alates SQL Server 2000st teha eraldi valemit sisaldava välja (computed column)!

Linna tarbeks kasutame juba varem loodud Linn_tbl tabelis olevaid linnakoode. Et kahe tabeli ühendamine oleks lihtsam, peavad mõlemal pool olema väljad ühte tüüpi ning ühesuurused.

Seega võiks laste tabeli teha järgneva SQL lausega:

```
CREATE TABLE dbo.Laps_tbl
(
     LapsID INT NOT NULL IDENTITY (1,1) PRIMARY KEY
     , Nimi VARCHAR(40) NOT NULL
     , Pikkus SMALLINT NULL
     , Synniaasta SMALLINT NULL
     , SynniLinn INT NULL
     , Vanus AS YEAR(GETDATE()) - Synniaasta
)
```

Väli vanus on arvutuslik. Funktsioon GETDATE() annab hetke aja e. kuupäev kellaaeg. Funktsioon YEAR eraldab antud kuupäevast aasta. Seega leitakse vanus arvutusega jooksev aasta miinus sünniaasta.

Lisame sellesse tabelisse ka mõned andmed:

Nimi	Pikkus	Synniaasta	SynniLinn
Juku	155	1997	1
Kati	158	1997	2
Mati	164	1995	2
Ats	163	1996	1
Siiri	153	1996	1
Madis	174	1995	1
Siim	163	1997	2

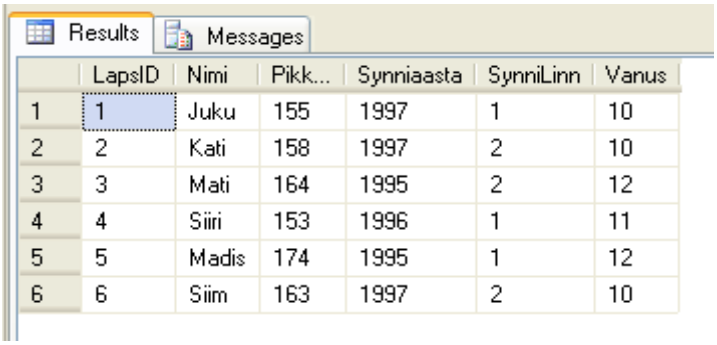
Selleks moodustame iga sisestatava rea tarbeks eraldi SQL lause:

```
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Juku', 155, 1997, 1)
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Kati', 158, 1997, 2)
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Mati', 164, 1995, 2)
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Siiri', 153, 1996, 1)
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Madis', 174, 1995, 1)
INSERT INTO [baas1].[dbo].[Laps_tbl]
([Nimi], [Pikkus], [Synniaasta], [SynniLinn])
VALUES ('Siim', 163, 1997, 2)
```

Avades tabeli vaatamiseks või kirjutades SQL Lause kujul:

```
SELECT *
FROM dbo.Laps_tbl
```

Saame tulemuseks tabeli, milles 6 rida:



Results | Messages

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus
1	1	Juku	155	1997	1	10
2	2	Kati	158	1997	2	10
3	3	Mati	164	1995	2	12
4	4	Siiri	153	1996	1	11
5	5	Madis	174	1995	1	12
6	6	Siim	163	1997	2	10

Nagu näeme on kõigile lastele tekitatud (sisestamise järjekorras) koodid ning vaatamiseks arvutatud vanused. Vanuseid reaalselt kusagile salvestatud ei ole ning need arvutatakse lahutades käesolevast aastast sünniaasta.

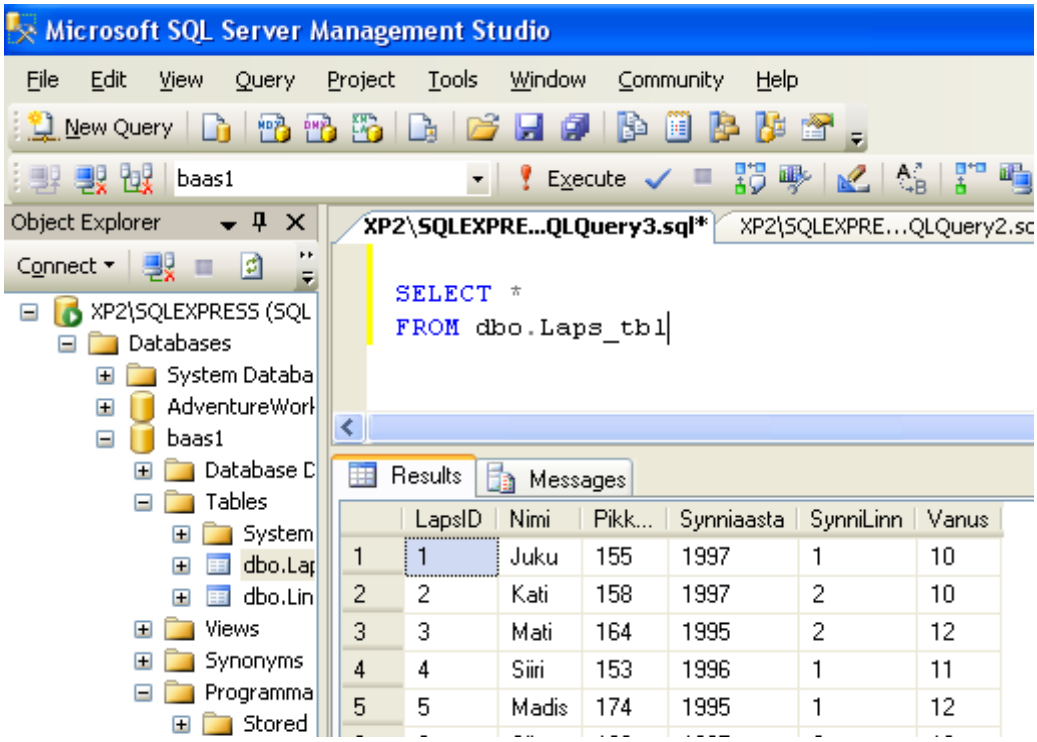
Lihtsamad päringud

Enne, kui andmebaasi loomisega edasi läheme vaatame, kuidas oleks võimalik juba sisestatud andmeid vaadata ja uurida.

Olemasolevate andmete kättesaamiseks sobib päringulause SELECT. Lihtsaim käsk kõigi olemasolevate andmete tabelist kätte saamiseks:

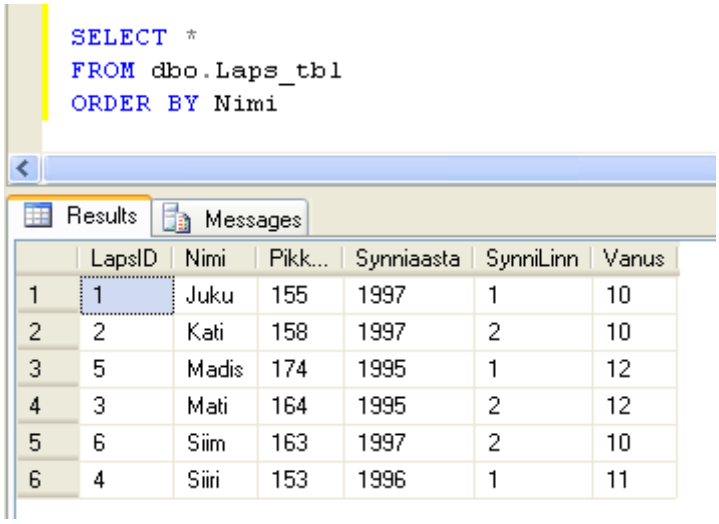
```
SELECT *
FROM dbo.Laps_tbl
```

Tulemusena joonistub rakenduse allserva kogu tabelitäis andmeid koos tulpade nimedega.



Sellist päringut kasutatakse vaid erandolukordades ning väga väikeste tabelite juures. Reaalses tööolukorras tuleks kindlasti loetleda ülesse kõik väljad, mida soovite vaadata ning seada piirangud ridade arvule!

Järjestamiseks piisab lisaklauslist ORDER BY, millele järgneb tulba nimi. Kui soovime kõik tabelis olevad andmed trükkida sorteerituna nimede järgi võime kirjutada:



Ja tulevadki andmed nimede järgi sordituna. Et esimeses tulbas olevad id-d näevad juhuslikult segi paisatutena välja, see on täiesti loomulik. Kui sorditakse nime järgi, siis tõstetakse read niimoodi ümber, et eesnimed lähevad tähestikulisse järjekorda. Iga rea andmed aga kuuluvad

endiselt kokku. Nii nagu Siiri oli algul 153 sentimeetrit pikk ja sündinud aastal 1996, nii on ta seda ka pärast järjestamist. Ja samuti tema id-number jääb neljaks.

Tahtes sorteerimisjärjekorra muuta vastupidiseks, tuleb tulba nimele lisada tähed DESC (sõnast descending). Ja ongi Siiri esimene ja Juku viimane.

```
SELECT *
FROM dbo.Laps_tbl
ORDER BY Nimi DESC
```

Results | Messages

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus
1	4	Siiri	153	1996	1	11
2	6	Siim	163	1997	2	10
3	3	Mati	164	1995	2	12
4	5	Madis	174	1995	1	12
5	2	Kati	158	1997	2	10
6	1	Juku	155	1997	1	10

Järjestust määravaid tulpi võib olla mitu. Sellisel juhul tuleb ORDER BY järgi loetleda väljad tähtsuse järgi. Nt võttes sorteerimise aluseks sünniaasta ning seejärel nime saame, et sõnniaastad on sorteeritud kasvavasse järjekorda ning kui samal aastal on sündinud mitu last on nad sorteeritud nimede järgi:

```
SELECT *
FROM dbo.Laps_tbl
ORDER BY Synniaasta, Nimi
```

Results | Messages

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus
1	5	Madis	174	1995	1	12
2	3	Mati	164	1995	2	12
3	4	Siiri	153	1996	1	11
4	1	Juku	155	1997	1	10
5	2	Kati	158	1997	2	10
6	6	Siim	163	1997	2	10

Sugugi alati pole andmete juures vaja kõiki tulpasid näha. Kui soovin vaadata vaid nime ja pikkust võin selle info panna ka SELECT lausesse, loetledes kõik vajalikud väljad SELECT´ i järel.

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
```

Results | Messages

	nimi	pikkus
1	Juku	155
2	Kati	158
3	Mati	164
4	Siiri	153
5	Madis	174
6	Siim	163

Samuti saab seada piirangu ridade näitamise suhtes. Siin vaid lapsed, kelle sünnilinn on Tallinn e. linna kood on 1

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
WHERE SynniLinn = 1
```

Results | Messages

	nimi	pikkus
1	Juku	155
2	Siiri	153
3	Madis	174

Või vaatame, millised lapsed on nooremad kui 12:

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
WHERE Vanus < 12
```

Results | Messages

	nimi	pikkus
1	Juku	155
2	Kati	158
3	Siiri	153
4	Siim	163

Kui tulemusse tekivad korduvad read saame nendest vabaneda kasutades DISTINCT märksõna. Näiteks soovime välja selgitada milliste sünniaastatega lapsed meil tabelis on? Kui kirjutame SELECT´i ilma DISCTINCT märksõnata saame loetelu, kui kõigi laste sünniaastad, kui lisame DISTINCTI saame kõik erinevad sünniaastad:

```
SELECT Synniaasta
FROM dbo.Laps_tbl
ORDER BY Synniaasta
```

Results | Messages

	Synniaasta
1	1995
2	1995
3	1996
4	1997
5	1997
6	1997

```
SELECT DISTINCT Synniaasta
FROM dbo.Laps_tbl
ORDER BY Synniaasta
```

Results | Messages

	Synniaasta
1	1995
2	1996
3	1997

Võime piiranguid seada ka numbrivahemike järgi. Selleks on kaks võimalust:

- Kasutada BETWEEN operaatorit
- Kombineerida kaks võrratust AND operaatoriga

```
SELECT Nimi, Synniaasta
FROM dbo.Laps_tbl
WHERE Synniaasta BETWEEN 1995 AND 1997
ORDER BY Nimi
```

Results | Messages

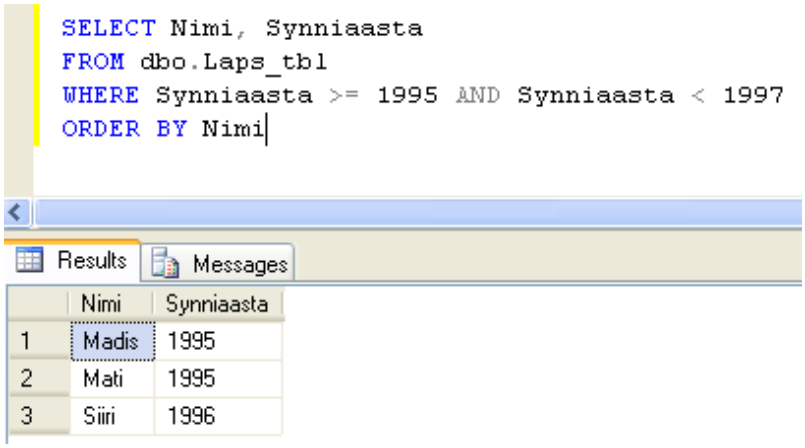
	Nimi	Synniaasta
1	Juku	1997
2	Kati	1997
3	Madis	1995
4	Mati	1995
5	Siim	1997
6	Siiri	1996

Sama tulemuse saaksime ka järgmise SQL lausega:

```
SELECT Nimi, Synniaasta
FROM dbo.Laps_tbl
```

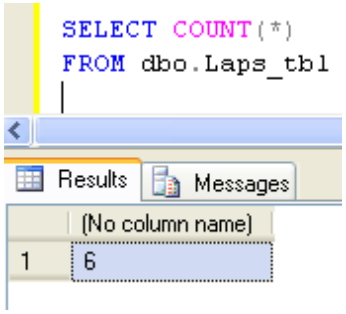
```
WHERE Synniaasta >= 1995 AND Synniaasta <= 1997
ORDER BY Nimi
```

BETWEEN operaatori eeliseks on lihtsus ja ülevaatlikus. Loogikaavaldise kasuks räägib aga paindlikkus - nimelt võime mõnest osapoolest võrduse ära võtta jättes täpsed väärtused välja. Näiteks otsime lapsi, kes on sündinud alates aastast 1995, kuid enne aastat 1997:



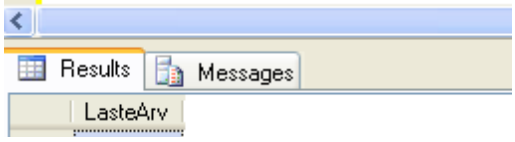
Agregaatfunktsioonid

Lihtsamad kokkuarvutamised saab samuti SQL-keele abil ära teha, nende jaoks pole vaja täiendavate programmide abi vaja otsida. Alljärgnevalt toodud funktsioonide töö tulemuseks on siinsetel juhtudel terve tabeli kohta vaid üks arv. Aga eks see summa, suurima, vähima, koguse või keskmise leidmisel nii olegi. Tahtes päringust välja tulnud ridu kokku lugeda, aitab funktsioon COUNT(*). Saame teada, et praegu nimekirjas olevaid lapsi on 6.



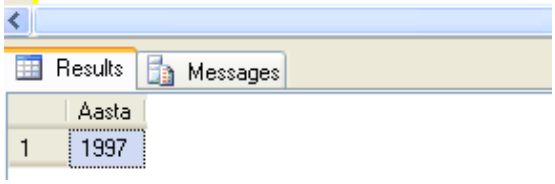
Nagu näeme puudub saadud tulemusel pealkiri. Kui soovime SELECT´is loetletud välju ümber nimetada või neile nimesid anda tuleb kasutada AS märksõna mille järgi saate kirjutada sobiva nime.

```
SELECT COUNT(*) AS LasteArv
FROM dbo.Laps_tbl
```



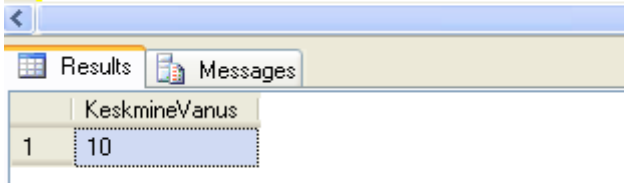
Suurima sünniaasta saab kätte käsuga MAX. Sarnaselt töötab ka MIN.

```
SELECT MAX(SynniAasta) AS Aasta
FROM dbo.Laps_tbl
```



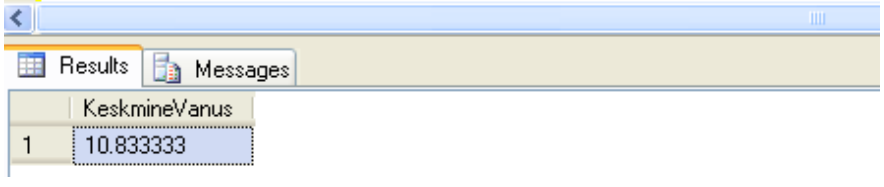
Või leiame tabelis olevate laste keskmise vanuse kasutades AVG funktsiooni (sõnast average)

```
SELECT AVG(Vanus) AS KeskmineVanus
FROM dbo.Laps_tbl
```



Nagu näeme on tulemus täpselt sama täpne kui lähteandmed. Kui soovime andmetüüpi arvutuste käigus muuta tuleb lähteandmed teisendada sobivale kujule ning sooritada tehte pärast teisendust. Näiteks kui teisendame esmalt kõik vanused reaalarvudeks ning arvutame seejärel keskmise saame hoopis põnevama tulemuse:

```
SELECT AVG(CAST(Vanus AS DECIMAL)) AS KeskmineVanus
FROM dbo.Laps_tbl
```



Funktsioon CAST ütleb SQLile, et käsitle seda andmevälja nii nagu ta oleks seda teist tüüpi. Keerukamate arvutuste jaoks tuleb kasutada CONVERT funktsiooni.

Kui on põhjust väärtused kokku liita, aitab funktsioon SUM. Praegusel kujul võib ette kujutada, et kui kõik lapsed heidaksid üksteise järele pikali nii, et ühe pea puudutab järgmise taldu, siis kokku oleks nende pikkus 9,67 meetrit.

```
SELECT SUM(Pikkus) AS KoguPikkus
FROM dbo.Laps_tbl
```

Results | Messages

	KoguPikkus
1	967

Ette rutates märkus, et need funktsioonid saavad hakkama ka oludes, kus mõni väärtus on tulpa sisestamata – näiteks pole paari lapse pikkust teada. Sellisel juhul puuduvaid ehk tühiväärtusi lihtsalt ei arvestata. Vaid COUNT(*) loeb kõik read kokku. Kui tahetaks saada vaid olemasolevate pikkustega ridu, siis aitaks COUNT(pikkus).

Muutmine

Muutmiste juures tuleb meeles pidada mitmeid olulisi asjaolusid:

- Kõik muudatused tehakse transaktsioonis e. kas kõik või mitte midagi. St kui soovite muuta tabelist kümmet rida ning ühe rea muutmine ei õnnestu, ei muudeta ka ülejäänud üheksat.

- Server eeldab, et käske jagab programm või intelligentne inimene ning käsu andja ei eksi st kõik käsud täidetakse vastu vaidlemata ning mingit tagasiteed ei ole. Kui andmebaas on osavalt seadistatud on võimalik andmebaasist andmeid taastada sekundi või isegi transaktsiooni täpsusega, kuid kõik mis juhtus peale seda läheb kaduma. See tähendab seda, et kui kogemata muudate midagi ära, avastate, et muudatus oli vale ning soovite, et administraator taastaks selle andmebaasi teie tegevusele eelnenud seisus ning selle otsuse vastuvõtmiseks kulus 5 minutit siis sisuliselt kustuvad kõik viimase 5 minuti muudatused!

- Sel ajal kui teie muudate ei saa teised samu andmeid kasutada (kui administraator pole korraldanud teisiti ;)

Olge muutmiste juures äärmiselt ettevaatlikud!

Püüame muuta Mati (kood 3) pikkust. Uueks pikkuseks on 171.

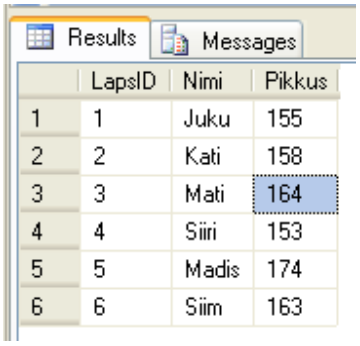
Andmete muutmiseks on UPDATE lause, mille järgi tuleb kirjutada muudetav tabel, seejärel peale SET märksõna loetleda ülesse kõik muudetavad väljad ning nende uued väärtused. Ärge unustage piiranguid ridadele! Kui ridade arvu pole piiratud tehakse muudatus terves tabelis!

```
UPDATE dbo.Laps_tbl
```

```
SET Pikkus = 171
WHERE LapsID = 3
```

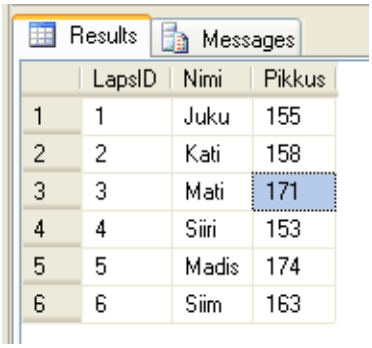
Selles lauses on näha ka kui oluline on unikaalne kood. Kui koode ei oleks ning me püüaksime muuta nime baasile e. Nimi = 'Mati' siis muudetaks kõigi Matide pikkused!

Enne:



	LapsID	Nimi	Pikkus
1	1	Juku	155
2	2	Kati	158
3	3	Mati	164
4	4	Siiri	153
5	5	Madis	174
6	6	Siim	163

Pärast:

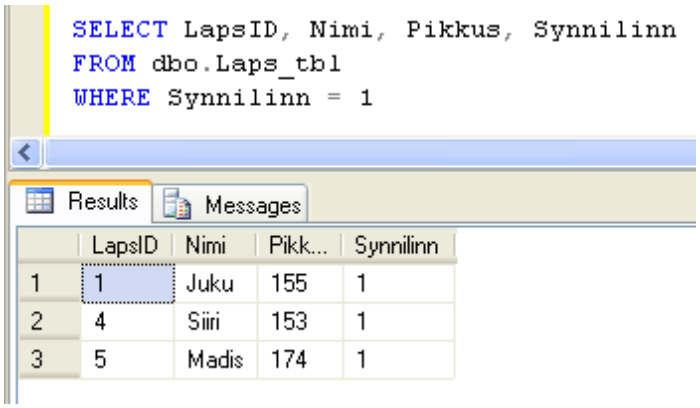


	LapsID	Nimi	Pikkus
1	1	Juku	155
2	2	Kati	158
3	3	Mati	171
4	4	Siiri	153
5	5	Madis	174
6	6	Siim	163

UPDATE-käskluse puhul peab alati ettevaatlik olema, et WHERE-osa puudu ei jääks või vigaseks ei osutuks. Puuduva osa puhul muudetakse ära kõikide ridade tulbad etteantud väärtuseks. Edasi võib ainult varukoopia peale loota. Vigase WHERE puhul satub muutmise alla lihtsalt vale rida.

Sellest tulenevalt oleks kasulik enne muutmist kontrollida, milliseid ridu muutma asute. Näiteks käisid mõõtmisel kõik Tallinna lapsed ning selgus, et kõik on 5cm pikemaks kasvanud.

Sellisel juhul otsime esmalt ülesse kõik Tallinna lapsed:



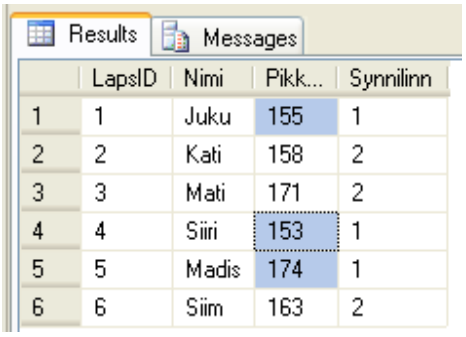
	LapsID	Nimi	Pikk...	Synnilinn
1	1	Juku	155	1
2	4	Siiri	153	1
3	5	Madis	174	1

Seega kui asume muutma kasutades sama WHERE tingimust peavad muutuma 3 lapse (Juku, Siiri ja Madis) pikkused.

```
UPDATE dbo.Laps_tbl
```

```
SET Pikkus = Pikkus + 5
WHERE Synnilinn = 1
```

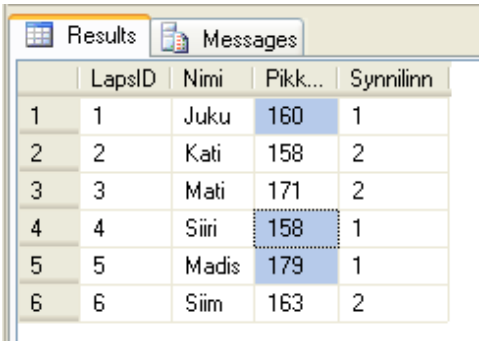
Enne:



Results | Messages

	LapsID	Nimi	Pikk...	Synnilinn
1	1	Juku	155	1
2	2	Kati	158	2
3	3	Mati	171	2
4	4	Siiri	153	1
5	5	Madis	174	1
6	6	Siim	163	2

Pärast:



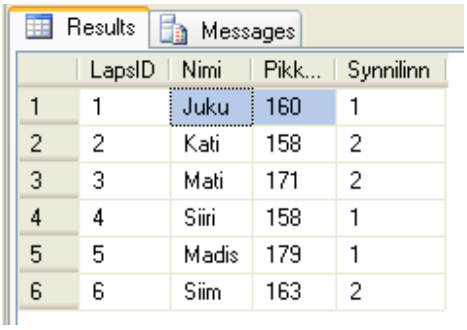
Results | Messages

	LapsID	Nimi	Pikk...	Synnilinn
1	1	Juku	160	1
2	2	Kati	158	2
3	3	Mati	171	2
4	4	Siiri	158	1
5	5	Madis	179	1
6	6	Siim	163	2

Ühe UPDATE-lausega saab muuta ka mitme tulba väärtusi korraga. Selleks tuleb omistamised eraldada komadega. Ehk siis näiteks kui Juku kasvab pikemaks ning saab Juhaniks siis saame selle vormistada järgmise SQL lausega:

```
UPDATE dbo.Laps_tbl
SET Pikkus = 185, Nimi = 'Juhan'
WHERE LapsID = 1
```

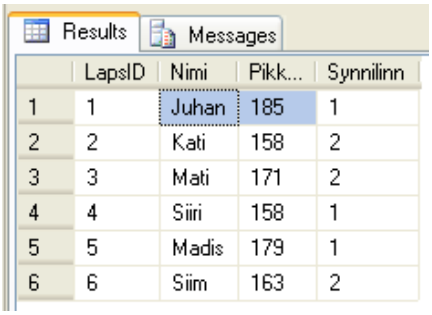
Enne



Results | Messages

	LapsID	Nimi	Pikk...	Synnilinn
1	1	Juku	160	1
2	2	Kati	158	2
3	3	Mati	171	2
4	4	Siiri	158	1
5	5	Madis	179	1
6	6	Siim	163	2

Pärast



Results | Messages

	LapsID	Nimi	Pikk...	Synnilinn
1	1	Juhan	185	1
2	2	Kati	158	2
3	3	Mati	171	2
4	4	Siiri	158	1
5	5	Madis	179	1
6	6	Siim	163	2

Kustutamine

Kustutamiseks saame kasutada DELETE lauset, mille järgi kirjutame tabeli, millest soovime ridu kustutada. Kindlasti tuleks lisada ka tingimus selle kohta, milliseid ridu kustutame! Kui tingimus ära jätta e. kirjutada nt:

```
DELETE dbo.Laps_tbl
```

Siis kustutatakse laste tabelist kõik read kuid tabel jääb alles!

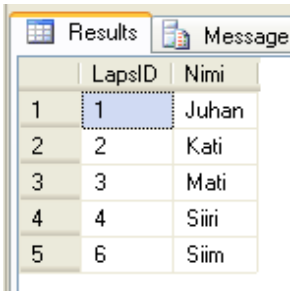
Kui soovime laste hulgast eemaldada Madise (kood 5) saame kirjutada järgmise SQL lause:

```
DELETE dbo.Laps_tbl
WHERE LapsID = 5
```

Enne

	LapsID	Nimi
1	1	Juhan
2	2	Kati
3	3	Mati
4	4	Siiri
5	5	Madis
6	6	Siim

Pärast



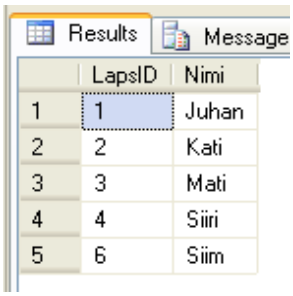
	LapsID	Nimi
1	1	Juhan
2	2	Kati
3	3	Mati
4	4	Siiri
5	6	Siim

Tulemusena paistab, et rida id-numbriga 5 andmete hulgas puudub.

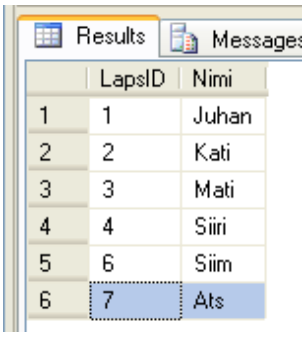
DELETE kohta kehtivad samad hoiatused, mis UPDATE juures. Kui WHERE-tingimus jääb määramata, siis on ühekorraga kõik andmed kadunud. DELETE juures pole vaja tulpade nimesid määrata, sest alati kustutatakse terve tabel korraga. UPDATE puhul tuleb aga iga tulba puhul öelda, millise nimega tulpa muudetakse ja milline on sealne uus väärtus.

Kui kustutamise teel on üks rida tabelist eemaldatud, siis sama id-d samasse tulpa ei anta enam kunagi välja. Isegi siis kui kustutasite kõige suurema IDga rea! Tuleb juurde uus laps, siis tema järjekorranumber suureneb endiselt – sõltumata asjaolust, et loetelus juba vaba koht leidub.

```
INSERT INTO dbo.Laps_tbl (nimi, pikkus, synniaasta)
VALUES ('Ats', 165, 1996)
```



	LapsID	Nimi
1	1	Juhan
2	2	Kati
3	3	Mati
4	4	Siiri
5	6	Siim



	LapsID	Nimi
1	1	Juhan
2	2	Kati
3	3	Mati
4	4	Siiri
5	6	Siim
6	7	Ats

Alati uus number on vajalik segaduste vältimiseks. Kui näiteks juhtuks, et ennist kuuendal kohal paiknenud Madisel oli trenniraha maksmata. ning nüüd uustulnukana saabunud Marile antaks välja number kuus, siis oleks oht, et Madise maksmata arved jõuaksid Marile – sellist muret aga ei pea üks infosüsteem tekitama. Kui igaühel oma järjekorranumber, siis jäävad sellised segadused olemata.

Harjutus (lihtsad päringud)

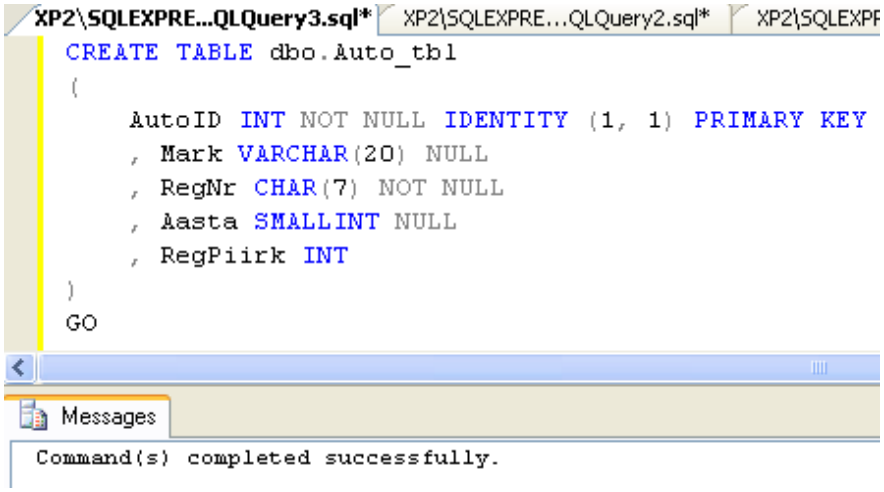
- Loo autode tabel, kus igaühe kohta on kirjas mark, registrinumber ja tootmisaasta ning registripiirkond
- Sisesta järgmised autod:

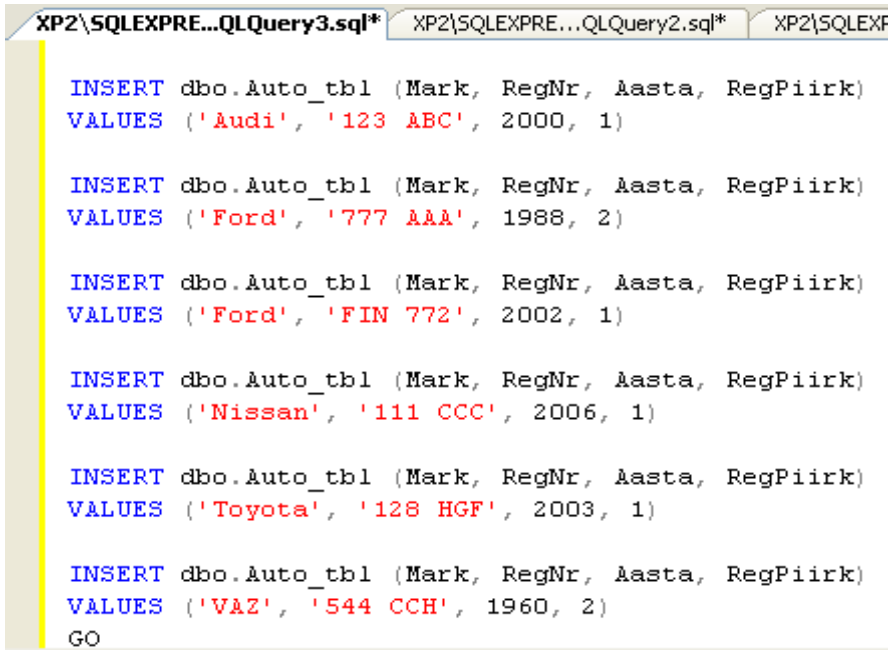
Mark	RegNr	Aasta	RegPiirk
Audi	123 ABC	2000	1
Ford	777 AAA	1988	2
Ford	FIN 772	2002	1
Nissan	111 CCC	2006	1
Toyota	128 HGF	2003	1
VAZ	544 CCH	1960	2

- Järjesta autod tootmisaasta järgi kahanevasse järjekorda
- Väljasta kõik erinevad margid
- Väljasta enne 1993. aastat toodetud autode registrinumbrid
- Väljasta enne 1993. aastat toodetud autode registrinumbrid tähestiku järjekorras
- Väljasta autode kõige varasem väljalaskeaasta (MIN)
- Muuda registrinumbrit autol, mille id on 3 (uus number 333 KKK)
- Kustuta auto id-ga 4
- Lisa uus masin nimekirja. Vaata tabeli sisu.

Mark	RegNr	Aasta	RegPiirk
Nissan	555 NNN	2007	2

Harjutuste vastused (lihtsad päringud)





XP2\SQLEXPRE...QLQuery3.sql* | XP2\SQLEXPRE...QLQuery2.sql*

```
SELECT AutoID, Mark, RegNr, Aasta, RegPiirk
FROM dbo.Auto_tbl
ORDER BY Aasta DESC
GO
```

Results | Messages

	AutoID	Mark	RegNr	Aasta	RegPiirk
1	4	Nissan	111 CCC	2006	1
2	5	Toyota	128 HGF	2003	1
3	3	Ford	FIN 772	2002	1
4	1	Audi	123 ABC	2000	1
5	2	Ford	777 AAA	1988	2
6	6	VAZ	544 CCH	1960	2

XP2\SQLEXPRE...QLQuery3.sql*

```
SELECT DISTINCT MARK
FROM dbo.Auto_tbl
GO
```

Results | Messages

	MARK
1	Audi
2	Ford
3	Nissan
4	Toyota
5	VAZ

XP2\SQLEXPRE...QLQuery3.sql*

```
SELECT RegNr
FROM dbo.Auto_tbl
WHERE Aasta < 1993
GO
```

Results | Messages

	RegNr
1	777 AAA
2	544 CCH

XP2\SQLEXPRE...QLQuery3.sql*

```
SELECT RegNr
FROM dbo.Auto_tbl
WHERE Aasta < 1993
ORDER BY RegNr
GO
```

Results | Messages

	RegNr
1	544 CCH
2	777 AAA

XP2\SQLEXPRE...QLQuery3.sql* | XP2\SQLEXPRI

```
SELECT MIN(Aasta) AS Varaseim
FROM dbo.Auto_tbl
GO
```

Results | Messages

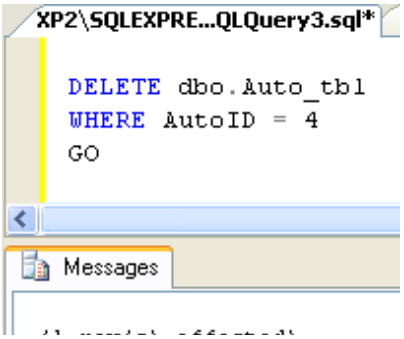
	Varaseim
1	1960

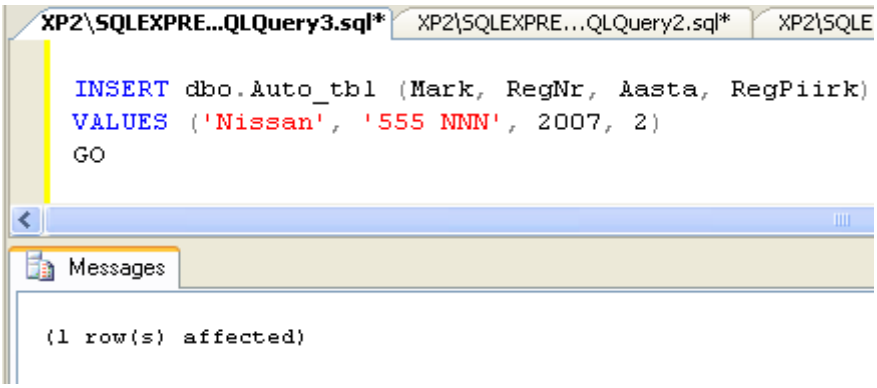
XP2\SQLEXPRE...QLQuery3.sql*

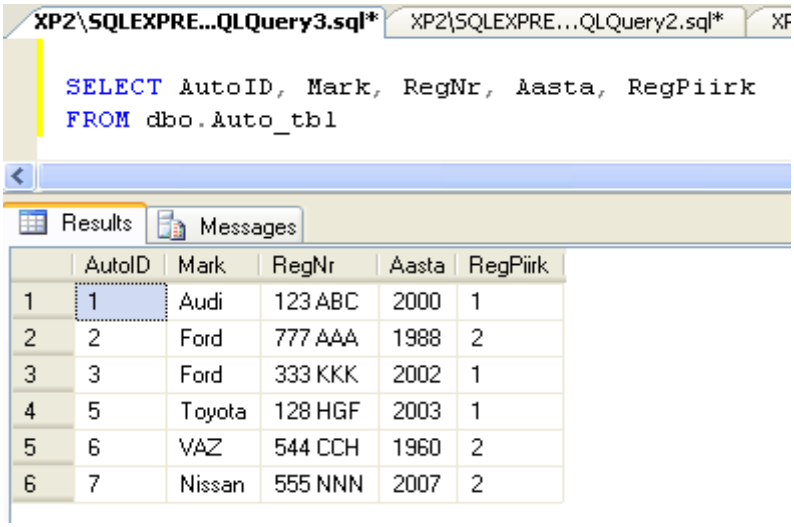
```
UPDATE dbo.Auto_tbl
SET RegNr = '333 KKK'
WHERE AutoID = 3
GO
```

Messages

```
(1 row(s) affected)
```







	AutoID	Mark	RegNr	Aasta	RegPiirk
1	1	Audi	123 ABC	2000	1
2	2	Ford	777 AAA	1988	2
3	3	Ford	333 KKK	2002	1
4	5	Toyota	128 HGF	2003	1
5	6	VAZ	544 CCH	1960	2
6	7	Nissan	555 NNN	2007	2

Tabelite vahelised seosed

Ühes andmebaasitabelis hoitakse üldjuhul ainult ühte liiki andmeid, mille kohta annab soovitavalt selge seletuse ka tabeli pealkiri. Kui on tunda, et lisanduvad andmed ei taha enam selle pealkirja või olemasolevate väljade peale ära mahtuda, siis enamasti on targem uus tabel teha. Tabelite rohkust ei pea pelgama. Pigem on kasulik teha mitu tabelit, kui ühte tabelisse suruda kokku mitmesuguseid väärtusi, mis sinna ei taha passida.

Samuti on mõistlik tabeleid lisada, kui paistab, et samu andmeid tuleks muidu lisada korduvalt. Ühelt poolt tekitab samade andmete korduv sisestamine ohu, et kusagil tehakse sisestamisel viga ning selle tulemusena näidatakse tulevikus kord õigeid, kord valesid

andmeid. Teiseks ühekordse sisestuse eeliseks on, et andmete muutumisel piisab muutusest vaid ühes kohas.

Enamasti on andmebaasides tabelid omavahel ühendatud. Siin näites koostame lemmikloomade tabeli, kus iga looma juures võib lisaks nimele olla pikkus, mass, sünniaeg. Samas igal lemmikloomal on peremees, kel on enesel nimi, isikukood ja muud inimesele omased tunnused. Kui püütaks kõik andmed ühte tabelisse kokku toppida, siis tuleks iga uue lemmiklooma puhul kirjutada uuesti ka tema peremehe andmed – muidu jääksid vastavad lahtrid lihtsalt tühjaks ja poleks kindel, kelle juurde loom kuulub. Et aga lemmiklooma peremeheks sobivad inimesed on eraldi tabelis juba kirjas, piisab, kui lisada iga looma juurde tema peremehe id-number ning ongi üheselt looma peremees määratud.

Järgnevalt siis lemmikloomade tabeli loomiskäsk. Igale tabelile iseloomulikult id-tulp isesuureneva primaarvõtmena, et oleks kindel järjekorranumber, mille kaudu loomale viidata. Looma nimi – tekst pikkusega kuni 50 sümbolit. Arv peremehe id-numbri meelespidamiseks. Ning lõpuks teade baasile

```
  FOREIGN KEY (peremehe_id) REFERENCES lapsed(id)
```

ehk siis võõrvõti (väärtus lemmikloomade väljast peremehe_id) näitab tabeli lapsed tulbale id.

Selle lause järgi oskab SQL Server kontrollida, et tabelisse lubatakse lisada vaid lemmikloomi, kelle peremehe_id näitab tabelis olemasolevale lapsele.

```
CREATE TABLE lemmikloomad(
  id INT identity PRIMARY KEY,
  loomanimi VARCHAR(50),
  peremehe_id INT,
  FOREIGN KEY (peremehe_id) REFERENCES lapsed(id)
)
```

Andmete lisamine INSERT lause abil nagu igal pool mujalgi. id-tulba väärtuse määrab programm ise, loomanimi ja peremehe identifikaator antakse ette lausega. Kui vastava järjekorranumbriga peremees on tabelis olemas, siis õnnestub kõik ilusti.

```
INSERT INTO lemmikloomad (loomanimi, peremehe_id)
VALUES ('Miisu', 5);
INSERT INTO lemmikloomad (loomanimi, peremehe_id)
VALUES ('Pauka', 7);
```

Madis ehk tegelane number kuus sai aga eespool tabelist kustutatud. Kui nüüd püütakse Muri kirja panna Madise koerana, siis annab arvuti vastu veateate.

```
INSERT INTO lemmikloomad (loomanimi, peremehe_id)
VALUES ('Muri', 6);
Msg 547, Level 16, State 0, Line 1
```

```
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK__lemmikloo__perem__117F9D94". The conflict occurred in database
"baas1", table "dbo.lapsed", column 'id'.
The statement has been terminated.
```

Öeldakse, et sisestatud võõrvõti ei sobi tabeli lapsed veeru id väärtusega. Ning Muri jääb sisestamata. Selle üle võib veenduda ka lemmikloomade tabelist andmeid küsides:

```
SELECT * FROM lemmikloomad
id  loomanimi  peremehe_id


    Miisu      5

1
    Pauka      7

2
```

Ehk siis said kirja Miisu ja Pauka, aga mitte Muri. Sest Muri puhul polnud võimalik üles märkida tabelis kirjas olevat peremeest.

Seoseid on võimalik luua ka juba valmis tabelite vahele. Selleks tuleb tabelit muuta ALTER TABLE käsuga. Näiteks kui soovime luua seose tabelite Laps_tbl ja Linn_tbl vahele võiksime kasutada järgmist konstruktsiooni:

```
ALTER TABLE dbo.Laps_tbl
ADD CONSTRAINT FK_Laps_Linn
      FOREIGN KEY ( SynniLinn )
      REFERENCES dbo.Linn_tbl (LinnID)
      ON UPDATE NO ACTION
      ON DELETE NO ACTION
```

Nagu näha saab välisvõtmele lisada juurde ka käitumisreeglid juhuks kui peatabelis võti muutub või kustub. Valikuid tegevusteks on neli:

- NO ACTION – ei tehta midagi st kui Laps tabelis on mõnel lapsel linn, mida üritatakse kustutada siis kustutamine katkestatakse ning antakse veateade

- CASCADE – antakse edasi e. kui kustutad/muuta linna kustuvad/muutuvad automaatselt (ILMA HOIATUSTETA) ka kõik selle linna lapsed

- SET NULL – kui kustub/muutub linn pannakse kõigi selle linna laste sünnilinnaks NULL e. määramata. See eeldab, et NULL väärtused on lubatud-

- SET DEFAULT – kui linn kustub/muutub siis taastatakse lastel vaikimisi määratud e DEFAULT linnad. Kui DEFAULT on määramata üritatakse panna NULL väärtust. Kui ka see ei õnnestu siis tegevus katkestatakse.

Tabelite ühendamine päringutes

Praegu on meil olemas kaks eraldi tabelit. Laste loetelu ning lemmikloomade loetelu, iga lemmiklooma juures on kirjas lapse id, kelle ülesandeks on vastava looma eest hoolt kanda. Soovides teada, kelle oma on Miisu, tuleb "käsitsi" uurides minna kõigepealt lemmikloomade tabelisse, otsida sealt üles Miisu peremehe_id ning siis minna selle arvu järgi laste tabelist peremehe nime ja muid andmeid otsima. Et selline tabelite ühendamine aga on andmebaaside juures sage ja hädavajalik, siis on ühendamise jaoks loodud omad käsklused.

Järgnevalt ühendatakse laste andmetabeli külge lemmikloomade andmetabel, kusjuures ridade kõrvutamise tingimuseks on, et lapse id-number ning lemmiklooma peremehe_id-number oleksid võrdsed. Tärn SELECT´i järel teatab, et näidataks kõiki võimalikke veerge. Ridadest on praeguse päringu puhul nähtavad ainult need lapsed, kel lemmikloom olemas. Ja kui mõnel lapsel oleks mitu lemmiklooma, siis näidataks ka selle lapse andmed mitmekordselt. Mitmekordsest näitamisest hoolimata talletatakse aga selle lapse andmeid baasis ikkagi ühekordselt. Nii et kui kellegi pikkus peaks muutuma, siis piisab selle märkimisest ühes kohas.

```
SELECT * FROM lapsed
INNER JOIN lemmikloomad
ON lemmikloomad.peremehe_id=lapsed.id

id  eesnimi  pikkus  synniaasta  id  loomanimi  peremehe_id


    Siiri    153     1996        1   Miisu      5

5
    Siim     163     1997        2   Pauka      7

7
```

INNER JOINi nimeline süntaks on levinud SQL Serveris. Sama toimingu kirjapanekuks aga on ka teine viis, mis töötab nii siin kui teiste SQL standardit arvestavate andmebaaside peal. FROM-sõna järgi kirjutatakse kõikide osalevate tabelite loetelu ning WHERE-tingimusega seatakse, millised read peavad omavahel võrdsed olema. Nagu näha, on tulemus eelmise päringuga võrreldes samasugune.

```
SELECT * FROM lapsed, lemmikloomad
WHERE lemmikloomad.peremehe_id=lapsed.id
```

```
id  eesnimi  pikkus  synniaasta  id  loomanimi  peremehe_id

    Siiri    153     1996        1   Miisu      5

5
    Siim     163     1997        2   Pauka      7

7
```

Tabeleid ühendades saab lihtsalt mitmest tabelist kokku ühe. Muud tingimused ja järjestamised käivad ikka samamoodi. Ehk siis, kui tahta panna andmed peremeeste nimede järgi tähestikulisse järjekorda, siis aitab endiselt ORDER BY eesnimi.

```
SELECT * FROM lapsed, lemmikloomad
WHERE lemmikloomad.peremehe_id=lapsed.id
ORDER BY eesnimi
id  eesnimi  pikkus  synniaasta  id  loomanimi  peremehe_id

    Siim     163     1997        2   Pauka      7

7
    Siiri    153     1996        1   Miisu      5

5
```

Tabelite järjekord päringus määrab ka nende järjekorra trükitavas vastuses. Kui andmete poole pöördutakse tulba nime järgi, pole sel erilist vahet. Kui aga kasutatakse tulba järjekorraumbrit, siis peab teadma, mitmendana milline tulp kus asetseb. Siin siis lemmikloomad ees ja lapsed järgi.

```
SELECT * FROM lemmikloomad
INNER JOIN lapsed
ON lemmikloomad.peremehe_id=lapsed.id
id  loomanimi  peremehe_id  id  eesnimi  pikkus  synniaasta

    Miisu      5            5   Siiri    153     1996

1
    Pauka      7            7   Siim     163     1997

2
```

Tahtes näha vaid osa tulpasid kõigi asemel, tuleb nende tulpade nimed ette lugeda. Ikka sarnaselt nagu ühestki tabelist tehtavate päringute korral.

```
SELECT eesnimi, loomanimi FROM lemmikloomad
INNER JOIN lapsed
ON lemmikloomad.peremehe_id=lapsed.id
Siiri Miisu
Siim  Pauka
```

Et INNER JOIN on SQL Serveri jaoks vaikimisi ühendusviis, võib sõna INNER ära jätta – päring töötab ikka samamoodi.

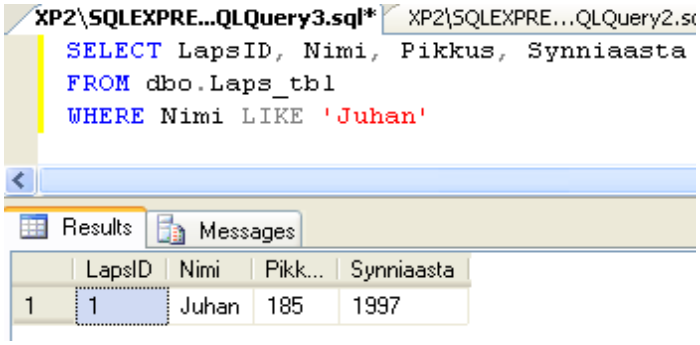
```
SELECT eesnimi, loomanimi FROM lemmikloomad
JOIN lapsed ON lemmikloomad.peremehe_id=lapsed.id
Siiri Miisu
Siim  Pauka
```

Edasijõudnutele

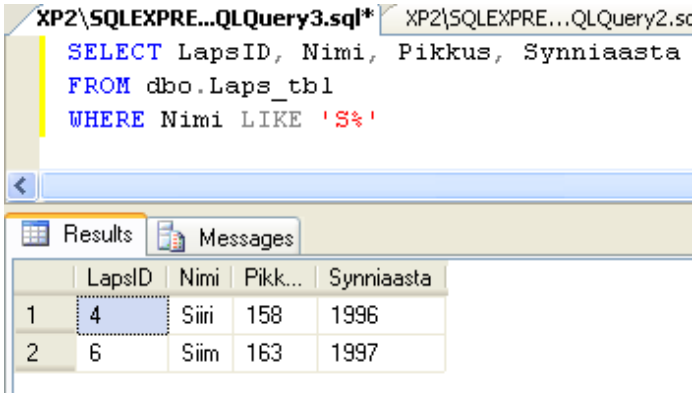
Pikemad päringud

LIKE

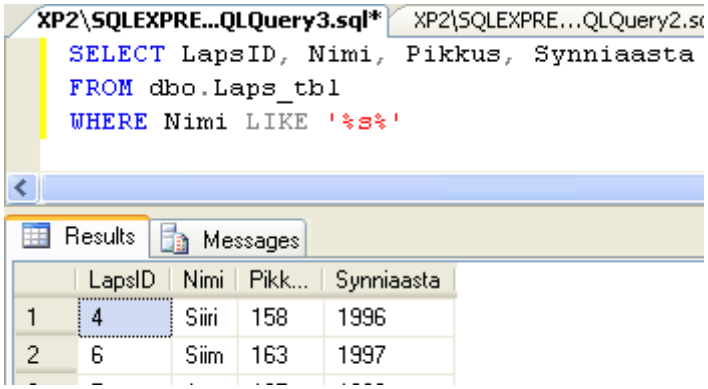
Tekstidest otsimise juures aitab sobivat vastet leida võrdlus LIKE. Kui sinna anda ette lihtsalt tekst, siis käitub LIKE võrdusmärgina:



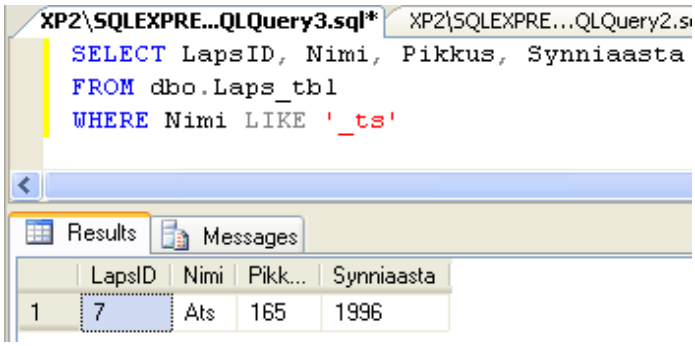
Käskluse põhivõlu seisneb aga võimaluses otsida metamärkide järgi. Alljoon _ tähistab ühte suvalist sümbolit ning protsendimärk % suvalist arvu suvalisi sümboleid. Näiteks S% vastab kõigile tekstidele, mis algavad S-iga.



Ning %s% vastab kõigile tekstidele, mis sisaldavad s-i.

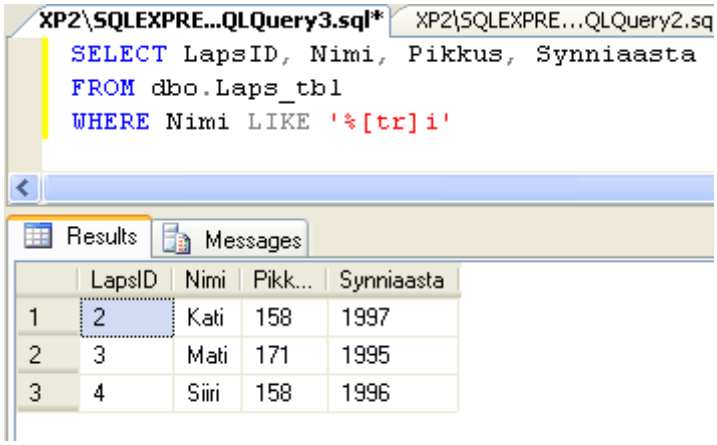


Alljoon, nagu öeldud, vastab vaid ühele sümbolile.



Nii et kui nimeks olnuks Pets, siis sinna '_ts' ei laiene.

Ette saab anda ka tähtede loetelu – selleks vajalikud kandilised sulud. '%[tr]i' tähendab, et alguses võivad olla suvalised sümbolid, siis peab tulema t või r ning lõppu i. Nagu näha – selliseid nimesid on meie loetelus päris mitu.



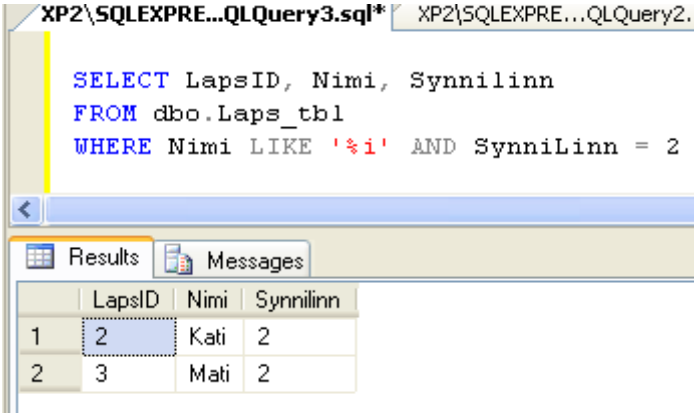
LIKE võtab arvutilt küllalt palju mõtlemisaega – seetõttu ei soovitata suuremate tabelite juures vastavat käsklust ilmaasjata pruukida. Samas aga on see küllalt mugav vahend sobivate nimede ja koodide leidmiseks, nii et mõõduka kasutamise juures on ta täiesti omal kohal.

Pikemate, nö täistekstide juures, on SQL-serveril omad käsud CONTAINS ning FREETEXT, mille abil võimalik omale sobivaid andmeid kätte saada.

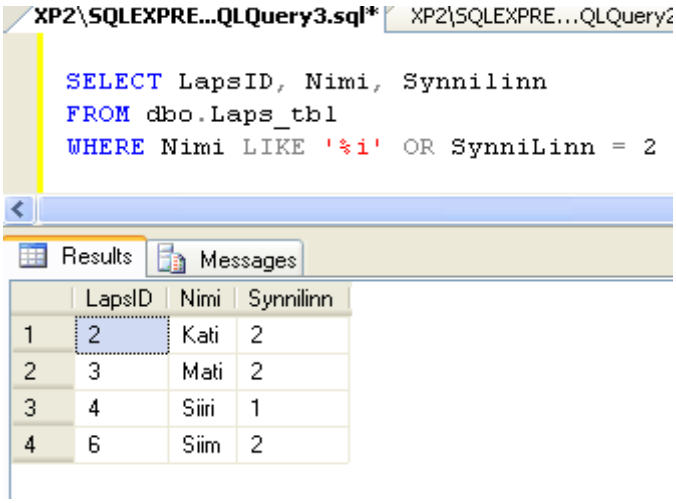
Kuna täisteksti otsinguid ei tee mitte SQL Server vaid Windowsi Indekseerimise ja otsimise teenus siis vajab nende funktsioonide kasutamine administraatori poolset andmete ettevalmistust ning hoolitust.

Tingimuste kombineerimine

Olemasolevaid tingimusi saab alati omavahel kombineerida. AND nõuab, et mõlemad tingimuse pooled oleksid täidetud, OR seevastu piirdub nõudega, et vähemalt üks tingimustest oleks tõene. Kõigepealt siis nimed, mis lõppevad i-ga ning sünnilinn on Tartu (kood 2)

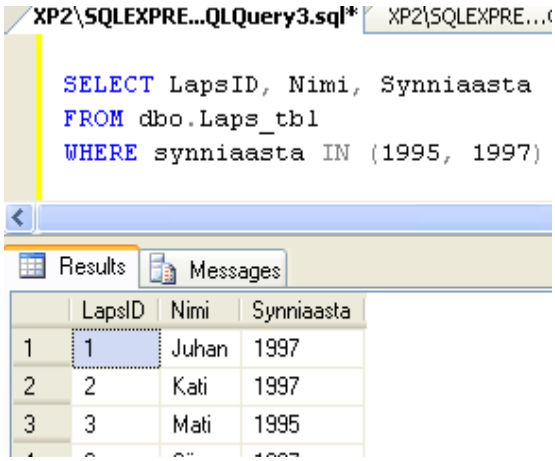


Edasi kõik i-lõpulised nimed pluss veel lisaks kõik, kes sündinud Tartus. Nimesid korduvalt siiski ei näidata. Kuigi Siiri ja Mari vastavad mõlemale tingimusele, on nad nimekirjas siiski ainult ühe korra.



IN

Tahtes ette anda lubatud väärtuste hulka, millele otsitav peab vastama, aitab käsklus IN. Järgnevalt siis tegelased, kelle sünniaastaks on kas 1995 või 1997.



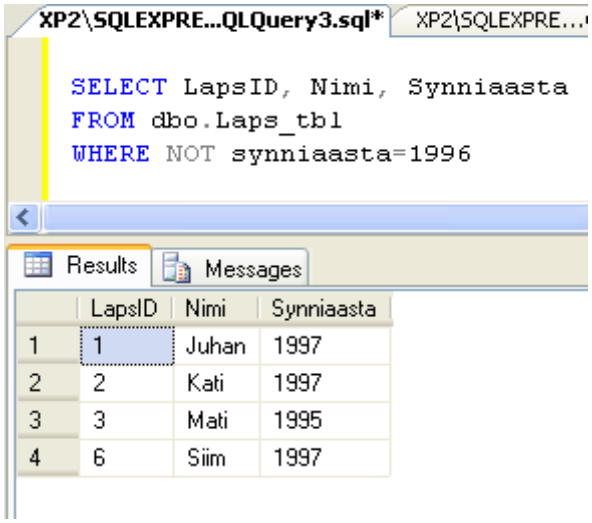
Sama tulemuse saab ka kombineerides OR operaatoriga otsesed võrdused:

```
SELECT LapsID, Nimi, Synniaasta
FROM dbo.Laps_tbl
WHERE synniaasta = 1995 OR synniaasta = 1997
```

Kuigi tulemus on sama ja SQL Server käsitleb neid päringuid ühtemoodi on IN operaatori kasutamine ülevaatlikum.

NOT

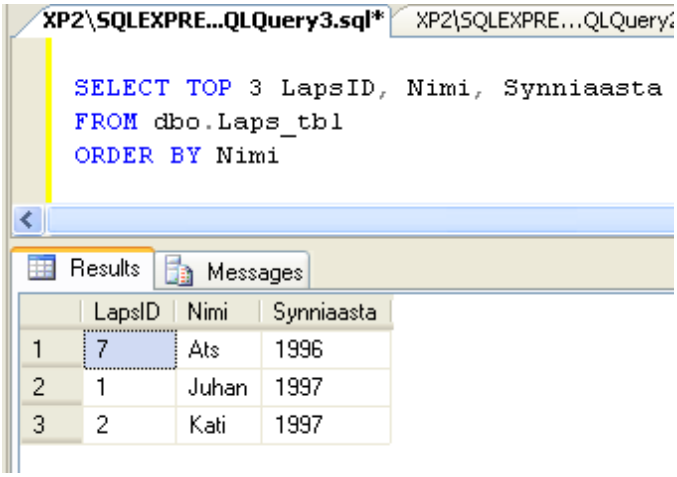
NOT pöörab tulemuse ümber. Ehk siis kõik need lapsed, kes ei ole sündinud aastal 1996.



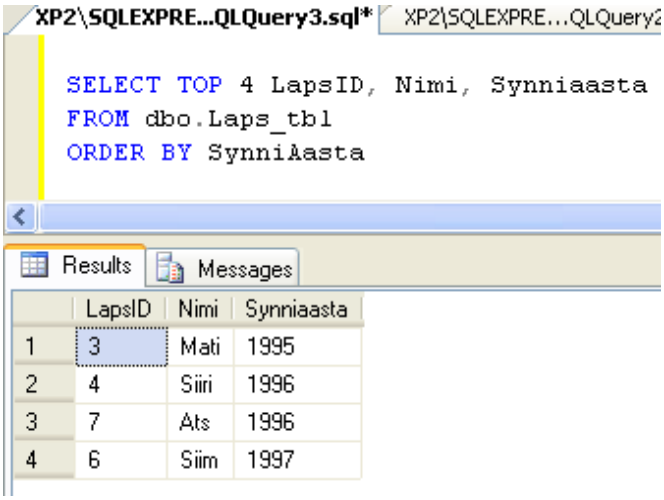
Võimaluse korral soovitatakse NOTi mitte pruukida, sest enamasti peab sel juhul andmebaasimootor vaatama läbi tabeli kõik read, mis on suurte andmemahtude juures küllalt suur töö. Aga kui muidu läbi ei saa, eks siis peab ikka selle sõna kirjutama.

TOP, päringu algusosa

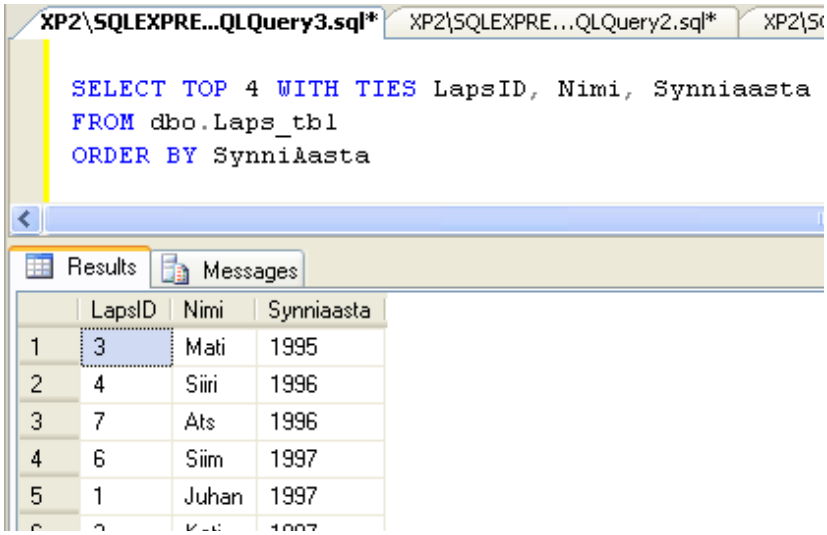
Lihtsalt andmetest ülevaate saamiseks ei ole vaja sageli kõike näha. Samuti, kui soovime viite kiiremat jooksjat või viite vanemat autot, siis on mugav, kui päring kohe annabki meile soovitu kätte, mitte ei pea hakkama ise pead vaevama, kuidas soovitud kohast andmeid võtma hakata. Laste tabelist tähestiku järjekorras kolm esimest nime näiteks saab kätte nii.



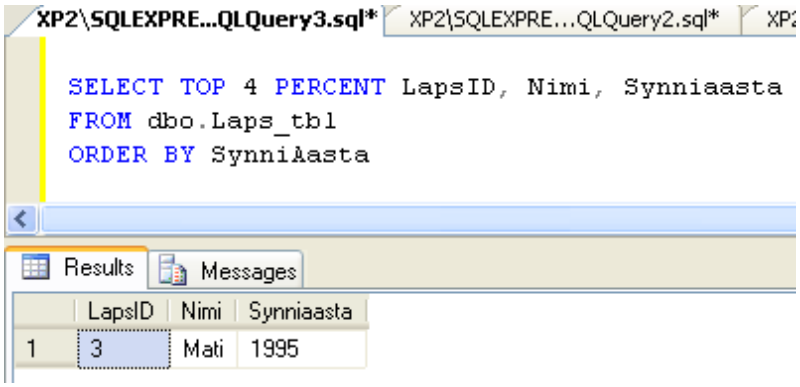
Tahtes saada neli vanemat last, tuleb andmed sorteerida sünniaasta järgi.



Tekib aga probleem: kuna andmed on salvestatud aasta täpsusega, siis võetakse 1996ndal aastal sündinutest lihtsalt juhuslik komplekt ning ülejäänud jäävad näitamata. Mõnikord pole sellest hullu – saadi juhuslikud tegelased kokku ja sobib küll. Teinekord aga võivad samade tunnustega osalejad porisema hakata, kui üks neist kaasa võeti ja teine mitte. Et saaks kõik ausalt kaasa, kes teistega võrdsed, selleks saab TOP käsklusele lisada WITH TIES. Nii võetakse siin näites vähemalt kolm. Ning kui jagub järjestatava tunnuse alusel viimasega võrdseid mahajääjaid, võetakse ka nemad kaasa.



Lisaks fikseeritud ridade arvule võime määrata ridade arvu ka proportsionaalselt kogu ridade hulgast e. kasutada protsenti. Näiteks võime välja tuua neli protsenti kõige vanemaid lapsi:



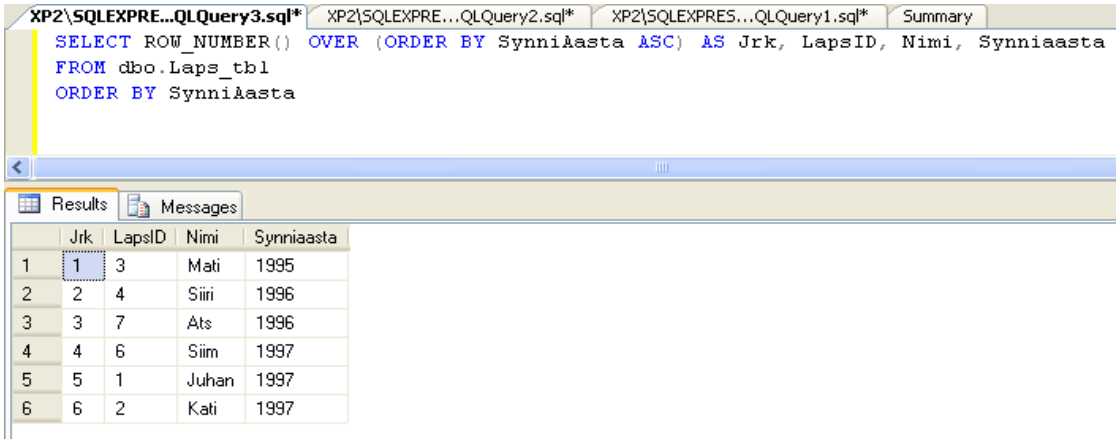
4 protsenti kuuest lapsest on küll suhteliselt tilluke arv. Aga et näidates ümardatakse arve ülespoole, siis näeme ikkagi vähemasti ühe lapse andmeid.

SQL Server 2005 täiendab TOP käsu süntaksid ühe väikese, kuid äärmiselt olulise võimalusega: nimelt on võimalik ridu piirava konstandi asemel kasutada ka muutujat või isegi alampäringut! See annab juurde väga palju uusi võimalusi päringute ja ka salvestatud protseduuride loomisel. Nii saab luua rakenduse, kus kasutaja ise ütleb, mitut rida ta soovib näha. Näiteks loome protseduuri, mis tagastab soovitud hulga vanimaid lapsi:

```
CREATE PROC VanimadLapsed_proc
      @LasteArv int
AS
      IF (@LasteArv >0)
SELECT TOP (@LasteArv) eesnimi, synniaasta
FROM lapsed
ORDER BY synniaasta
      ELSE
```

```
PRINT 'Lähteandmed päringu tegemiseks on vigased!'
```

Enne SQL 2005 puudus ka võimalus vahepealt valimiseks e. kui soovite tuua alates 3ndast kuni 5nda reani. SQL 2005 on tekitada tulemusse reanumbrid ning nende järgi ka filtreerida. Selleks saab kasutada ROW_NUMBER() funktsiooni.



	Jrk	LapsID	Nimi	Synniaasta
1	1	3	Mati	1995
2	2	4	Siiri	1996
3	3	7	Ats	1996
4	4	6	Siim	1997
5	5	1	Juhan	1997
6	6	2	Kati	1997

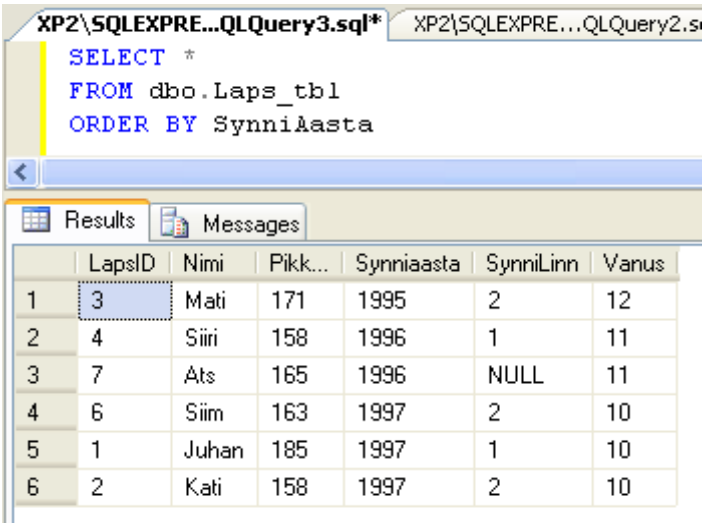
Süntaks on siis järgmine: ROW_NUMBER() OVER (partitsioon) st OVER märksõna järgi sulgudes tuleb öelda, mis moodi on read nummerdatud. Antud näites nummerdatakse sünniaastate järgi kasvavasse järjekorda.

Grupeerimine

Eelnevalt uurisime agregaatfunktsioone suurima, vähima, keskmise, summa ja koguse leidmiseks. Nad on kogu tabeli kohta head abilised. WHERE-tingimuse abil saab filtreerida sobiva tunnuse väärtuse alusel read välja ning siis nende põhjal kokkuvõtteid teha. Näiteks leida kõikide nende laste keskmise pikkuse, kes sündinud aastal 1996. Selgub aga, et käsklus lubab veelgi peenema statistika ette võtta.

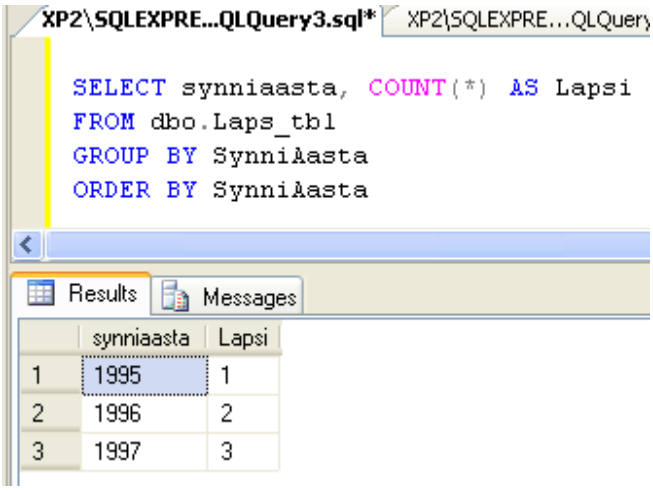
Seik autori oma kogemusest. Kord oli vaja ühele firmale teha veebipõhine rakendus komandeeringuaruannete sisestamiseks ning kokkuvõtete vaatamiseks. Iseenesest pealtnäha lihtne ülesanne: igaüks annab teada, kus ta käis, mida tegi ning kui palju raha kulus ja pärast loetakse nädalate, kuude, aastate ja isikute lõikes kõikvõimalikud andmed kokku. Muuhulgas oli vaja teada, mitu korda konkreetsel aastal millist linna on komandeeringu raames külastatud. SQL oli tuttav ligikaudu samapalju, kuivõrd lugeja kirjutises siiamaani jõudes. Et ka paarilt tuttavalt nõu küsimine ei aidanud edasi, tuli ise vastav programmike kirjutada. Pool päeva tööd, paar lehekülge koodi ning tulemus oli valmis ja sobis tööandjale. Suur oli aga üllatus, kui paar päeva hiljem SQLi manuaale uurides leidus võimalus seesama töö ühe suhteliselt lihtsa lausega kirja panna.

Nüüd siis mõned näited ja seletused, et siinse kirjutise lugejad ei peaks sama pikka ja okkalist teed läbi käima. Algul meeldetuletuseks laste andmed, et oleks näha, mida ja kuidas grupeeritakse.



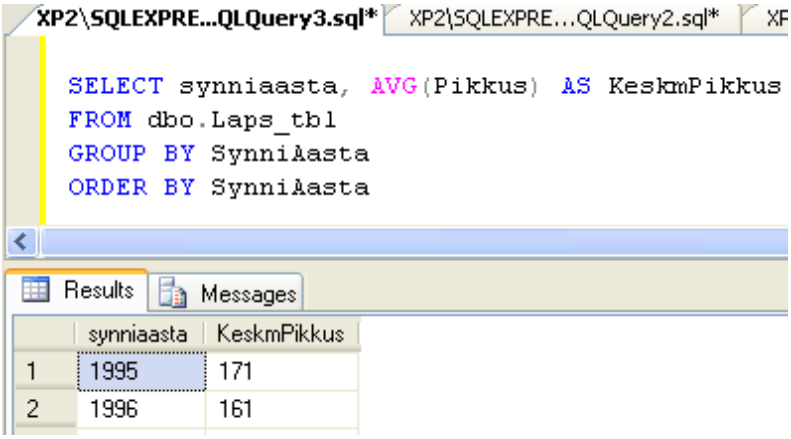
	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus
1	3	Mati	171	1995	2	12
2	4	Siiri	158	1996	1	11
3	7	Ats	165	1996	NULL	11
4	6	Siim	163	1997	2	10
5	1	Juhan	185	1997	1	10
6	2	Kati	158	1997	2	10

Tahtes iga aasta kohta teada, mitu last meie nimekirjast vastaval aastal sündinud on, aitab järgnev lause. COUNT(*) loeb kokku plokis olevad read. Et päringu lõpus on GROUP BY synniaasta, siis loetakse iga erinev sünniaasta omaette plokiks. Tahtes sünniaastat ka ennast näha, tuleb ka see SELECT´i järele tulpade loetellu kirjutada. Grupeerimisfunktsioonide puhul tohibki vastusesse küsida väärtusi vaid nendest tulpadest, mille järgi grupeeritakse. Muidu tekiks ju segadus, sest kui tahaks võtta väljundisse ka pikkust, aga iga sünniaasta juurde võib kuuluda lapsi ja seega ka pikkusi mitu, siis ei tuleks vastus tabeli kujuline ning seetõttu ei sobiks relatsioonilise ehk tabelitel põhineva andmebaasi juurde. Kui aga sünniaasta järele grupeeritakse ja viimane ka ilusti näha on – siis püsib kõik korras. Pigem tunduks imelik, kui näidataks küll loendamise tulemusi 1, 3 ja 3, aga poleks näha, millise aasta juurde milline arv käib.

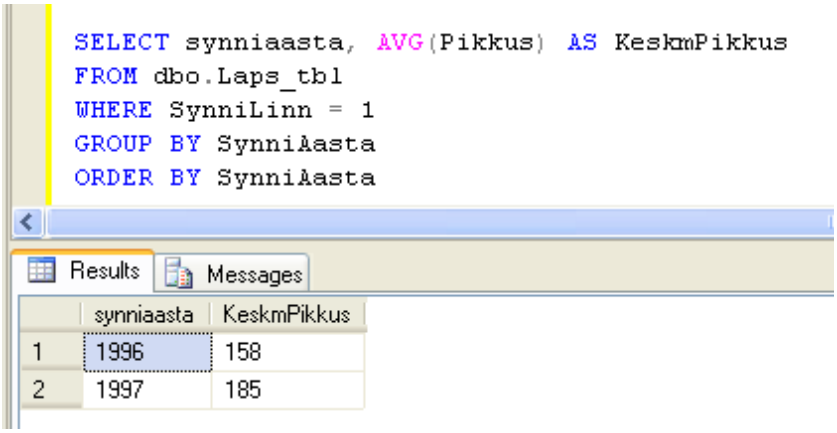


	synniaasta	Lapsi
1	1995	1
2	1996	2
3	1997	3

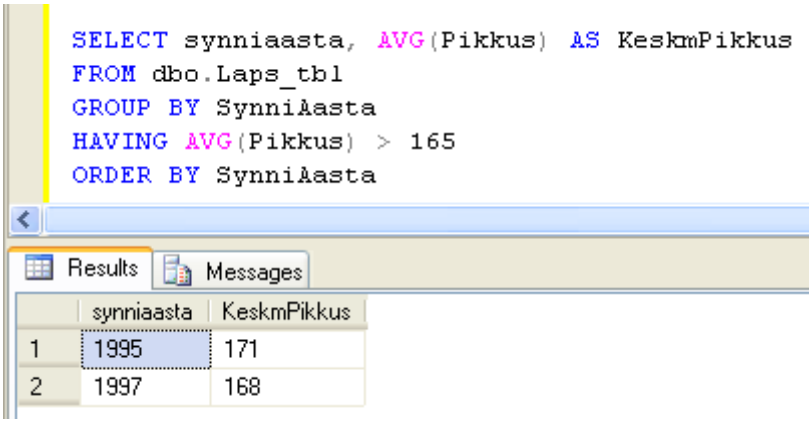
Sarnaselt nagu võib ridu kokku lugeda, saab ka teisi grupeerimisfunktsioone kasutada. Siin leitakse iga sünniaasta kohta sealsete laste keskmine pikkus.



Kui rakendame päringule piiranguid WHERE abil siis esmalt filtreeritakse lähteandmed ning alles peale seda hakatakse gruppe looma:



Kui soovime juba grupeeritud tulemusele piiranguid seada siis saame kasutada HAVING lauseosa. Näiteks soovime moodustada korvpallimeeskonda ning tahame teada, milliste vanusegruppide keskmine pikkus on üle 165 cm



ROLLUP, gruppide koondinfo

Grupeerimise juures on vahel võimalik ja vajalik päris mitmesuguseid andmeid koguda. Ja mõnikord on mugav, kui ei pea iga väärtuse jaoks omaette päringut tegema, vaid võib kõik andmed tulemusplokis ette võtta ja nendega toimetama asuda. Lihtsama näite puhul loendatakse lapsi aastate kaupa ning lõpuks võetakse kokku, palju neid üldse nimekirjas oli. Nagu alt näha – 6. Koguhulga juures pannakse sünniaasta kohale NULL, sest see ei käi enam mitte ühe konkreetse sünniaasta kohta, vaid kõigi peale kokku. Sellise lisarea annab käskluse osa WITH ROLLUP.

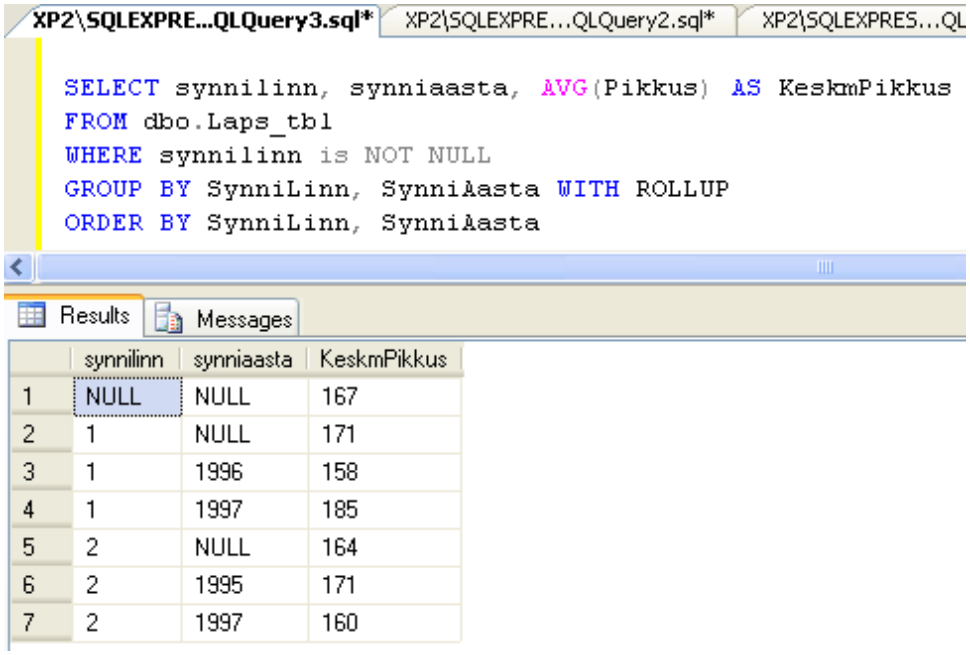
```
SELECT synniaasta, COUNT(*) AS Lapsi
FROM dbo.Laps_tbl
GROUP BY SynniAasta WITH ROLLUP
ORDER BY SynniAasta
```

Results | Messages

	synniaasta	Lapsi
1	NULL	6
2	1995	1
3	1996	2
4	1997	3

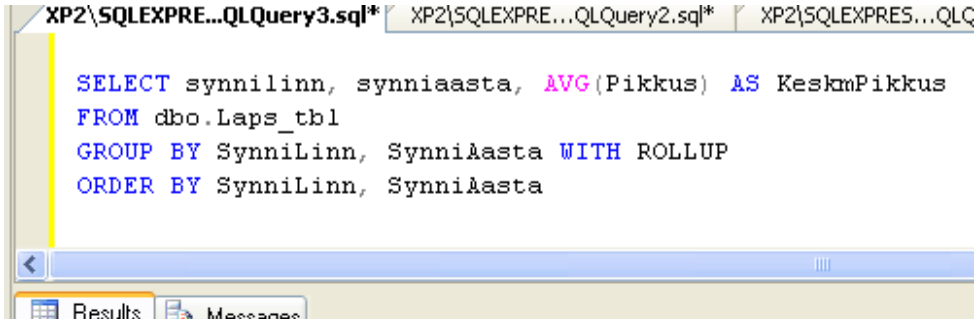
Kui grupeeritavaid tulpasid on rohkem, siis saab ka sellist lisastatistikat rohkem välja lugeda. Järgnevas näites grupeeriti lapsed sünnilinna ja sünniaasta järgi. See tähendab, et ühte gruppi sattunuksid nad vaid juhul, kui nad sündinuksid samal aastal ja oleksid ühepikkused. Iga muu kombinatsioon annab uue grupi. Nii see loend siis ka tuleb, kui algusest lugema hakata.

Kõigepealt esimeses reas teatatakse kõigi laste keskmine pikkus 167. Seejärel järgmises reas on esimese linna e. Tallinna laste keskmine pikkus 171. Seejärel tulevad keskmised pikkused Tallinna lastel, kes sündinud aastatel 1996 ja 1997. Viiendas reas on Tartu laste keskmine pikkus ning see järel aastatel 1995 ja 1997 sündinud Tartu laste keskmised pikkused.



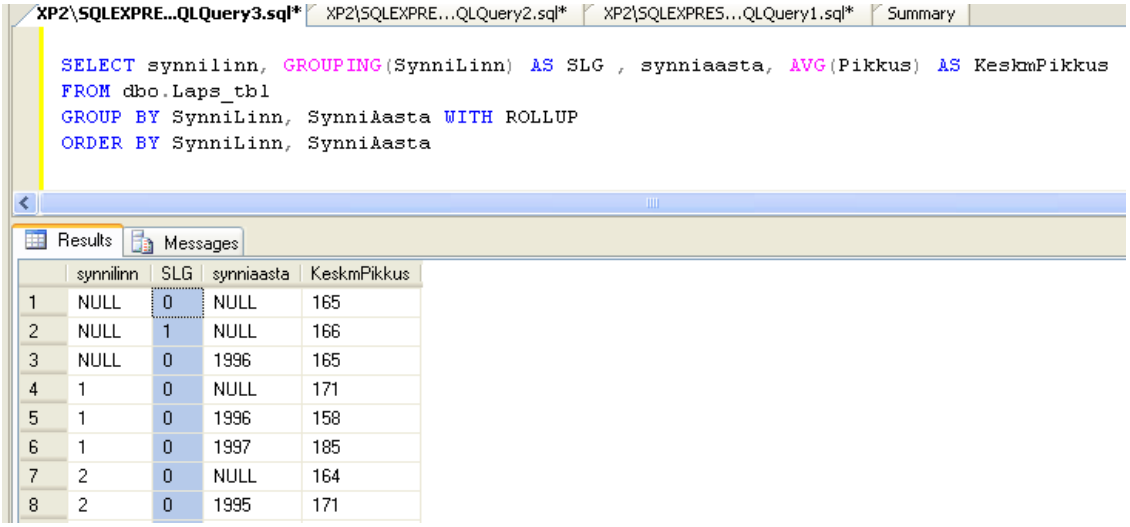
	synnilinn	synniaasta	KeskmPikkus
1	NULL	NULL	167
2	1	NULL	171
3	1	1996	158
4	1	1997	185
5	2	NULL	164
6	2	1995	171
7	2	1997	160

Nagu näha tähistatakse üldkokkuvõtted määramata e. NULL väärtusega kokku võetud väljal. Võib tekkida olukord kus kokkuvõetav väli ise võib sisaldada määramata väärtuseid. Näiteks meie tabelis on ühel lapsel sünnilinn teadmata. Kui nüüd leida grupid saame tulemuseks kaks väga sarnast rida:



	synnilinn	synniaasta	KeskmPikkus
1	NULL	NULL	165
2	NULL	NULL	166
3	NULL	1996	165
4	1	NULL	171
5	1	1996	158
6	1	1997	185
7	2	NULL	164
8	2	1995	171
9	2	1997	160

Tekib küsimus kus on kõigi laste keskmine ning kus on teadmata sünnilinnaga laste keskmine pikkus. Selle probleemi lahendamiseks saame probleemsele väljale rakendada GROUPING funktsiooni:

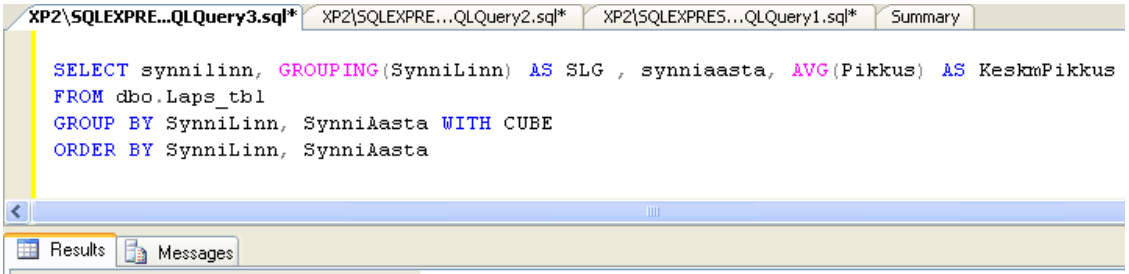


	synnilinn	SLG	synniaasta	KeskmPikkus
1	NULL	0	NULL	165
2	NULL	1	NULL	166
3	NULL	0	1996	165
4	1	0	NULL	171
5	1	0	1996	158
6	1	0	1997	185
7	2	0	NULL	164
8	2	0	1995	171

GROUPING funktsioon tekitab meile veeru, kus on 1 juhul kui on tegemist üldkokkuvõttega ning vastasel juhul on väärtus 0. Seega saame teada, et lastel kelle sünnilinna me ei tea on keskmine pikkus 165, kõigi laste keskmine pikkus on 166 ning lastel, kelle sünnilinna me ei tea ja sünniaasta on 1996 on keskmine pikkus 165.

CUBE, täiendatud koondinfo

Kui eelmises päringus olnud WITH ROLLUP asendada reaga WITH CUBE, siis tehakse grupeerimist kaks korda vastupidistes suundades ning näidatakse tulemuseks neid kahte tulemust ühendatuna. Seega saame lisaks teada, et 1995 sündinute keskmine pikkus on 171, 1996 sündinutel 161 ning 1997 sündinutel 168.



	synnilinn	SLG	synniaasta	KeskmPikkus
1	NULL	0	NULL	165
2	NULL	1	NULL	166
3	NULL	1	1995	171
4	NULL	1	1996	161
5	NULL	0	1996	165
6	NULL	1	1997	168
7	1	0	NULL	171
8	1	0	1996	158
9	1	0	1997	185
10	2	0	NULL	164
11	2	0	1995	171
12	2	0	1997	160

Et ühe sentimeetri kaupa grupeering on nii väikese inimeste arvu puhul ilmselt liiast, võib võtta inimeste jaotuse mõnevõrra suurema piirkonna ehk detsimeetri järgi. Avaldis pikkus/10 annab täisarvude puhul jagatise täisosa. Ehk siis 157/10 annab tulemuseks 15 ja 163/10 tuleb 16. Selliselt saab lapsed 10 sentimeetri kaupa gruppidesse jagada ning grupi andmetel on juba mõnevõrra mõistlikum sisu. Et väljatrükil poleks näha mitte 15 ja 16, vaid 150 ja 160, selleks korrutati SELECT real täisarvuks muutunud jagatis uuesti kümnega. Saadud tulemustest võib välja lugeda, et 1996ndal sündinute hulgas on kaks last 150ndates ning üks 160ndates. Ning kõigi aastate peale kokku on 4 inimest 150 ja 160 vahel ning 3 inimest 160 ja 170 vahel.

Harjutused (pikemad päringud)

Loo autode tabel, kus on iga masina kohta kirjas mark, registreerimisnumber ja väljalaskeaasta

Trüki välja kõik autod, mille registreerimismärk sisaldab A-d

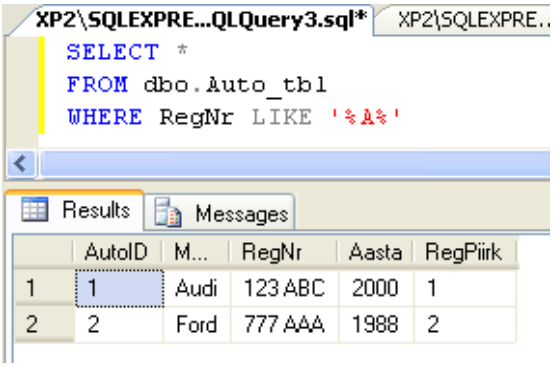
Väljasta iga margi kohta, mitu eksemplari seda on.

Väljasta iga margi ja väljalaskeaasta komplekti kohta, mitu seda on.

Väljasta iga margi kohta keskmine väljalaskeaasta

Väljasta iga margi kohta suurima ja vähima väljalaskeaasta vahe.

Katseta ROLLUP ja CUBE lisainfo võimalusi eelmiste päringute juures



```
SELECT Mark, COUNT(*) AS Arv
FROM dbo.Auto_tbl
GROUP BY Mark
```

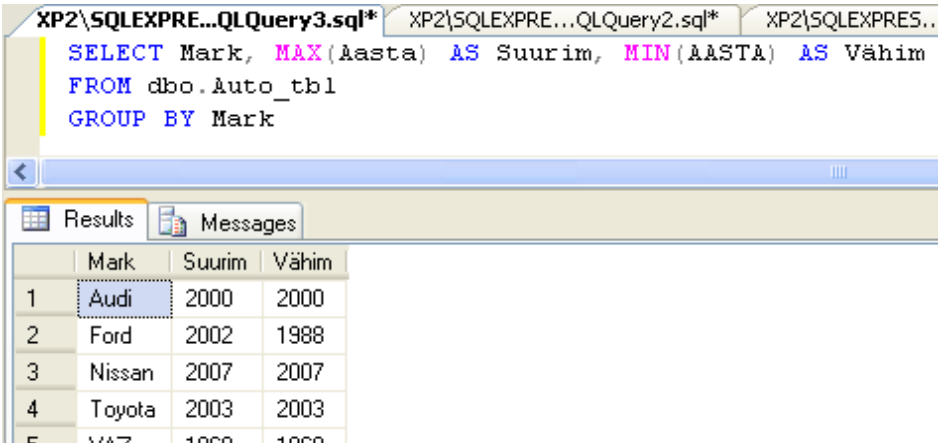
	Mark	Arv
1	Audi	1
2	Ford	2
3	Nissan	1
4	Toyota	1

```
SELECT Mark, Aasta, COUNT(*) AS Arv
FROM dbo.Auto_tbl
GROUP BY Mark, Aasta
```

	Mark	Aasta	Arv
1	VAZ	1960	1
2	Ford	1988	1
3	Audi	2000	1
4	Ford	2002	1
5	Toyota	2003	1
6	Nissan	2007	1

```
SELECT Mark, AVG(Aasta) AS Arv
FROM dbo.Auto_tbl
GROUP BY Mark
```

	Mark	Arv
1	Audi	2000
2	Ford	1995
3	Nissan	2007
4	Toyota	2003
5	VAZ	1960



Keerukamad seosed tabelite vahel

LEFT ja RIGHT JOIN

Kõige tavalisema ühendamise puhul saime kahest tabelist kätte need read, mis mõlemas olemas olid. Ehk siis loetelus olid vaid lemmikloomaga lapsed ning samuti igas loetelus olnud lemmikloomal oli kõrval peremees. Et praeguses näites ei lubata peremeheta lemmikloomi tabelisse lisada, siis jääb ära ka võimalus ülejäänud lemmikloomade näitamiseks. Küll aga võib mõnikord olla soov näha ka neid lapsi, kel pole oma koera või kassi. Ning samas loomaomanikele panna kõrvale ka loomade andmed. Sellise tööga saab hakkama LEFT JOIN. Loetelus esimesena olnud tabelist ehk vasakust näidatakse välja kõik read. Paremast aga vaid need, kus seos vasaku tabeliga olemas. Kel looma pole, sel tuleb loomanime kohale tühiväärtus NULL.

```
SELECT eesnimi, loomanimi FROM lapsed
LEFT JOIN lemmikloomad
ON lemmikloomad.peremehe_id=lapsed.id
```

eesnimi	loomanimi
Juku	NULL
Kati	NULL
Mati	NULL
Ats	NULL
Siiri	Miisu

Siim Pauka

Mari NULL

Sarnaselt töötab RIGHT JOIN. Ainult selle vahega, et näidatakse välja kõik parempoolses tabelis olevad andmed. Kui mõnele reale ei vasta kirjet vasakpoolses tabelis, siis näidatakse selle koha peal vasakpoolse tabeli väljade kohal NULL. Et siin aga on igal loomal peremees, siis tühiväärtusi ei teki.

```
SELECT eesnimi, loomanimi FROM lapsed
RIGHT JOIN lemmikloomad
ON lemmikloomad.peremehe_id=lapsed.id
Siiri Miisu
Siim  Pauka
```

LEFT JOINi ja RIGHT JOINi pikem kuju on LEFT OUTER JOIN ning RIGHT OUTER JOIN. Aga nagu näha, tulemus jääb samaks.

```
SELECT eesnimi, loomanimi FROM lapsed
RIGHT OUTER JOIN lemmikloomad
ON lemmikloomad.peremehe_id=lapsed.id
Siiri Miisu
Siim  Pauka
```

Nende ühendamiste puhul peab kindlasti silmas pidama tabelite järjekorda. Kui panna lemmikloomad vasakuks tabeliks ja lapsed parempoolseks tabeliks ning ühendamisel kasutada RIGHT JOINi ning tulbad nime järgi välja kutsuda, siis on tulemus sama, kui oleks kasutanud tabeleid teises järjekorras ning ühendamiseks LEFT JOINi.

```
SELECT eesnimi, loomanimi FROM lemmikloomad
RIGHT JOIN lapsed
ON lemmikloomad.peremehe_id=lapsed.id
```

eesnimi loomanimi

Juku NULL

Kati NULL

Mati NULL

Ats NULL

Siiri Miisu

Siim Pauka

Mari NULL

CROSS JOIN

Kõikide võimalike kombinatsioonide väljatrükiks sobib CROSS JOIN. Sel juhul võtmeid tabelite ühendamiseks ei kasutata, vaid trükitakse välja kõik võimalikud kombinatsioonid, kuidas esimese tabeli read saavad olla ühendatud teise tabeli ridadega. Ehk siis siin näites pakutakse välja kõik kombinatsioonid, milline laps saab millise lemmikloomaga koos olla.

```
SELECT eesnimi, loomanimi FROM lemmikloomad
CROSS JOIN lapsed
```

```
Juku  Miisu
Kati  Miisu
Mati  Miisu
Ats   Miisu
Siiri Miisu
Siim  Miisu
Mari  Miisu
Juku  Pauka
Kati  Pauka
Mati  Pauka
Ats   Pauka
Siiri Pauka
Siim  Pauka
Mari  Pauka
```

Eks sellist segapudru läheb suhteliselt harvem vaja, aga ilus on vaadata, kes võib kellega koos olla. Samuti sobib CROSS JOIN olukordade jaoks, kui tahetakse kõikide võimalike variantide hulgast sobivat välja otsida. Näiteks soovitakse otsida kombinatsioonid, kus lapse ja looma nimed algavad sama tähega, või siis on nad sündinud samas kuus. Siinse näite puhul on tingimused pastakast välja imetud, aga mõne tutvumisõhtu puhul või laborikatsete juures võivad sellised valikud täiesti omal kohal olla.

CROSS JOINiga sama tulemuse annab, kui päringusse kirjutada lihtsalt tabelite nimed ilma täiendavaid tingimusi seadmata.

```
SELECT eesnimi, loomanimi FROM lemmikloomad, lapsed
```

```
Juku  Miisu
Kati  Miisu
Mati  Miisu
Ats   Miisu
Siiri Miisu
```

```
Siim  Miisu
Mari  Miisu
Juku  Pauka
Kati  Pauka
Mati  Pauka
Ats   Pauka
Siiri Pauka
Siim  Pauka
Mari  Pauka
```

Seos sama tabeliga

Esimese hooga võib tunduda imelik, miks peaks olema vaja siduda tabelit iseenesega. Aga rakendusi kirjutades tekib selliseid seostamiskohti üllatavalt palju. Näiteks kui kataloogid on kataloogipuus, siis seda struktuuri saab tabelisse salvestada nii, et iga kataloogi puhul kirjutatakse eraldi tulpa tema ülemkataloogi ID. Ning juurkataloogi puhul see arv näitab iseenesele või ei näita kuhugi. Samuti foorumi kirjade puhul, kui tahetakse meeles pidada, milline kiri millisele vastab. Siin aga vaatame, kuidas seos sama tabeliga toimub sünniaastate kaudu. Esialgu koostatakse päring, kus näidatakse kõikide laste paarid nendega samal aastal sündinud lastega. Et saaks tabelit iseenesega seostada, tuleb tabelist teha päringu ajaks kaks koopiat. Nii nagu sai päringus tulpasid ümber nimetada, nii saab ümber nimetada ka tabeleid.

```
SELECT * FROM lapsed as tabel1, lapsed as tabel2
```

ütleb, et võta tabel lapsed kõigepealt märksõna all tabel1 ning seejärel tabel lapsed ka märksõna all tabel2. Edasi juba võib need tabelid tingimus(t)e abil kokku siduda, sest muidu näidataks kõikide ridade omavahelised võimalikud kombinatsioonid. Et siin aga soovime paare vaid sünniaastate kaupa, siis nõuame, et eri tabeli ridade kõrvuti panekuks peavad nende laste sünniaastad kattuma.

```
SELECT * FROM lapsed as tabel1, lapsed as tabel2
WHERE tabel1.synniaasta=tabel2.synniaasta
```

id	eesnimi	pikkus	synniaasta	id	eesnimi	pikkus	synniaasta
1	Juku	155	1997	1	Juku	155	1997
2	Kati	158	1997	1	Juku	155	1997
7	Siim	163	1997	1	Juku	155	1997
1	Juku	155	1997	2	Kati	158	1997
2	Kati	158	1997	2	Kati	158	1997

7	Siim	163	1997	2	Kati	158	1997
3	Mati	164	1995	3	Mati	164	1995
4	Ats	165	1996	4	Ats	165	1996
5	Siiri	153	1996	4	Ats	165	1996
8	Mari	158	1996	4	Ats	165	1996
4	Ats	165	1996	5	Siiri	153	1996
5	Siiri	153	1996	5	Siiri	153	1996
8	Mari	158	1996	5	Siiri	153	1996
1	Juku	155	1997	7	Siim	163	1997
2	Kati	158	1997	7	Siim	163	1997
7	Siim	163	1997	7	Siim	163	1997
4	Ats	165	1996	8	Mari	158	1996
5	Siiri	153	1996	8	Mari	158	1996
8	Mari	158	1996	8	Mari	158	1996

Üllatusena avastame, et Juku paariliseks on pandud ka Juku ise. Ning paarina on olemas nii Juku Katiga kui Kati Jukuga. Selliseid anomaaliaid saab tingimuste täpsustamisega vähendada või kaotada.

Kui soovida, et sama isik ei oleks iseenesega kõrvuti, siis aitab tingimus, et kõrvuti seatud tabelikoopiate id-numbrite väärtused ei oleks võrdsed. Kui soovida, et sama paari korduvalt ei näidataks, siis võib seada näiteks tingimuse, et teisest tabelist tuleva inimese id-number oleks suurem kui esimesest tabelist tulev id-number. Sellisel juhul jääb igast paarist alles vaid üks väljatrükk – selline, mis vastab tingimustele.

Tahame ainult ühe konkreetse isiku eakaaslasi kätte saada, võib tema eraldi ära määrata. Kindlam oleks küll id kaudu, sest mitme Juku puhul võivad tekkida segadused. Meil aga on vaid üks Juku, seetõttu on loota, et vastus sobib ning tema eakaaslasteks on siin tabelis vaid Kati ja Siim.

```
SELECT tabel2.eesnimi from lapsed as tabel1, lapsed as tabel2
WHERE tabel1.synniaasta=tabel2.synniaasta
AND tabel1.id<>tabel2.id and tabel1.eesnimi='Juku'
```

```
Kati
Siim
```

Päringutulemuste ühendamine

Vahel on mugav, kui päringuga saab soovitud tulemuse võimalikult täpselt ette anda – et hiljem poleks muretsemist, kuidas üksikutest lõikudest tervet vastust kokku panna. Üheks mooduseks on päringutulemuste ühend – UNION. Tahtes teha nimekaarte kõigile lastele ja lemmikloomadele, võib nende nimed kokku liita.

```
SELECT eesnimi FROM lapsed
UNION
SELECT loomanimi FROM lemmikloomad
```

Vastuses oleva tulba nimi võetakse esimese päringu järgi. Ja ongi kõik erinevad nimed siin.

```
eesnimi
Ats
Juku
Kati
Mari
Mati
Miisu
Pauka
Siim
Siiri
```

Samuti võib mingil põhjusel olla soov välja tuua kõik tabelis leiduvad arvud. Kaks päringut ühtejärge, UNION vahele ning loetelu ongi käes.

```
SELECT pikkus FROM lapsed
UNION
SELECT synniaasta FROM lapsed
```

```
pikkus
153
155
158
163
164
165
1995
1996
```

```
1997
```

Nagu näidete põhjal aimata sai, lähevad UNION käsu põhjal korduvad väärtused kaduma. Tegemist matemaatilises mõttes hulgatehtega, mille tulemusena jäetakse alles vaid erinevad väärtused. Tahtes, et ka kõik korduvad arvud või sõnad näha oleksid, selleks sobib kahe päringu vahele panna UNION ALL. Nii näeme mõlema päringu tulemuste summat tervikuna.

```
SELECT pikkus as arvud FROM lapsed
UNION ALL
SELECT synniaasta FROM lapsed
ORDER BY arvud
```

```
arvud
153
155
158
158
163
164
165
1995
1996
1996
1996
1997
1997
1997
```

Harjutused (tabelite ühendamine)

Ühenda UNION ALL abil laste sünniaastad ja autode väljalaskeaastad

Lisa seos Linna_tbl ning Auto_tbl vahele

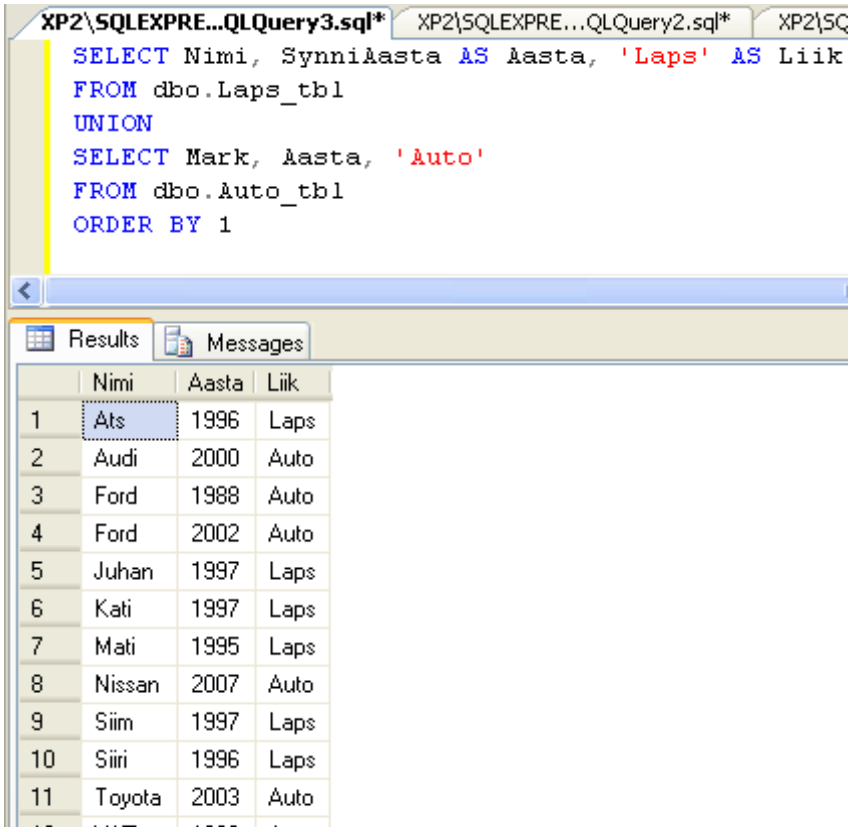
Väljasta kõik autod koos Linna nimetustega, kus nad registreeritud

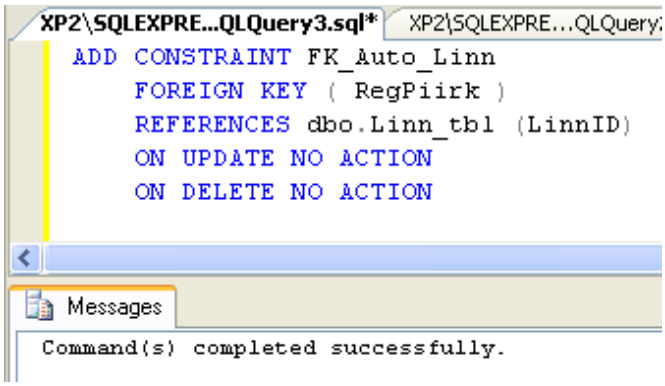
Väljasta autode arv grupeerituna Linnade kaupa

Lisa Linnade tabelisse uus linn

Väljasta LEFT JOINi abil kõik linnad koos neis registreeritud autodega. Registreeritud autodeta maakonnast väljasta vaid nimi.

Kingi igale lapsele auto, mille margiks on FORD, väljalaske aasta on lapse sünniaasta ning numbrimärk moodustub lapse pikkus + esimesed 3 tähte lapse nimest.


XP2\SQLEXPRE...QLQuery3.sql*
XP2\SQLEXPRE...QLQuery2.sql*
SELECT Nimi, SynniAasta AS Aasta, 'Laps' AS Liik
FROM dbo.Laps_tbl
UNION
SELECT Mark, Aasta, 'Auto'
FROM dbo.Auto_tbl
ORDER BY 1
Results
Messages
Nimi Aasta Liik
1 Ats 1996 Laps
2 Audi 2000 Auto
3 Ford 1988 Auto
4 Ford 2002 Auto
5 Juhan 1997 Laps
6 Kati 1997 Laps
7 Mati 1995 Laps
8 Nissan 2007 Auto
9 Siim 1997 Laps
10 Siiri 1996 Laps
11 Toyota 2003 Auto


XP2\SQLEXPRE...QLQuery3.sql*
ADD CONSTRAINT FK_Auto_Linn
FOREIGN KEY (RegPiirk)
REFERENCES dbo.Linn_tbl (LinnID)
ON UPDATE NO ACTION
ON DELETE NO ACTION
Messages
Command(s) completed successfully.

XP2\SQLEXPRE...QLQuery3.sql* | XP2\SQLEXPRE...QLQuery2.sql* | XP2\SQLEXPRES...QLQuer

```
SELECT L.Nimi AS Linn, A.AutoID, A.Mark, A.RegNr, A.Aasta
FROM dbo.Auto_tbl AS A
    INNER JOIN dbo.Linn_tbl AS L ON A.RegPiirk = L.LinnID
```

Results | Messages

	Linn	AutoID	Mark	RegNr	Aasta
1	Tallinn	1	Audi	123 ABC	2000
2	Tartu	2	Ford	777 AAA	1988
3	Tallinn	3	Ford	333 KKK	2002
4	Tallinn	5	Toyota	128 HGF	2003
5	Tartu	6	VAZ	544 CCH	1960
6	Tartu	7	Nissan	555 NNN	2007

XP2\SQLEXPRE...QLQuery3.sql* | XP2\SQL

```
INSERT dbo.Linn_tbl (Nimi)
VALUES ('Paide')
```

Messages

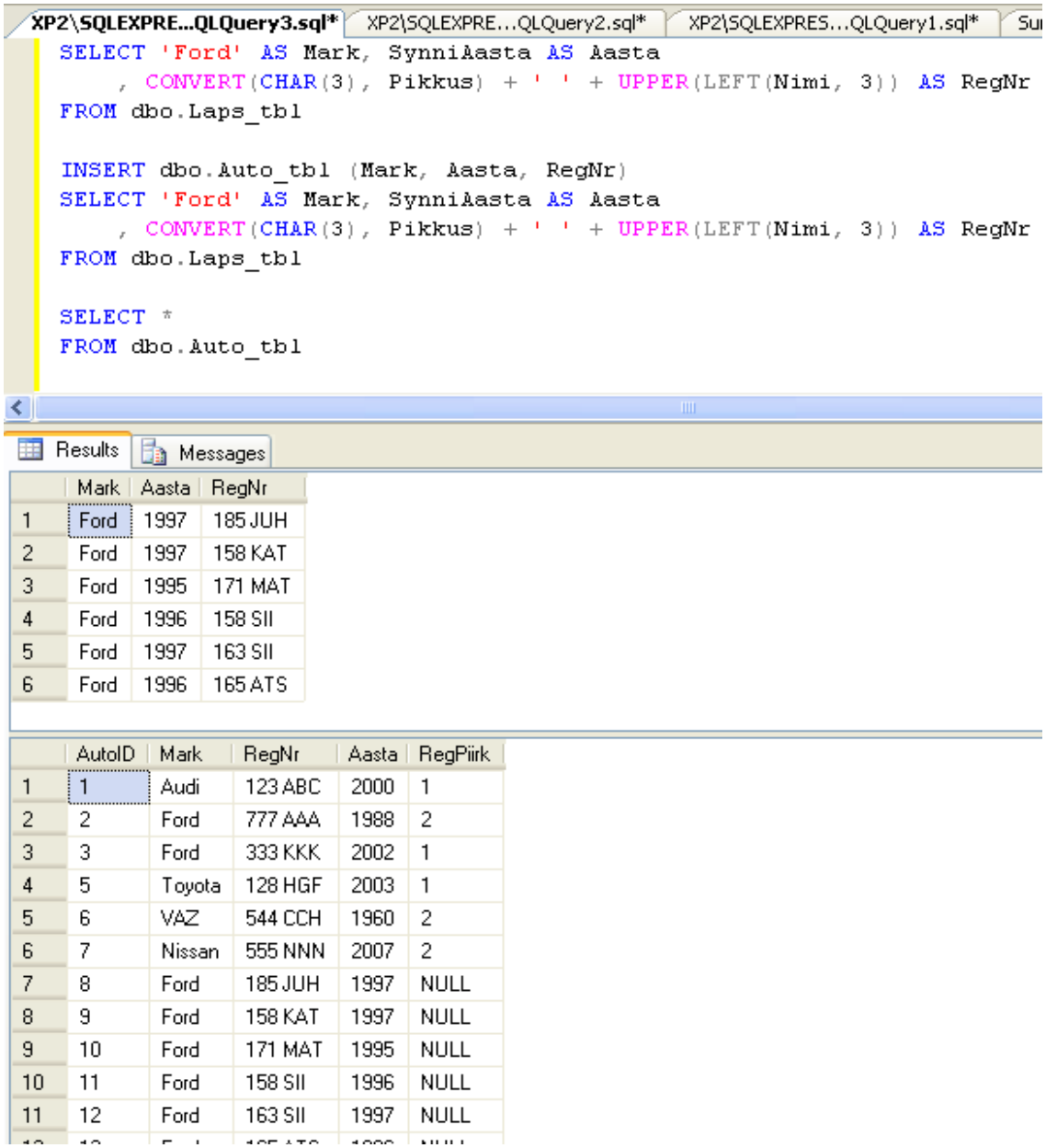
```
(1 row(s) affected)
```

XP2\SQLEXPRE...QLQuery3.sql* | XP2\SQLEXPRE...QLQuery2.sql* | XP2\SQLEXPRES...QLQuery1.sq

```
SELECT L.Nimi AS Linn, A.AutoID, A.Mark, A.RegNr, A.Aasta
FROM dbo.Linn_tbl AS L
    LEFT OUTER JOIN dbo.Auto_tbl AS A ON A.RegPiirk = L.LinnID
```

Results | Messages

	Linn	AutoID	Mark	RegNr	Aasta
1	Tallinn	1	Audi	123 ABC	2000
2	Tallinn	3	Ford	333 KKK	2002
3	Tallinn	5	Toyota	128 HGF	2003
4	Tartu	2	Ford	777 AAA	1988
5	Tartu	6	VAZ	544 CCH	1960
6	Tartu	7	Nissan	555 NNN	2007
7	Paide	NULL	NULL	NULL	NULL



Alampäringud

Kui üksik päring kipub liialt keerukaks minema või ei paista mõnd tulemust olema lootustki tavalise päringuga välja arvutada, siis võib aidata alampäring. Nii nagu avaldiste kirjutamisel võib igasugu väärtused asendada funktsioonidega, nii saab SQL-päringute puhul olemasolevad kohad asendada alampäringutega. Kusjuures tasub eristada kolme võimalust.

Ühel juhul antakse alampäringu vastuseks terve tabel (nt. SELECT * FROM lapsed). Sel juhul saab alampäringu panna kohale, kus muidu oli tabeli nimi.

Teisel juhul väljastab alampäring ühe veerutäie andmeid. Sel juhul võib kontrollida, kas uuritav rida vastab vähemasti ühele päringus väljastatud väärtustest. Kontrollimiseks käsklus IN, millest edaspidi.

Kolmas ja loodetavasti kõige lihtsam võimalus on, kus alampäring väljastab vaid ühe arvu. Sel juhul saab päringu panna selle väärtuse kohale. Alampäring kirjutatakse alati sulgudesse.

Tabeli asendaja

Siin tehti võimalikult lihtne näide, kus päringu tulemusena loodud tabelist küsitakse eraldi väärtused välja. SELECT * FROM lapsed annab tulpadeks id, eesnimi, pikkus ja synniaasta. Täiend "as tabel1" ütleb, et selle alampäringu tulemust saab edaspidises päringus kasutada nime all tabel1. Ning praegu lihtsalt küsitaksegi sealt soovitud tulbad välja.

```
SELECT tabel1.eesnimi, tabel1.pikkus
FROM (SELECT * FROM lapsed) as tabel1

Juku  155
Kati  158
Mati  164
Ats   165
Siiri 153
Siim  163
Mari  158
```

Selline vahetabelist edasi küsimine võib aga toimuda ka tunduvalt keerulisemate päringute puhul, kus ühe päringu tulemusena saadakse tabelikujuline vastus kokku ning seda asutakse järgmise päringuga edasi töötlema.

Väärtuse asendaja

Agregaatfunktsiooni või ka ühe konkreetse rea lahtri küsimise peale saab SQL-päringu panna väljastama vaid üht väärtust. Seda üksikut väärtust võib taas edaspidises päringus kasutada. Siin leitakse kõigepealt alampäringuga laste keskmine pikkus. Edasi väljastatakse kõikide laste andmed, kelle pikkus ületab keskmist.

```
SELECT eesnimi, pikkus FROM lapsed
WHERE pikkus>(SELECT AVG(pikkus) FROM lapsed)

Mati  164
Ats   165
Siim  163
```

Kontrolliks saab alampäringu väärtust ka eraldi vaadata. Näeme, et laste keskmine pikkus tabelis on 159 sentimeetrit.

```
SELECT AVG(pikkus) FROM lapsed

159
```

Väljaarvutatud väärtust võib ka avaldise sees tarvitada. Siin leitakse iga lapse pikkuse erinevus keskmisest pikkusest.

```
SELECT eesnimi, pikkus,
       pikkus - (SELECT AVG(pikkus) FROM lapsed) as erinevus
FROM lapsed
```

eesnimi	pikkus	erinevus
Juku	155	-4
Kati	158	-1
Mati	164	5
Ats	165	6
Siiri	153	-6
Siim	163	4
Mari	158	-1

Sarnase pikkusega lapsed

Järgnevalt mõned näited, kuidas sama ülesannet saab alampäringute abil mitmel moel rakendada. Lisatud näited on tehtud ühe tabeli andmete põhjal. Vähegi suuremas andmebaasis aga käiakse väärtusi sageli küllalt kaugelt küsimas.

Läbimängitava ülesandena otsitakse tabelist iga lapse kohta, kui palju on temaga sarnase pikkusega teisi lapsi. See tähendab arvuti keeles, et otsitakse iga lapse puhul, mitu on neid lapsi, kelle pikkus erineb temast mitte rohkem kui ühe sentimeetri võrra. Üheks võimaluseks on teha lihtsalt eraldi tulp. Selles tulbas väärtuse leidmiseks tuleb andmebaasimootoril igal korral vastav päring uuesti käivitada. Iga kord, kui välimises päringus võetakse ette uus inimene, loetakse kolmanda tulba ehk sarnase pikkusega laste leidmiseks uuesti kokku kõik lapsed, kelle pikkuse erinevust just sellest konkreetsest trükitavast lapsest on 1 sentimeeter või vähem. ABS tähendab absoluutväärtust. Miinus üks tulbaavaldise lõpus on vajalik, kuna trükitav laps loetakse alampäringu abil ka ise iseendaga ühepikkuste laste hulka. Et igaüks on enesega sama pikk, saabki lahutamistehte abil väärtuse õigeks.

```
SELECT eesnimi, pikkus,
       (
```

```
            SELECT COUNT(*) FROM lapsed as tabel2
            WHERE ABS(tabel2.pikkus-tabel1.pikkus)<=1
        )-1 as sarnaseid
FROM lapsed as tabel1
```

eesnimi	pikkus	sarnaseid
Juku	155	0
Kati	158	1
Mati	164	2
Ats	165	1
Siiri	153	0
Siim	163	1
Mari	158	1

Tollest miinus ühest saab vabaneda, kui eraldi tingimusse lisada, et trükitavat last ennast samapikkade laste kokku lugemisel ei arvestata. Ehk siis trükitava lapse id (tabel1.id) ning loetava lapse id (tabel2.id) ei tohi kattuda.

```
SELECT eesnimi, pikkus,
        (
            SELECT COUNT(*) FROM lapsed as tabel2
            WHERE ABS(tabel2.pikkus-tabel1.pikkus)<=1
                  AND tabel1.id<>tabel2.id
        ) as sarnaseid
FROM lapsed as tabel1
```

Ehkki kood läks veidi pikemaks, võib see hiljem paremini loetav olla. Sest salapärane -1 võib võõrale lugedes päris palju peavalu valmistada. Kui aga ilusti tingimuse abil kontrollitakse, et trükitav tegelane ei oleks kokkuloetavate hulgas – see on loodetavasti kergemini mõistetav. Tulemused on samad nagu eelmise päringu korral.

eesnimi	pikkus	sarnaseid
Juku	155	0
Kati	158	1
Mati	164	2
Ats	165	1
Siiri	153	0
Siim	163	1
Mari	158	1

Järgnevalt kasutame pikkuskaimude leidmiseks EXISTS-lauset päringu tingimusosas. Kui ennist loeti kokku, mitu sobiva pikkusega kaaslast leiti, siis siin küsitakse iga uuritava lapse puhul soovitud tingimusele vastavad kaaslased. Pikkuskaimu leidumise korral on EXISTS-kontrolli tingimus tõene ning vastava eesnime ja sünniaasta võib välja kirjutada.

```
SELECT eesnimi, pikkus
FROM lapsed as tabel1
WHERE EXISTS (
            SELECT * FROM lapsed as tabel2
            WHERE ABS(tabel2.pikkus-tabel1.pikkus)<=1
                  AND tabel1.id<>tabel2.id
       )
```

```
Kati  158
Mati  164
Ats   165
Siim  163
Mari  158
```

Sama ülesande võib lahendada veel kolme SELECT´i abil. Algus on sarnane nagu esimeses näites, kus trükkimise kolmandas tulbas arvutati välja, mitu pikkuskaimu iga lapse puhul on. Kuid kui nüüd me ei taha saada mitte arvu, vaid loetelu nendest, kel kaimud olemas, siis saab väljastatud tabelile lihtsalt veel ühe päringu ümber panna. Eesnimi ja pikkus väljastatakse ka välimises päringus. Kolmanda tulba ehk "sarnaseid" väärtust aga kasutatakse otsustamiseks, kas vastavat rida näidata või mitte.

```
SELECT eesnimi, pikkus FROM
```

```
   (SELECT eesnimi, pikkus,
       (
           SELECT COUNT(*) FROM lapsed as tabel2
           WHERE ABS(tabel2.pikkus-tabel1.pikkus)<=1
                 AND tabel1.id<>tabel2.id
       ) as sarnaseid
     FROM lapsed as tabel1
   )as tabel3
WHERE tabel3.sarnaseid>0
```

```
Kati  158
Mati  164
Ats   165
Siim  163
Mari  158
```

Veeru asendaja

Kontroll IN võimaldab tingimuses uurida, kas otsitav väärtus kattub mõnega teises päringus väljastatud väärtustest. Siinsel juhul siis sisemises päringus leitakse kõik 1997. aastal sündinud laste pikkused. Edasi välimises päringus väljastatakse kõikide laste andmed, kelle pikkus kattub kasvõi ühega eelpoolleitud pikkustest.

```
SELECT eesnimi, synniaasta FROM lapsed
WHERE pikkus IN (
   SELECT pikkus FROM lapsed
   WHERE synniaasta=1997
)
```

Nagu tulemustest näha, on tulemusridade hulgas lisaks 1997. sündinutele ka üks 1996. aastal sündinu, kel pikkust samapalju kui mõnel aasta nooremal. Tõepoolest – Mari ja Kati on ühepikkused. Ehkki esimene neist sündinud 1996. ning teine 1997. aastal.

```
Juku  1997
Kati  1997
Siim  1997
Mari  1996
```

Tekkinud tabelite ühendamine

Ka alampäringus tekkinud tabeleid saab teistega ühendada sarnaselt tavalistele võimalustele. Siin leitakse tagumises päringus iga sünniaasta kohta suurim pikkus. Ning tabelite ühendamise kaudu (ehkki praegu ühendatakse laste tabelist saadud tulemus algse tabeli enesega) leitakse iga suurima pikkuse kohta inimese nimi ja sünniaasta, kellel selline pikkus on. Juhul, kui juhtuks aastakäigu suurima pikkusega olema võrdselt mitu inimest, siis trükitaks nad kõik nagu tabelite ühendamise puhul kombeks.

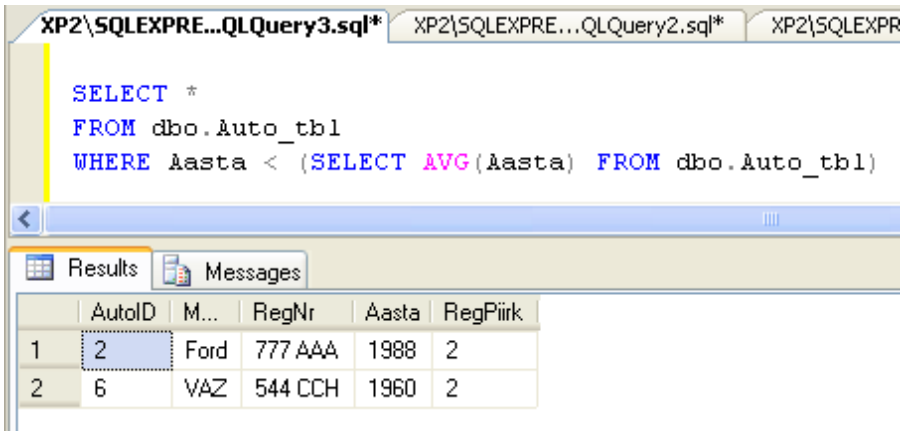
```
SELECT eesnimi, synniaasta, pikkus FROM lapsed as tabel1
INNER JOIN
(SELECT MAX(pikkus) as suurim FROM lapsed
GROUP BY synniaasta)as tabel2
ON tabel1.pikkus=tabel2.suurim


Mati  1995  164
Ats   1996  165
Siim  1997  163
```

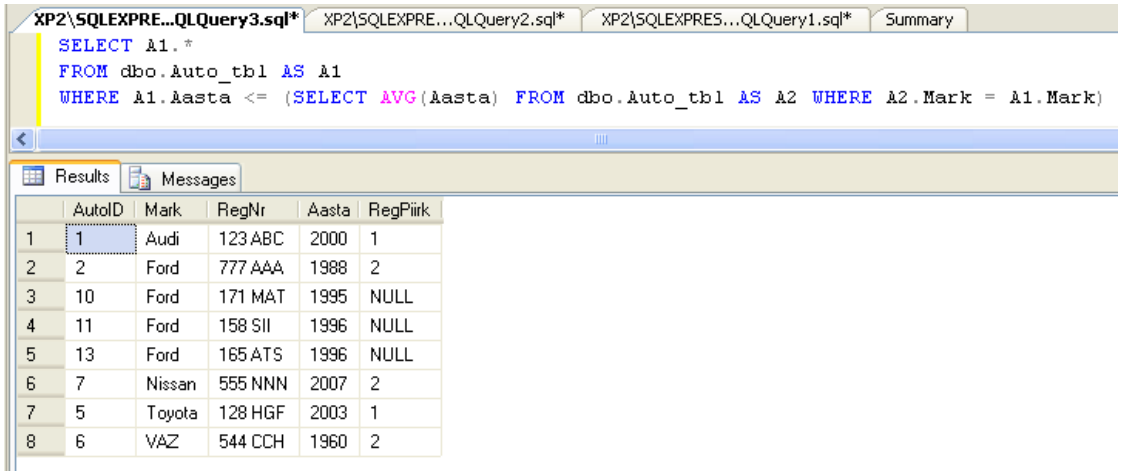
Harjutused (Alampäringud)

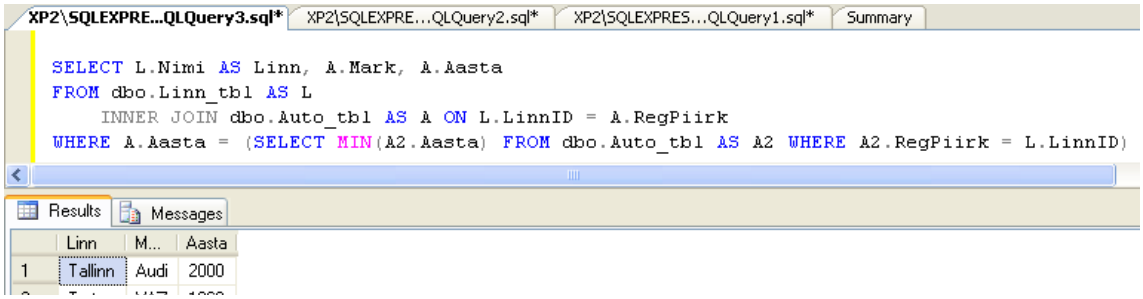
Teata alampäringu abil kõik keskmisest vanemad autod

Teata alampäringu abil markide kaupa kõik selle margi keskmisest vanemad autod

Teata iga linna kohta selles linnas liikuva kõige vanema sõiduki mark/margid



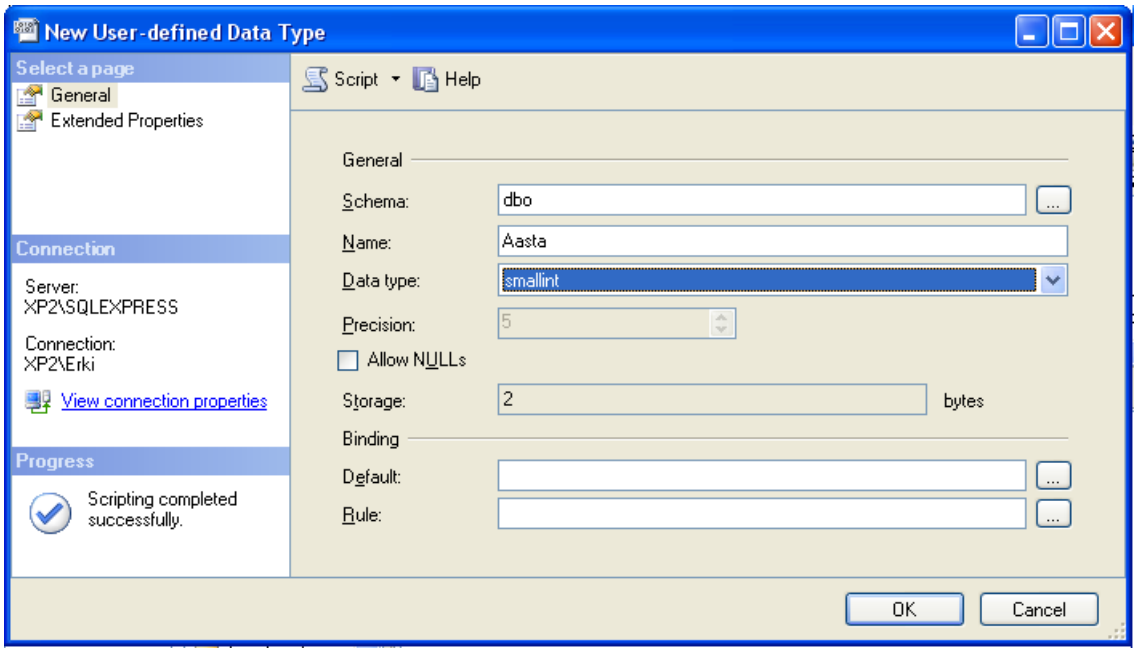




Lisavõimalused

Andmetüüpide loomine

Andmetüüpide loomiseks on vaja andmebaasi programmeerimisobjektide alt ülesse otsida Types\Use-defined Data Types ning lisada sinna uus:



Andmetüübi loomisel saate ära määrata uue andmetüübi nime, süsteemse andmetüübi, millele uus andmetüüp baseerub, kas vaikimisi lubatakse määramata väärtusi või mitte ning milliseid reegleid rakendatakse. Reeglitest räägime pisut hiljem.

Kogu selle töö saab ära teha ka lihtsa SQL Lausega:

```
CREATE TYPE [dbo].[Aasta] FROM [smallint] NOT NULL
```

Uue tabeli loomisel saame oma tehtud andmetüüpi kasutada nagu süsteemsetki. Kui soovime nüüd tagantjärele olemasolevates tabelites asenduse teha siis on olukord keerulisem kuna SQL Server selliseid asendusi ei luba. Selle asenduse saame aga teha väikese skriptiga, mis esmalt loob uue välja, seejärel kopeerib kõik andmed vanalt väljalt uuele ning kustutab vana välja.

```
ALTER TABLE dbo.Auto_tbl
ADD tmp_Aasta dbo.Aasta
GO
UPDATE dbo.Auto_tbl
SET tmp_Aasta = Aasta
ALTER TABLE dbo.Auto_tbl
DROP COLUMN Aasta
GO
EXECUTE sp_rename N'dbo.Auto_tbl.tmp_Aasta', N'Aasta', 'COLUMN'
GO
```

Isetehtud andmetüübid võimaldavad lihtsama vaevaga ühtlustada/hoida ühtsena sarnaste tunnustega andmete salvestamise.

Andmete ühtsuse tagamine

Andmete ühtsuse tagamiseks on SQL serveris mitmeid võimalusi:

- andmetüübid
- konstraandid e. piirangud
- indeksid
- võtmed
- triggerid e. päästikud
- programsed vahendid

Piirangud

Kuigi enamikes olukordades on andmetüübist tulenev piirang täiesti piisav, on palju olukordi, kus oleks soov täpsemalt määratleda andmete iseloom.

Andmetele piirangute seadmisel tulevad appi konstraandid e. piirangud.

Piiranguid on võimalik seada nii ühele väljale kui ka kirjele.

Ühele väljale e. veerule seatavad piirangud on enamasti kitsendused andmetüübile. Näiteks oleme loonud aastaarvude hoidmiseks välja ning soovime, et meie aastad oleksid vahemikus 1900 kuni jooksev aasta.

```
ALTER TABLE dbo.Auto_tbl
ADD CONSTRAINT CK_Auto_Aasta
      CHECK (Aasta BETWEEN 1900 AND YEAR(GETDATE()))
```

Kirje piirangutega kirjeldatakse seoseid samal real asuvate väljade vahel. Näiteks kui me soovime hoida andmebaasis laste juures õpingute alguse ja lõpu aega, saaksime seada piirangu, mis ütleks, et algus peab olema enne lõppu.

Konstraantide rakendamine on väga kiire ning neist on abi nii andmete sisestamisel andmeühtsuse tagamisel kui ka päringute tegemisel.

Konstraante on võimalik salvestada andmebaasi ka eraldiseisvate objektide e. reeglitena.

```
CREATE RULE Aasta1900KuniTana
AS @Aasta BETWEEN 1900 AND YEAR(GETDATE())
```

Sellisel kujul reegleid on võimalik rakendada tabeli väljadele otseselt kui ka läbi oma andmetüüpide.

Reeglite kleepimiseks andmetüübi külge tuleb kasutada salvestatud protseduuri sp_bindrule:

```
EXEC sys.sp_bindrule
          @rulename=N'[dbo].[Aasta1900KuniTana]'
          , @objname=N'[dbo].[Aasta]'
```

Indeksid

Esmajärjekorras on indeksid mõeldud andmete otsingu kiirendamiseks. Indekseerimist tasub kaaluda kõigil väljadel mida kasutatakse WHERE ja GROUP BY lauseosas ning millel rakendatakse agregaatfunktsioone.

Teiseks saab indeksitega peale suruda andmete unikaalsust kas ühel väljal või väljade kombinatsioonis. Sisuliselt on unikaalne indeks ka juba varem kasutatud primaarvõti.

Indekseid saab luua CREATE INDEX lausega. Näiteks loome unikaalse indeksi auto registrinumbrite tarbeks:

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Auto_RegNr
ON dbo.Auto_tbl ( RegNr )
```

Või indekseerime sorteerimise lihtsustamiseks laste nimed

```
CREATE NONCLUSTERED INDEX IX_Laps_Nimi
ON dbo.Laps_tbl ( Nimi )
```

Miinuspoolelt võib välja tuua, et igasugused muudatused indekseeritud väljadel toovad endaga kaasa muudatused indeksites ning seega muutuvad muutmistegevused (lisamine,

muutmine, kustutamine) aeglasemaks. Samas aitavad indeksid kaasa muudetavate kirjete leidmisel.

Alates SQL Server 2000 on võimalik tekitada indekseid ka vaadetele!

Päästikprotsess

Andmebaasides saab toiminguid ka automaatselt tööle panna. Näiteks kui soovitakse mõningates tabelites toimunud muutused soovitud kohtadesse kokku logisse lugeda.

```
CREATE TABLE logi(
  id INT IDENTITY PRIMARY KEY,
  aeg DATETIME,
  toiming VARCHAR(20),
  andmed VARCHAR(20),
)
```

SQL 2005 võimaldab luua päästikuid ka DDL käskudele e. CREATE, ALTER ja DROP käskudele!

Loomine

Järgnev päästik kannab iga linnade tabelis toimunud muutuse kohta teate logitabelisse. Salapärane nimi "inserted" tähendab päästiku sees kasutatavat ajutist tabelit, mille kaudu saab kätte lisatud rea. Selle ajutise inserted-tabeli tulpade nimed ja tüübid on samad kui tegelikul tabelil, millega muutus toimus – siinjuhul tabel nimega linnad. Siin küsitakse linnanimi kõigepealt eraldi muutujasse, et seda oleks mugavam päästiku raames toimuva lisamise juures kasutada.

```
CREATE TRIGGER linnamuutus
ON linnad
FOR INSERT
AS
INSERT INTO logi (aeg, toiming, andmed)
SELECT GETDATE(), 'lisati', linnanimi
FROM inserted
```

Käivitus

Kui juhtutakse linnade tabelisse andmeid lisama, siis näen kahel korral teateid, et "ühele reale mõjus muutus". Üks teade on siis algse lisamise kohta ning teine teade päästiku abil toimunud logikande kohta.

```
INSERT INTO linnad(linnanimi, rahvaarv)
VALUES ('Valga', 25000)

(1 row(s) affected)
```

```
(1 row(s) affected)
```

Nõnda iga muutuse korral.

```
INSERT INTO linnad(linnanimi, rahvaarv)
VALUES ('Jõgeva', 17000)
```

Ja kui edasi minna ja vaadata, mis logisse kirjutatud, siis on teated ilusti näha. Mitmes sündmus, millal, mida tehti ja millise linnaga on tegemist.

```
SELECT * FROM logi

1     2006-08-03 08:39:01.693    lisati     Valga
2     2006-08-03 08:39:18.387    lisati     Jõgeva
```

Ülesandeid

* Koosta logitabel autoregistris toimuvate muudatuste logimiseks. Tulbad

(id, auto_id, toiming, aeg, mark, aasta, maakonna_id).

* Loo päästikud lisamiste, kustutamiste, muutmiste logimiseks. Mõtle, mida

ja millistesse tulpadesse on kindlasti vajalik kirjutada, et logi järgi

oleks võimalik vana seis taastada.

* Loo salvestatud protseduur etteantud id-ga autoga tehtud toimingute

väljatrükiks inimkeelsete lausetena ajas tagant ette.

* Loo salvestatud protseduur etteantud id-ga auto seisundi näitamiseks

etteantud ajal.

Ajutiste tabelite kasutamine

Kirjutades keerukamaid programme on aeg-ajalt kasulik mingid vahetulemused meelde jätta. Üks võimalus selleks on ajutiste tabelite kasutamine.

Ajutised tabelid on tabelid, mida püsivalt pole vaja, mis on vajalikud vaid ühe konkreetse skripti täitmiseks. Selliseid tabeleid ei hoita enamasti samas kohas päris andmetega. SQL Serveril on sedasorti andmete tarbeks eraldi andmebaas tempdb. Selleks, et tabel läheks ajutisse andmebaasi tuleb tabeli nime ette panna trellid #.

Kui on soov salvestada nt autode loetelu ajutisse tabelisse saame seda teha lihtsa SELECT lausega, millele lisame INTO võtmesõna:

```
SELECT *
INTO #Autod
FROM dbo.Auto_tbl
SELECT *
FROM #Autod
```

Sellega oleme tekitanud ajutisse andmebaasi koopia autode tabelist. Ühe trelliga tähistatud ajutised tabelid on sessioonipõhised e. kui katkeb serveriga ühendus, mille kaudu see tabel tehti, kustutatakse tabel automaatselt.

On võimalik luua ka globaalseid ajutisi tabeleid. Selleks tuleb nime ette panna kahed trellid ##. Sellised tabelid kustuvad, kui katkeb viimane seda tabelit kasutav ühendus.

Ajutised tabelid on kasulikud keerukate arvutustulemuste hoidmiseks või keerulistest JOIN lausetest saadud tulemuste hoidmiseks edasiseks töötlemiseks.

Samas tuleb arvestada, et ajutistel tabelitel pole enam mingit seost läheandmetega e. sealt otsimiseks ei saa kasutada indekseid

Tsükkel, valik

Transact-SQL´il on kasutada mitmedki programmeerimiskeelele omased tunnused, kaasa arvatud muutujad, tsükkel ja valik. Nende tutvustamiseks väike koodilõik. Koodilõiku saab eraldi käivitades proovida. Pärast, kui on veendutud, et lõik töötab, saab selle protseduuriks vormistada, kirjutades ette CREATE PROCEDURE protseduurinimi AS.

Siinses näites luuakse kõigepealt kaks muutujat ning määratakse nende tüübid. Nagu näha, on muutujatel @-märk ees.

```
DECLARE @i INT, @s as VARCHAR(max)
```

Edasi saavad muutujad enesele väärtused

```
SET @i=1
SET @s=''
```

Tsükkel toimib sarnaselt nagu mõnes teiseski keeles. Tsükli keha piiratakse BEGIN ja ENDiga.

```
WHILE(@i<=10) BEGIN
```

Sama kehtib valiku kohta. Nagu näha juhul, kui pole tegemist esimese läbimiskorraga, lisatakse olemasoleva teksti lõppu koma. IF-ile saab soovi korral lisada ka ELSE-osa.

```
  IF (@i>1) BEGIN
     SET @s=@s+','
  END
```

Edasi lisandub teksti lõppu tsükli läbimiskorra number

```
  SET @s=@s+str(@i)
```

Ning et tsükkel lõputult kordama ei jääks, tuleb järgmise sammuna hoolitseda, et loenduri väärtus suureneks.

```
  SET @i=@i+1
END
```

Lõpuks võib SELECT-käskluse abil saadud tulemuse päringu käivitajale nähtavaks teha. Edasi saab sellega käituda juba nagu tavalise päringu vastusega.

```
SELECT @s as tulemus
```

Edasi kood tervikuna vaatamiseks

```
CREATE PROCEDURE arvujada AS
DECLARE @i INT, @s as VARCHAR(max)
SET @i=1
SET @s=''
WHILE(@i<=10) BEGIN
  IF (@i>1) BEGIN
     SET @s=@s+','
  END
  SET @s=@s+str(@i)
  SET @i=@i+1
END
SELECT @s as tulemus
```

ja väljastatud tulemus:

```
EXEC arvujada
tulemus
1,2,3,4,5,6,7,8,9,10
```

Muutujasse lugemine

Tahtes üht väärtust päringust kätte saada, saab selle omistada otse SELECT-lause juures.

```
DECLARE @suurimpikkus AS INT
SELECT @suurimpikkus=MAX(pikkus) FROM lapsed
PRINT @suurimpikkus
```

annab tulemuseks

```
165
```

Samuti võib muutujaid olla rohkem. Lihtsalt iga SELECT´iga küsitud tulba ette tuleb vastav muutuja omistamiseks panna.

Schema

SQL Server 2005 on iga objekti poole võimalik pöörduda objekti täispika nime kaudu. Täispikk nimi koosneb neljast osast ning on kujul server.andmebaas.schema.nimi. Täispikast nimest on võimalik ära jätta kõik peale objekti nime. Schemad e nimeruumid võimaldavad andmebaasi objekte loogiliselt grupeerida ning selle abil on lihtsam hallata ligipääsuõiguseid. Ligipääsu on seega võimalik seadistada serveri tasemel, andmebaasi tasemel, schema tasemel ning loomulikult ka objekti tasemel.

Schema loomiseks on CREATE SCHEMA lause:

```
CREATE SCHEMA yldasjad
```

Tabelite lisamiseks konkreetsesse nimeruumi tuleb tabeli loomisel näidata ära, millisesse nimeruumi tabel läheb. Vaikimisi on igas andmebaasis kasutusel vähemalt üks nimeruum dbo.

Lisame nimeruumi üldasjad tabeli proovikas:

```
CREATE TABLE yldasjad.proovikas
(
      ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
      MingiInf VARCHAR(100) NOT NULL
)
```

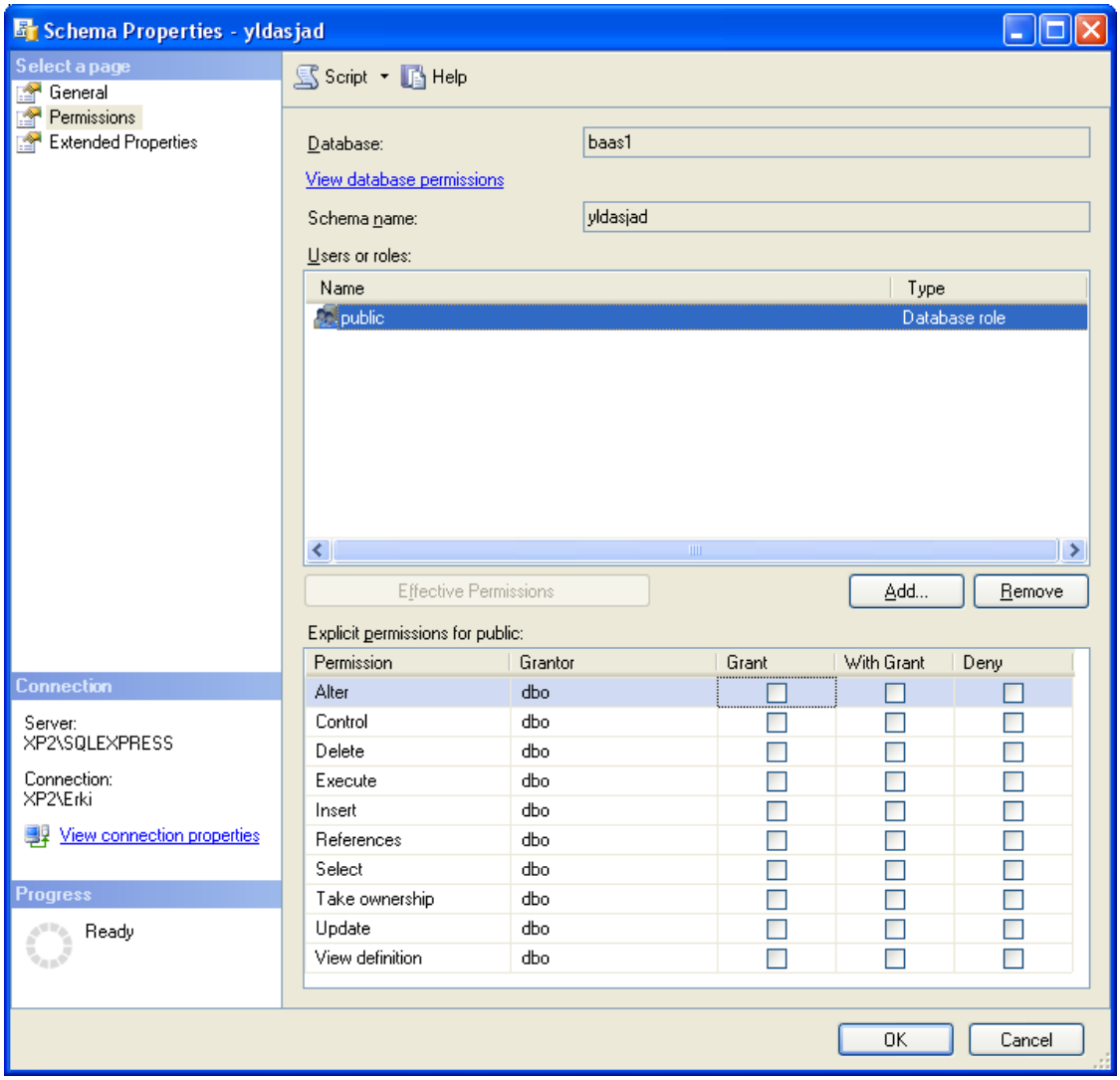
Igale kasutajale on võimalik määrata default schema:

```
CREATE USER [tegijad]
FOR LOGIN [BUILTIN\Users]
```

```
WITH DEFAULT_SCHEMA=[yldasjad]
```

Kui kasutaja ei täpsusta objekti loomisel/kasutamisel nimeruumi pöördutakse tema default nimeruumi poole. Kui sealt objekti ei leita otsitakse objekti dbo nimeruumist.

Lisaks sellele on võimalik igale kasutajale või kasutajate grupile e. rollile määrata, milliseid tegevusi ta nimeruumis võib teha:



Taas on võimalik kõike korraldada ka kasutades SQL Lauseid:

```
GRANT ALTER ON SCHEMA::[yldasjad] TO [public]
GRANT SELECT ON SCHEMA::[yldasjad] TO [public]
DENY TAKE OWNERSHIP ON SCHEMA::[yldasjad] TO [public]
```

Common Table Expression

Common Table Expression CTE on väga sarnane ajutisele vaatele, mida saab kasutada päringu FROM lauseosas.

Üldine süntaks näeb välja järgmine:

```
WITH <CTE nimi>(<väljanimed>)
AS
(
      <tegelik päring>
)
SELECT * FROM <CTE nimi>
```

CTE abil on võimalik vältida ajutiste tabelite ning alampäringute kasutamist. See omakorda muudab päringu ülevaatlikumaks ning lihtsamaks.

Näiteks soovime tegelda ainult Tallinnas (kood 1) sündinud lastega. Sellisel juhul saame endale kirjeldada CTE, milles sisalduvad vaid Tallinna lapsed ning kasutada seda oma päringus nagu iga teist tabelit või vaadet:

```
WITH TallinnaLapsed
AS
(
      SELECT *
      FROM dbo.Laps_tbl
      WHERE SynniLinn = 1
)
SELECT *
FROM TallinnaLapsed
ORDER BY Nimi
```

CTE tõeline jõud tuleb aga ilmsiks kui läbi rekursiivsete päringute. Rekursiivsed SQL päringud olid ka põhiline idee CTE loomise taga.

Rekursiivse päringu loomiseks tuleb CTE tekitada UNION päringu abil, milles on kaks liiget ankur (alguspunkt päringule) ning rekursiivne liig (iseendale viitav päring).

Et seda katsetada teeme mõned täiendused oma Laste tabelisse ning lisame sinna uue välja Rühmajuht ning määrame igale lapsele sobiva juhi:

```
-- Lisame tabelisse uue välja
ALTER TABLE dbo.Laps_tbl
ADD Ryhmajuht INT NULL REFERENCES dbo.Laps_tbl (LapsID)
GO
-- paneme rühmajuhtideks igas linnas esimese kõige vanemate
linnakodanike hulgast
```

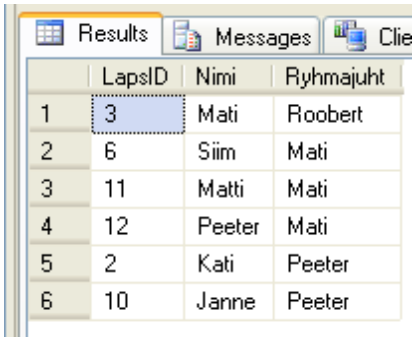
```
WITH VanimSynniA AS
( -- Iga linna varaseim sünniaasta
      SELECT SynniLinn, MIN(SYNNIAASTA) AS Aasta
      FROM dbo.Laps_tbl AS L1
      GROUP BY SynniLinn
),
Pealikud AS
( -- esimene väikseima sünniaastaga laps
      SELECT L2.SynniLinn, MIN(L2.LapsID) AS LapsID
      FROM dbo.Laps_tbl AS L2
            INNER JOIN VanimSynniA as VA ON L2.SynniLinn =
VA.SynniLinn AND L2.SynniAasta = VA.Aasta
      GROUP BY L2.SynniLinn
)
UPDATE dbo.Laps_tbl
SET Ryhmajuht = P.LapsID
FROM dbo.Laps_tbl AS L3
      INNER JOIN Pealikud AS P ON L3.SynniLinn = P.SynniLinn
WHERE L3.LapsID <> P.LapsID
GO
-- Määrame ka peetri kahe lapse rühmajuhiks
UPDATE dbo.Laps_tbl
SET RyhmaJuht = 12
WHERE LapsID IN (2,10)
-- lisame uue Lapse, kellest saab suurib pealik
DECLARE @UusLapsID INT
INSERT INTO dbo.Laps_tbl (Nimi, Synniaasta)
VALUES ('Roobert', 1988)
SET @UusLapsID = @@identity
UPDATE dbo.Laps_tbl
SET Ryhmajuht = @UusLapsID
WHERE Ryhmajuht IS NULL AND LapsID <> @UusLapsID
```

Kui kõik tehtud on meie tabel järgmine:

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus	Ryhmajuht
1	1	Juhan	185	1997	1	10	4
2	2	Kati	158	1997	2	10	3
3	3	Mati	171	1995	2	12	13
4	4	Siiri	158	1996	1	11	13
5	6	Siim	163	1997	2	10	3
6	7	Ats	165	1996	NULL	11	13
7	8	Margus	123	NULL	1	NULL	4
8	9	Salme	150	NULL	1	NULL	4
9	10	Janne	151	NULL	2	NULL	3
10	11	Matti	173	NULL	2	NULL	3
11	12	Peeter	169	NULL	2	NULL	3
12	13	Roobert	NULL	1988	NULL	19	NULL

Alustame hästi lihtsast rekursiivsest päringust püüdes välja selgitada, kes alluvad Matile:

```
WITH Alluvad
AS
(
      SELECT LapsID, Nimi, Ryhmajuht
      FROM dbo.Laps_tbl
      WHERE LapsID = 3
      UNION ALL
      SELECT L.LapsID, L.Nimi, L.Ryhmajuht
      FROM dbo.Laps_tbl AS L
            INNER JOIN Alluvad AS A ON L.RyhmaJuht = A.LapsID
)
SELECT A.LapsID, A.Nimi, L.Nimi AS Ryhmajuht
FROM Alluvad AS A
      INNER JOIN dbo.Laps_tbl AS L ON A.RyhmaJuht = L.LapsID
```


Results Messages Clie

	LapsID	Nimi	Ryhmajuht
1	3	Mati	Roobert
2	6	Siim	Mati
3	11	Matti	Mati
4	12	Peeter	Mati
5	2	Kati	Peeter
6	10	Janne	Peeter

Järgmisena alustame uuringut kõige suurematest pealikest e. kellel Rühmajuht puudub ning püüame välja selgitada mitmenda taseme alluvaga on tegemist:

```
WITH Alluvad
AS
(
      SELECT LapsID, Nimi, Ryhmajuht, 0 AS Tase
      FROM dbo.Laps_tbl
      WHERE RyhmaJuht IS NULL
      UNION ALL
      SELECT L.LapsID, L.Nimi, L.Ryhmajuht, A.Tase + 1
      FROM dbo.Laps_tbl AS L
            INNER JOIN Alluvad AS A ON L.RyhmaJuht = A.LapsID
)
SELECT A.Tase, A.LapsID, A.Nimi, L.Nimi AS Ryhmajuht
FROM Alluvad AS A
      LEFT OUTER JOIN dbo.Laps_tbl AS L ON A.RyhmaJuht = L.LapsID
ORDER BY A.Tase
```

Results | Messages | Client Statistic

	Tase	LapsID	Nimi	Ryhmajuht
1	0	13	Roobert	NULL
2	1	3	Mati	Roobert
3	1	4	Siiri	Roobert
4	1	7	Ats	Roobert
5	2	1	Juhan	Siiri
6	2	8	Margus	Siiri
7	2	9	Salme	Siiri
8	2	6	Siim	Mati
9	2	11	Matti	Mati
10	2	12	Peeter	Mati
11	3	2	Kati	Peeter
12	3	10	Janne	Peeter

Ülesandeid

- Koosta tabel isikud tulpadega id, eesnimi, isaID. Lisa mõned andmed. Juhul, kui isa on tabelis olemas, siis isaID näitab isiku id-le, kes on vastava rea inimese isaks.
- Trüki välja kõikide isikute nimed, kellel tabelis isa puudub
- WITH-lausega loo vahetabel isikutest, kel tabelis isa olemas, all päringus näita nende inimeste andmed välja.
- Lisaks eelmisega ühenda JOINi abil isiku nime juurde ka tema isa nimi
- Trüki välja etteantud inimesest alates pärinev sugupuu – alati kirjas, kes on kelle isa.
- Lisa juurde arv, mitmes põlv ta tabelis näidatud esiisast on
- Näita iga inimese kohta, mitu järeltulijat tal tabeli kaudu leitavas sugupuus on.

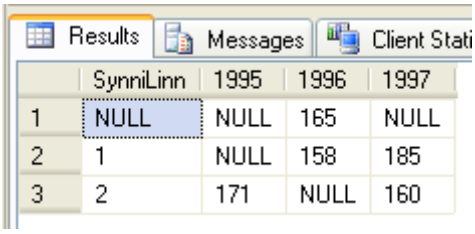
PIVOT JA UNPIVOT

Pivot tekitab tavalisest tabelist kahemõõtmelise risttabeli. Unpivot teeb täpselt vastupidist.

Püüame tekitada risttabeli, milles oleks ridades laste sünnilinn ning veergudes sünniaastad ning andmetena laste keskmised pikkused:

```
SELECT *
FROM
      ( SELECT SynniAasta, SynniLinn, Pikkus
        FROM dbo.Laps_tbl ) AS data
PIVOT (
```

```
      AVG (Pikkus)
      FOR Synniaasta IN ([1995], [1996], [1997])
) AS piv
```



	SynniLinn	1995	1996	1997
1	NULL	NULL	165	NULL
2	1	NULL	158	185
3	2	171	NULL	160

Vastupidise teisenduse saame korraldada UNPIVOT käsuga. Selleks salvestan eelnevalt eelmise pärinu tulemuse ajutisse tabelisse kasutades SELECT ... INTO #ristt … konstruktsiooni, ehk siis

```
SELECT * INTO #ristt
FROM
      ( SELECT SynniAasta, SynniLinn, Pikkus
        FROM dbo.Laps_tbl ) AS data
PIVOT (
      AVG (Pikkus)
      FOR Synniaasta IN ([1995], [1996], [1997])
) AS piv
```

```
SELECT *
FROM
    ( SELECT * FROM #ristt ) as piv
UNPIVOT (
    KeskmPikkus FOR Aasta IN ([1995],[1996],[1997])
) AS unpiv
```

	SynniLinn	KeskmPikkus	Aasta
1	NULL	165	1996
2	1	158	1996
3	1	185	1997
4	2	171	1995
5	2	160	1997

Iseenesest õnnestus ka varem moodustada analoogseid konstruktsioone, kasutades CASE valikuid SELECT loetelus. PIVOT on muutnud selle protsessi oluliselt lihtsamaks kuid risttabelisse minevad väljanimed (veerud) tuleb endiselt sisse trükkida ning neid pole võimalik tekitada dünaamiliselt.

Lahenduseks on sellele dünaamiline SQLi koostamine ning käivitamine EXEC abil.

Selleks tuleb luua skript, mis tekitab komadega eraldatud loetelu kõigist risttabelisse minevatest väljadest ning kleepida see SQLis õige koha peale.

Kuna kasutame SQL 2005 siis teeksin selle tegevuse kasutades CTE abi:

```
DECLARE @aastad AS VARCHAR(MAX);
WITH AastaCTE
AS
(
     SELECT DISTINCT SynniAasta AS Aasta FROM dbo.Laps_tbl
)
SELECT @aastad = ISNULL(@aastad + ',[', '[') +
CAST(Aasta AS CHAR(4)) + ']'
FROM AastaCTE
ORDER BY Aasta
DECLARE @sql NVARCHAR(max)
SET @sql = N'SELECT *
     FROM ( SELECT Pikkus, SynniAasta, SynniLinn
            FROM dbo.Laps_tbl ) AS Lapsed
     PIVOT ( AVG(Pikkus) FOR SynniAasta IN(' + @aastad + N')) AS
Piv'
EXEC (@sql)
```

Ülesandeid

- Näita laste tabelis iga nime ja etteantud kohta, mitu last on sel aastal sündinud (nt. mitu Jukut aastal 1998)
- Kasuta SQL-lause genereerimist, et saaks loetellu kõik olemasolevad sünniaastad.
- Salvesta tulemus abitabelisse
- Eralda andmed taas UNPIVOT käsu abil.

APPLY

APPLY on uus operaator FROM lauseosas, mis võimaldab Teil teha väljakutseid tabelit tagastavatele funktsioonile iga rea kohta peapäringus.

Näiteks huvitavad, meid iga linna kõige pikemad lapsed. Selleks loome funktsiooni, mis loetleb meile ülesse konkreetse linna kõige pikemad lapsed:

```
CREATE FUNCTION dbo.fn_Pikimad (@LinnID int, @Top int)
RETURNS TABLE
AS
     RETURN SELECT TOP (@Top) *
            FROM dbo.Laps_tbl
            WHERE SynniLinn = @LinnID
```

```
ORDER BY Pikkus DESC
```

Selleks, et vaadata seda infot konkreetsete linnade kaupa saame kirjutada päringu:

```
SELECT *
FROM dbo.Linn_tbl
CROSS APPLY dbo.fn_Pikimad(LinnID, 2)
```

Nummerdamised

ROW_NUMBER() – nummerdab kõik read tulemuses

RANK() – ütleb, mitmes on loetelus antud väärtus

DENSE_RANK() – analoogne RANK funktsiooniga, kuid ei jäta numbreid vahele

NTILE(n) – jagab tulemuse N grupiks

```
SELECT *
    , ROW_NUMBER() OVER (ORDER BY Pikkus DESC) AS jrk
    , RANK() OVER (ORDER BY Pikkus DESC) AS koht
    , DENSE_RANK() OVER (ORDER BY Pikkus DESC) AS koht2
    , NTILE(3) OVER (ORDER BY Pikkus DESC) AS grupp
FROM dbo.Laps_tbl
```

Results | Messages | Client Statistics

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus	Ryhmajuht	jrk	k...	ko...	grupp
1	1	Juhan	185	1997	1	10	4	1	1	1	1
2	11	Matti	173	NULL	2	NULL	3	2	2	2	1
3	3	Mati	171	1995	2	12	13	3	3	3	1
4	12	Peeter	169	NULL	2	NULL	3	4	4	4	1
5	7	Ats	165	1996	NULL	11	13	5	5	5	2
6	6	Siim	163	1997	2	10	3	6	6	6	2
7	4	Siiri	158	1996	1	11	13	7	7	7	2
8	2	Kati	158	1997	2	10	12	8	7	7	2
9	10	Janne	151	NULL	2	NULL	12	9	9	8	3
10	9	Salme	150	NULL	1	NULL	4	10	10	9	3
11	8	Margus	123	NULL	1	NULL	4	11	11	10	3
12	13	Roobert	NULL	1988	NULL	19	NULL	12	12	11	3

Kõiki nummerdamise funktsioone on võimalik kasutada ka üle gruppide jagades andmed sobiva tunnuse alusel partitsioonideks:

```
SELECT *
     , ROW_NUMBER() OVER (PARTITION BY RyhmaJuht ORDER BY Pikkus DESC) AS jrk
     , RANK() OVER (PARTITION BY RyhmaJuht ORDER BY Pikkus DESC) AS koht
     , DENSE_RANK() OVER (PARTITION BY RyhmaJuht ORDER BY Pikkus DESC) AS koht2
     , NTILE(3) OVER (ORDER BY Pikkus DESC) AS grupp
FROM dbo.Laps_tbl
ORDER BY RyhmaJuht, Pikkus DESC
```

Results | Messages | Client Statistics

	LapsID	Nimi	Pikk...	Synniaasta	SynniLinn	Vanus	Ryhmajuht	jrk	k...	ko...	grupp
1	13	Roobert	NULL	1988	NULL	19	NULL	1	1	1	3
2	11	Matti	173	NULL	2	NULL	3	1	1	1	1
3	12	Peeter	169	NULL	2	NULL	3	2	2	2	1
4	6	Siim	163	1997	2	10	3	3	3	3	2
5	1	Juhan	185	1997	1	10	4	1	1	1	1
6	9	Salme	150	NULL	1	NULL	4	2	2	2	3
7	8	Margus	123	NULL	1	NULL	4	3	3	3	3
8	2	Kati	158	1997	2	10	12	1	1	1	2
9	10	Janne	151	NULL	2	NULL	12	2	2	2	3
10	3	Mati	171	1995	2	12	13	1	1	1	1
11	7	Ats	165	1996	NULL	11	13	2	2	2	2

Vaade

Sagedamini vajaminevad päringud on mõistlik vormistada vaatena. Nõnda optimeerib server päringu ühe korra ära ning edaspidi saab tema tulemusi kiiremini vaadata. Samuti pole põhjust keerukamaid ridu hakata taas uuesti kokku panema. Ka aitavad vaated olukordades, kus päringud kipuvad muidu liialt keerukaks minema. Nii nagu alampäring tekitab vajadusel edasi töödeldavad andmed, nii saab ka vaateid edasi kasutada sarnaselt, nagu oleksid need täiesti tavalised tabelid.

Määratud õigustega andmebaasis saab vaadete järgi kergemini ja täpsemalt sättida, millised kasutajad millistele andmetele ligi pääsevad. Muul juhul võib tükk tegemist olla, et kasutaja näeks vaid üksikutest veergudest kindlatele tunnustele vastavaid ridu. Kui aga kasutajal lubatakse vaid vaadetest andmeid näha ning vaate juures on sobivad piirangud õigesti seatud, siis toimib kõik nõnda nagu liidese looja vajalikuks peab.

Vaated aitavad ka pidada andmete hoidmist ja kasutamist paindlikumana. Kui kasutaja või kasutav programm küsib andmeid vaid vaadete kaudu, siis saab vajadusel tabelistruktuuri enese julgesti ära muuta. Jääb vaid hoolitseda, et pärast muutust vaade jälle õigetest kohtadest omale andmed kätte saaks.

Vaate kirjutamine näeb üldjuhul välja nagu täiesti tavalise SELECT-päringu kirjutamine. Lihtsalt algusritta tuleb määrata vaate nimi. Pikemate laste leidmise päring siis järgmiselt:

```
CREATE VIEW pikadlapsed AS
SELECT eesnimi, pikkus FROM lapsed
WHERE pikkus > 160
```

Õnnestunud loomise peale teatatakse lihtsalt, et

```
Command(s) completed successfully.
```

Vaatest andmete küsimine näeb välja sarnaselt tabelile.

```
SELECT * FROM pikadlapsed
```

annab välja piisavalt pikkade laste loetelu nagu vaate loomisel sai ette nähtud.

```
Mati  164
Ats   165
Siim  163
```

Kui mõnel põhjusel soovitakse loodud vaatest loobuda või vaade uuega asendada, siis lahti saab sellest käsuga DROP. Näiteks

```
DROP VIEW pikadlapsed
```

Ülesandeid

* Loo maakondade tabel (id, maakonnanimi)

* Loo autode tabel (id, mark, aasta, maakonna_id)

* Loo vaade, kus näha iga auto mark, väljalaskeaasta ning maakonna nimi.

* Loo vaade, kus näha registreeritud autode arv maakondade kaupa.

* Veendu vaadete toimimist andmete muutmisel.

* Loo vaade, mis kasutaks eelnevalt loodud vaadet ning näitaks vaid neid

maakondi, kus autosid rohkem kui 1.

Salvestatud protseduur

Levinud tegevuste tarbeks on võimalik andmebaasis salvestada protseduur. Nii saab andmebaasis teha valmis keerukad toimingud, mida hiljem vajaduse korral vaid ühe käsuga võib baasi haldusliidese või omakoostatud programmi kaudu välja kutsuda.

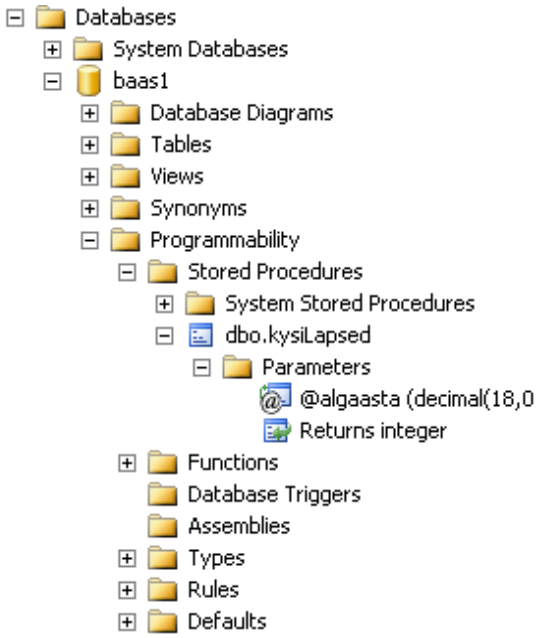
Loomine

Kõigepealt loomise näide:

```
CREATE PROCEDURE kysiLapsed(@algaasta decimal)
AS
SELECT eesnimi, synniaasta FROM lapsed
WHERE synniaasta>=@algaasta
```

Kui selline käsujada tipitakse SQL-serveri haldusliidesesse või sisestatakse muul moel käsujadana, siis salvestatakse andmebaasi protseduur nimega kysiLapsed.

Tulemusena tekib baasi haldusliidesesse alajaotisesse Programmability -> Stored Procedures vastav salvestatud protseduur, mille kohta saab soovi korral ka uurida, millist tüüpi andmeid ta ette tahab ning millisel kujul tulemuse väljastab.



Protseduur eeldab, et baasis leidub tabel "lapsed", millel tulpadeks on vähemasti eesnimi ja synniaasta. Protseduur väljastab laste andmed, kes on sündinud etteantud aastal või hiljem. Halduskeskkonnas sobib käivitamiseks käsklus kujul

```
EXEC kysiLapsed 1997
```

Väljundiks loetelu nagu soovitud:

```
Juku  1997
Kati  1997
Siim  1997
```

Ülesandeid

* Loo salvestatud protseduur näitamaks, mitu autot on registris varasemad kui etteantud väljalaskeaasta.

* Loo salvestatud protseduur auto lisamiseks registrisse. Parameetriteks

mark, väljalaskeaasta ja maakonna id.

* Võrreldes eelmisega sisestatakse maakonna nimi.

* Sobiva maakonnanime puudumisel antakse veateade.

* Väljalaskeaasta määramata jätmisel pannakse selleks käesolev aasta.

* Loo salvestatud protseduur, mis koostaks ja väljastaks tabeli, kus ühes

tulbas on maakondade nimed ning kõrval teises tulbas komadega eraldatult

sinna maakonda registreeritud erinevate sõidukite margid.

Transaktsioonid (Transaction)

Transaktsioonid on tegevused, kus kõik tegevused kas õnnestuvad edukalt või ei õnnestu üldse. SQL Server tunnistab kahte tüüpi transaktsioone:

- Automaatsed transaktsioonid – Server tekitab need ise kõigi muutmistegevustega st UPDATE, INSERT ja DELETE lausetega. Sisuliselt tähendab see seda, et kui nt üritate muuta 10 kirje väärtust ja ühe kirje väärtuse muutmine ei õnnestu, siis ei muudeta neist ühtegi.

- Käsitsi tehtud transaktsioonid – Programmeerija loodud tegevuste jadad, mis peavad kõik õnnestuma.

Transaktsiooni tekitamiseks on käsk BEGIN TRAN. Selle järele tulevad kõik soovitud tegevused. Kui transaktsioon lõpeb edukalt, siis saab selle lõpetada käsuga COMMIT TRAN. Kui midagi läheb valesti, saab selle tagasi kerida käsuga ROLLBACK TRAN.

Vaatame näiteks, kuidas võiks luua protseduuri pangaülekandeks. Kuna SQL Server 2005 tunnistab uut TRY CATCH konstruktsiooni vigade haldamiseks, siis vaatleme seda näidet nii SQL Server 2000 kui ka SQL Server 2005 baasil.

```
CREATE PROC ylekanne
      @kellelt int,
      @kellele int,
      @summa money
AS
      BEGIN TRAN
            UPDATE konto
            SET jaak = jaak - @summa
            WHERE omanik = @kellelt
            IF @@error <> 0 BEGIN
                  ROLLBACK TRAN
                  RETURN
            END
            UPDATE konto
            SET jaak = jaak + @summa
            WHERE omanik = @kellele
            IF @@error <> 0 BEGIN
                  ROLLBACK TRAN
                  RETURN
            END
      COMMIT TRAN
GO
```

```
CREATE PROC ylekanne2
      @kellelt int,
      @kellele int,
      @summa money
AS
      BEGIN TRAN
      BEGIN TRY
            UPDATE konto
            SET jaak = jaak - @summa
            WHERE omanik = @kellelt
            UPDATE konto
            SET jaak = jaak + @summa
            WHERE omanik = @kellele
      END TRY
      BEGIN CATCH
            IF @@trancount > 0
                  ROLLBACK TRAN
      END CATCH

      IF @@TRANCOUNT > 0
            COMMIT TRANSACTION;
GO
```

Samast raha ülekandmisest nüüd pikem näide. Pigem jäägu ülekanne teostamata, kui et toimingu sees osa raha ära kaob või juurde tekib. Et kontodega toimetada, sai loodud võimalikult lihtne tabel – vaid konto number ning seal olev saldo.

```
create table kontod(
  id int identity not null primary key,
  saldo money
)
```

Edasi luuakse salvestatud protseduur raha ülekandeks. Kust kontolt võtta, kuhu panna ning kui suur on summa. Esimesed kaks arvu tähendavad siis vastavate kontode numbreid.

```
create procedure ylekanne
  (@kust int, @kuhu int, @summa money)
as
```

Abimuutujas hoitakse meeles, kas kõik õnnestus hästi.

```
declare @korras as int
```

Esialgu probleeme pole, nii et @korras saab väärtuseks 1.

```
set @korras=1
```

Kogu järgnev toiming pannakse transaktsiooni sisse. See tähendab, et sealsed muutused kas toimivad tervikuna või jäävad sootuks ära.

```
begin transaction
```

Kõigepealt kontrollitakse, kas esimeselt kontolt on võimalik vastav summa maha võtta. Kui saab, siis võetakse, muul juhul väljastatakse veateade ja muutujasse @korras antakse teada, et kord läks kaduma.

```
    if (select saldo from kontod where id=@kust)>=@summa begin
       update kontod set saldo=saldo-@summa where id=@kust
    end else begin
       set @korras=0
       raiserror('raha otsas', 1, 1)
    end
```

Järgmise sammuga kontrollitakse, et ka see konto ikka olemas on, kuhu raha kanda soovitakse. Hariliku UPDATE-lause puhul ei antaks isegi veateadet juhul, kui saajakontot

olemas poleks. Siinse kontrolliga aga tehakse olemasolu kindlaks ja vaid sel juhul suurendatakse sealset summat. Muul juhul jäetakse jälle meelde, et asjad pole korras.

```
     if exists(select saldo from kontod where id=@kuhu) begin
        update kontod set saldo=saldo+@summa where id=@kuhu
     end else begin
        set @korras=0
     end
```

Edasi jääb üle vaid muutuja järgi otsustada, kas toiming kinnitada või tagasi lükata.

```
if @korras=1
   commit transaction
else begin
   rollback transaction
   print 'probleem'
end
```

Ja et protseduuri käivitamise tulemusena oleks ka kontode operatsioonijärgset seisu näha, selleks lõppu üks SELECT-lause.

```
select * from kontod where id in (@kust, @kuhu)
```

Ning kood tervikuna.

```
create procedure ylekanne
  (@kust int, @kuhu int, @summa money)
as
declare @korras as int
set @korras=1
begin transaction
     if (select saldo from kontod where id=@kust)>=@summa begin
        update kontod set saldo=saldo-@summa where id=@kust
     end else begin
        set @korras=0
        raiserror('raha otsas', 1, 1)
     end
     if exists(select saldo from kontod where id=@kuhu) begin
        update kontod set saldo=saldo+@summa where id=@kuhu
     end else begin
        set @korras=0
     end
if @korras=1
   commit transaction
else begin
   rollback transaction
   print 'probleem'
end
select * from kontod where id in (@kust, @kuhu)
```

Tahtes loodud protseduur käivitada, tuleb siis ette anda andjakonto, saajakonto ja ülekantav summa. Ning juhul, kui ülekanne on võimalik, see ka tehakse.

```
EXEC ylekanne 1, 4, 100
```

Ülesandeid

* Loo salvestatud protseduur sõiduki ühest maakonnast teise ümber registreerimiseks. Parameetriteks sõiduki id ning maakonna id. Kõigepealt võetakse sõiduk algsest maakonnast maha. Kui edasi selgub, et uuele id-le ei vasta maakonda, kuhu registreerida, siis taastatakse algseisund ROLLBACK´i abil. Kontrolli toimimist.

XML andmete kasutamine

XML andmete tugi on olnud juba alates SQL 2000st. SQL 2005 on seda tuge oluliselt parandatud ning lisandunud on isegi spetsiaalne andmetüüp. XML andmeväli erineb tavalisest tekstist selle poolest, et seal hoitakse andmeid programselt lihtsalt kasutatava objektina ning sellelt väljalt on lihtsam teha XML päringuid ning sinna saab tekitada ka XML indekseid. Selle välja juures tuleb aga meeles pidada, et sinna salvestuvad küll kõik andmed, kuid kuna andmeid hoitakse puu kujul, kaob esialgsetest andmetest ära tühiruum. Seega kui Teile on oluline, et andmed säiliksid täielikult sellisel kujul nagu need sisestati siis tuleb xml andmetüübi asemel kasutada tavalist tekstivälja.

FOR XML

Kõige lihtsam moodus XML andmete genereerimiseks on lisada SELECT lause lõppu FOR XML märksõna ning andmed teisenevad XML kujule. Teisenduse protsessi kontrollimiseks on kasutada neli erinevat metoodikat.

Kõige lihtsam neist on RAW. RAW tekitab kõigist tulemuses olevates ridadest XML elemendid nimega „row“, kus kõik SELECT loetelus olevad väljad on atribuutideks. Seega peavad kõikide väljade nimed olema erinevad!

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
FOR XML RAW
```

```
<row nimi="Juhan" pikkus="185" />
<row nimi="Kati" pikkus="158" />
<row nimi="Mati" pikkus="171" />
<row nimi="Siiri" pikkus="158" />
<row nimi="Siim" pikkus="163" />
<row nimi="Ats" pikkus="165" />
```

Nagu näeme ei ole loodud XMLil juurelement e. tegemist ei ole well formed XMLiga. Selle vea saame lihtsalt parandada lisades juurde ROOT märksõna:

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
FOR XML RAW, ROOT

<root>
  <row nimi="Juhan" pikkus="185" />
  <row nimi="Kati" pikkus="158" />
  <row nimi="Mati" pikkus="171" />
  <row nimi="Siiri" pikkus="158" />
  <row nimi="Siim" pikkus="163" />
  <row nimi="Ats" pikkus="165" />
</root>
```

Loomulikult võime nii juurkale kui ka elementidele anda ka mingid teised nimed, näidates uued nime vastava märksõna järel sulgudes:

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
FOR XML RAW('laps'), ROOT('lapsed')

<lapsed>
  <laps nimi="Juhan" pikkus="185" />
  <laps nimi="Kati" pikkus="158" />
  <laps nimi="Mati" pikkus="171" />
  <laps nimi="Siiri" pikkus="158" />
  <laps nimi="Siim" pikkus="163" />
  <laps nimi="Ats" pikkus="165" />
</lapsed>
```

Saame teha ka nii, et atribuutide asemel genereeruvad elemendid. Selleks lisame juurde ELEMENTS märksõna:

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl
FOR XML RAW('laps'), ROOT('lapsed'), ELEMENTS

<lapsed>
  <laps>
    <nimi>Juhan</nimi>
    <pikkus>185</pikkus>
  </laps>
  <laps>
    <nimi>Kati</nimi>
    <pikkus>158</pikkus>
  </laps>
  <laps>
    <nimi>Mati</nimi>
    <pikkus>171</pikkus>
  </laps>
  <laps>
    <nimi>Siiri</nimi>
    <pikkus>158</pikkus>
```

```
  </laps>
  <laps>
    <nimi>Siim</nimi>
    <pikkus>163</pikkus>
  </laps>
  <laps>
    <nimi>Ats</nimi>
    <pikkus>165</pikkus>
  </laps>
</lapsed>
```

Järgmine võimalus on genereerida XMLi automaatselt:

```
SELECT nimi, pikkus
FROM dbo.Laps_tbl AS Laps
FOR XML AUTO, ROOT('lapsed')

<lapsed>
  <Laps nimi="Juhan" pikkus="185" />
  <Laps nimi="Kati" pikkus="158" />
  <Laps nimi="Mati" pikkus="171" />
  <Laps nimi="Siiri" pikkus="158" />
  <Laps nimi="Siim" pikkus="163" />
  <Laps nimi="Ats" pikkus="165" />
</lapsed>
```

Nagu näeme valitakse sellisel juhul elementide nimed vastavalt kasutatud tabelitele. Automaatse genereerimise juures on veel tore see, et see automaatika oskab järgida JOIN lauseid ning tekitab hierarhilise XMLi.

```
SELECT Linn.Nimi, Laps.Nimi, Laps.Pikkus
FROM dbo.Linn_tbl as Linn
     INNER JOIN dbo.Laps_tbl AS Laps ON Linn.LinnID = Laps.Synnilinn
ORDER BY Linn.Nimi
FOR XML AUTO, ROOT('lapsed')

<lapsed>
  <Linn Nimi="Tallinn">
    <Laps Nimi="Juhan" Pikkus="185" />
    <Laps Nimi="Siiri" Pikkus="158" />
  </Linn>
  <Linn Nimi="Tartu">
    <Laps Nimi="Kati" Pikkus="158" />
    <Laps Nimi="Mati" Pikkus="171" />
    <Laps Nimi="Siim" Pikkus="163" />
  </Linn>
</lapsed>
```

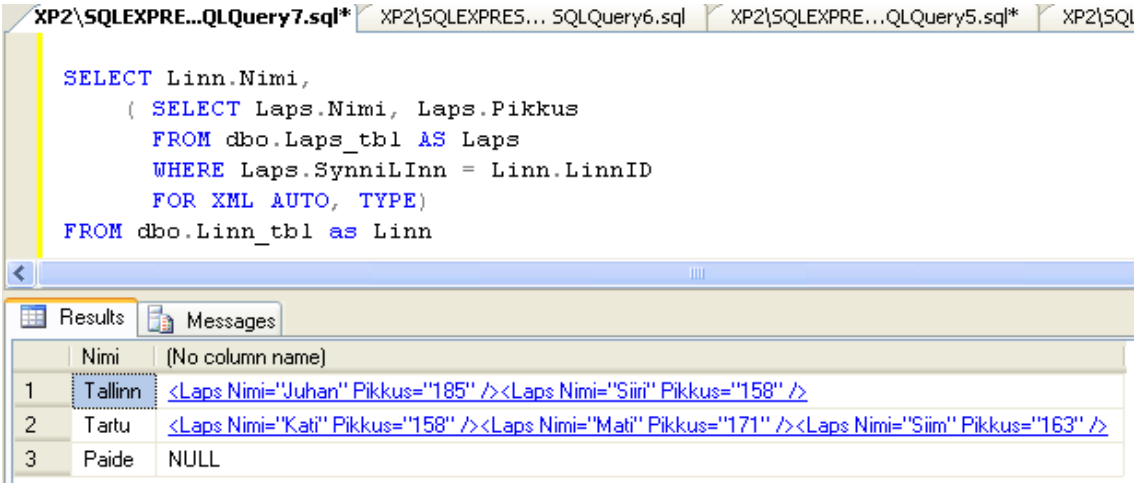
Eelpool toodud kahest variandist keerukamate XML struktuuride tekitamiseks on EXPLICIT ning PATH märksõnad.

Kui soovite, et päringu või alampäringu tulemuks oleks XML andmetüübis tuleb lisada FOR XML lauseosasse TYPE märksõna. Sellisel juhul on võimalik genereerida XMLi XMLi sisse.

```
SELECT Linn.Nimi,
      ( SELECT Laps.Nimi, Laps.Pikkus
        FROM dbo.Laps_tbl AS Laps
        WHERE Laps.SynniLInn = Linn.LinnID
        FOR XML AUTO, TYPE)
FROM dbo.Linn_tbl as Linn
FOR XML AUTO

<Linn Nimi="Tallinn">
  <Laps Nimi="Juhan" Pikkus="185" />
  <Laps Nimi="Siiri" Pikkus="158" />
</Linn>
<Linn Nimi="Tartu">
  <Laps Nimi="Kati" Pikkus="158" />
  <Laps Nimi="Mati" Pikkus="171" />
  <Laps Nimi="Siim" Pikkus="163" />
</Linn>
<Linn Nimi="Paide" />
```

SELECT loetelus kasutatav alampäring tagastab mitu kirjet ning tavaolukorras ei oleks selline asi lubatud, kuid teisendades andmed XML kujule on kõik lubatud ;)

Lisaks sellele võime väliselt päringult FOR XML lauseosa ära jätta ning tekib tabel e. recordset, kus on kombineeritud relatsioonilised andmed ja XML andmed:



OPENXML

FOR XML võimaldab meil XMLi genereerida edasi saatmiseks. OPENXML on vastupidine protsess, mis võimaldab meil saabunud XMLi töödelda.

XMLi töötlemine OPENXML käib viie sammulise protsessina:

- Saabub XML dokument
- sp_xml_preparedocument salvestatud protseduuri abil tekitame mällu XML puu
- kasutades OPENXMLi teisendame puust sobivad tükid relatsioonilisele e. SQL Serverile kodusele kujule
- Teeme andmetega vajalikud toimingud nt salvestame info tabelitesse
- kasutades sp_xml_removedocument kustutame XMl puu ning vabastame sellega seotud mälu

sp_xml_preparedocument kasutamiseks on meil vaja INT tüüpi muutujat, millesse salvestada viide XML puule mälus. Saadud viidet vajavad nii OPENXML kui ka sp_xml_removedocument. Seega peaks protsess nägema välja u. sarnane:.

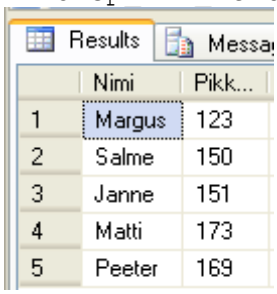
```
DECLARE @doc xml
-- kusgilt saabub mingi XML dokument
DECLARE @hdoc INT
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
-- teeme vajalikud tehingud
EXEC sp_xml_removedocument @hdoc
```

OPENXML süntaks on järgmine:

OPENXML(idoc, rowpattern [, flags]) WITH (SchemaDeclaration | TableName)

- rowpattern – Xpath päring, mis kirjeldab tagastatavad XML oksad
- idoc – viide mälus asuvale XML puule
- flags . kas tuua:
- atribuudid (default)
- atribuudid
- elemendid
- nii atribuudid kui elemendid
- SchemaDeclaration – tagastatava tabeli kirjeldus
- TableName – olemasolev tabel, mille Schemat kasutada

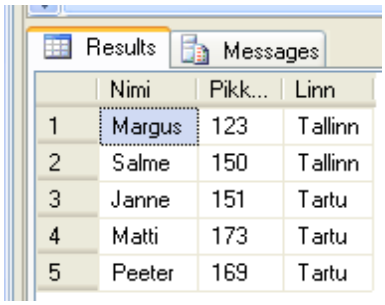
```
DECLARE @doc xml
SET @doc =
      '<lapsed>
        <Linn Nimi="Tallinn">
           <Laps Nimi="Margus" Pikkus="123" />
           <Laps Nimi="Salme" Pikkus="150" />
        </Linn>
        <Linn Nimi="Tartu">
           <Laps Nimi="Janne" Pikkus="151" />
           <Laps Nimi="Matti" Pikkus="173" />
           <Laps Nimi="Peeter" Pikkus="169" />
        </Linn>
      </lapsed>'
DECLARE @hdoc INT
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
SELECT *
FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')
       WITH (Nimi VARCHAR(40), Pikkus SMALLINT)
EXEC sp_xml_removedocument @hdoc
```



	Nimi	Pikk...
1	Margus	123
2	Salme	150
3	Janne	151
4	Matti	173
5	Peeter	169

Kui soovime jätta alles ka linnade nimed tuleks teha väike parandus schemasse:

```
DECLARE @doc xml
SET @doc =
      '<lapsed>
        <Linn Nimi="Tallinn">
           <Laps Nimi="Margus" Pikkus="123" />
           <Laps Nimi="Salme" Pikkus="150" />
        </Linn>
        <Linn Nimi="Tartu">
           <Laps Nimi="Janne" Pikkus="151" />
           <Laps Nimi="Matti" Pikkus="173" />
           <Laps Nimi="Peeter" Pikkus="169" />
        </Linn>
      </lapsed>'
DECLARE @hdoc INT
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
SELECT *
FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')
       WITH (   Nimi VARCHAR(40)
                , Pikkus SMALLINT
                , Linn VARCHAR(20) '../@Nimi')
EXEC sp_xml_removedocument @hdoc
```



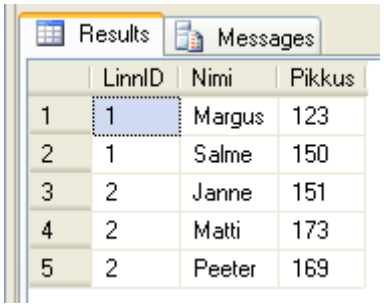
Results | Messages

	Nimi	Pikk...	Linn
1	Margus	123	Tallinn
2	Salme	150	Tallinn
3	Janne	151	Tartu
4	Matti	173	Tartu
5	Peeter	169	Tartu

PS! Olge ettevaatlikud. XML on tõusutundlik e. @nimi ei ole sama, mis @Nimi!

Saadud tabelitega võime teha, mida iganes soovime. Näiteks leiame linna nimede asemele nende koodid:

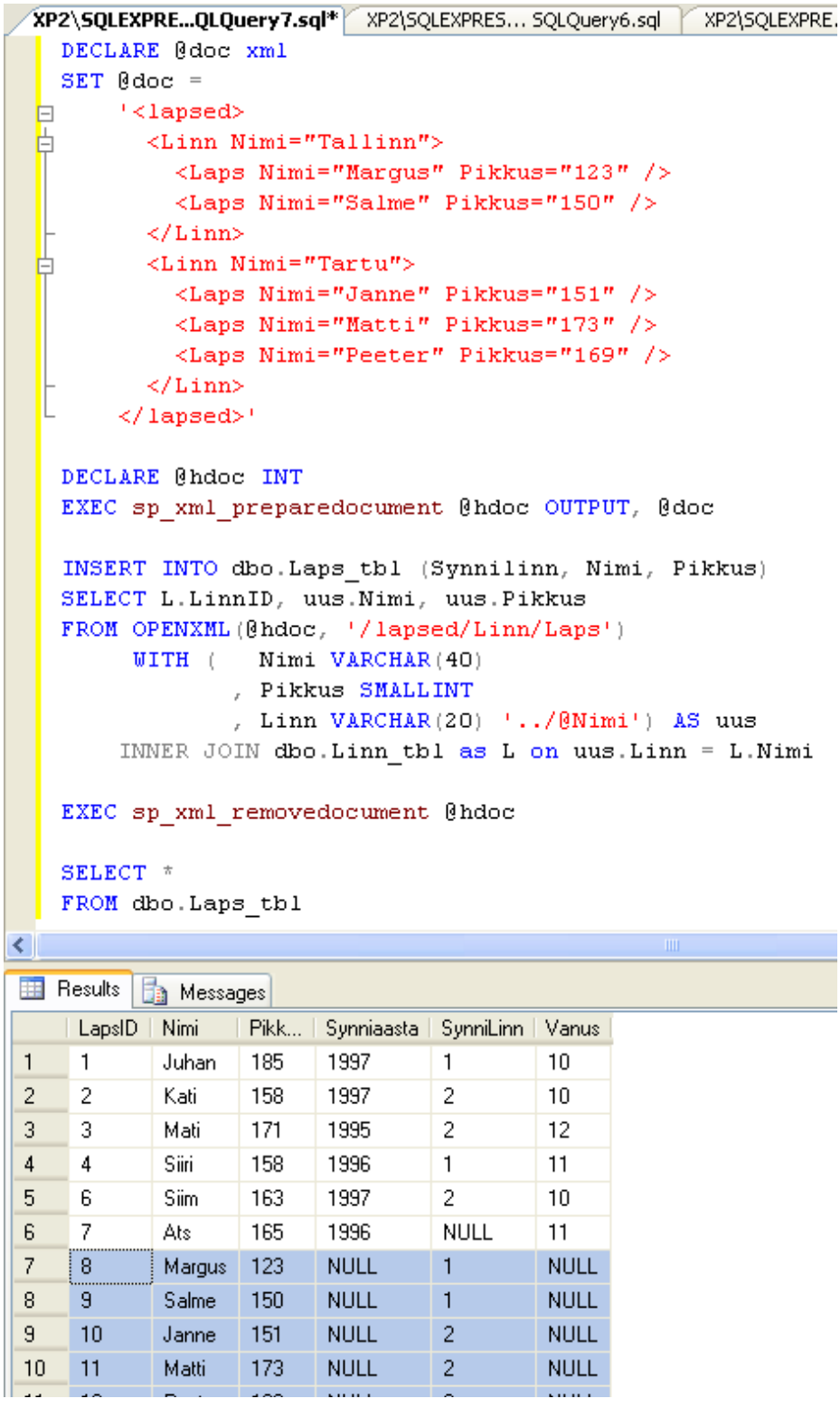
```
SELECT L.LinnID, uus.Nimi, uus.Pikkus
FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')
       WITH (   Nimi VARCHAR(40)
                 , Pikkus SMALLINT
                 , Linn VARCHAR(20) '../@Nimi') AS uus
      INNER JOIN dbo.Linn_tbl as L on uus.Linn = L.Nimi
```



Results | Messages

	LinnID	Nimi	Pikkus
1	1	Margus	123
2	1	Salme	150
3	2	Janne	151
4	2	Matti	173
5	2	Peeter	169

Või lisame saadud andmed Laps_tbl tabelisse:



Ülesandeid

- Väljasta kõikide tabelis olevate sõidukite andmed XMLina

- Katseta XML RAW ja AUTO võimalusi. Samuti juurelemendi määramist ning ELEMENTS täiendit

- Ühenda autode ja maakondade tabel ning väljasta sealsed tulemused XMLina maakondade kaupa

- Paiguta alampäringuga iga maakonna sisse selles maakonnas sõitvate kõige vanemate sõidukite margid XMLina

- Sisendiks autode loetelu XMLina. Väljasta OPENXMLi abil andmed tabelina.

- Sisendiks XML, kus autod maakondade kaupa. Väljundiks autod koos maakondade nimedega.

- Lisa etteantud XMList autod olemasolevasse tabelisse.

Varukoopia

Masinatega juhtub ikka õnnetusi. Mõni läheb rikki, teine varastatakse ära. Ja vahel piisab ainult vigaselt kirjutatud DELETE või UPDATE lausest, et paljust tähtsast ilma jääda. Kui aga andmetest kindlas kohas koopia(d) olemas, saab töö pärast taastamistoimetusi jätkuda.

Põhjalikum koopiate loomine ja haldamine on terve teadus: millal teha täielik koopia ning millal salvestada vaid erinevused. Ning kuidas taastada olukord kindlal kellaajal ja kuidas hilisematest muutustest vaid õiged välja valida. Kogu see uurimine jääb siinsest kirjutisest välja. Aga lihtsalt teadmiseks kõrvale, et vajaduse korral saab varukoopiamajanduse küllalt peenelt paika sättida. Siin ainult paar lihtsat käsku, millega saab baasist tervikuna koopia teha ning siis jälle tervikuna taastada, sest ka ilma seadistusteta tehtud varukoopia on tunduvalt parem kui andmete täielik häving.

Loomine

SQL-serveril on sobiv käsk olemas. Tuleb vaid määrata, millise nimega baas kuhu faili salvestatakse. Ja edasi võib juba edasi failiga toimetada.

```
BACKUP DATABASE baas1
TO DISK='d:\kodu\baas1.bak'
```

Taastamine

Kui varukoopiafail ilusti hoitud, siis endise seisu saab tagasi, lugedes varukoopia algse baasi kohale. Salvestushetkele järgnenud muutused lähevad küll kaduma, kuid vähemalt tallel hoitud põhiosa on võimalik ilusti tagasi saada.

```
RESTORE DATABASE baas1
```

```
FROM DISK='d:\kodu\baas1.bak'
```

Binaarandmete asemel on võimalik baasi ja tabelite koopiaid ka SQL-lausetena luua ja hoida. Selleks sobib näiteks SQL Server Management Studio pakutav võimalus vastavad skriptid genereerida. Andmete taastamiseks tuleb nad lihtsalt käima panna ja jällegi võib taastatud andmetest rahulolu tunda.

Kui soovitakse varukoopia põhjal täiesti uus baas importida, sel juhul käib toiming kahe sammuga. Kõigepealt faililoetelu loomine

```
RESTORE FILELISTONLY
FROM DISK='c:\jaagup\esimene.bak'
```

ning edasi luuakse baas ise, loetakse andmed sisse ning vanale baasinimele (siin juhul baas1) määratakse andmete hoidmise asukohaks uued failid.

```
RESTORE DATABASE proov2
   FROM DISK = 'c:\jaagup\esimene.bak'
   WITH MOVE 'baas1' TO 'c:\jaagup\proov2db.mdf',
   MOVE 'baas1_log' TO 'c:\jaagup\proov2db.ldf'
```

Nõnda võib loodetavasti julgesti oma andmetega toimetada nii, et ka suuremate tehniliste viperuste korral jäävad tähtsamad andmed alles ja kasutatavaks.

Ülesandeid

* Loo andmebaasist varukoopia.

* Vaheta koopia naabriga.

* Taasta naabri baas oma arvutisse. Kontrolli päringute ja käskluste

toimimist.

Kokkuvõte

Nagu näha, võib SQL-lausete abil suhteliselt lühidalt küllalt palju tarvilikku kirja panna ning alampäringute kaudu saab õige mitmekesiseid lauseid kokku. Enne, kui asuda usinasti omi andmetöötlusprogramme kirjutama, tasub uurida, kas sama tulemust mitte SQLi kaudu tunduvalt lihtsamalt kätte ei saa. See päringute koostamine on mõnes mõttes nagu matemaatiliste avaldiste lihtsustamine või malemäng, kus reeglid teada. Kuid hea lõpptulemuseni jõudmiseks tuleb vahel osata mitu käiku ette näha. Ning veel raskem on kindlaks teha, et vastavat ülesannet polegi võimalik olemasolevate vahenditega lahendada. Aga – kes püüab kõigest väest, saab üle igast mäest.

Andmetele ligipääs ADO.NET

Andmehoidla on meetod infoühikute hoidmiseks, mis kokku moodustavad informatsiooni. Üksikud infoühikud on seejuures suhteliselt kasutud, nad omandavad väärtuse kui nad panna konteksti teiste infoühikute juurde.

Andmete hoidmiseks on viis meetodit:

Meetod	Kirjeldus	Näide
Struktureerimata	Andmetel ei ole mingit loogilist järjekorda	Lihtsad märkmed ja kommentaarid
Struktureeritud, kuid mitte hierarhiline	Andmed on grupeeritud üksusteks, kuid üksused on organiseeritud vaid järjekorra alusel	Exceli tabelid, CSV failid jne
Hierarhiline	Andmed on organiseeritud puu kujule.	XML dokumendid
Relatsiooniline andmebaas	Andmed on organiseeritud tabelitesse, kus veergudes on konkreetset tüüpi andmed ja iga rida sisaldab kirjet. Tabelid on seotavad teiste tabelitega, kus leidub sarnaste andmetega veerge.	SQL Serveri andmebaasid, Accessi andmebaasid, Oracle andmebaasid
Objektorienteeritud andmebaas	Andmed on organiseeritud objektidena	

ADO.NET võimaldab andmeid hoida kõigil kirjeldatud viisidel ning ühendada kõikvõimalike väliste andmehoidlate/andmebaaside külge.

Andmete kasutamise keskkond võib olla nii ühendatud kui ka ühenduseta. ADO.NET võimaldab kasutada neid mõlemaid.

Andmebaasi kasutamine käib enamasti viiesammulise protsessina:

- Tuleb luua ühendus kasutades ühendusteksti (ConnectionString)

- Tuleb luua objekt, mis sisaldab andmebaasi käske
- Avada ühendus
- Käivitada käsk
- Sulgeda ühendus

Üsna pikalt oli ainukeseks andmete kasutamise võimaluseks ühendatud keskkond. Ühendatud keskkond on keskkond, kus kasutaja või programm on pidevalt ühendatud andmeallika külge.

Ühendatud keskkonnal on mitmeid eeliseid:

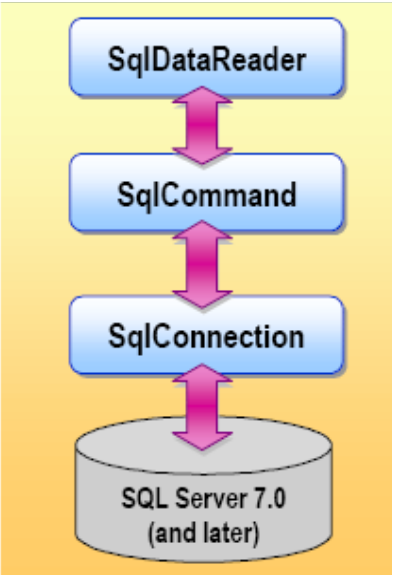
- Turvalist keskkonda on lihtsam hallata
- Konkurentsi on lihtsam kontrollida
- Andmed on värsked

Samas on ühendatud keskkonnal ka mõned puudused:

- On vaja pidevat võrguühendust
- Laiendatavus on raskendatud

Ühendatud keskkonna kasutamiseks on XxxDataReader klass. Programmi loogika oleks siis järgmine

- Ava ühendus
- Käivita käsk
- Töötle lugeja poolt tagastatavad kirjed
- Sulge lugeja
- Sulge ühendus



Koos Interneti arenguga on hakanud tavaliseks muutuma ühenduseta keskkonnad. Ühenduseta keskkond on keskkond, kus kasutaja või programm ei ole pidevalt ühendatud andmeallikaga. Kõige tüüpilisemaks näiteks on mobiilsete seadmete (nt sülearvuti) kasutajad, kes võtavad

mingi hulga andmeid endaga kaasa ning kui taas levisse satuvad, siis sünkroniseerivad andmeid.

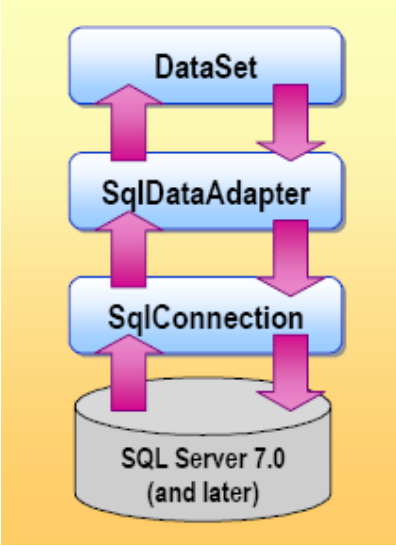
Ühenduseta keskkonnal on mitmeid eelised:

- on võimalik töötada andmetega siis, kui see kõige paremini sobib ning on võimalik ühendada andmeallika külge siis, kui see on võimalik.
- Sel ajal kui ei kasuta ise andmeallikat, võivad seda teha teised kasutajad
- Oluliselt parem laiendatavus ja tööjõudlus

Samas on ka mõned puudused:

- Andmed ei ole alati kõige värskemad
- Muudatused võivad tekitada konflikte ja need tuleb lahendada

Ühenduseta keskkonna kasutamiseks on XxxDataAdapeter klass ning DataSet'id. Programmi loogika oleks järgmine:



- Ava ühendus
- Täida DataSet
- Sulge ühendus
- Töötle andmeid DataSet'is
- Ava ühendus
- Värskenda andmeid
- Sulge ühendus

ADO.NET on hulk klasse, mis on mõeldud andmetega töötamiseks. Andmetega seotud nimeruumid on:

System.Data	ADO.NET põhilised osad, mis võimaldavad kasutada ühenduseta keskkonda.

System.Data.Common	Lisavahendid ja näod, mis on .NET raamistikus realiseeritud
System.Data.SqlClient	SQL Serveri andmeallikas
System.Data.OleDb	OLE DB andmeallikas
System.Data.SqlTypes	Klassid ja struktuurid SQL andmetüüpide kirjeldamiseks
System.Xml	Klassid ja näod XML kujul andmete töötlemiseks

Andmeallika külge ühendumine

Ennem, kui saate hakata tegelema andmetega, peate fikseerima andmeallika. Andmeallikad on üks ADO.NET põhikomponentidest, mis võimaldavad programmil suhelda andmeid hoidvate süsteemidega. .NET raamistikuga on kaasas SQL Server .NET Data Provider (Optimeeritud SQL Serverite kasutamiseks alates SQL Server 7.0st) ja OLE DB .NET Data Provider (Võimaldab kasutada kõiki andmeallikaid). Lisaks on saadaval ka spetsiaalseid allikaid ODBC ja Oracle andmete kasutamiseks.

Iga andmeallikas pakub järgmiseid klasse:

- XxxConnection nt SqlConnection SQL Serveriga suhtlemiseks. Suhtluse kontrollimiseks on sealjuures veel XxxTransaction, XxxExeption ja XxxError klassid.
- XxxCommand nt SqlCommand käivitab andmeallikas käsu. Käsu parameetreid saab kontrollida läbi XxxParameter klassi.
- XxxDataReader nt SqlDataReader avab ainult loetava andmevoo, tavaliselt mingi SELECT lause tulemus
- XxxDataAdapter nt SqlDataAdapter kasutab SqlCommand objekte DataSeti täitmiseks ning haldab ka DataSetis toimuvaid muudatusi
- XxxPermission – Õigustega seonduv

Andmeallika leidmiseks tuleb ühenduse loomisel ära määrata ühendustekst. Ainukene erinevus OLE DB ja Sql andmeallika vahel seisneb selles, et OLE DB puhul tuleb täpsustada, mis sorti andmeallikaga on tegemist. Järgnevalt mõned näited:

Andmebaas	Ühendustekst
SQL Server 6.5	`Provider=SQLOLEDB;Data Source=London;Initial Catalog=pubs;User ID=sa;Password=2389;`
Oracle Server	`Provider=MSDAORA;Data Source=ORACLE8I7;User ID=OLEDB;Password=OLEDB;`
Accessi andmebaas	`Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\bin\LocalAccess40.mdb;`

Ühendusteksti luues on vaja määrata:

- Provider (kasutades OLE DB andmeallikat) – mis tüüpi andmebaasiga on tegemist
- Data Source – Millise andmebaasiserveri poole pöördutakse
- Initial Catalog – Millist andmebaasi serverist soovitakse
- User ID (uid) ja Password (pwd) – millise kasutaja õigustes andmebaasiga suheldakse ning tema parool.
- Integrated Security - kui kasutajanime ja parooli ei taha sisse kirjutada, võib kasutada Windowsi autentimist. Kuid see vajab pisut ettevalmistusi ka Windowsi seadistuse poole pealt.
- Presist Security – Kui on False, siis tundliku infot nagu nt kasutajanimi avatud ühenduses ei vahetata.

Lisaks neile parameetritele võib lisada veel määranguid selle kohta, kui suurte tükkidena andmeid vahetatakse, kaua see ühendus võib avatuks jääda jne.

Hostingu keskkonnas tuleb kasutada ühendusteksti järgmisel kujul:

```
packet size=4096;uid=KASUTAJANIMI;pwd=PAROOL;data
source=dotnet;persist security info=False;initial
catalog=KASUTAJANIMI
```

Ühendusteksti saab määrata XxxConnenction objekti luues (konstruktoris)

```
SqlConnection conn = new
SqlConnection(”uid=KASUTAJANIMI;pwd=PAROOL;data
source=dotnet;persist security info=False;initial
catalog=KASUTAJANIMI”);
```

või tagantjärele, kasutades ConnectionString omadust. Tagantjärgi saab ConnectionString omadust muuta võid juhul, kui ühendus on suletud.

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = ”uid=KASUTAJANIMI;pwd=PAROOL;data
source=dotnet;persist security info=False;initial
catalog=KASUTAJANIMI”;
```

Kui ühendus on algväärtustatud, saate Open ja Close meetodite abil seda ühendust avada ja kinni panna.

Ühenduste sulgemine on oluline, kuna neid ei sulgeta automaatselt kui ühendust hoidev muutuja väljub vaateväljast! Seega peaks andmetega töötamine käima järgnevalt:

```
// Algväärtustame ühenduse
SqlConnection conn = new SqlConnection();
conn.ConnectionString = ”uid=KASUTAJANIMI;pwd=PAROOL;data
source=dotnet;persist security info=False;initial
catalog=KASUTAJANIMI”;
// avame ühenduse
conn.Open();
// mingid tegevused selles ühenduses
// sulgeme ühenduse e. peatame andmevahetuse selles ühenduses
// kui ühendus pärines basseinist siis pannakse ühendus basseini
tagasi
conn.Close();
// Eemaldame ühenduse basseinist
conn.Dispose();
// hävitame ühenduse objekti ja lubame prügiautol mälu vabastada
conn = null
```

Kui soovite reageerida ühenduse staatuse muutmisele, võite kinni püüda StateChange sündmuse. Staatuse kontrollimiseks on ühenduse küljes State omadus.

Kuna ühenduse loomisel või andmete toomisel võib ette tulla mitmeid probleeme, tuleks kogu protseduuri teha kindlasti `Try...Catch` konstruktsiooni sees ning kinni tuleks püüda kõik probleemid, millele teie programm oskab viisakad lahendused pakkuda.

Lisaks probleemidele, mida tuvastab ADO.NET, võivad probleeme tekitada ka serverisse saadetud käsud. Sqli serveri poolt tekitatud veateateid saab hallata püüdes kinni SqlException’i. SqlExceptionist saab kätte SqlError klassi, mille omadustest on võimalik välja lugeda, millega tegemist:

Class	Vea kriitilisus
LineNumber	Rida, mis põhjustas vea
Message	Vea tekst
Number	Vea numbriline kood

Vigade kirjeldusi ja võimalikke lahendusi on võimalik otsida SQL Serveri spikrist BooksOnline.

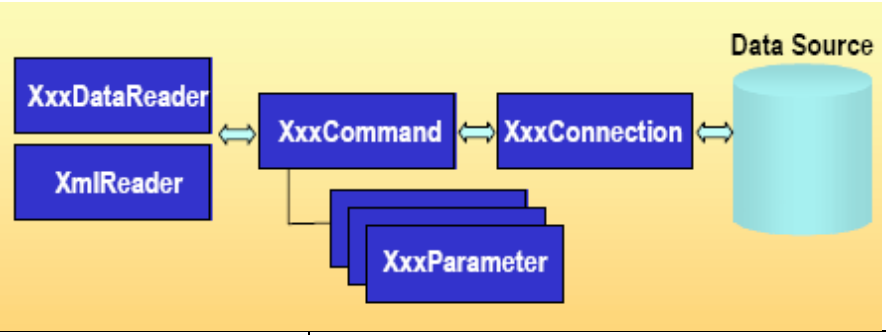
Töötamine andmebaasiga ühendatud keskkonnas

Kuigi ühenduseta keskkonnal on mitmeid häid omadusi, on siiski olukordi, kus on mõttekam jätta vahele DataSet'idest tulenev keerukus ja käivitada käsud otse andmebaasis.

Järgnevalt mõned olukorrad, kus on enamasti otstarbekam kasutada ühendatud keskkonda:

- Päringute tegemine andmetele, mis on rakenduses ainult loetavad. Sh otsingud andmebaasist.
- ASP.NET rakenduste disain, kus andmeid edastatakse vaid üks kord nt otsingu tulemuse näitamine
- Päringute käivitamine, mis tagastavad vaid ühe väärtuse või ei tagasta üldse midagi.
- Andmebaasi struktuuri muutmine

Kui soovite käivitada käske, mis ei tagasta andmetabelit nt tabelite loomine või muutmistegevused, siis ei ole võimalik DataSet'i kasutada ning tuleb kasutada ühendatud keskkonnas käivitatavaid käske.



Klass	Kirjeldus
XxxConnection	Tekitab ühenduse soovitud andmeallikaga. Nt SqlConnection loob ühenduse Microsoft SQL Serveriga
XxxCommand	Käivitab käsu olemasolevas ühenduses. Nt SqlCommand käivitab käsu Microsoft SQL Serveris
XxxDataReader	Ainult edasiloetav andmejada andmeallikast. Nt SqlDataReader klass loeb andmeid Microsoft SQL Serverist. Lugeja saab tekitada käsu XxxCommand (SqlCommand) meetodiga ExecuteReader.

	Enamasti on tegemist SELECT lausete tulemustega
XxxXMLReader	Pakub kiiret, ainult edasi lugevat ligipääsu XML andmetele

XxxCommand

Käsu objekt annab ühendatud keskkonnas otsese ligipääsu andmetele. Käsu objekti saab kasutada järgmiste tegevuste tarbeks:

- SELECT lause käivitamiseks, kui tulemuseks on 1 väärtus
- SELECT lause käivitamiseks, mis tagastab rea andmeid
- DDL (Data Definition Language – Andmebaasi kirjeldamislaused) lausete käivitamine nt tabelite loomine, muutmine, protseduuride loomine jne
- DCL (Data Control Language – Andmete ligipääsu kontrollimise laused) lausete käivitamiseks nt andmete lugemise keelamine või lubamine
- XML formaadis andmete lugemine

Käsu objekti omadused sisaldavad käsu käivitamiseks kogu vajalikku informatsiooni:

- Connection – Viide ühendusele, kus käsk käivitatakse
- CommandType – Käsu tüüp: Text, StoredProcedure, TableDirect
- CommandText – Käsk ise st SQL lause või StoredProcedure nimi
- Parameters – Kollektsioon parameetritest. Vastavalt vajadusele võivad parameetrid puududa ning neid võib olla ka mitmeid.

Kui käsu omadused on määratud, tuleb käsk käivitada, kutsudes välja mõne meetodi käsu objekti küljest.

Meetod	Kirjeldus
ExecuteScalar	Käivitab käsu, mis tagastab ühe väärtuse
ExecuteReader	Käivitab käsu, mis tagastab mingi hulga ridasid
ExecuteNonQuery	Käivitab käsu, mis otseselt midagi ei tagasta. Nt tabelis

	ridade kustutamine või muutmine. Tulemusena tagastatakse ridade arv, mida käsk mõjutas
ExecuteXmlReader (ainult SqlCommand'i puhul)	Käivitab käsu, mis tagastab XML andmed

Käsu objekti loomise võib korraldada nt järgmise koodireaga:

```
SqlCommand cmd = new SqlCommand();
```

Sellise tühja käsuga pole loomulikult palju peale hakata, seega järgnevalt tuleks väärtustada kõik olulisemad omadused: käsu tüüp, käsk ja ühendus.

```
cmd.CommandType = CommandType.Text;
cmd.CommandText = "UPDATE Toode SET Hind = Hind * 1.1";
cmd.Connection = conn;
```

Käsu käivitamiseks tuleb välja kutsuda sobiv meetod. Kuna hetkel on tegemist muutmislausega, mis tulemust ei tagasta, siis oleks kõige sobivamaks ExecuteNonQuery meetod. Enne käsu käivitamist tuleb avada ka andmebaasi ühendus.

```
cmd.Open();
int read = cmd.ExecuteNonQuery();
cmd.Close();
Muutujasse read salvestatakse muudetud ridade arv ning seda saab
kasutada edaspidises koodis. Kui sellist informatsiooni vaja ei ole,
siis võib selle muutuja omistamise ära jätta ning käivitada käsu
järgmiselt:
cmd.ExecuteNonQuery();
```

Parameetrite kasutamine

SQL laused ja protseduurid võivad kasutada nii sisend- kui ka väljundparameetreid. Nende parameetrite kasutamiseks on käsu tüübile vastav XxxParameter klass. SqlCommand'i puhul on selleks SqlParameter.

Enne käsu käivitamist tuleb omistada väärtused kõigile sisendparameetritele.

Peale käsu käivitamist õnnestub lugeda väärtusi väljundparameetritest.

Järgnevalt täiendame eelpool moodustatud käsku nii, et muudetud saaks vaid ühe toote hind.

Selleks tuleb esmalt teha muudatus käsus, kus ütleme, et soovime vaid muuta selle toote hinda, mille määrame parameetriga @TooteKood.

```
cmd.CommandText =
"UPDATE Toode SET Hind = Hind * 1.1 WHERE ToodeID = @TooteKood";
```

Seejärel tuleb meil tekitada sobivate omadustega parameeter:

```
SqlParameter p = new SqlParameter("@TooteKood",SqlDbType.Int);
p.Direction = ParameterDirection.Input;
p.Value = 5;
```

Ning kleepida see parameeter käsu külge:

cmd.Parameters.Add(p);

Kui käsk vajab rohkem parameetreid, siis tuleb seda protseduuri korrata iga parameetriga.

Kui kõik parameetrid lisatud, siis võib käsu käivitada nii nagu varemgi.

Ridade lugemine väljundist (DataReader)

DataReader on kiire ainult edasi lugev tsükkel, mis loeb läbi andmeallikast tulevad read.

Kui käsu nt SqlCommand'i tulemuseks on tabel, siis kasutades SqlDataReader'it, saab vaadata, mis neis ridades kirjas.

DataReaderi tekitamiseks on käsu küljes ExecuteReader meetod. Käsuks võib olla nii SELECT lause kui ka StoredProtcedure.

Lisaks andmetele annab DataReader andmete kohta ka metainfot e. mis tüüpi andmetega on tegemist ning mis on väljade nimed.

Sql lause tulemuseks olevate ridade läbi käimiseks on DataReader'il meetod Read. Read meetod liigub tulemuses järgmisele reale. Kui Read meetod tagastab false, siis on kõik read läbi vaadatud. Sel hetkel oleks ka sobiv kutsuda välja Close meetod, mis sulgeb DataReaderi ja vabastab ühenduse.

Aktiivselt realt andmete lugemiseks on mitmeid võimalusi:

- Kasutada Item omadust. Item on kahtepidi indekseeritud massiiv e. te võite küsida väärtusi nii välja nime järgi lugeja["TooteID"] kui ka positsiooni järgi lugeja[3].

- Väärtusi saab lugeda ka GetDateTime, GetDouble, GetGuid, GetInt32 jne meetoditega, mis tagastavad teile vastava välja väärtuse teisendatuna konkreetsesse andmetüüpi.

- Viimase võimalusena on võimalik kasutada GetValues meetodit, mis tagastab objektide massiivi kõigist rea väljadest.

Üks salapärane väärtus andmebaasis on määramata väärtus NULL. Selleks, et kindlaks teha, kas mingil väljal on väärtus puudu, saate kasutada IsDbNull meetodit. Kui selle meetodi tulemuseks on true, siis on väljal väärtus puudu.

Järgnevalt üks näide, kuidas lugeda andmebaasist 10ne kõige kallima toote andmed ning trükkida need konsooli aknasse.

```
SqlCommand cmd = new SqlCommand();
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT TOP 10 ToodeID, Nimi, Hind " +
" FROM Toode ORDER BY Hind DESC";
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
while (rdr.Read()) {
      Console.WriteLine("{0}: {1} ({2})",
rdr.GetInt32(0),rdr.GetString(1),rdr.GetDecimal(2));
}
rdr.Close();
```

Transaktsioonid

Keerukate muutmistegevuste juures tuleb tihtipeale kasutada transaktsioone. Transaktsioon on tegevuste jada, kus kõik tegevused kas lõpevad edukalt või neid ei sooritata üldse.

Üheks lihtsamaks näiteks on pangaülekanne. Pangaülekanne koosneb kahest tegevusest: esmalt tuleb ühelt kontolt sobiv summa maha võtta ja seejärel tuleb teisele kontole sama summa juurde liita. Need tegevused peavad õnnestuma mõlemad, muidu läheb kusagilt raha kaduma või tekib lubamatult juurde.

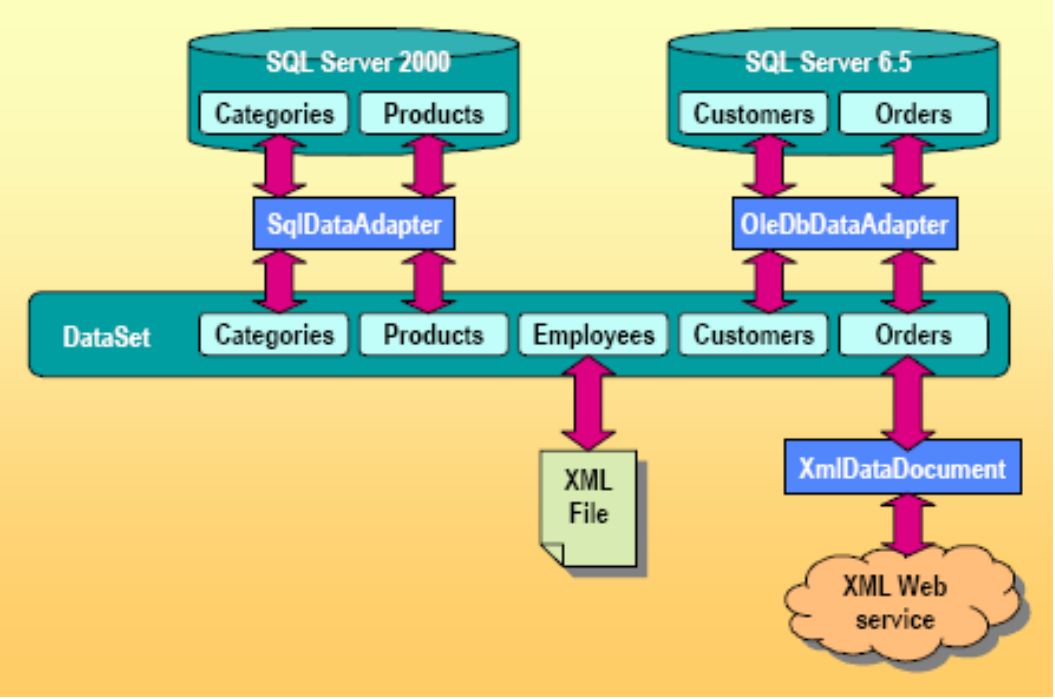
Selliste tegevuste haldamiseks on 2 varianti. Esimene on kasutada SQL lauseid (vaata eestpoolt), teine võimalus on korraldada transaktsioonid ADO.NET tasemel.

```
conn.Open();
SqlTransaction tran = conn.BeginTransaction();
SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.Transaction = tran;
cmd.CommandType = CommandType.Text;
try {
      cmd.Command.Text =
” UPDATE konto SET jaak = jaak – 100 WHERE omanik = 1”
      cmd.ExecuteNonQuery();
```

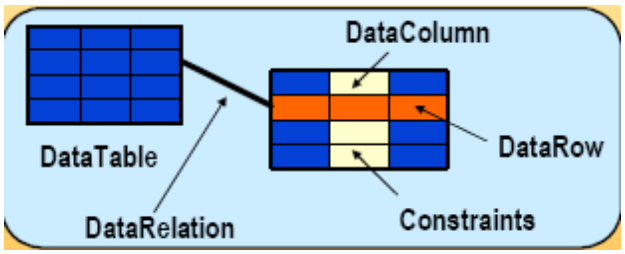
```
      cmd.Command.Text =
" UPDATE konto SET jaak = jaak + 100 WHERE omanik = 2"
      cmd.ExecuteNonQuery();
      tran.Commit();
}
catch {
      tran.Rollback();
}
finally {
      conn.Close();
}
```

Töötamine ühenduseta keskkonnas (DataSets)

Ühenduseta keskkonnas on võimalik töötada ilma püsiva andmebaasi ühenduseta ning kasutada paralleelselt mitmetest erinevatest andmebaasidest pärit andmeid.



DataSeti võib võtta kui arvuti mällu salvestatud andmebaasi, mis töötab analoogselt relatsioonilise andmebaasiga. Samas tuleb DataSeti loomisel olla ülimalt ettevaatlik, sest teie programme jooksutavatel arvutitel on mälu oluliselt vähem kui andmebaase hoidvatel serveritel kõvaketta ruumi.



Seega ei maksa DataSettidesse koguda mitte tervet andmebaasi, vaid ainult kõige olulisemad andmed, mida vajate oma programmis.

DataSeti tegemiseks tuleb kutsuda välja DataSet klassi konstruktor:

```
DataSet ds = new DataSet();
```

Tabeli lisamiseks DataSeti tuleb tekitada tabel ning lisada ta sobivasse DataSeti. Kuna ühes DataSetis võib tabeleid olla mitu, on kasulik anda igale tabelile oma nimi.

```
DataTable toode = new DataTable(”toode”);
ds.Tables.Add(toode);
```

Loomulikult on tegemist veel abstraktse tabeli objektiga, millel puuduvad struktuur ja andmed. Struktuuri tekitamiseks tuleb sellesse tabelisse lisada veerud. Lisame näiteks unikaalse koodivälja, toote nimetuse ja hinna.

```
DataColumn kood = toode.Columns.Add(”ToodeID”, typeof(Int32));
kood.AllowDBNull = false;
kood.Unique = true;
DataColumn nimi = toode.Columns.Add(”Nimi”, typeof(String));
nimi.AllowDBNull = false;
DataColumn hind = toode.Columns.Add(”Hind”, typeof(Decimal));
hind.AllowDBNull = false;
```

Kuigi koodi näol on tegemist igati sobiva väljaga tabeli primaarvõtmeks, ei käsitle DataSet seda kui primaarvõtit enne, kui oleme seda näidanud läbi tabeli PrimaryKey omaduse. Kuna primaarvõti võib olla ka kombinatsioon mitmest väljast, siis tuleb meie veerg paigutada massiivi.

```
toode.PrimaryKey = new DataColumn[] {kood};
```

Analoogselt võiks lisada ka piiranguid (Constraint) ja viiteid (Reference).

Andmetabelitesse saab lisaks reaalseid andmeid sisaldavatele väljadele lisada ka arvutatud välju. Nt kui teame, et hind on meil Eesti kroonides, siis võib tekkida vajadus esitleda hinda ka eurodes. Selleks võime lisada oma tabelisse arvutatud välja:

```
DataColumn hindEur = toode.Columns.Add(”HindEUR”, typeof(Decimal));
hind.Expression = ”Hind * 15,56”;
```

Loomulikult pole sellise tühja andmebaasiga midagi peale hakata. Selleks, et midagi huvitavat korda saata, tuleb DataSetis olevatesse tabelitesse ka andmeid lisada. Andmete lisamiseks on kaks moodust: me võime andmed ise tekitada ja rea kaupa tabelisse kirjutada või loeme andmed kusagilt andmebaasist.

Alustame käsitsi rea lisamisest. Selleks tuleb esmalt tekitada sobiva struktuuriga andmerea objekt.

```
DataRow rida = toode.NewRow();
```

Seejärel tuleb värskelt loodud ritta sisestada sobivad andmed. Väljade poole saab pöörduda nii välja nime kui ka positsiooni järgi.

```
rida[0] = 1;
```

```
rida[”Nimi”]=”Kapsas”;
rida[”Hind”]=3.3;
```

Kui rida sobivate väärtustega täidetud, lisame ta sobivasse tabelisse.

```
toode.Rows.Add(rida);
```

Ridasid on võimalik lisada ka läbi objektide massiivi nagu näha allolevas koodireas.

```
toode.Rows.Add(new Object[] {2, ”Porgand”, 5.0});
```

Tekkinud andmetabeli võime kuvada konsooliaknas nt järgmise for-lausega:

```
foreach (DataRow dr in toode.Rows) {
foreach (object o in dr.ItemArray)
Console.Write("{0}\t",o);
Console.WriteLine();
}
```

Loomulikult on olemas ka DataSetis olevate andmete kuvamiseks ning töötlemiseks mugavamaid vahendeid. Enne kui neid saame vaadata, tuleb aga üle vaadata graafilise kasutajaliidese (nt ASP.NET) põhitõed.

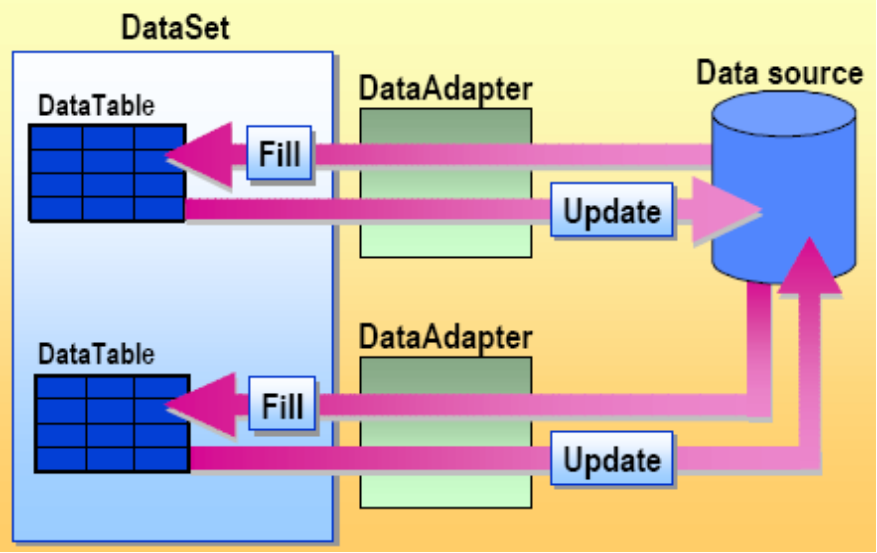
Ülesandeid

Koosta DataSet tabeliga autode andmete hoidmiseks. Mass tonnides arvutatakse kilogrammides olevate andmete põhjal automaatselt. Lisa mõned andmed. Trüki andmed ekraanile.

Olemasolevate andmete põhjal DataSeti loomine

Lisaks käsitsi andmete tekitamisele on võimalik DataSetis kasutada ka juba olemasolevaid andmeid. Olemasolevate andmete kasutamiseks on kaks võimalust:

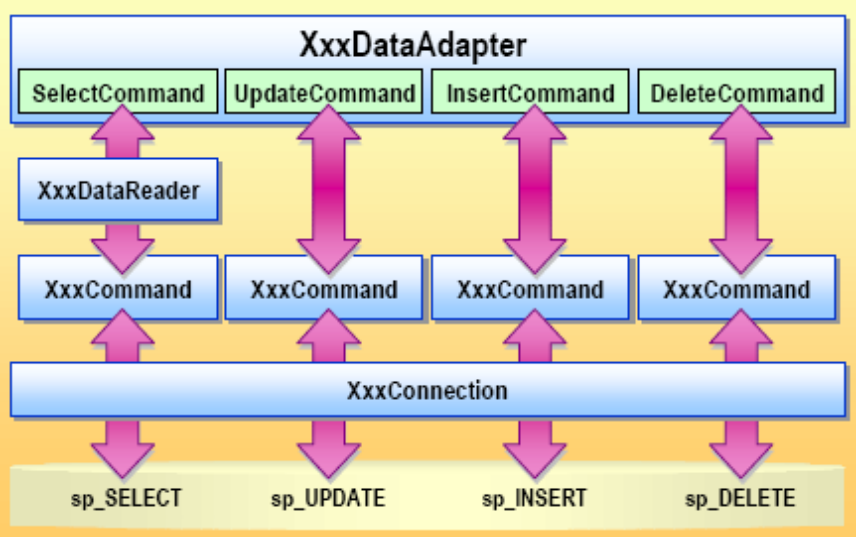
- kasutada sama metoodikat, mis käsitsi uute DataSettide loomisel ning andmed lisada rea kaupa kasutades XxxDataReader'it
- kasutada DataAdapter objekti abi



DataSet on oma olemuselt andmebaasis olevate andmete koopia. Tegemist on väga kasuliku objektiga andmete töötlemisel. Samas pole ilma reaalseid andmebaasis olevaid andmeid värskendamata sellest võimsast vahendist eriti kasu.

Andmetabelis olevate andmete sünkroniseerimiseks andmebaasiga on DataAdapter objekt. DataAdapter klass on kogumik andmebaasi käske ja ühendusi, mida saab kasutada tabelite täitmiseks e. lugemiseks andmebaasist (Fill) ja muudatuste ülekandmiseks andmebaasi (Update). Iga DataAdapter tegeleb vaid ühe tabeli andmetega. Kuna DataAdapter tegeleb otseselt andmebaasiga, tuleb valida DataAdapter vastavalt andmebaasile. .NET raamistikuga on kaasas kaks DataAdapteri klassi

- OleDbDataAdapter – igasuguste andmeallikatega suhtlemiseks
- SqlDataAdapter – Suhtlemiseks SQL Serveritega alates versioonist 7.0



XxxDataAdapteri kaudu on võimalik sooritada kõiki DML (Data Manipulation Language - andmetöötlus) tegevusi (andmete valimine, muutmine, lisamine, kustutamine).

DataAdapteri küljes on väga palju erinevaid omadusi ja meetodeid. Olulisemad neist on:

- SelectCommand – Käsk, millega tuuakse andmed andmebaasist andmetabelisse
- InsertCommand – Käsk, mis sisestab lisatud read andmebaasi
- UpdateCommand – Käsk, mis viib andmetabelis tehtud muudatused andmebaasi
- DeleteCommand - Käsk, mis kustutab andmetabelist kustutatud read ka andmebaasist
- Fill meetod – täidab andmetabeli so SELECT lause või StoredProtseduuri tulemus
- Update meetod – Kutsub vastavalt vajadusele välja vajalikud INSERT, UPDATE ja DELETE käsud

DataAdapteri kasutamiseks tuleb luua DataAdapteri objekt ning määrata vajalikud omadused.

```
SqlDataAdapter daTooted = new SqlDataAdapter();
SqlConnection yhendus = new SqlConnection(
”uid=KASUTAJANIMI;pwd=PAROOL;data source=dotnet;” +
”persist security info=False;initial catalog=KASUTAJANIMI”);
SqlCommand cmSelect = new SqlCommand(
"SELECT TOP 20 ToodeID, Nimi, Hind FROM Toode", yhendus);
daTooted.SelectCommand = cmSelect;
```

Selleks, et värskelt loodud adapteri abil täita andmetabel, tuleb välja kutsuda Fill meetod. Fill meetod on mitu korda üle kirjutatud/maskeeritud ning oskab täita nii DataSeti, DataTable’t kui ka mõnda konkreetset tabelit DataSetist.

Varem loodud DataSetis oleva tabeli Toode võime täita nt järgmise koodireaga.

```
int read = daTooted.Fill(ds, ”toode”);
```

DataAdapter on ise niipalju intelligente, et oskab enne andmete lugemist avada ühenduse andmebaasiga ning sulgeb selle, kui kõik andmed on käes.

Igal real DataTable’s on RowState omadus. Tegemist on ainult loetava omadusega, mis näitab, kas see rida on peale viimast andmete värskendamist muudetud, lisatud või kustutatud.

RowState võib omada järgnevaid väärtuseid:

RowState väärtus	Selgitus
DataRowState.Added	Uus rida
DataRowState.Deleted	Kustutatud rida
DataRowState.Modified	Real olevaid andmeid on muudetud
DataRowState.UnChanged	Andmeid ei ole muudetud

DataSet hoiab iga rea kohta kahte andmete komplekti: hetkel kehtivad andmed ja esialgsed andmed. DataSeti kasutades pääsete ligi mõlemale andmete komplektile.

DataRowVersion omaduse väärtuseks saab vastavalt vajadusele panna, kas DataRowVersion.Current või DataRowVersion.Original.

Järgnev näide vaatab läbi DataSet’is oleva tabeli kõik read ning näitab rea staatust. Kui rida on uus või muutmata, näidatakse andmete hetkel kehtivat versiooni. Kui rida on kustutatud, näidatakse esialgseid andmeid e. andmed, mis kustutati. Kui andmed on muudetud, näidatakse nii esialgseid kui ka uusi andmeid.

```
foreach (DataRow row in ds.Tables["toode"].Rows)
{
  Console.WriteLine("Rea staatus: " + row.RowState);
  switch (row.RowState){
    case DataRowState.Added:
    case DataRowState.Unchanged:
      Console.WriteLine("Hetkel kehtivad andmed\n" +
        row["ToodeID", DataRowVersion.Current] + ", " +
        row["Nimi", DataRowVersion.Current]+ ", " +
        row["Hind", DataRowVersion.Current] + "\n\n");
      break;
    case DataRowState.Deleted:
      Console.WriteLine("Esialgsed andmed\n" +
        row["ToodeID", DataRowVersion.Original] + ", " +
```

```
        row["Nimi", DataRowVersion.Original]+ ", " +
        row["Hind", DataRowVersion.Original] + "\n\n");
      break;
    case DataRowState.Modified:
      Console.WriteLine("Esialgsed andmed\n" +
        row["ToodeID", DataRowVersion.Original] + ", " +
        row["Nimi", DataRowVersion.Original]+ ", " +
        row["Hind", DataRowVersion.Original] );
      Console.WriteLine("Hetkel kehtivad andmed\n" +
        row["ToodeID", DataRowVersion.Current] + ", " +
        row["Nimi", DataRowVersion.Current]+ ", " +
        row["Hind", DataRowVersion.Current] + "\n\n");
      break;
    }
}
```

Selleks, et muudatused jõuaksid ka andmebaasi, tuleb DataAdapteri küljes ära määrata vajalike tegevuste käsud InsertCommand, UpdateCommand, DeleteCommand.

```
SqlCommand cmInsert = new SqlCommand(
     "INSERT Toode (Nimi, Hind) VALUES (@Nimi, @Hind)", yhendus);
cmInsert.Parameters.Add(new SqlParameter("@Nimi",
    SqlDbType.NVarChar, 100, ParameterDirection.Input, false,
    0, 0, "Nimi", DataRowVersion.Current, null));
cmInsert.Parameters.Add(new SqlParameter("@Hind",
    SqlDbType.Money, 8, ParameterDirection.Input, false,
    0, 0, "Hind", DataRowVersion.Current, null));
daTooted.InsertCommand = cmInsert;
SqlCommand cmUpdate = new SqlCommand(
    "UPDATE Toode SET Nimi = @Nimi, " +
    "Hind = @Hind WHERE (ToodeID = @ToodeID)", yhendus);
cmUpdate.Parameters.Add(new SqlParameter("@ToodeID",
    SqlDbType.Int, 4, ParameterDirection.Input, false,
    0, 0, "ToodeID", DataRowVersion.Original, null));
cmUpdate.Parameters.Add(new SqlParameter("@Nimi",
    SqlDbType.NVarChar, 100, ParameterDirection.Input, false,
    0, 0, "Nimi", DataRowVersion.Current, null));
cmUpdate.Parameters.Add(new SqlParameter("@Hind",
    SqlDbType.Money, 8, ParameterDirection.Input, false,
    0, 0, "CustomerID", DataRowVersion.Current, null));
daTooted.UpdateCommand = cmUpdate;
SqlCommand cmDelete = new SqlCommand(
    "DELETE Toode WHERE ToodeID = @ToodeID", yhendus);
cmDelete.Parameters.Add(new SqlParameter("@ToodeID",
    SqlDbType.Int, 4, ParameterDirection.Input, false,
    0, 0, "ToodeID", DataRowVersion.Original, null));
daTooted.DeleteCommand = cmDelete;
```

DataSetis olevate muudatuste salvestamine andmebaasi peaks käima 4 sammuga.

- Kutsuda välja GetChanges meetod, mis tekitab uue DataSeti, millesse koondatakse vaid need kirjed, mis on muutunud. Muutunud kirjeid on võimalik leida vastavalt

muudatuse tüübile. GetChanges meetodist on kasu juhul, kui soovitakse kontrollida, mis järjekorras muudatused andmebaasi salvestada.

- Kutsuda välja iga vajaliku DataAdapteri Update meetodi, et muudatused jõustuksid andmebaasis

- Kutsuda välja Merge meetodi mis ühendab kaks DataSeti taas üheks. Vajalik, kui GetChanges abil eraldatud DataSetist mingi osa.

- Kasutada AcceptChanges meetodit, et kinnitada muudatused DataSetis

Koodis näeb see protseduur välja järgnev:

```
if (ds.HasChanges()) {
DataSet dsTemp = ds.GetChanges();
daTooted.Update(dsTemp,"toode");
ds.Merge(dsTemp);
ds.AcceptChanges();
}
```

Loomulikult on võimalik küsida muudatusi iga tabeli kohta ning sealt tehtud muudatuse tüübi kohta eraldi. Samuti on võimalik muudatused kinnitada lisaks DataSet tasemele ka tabeli või tabeli rea tasemel.

Kui töötate andmetega ühenduseta keskkonnas, võivad tekkida andmete muutmisel konfliktid. ADO.NET kasutab ühenduseta keskkonnas optimistlikku lukustamist st. lukk eemaldatakse kohe, kui andmed on loetud, st andmete lugemise ja rakenduse poolt tehtud muudatuste salvestamise vahel võib keegi neid samu andmeid muuta.

Vigadele reageerimiseks pakuvad DataSet, DataTable ja DataRow klassid omadust HasErrors. Läbi selle omaduse on võimalik tuvastada, millised andmed on sattunud konflikti. DataRow klassil on lisaks olemas veel GetColumnsInError meetod, mis tagastab konkreetsed väljad, mis antud real on konfliktis.

Konfliktide tekkimisel oleks mõistlik uuendusi mitte peale suruda, vaid paluda kasutajal antud konfliktid lahendada. Samuti on võimalik konfliktsed muudatused tagasi kerida, selleks võib kutsuda välja RejectChanges meetodi konfliktis oleva DataSeti'i, DataTable või DataRow küljest.

Kui soovitakse kasutaja poolt tehtud muudatused tühistada ja lugeda andmebaasist asemele värsked andmed, tuleks DataSet või DataTable puhastada kasutades Clear meetodit ning seejärel täita uuesti DataAdapteri Fill meetodi abil.

Järgnevas näites tühistatakse kõik kasutaja poolt tehtud konfliktsed muudatused. Selleks ütleme adapterile, et ta jätkaks muudatustega ka peale vea tekkimist.

```
daTooted.ContinueUpdateOnError = true;
daTooted.Update(ds);
```

Kui Update käsk lõpetab, siis on kõik mitte konfliktsed kirjed ära muudetud ning konfliktsed jäänud muutmata. Et ka kasutajal oleks mingi ülevaade toimunust, loeme ette, millistes tabelites, millistel ridadel ja väljadel probleem tekkis ning tühistame tehtud muudatused.

```
if(ds.HasErrors){
  foreach(DataTable table in ds.Tables){
    if(table.HasErrors){
      Console.WriteLine("Probleem tabelis '{0}'.", table.Name);
      foreach(DataRow row in table.Rows){
        if(row.HasErrors){
          Console.WriteLine(
                      "Probleem tootega nr {0}.", row["ToodeID"]);
          Console.WriteLine("Vea kirjeldus: {0}", row.RowError);
          foreach(DataColumn col in row.GetColumnsInError()){
            Console.WriteLine(
                        col.ColumnName, "Probleem selle väljaga");
          }
          Console.WriteLine("Muudatus tühistati!\n");
          row.ClearErrors();
          row.RejectChanges();
        }
      }
    }
  }
}
```

Kui probleemid lahendatud, lubame DataSetil muudatused salvestada:

```
daToode.AcceptChanges();
```

Kuigi selline automaatne muudatuste tagasi lükkamine tagab programmi töö ka konfliktide tekkimise korral, on enamasti mõistlik lasta kasutajal otsustada, kas peale jäävad tema tehtud muudatused, või need andmed, mis on andmebaasis.

Lisaks eelpool mainitule on selle värskendamise protseduuri juures veel teine konks: nimelt kui andmebaasis on vahepeal midagi muudetud, siis neid muudatusi me oma DataSeti ei loe. Muudatuste lugemiseks on kaks meetodit: kas leiate mingi kavala SQLi abil read, mis on vahepeal muutunud ja vahetate need välja/lisate juurde või teete DataSeti tühjaks ja täidate uuesti. Viimast on oluliselt lihtsam rakendada. Selleks tuleb AcceptChanges asendada järgmiste meetodi väljakutsetega:

```
ds.Clear();
daToode.Fill(ds, "toode");
```

Kuigi DataSet on mõeldud peamiselt andmete hoidmiseks ühenduseta keskkonnas, oskavad mitmed Windowsi ja ASP.NET graafilise liidese komponendid (nt DataGrid jt) neid andmeid kuvada ja pakuvad juurde ka sorteerimise, muutmise jt võimalusi. Sellest aga lähemalt juba ASP.NET juures leheküljel **Tõrge! Järjehoidjat pole määratletud.**.

Ülesandeid

- Täida DataSet autode tabelist tulevate andmetega
- Muuda andmeid DataSetis
- Kirjuta muutused baasis olevasse andmetabelisse
- Katseta, mis juhtub, kui baasis olevaid andmeid on vahepeal muudetud. Püüa muutusi hallata, lasta kasutajal valida mida teha.

XML

Extensible Markup Language ehk XML on lihtne ja väga paindlik tekstiformaat, mis baseerub SGML'il (Standard Generalized Markup Language). XML töötati välja XML-töögrupi (algselt tuntud kui SGMLi redaktsiooniliselt läbivaatava nõukogu – SGML Editorial Review Board) poolt, mis loodi World Wide Web Konsortsiumi patronaaži all 1996. aastal. Töögruppi juhtis Jon Bosak, Sun Microsystemsist aktiivses koostöös W3C poolt organiseeritud XML-erihuvigrupiga (varem tuntud SGML-töögrupina).

XML'i loomise eesmärgid:

- XML peab olema kasutatav üle Interneti.
- XML peab olema loetav nii inimesele kui arvutile.
- XML rakenduste ampluaa peab olema võimalikult lai: sisestamine, lehitsemine, otsing, infotöötlus, sisutöötlus.
- XML ühildub SGMLga
- XML dokumentide töötlemise programme peab olema lihtne kirjutada
- XML disain peab olema formaalne ja kompaktne
- XML dokumentide genereerimine peab olema lihtne. Kuigi XML dokumentide koostamiseks kasutatakse enamasti spetsiaalseid redaktoreid, peab nende tekitamine olema võimalik mistahes redaktoriga.
- XML märgistuse napisõnalisuse nõue pole tähtis.

Aastal 1998 tuldi välja XML'i versiooniga 1.0, millele lisati väiksed täiendused aastal 2000 ning see versioon on ka hetkel ainukene.

XML failide loomiseks ei ole vaja spetsiaalseid programme, piisab vaid programmist, milles on võimalik tööd salvestada tekstifailina.

Kui RTF ja HTML failid on orienteeritud välisilmele ehk sellele, kuidas dokumente inimesele näidata, siis XML on orienteeritud andmete paremale kirjeldamisele e. kuidas andmeid võimalikult hästi programmidele arusaadavaks teha. Sellest tulenevalt on nendel failiformaatidel ka mõned olulised erinevused. RTF ja HTML failis paiknevad andmed ja nende kujundus läbisegi, XML'is moodustatakse eraldi failid nii andmetele, nende kujundusele ja struktuurile. See muudab XML andmete programse kasutamise oluliselt

lihtsamaks, kui on seda RTF või HTML andmete töötlemine. Ühtlasi võimaldab anda samadele andmetele erinevaid kujundusi lihtsamalt.

XML’i kirjutamise reeglid

XML’i fail koosneb kahest osast: faili alguses on deklaratsioonid selle faili töötlemiseks ning sellele järgnevad andmed.

Reeglid

- XML-dokumendi alguses peab olema näidatud, mis versiooni kasutatakse (töö kirjutamise hetkel on ainukeseks versiooniks 1.0)

```
<?xml version=”1.0”?>
```

- XML-dokument peab omama ainult üht juurelementi, mis sisaldab kõiki teisi elemente dokumendis

```
<juur>
      <laps>
            <alamlaps>...</alamlaps>
      </laps>
</juur>
```

- Igal elemendil (ka tühjal) peab olema algusmärgis ja lõppmärgis

Algusmärgis	Sisu	Lõppmärgis
`<nimi>`	`Ants`	`</nimi>`

- XML’is kasutatavad atribuudid peavad alati olema jutumärkide või ülakomade vahel. Ei ole vahet, kumba kasutada, kuid oluline on see, et alustav ja lõpetav märk oleksid samad. Seega võib juhtuda, et ühe elemendi üks atribuut on kirjutatud ülakomade, teine jutumärkide vahele.

- XML’is tehakse vahet suur-ja väiketähtedel.
  Näiteks `<kuupäev>` ja `<Kuupäev>` on erinevad elemendid

- Alamelement peab lõppema enne peaelementi. Kui HTMLi vanemates versioonides on lubatud nt järgmine konstruktsioon , siis XMLis sellist asja olla ei tohi ehk korrektne oleks

Kui XML fail vastab eelpool loetletud reeglitele, on tegemist trimmis (Well Formed) XML’iga.

XML'i elemendid

XML-elementidele nimede panemisel kehtivad järgmised reeglid:

- element esitatakse tagidena. Tage kirjutatakse järgmiselt: <taginimi>sisu</taginimi>. Esimest nimetatakse alustavaks tagiks, teist lõpetavaks.

- nimi võib sisaldada tähti, numbreid ja mõningaid sümboleid (sümbolit „:" ei tohi kasutada, kuna see on kasutusel nimeruumidele viitamisel);

- nimi ei tohi alata numbri või kirjavahemärgiga;

- nimi ei tohi alata kombinatsiooniga xml (samuti ka XML või Xml jne);

- nimi ei tohi sisaldada tühikut;

- elementide nimedes tehakse vahet suurtel ja väikestel tähtedel

- teinekord võib elementidel sisu puududa. Siis kasutatakse nn tühje elemente
  Näiteks element `<dokument failinimi="arve.doc"></dokument>` on sama, mis element `<dokument failinimi="arve.doc"/>`

Sisu poolest võib elemendid jagada kolmeks:
- tühjad elemendid
- lihtsad elemendid
- keerukad elemendid

Tühjad elemendid: tühjadeks nimetatakse elemente, millel sisu puudub ja nende esitamiseks on kaks võimalust:

- kasutatakse alustatavat ja lõpetavat tagi, nii et nende vahele ei jää midagi
  `<kontakt></kontakt>`

- kasutada spetsiaalset tagi, kus tagi lõpp sisaldub alustavas tagis
  `<kontakt />`

Lihtsad elemendid: elemendid, kus alustava ja lõpetava tagi vahel on mingi lihtne väärtus, nt sõna, lause, kuupäev, number.

```
<nimi>Mati</nimi>
```

Keerukad elemendid: elemendid, mille sisuks on teised elemendid

```
<firma>
```

```
      <nimi>AS Virumaa Mets</nimi>
      <aadress>Metsa 7</aadress>
      <regnumber>0011223344</regnumber>
      <faks />
</firma>
```

Loetavuse parandamiseks võib kasutada treppimist, sest XML'i parserid ignoreerivad tühiruumi[1]

Elementidele saab lisada ka atribuute.

Atribuudid

Atribuute on mõistlik kasutada andmete metadata jaoks (ehk atribuut on andmed andmete kohta). Atribuudid kirjutatakse elemendi alustava tagi sisse. Ühe elemendi sees peavad olema kõik atribuutide nimed erinevad. Atribuudid kirjutatakse kujul: atribuudinimi=väärtus. Väärtused peavad olema kas jutumärkide või ülakomade vahel.

Näiteks:

```
<isik sugu="mees">Ants Aavik</isik>
```

Atribuudinimedele kehtivad samad reeglid, mis elementide nimedele.

Võib tekkida küsimus, milliseid andmeid esitada elementidena, milliseid atribuutidena? Sellele küsimusele ühtset vastust ei ole, kõik sõltub vajadustest, kuid heaks tavaks on see, et atribuutides ei ole mitte andmed, vaid lisainfo andmete kohta ehk metadata. Samuti pannakse sinna vahel ka andmed, mis on lühikesed ning mille puhul on aimata, et neid pole põhjust hiljem laiendada (osadeks jaotada).

[1] tühiruum on tühikud, tabulaatorid, Enterid

XHTML

XHTML on XML'i reeglite järgi kirjutatud HTML. Täiendavalt on kokkulepe, et kõik tag'id kirjutatakse väikeste tähtedega. Kui soovite kontrollida, kas teie poolt valmistatud HTML dokument on ka XHTML dokument, siis võite seda teha W3 portaalis paikneva DTD dokumendi abil:

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

Lehestiku kontrollimiseks sobib valideerimisteenus aadressil http://validator.w3.org

Nimeruum

Mõnikord võib tekkida situatsioon, kus mitu osapoolt kasutavad XML-dokumentides ühte ja sama elementi erinevas tähenduses. Seetõttu on vaja eristada, millisesse sõnavarasse üks või teine märgis kuulub. Lahenduse antud probleemile toob XML-nimeruum (XML namespace).

W3C standardi järgi on XML-nimeruum nimede kollektsioon, mis on kasutusel XML-dokumendis elementide ja atribuutidena. Üks XML-dokument võib sisaldada elemente ja atribuute rohkem kui ühest nimeruumist.

Allolevas näites on kirjeldatud firma andmed nii, et firma nime ja registrinumbri kirjeldus on kirjeldatud nõnda, nagu riigis kokkulepitud. Kõigi ülejäänud elementide kirjeldused on väljamõeldud mingi firma enese poolt:

- iga element, millel pole eesliidet, kasutab nimeruumi
  `xmlns="http:/www.mingifirma.ee/firma"`

- iga element, millel on eesliide ee: kasutab nimeruumi
  `xmlns:ee="http://www.riik.ee/firma"`

```
<firma xmlns="http://www.mingifirma.ee/firma"
xmlns:ee="http://www.riik.ee/firma">
      <ee:nimi>AS Virumaa Mets</ee:nimi>
      <aadress>Metsa 7</aadress>
      <ee:regnumber>0011223344</ee:regnumber>
      <faks />
</firma>
```

XML'i valideerimine

Et XML andmeid vahetada või töödelda, on vähe kasu teadmisest, et XML on WellForm, kuna XML'i moodustamise reeglid on väga lõdvad ja täpselt samu andmeid on võimalik esitada mitmel erineval moel. Et fikseerida, mis kujul andmed peaksid XML failis olema, on kasutusel XML'i valideerimine. Neid reegleid on võimalik seada kas kasutades DTD (Document Type Definition) või XML skeeme (XML Schema). Kui XML fail vastab DTD's või XML skeemis kirjeldatud reeglitele, siis nimetatakse seda XML'i trimmis XML (valid XML)

DTD kirjeldus võib paikneda XML andmetega samas failis või võib olla täiesti eraldi seisev fail. Nende lausetega pannakse paika, millised elemendid on nõutavad ja milline on elementide järjestuse kord.

DTD suurimaks puuduseks on see, et ei ole võimalik määrata, mis tüüpi peavad olema elementides või atribuutides olevad andmed (ainukene andmetüüp on tekst). Seetõttu kasutatakse keerukamate süsteemide puhul XML skeeme.

XML skeemid

XML skeemid on reeglite kogum, kus on kirjas, mis võib ja mis ei või olla XML-andmefaili erinevates osades. Reeglitega on võimalik määrata, millised elemendid, millises järjekorras peavad olema ning mis tüüpi andmeid võivad sisaldada atribuudid ja elemendid. Andmetüüpidele on võimalik lisada piiranguid, nt vähim või suurim võimalik väärtus numbrite puhul, tekstilistel andmetel teksti pikkus. Lisaks sellele on võimalik määrata konkreetset väärtuste hulka.

Skeemides jagunevad elemendid vastavalt sisule. Lihtsaks elemendiks (simple type) nimetatakse elemente, millel puuduvad atribuudid ja alamelemendid.

Näiteks: `<nimi>AS Virumaa Mets</nimi>`

XML skeemis võiks seda kirjeldada järgmiselt:

```
<xs:element name="nimi" type="xs:string" minOccurs="1"
maxOccurs="unbounded"/>
```

Antud XML failis peab esinema element Nimi vähemalt 1 korra, kuid võib esineda ka rohkem. Nimi on tekstilist tüüpi (string).

Elemendid, millel on atribuudid ja/või alamelemendid, on keerulised elemendid (complex type).

Näiteks:

```
<firma>
      <nimi>AS Virumaa Mets</nimi>
      <aadress>Metsa 7</aadress>
      <regnumber>0011223344</regnumber>
      <faks />
</firma>
```

XML skeemis võiks seda kirjeldada järgmiselt:

```
<xs:element name="firma">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nimi" type="xs:string"/>
      <xs:element name="aadress" type="xs:string"/>
      <xs:element name="regnumber" type="xs:string"/>
      <xs:element name="faks" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Ülesandeid

- Koosta XML-fail autode loeteluga
- Koosta skeem ühe auto andmete hoidmise kirjelduseks
- Koosta skeem autode loetelu andmete kirjelduseks

XMLi kasutamine SqlServeris

Juba alates SQL Server 2000st on SQL Serveris olnud XMLi tugi. SQL Server 2005s on seda oluliselt täiendatud. Juurde on tulnud isegi xml andmetüüp. Kuigi SQL päringuid tehes ei ole XML kujul palju rakendust, on see funktsionaalsus äärmiselt vajalik rakendustes, kuna .NET raamistik saab XML andmetega väga hästi hakkama.

XMLi genereerimine relatsioonilistest andmetest

Kõige lihtsam on SQL Serverist XMLi küsida SELECT lause abil, lisades lõppu FOR XML märksõnad. Näiteks järgmise SQL lause abil saame kolme toote nimed XML kujul

```
SELECT TOP 3 nimi, hind
FROM toode
ORDER BY nimi
FOR XML AUTO
```

Tulemus on:

```
<toode nimi="kala" hind="55.0000"/>
<toode nimi="kapsas" hind="5.0000"/>
<toode nimi="kurk" hind="5.0000"/>
```

Nagu näeme, on saadud XML ilma juurelemendita. See tähendab, et otseselt XML faili sellist tulemust salvestada ei saa, erinevad programmid sh .NET võtavad XMLi vastu ka ilma juurelemendita. Juurelemendi võite saadud XMLile lisada, kas käsitsi läbi programmi või siis SQLis lisades FOR XML ... lause lõppu, ROOT('juurikanimi').

Lisaks andmetele ühest tabelist on võimalik sama meetodiga pärida ka seotud andmeid. Tulemusena genereeritakse hierarhiline XML:

```
SELECT tellimus.klient, toode.nimi AS toode, toode.hind
FROM tellimus
      INNER JOIN tellimustoode
ON tellimus.kood = tellimustoode.tellimus_kood
      INNER JOIN toode ON tellimustoode.toode_kood = toode.kood
FOR XML AUTO
```

Tulemus on:

```
<tellimus klient="madis">
  <toode toode="kala" hind="55.0000" />
```

```
  <toode toode="kurk" hind="5.0000" />
  <toode toode="piim" hind="7.0000" />
</tellimus>
<tellimus klient="jaan">
  <toode toode="kurk" hind="5.0000" />
  <toode toode="limonaad" hind="5.0000" />
</tellimus>
<tellimus klient="anni">
  <toode toode="kapsas" hind="5.0000" />
  <toode toode="õun" hind="10.0000" />
</tellimus>
```

Loomulikult on võimalik kasutada ka teistsuguseid XML struktuure. Näiteks kui asendame SELECT lause lõpus märksõna AUTO märksõnaga RAW, tekitatakse meile iga kirje kohta üks XML element nimega row, kus kõik väljad on atribuutidena:

```
SELECT tellimus.klient, toode.nimi AS toode, toode.hind
FROM tellimus
      INNER JOIN tellimustoode
ON tellimus.kood = tellimustoode.tellimus_kood
      INNER JOIN toode ON tellimustoode.toode_kood = toode.kood
FOR XML RAW
```

Tulemus on:

```
<row klient="madis" toode="kala" hind="55.0000" />
<row klient="madis" toode="kurk" hind="5.0000" />
<row klient="madis" toode="piim" hind="7.0000" />
<row klient="jaan" toode="kurk" hind="5.0000" />
<row klient="jaan" toode="limonaad" hind="5.0000" />
<row klient="anni" toode="kapsas" hind="5.0000" />
<row klient="anni" toode="õun" hind="10.0000" />
```

Kui need kaks meetodit ei sobi, siis on loomulikult võimalik ka ise struktuur ette anda. Selleks tuleb SELECT lause abil moodustada universaalne tabel, kus oleksid järgmised väljad:

- Tag – elemendi tase XML struktuuris
- Parent – peaelemendi tase XML struktuuris
- Väljad ja atribuudid, igaüks eraldi veerus. Väljade nimed tuleb kirjutada kujul Element!tase!atribuut!täpsustus

Üritame eelpool kasutatud tooted teisendada järgmisele XML kujule:

```
<Tellimus TellimusID="1">
  <Klient>madis</Klient>
  <TellitudToode ToodeID="1" Toode="kala" Hind="55.0000" />
  <TellitudToode ToodeID="2" Toode="kurk" Hind="5.0000" />
  <TellitudToode ToodeID="3" Toode="piim" Hind="7.0000" />
```

```
</Tellimus>
<Tellimus TellimusID="2">
  <Klient>jaan</Klient>
  <TellitudToode ToodeID="2" Toode="kurk" Hind="5.0000" />
  <TellitudToode ToodeID="4" Toode="limonaad" Hind="5.0000" />
</Tellimus>
<Tellimus TellimusID="3">
  <Klient>anni</Klient>
  <TellitudToode ToodeID="5" Toode="kapsas" Hind="5.0000" />
  <TellitudToode ToodeID="6" Toode="õun" Hind="10.0000" />
</Tellimus>
```

Selleks tuleb meil tekitada universaalne tabel, mis näeb välja järgmine:

Tag	Parent	Tellimus! 1! TellimusID	Tellimus! 1!Klient! element	Tellitud Toode!2! ToodeID	TellitudToode!2!Toode	Tellitud Toode!2! Hind
1	NULL	1	madis	NULL	NULL	NULL
2	1	1	NULL	1	kala	55,00
2	1	1	NULL	2	kurk	5,00
2	1	1	NULL	3	piim	7,00
1	NULL	2	Jaan	NULL	NULL	NULL
2	1	2	NULL	2	kurk	5,00
2	1	2	NULL	4	limonaad	5,00
1	NULL	3	Anni	NULL	NULL	NULL
2	1	3	NULL	5	kapsas	5,00
2	1	3	NULL	6	õun	10,00

Sellise tabeli tekitamiseks tuleb moodustada UNION päring, mis paneb taseme kaupa tabeli kokku. Esmalt loeme sisse tellimused ja seejärel tooted. Kuna XMLi hakatakse genereerima vastavalt ridade järjekorrale, siis peame ka selle eelnevalt fikseerima. Sorteerida tuleb esmalt TellimusID järgi, kuna me soovime, et üks tellimus oleks ühes elemendis ning seejärel Parent välja järgi, et esmalt oleks Tellimuse element moodustatud ning sinna järele tuleksid

TellitudTooted. Kui päring valmis, tuleb lisada lõppu FOR XML EXPLICIT ning ongi valmis:

```
SELECT 1 AS Tag, null AS Parent,
      kood AS [Tellimus!1!TellimusID],
      klient AS [Tellimus!1!Klient!element],
      null AS [TellitudToode!2!ToodeID],
      null AS [TellitudToode!2!Toode],
null AS [TellitudToode!2!Hind]
FROM tellimus
UNION ALL
SELECT 2 AS Tag, 1 AS Parent, tellimustoode.tellimus_kood, null,
      toode.kood, toode.nimi, toode.hind
FROM tellimustoode
      INNER JOIN toode ON tellimustoode.toode_kood = toode.kood
ORDER BY 3, 2
FOR XML EXPLICIT
```

Ülesandeid

- Koosta XML EXPLICIT abil ühe auto andmetele vastav dokument, kus mark on elemendina ning valmistamisaasta atribuudina märgitud

- Koosta eelnevat näidet arvestades XML EXPLICIT jaoks tabel ning sealtkaudu XML-dokument, kus autode andmed on maakondade kaupa loetelus. Iga maakonna juures on näha ka maakonnakeskus autoregistri asukohana.

XML andmetüübi kasutamine

XML dokumentide ja XML dokumendi osade hoidmiseks SQL Serveris saab kasutada xml andmetüüpi. Ühele väljale saab panna kuni 2GB XML andmeid. Vajadusel saab xml välja siduda ka schemaga, mille järgi väljale sisestatavat XMLi kontrollitakse.

Näiteks loome ühe lihtsa tabeli, mis koosneb võtmeväljast ning XML väljast:

```
CREATE TABLE xmlstuff(
      kood int IDENTITY(1,1) PRIMARY KEY,
      xmljutt xml NOT NULL,
)
```

Lisame sinna ka mõned read. Allolevas näites on XMLi lühendatud. Terve lisatav XML on ära toodud eelmise peatüki näidetes ning ka järgmisel lehel olevas tabelis..

```
insert xmlstuff (xmljutt) VALUES(N'<tellimus klient="madis"><toode to... ')
insert xmlstuff (xmljutt) VALUES(N'<row klient="madis" toode="kala" h... ')
insert xmlstuff (xmljutt) VALUES(N'<tellimus TellimusID="1"><Klient>m... ')
```

Tulemuseks on meil 3 reaga XML tabel:

kood	Xmljutt
1	<tellimus klient="madis"><toode toode="kala" hind="55.0000" /><toode toode="kurk" hind="5.0000" /><toode toode="piim" hind="7.0000" /></tellimus><tellimus klient="jaan"><toode toode="kurk" hind="5.0000" /><toode toode="limonaad" hind="5.0000" /></tellimus><tellimus klient="anni"><toode toode="kapsas" hind="5.0000" /><toode toode="õun" hind="10.0000" /></tellimus>
2	<row klient="madis" toode="kala" hind="55.0000" /><row klient="madis" toode="kurk" hind="5.0000" /><row klient="madis" toode="piim" hind="7.0000" /><row klient="jaan" toode="kurk" hind="5.0000" /><row klient="jaan" toode="limonaad" hind="5.0000" /><row klient="anni" toode="kapsas" hind="5.0000" /><row klient="anni" toode="õun" hind="10.0000" />
3	<tellimus tellimusid="1"><klient>madis</klient><tellitudtoode toodeid="1" toode="kala" hind="55.0000"/><tellitudtoode toodeid="2" toode="kurk" hind="5.0000"/><tellitudtoode toodeid="3" toode="piim" hind="7.0000"/></tellimus><tellimus tellimusid="2"><klient>jaan</klient><tellitudtoode toodeid="2" toode="kurk" hind="5.0000"/><tellitudtoode toodeid="4" toode="limonaad" hind="5.0000"/></tellimus><tellimus tellimusid="3"><klient>anni</klient><tellitudtoode toodeid="5" toode="kapsas" hind="5.0000"/><tellitudtoode toodeid="6" toode="õun" hind="10.0000"/></tellimus>

Kuigi meetodeid XMLi haldamiseks SQL Serveri enda vahenditega on mitmeid, vaatleme siinkohal vaid kaht:. xml.query ja xml.exist. Need on xml andmetüübi meetodid, mis võimaldavad XML väljalt XPath päringute abil otsida sobivaid väärtuseid ning kontrollida, kas sellised väärtused on olemas.

Näide 1: toome välja kogu XML välja sisu, kui selles XMLis on juurelemendiks tellimus:

```
select kood, xmljutt.query('/') AS xmljutt
from xmlstuff
where xmljutt.exist('/tellimus') = 1
```

Tulemus on järgmine:

kood	Xmljutt

1	<tellimus klient="madis"><toode toode="kala" hind="55.0000" /><toode toode="kurk" hind="5.0000" /><toode toode="piim" hind="7.0000" /></tellimus><tellimus klient="jaan"><toode toode="kurk" hind="5.0000" /><toode toode="limonaad" hind="5.0000" /></tellimus><tellimus klient="anni"><toode toode="kapsas" hind="5.0000" /><toode toode="õun" hind="10.0000" /></tellimus>
3	<tellimus tellimusid="1"><klient>madis</klient><tellitudtoode toodeid="1" toode="kala" hind="55.0000"/><tellitudtoode toodeid="2" toode="kurk" hind="5.0000"/><tellitudtoode toodeid="3" toode="piim" hind="7.0000"/></tellimus><tellimus tellimusid="2"><klient>jaan</klient><tellitudtoode toodeid="2" toode="kurk" hind="5.0000"/><tellitudtoode toodeid="4" toode="limonaad" hind="5.0000"/></tellimus><tellimus tellimusid="3"><klient>anni</klient><tellitudtoode toodeid="5" toode="kapsas" hind="5.0000"/><tellitudtoode toodeid="6" toode="õun" hind="10.0000"/></tellimus>

Näide 2: toome välja XMLis oleva elemendi Klient sisu:

```
select kood, xmljutt.query('//klient') AS xmljutt
from xmlstuff
where xmljutt.exist('//klient') = 1
```

Tulemus on järgmine:

kood	Xmljutt
3	<klient>madis</klient><klient>jaan</klient><klient>anni</klient>

Näide 3: toome välja XMLis sisalduvad tooted

```
select kood, xmljutt.query('//toode') AS xmljutt
from xmlstuff
where xmljutt.exist('//toode') = 1
```

kood	Xmljutt
1	<toode toode="kala" hind="55.0000" /><toode toode="kurk" hind="5.0000" /><toode toode="piim" hind="7.0000" /><toode toode="kurk" hind="5.0000" /><toode toode="limonaad" hind="5.0000" /><toode toode="kapsas" hind="5.0000" /><toode toode="õun" hind="10.0000" />

Näide 4: jätame toodetest nähtavale vaid need, mille hind on vähemalt 10:

```
select kood, xmljutt.query('//toode[@hind >= 10]') AS xmljutt
from xmlstuff
where xmljutt.exist('//toode') = 1
```

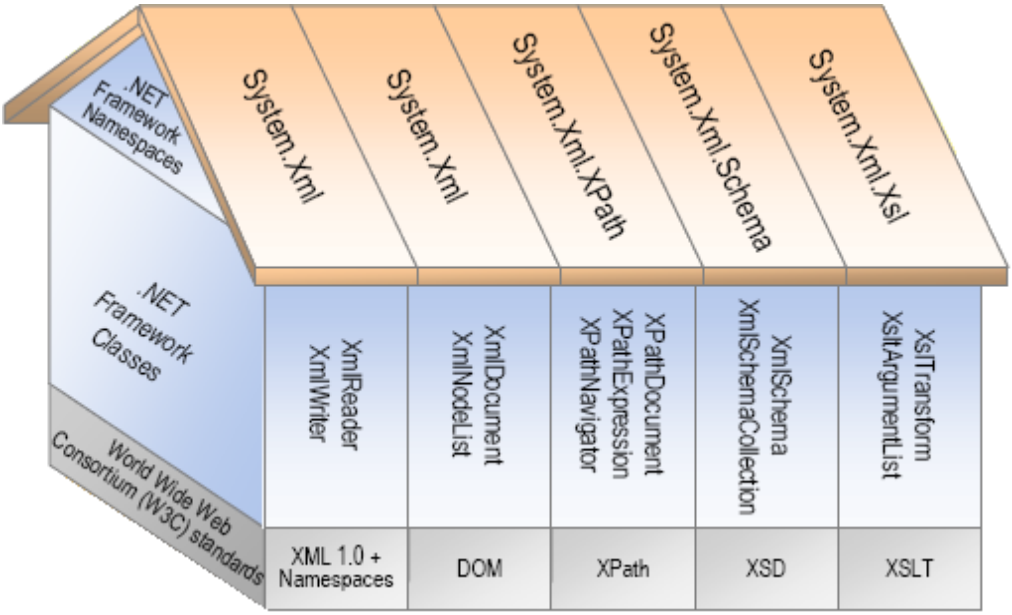
kood	Xmljutt
1	<toode toode="kala" hind="55.0000" /><toode toode="õun" hind="10.0000" />

Ülesandeid

- Loo tabel tulbaga autode andmete hoidmiseks XMLina
- Küsi sealt välja talletatud toodete margid
- Näita vaid nende autode andmeid, mis on valmistatud enne 1995ndat aastat.

XML andmete kasutamine .NET raamistikus

XML on väga hea vahend erinevatest andmeallikatest pärit andmete kokku koondamiseks ja haldamiseks. XMLi paremaks töötlemiseks on .NET raamistikus terve hulk erinevaid klasse, mis jagunevad mitmete nimeruumide vahel vastavalt W3 standarditele.



Olulisemad klassid, mida XMLiga töötades vaja läheb on:

Abstraktne klass	Kasutusala	Päritud klassid
XmlReader	XML voog (strem) XML andmete lugemiseks. Lugemise ajal on võimalik ka dokumentide kontrollimine e. valideerimine	XmlTextReader XmlNodeReader
XmlWriter	Tekitab XML voo kusagile edasi saatmiseks (teine rakendus, andmekandja)	XmlTextWriter
XmlNavigator	Kasutatakse dokumendis liikumiseks, kui kogu dokumendi mällu laadimine ei ole otstarbekas	XmlPathNavigator
XmlResolver	Väliste XML ressursside leidmine URI abil	XmlUrlResolver

XMLi parsimine

Parsimine tähendab andmete lugemist ning seejärel loetud andmetega mingite tegevuste sooritamist. Parsimine võimaldab XML failist leida just seda infot, mida teil kõige rohkem vaja läheb.

Püüame lugeda XML andmeid kasutades XmlReader klassi. Raamistiku poolt on tehtud kolm erinevat klassi XML andmete lugemiseks. Kui nende funktsionaalsus ei ole piisav, siis võite ise alati neid klasse juurde tekitada.

XMLi lugemiseks saab kasutada tavalisi System.IO klasse. Läbi IO klasside on võimalik XMLi lugeda nii failist kui ka voost. Sisuliselt ei ole vahet, kumba meetodit kasutada aga, et oleks lihtsam jälgida, siis võite ette kujutada nii, et Fail on andmekandjale salvestatud nimeline baidijada, voog on aga kusagilt mujalt (võrk, andmebaas, jne) tulev baidijada.

Proovime alustuseks lugeda sisse tavalise tekstifaili.

Selleks on meil esmalt vaja System.IO nimeruumi

```
using System.IO;
```

ning seejärel kirjutame protseduuri, mis avab tekstifaili ja trükib selle rea kaupa ekraanile.

```
string FailiNimi = @"c:\mingifail.txt";
if (File.Exists(FailiNimi)) { // kui fail on olemas
StreamReader FailiLugeja = File.OpenText(FailiNimi);
string rida = FailiLugeja.ReadLine();
while (rida != null) {
            Console.WriteLine(rida);
rida =FailiLugeja.ReadLine();
}
FailiLugeja.Close();
}
```

Kui soovite kogu faili mällu laadida ja seejärel temaga edasi toimetada, tuleks System.Text.StringBuilder klassi abil ehitada faili sisaldav string.

Analoogselt tekstifaili lugemisega käib ka XML faili lugemine. Lugeja on lihtsalt XmlTextReader tüüpi objekt. XmlTextReader oskab XMLi lugeda voost, stringist ja TextReader objektist.

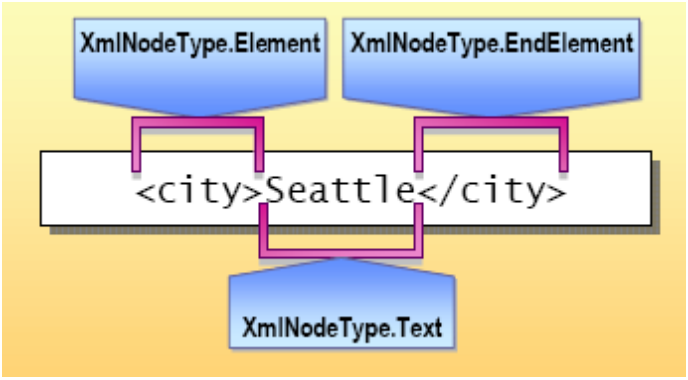
XmlTextReader objekti saame tekitada järgneva koodireaga:

```
XmlTextReader MinuXmlLugeja = new XmlTextReader(@"c:\mingifail.xml");
```

Kui lugeja on olemas, saame XMLi oksa (node) kaupa lugema hakata. Enamasti on seda kasulik teha mingi korduslausega:

```
while (MinuXmlLugeja.Read()) {
      // tee midagi
}
```

Kui on soov XML andmeid analüüsida või muul moel kasutada, on vaja teada, mis tüüpi oksal parajasti olete. Oksa tüübi selgitamiseks on vaja kontrollida NodeType omadust, selle omaduse väärtus võib olla üks järgnevatest:



Need kolm oksa tüüpi Element, TypeText, EndElement on kõige olulisemad, kuid lisaks neile on veel olemas:

Oksa tüüp	Selgitus
XmlNodeType.CDATA	Mitte parsetav tekst
XmlNodeType.Comment	XML kommentaar
XmlNodeType.ProcessingInstruction	Töö juhised erinevatele parseritele ja rakendustele
XmlNodeType.WhiteSpace	Tühjus elementide vahel
XmlNodeType.XmlDeclaration	XMLi kirjeldused

Kui element võib olla tühi e. algus ja lõpu tag on üks ja seesama, siis tuleks seda kontrollida IsEmptyElement omaduse kaudu.

Järgnevas näites genereerime XmlReader'it kasutades väljundisse uue XMLi.

```
XmlTextReader MinuXmlLugeja = new XmlTextReader("mingifail.xml");
```

```
while (MinuXmlLugeja.Read()) {
  switch (MinuXmlLugeja.NodeType) {
    case XmlNodeType.Comment:
      Console.WriteLine("<!--" + MinuXmlLugeja.Value + "-->");
      break;
    case XmlNodeType.Element:
      if (MinuXmlLugeja. IsEmptyElement) {
        Console.WriteLine("<" + MinuXmlLugeja.Name + " />");
      }else {
        Console.WriteLine("<" + MinuXmlLugeja.Name + ">");
      }
      break;
     case XmlNodeType.EndElement:
       Console.WriteLine("</" + MinuXmlLugeja.Name + ">");
       break;
     case XmlNodeType.Text:
       Console.WriteLine(MinuXmlLugeja.Value);
       break;
     default:
       // siin võiks midagi teha ülejäänud oksadega
       Console.WriteLine(" --- Tundmatu oks --- ");
       break;
     }
}
```

Lisaks tekstilisele sisule võivad elemendid omada ka atribuute. Kas elemendil on atribuut, saame teada läbi HasAttribute omaduse. Atribuutide arvu saame teada AttributesCount omadusest. Edasi on võimalik küsida atribuute, kas nime või indeksi järgi kasutades GetAttribute meetodit

```
MinuXmlLugeja.GetAttribute(0); // esimese atribuudi väärtus
MinuXmlLugeja GetAttribute(”ID”); // atribuudi ID väärtus
```

või palume lugejal liikuda järjest järgmisele atribuudile.

```
if (MinuXmlLugeja.HasAttributes){
    for (int i = 0; i < MinuXmlLugeja.AttributeCount; i++){
        MinuXmlLugeja.MoveToAttribute(i);
        Console.Write(" {0}='{1}'", MinuXmlLugeja.Name,
                      MinuXmlLugeja.Value);
    }
    MinuXmlLugeja.MoveToElement();
}
```

Nagu iga teisegi sisend/väljund protseduuri, nii ka XML parsimise juures võib tekkida vigu. Kui XmlReader avastab vea, annab ta sellest teada läbi XmlExeption’i. Seega peaks kogu XMLi lugemine olema try ... catch struktuuri sees:

```
XmlTextReader MinuXmlLugeja = new XmlTextReader("mingifail.xml");
try {
  while(MinuXmlLugeja.Read()) {
  // tee midagi
  }
```

```
}
catch(XmlException e) {
  Console.WriteLine(e.Message);
  Console.WriteLine("Pronleem XML failis – rida {0}, veerg {1}",
      MinuXmlLugeja.LineNumber, MinuXmlLugeja.LinePosition);
}
```

Ülesandeid

- Koosta autode andmete fail XMLina
- Tee kindlaks, mitme auto andmed on faili kirjutatud
- Trüki välja leitud automarkide nimed.

XMLi valideerimine

XML failil on kaks staatust, mis näitavad tema kvaliteeti:

- Well-Formed – korrektselt vormistatud, st XML vastab W3 poolt seatud XMLi reeglitele
- Valid –Well-Formed ning lisaks vastab see XML teie poolt seatud reeglitele

Kasutades XmlTextReader klassi, saame vea siis, kui XML ei ole Well-Formed. Täpsemaid kontrolle (millised elemendid on olemas, kas nad on õiges järjestuses jne) aga ei rakendata.

XMLi täpsemaks kontrollimiseks on kaks moodust: DTD dokumendid ja XML Schema. Siinkohal ei hakka vaatama, kuidas neid dokumente moodustada, vaid vaatame, mis saab siis, kui teil on selline dokument olemas ja tahate teada, kas XML sisend vastab sellele.

Järgnevalt vaatleme kahte erinevat lähenemist XML valideerimisele. .NET raamistiku 1.x versioonis oli XMLi valideerimiseks XmlValidatingReader klass, raamistiku 2.0 versioonis seda enam ei ole ning selle asemel saab kasutada XmlReader klassi. Suurim erinevus nende kahe vahel seisneb selles, et kui ValidatingReader avastas vea, sai selle kinni püüda try ... catch konstruktsiooni abil, nüüd tuleb selleks aga kasutada sündmusi.

XmlReader on huvitav klass ka selle poolest, et lugemise määranguid ei edastata mitte omaduste kaudu, vaid spetsiaalse omaduste objektina. Seega, kui soovime XML failist lugeda ning samal ajal kontrollida, et loetav XML oleks korrektne, peame tegema lugemismäärangute objekti ning edastama selle lugeja objektile.

Valideerimiseks kasutame schemat, mis on määratud XML failis.

```
<juurikas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation ="naidis.xsd">
```

Esmalt vaatame, kuidas valideerida parsimise ajal:

```
XmlReaderSettings maarangud = new XmlReaderSettings();
// kontrollimiseks kasutame schemat
maarangud.ValidationType = ValidationType.Schema;
// schemat asukoht on näidatud XML failis
maarangud.ValidationFlags = XmlSchemaValidationFlags.ProcessSchemaLocation;
// probleeme lahendame meetodiga xvr_ValidationEventHandler
maarangud.ValidationEventHandler +=
                    new ValidationEventHandler(xvr_ValidationEventHandler);
// Teeme objekti XMLi lugemiseks
XmlReader lugeja = XmlReader.Create(@"C:\Erki\mingifail2.xml", maarangud);
// Hakkame parsema
while (lugeja.Read());
reader.Close();
```

Probleeme haldav meetod on üsna lihtsakoeline:

```
private void xvr_ValidationEventHandler(object sender,
                                                    ValidationEventArgs e) {
    Console.WriteLine("Viga! {0}", e.Message);
    Console.WriteLine("XML exception: Ln {0} Col {1}",
                          e.Exception.LineNumber,e.Exception.LinePosition);
}
```

Teine meetod on mitte XMLi jupi kaupa parsida, vaid lugeda kogu XML mällu. Kasulik väikeste XML failide juures. Et näide oleks põnevam, loeme XMLi kasutades andmevoogu, mille saame tavalisest faili lugemisest.

```
FileStream fs = File.Open(@"C:\Erki\mingifail2.xml", FileMode.Open);
XmlDocument xdoc = new XmlDocument();
XmlReaderSettings maarangud = new XmlReaderSettings();
maarangud.ValidationType = ValidationType.Schema;
maarangud.ValidationFlags = XmlSchemaValidationFlags.ProcessSchemaLocation;
maarangud.ValidationEventHandler +=
                    new ValidationEventHandler(xvr_ValidationEventHandler);
XmlReader lugeja = XmlReader.Create(fs, maarangud);
xdoc.Load(lugeja); // edaspidi on kogu XML kasutatav läbi xdoc'i
fs.Close();
```

Kui XML failis pole schemat määratud või lihtsalt soovite kontrollida mõne teise schema järgi, siis saate sobiva schema määrata programselt. Kontrolli võib teostada peale XML faili mällu laadimist kasutades Validate meetodit:

```
XmlDocument xdoc = new XmlDocument();
xdoc.Load("mingifail2.xml");
xdoc.Schemas.Add(null, "naidis.xsd");
ValidationEventHandler veh =
                    new ValidationEventHandler(xvr_ValidationEventHandler);
xdoc.Validate(veh);
```

Või XML faili laadimise ajal, määrates schema ära määrangutes:

```
FileStream fs = File.Open("mingifail2.xml", FileMode.Open);
XmlDocument xdoc = new XmlDocument();
XmlReaderSettings settings = new XmlReaderSettings();
settings.Schemas.Add(null, "naidis.xsd");
settings.ValidationType = ValidationType.Schema;
settings.ValidationEventHandler +=
                    new ValidationEventHandler(xvr_ValidationEventHandler);
XmlReader reader = XmlReader.Create(fs, settings);
xdoc.Load(reader);
fs.Close();
```

Lisaks XmlDocument objektile on võimalik XMLi laadida ka DataSet'i sisse. Kuna DataSet hoiab kõiki andmeid XML kujul, siis on XML andmete DataSeti laadimine tehtud äärmiselt lihtsaks. Järgnevalt loeme DataSeti XML faili, koos seal näidatud Schemaga:

```
myDS = new DataSet();
myDS.ReadXml("C:\Erki\mingifail2.xml", XmlReadMode.ReadSchema);
```

Kui XML failis ei ole Schemat näidatud, on võimalus see genereerida vastavalt laetava XML faili struktuurile:

```
myDS = new DataSet();
myDS.ReadXml("C:\Erki\mingifail.xml", XmlReadMode.InferSchema);
```

Loomulikult on võimalik ka ette määrata, kust tuleb schema ja kust tulevad andmed:

```
myDS = new DataSet();
myDS.ReadXmlSchema(@"C:\Erki\naidis.xsd");
myDS.ReadXml(@"C:\Erki\naidis.xml", XmlReadMode.IgnoreSchema);
```

Ülesandeid

- Koosta skeem ühe auto andmete tarvis
- Koosta skeem autode loetelu andmete tarvis
- Kontrolli, kas olemasolev autoandmete dokument vastab loodud skeemile

XMLi salvestamine

Kõige lihtsam on XMLi salvestada, kui olete loonud XmlDocument objekti. Sellisel juhul tuleb välja kutsuda Save meetod ja ongi salvestatud.

```
xdoc.Save(@"C:\Erki\mingifail3.xml");
```

Loomulikult on võimalik genereerida XMLi ka siis, kui XmlDocument objekti ei ole. Sellistel puhkudel saab kasutada XmlTextWriter objekti.

```
XmlTextWriter XmlKirjutaja = new XmlTextWriter(
                                    @"mingifail4.xml", Encoding.UTF8);
XmlKirjutaja.Formatting = Formatting.Indented;
```

Kui kirjutaja loodud, saate hakata moodustama XML faili:

```
XmlKirjutaja.WriteStartDocument();
XmlKirjutaja.WriteStartElement("juurikas");
XmlKirjutaja.WriteStartElement("tellimus");
XmlKirjutaja.WriteAttributeString("tellimusid", "1");
XmlKirjutaja.WriteElementString("klient", "madis");
XmlKirjutaja.WriteEndElement();
XmlKirjutaja.WriteEndElement();
XmlKirjutaja.WriteEndDocument();
XmlKirjutaja.Close();
```

Tulemuseks on Well-Formed XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<juurikas>
      <tellimus tellimusid=”1” >
            <klient>madis</klient>
      </tellimus>
</juurikas>
```

XMLi on lisaks eelnevatele meetoditele võimalik salvestada ka otse DataSetist. Selleks on DataSet’il kaks väga kasulikku meetodit:

```
myDS.WriteXml("C:\Erki\uus.xml", XmlWriteMode.IgnoreSchema);
myDS.WriteXmlSchema("C:\Erki\uus.xsd");
```

Ülesandeid

- Küsi kasutajalt auto andmed ning väljasta need XML-faili. Aasta salvestatakse atribuudina, mark elemendina.

- Loe andmebaasist autode andmed DataSeti. Salvesta andmed XML-faili. Salvesta skeem eraldi xsd-faili

LINQ - .NET Language-Integrated Query

LINQ lisab programmeerimiskeelde (nagu C#, VB.NET jne) päringute kirjutamise käsustiku, mille abil on võimalik andmeid valida, filtreerida ning teha kokkuvõtteid.

Kõik LINQ vahendid paiknevad System.Linq nimeruumis.

All olev näide kasutab LINQ päringut selleks, et töödelda massiivis olevaid andmeid.

```
using System;
using System.Linq;
using System.Collections.Generic;
class app {
  static void Main() {
    string[] names = { "Burke", "Connor", "Frank",
                       "Everett", "Albert", "George",
                       "Harris", "David" };
    IEnumerable<string> query = from s in names
                                where s.Length == 5
                                orderby s
                                select s.ToUpper();
    foreach (string item in query)
      Console.WriteLine(item);
  }
}
```

Tulemuseks on:

```
BURKE
DAVID
FRANK
```

LINQ mõistmiseks tuleks esmalt vaadelda programmi esimest lauset:

```
IEnumerable<string> query = from s in names
                            where s.Length == 5
                            orderby s
                            select s.ToUpper();
```

Kohalik muutuja Query väärtustatakse päringu avalisega. Päringuavaldis võtab andmeid ühest või mitmest andmeallikast ning võimaldab sooritada mitmeid operatsioone. Antud päring kasutab kolme standardset operaatorit: Where, OrderBy ja Select.

Seega võiks seda päringut lugeda järgmiselt: loome loendatava teksti sisaldava massiivi Query, valides andmed tekstimassiivist names kus teksti pikkuseks on 5 märki ning

sorteerime tulemuse tähestikuliselt ning tulemusena näitame sobivaid tekstiväärtusi suurte tähtedega.

LINQ päringuid saab teha kõigi loendatavate kollektsioonide ja massiivide peal.

Kõik LINQ päringud koosnevad kolmest tegevusest:

- Andmeallika tekitamine
- Päringu loomine
- Päringu käivitamine

```
using System;
using System.Linq;
using System.Collections.Generic;
class app {
  static void Main() {
        // Linq päringu kolm osa:
        //  1. Andmeallika loomine.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
        // 2. Päringu loomine.
        // numQuery tekitatakse IEnumerable<int> tüüpi
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;
        // 3. Päringu käivitamine.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
  }
}
```

ASP.NET

ASP.NET on .NET raamistiku moodul, mis võimaldab sul luua veebirakendusi kasutades sealjuures minimaalselt koodi.

ASP.NET ei ole mitte ASP (Active Server Pages) uus versioon, vaid täiesti uus lähenemine veebi rakenduste loomisele. Erinevalt ASPist ja ka PHPst, mis on peamiselt skriptimise keeled, on ASP.NET lehtede taga olev kood täielikult objektorienteeritud. Seega tuleks ASP.NETi võrrelda mitte PHP vaid JAVA rakendustega.

Koodi ASP.NET lehtede tarbeks võib kirjutada ükskõik millises .NET keeles. Lisaks veebivormidele on võimalik oma rakendust veebis serveerida ka läbi veebiteenuste.

Ka ASP.NETist on olemas mitmeid versioone. Kui võrrelda ASP.NET versioone 1.1, 2.0 ja 3.5 siis võib öelda, et palju on jäänud samaks, kuid üht teist on ka ümber tehtud ja lisatud. Hea uudis on see, et kõik vanad konstruktsioonid töötavad, kuid juurde on tulnud mitmed uued meetodid. Tänu provaideri(teenusepakkuja) põhisele lähenemisele on rakenduse loomine alates 2.0 versioonist muutunud märksa abstraktsemaks ja lihtsamaks. Palju koodi on viidud lehekülje tagustelt koodilehtedelt teenusepakkujatesse. ASP.NET versioonis 3.5 on juurde tulnud toetus AJAXile (Asyncronious Java and Xml), mõned uued serveri kontrollid ning LINQ kasutamise võimalus. AJAX võimaldab andmeid lehel reaalajas muuta ilma, et peaks kogu lehte selle tarbeks uuesti laadima.

ASP.NET lehed koosnevad tekstifailidest, mida saab serveerida läbi IIS (Internet Information Service) virtuaalkaustade. Lehtede loomiseks sobivad kõik tekstiredaktorid. Abivahendeid usaldavale inimesele on kõige parem kasutada kirjutamiseks sellised abivahendeid, mis lehtede loomisel kiirendavad koodi kirjutamist lõpetades alustatud sõnu, kontrollivad jooksvalt süntaksit ning aitavad HTMLi loomisel. Üheks selliseks abivahendiks on Visual Studio. Ehkki palja "rumala" tekstiredaktoriga kirjutamisel on eeliseks lihtsus, siis nt Visual Studio Web Developer Expressiga on lootus ka algajal nõnda läbi saada, et ta paljudesse menüüdesse ära ei upu.

Visual Studio paigaldamine

Visual Studio on arendusvahend, mida kasutavad väga paljud programmeerijad ning programmeerimisfirmad sh ka Microsoft. Visual Studio´st on olemas mitmeid versioone, neist enamus on mõeldud professionaalsetele programmeerijatele ning programmeerijate meeskondadele. Lisaks kommertsversioonidele on olemas ka tasuta versioonid (Express versioonid), mis on mõeldud lihtsalt programmeerimishuvilistele ning õpilastele.

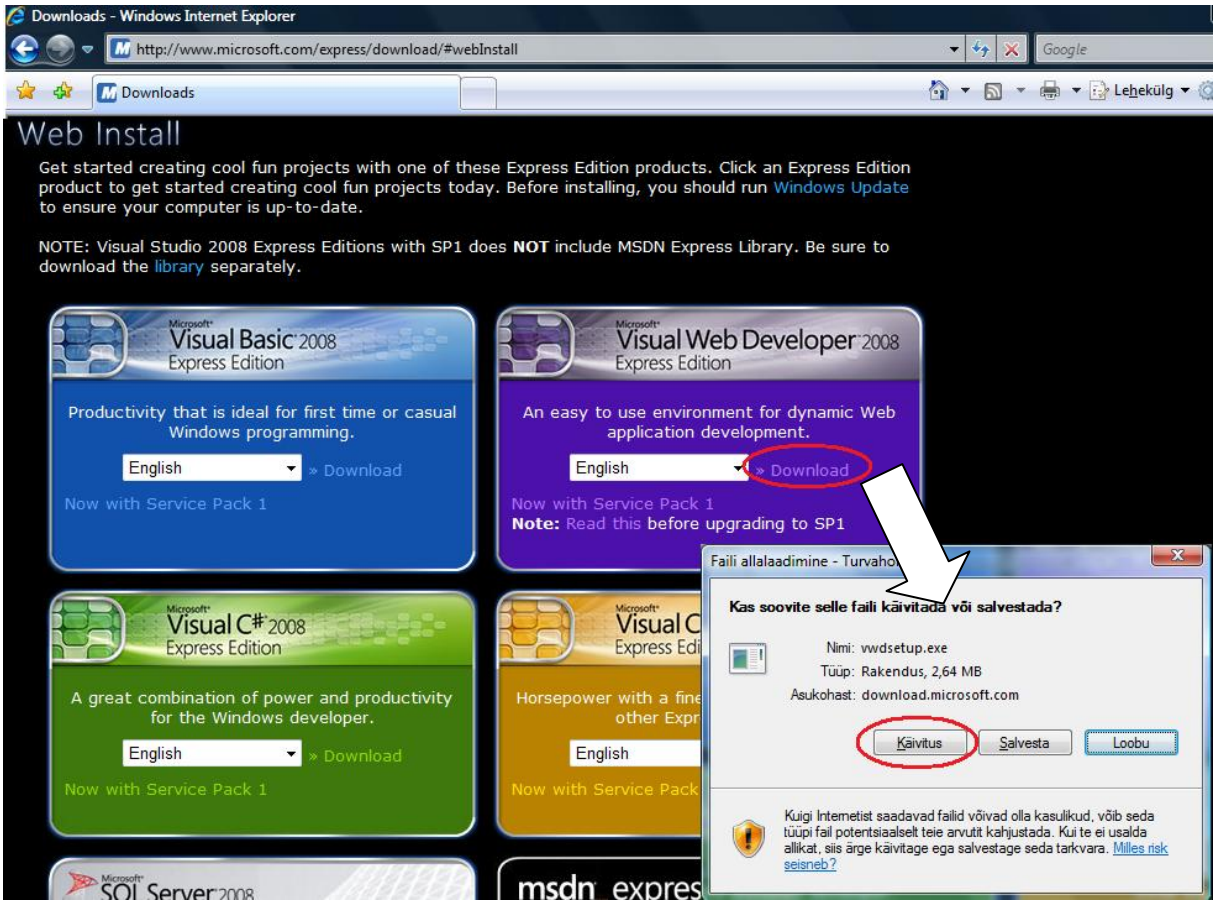
Visual Studio Express versiooni saamiseks tuleb minna aadressile www.microsoft.com/express/ ning tõmmata endale sealt sobiv versioon. Veebilehestike koostamiseks nagu nimigi ütleb, sobib enesele laadida Visual Web Developer Express.

Lisaks Visual Studio´le oleks kasulik paigaldada ka andmebaasimootor SQL Server Express Edition koos SQLi haldus- ja arendusvahendiga SQL Server Management Studio Express.

WDE paigaldamiseks peaks arvutil olema vähemalt 600 MHz protsessor (soovitavalt üle 1 GHz), vähemalt 192 MB mälu (soovitav 256 MB või koos andmebaasiga 512 MB) ning sõltuvalt paigaldatavatest komponentidest 500 MB kuni 1,8 GB vaba kettaruumi. Enamuse sellest pea 2GB suurusest mahust võtab spikker (Help). Kui spikrit ei paigalda on vajalik kettaruum üle GB väiksem! Spikker on vajalik siis kui Teil ei ole pidevat Interneti ühendust või töötate kohtades, kus alati ei ole võimalik Internetti kasutada. Kui on olemas püsiv Internetiühendus võite kasutada veebis olevat spikrit, mis asub aadressil http://msdn.microsoft.com. Online spikker on integreeritud ka WDE sisse, mis võimaldab sealt infot otsida ka WDE abil!

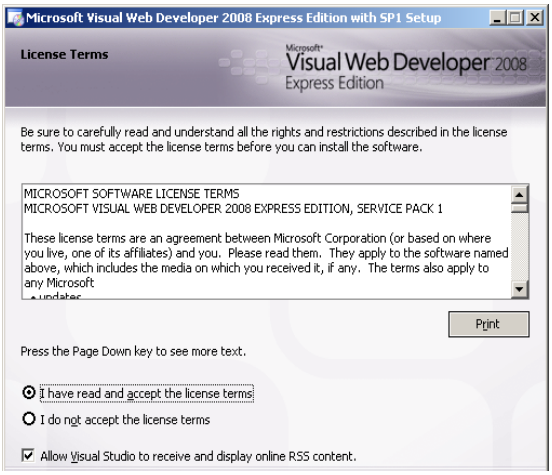
Installeerimiseks peavad teil olema arvutis administraatori (süsteemiülema) õigused!

Visual Web Developer 2008 Express Edition´i paigaldamiseks otsige see Microsofti veebist ülesse ning käivitage allalaadimine. Esmalt laetakse alla programm, mis tõmbab alla päris installika e. seda esimest allalaetavat 2,6MB rakendus ei ole mõtet salvestada – selle võib käivitada veebist.



Järgnevalt avaneb teil võimalus jagada oma installeerimisekogemust Microsoftiga ☺ Sisuliselt tähendab see seda, et peale installeerimise lõppu saadetakse Microsofti aruanne infoga, mida te valisite ja kuidas paigaldamine õnnestus.

Kui vajutate next avaneb litsentsileping (mis tuleks kindlasti läbi lugeda), millega tuleb jätkamiseks nõustuda. Lisaks sellele on teil võimalik tellida WebDeveloperi avalehele uudised temaatilise RSS kanali kaudu.

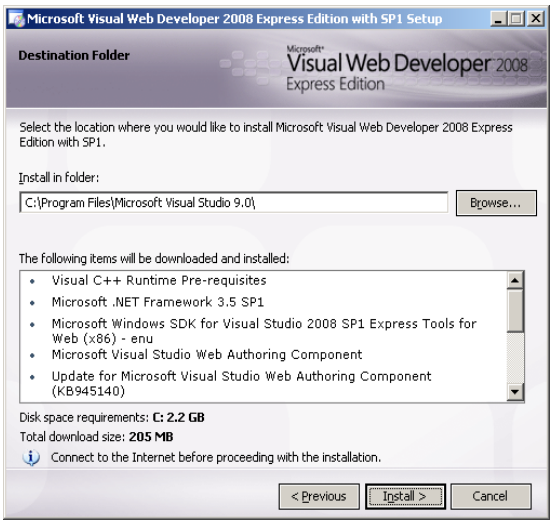


Installi järgmisest sammust saate valida, milliseid lisakomponente soovite koos WebDeveloperiga paigaldada. Nendeks komponentideks on:

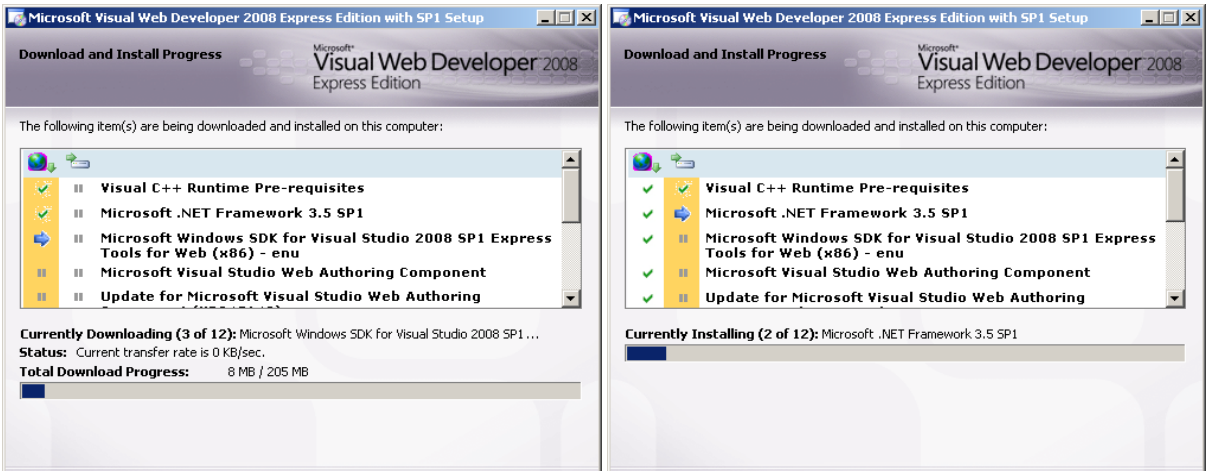
- Silverlight runtime, mis võimaldab vaadata animeeritud veebilehti.
- SQL Server 2008 Express – lihtne andmebaasiserver andmebaaside haldamiseks ja kasutamiseks

Kui kõik vajalikud komponendid valitud saate näidata millisesse kausta WebDeveloper paigutada ning näete kokkuvõtet kõigist alla laetavatest ja paigaldatavatest komponentidest.

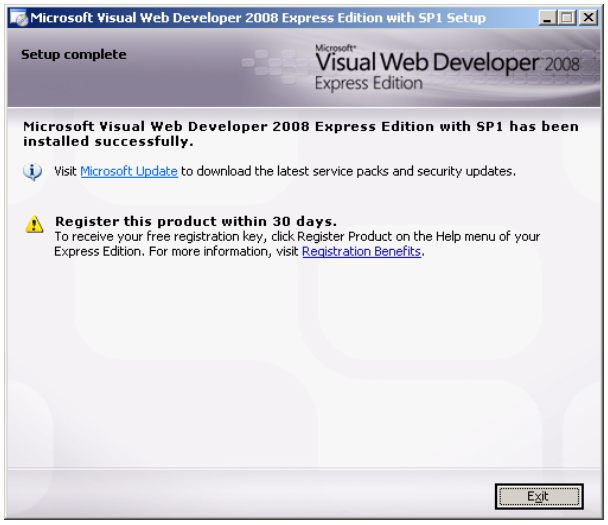
Komponentide hulk ja maht sõltu palju sellest, milliseid programme ja lisakomponente on sellesse arvutisse varasemalt installeeritud.



Järgnevalt saate rahulikult tegelda muude asjadega kuni kõik vajalik kohale tõmmatakse ning paigaldatakse ☺



Üldiselt on paigaldusprotsess täisautomaatne, kuid paigaldatavate komponentide hulgas võib olla ka selliseid komponente, mis vajavad peale paigaldamist arvuti taaskäivitamiseks.

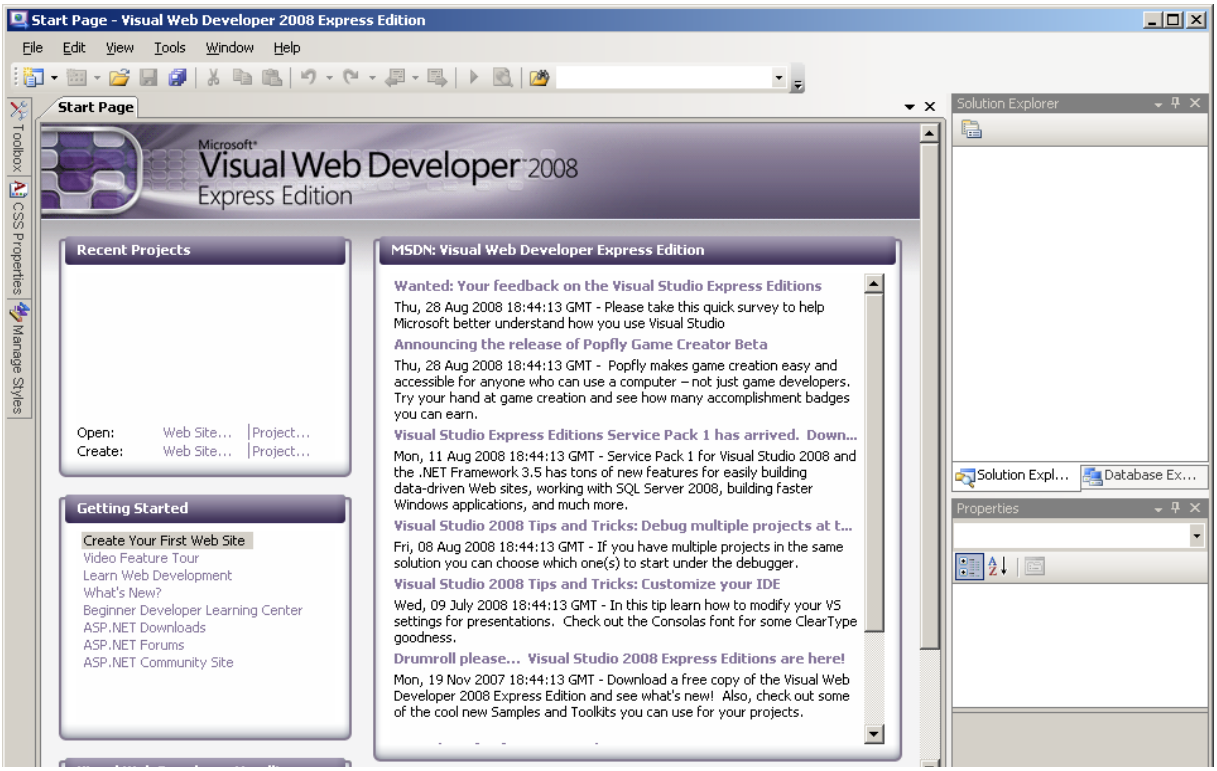


Ja ongi Visual Web Developer 2008 Express edition paigaldatud.

Peale paigaldamist oleks kasulik toode registreerida. Sel juhul saate seda programmi kasutada rohkem kui 30 päeva!

Lisaks sellele tasuks minna Microsoft Update veebisaidile http://www.update.microsoft.com ning paigaldada ka kõik kriitilised uuendused ja võib-olla ka mõned mittekriitilised täiendused!

Esimene Visual Studio käivitamine võtab pisut aega, kuid lõpuks ta siiski käivitub ning saate alustada ilusate veebisaitide loomisega. Kui te paigaldamise käigus tellisite ka RSS uudised siis need on nähtaval Visual Studio avalehel, koos viidetega juhenditele ning asjakohastele uudistegruppidele (newsgroups), foorumitele ja mailililistidele (mailing lists).


Start Page - Visual Web Developer 2008 Express Edition
File Edit View Tools Window Help
Start Page
Toolbox
CSS Properties
Manage Styles
Microsoft
Visual Web Developer 2008
Express Edition
Recent Projects
Open: Web Site... Project...
Create: Web Site... Project...
Getting Started
Create Your First Web Site
Video Feature Tour
Learn Web Development
What's New?
Beginner Developer Learning Center
ASP.NET Downloads
ASP.NET Forums
ASP.NET Community Site
MSDN: Visual Web Developer Express Edition
Wanted: Your feedback on the Visual Studio Express Editions
Thu, 28 Aug 2008 18:44:13 GMT - Please take this quick survey to help Microsoft better understand how you use Visual Studio
Announcing the release of Popfly Game Creator Beta
Thu, 28 Aug 2008 18:44:13 GMT - Popfly makes game creation easy and accessible for anyone who can use a computer – not just game developers. Try your hand at game creation and see how many accomplishment badges you can earn.
Visual Studio Express Editions Service Pack 1 has arrived. Down...
Mon, 11 Aug 2008 18:44:13 GMT - Service Pack 1 for Visual Studio 2008 and the .NET Framework 3.5 has tons of new features for easily building data-driven Web sites, working with SQL Server 2008, building faster Windows applications, and much more.
Visual Studio 2008 Tips and Tricks: Debug multiple projects at t...
Fri, 08 Aug 2008 18:44:13 GMT - If you have multiple projects in the same solution you can choose which one(s) to start under the debugger.
Visual Studio 2008 Tips and Tricks: Customize your IDE
Wed, 09 July 2008 18:44:13 GMT - In this tip learn how to modify your VS settings for presentations. Check out the Consolas font for some ClearType goodness.
Drumroll please... Visual Studio 2008 Express Editions are here!
Mon, 19 Nov 2007 18:44:13 GMT - Download a free copy of the Visual Web Developer 2008 Express Edition and see what's new! Also, check out some of the cool new Samples and Toolkits you can use for your projects.
Solution Explorer
Solution Expl...
Database Ex...
Properties

Põhivõimalused

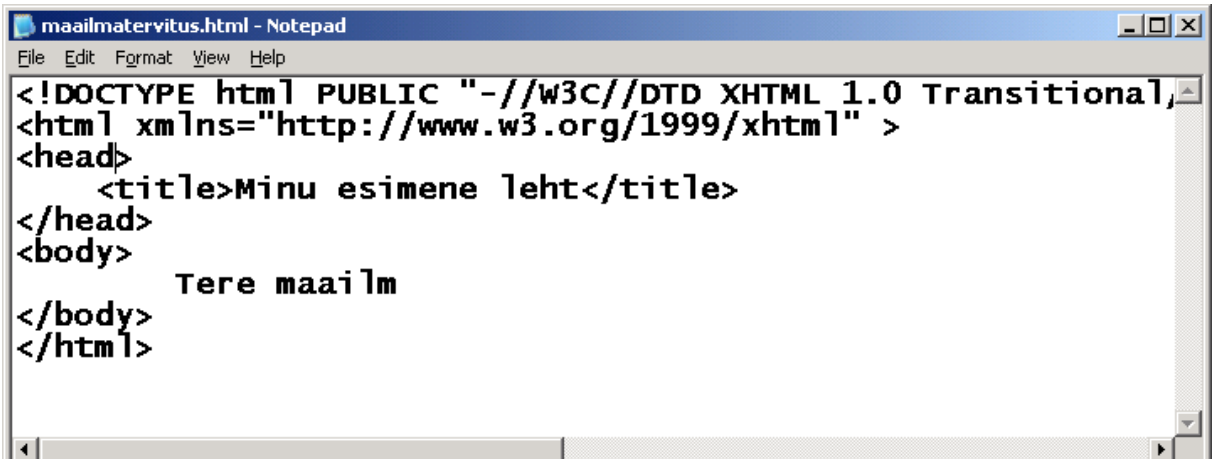
Lihtsa veebilehestiku loomine HTML keele abil

Esimene veebileht

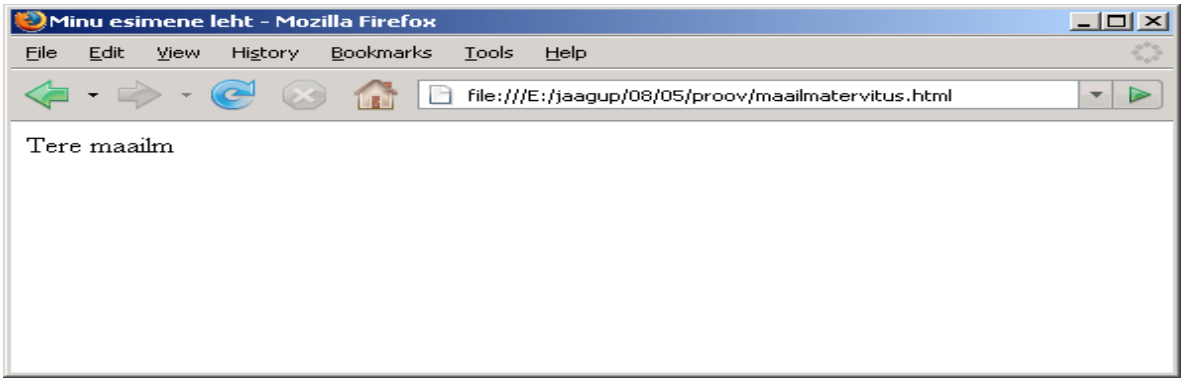
Kõige lihtsam veebirakendus koosneb ühest lehest. Veebilehtede levinumaks keeleks on HTML, mida veebilehitsejad mõistavad lugeda ning loetud teksti põhjal kasutajale lehe ette kuvada. Mitmete veebitehnoloogiate (ASP.NET, Java servlet, PHP, Python ...) tulemusena lehitsejasse saadetav tekst on ikkagi „puhas“ HTML, nii et osava peitmise korral ei pruugi veebisaidi vaatajal kuidagi võimalik olla kindlaks teha, millise tehnoloogia abil vastav lehestik on kokku pandud. Ning nagu varemalt kombeks ning praegugi lihtsamate, pidevat muutmist mitte vajavate lehtede puhul kasutatakse, võibki veebileht olla üks harilik HTMLi reeglitele vastav tekstifail, mida veebilehitsejas näidatakse. Mitmesugused tehnoloogiad on lihtsalt leidnud võimalusi, kuidas võimalikult mugavalt lehtedel olevad andmeid määrata vastavalt kasutaja soovidele. Lihtsaim HTMLi reeglitele vastav veebileht näeb aga välja järgmine:

```
<!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>
    <title>Minu esimene leht</title>
</head>
<body>
        Tere maailm
</body>
</html>
```

Selle võib salvestada omale sobiva tekstiredaktoriga (nt. Notepad või ka vastinstalleeritud Visual Web Developer). Panna failile laiendiks .html (siis teab veebilehitseja, et vastavat teksti tuleb kujundada HTML-i reeglitele vastavalt), jätta meelde kuhu fail salvestati.



Edasi juba lehitsejas avada nagu tavalist kohaliku masina faili (failimenüüst ava). Tulemuseks leht, kus ülal pealkirjariba peal on lehe pealkiri ning allpool sisu osas lehe sisu.



Veidi lähemad seletused, et mida miski HTMLi lehe osa tähendada võiks.

Avakäsklus teatab HTMLi versiooni. Nii nagu nt. Wordi dokumentidel on versioonid 2.0, 6.0, 97, 2000, 2003, 2007 jne, nõnda ka HTML on oma arengu käigus muutunud. Kõiki eri versioone ja „murrakuid“ lugedes saaks neid kokku õite mitukümmend. Siit võib välja lugeda, et versiooniks on XHTML 1.0 Transitional – ehk siis 1999ndast ligi kümnendi püsinud kirjapanekuvorming.

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

Edasi hakkavad HTMLi käsklused. Märkide < ja > vahel olevate sõnadega antakse teada, mis nüüd tulemas on. Esimese elemendi nimeks ongi html, ehk siis lehitsejale teadmiseks, et tulemas htmli dokument. Juuresolev atribuut xmlns (xml namespace) näitab, millise nimeruumiga elemendid seotud on. XHTML 1.0 Transitionali puhul siis järgnev nimi: http://www.w3.org/1999/xhtml. Näeb välja väga hüperlingi moodi. Aga selline kuju võeti nimeruumide nimetuste puhul ette vaid selleks, et kogemata ei satuks mitmel eri firmal ette samanimelist nimeruumi. Kas vastavale lehele ka selgitav tekst andmete kohta pannakse – see on juba vajaduse ja viisakuse küsimus.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

Veebilehe kood koosneb kahest suhteliselt iseseisvast osast. Ühe nimeks head (päis) ning teiseks body (sisu). Esimesse neist pannakse pealkiri (title) ning soovi korral lisaks igasugu muud andmed lehe kohta, mida otse näidata ei soovita. Näiteks lehe autor, märksõnad, kooditabel jm.

```
<head>
    <title>Minu esimene leht</title>
</head>
```

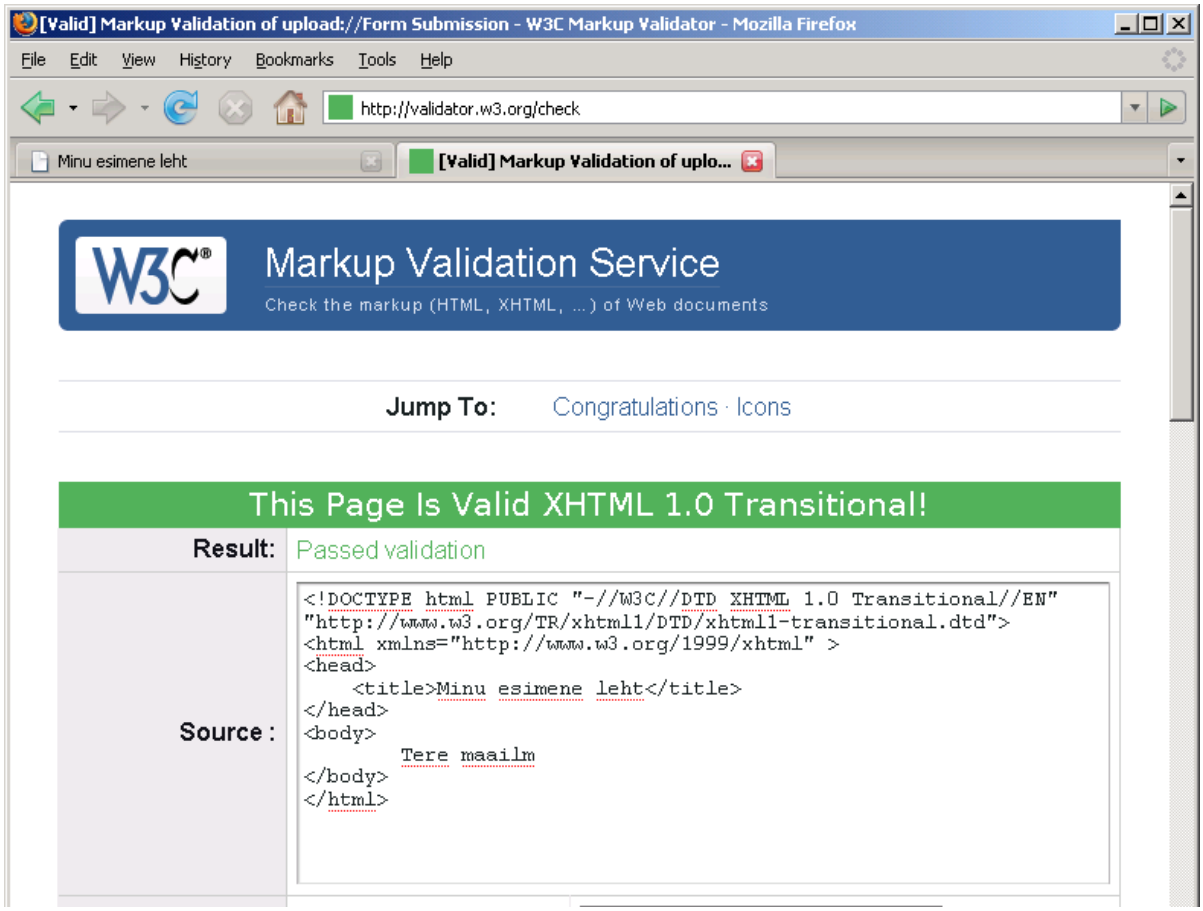
Sisu osa lehel on kõigile nähtav. Lihtsamal juhul ongi siin paljas tervitav tekst, aga eks hiljem saab kujundust keerukamaks hakata muutma.

```
<body>
        Tere maailm
</body>
```

Nagu näha, siis kõik alustavad elemendid peavad kusagil ka lõppema. Lehe sisuosa tähistav element body lõpeb lõpetava tähisega </body> ning dokumendi enese </html>.

```
</html>
```

Et suurem lehekülg võib ülesehituselt päris keerukaks minna, siis on vaja abivahendeid kontrollimaks, kas kõik ikka reeglitele vastab. Enamik lehitsejaid suudab küll mõningad (trüki)vead ka ise ära aimata ning siiski lehe viisakalt välja näidata. Kuid sealjuures võib hakata tekkima probleeme. Näiteks, et kas tekstid olid mõeldud üksteise kõrvale või üksteise alla. Kui veebilehe looja on oma lehe validaatoris järgi kontrollinud, siis võib loota, et lehitsejad saavad sellest enamvähem sarnaselt aru. Ametlikuks veebilehe tehnilise korrektsuse kontrolliks on loodud teenus aadressil http://validator.w3.org. Seal võimalik kontrollida kas juba ülesriputatud lehe korrektsust, laadida üles fail või kopeerida olemasolev HTMLi kood otse tekstiaknasse. Lihtsa väikese koodi puhul, nagu siinne alustus, on viimatine võimalus ehk mugavaim. Kui mõni viga juhtub, antakse sellest teada. Muul juhul aga teatatakse uhkesti rohelisel kirjal, et sisestatud kood on reeglipärane.



Ülesandeid

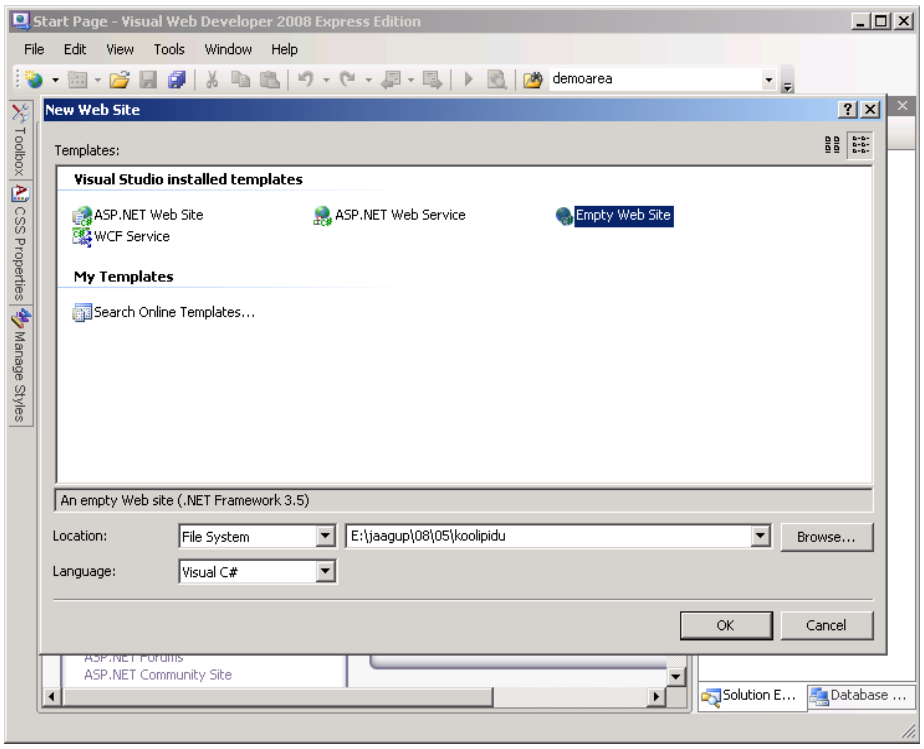
- Tee näide läbi. Vaata tulemust.
- Lisa tervitusele hüüumärk. Salvesta ja vaata veebilehitsejas tulemust.
- Veendu, et kood valideerub.
- Tekita HTML-koodi sisse viga (nt. eemalda üks < märk). Tutvu vastavate validaatori veateadetega.
- Paranda kood taas õigeks. Kopeeri validaatori antud „kvaliteedimärgi“ lõik oma koodi sisse lehe lõppu (enne </body>). Veendu, et pilt tuli nähtavale.

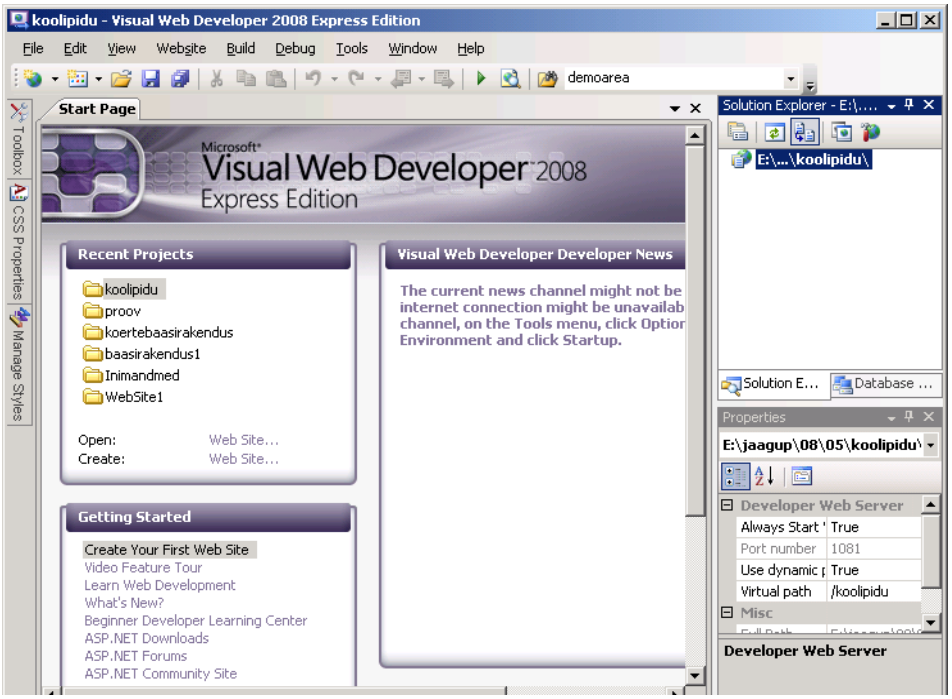
Esimene veebileht Visual Studio abil

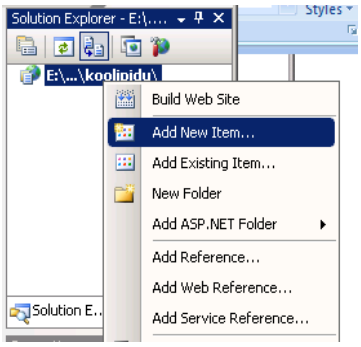
Nagu näha, võib lehti luua igasuguse tekstiredaktori abil. Piisab teksti kirjutamisest, salvestamisest ning veebilehitseja mõistab lehel olevate HTMLi käskude ning nende vahel oleva sisu põhjal tulemuse välja näidata.

HTMLi käske on aga palju ning kõik neist ei pruugi sugugi kohe tuttavad olla. Samas aga enamik meist on kujundanud tekste mõne redaktori abil, kus saab valida värve, suurust jm. Seetõttu võib esmane lehtede loomine minna libedamalt mõne selleks tarbeks mõeldud vahendi abil. Et siin materjalis tutvustatakse Microsofti veebivahendeid, jääb paratamatult ette Visual Web Developer, mil ka veebilehtede koostamiseks omaette nurk olemas.

Uue veebilehestiku loomiseks tasub valida failimenüüst „New Web Site“. Ning et liialt palju lehti silmade kirjuks muutmiseks ette valmis ei genereeritaks, siis sobib valikust Empty Web Site. Allpool tasub märkida/luua kataloog, kuhu sisse loodav lehestik tuleb.

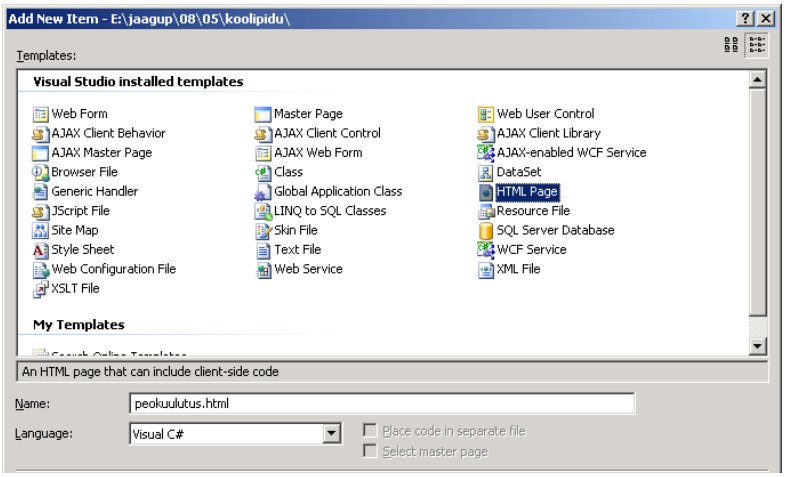




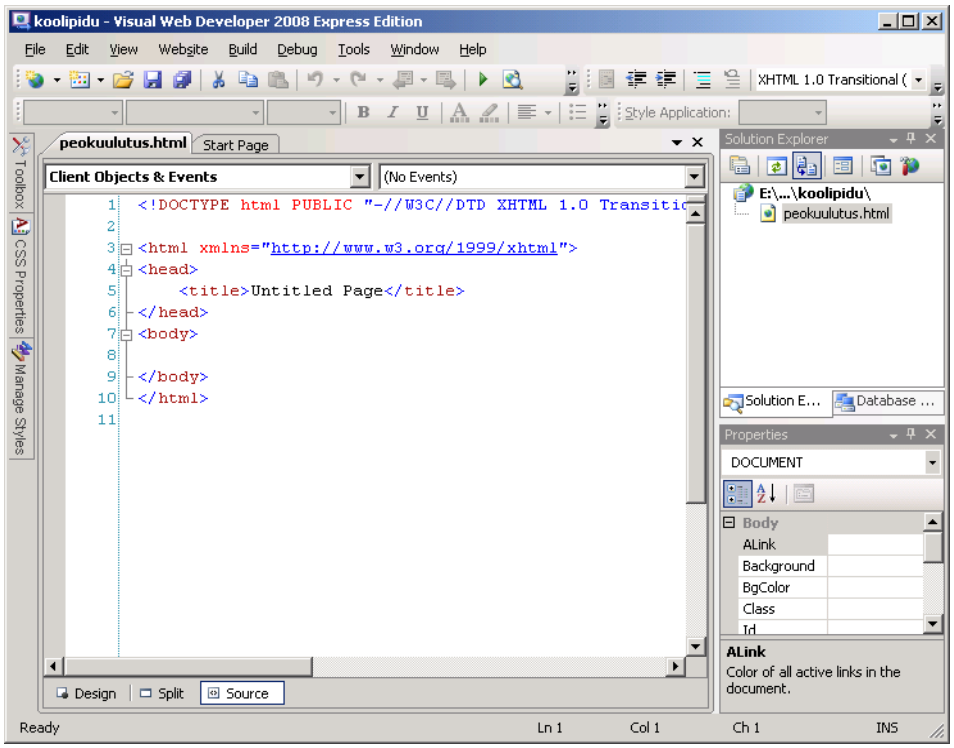


Toimetuse tulemusena tekkis masinasse vastav kataloog ilma ühegi failita. Sellest ei tasu end segada lasta. Web Developeri paremas servas võiks olla nähtaval Solution Explorer. Või kui seda mingil põhjusel seal ei paista, siis võiks aidata View menüüst valitud Solution Explorer. Selles parem hiireklõps rakenduse kataloogi nime peal ning võib valida enesele veebilehestiku uue elemendi.

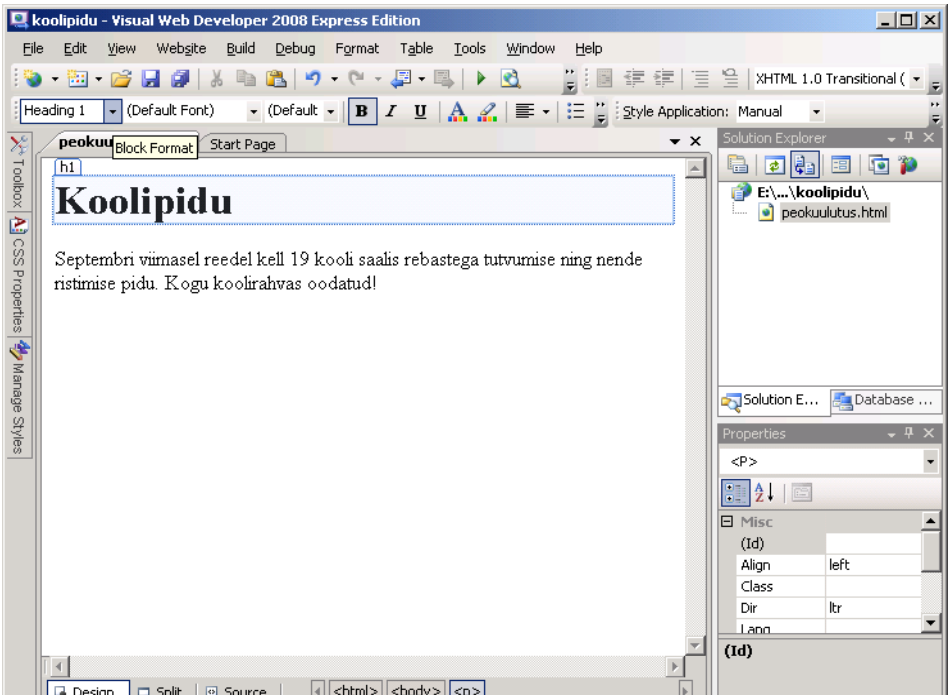
Sealt omakorda HTML-lehe ning lehe nimeks näiteks peokuulutus.html.



Tulemusena tekib HTMLi kest, mille sisse saab sobivaid teateid lisama hakata.



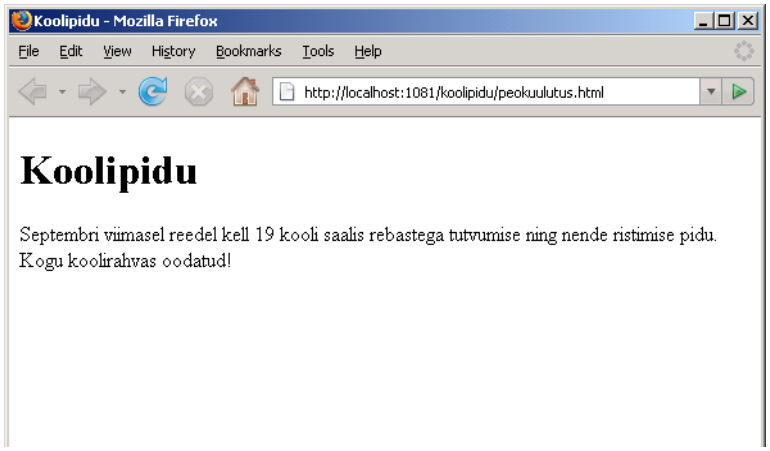
Pealkirjas oleva „Untitled Page“ asemele on mugav siin sobiv teade panna – hiljem kipub see kergesti ununema. Ning isegi tähtsates ametlikes lehtedes võib mõnikord näha kohti, kus see automaatselt pakutud „nimetu“ ilutseb. Edasi võib aga vaadete alt valida „Design“ ning saab lehte kujundada juba küllalt „tavalist“ redaktorit kasutades. Pealkirja tarbeks võib stiilide alt valida „Heading 1“ – selle tulemusena mõistetakse too rida juba ise nähtavamaks vormida.



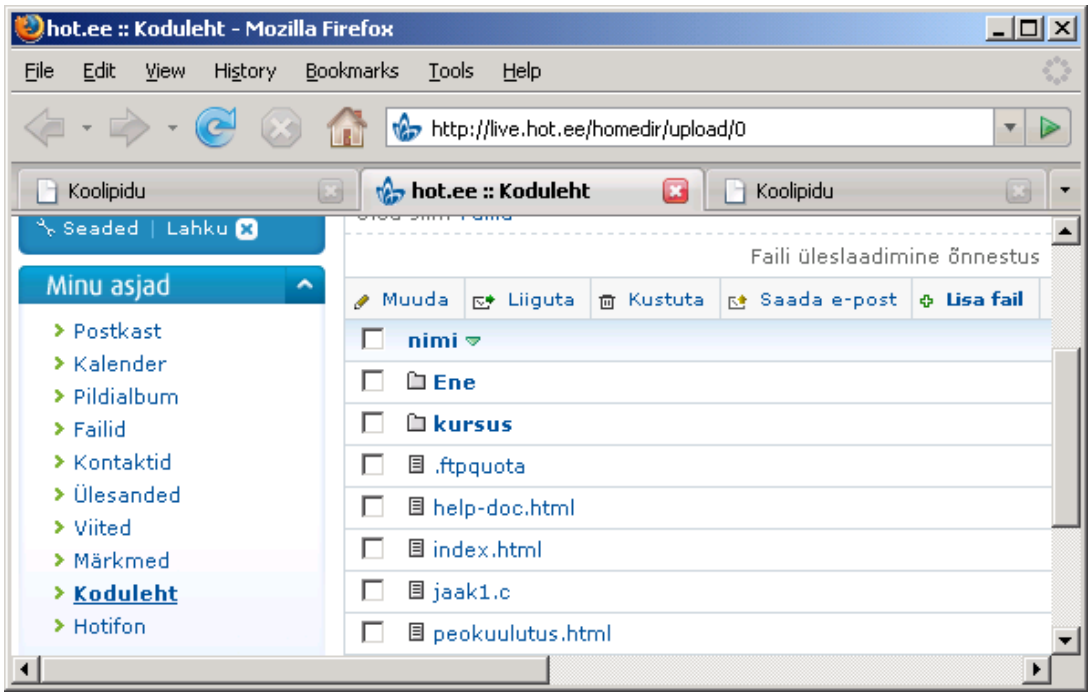
Vahepeal võib huvi pärast lähtekoodi ehk Source vaatest piiluda, mis siis lehele pealkirja märkimise tulemusena tehti. Nagu näha, pandi sõna „Koolipidu“ h1-nimeliste käskluste vahele. Samuti pandi lõigumärk p (paragraph) ümber tavalisele nähtavale tekstile. Kui lõike oleks rohkem, siis saab nad lihtsalt nõnda üksteise järgi kirjutada, igaühele alustuseks ning lõpetuseks.

```
<!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>
    <title>Koolipidu</title>
</head>
<body>
    <h1>Koolipidu</h1>
    <p>Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise
ning nende ristimise pidu. Kogu koolirahvas oodatud!</p>
</body>
</html>
```

Lehe vaatamiseks tasub Web Developeris vajutada nupule, mille kirjelduseks „View in Browser“. Tulemuseks pannakse kohalikus masinas tööle veebiserver ning veebilehitseja ja loodud serveri kaudu saab nimetatud lehte vaadata. Iseenesest poleks palja HTML-lehe tarbeks serveri käivitamine hädavajalik – seda saab vaadata ka lihtsalt failina kohalikust kataloogist. Aga kuna Web Developer pigem mõeldud siiski iga kord serveris uuesti genereeritavate lehtede loomise tarbeks, siis on talle vastav võimalus sisse ehitatud. Nii võibki lehte rahumeeli imetleda aadressilt, mis hakkab tähekombinatsiooniga http – HyperText Transfer Protocol, järgneb masina nimi, kust lehte vaadatakse. Nimi localhost tähistab kohalikku masinat. Edasi pordi ehk värati number mis vajalik, et samas masinas olevad mitmed veebiühendust pakkuvad programmid omavahel tülli ei läheks. Siis juba rakenduse nimi ja sealt seest rakenduse sees oleva faili nimi.



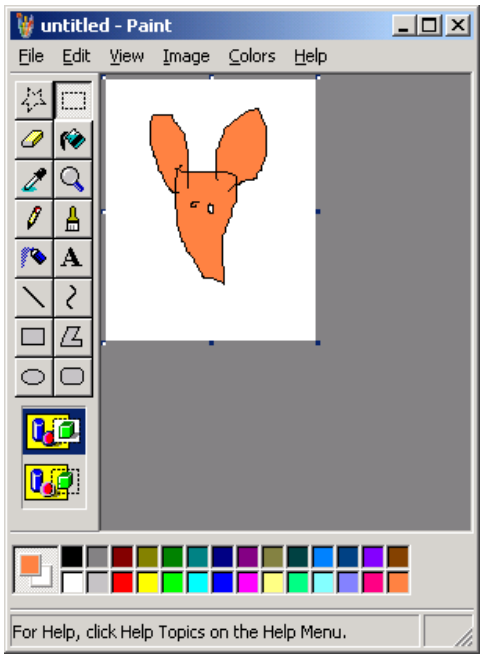
Kui tahta loodud kuulutus teistele üle võrgu ka kättesaadavaks teha, tuleb see laadida lihtsalt kohta, kus see kõigile näha. Olgu selleks õpilasele koolivõrgus eraldatud veebiruum või ka mõnes avalikus serveris asuv koht. Eestis näiteks on levinud hot.ee nimeline veebikeskkond, kus rahumeeli igaüks saab omi HTML-faile hoida ja teistele näidata. Kel konto olemas, võib Kodulehe sektsiooni vastava faili laadida.

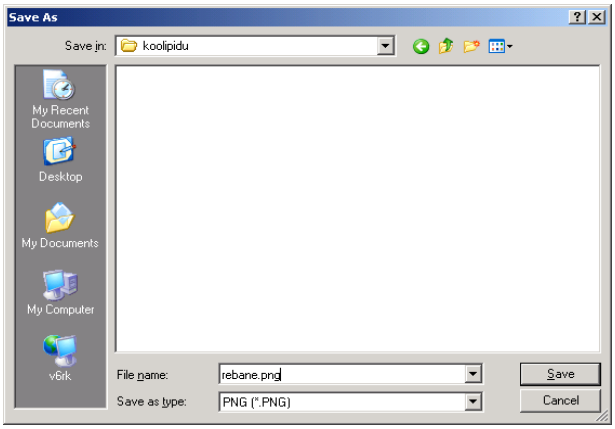


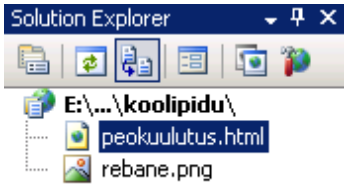
Edasi piisab juba veebilehitsejasse aadressi kirjutamisest ning igaüks võib üle võrgu vastavat teadet vaadata. Koolis saab viite panna nt. avalehele või siis huvilistele kirja teel kohale saata.

Piltide kasutamine veebilehel

Veebileht võib piirduda ainult tekstilise teabega. Tahtes seda aga ilusamaks muuta, on pildid igati omal kohal. Kes autoriõigustega ei karda pahuksisse minna, saab omale veebist igasuguseid vahvaid kujutisi muretseda. Kui aga joonis ise kokku pandud, siis ei saa kellelgi muret olla, et seda üles panna ei tohiks. Siin kuulutuse juures joonistati kokku rebase moodi kujutis. Ning et seda oleks mugav veebilehestiku juures pruukida, siis salvestati olemasoleva HTML-failiga samasse kausta. Veebis enamvähem kindlapeale näidatavad vormingud on gif, jpg ja png. Esimene neist jooniste tarbeks, teine fotodele ning viimatine saab enamvähem kõigega viisakalt hakkama. Igasugu muude vormingute nagu näiteks bmp näitamine sõltub rohkem konkreetse veebilehitseja oskustest ja seadistustest. Siin salvestati siis peokuulutus.html-iga samasse kausta pilt nimega rebane.png.

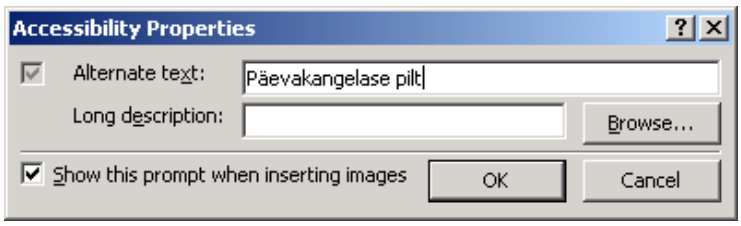






Pärast pildi salvestamist ei ilmu see veel automaatselt failide juures nähtavale. Aga kui Solution Exploreris vajutada refresh-nuppu  , siis peaks pildifaili nimi failide loetellu ilmuma.

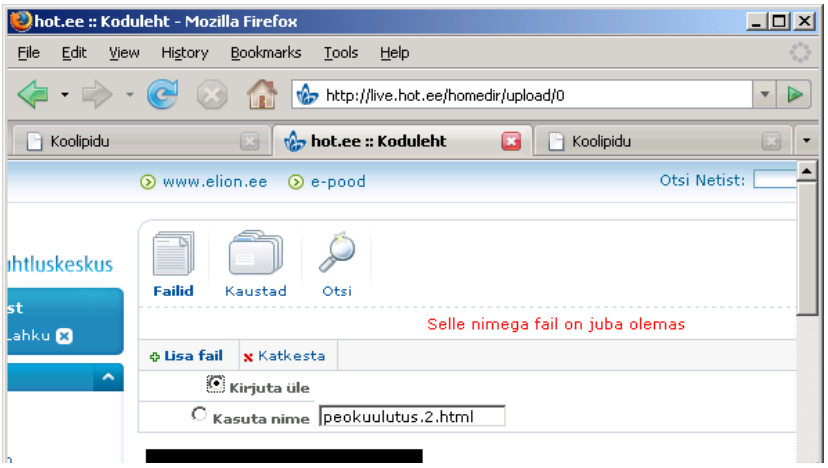
Edasi võib pildi lihtsalt lehe teksti sees sobivasse kohta – näiteks lõppu – lohistada. Keskkond võib küsida alternatiivteksti – nende jaoks, kes mingil põhjusel pilte ei saa vaadata. Näiteks, kui tegemist mobiiliekraani kaudu veebi lehitsemisega ning kiiruse ja odavuse huvides on pildid välja lülitatud.



Kui lähtekoodi piiluda, siis on näha, et tekkis uus lõik (element nimega p). Lõigu sees pilt (element nimega img) ning viimasel küljes atribuutidena mõned andmed. Tähtsam neist pildifaili nimi, siin näites rebane.png . Teiseks kohustuslikuks väljaks pildi juures on toosama alternatiivtekst, mis sisestusaknast küsituna vastavasse kohta kirjutati. Kes veebilehte koodina kirjutab, mõistab selle sobivasse alt-kohta ka ise paigutada. Ning võibki pilti vaadata.

```
<p>
    <img alt="Päevakangelase pilt" src="rebane.png"
        style="width: 134px; height: 165px" /></p>
```

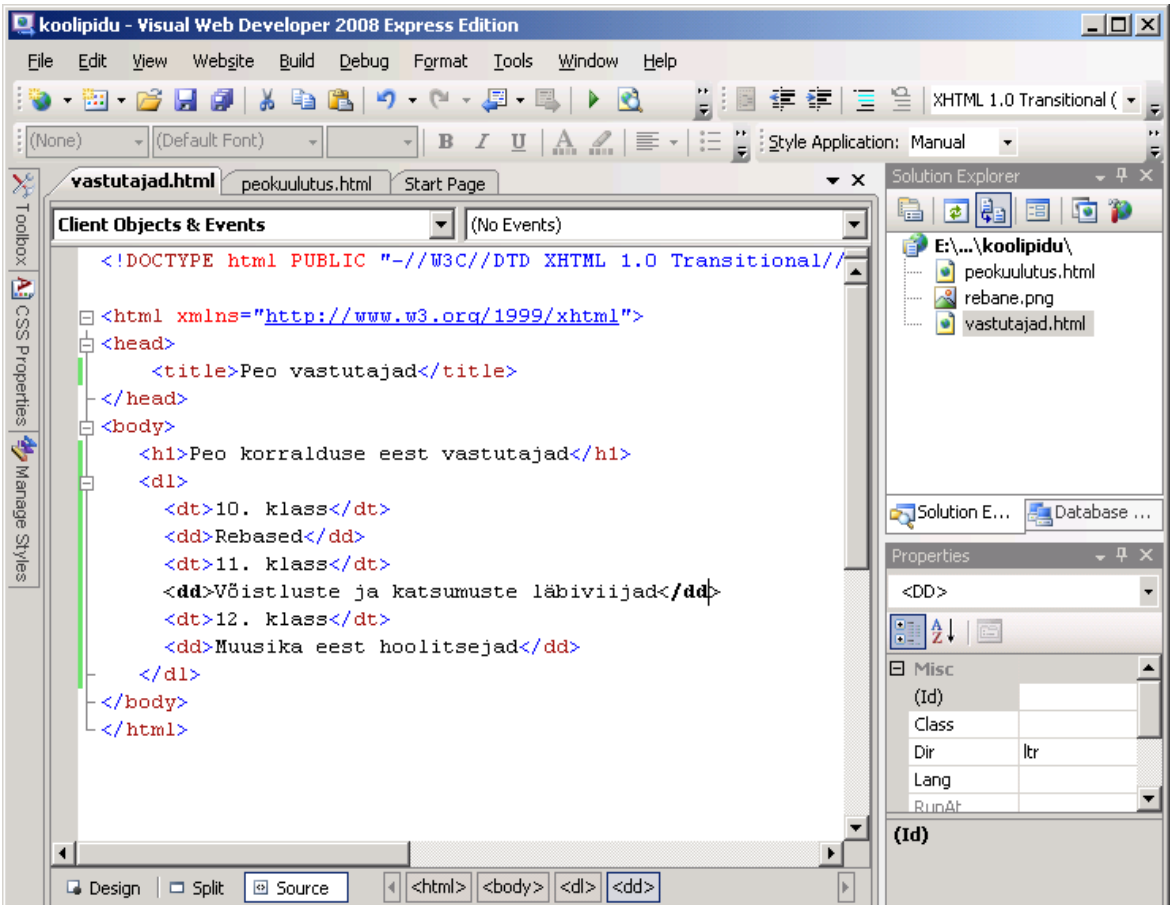
Kui nüüd pildiga kuulutus tahta kuhugi veebiserverisse nähtavale kohale üles panna, siis tuleb veebi kopeerida mõlemad failid. Nii pildifail kus rebane peal kui ka muudetud sisuga kuulutusfail, kus öeldud, kus kohal see pilt on vajalik lehel näidata. Enne kopeerimist tasub jälgida, et Web Developeris oleksid failid salvestatud. Pildi näitamine arendusvahendi lehel ei tähenda veel andmete jõudmist kettale. Et peokuulutus.html oli eelmises variandis olemas, siis mõnes kohas võidakse kopeerimisel uurida, et kas soovitakse vastavat faili üle kirjutada. Et soovime vana variandi uuega asendada, siis tuleb sellele küsimusele jaatavalt vastata.



Kui nüüd lehte uuesti vaadata, võiks pilt näha olla. Mõnel pool hoitakse veel vanu andmeid meeles – seega tuleb uue tulemuse nägemiseks mõnevõrra oodata. Või siis vajutada CTRL+SHIFT+Reload.

Seotud lehed

Eelnevast näitest paistis, et paljaks tutvustuseks piisab ühest lehest. Rohkemate andmete põhjal pole aga põhjust kõike korraga näidata – vastavalt vajadusele saab lugeja valida omale parasjagu vajalikke lehti. Enne lehtede omavahelist sidumist tuleb aga need kõigepealt valmis teha – siis on mugavam teada anda, kust kuhu liigelda saab. Siin näites lisame peokuulutuse kõrvale lehe, kus kirjas, mille eest keegi korralduse juures vastutab. Ehk siis taas Solution Exploreris rakenduse kausta juures parem hiireklõps, sealt uus HTML-leht, mil nimeks vastutajad.html. Osa kujundust on mugav disainivaates teha. Et tulemuseks aga HTML-kood, siis ei saa disainivaates teha midagi sellist, mida otse koodi sisse kirjutada ei saa. Vastupidi aga küll. Üheks selliseks mooduseks on definitsiooniloend – sõna ja tema seletus. Ehk siin on sõna ehk definitsiooni rollis klassi number, selgituseks aga nende roll peol. Kogu selle loo ülesmärkimiseks on vaja kolme eri HTML-käsklust. Käsk dl ehk definition list teatab, et nüüd loetelu algab, nüüd lõpeb. Definition term ehk dt märgib seletatava termini, dd ehk definition määrab termini vaste. Ning nende abil saab iga klassi koos oma rolliga kirja panna.



Kood otse eraldi ka.

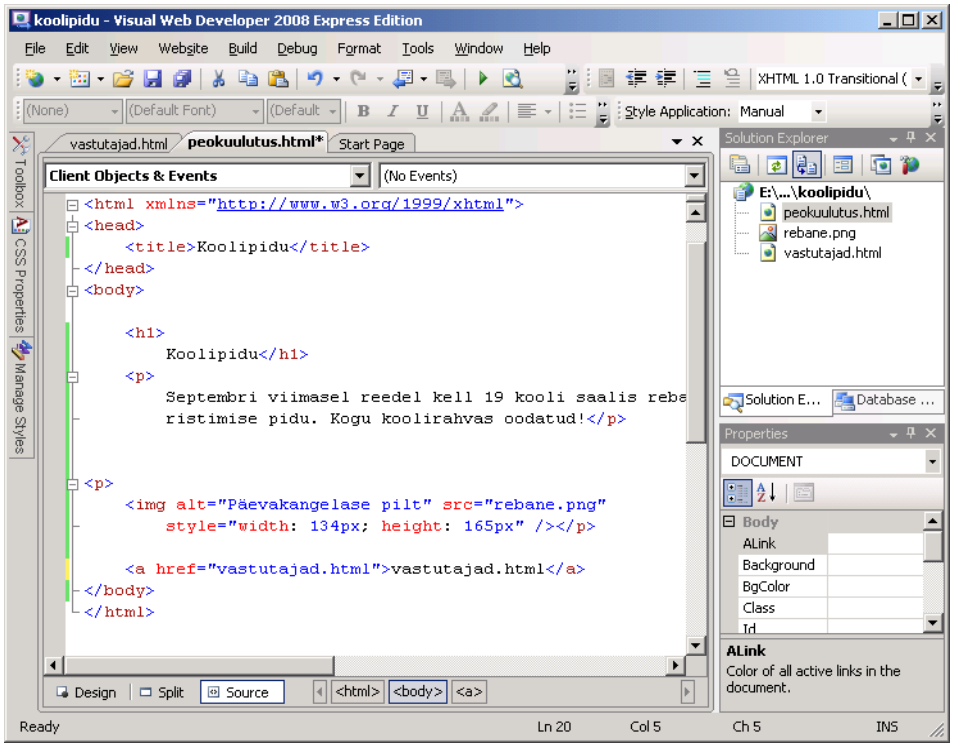
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Peo vastutajad</title>
</head>
<body>
   <h1>Peo korralduse eest vastutajad</h1>
   <dl>
     <dt>10. klass</dt>
     <dd>Rebased</dd>
     <dt>11. klass</dt>
     <dd>Võistluste ja katsumuste läbiviijad</dd>
     <dt>12. klass</dt>
     <dd>Muusika eest hoolitsejad</dd>
   </dl>
</body>
</html>
```

Peo korralduse eest vastutajad

10. klass
    Rebased
11. klass
    Võistluste ja katsumuste läbiviijad
12. klass
    Muusika eest hoolitsejad

Disainivaates on näha, et klassid on tõmmatud vasakule, nende juurde kuuluvad seletused aga mõningasse taandesse sinna kõrvale.

Kaks eraldi lehte olemas, nüüd siis tarvis nad omavahel siduda. Ja praegusel juhul soovitavalt mõlemat pidi – nii et avalehelt saaks vastutajate lehele ning viimasest omakorda tagasi avalehele. Viite loomiseks on vajalik teada anda kaks asja: mille peale vajutades viide käivitub ning kuhu vajutades satutakse. Viidetega ümber käimiseks on HTMLi sisse mõeldud element nimega a. Selline lihtne ühetäheline nimi, pärit sõnast anchor (ankur). Kõige lihtsam ühendusmoodus on avada avalehe lähtekood ning sinna sobivasse kohta lihtsalt vedada Solution Exploreri alt vastav fail. Ehk siis haaran hiirega kinni failinimest vastutajad.html ning sikutan ta lehe lähtekoodis kohta, kus võiks viide olla.



Tulemusena tekkis siis rida

```
<a href="vastutajad.html">vastutajad.html</a>
```

Atribuudi href (hyperlink reference) juures on kirjas avatava faili nimi. Elemendi a sisu (ehk siis ja vahelt on lehel nähtav tekst. Praegu seal lihtsalt sama

faili nimi, sest lohistamise peale ei mõistnud veebiredaktor sinna paremat selgitust anda. Ise aga saab sinna mõnevõrra pikemalt kirjutada. Näiteks

```
<a href="vastutajad.html">Peo osaliste tööjaotus</a>
```

mis siis veebilehel näeb välja ilusasti viitena

Terviklik avalehe kood siis

```
<!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>
    <title>Koolipidu</title>
</head>
<body>
    <h1>
        Koolipidu</h1>
    <p>
        Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise
ning nende
        ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>
    <img alt="Päevakangelase pilt" src="rebane.png"
        style="width: 134px; height: 165px" /></p>

    <a href="vastutajad.html">Peo osaliste tööjaotus</a>
```

```
</body>
</html>
```

Sarnane viide ka vastutajate lehe koodi

```
<!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>
    <title>Peo vastutajad</title>
</head>
<body>
   <h1>Peo korralduse eest vastutajad</h1>
   <dl>
     <dt>10. klass</dt>
     <dd>Rebased</dd>
     <dt>11. klass</dt>
     <dd>Võistluste ja katsumuste läbiviijad</dd>
     <dt>12. klass</dt>
     <dd>Muusika eest hoolitsejad</dd>
   </dl>

   <a href="peokuulutus.html">Avalehele</a>
</body>
</html>
```

ning juba võibki viiteid pidi kahe lehe vahet liikuma hakata.

Viited aga ei pea sugugi olema vaid paari kohaliku lehe vahel. Kuna veebiaadressi kaudu saab ligi lehtedele üle Interneti, siis piisab sobiva viite lisamisest oma lehele. Lihtsa failinime puhul saab lehitseja aru, et seda otsitakse kohalikust kataloogist. Kui aga aadressil on kirjas ees http:// (või mõni muu protokolli tähis nt. ftp), siis teatakse, et vastavat faili tuleb välisvõrgust otsima hakata. Sellest ka põhjus, miks viidetele tuleb nood http-d ette kirjutada.

Muul juhul kipub lehitseja otsima kohalikust kataloogist faili nimega www.ilm.ee (või näiteks faili nimega microsoft.com) ja ei leia kuidagi.

```
<p>
    <a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />
    Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv
riietus!
</p>
```

Käsklus br tähistab reavahetust (break). Segadust kipub vahel tekitama, miks tol elemendil on kaldkriips lõpus, kui mõneski muus kohas kipub kaldkriips olema enne elemendi nime. Selgituseks, et siin on elemendi algus ja lõpp koos, st. tegemist on käsu
</br> lühendatud kujuga. XHTML vastab XMLi reeglitele ning sealtkaudu nõutakse iga elemendi puhul algust ja lõppu. Elemendil a on näiteks sinna vahele põhjust atribuudina panna avatav aadress ning tekstilise väärtusena kasutajale nähtav jutt. Käsk br tähistab aga ainult reavahetust ning kui sinna täiendavaid andmeid (nt. rea kõrgust) ei lisata, siis piisabki vaid elemendi nimest. Mille taga kaldkriips annab teada, et sellega on vastava elemendi mõjupiirkond ühtlasi lõppenud.

Lõpetuseks siis lehitsejas töötav viidetega leht.

ja tema lähtekood

```
<!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>
```

```
    <title>Koolipidu</title>
</head>
<body>
    <h1>
        Koolipidu</h1>
    <p>
        Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise
ning nende
        ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>
    <img alt="Päevakangelase pilt" src="rebane.png"
        style="width: 134px; height: 165px" /></p>
<p>
    <a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />
    Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv
riietus!
</p>
</body>
</html>
```


http://video.msn.com/video.aspx?vid=ac6e0c0d-f8b9-46f6-ba9c-07fba46fa2fc

Ülesandeid

- Loe näide mõttega läbi.
- Koosta korvpallivõistluse tarbeks kuulutusleht.
- Vaata lehte veebilehitsejas.
- Otsi või joonista lehele sobiv pilt.
- Koosta eraldi lehekülg, kus kirjas mängivate võistkondade nimed.
- Seo lehed omavahel viidetega.
- Pane avalehele viide korvpallireeglitele. Näiteks http://www.basket.ee/uploads/docs/korvpalli_reeglid.pdf
- Otsi korvpalliga seotud pilte. Koonda nad ühele galeriilehele. Lisa viide avalehelt galeriisse ja tagasi
- Koosta iga võistkonna kohta omaette väike tutvustav leht. Avalehelt tee viide lehele, kus asub võistkondade loetelu. Loetelust saab viite abil iga võistkonna juurde. Lisa ka tagasisuunas toimivad viited.

- Võimaluse korral pane loodud lehestik üles avalikult kättesaadavasse veebiserverisse. Veendu, et kõik on ilusti nähtav. Serveri puudumisel püüa lehestik nähtavaks saada sõbra arvutis või oma arvuti teises kataloogis.

- Tee oma lehestikus mõni täiendus (näiteks lisa üks korvpallimeeskond). Hoolitse, et uus versioon oleks nähtav ka ülespanekukohas.

Astmelised laadilehed (CSS) – cascading style sheets

Lehestiku kujundamisel on mitmesuguseid mooduseid. Üksiku väikese lehe puhul on tõenäoliselt lihtsaim variant graafilise redaktori abil sobivasse kohta suurused ja värvid määrata ning ongi valmis. Kui aga soovitakse, et mitmes kohas oleks sarnane kujundus – näiteks pealkirjad kindla suuruse ja paigutusega, viited üle kogu dokumendi ühtlased, siis pikema dokumendi või lehtede komplekti korral on viisakas vastavat tüüpi andmete kujundus eraldi ühes kohas määrata ning pärast seda määrangut kasutada. Liiatigi on nõnda võimalik kogu kasvõi suurema lehestiku põhikujundust hiljem muuta ilma, et peaks ükshaaval kõiki kohti läbi käima.

HTMLi varasemates versioonides oli hulk käsklusi kujundamise tarbeks. Leiti aga, et keelt tasuks lihtsamaks teha, kuna muidu on tülikas mobiilseadmetele ja muudele väiksema arvutusvõimsusega masinatele HTML-i lugejaid teha. Samas ei tahetud loobuda mitmekülgsetest kujundusvõimalustest. Nõnda koos XHTMLi tulekuga 1999ndal aastal otsustatigi enamik kujunduskäsklusi HTMLi keele seest välja visata või vähemasti ebasoovitavaks muuta ning selle asemel soovitatakse kasutada laadi- ehk stiililehtede võimalusi. Kui seade suudab sealseid kujundussoovitusi arvestada, siis ta arvestab. Kui mitte, siis mitte. Aga vähemasti põhitekst on igale kasutajale nähtav. Samuti on CSSi juures kujunduskäsklusi ja -võimalusi tunduvalt rohkem kui ennist HTMLi küljes.

Näite kopeerimine

Õppida ja katsetada aga on kindlam näite peal. Siis saab proovida ja vaadata, mis toimub. Võtame aluseks eelmises peatükis valminud koolipeo kuulutuse näite. Kui tahta, et vana näide tervikuna alles jääb ja uue peal rahus katsetada võib, siis on vanast hea koopia teha. Ning siinse veebirakenduse puhul tasub koopia teha tervest kataloogist. Enne tal nimeks koolipidu, uueks nimeks paneme koolipidu2. Tervikrakenduse avamiseks aitab Web Developeri failimenüü käsklus „Open Web Site“. Võidakse algul küsida, et kas soovime rakenduse versiooninumbrit muuta. Kuna aga tegemist puhaste HTML-lehtedega ilma mingi konfiguratsioonifailita, siis tasub pigem vastata, et ei soovi. HTMLid ikka igal pool ühesugused ning lihtsam, kui meile koodi ise juurde ei genereerita.

Pildi suuruse muutmine

Kui rakendust piiluda, siis ühes kohas ongi keskkond juba stiilikäskluse genereerinud. Pildi juures paistavad olema märgitud mõõdud

```
<img alt="Päevakangelase pilt" src="rebane.png"
        style="width: 134px; height: 165px" />
```

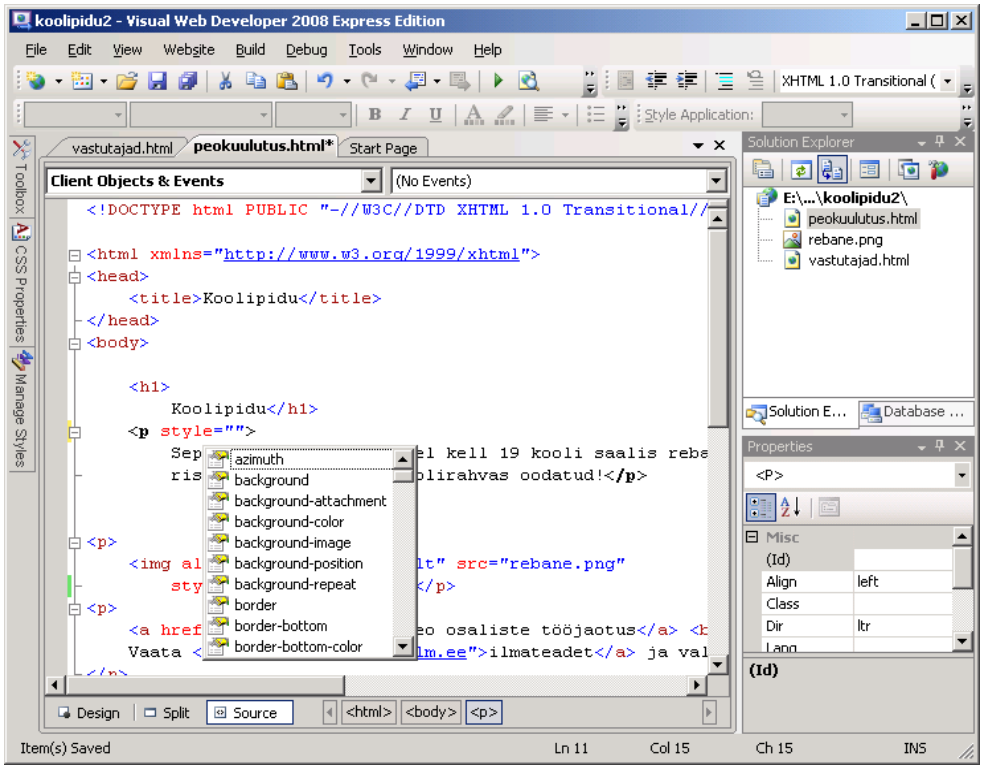
Tasub proovida neid muuta ja veenduda, et pilt ka rakenduse juures ekraanil muutub.

`style="width: 50px; height: 165px" juures näiteks pressiti pilt kokku`

`lihtsalt style="width: 50px;" juures aga määratakse laius kindalt paika, kõrgus võetakse proportsioonis vastavalt laiusele.`

Käskluste valik

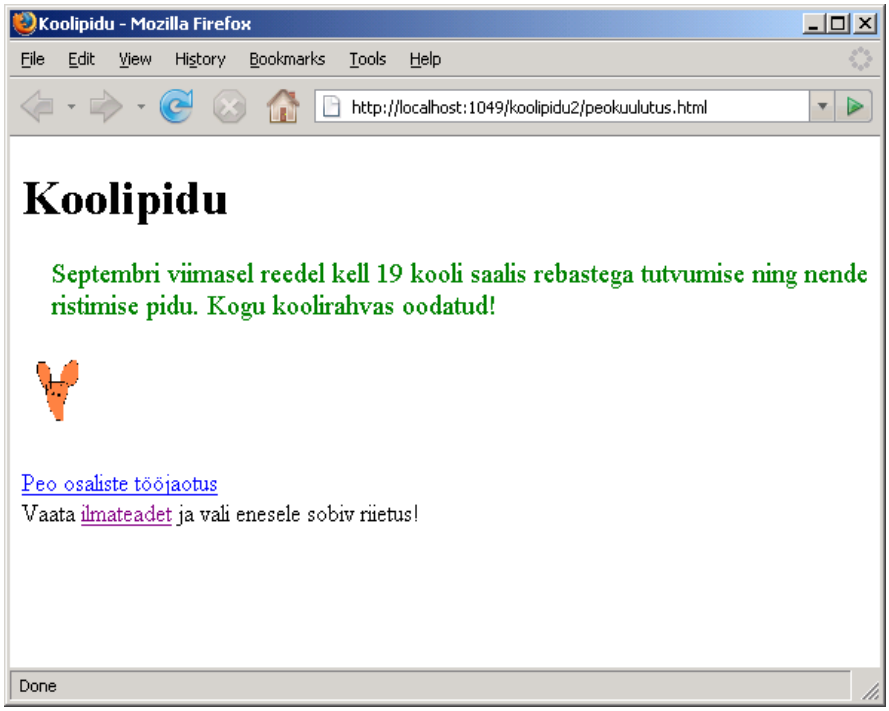
Laadikäskluse võib aga pea iga elemendi külge kirjutada. Ning keskkond on piisavalt tark, et sinna võimalikke käske juurde pakkuma hakata. Ehk siis kui esimese lõigu juurde sai kirjutatud style=“““ , siis sinna juurde tekkis valikmenüü päris mitmesuguste käskudega, millest saab hakata omale sobivaid kombineerima. Nagu aimata võib – backgroundiga algavad käsud määravad tausta, borderiga jooned, color seab esiplaani värvi, font ja tekst teksti fondi, suuruse ja joondusega seotu.



Määrasin näiteks kuulutuselõigule omadused, et kaugus vasakust servast on 20 pikslit (ekraanipunkti), tekst on suur (large) ning värv roheline.

```
<p style="margin-left: 20px; font-size: large;
color:Green">Septembri viimasel reedel kell 19 kooli saalis
rebastega tutvumise ning nende  ristimise pidu. Kogu koolirahvas
oodatud!</p>
```

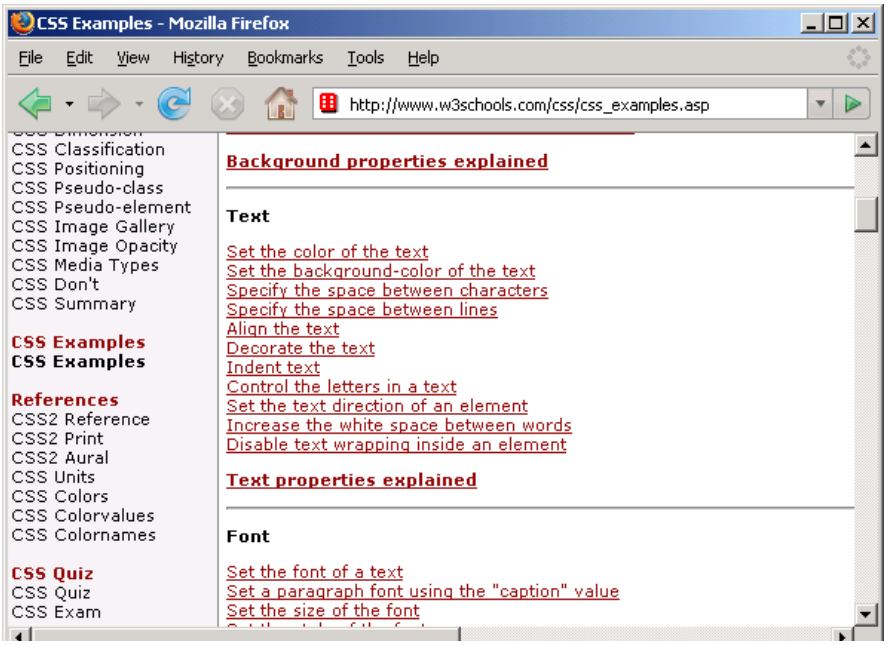
Tulemust näeb pildil:



Valmisnäidete kasutamine

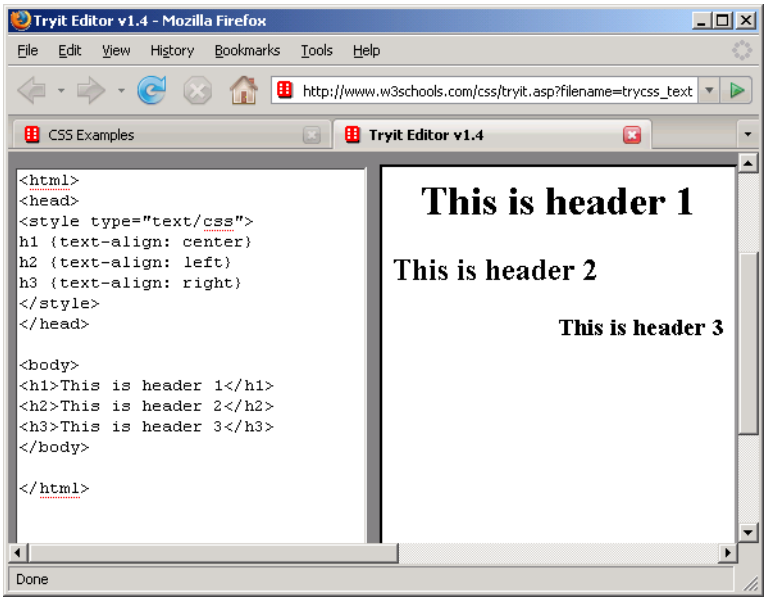
Edasi on juba proovimise ja katsetamise asi. Hulga ilusaid näiteid leiab veebiõpetuste lehelt w3schools.com aadressilt http://www.w3schools.com/css/css_examples.asp. Tuleb vaid omale sobiv jupp leida ning töötavana oma lehele üle kanda.

Muude hulgas paistab teksti joondamise näide (align the text)



Sealt leiab järgneva näite

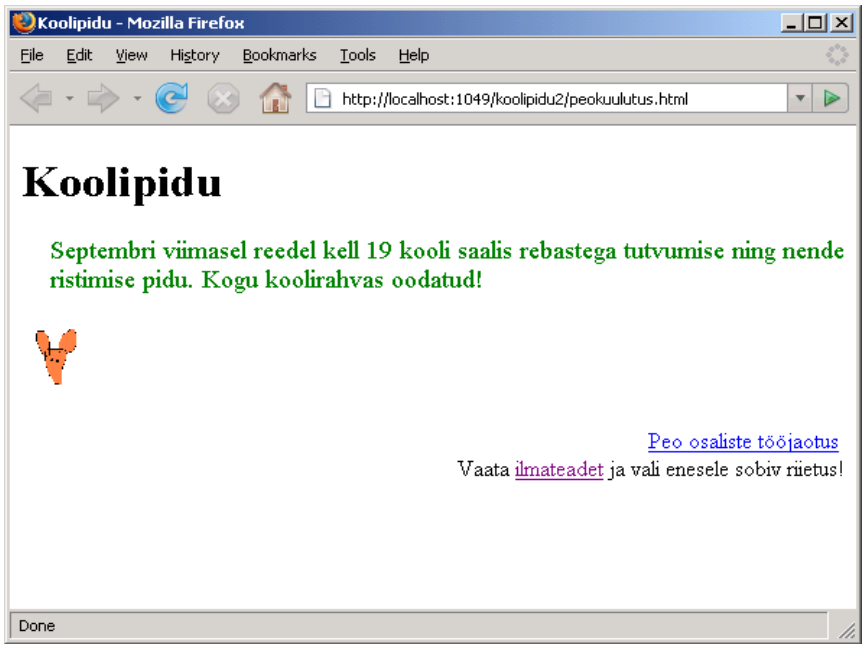
```
<html>

<head>

<style type="text/css">

h1 {text-align: center}

h2 {text-align: left}

h3 {text-align: right}

</style>

</head>

<body>

<h1>This is header 1</h1>

<h2>This is header 2</h2>

<h3>This is header 3</h3>

</body>

</html>
```



Edasi tuleb nuputada, kuidas näitest omale kasulik rida ära näpata. Siin paistavad laadid olema kirjas lehe päiseosas (head). Kui praegu aga katsetame vaid ühele lõigule omaduste andmist, siis on meil vaja vaid sobiv käsklus õigesse kohta tõsta. Siin on märgitud kolme taseme pealkirjad. Ning ülal on igaühele neist määratud, kus ta paikneb. Teksti paremale joondamise käsuks on siis järelikult text-align:right. Tahtes näiteks tööjaotuse ja ilmateate lõigu paremale joondada, kopeerime vastava käsu sinna lõigu juurde.

```
<p style="text-align: right">
    <a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />
    Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele
sobiv riietus!
</p>
```

ning pärast salvestamist võibki alumist teksti nõnda paremale joondatult vaadata.

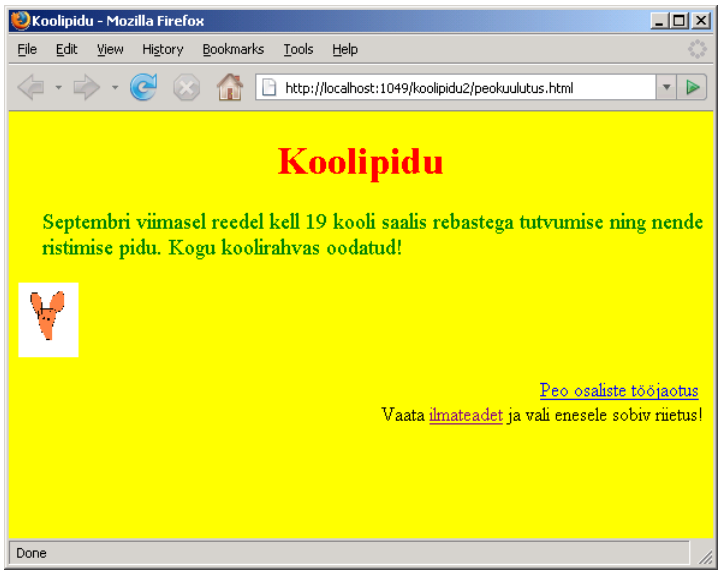


Määramine päises

Elementide omadused aga saab määrata ka lehe päiseosas – samuti nagu w3schools.com näite juures tehti. Selleks head osa sisse element style, sinna määratava kujundusega HTMLi käskude loetelu ning iga elemendi taha looksulgudes ja semikoolonitega eraldatult need käsud, mis vastava elemendi külge tahetakse panna.

```
<style type="text/css">
   body{background-color: Yellow}
   h1{text-align: center; color: Red}
</style>
```

teatab siis, et lehe sisu (body) on kollase taustaga. Esimese taseme pealkiri (h1) asetseb keskel ning on punast värvi.



Ning selguse huvides ka kogu avalehe kood:

```
<!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>
    <title>Koolipidu</title>
    <style type="text/css">
       body{background-color: Yellow}
       h1{text-align: center; color: Red}
    </style>
</head>
<body>
    <h1>Koolipidu</h1>
    <p style="margin-left: 20px; font-size: large; color:Green">
        Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise
ning nende ristimise pidu. Kogu koolirahvas oodatud!</p>
<p>
    <img alt="Päevakangelase pilt" src="rebane.png"
        style="width: 50px;" /></p>
<p style="text-align: right">
    <a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />
    Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv
riietus!
</p>
</body>
</html>
```

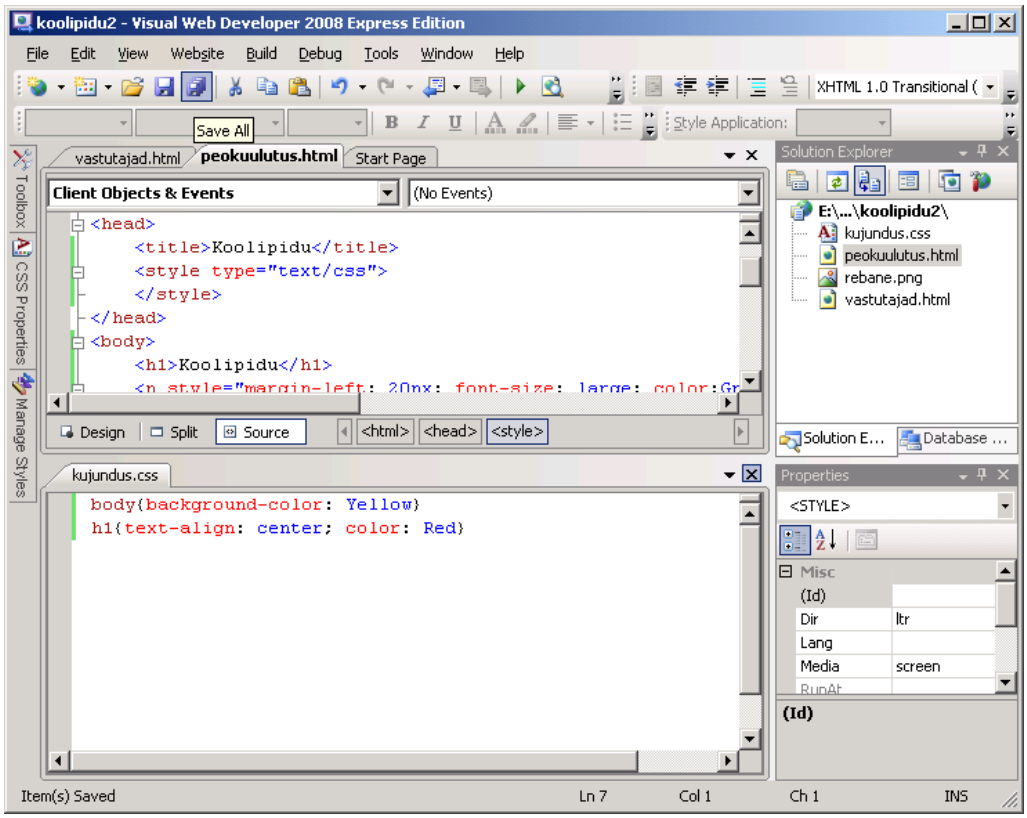
Laadileht eraldi failis

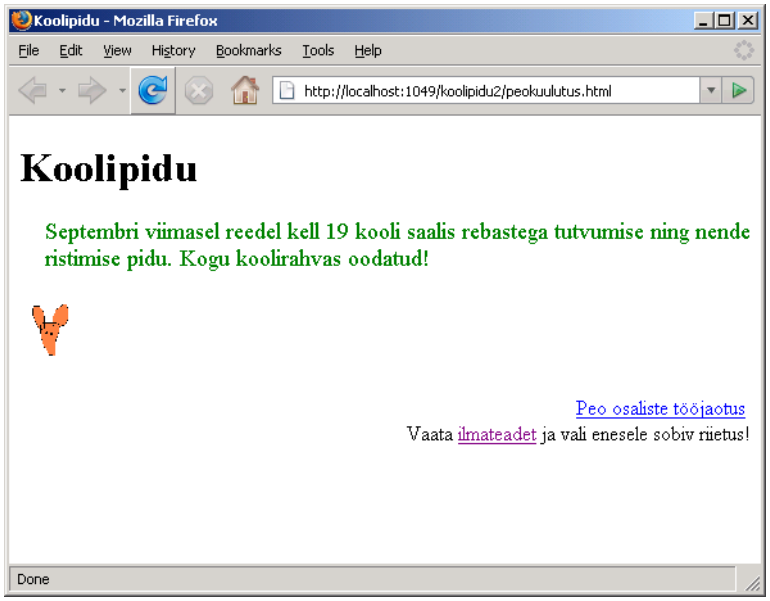
Lihtsa teksti puhul piisab kujunduse määramisest lõigu või sõna juures. Pikemal lehel tasub vähemasti korduvate elementide sõnad panna päisesse. Kui aga tahta sarnast kujundust kasutada mitme lehe juures, siis on viisakas kujunduskäsud tõsta eraldi faili ning need siis igal

lehel sisse lugeda. Nii mõjuvad kujundusfaili käsud kogu lehestikule ning näiteks kõigi lehtede taust on kerge asendada sobivaga. Viimatises näites on määratud lehe taust ning esimese taseme pealkirja andmed.

```
body{background-color: Yellow}
h1{text-align: center; color: Red}
```

Nende jaoks tekitamegi eraldi faili. Solution Explorer -> paremklõps rakenduse nimel -> Add New Item -> Style Sheet ning nimeks näiteks kujundus.css. Viimasesse on juba „seemneks“ pandud body-elemendi nimi. Kopeerin aga julgesti kogu olemasolevad kaks kujundusrida sinna faili ning algsest kustutan ära.



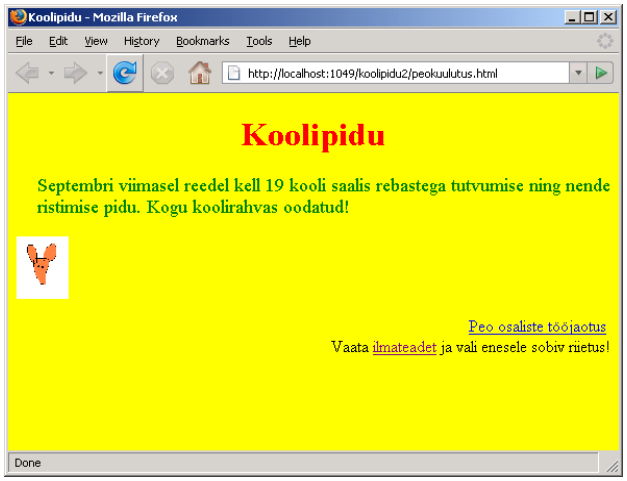


Nõnda läheb ülal määratud kujundus esimese hooga veebilehelt kaduma. Nii et veebilehitsejas paistab endine valge taustaga kuulutus, kus pealkiri on vasakul.

Edasi aga tasub kuulutuslehel öelda, et kujundus loetakse eraldi failist sisse. Selleks on CSSis käsk @import. Ehk siis päiseosas annan teada:

```
<style type="text/css">

   @import "kujundus.css";

</style>
```

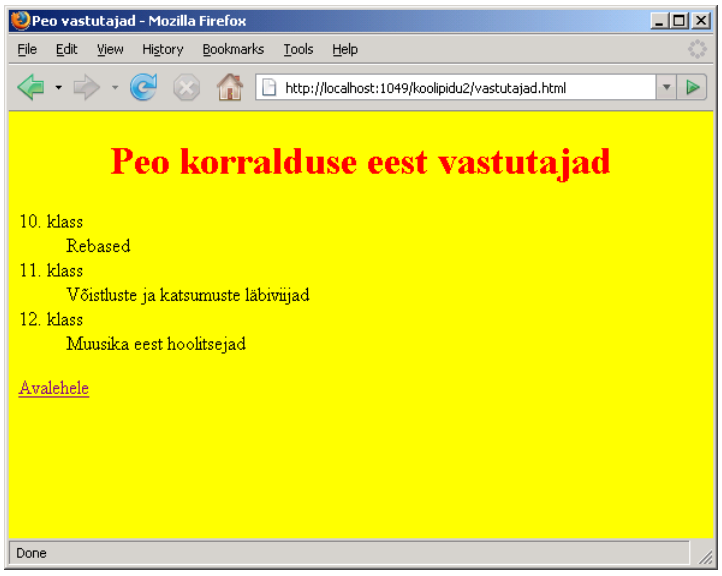
ning leht on jälle kollane.



Kui nüüd soovida sama kollase tausta ja punase pealkirjaga kujundus panna ka tööjaotuse lehele, pole muud, kui sama @import ka sinna lisada.

```
<!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>
    <title>Peo vastutajad</title>
    <style type="text/css">
      @import "kujundus.css";
    </style>
</head>
<body>
   <h1>Peo korralduse eest vastutajad</h1>
   <dl>
     <dt>10. klass</dt>
     <dd>Rebased</dd>
     <dt>11. klass</dt>
     <dd>Võistluste ja katsumuste läbiviijad</dd>
     <dt>12. klass</dt>
     <dd>Muusika eest hoolitsejad</dd>
   </dl>

   <a href="peokuulutus.html">Avalehele</a>
</body>
</html>
```



Kujundusklass

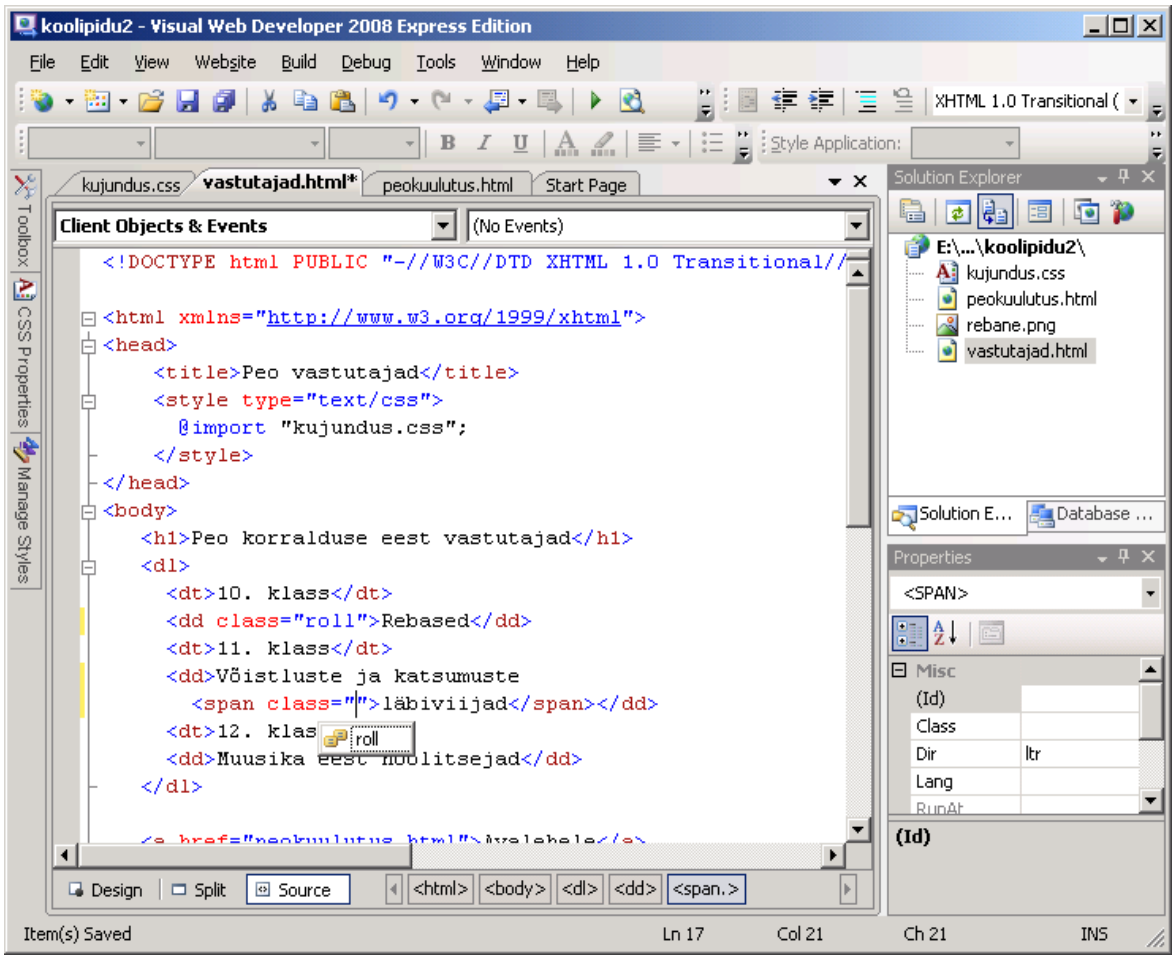
Kujundusomadusi saab määrata elemenditüüpide kaupa nagu eelnenud näitest näha: kõik esimese taseme pealkirjad paigutati keskele ja värviti punaseks. Mõningaid omadusi aga soovitakse panna ainult mõnede vastavate elementide juurde või siis sootuks erisuguste elementide külge. Näiteks soovitakse kõik uued teated lehel eraldi ära märkida sõltumata sellest, kas teated on tabeli, loetelu, lõigu või mõne muu tehnilise vahendi abil lehele kuvatud. Siinses väljamõeldud näites sobib eraldi rõhutada vastavast klassist osalejate roll. 10. klassi rahvas on rebased, 11. klassi omad peo võistluste läbiviijad. Kujundusfailis võib defineerida rolli klassi, edasi juba määrata, milline sõna või lõik selle klassi järgi kujundatakse. Ning kui juhtub, et kujundusklass tahetakse kinnitada osa külge, mis pole eraldi elemendiga piiritletud, siis piiritlemiseks sobib element nimega span. Element ise ei tee midagi, küll aga saab sinna külge kujunduskäsklusi ja –klasse kinnitada. Lisan kujundusfaili rea

```
.roll{text-decoration: underline;}
```

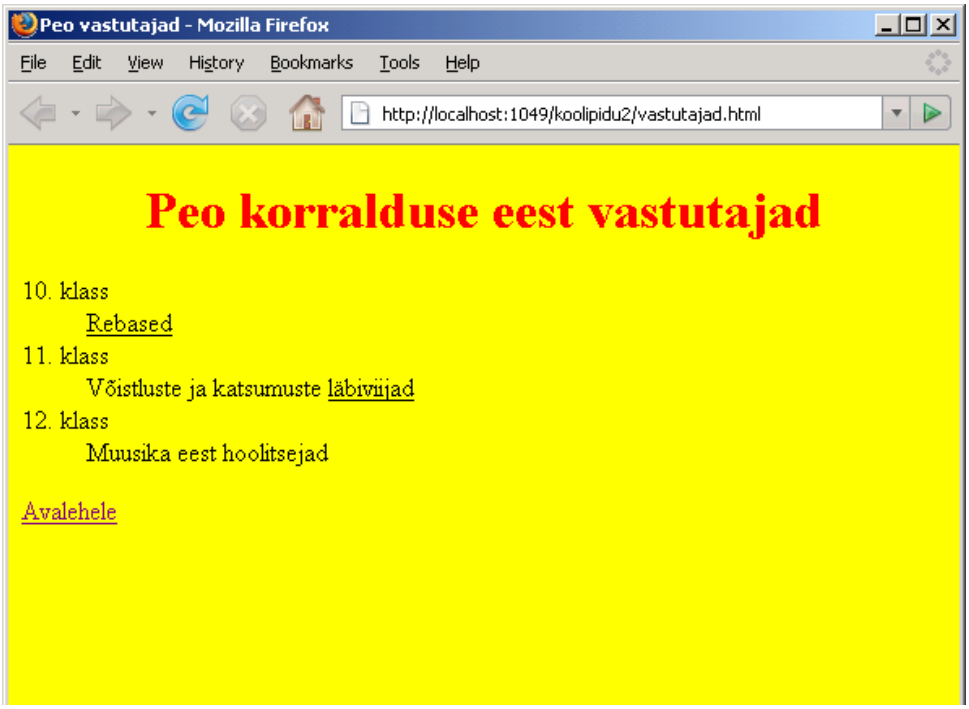
ning vastutajate loetelu juures märgin ära, millised kohad tähistavad rolli.

```
<dl>
   <dt>10. klass</dt>
   <dd class="roll">Rebased</dd>
   <dt>11. klass</dt>
   <dd>Võistluste ja katsumuste
     <span class="roll">läbiviijad</span></dd>
   <dt>12. klass</dt>
   <dd>Muusika eest hoolitsejad</dd>
 </dl>
```

Nagu näha, mõistab arenduskeskkond olemasoleva(d) klassi(d) ise välja pakkuda.



Ning ongi rolli tähistavad kohad alla joonitud.



Ülesandeid

- Loo oma korvpallimeeskonna lehestikust koopia või koosta ise väike lihtne lehestik.
- Värvi mõni lõik ja mõne lõigu taust, katseta teksti suuruse ja paigutusega.
- Tutvu http://www.w3schools.com css-i näidetega. Katsu sealt vähemasti kümmekond enesele üle võtta (nt borderi abil jooned koos värvi ja laiusega, joondus, rasvasus (font-weight)).
- Määra lehe taust ja vaba serva laius (margin-left) päises. Katseta.
- Tõsta need määrangud eraldi laadifaili. Veendu, et algsest lehest on kujundus kadunud.
- Ühenda veebileht ja laadifaili @import käsu abil
- Ühenda sama laadifail ka teise veebilehega ning veendu, et kujundus tuli mõlemasse üle.
- Muuda laadifailis lehe tausta ja veendu, et see mõjub mõlemale lehele.
- Loo klass nt. noortemeeskondade tähistamiseks. Sealne tekst rohelise värviga. Tähista noortemeeskondade nimed vastava klassiga. Veendu, et kõikidel lehtedel on nende

nimed rohelised. Muuda noortemeeskondade klassi nõnda, et roheline pole mitte tekstivärv vaid taustavärv. Veendu muutuse toimumises.

- Koosta täiesti uus lehestik näiteks kavandatava matka kohta. Märgi paberile üles teemad, mis võiksid lehestikus olla. Koosta paberile struktuur, kust lehelt kuhu võiks liikuda, mõtle lehtedele sobivad pealkirjad ning failinimed. Joonista valmis eskiisid, millised võiksid lehed välja näha – st. kas kõik lehed ühesuguse plaaniga või nt. avaleht ja menüülehed teistest erinevad. Koosta iga lehetüübi jaoks blankett. Stiilikäsklused koonda eraldi kujundusfaili, mis igasse blanketti imporditakse. Tee blankettidest iga lehe jaoks koopia, kuhu ainult märgitakse teema, näiteks „siia tuleb matka teise päeva söögikoha kirjeldus“. Seo lehed omavahel viidetega. Joonista matka marsruut kaardile ning pane ka see leht lehestikku. Palu naabril otsida üles soovitud leht ning vaata, kui kergesti ta selle menüüde kaudu leidis. Vajadusel lisa selgitusi või keerukamal juhul muuda oma lehestiku struktuuri. Mängi kujundusfailiga, et lehestik võimalikult meeldiv välja näeks. Täida lehed võimalikult meeldivalt ja usutavalt. Ning lõpetuseks – kui lehestik võimalikult hea ja nii omale kui ka sõpradele huvitav, siis tasub see matk võimaluse korral ka teoks teha :-)

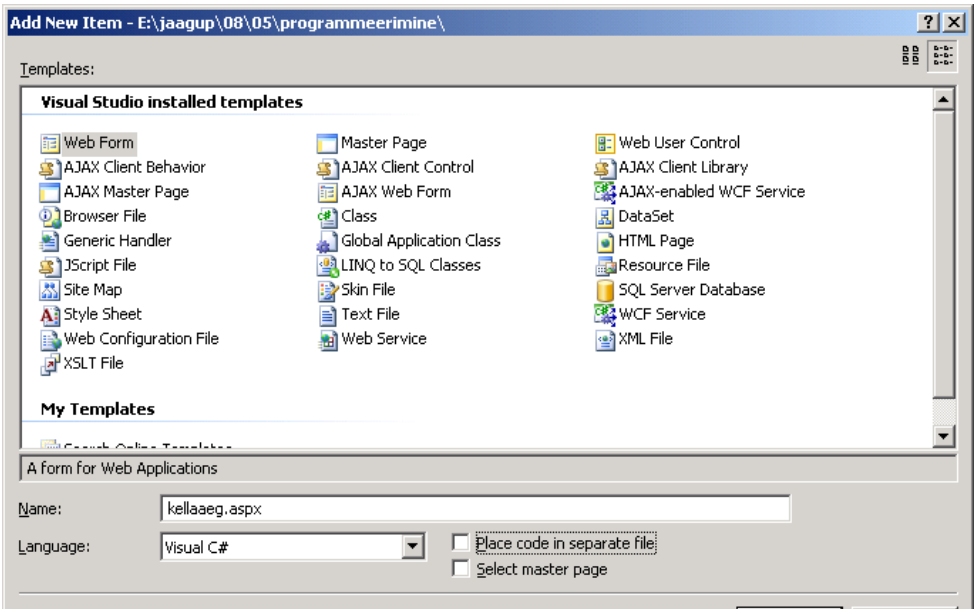
Programmeeritavad veebilehed

Eelnevas peatükis kokku pandud veebilehti nimetatakse staatilisteks. See tähendab, et ükskõik kes või millal seda lehte vaatab – leht on ikka sama sisuga. Ning alles siis, kui autor midagi muudab võib lehe peal uut sisu nägema hakata. Nagu näha, nii peokuulutuse kui matkalehe kannatab sellisena täiesti kokku panna ning sageli polegi teabe edastamiseks rohkemat vaja. Selline veeb oli aga küllalt viisakal tasemel juba 1997ndaks aastaks olemas, edasi on jõudsalt mitmesuguseid lisavõimalusi arendatud, et just konkreetsele kasutajale meelepärane olla, või siis kasutajate sisestatud andmeid töödelda nõnda, et sellest midagi suurt ja ilusat sünniks. Kogu ASP.NET on taoliste võimaluste tarbeks loodud. Staatilisi lehti võiks südamerahus kirjutada ükskõik millega ja panna üles ükskõik kuhu. Visual Web Developeri juures on „hariliku“ lehe kujundusvahendid lihtsalt üks killuke paljude programmeerimisvahendite hulgas.

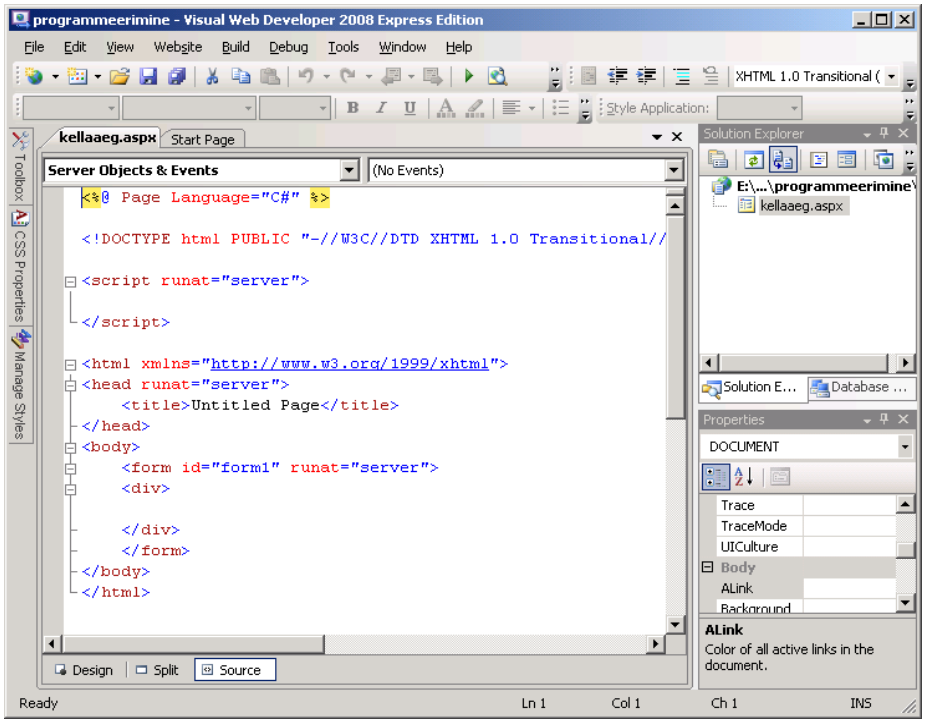
Programmeeritaval veebilehel e. aspx lehel on samuti võimalik kasutada staatilisi HTML elemente, kuid lisaks neile avaneb võimalus kasutada programmelt kontrollitavaid ja hallatavaid ASP.NET elemente e. ServerControl’e

Kellaaeg

Programmeerimisvahendite katsetamiseks loon uue tühja veebiprojekti (failimenüü → New Web Site → Empty Web Site, nimeks programmeerimine). Solution Explorerist taas New Item, aga sedakorda Web Form. Nimeks kellaaeg.aspx. Keeleks edasise huvides Visual C# ning hoolitsen, et „Place code in separate file“ juures ei oleks linnukest – sedaviisi ei teki esialgu tarbetut lisafaili.



Tekkis ilus lehe blankett, kuhu saab tasapisi omi andmeid kirjutama hakata.



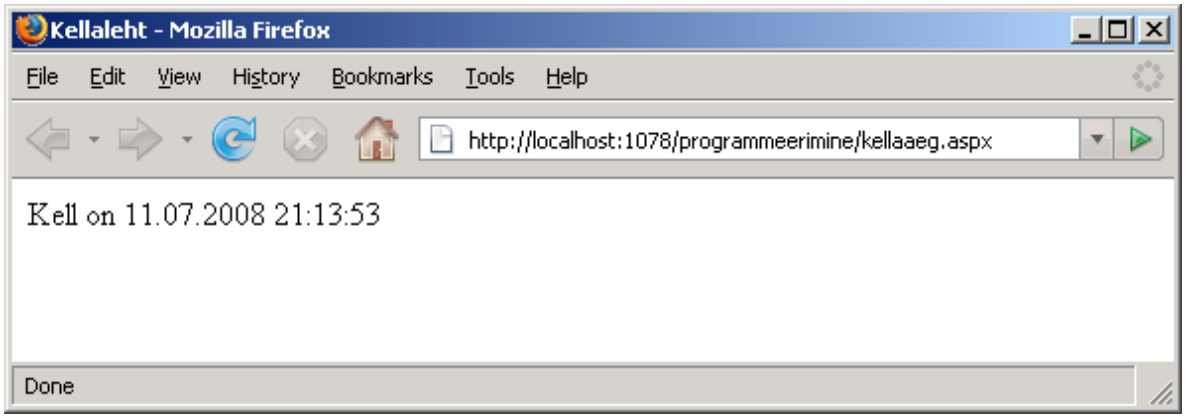
Lähemal uurimisel paistab, et sealne kood nagu oleks HTML ja nagu ei oleks ka. !DOCTYPE on täiesti olemas. Samuti <html> oma alguse ja lõpuga </html>. Ja <head> ja <body>. Aga üles on ilmunud salapärane rida

```
<%@ Page Language="C#" %>
```

Ning päris mitmel pool on juures runat="server". Nende järgi võib aimata, et tõenäoliselt on tegemist ASP.NETi lehega. Ehk pea kõik, mis on HTMLina, saadetakse otse veebilehitsejasse. Kus aga juures kirjas runat="server", selle osaga võib lehe serveerimise käigus midagi juhtuda – sinna võidakse miskit programmikoodi abil juurde kirjutada või seal midagi muuta. Samuti on võimalik otse HTMLi koodi programmi kaudu midagi juurde kirjutada. Viimast võimalust ei peeta küll kuigi viisakaks. Aga kuna vanem ASP just nõndaviisi käis ning nii on lihtsaid asju kerge näidata, siis vastav alustusnäide ka siia. DateTime.Now annab praeguse kellaaja, Kell on <%=DateTime.Now %> abil trükitakse see just sinna kohta lehele. Nii et siinne lihtne lehe kood

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Kellaleht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      Kell on <%=DateTime.Now %>
    </div>
    </form>
</body>
</html>
```

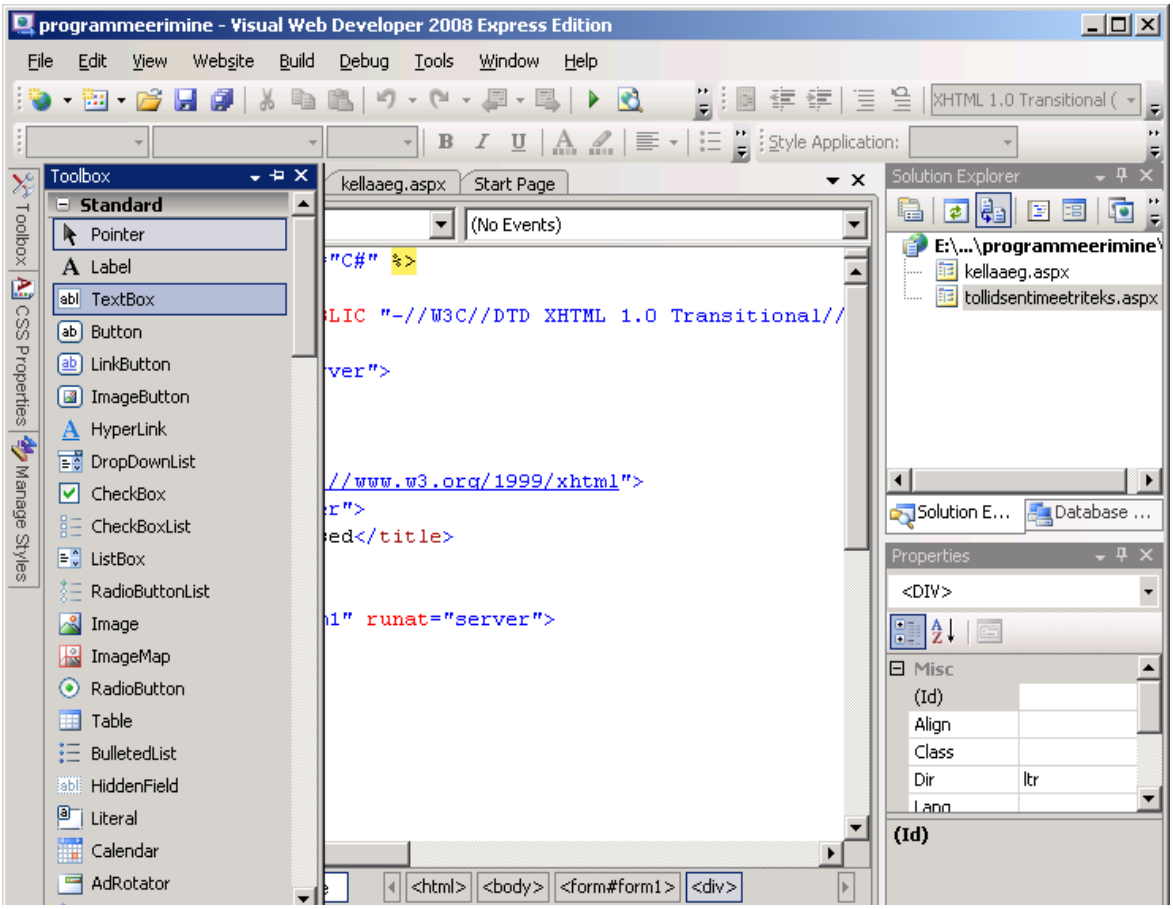
annab kokku lehe, kus nähtaval praegune kellaaeg.



Tegemist siis kõige ehtsama dünaamilise veebilehega – iga kord kui avad, on sisu uus.

Arvutav veebileht

Inimeselt andmete kättesaamiseks tuleb luua sisestuskoht. Üsna lihtne vahend selleks on tekstiväli. Sisestus- ja muid elemente saab valida tööriistakastist (Toolbox). Kui seda pole parajasti näha, siis View-menüüst saab selle esile kutsuda. Lohistades tekstikasti HTML-koodis sobivasse kohta, ilmub sinna tekst, mille abil ASP.NET teab veebilehe avamisel tekstikasti HTML-käskluse kuvada.



Automaatselt pannakse elementidele ka id-d. Nende järgi on hiljem programmikoodis võimalik nende poole pöörduda. Ning runat="server" jällegi iga elemendi külge, et server sellega midagi peale hakata tohiks.

```
<form id="form1" runat="server">
    <div>
      Tollide arv:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

    </div>
 </form>
```

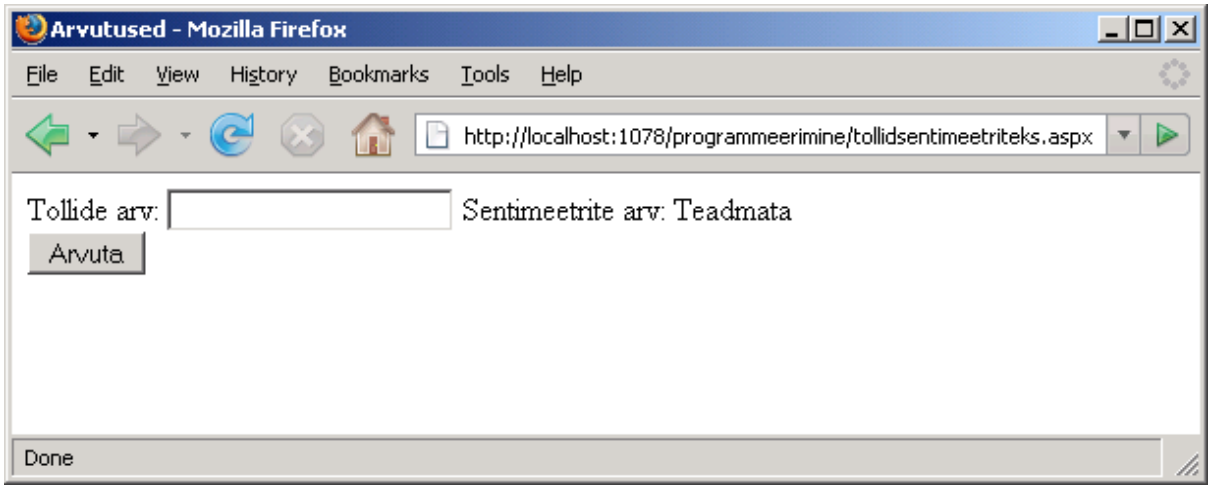
Sentimeetrites oleva tulemuse vaatamiseks piisab kohast, kuhu programmiga pääseb kirjutama. Tekstiväli lubab üldjuhul ka inimesel andmeid sisestada. Paljalt andmete näitamiseks on aga loodud element nimega Label. Selle siis lehele lohistamegi ning tulemuseks on järgnev koodilõik:

```
    <form id="form1" runat="server">
    <div>
      Tollide arv:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      Sentimeetrite arv:
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

```
    </div>
    </form>
```

Et millegi peale oleks võimalik arvutama hakata, selleks tasub lehele lisada nupp – element tüübist Button. Ning et vastusesildile ei jääks sõna „Label“ vaid midagi selgitavamat, siis määrasin sinna teksti „Teadmata“, samuti nupule teksti „Arvuta“.

```
    <form id="form1" runat="server">
    <div>
      Tollide arv:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      Sentimeetrite arv:
        <asp:Label ID="Label1" runat="server" Text="Teadmata"></asp:Label>
<br />

    </div>
    <asp:Button ID="Button1" runat="server" Text="Arvuta" />
    </form>
```



Nagu aimata, ei juhtu nupuvajutuse peale veel midagi – arvutile tuleb enne kõik asjad ette öelda. Saab küll tekstivälja kirjutada ning nupule vajutada, aga vastust ekraanile ei ilmu.

Vastuse leidmiseks tuleb nupu külge programmikoodi lõik siduda. Lihtsaimaks mooduseks on minna disainivaatesse ning vajutada nupule topeltklõps.

Tulemuse peale tehti koodis mõningad automaatsed täiendused.

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
    <title>Arvutused</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      Tollide arv:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      Sentimeetrite arv:
        <asp:Label ID="Label1" runat="server" Text="Teadmata"></asp:Label>
<br />
    </div>
    <asp:Button ID="Button1" runat="server" Text="Arvuta"
onclick="Button1_Click" />
    </form>
</body>
</html>
```

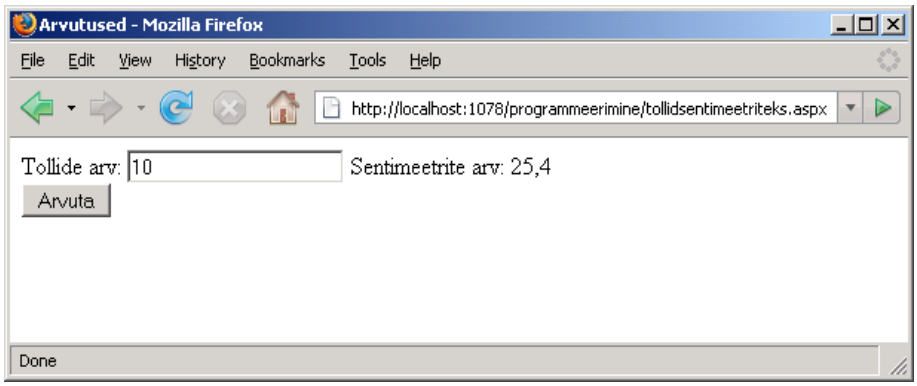
Kui teraselt algse versiooniga erisusi piiluda, siis paistab, et ülesse script-ossa on tekkinud alamprogramm nimega Button1_Click. Ning veebilehe sisuosas on nupu külge tekkinud juurde sündmuse kirjeldus ="Button1_Click" . Viimane tähendab, et kui nupule hiirega vajutatakse, siis käivitub märgitud nimega käsklus. Erinevalt mõnest muust programmeerimiskeelest ei ole siin kohustuslikku seost käskluse (funktsiooni) nime ning vajutatava nupu (või muu elemendi) nime vahel. Mõnikord võib panna ka mitmed elemendid sama käsklust välja kutsuma ning funktsiooni parameetrite (nime järel sulgudes olevate andmete) kaudu saab teada, millisele nupule tegelikult vajutati. Aga kuna praegu meil ainult üks nupp, siis pole mõtet lugu segaseks ajada.

Nupuvajutuse juurde siis programmikood, mis sammu kaupa teeb kõik vajalikud toimingud, et tähtedena (tekstina, stringina) sisestatud tollide arvust saaks vastusesse ka tähtedena sentimeetrite arv. Aga vahepealsete arvutuste juures on vaja andmetüüpi muuta, sest tekstidega ei saa C# keeles matemaatilisi tehteid ette võtta. Küll aga on arvude jaoks loodud omaette andmetüübid. Komadega hakkama saavaks arvutüübiks on double, ainult täisarve tunnistavaks tüübiks int. Levinumate andmetüüpide (nagu näiteks string ja int ja double) omavaheliseks tüübiteisenduseks sobivad klassi Convert käsklused. Rida

```
double TollidArvuna = Convert.ToDouble(TollidTekstina);
```

võtab tekstina olevad tollid, teisendab nad arvutimälus ümber reaalarvuna kirjutatavateks andmeteks ning jätab tulemuse meelde muutuja (märksõna) TollidArvuna alla. Viimase saab juba koefitsiendiga läbi korrutada, tulemuse tekstiks muundada ning vastusesildi peal näidata.

```
protected void Button1_Click(object sender, EventArgs e)
{
    string TollidTekstina = TextBox1.Text;
    double TollidArvuna = Convert.ToDouble(TollidTekstina);
    double koefitsient = 2.54;
    double SentimeetridArvuna = TollidArvuna * koefitsient;
    string SentimeetridTekstina = Convert.ToString(SentimeetridArvuna);
    Label1.Text = SentimeetridTekstina;
}
```



Kes Excelis või mujal valemeid kirjutanud, teab, et sarnased arvutused saab mõnikord ka lühemalt kirja panna. Siinsel puhul siis näiteks annaks sama tulemuse rida

```
Label1.Text=Convert.ToString(Convert.ToDouble(TextBox1.Text)*2.54);
```

Mõnikord on ühte rida isegi kergem lugeda kui pikemat programmilõiku. Kui aga kood nagunii nõnda pikk, et ühe pilguga ei haara, siis pärastise kontrolli ja arusaamise huvides on vahel kasulik toimetused sammude kaupa välja kirjutada.

Mõningast tähelepanu võivad vajada komad ja punktid. Nagu näha, programmikoodi sees on tollide ja sentimeetrite vaheline kordaja murdosa eraldaja kirjutatud punktina – 2.54. Veebilehitseja akna ekraanil aga vastus 25,4 kirjas komaga. C# keeles tuleb koodi sees eraldaja kirjutada alati punktiga – nii nagu keele sünnimaal tavaks. Kui aga arvutil seadeteks valitud Eestimaa, siis mõistetakse andmed sisse lugeda ja väljastada kasutades murdosa eraldajaks koma.

Tehtevalikuga kalkulaator

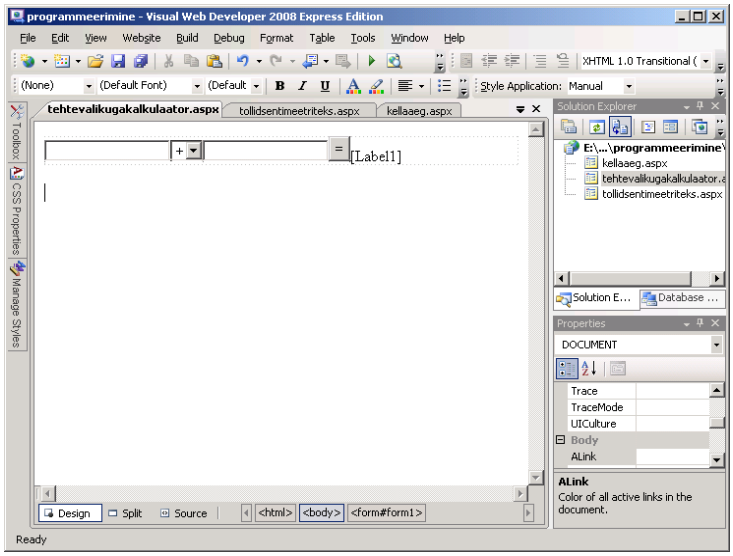
Eelnev kalkulaator suudab vaid tolle sentimeetriteks teha. Kui on tegemist ühe teisendusega, siis piisab taolisest alusest täiesti. Võib kindla kursi alusel arvutada kroone eurodeks või untse grammideks. Samuti kannatab mõningase mõtlemise peale eelneva näite põhjalt ehitada võimaluse, kus kahe või enama sisendandme põhjal vastus arvutatakse. Olgu näiteks sisendiks kauba kilode hind ja kilode arv ning väljundiks makstav summa. Lihtsalt tekstikaste tuleb rohkem ning igal neist peab olema oma, teistest erinev ID, mille kaudu vastavat väärtust küsida. Arvutamiseks tuleb andmetüüp doubleks muuta ning pärast näitamise tarvis jälle tagasi.

Kui aga ei soovita, et kõik tuleks kasutajal ise sisse kirjutada, vaid osa andmeid saaks ka mugavamalt ja veakindlamalt valida – siis aitab rippmenüü, DropDownList. Üksikud valikud selle sees saab ListItemite abil ära märkida. Ning programmikoodis saab rippmenüüst valitud väärtust kasutada sarnaselt kui igast muust sisestuselemendist tulnut.

Järgnev kood siis kõigepealt lehe kujunduseks, kus andmete sisestamiseks kaks tekstivälja, nende vahele rippmenüü. Ning teise tekstivälja järele nupp arvutuse alustamiseks ja nupu taha silt tulemuse näitamiseks.

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Tehtevalikuga kalkulaator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
     <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
     <asp:DropDownList ID="DropDownList1" runat="server">
       <asp:ListItem>+</asp:ListItem>
       <asp:ListItem>-</asp:ListItem>
       <asp:ListItem>x</asp:ListItem>
       <asp:ListItem>/</asp:ListItem>
    </asp:DropDownList>
     <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
     <asp:Button ID="Button1" runat="server" Text="="
onclick="Button1_Click" />
     <asp:Label ID="Label1" runat="server" text=""></asp:Label>
    </div>

    </form>
</body>
</html>
```

Nupu peal tekstiks võrdusmärk, sildi peal esiotsa tühjus. Et arendusvahendi disainivaates sildi olemasolu arusaadav oleks, selleks näidatakse seal kantsulgude vahel selle sildi nime. Veebilehel aga on vastaval kohal tühjus. Ning nupu külge siis taas seotud disainivaate topeltklõpsu kaudu onclick-sündmus, mille abil saab enesele vajaliku funktsiooni käima panna.

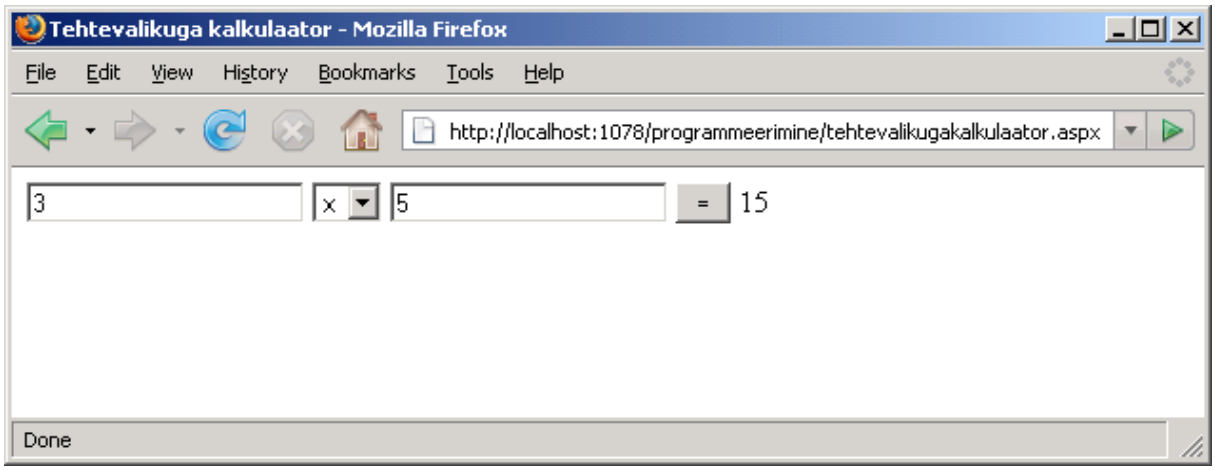


Funktsioonis nagu näha võetakse kõigepealt tekstikastidest välja väärtused ning teisendatakse need arvutamiseks sobivasse tüüpi. Millise tehte on inimene valinud, selle kindlakstegemiseks on üheks võimaluseks rippmenüü omandus SelectedIndex, mis annab valitud rea järjekorranumbri alates nullist. Iseenesest oleks võimalik ka rea väärtus SelectedValue abil kätte saada, aga järjekorranumber on kindlam. Näiteks rakenduse tõlkimise puhul võib nähtav jutt keelest sõltuda, kui aga järjekord samaks jätta, sama valiku peale saab ikka sama toimetuse teha.

Vastuse leidmiseks tuleb Vastuse muutuja kõigepealt deklareerida. Ning funktsiooni sees on kohustus sellele muutujale ka miski algväärtus anda. Ehkki me praegu teame, et tegemist on ühega neljast kindlaksmääratud valikust, tahetakse programmeerimiskeeles end kaitsta ka sellise võimaluse vastu, kus mõni valik on jäetud märkimata ning näitamiseks pole Vastusele kusagilt väärtust võtta. if-valikulausete abil tehakse sobiv tehe ning lõpuks näidatakse Vastus tekstiks (stringiks) muundatuna sildile Label1.

```
protected void Button1_Click(object sender, EventArgs e)
    {
        double Arv1 = Convert.ToDouble(TextBox1.Text);
        double Arv2 = Convert.ToDouble(TextBox2.Text);
        int Tehtenr = DropDownList1.SelectedIndex;
        double Vastus = -1;
        if (Tehtenr == 0) { Vastus = Arv1 + Arv2; }
        if (Tehtenr == 1) { Vastus = Arv1 - Arv2; }
        if (Tehtenr == 2) { Vastus = Arv1 * Arv2; }
        if (Tehtenr == 3) { Vastus = Arv1 / Arv2; }
        Label1.Text = Convert.ToString(Vastus);
    }
```

Edasi võibki juba imetleda, kuidas me armas väike kalkulaator töötab.



http://video.msn.com/video.aspx?vid=166cb365-9733-4b8b-ac0e-a2089e3e6908

Ülesandeid

- Võta aluseks tolle sentimeetriteks teisendav näide ning koosta selle põhjal leht, kus arvutatakse tunde ümber minutiteks.

- Paiguta lehele kaks nuppu: ühe abil saab arvutada tunde minutiteks, teise kaudu minuteid tundideks.

- Vali rippmenüüst kauba nimetus, kõrvale tekstivälja kirjuta kogus. Tulemusena väljasta makstav summa.

- Selliseid rippmenüü ja koguse komplekte on lehel mitu. Rippmenüüs on esimeseks valikuks „kaup valimata“ väärtusega null krooni. Leia kokku vajalik makstav summa.

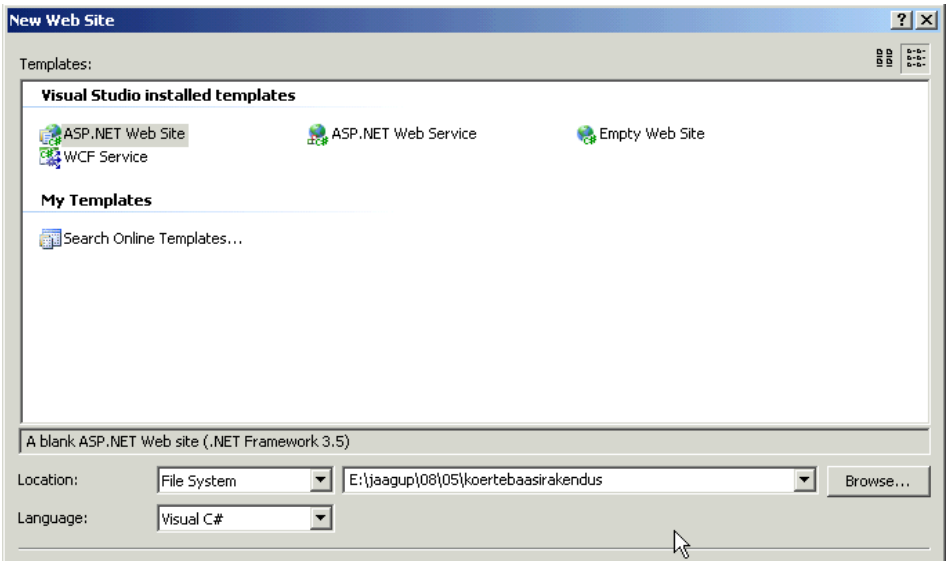
- Tekstiväljadesse kirjutatakse auto kütusekulu liitrites saja kilomeetri kohta ning läbitavate kilomeetrite arv. Rippmenüüst valitakse bensiini mark millele vastavalt on programmiskoodis kirjas selle bensiini liitrihind. Arvutuse tulemusena väljastatakse sõidu maksumus.

Andmebaasipõhise veebirakenduse loomine

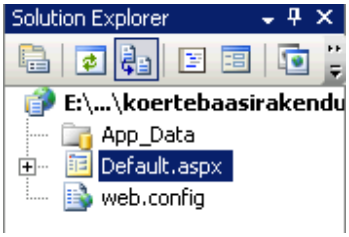
Eelnevalt näidatud kalkulaatoreid saab luua ka ilma veebipoolse programmeerimiseta – olgu siis Javaskripti, Flashi, Silverlighti või mõne muu kliendipoolse tehnoloogia abil. Kui aga soov otsida suurest andmehulgast serveris või sinna omapoolseid lisandusi teha – sellisel juhul on veebiserveripoolne programmeerimine vajalikum. Ning ASP.NETi juures on päris mitmekülgsed ja mugavad vahendid andmetabelitega ümberkäimiseks loodud.

Andmetabeli näitamine veebilehel

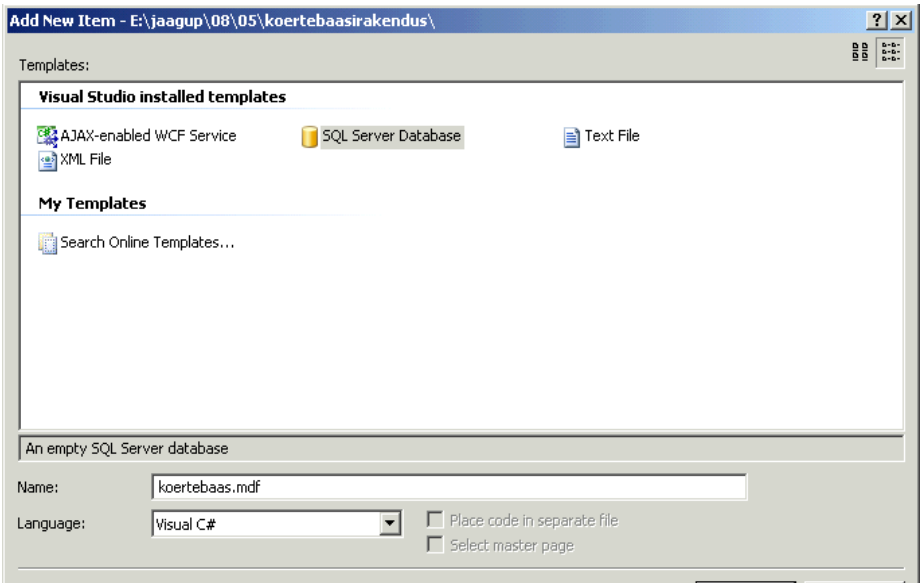
Alustame lihtsamast, ehk siis serveris tabelis olevate andmete vaatamisest Interneti kaudu. Nagu ikka, tuleb algul veebirakendus luua. Siiani olime lühiduse huvides valinud „Empty Web Site“. Kui soovida, et arenduskeskkond meile mõned vajalikud failid algul valmis genereeriks, siis sedakorda valime „ASP.NET Web Site“.



Võrreldes eelmise variandiga on juures Default.aspx-fail avalehe jaoks, App_Data kaust andmete salvestamiseks ning konfiguratsioonifail web.config.

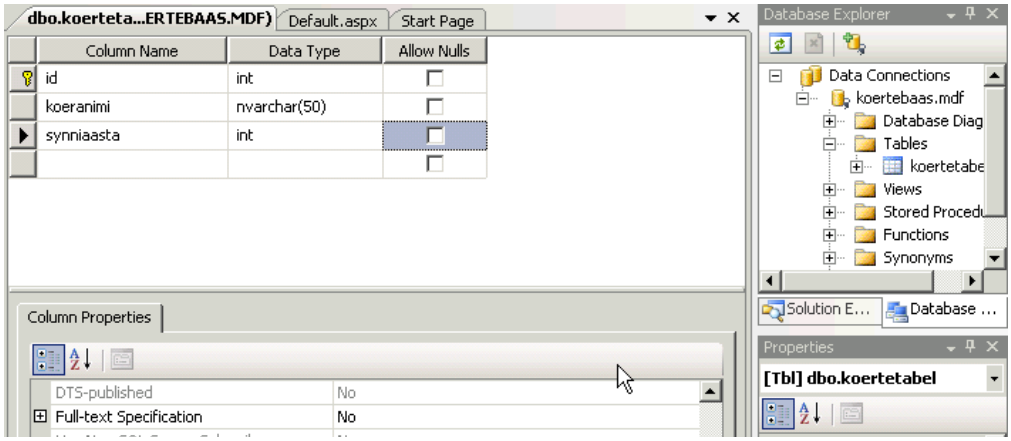


Et andmeid kusagil hoida oleks, selleks loome App_Data-nimelisse kausta sisse SQL Serveri andmebaasi. Nagu valikutest näha, oleksid teisteks võimalusteks tekstifail ning XML-fail. Siin näites andmebaasifailile nimeks koertebaas.mdf.

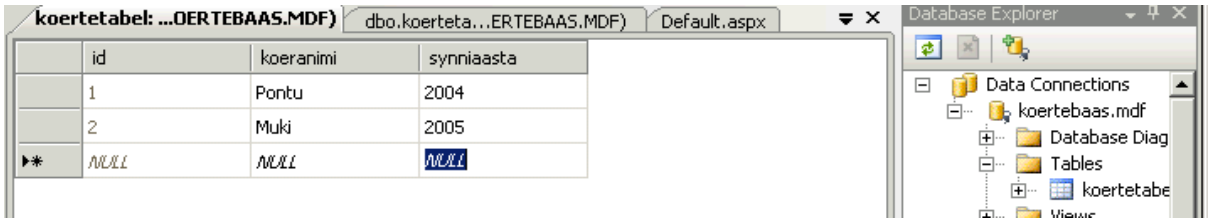


Esimeses ja lihtsamas lähenduses on andmebaas andmetabelite komplekt, igas tabelis olemas read ja veerud. Kõigepealt luuakse veerud, märkides neile ära nime ja tüübi. Siinses võimalikult lihtsas näites püüame veebilehel näidata koeri, igaühel koeranimi ja sünniaasta. Lisaks on andmebaasitabelite juures poolkohustuslikuks tulbaks identifikaatortulp, näiteks nimega id. Miks see vajalik? Kes näiteks Exceli tabeleid loob, sel tulevad reanumbrid automaatselt. Saab alati selgelt öelda, et vastav tekst on millises reas ja millises veerus. Andmebaasitabelites aga sellisel kujul reanumbreid ei hoita. Isiku identifitseerijaks võib olla näiteks isikukood. Need aga ei lähe sugugi mitte järjest. Küll aga on andmetabelite juures igati kasulik, kui on võimalik mingi kindla tunnuse järgi rida eristada. Muidu võib kergesti juhtuda, et näiteks sisestamisel tekib kaks korduvat rida ning tavavahenditega ei saagi naljalt öelda, et kustuta üks rida ära ja jäta teine alles. Siin siis eristajaks tulp nimega „id“. Nime ette tasub hiire parema klahviga valida primaarvõtmemärk. Et andmebaasiprogramm taipaks automaatselt id-numbreid välja mõelda, selleks tasub alt veeru omaduste seast valida, et tulba omadus „identity“ on sisse lülitatud. Ning siis asutakse automaatselt alates ühest ühe kaupa reanumbreid panema.

Andmetüüp int tähendab täisarve, sobib nii id kui sünniaasta puhul. Koera nimeks sai nvarchar(50) ehk siis national (lubab täpitähti) varchar ehk muutuva pikkusega, kuni 50 sümboli pikkune tekst. Omadus allow Nulls kipub vahel esmakordsel tutvusel segadust tekitama. Kõige lihtsam on sealne kast võimalusel märkimata jätta. Sel juhul tühiväärtused, ehk nullid (sõnaga) ei ole lubatud. Kui sisestame koera, peame sisestama ka tema sünniaasta. Kui aastat ei tea, saame aasta väärtuseks panna ka näiteks 0 või -1. Aga ei saa aastat panemata jätta.

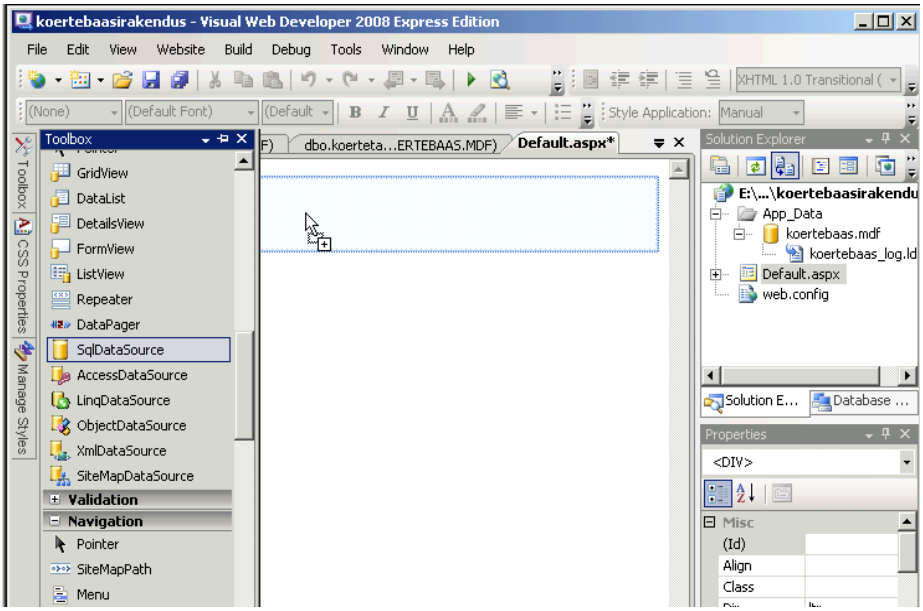


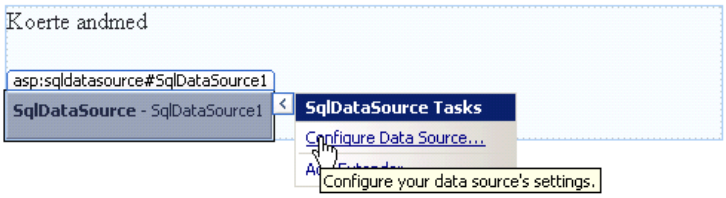
Edasi juba koerte eneste sisestamine. Selleks tuleb avada Database Explorer. Sealt valida tabelite alt koertetabel ning selle pealt parema hiireklahviga Show Table Data. Nimi ja sünniaasta tuleb kirjutada, id-number tuleb automaatselt juhul, kui Identity Specificationi juurde on ennist yes kirjutatud.



Nende toimetuste tulemusena on andmebaas ja andmetabel olemas. Edasi siis juba tasub vaadata, kuidas andmed veebilehele nähtavaks saab.

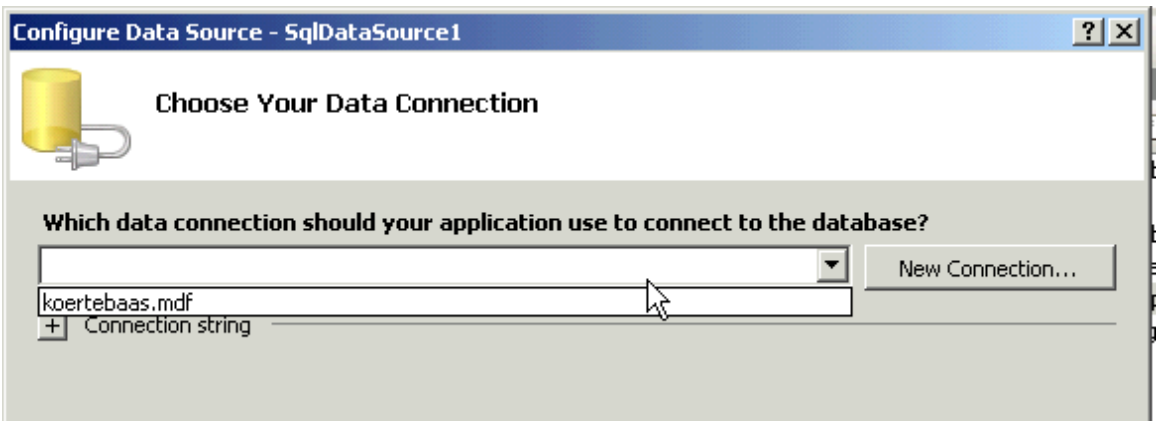
Andmete vahendamiseks on ASP.NETis andmeallikad, nende nime lõpus on DataSource. SQL Serveri jaoks sobib neist nagu nimigi ütleb SQL Data Source. Alustuseks tuleb Toolboxist andmeallikas oma lehele vedada. Tulemusena ilmub ta Visual Web Developeri lehele omaette elemendina. Veebilehe tarvis on aga tegemist nähtamatu elemendiga, st. lehitsejas avades andmeallikat eraldi näha pole.



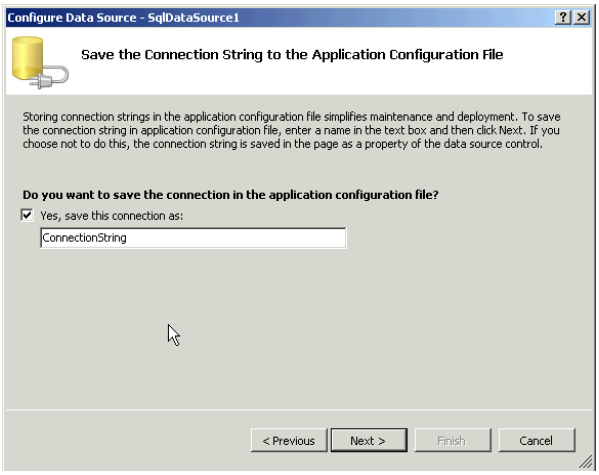


Graafilise osa juurest saab allikat ka edasi konfigureerida. Allika peale hiirega minnes ilmub nähtavale väike kolmnurk. Sealt tasub valida „Configure Data Source“.

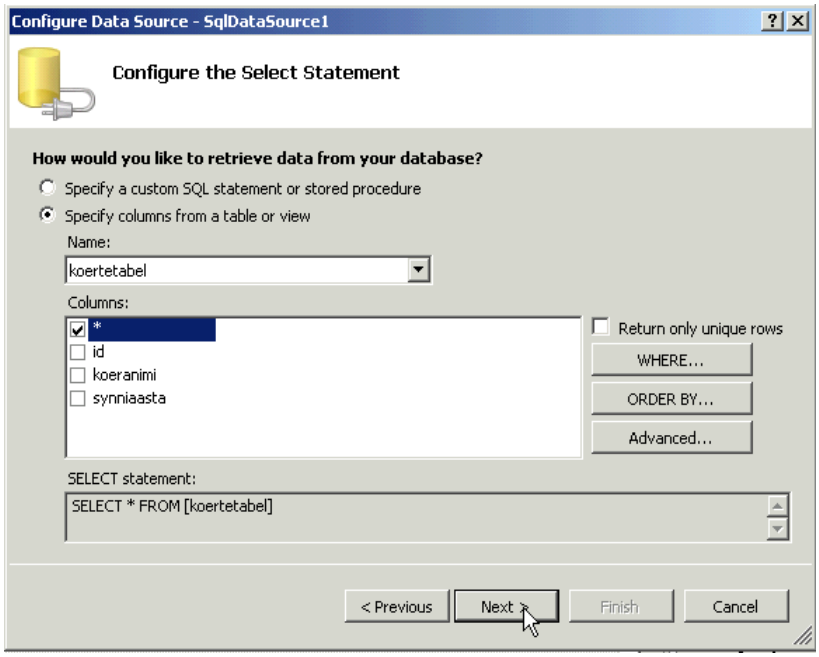
Praegu ainukesed andmed ongi App_Data kaustas olevas andmebaasifailis koertebaas.mdf. Selle valimegi.



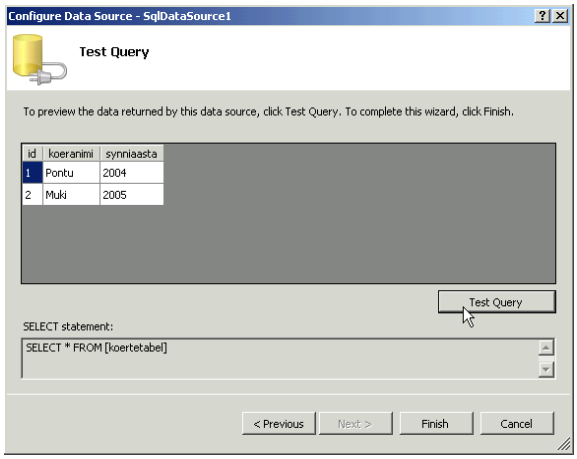
Edasi pakutakse andmeühenduse seadistuste salvestuse võimalust. Et oleks võimalik edasi rakendust hiirevalikute abil koostada, tuleb nad salvestada. Samuti on võimalik nõnda sama ühendust tulevikus mujal kasutada ilma, et peaks taas valima ja sättima hakkama. Siin näites on tegemist ühe failiga rakenduse kataloogis ning seetõttu ligipääs lihtne.



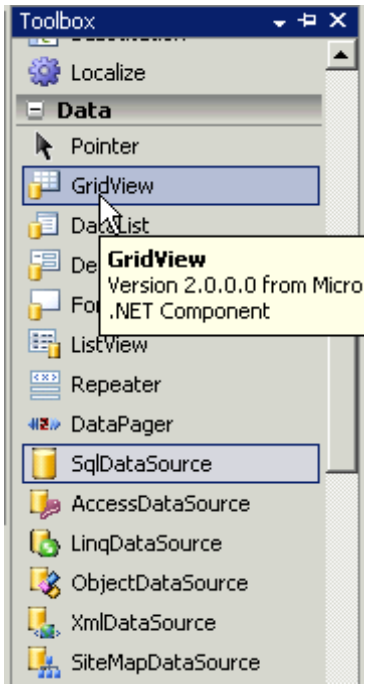
Mõnikord võib aga andmetele ligipääsuks olla vaja läbida õiguste ja võrgupõhine rägastik. Sel juhul ühenduse andmete korduvkasutus eriti vajalik. Samuti annab ühenduse andmete salvestamine võimaluse hiljem muuta andmebaasi asukohta ilma, et peaks selleks kõiki rakenduses leiduvaid andmebaasiga suhtlevaid lehti eraldi ümber tegema. Nimeks pakutakse praegu lihtsalt ConnectionString. No las ta olla – ehkki keerulisemate rakenduste puhul on vahel kasulik nimi määrata selle järgi, kuhu parajasti ühenduda tahetakse.



Järgmise sammuna uuritakse, et milliseid andmeid soovitakse vaadata. Et baasis on esialgu ainult üks tabel, pole valida kuigi paljude kohtade vahel. Ning kui tulbanimede alt valida tärn, siis tähendab see, et soovime näha kõikide tulpade andmeid.

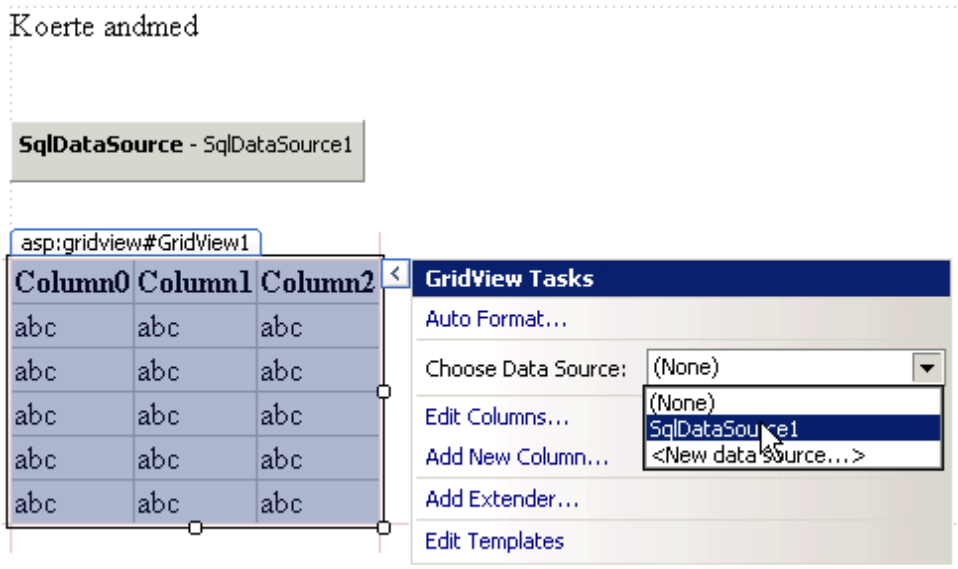


Enne päringu salvestamist on soovitav seda katsetada. Kuna küsisime kõiki koeri kõikide andmetega ja saime kõik koerad kõikide andmetega, siis võib andmeallika loomise õnnestunuks pidada.

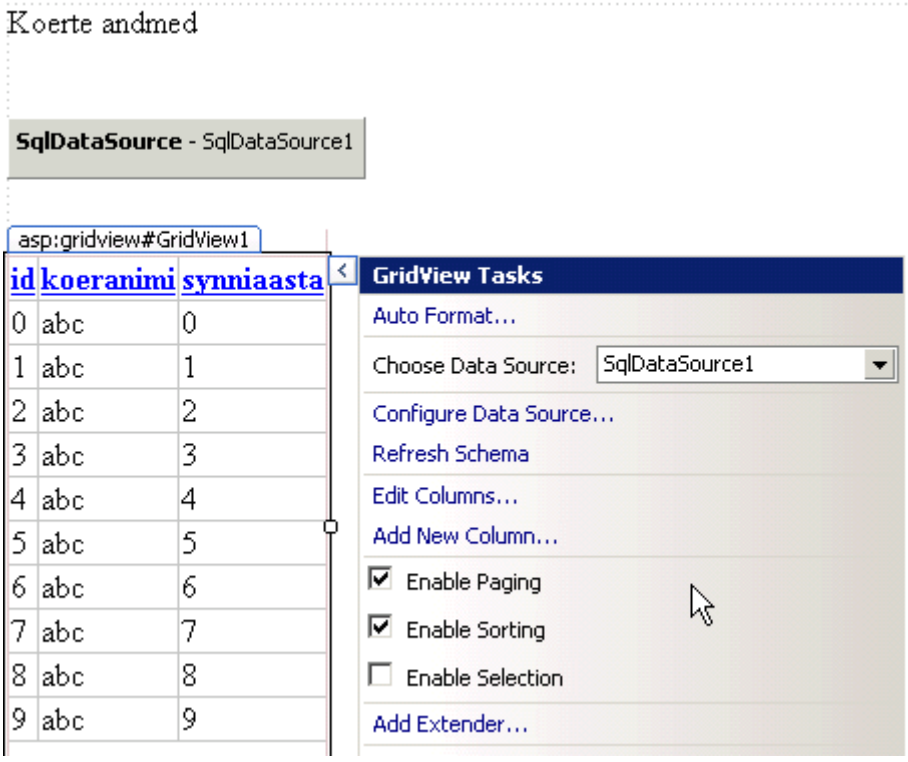


Kui andmeallikas olemas, siis järgmiseks on näitamisvahendi valik. Tabeli kujul olevad andmed saab lehele kõige lihtsamalt GridView Abil. Seal näidataksegi andmeid ridade ja veergudena.

Ka GridView on graafilise liidese abil seadistatav. Kõigepealt taas hiirega sinna peale liikumisel tekkinud väikesest kolmnurgast avada menüü ning sealt valida vajalik andmeallikas. Et meil siin lehel ainult üks allikas on, siis tuleb valida seesama SqlDataSource1.



Sealtkaudu saadakse juba aru, et tulpadeks on id, koeranimi ja sünniaasta. Kannatab lisada mõne „linnukese“. Enable Paging ütleb, et andmeid näidatakse lehekülgede kaupa. Ehk kui koeri on väga palju, siis nad ei ilmu mitte kõik korraga ekraanile, vaid näiteks kümne või mõne muu eraldi märgitava arvu kaupa. Enable Sorting järjestab andmed selle tulba järgi, millise pealkirjale kasutaja vajutas.



Ja polegi muud. Leht veebilehitsejas lahti ning võib koeri imetleda.

Lähtekood

Ehkki koerte loetelu sai suuremalt jaolt „hiire abil“ kokku lohistatud, on masinate jaoks kandev osa ikkagi tekkinud tekstil (mis sellest, et märgatav osa tekstist genereeritud). Kes tahab lihtsalt kiiresti standardlehekülgi kokku panna ja mitte viimistlemisega vaeva näha, võib siinsest lõigust esialgu üle hüpata. Pärastiseks süvenemiseks aga siia mõned seletused, et „mis siis tegelikult juhtus“.

Lehe ASP.NETi päis genereeriti eelmisest mõnevõrra pikem.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

Keelevalik (C#) nii nagu varasemateski lehtedes. AutoEventWireup näitab, et on võimalik mõningad toimetused automaatselt lehe avamise juurde kirjutada (mida siin praegu tegelikult ei kasutata). Koodifail määrab failinime, kuhu võimalik lehel toimuvat määrav programmikood kirjutada. Eelnevas kalkulaatorinäites kirjutati arvutusvalemid otse aspx-lehele. Kui aga programmiosa suurem, siis on viisakas see eraldi faili paigutada – kujundajad enamasti ei oska sellega midagi mõistlikku peale hakata, programmeerijat jälle häirivad liigsed HTML-i osad. Siin koertetutvustuse juures küll veel midagi programmikoodiga ei tehta, aga standardseadetes „ASP.NET Web Site“ annab just sellise mooduse, kus aspx-fail kujunduseks ning sama nimealgusega cs-fail võimalike arvutuste tarbeks. Et ei tekiks kiusatust ebaviisakat programmeerijat mängida ja aspx-lehele programmikoodi kirjutada. Inherits="_Default" läheb samasse valdkonda – seal kirjas cs-failis oleva koodiklassi nimi, kust seest lehe juurde kuuluvaid käsklusi loetakse.

Edasi HTML-kood nagu tavaliselt. Midagi põnevamat hakkab juhtuma alles andmetoimetuste juurde jõudes.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    SelectCommand="SELECT * FROM [koertetabel]">
</asp:SqlDataSource>
```

Andmeallikas, ehk lehel siis koodina kirja pandud element nimega asp:SqlDataSource. ID pandi talle automaatselt. Ühendusteksti ehk ConnectionStringi saamiseks oli ennist pikem toimetus – tuli määrata, millisest kohast või failist andmed leitakse. Ning otsimiskohaks paistab olema ConnectionStrings:ConnectionString. Tekst ise tegelikult paikneb rakenduse põhikaustas olevas web.config failis. Kui seal veidi otsida, leab sealt jaotuse connectionStrings ning selle alt praeguseks ainukese teksti nimega ConnectionString. Tema tegelik väärtus on siinsel juhul

```
"Data  Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\koertebaas.mdf;
Integrated Security=True;User Instance=True"
```

Ehk siis kohaliku masina andmebaasiühendus nimega SQLEXPRESS, andmekataloogis (App_Data) asuv andmebaasifail koertebaas.mdf. Ning ühendumiseks kasutatakse Windowsi õiguste süsteemi. Ehk siis kui veebiserver jookseb administraatoriõigustes, siis pääseb ta vabalt ka andmebaasile ligi.

SQL-lause `SELECT * FROM [koertetabel]` ütleb, et küsigu välja kõik andmed koertetabelist. Kandilised sulud tabeli nime ümber pole kohustuslikud. Aga kuna need aitavad mõnikord erisümbolitega paremini hakkama saada, siis generaator paneb need sulud ümber. Et SQL ise on levinud ja küllaltki võimalusterohkem keel, siis rakendusi edasi arendades saab siia panna tunduvalt keerukamaid ja vajalikumaid päringuid kui esialgse hiirega kokkulohistamise teel.

Ning lõpetuseks andmete näitamise GridView

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
    AllowSorting="True" AutoGenerateColumns="False"
    DataKeyNames="id" DataSourceID="SqlDataSource1">
</asp:GridView>
```

AllowPaging lubab andmeid reagruppide kaupa eraldi lehtedel näidata. AllowSorting vastava tulba järgi reastada. Kui AutoGenerateColums oleks True, siis igal lehe avamisel vaadatakse uuesti, et millised tulbad päringust tulevad. DataKeyNames ehk võtmeväljade nimi/nimed on tarvilikud alles siis, kui andmeid muutma/kustutama hakatakse. Aga generaator paneb nad juba nüüd suure hooga kohale. Ning viimasena GridView jaoks ehk kõige tähtsam teave, et ehk kust need näidatavad andmed võetakse. Praegusel juhul siis andmeallikast IDga SqlDataSource1.

Et tulpi automaatselt genereerida ei lubatud, siis tuleb andmete nägemiseks ise kirja panna, et milliste tulpadega on tegemist.

```
<Columns>
  <asp:BoundField DataField="id" HeaderText="id" InsertVisible="False"
      ReadOnly="True" SortExpression="id" />
  <asp:BoundField DataField="koeranimi" HeaderText="koeranimi"
      SortExpression="koeranimi" />
  <asp:BoundField DataField="synniaasta" HeaderText="synniaasta"
      SortExpression="synniaasta" />
</Columns>
```

Hiljem saab vajadusel siitkaudu ka tulpade kujundusi määrata. Nagu nimedest aimata võib, siis DataField näitab, et millise andmebaasi tulbaga seotud ollakse. HeaderText määrab veebilehel tabelis nähtava pealkirja. Ning SortExpressioni järgi reastatakse vastav tulp. Et siin kõik kokku langevad, see vaid praeguse lihtsuse küsimus. Tõlgitavas rakenduses võiks päise tekst vabalt muutuma hakata. Ning järjestamiseks ei pruugi alati sugugi olla kasvav järjekord, vaid võib näiteks olla hoopis isikukoodist välja võetud sünnikuude järjekord.

Nüüd aga lehe kood tervikuna.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!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>Koerte leht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        Koerte andmed<br />
        <br />
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT * FROM [koertetabel]">
        </asp:SqlDataSource>
        <br />
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
            AllowSorting="True" AutoGenerateColumns="False"
            DataKeyNames="id" DataSourceID="SqlDataSource1">
            <Columns>
               <asp:BoundField DataField="id" HeaderText="id"
               InsertVisible="False" ReadOnly="True" SortExpression="id" />
               <asp:BoundField DataField="koeranimi" HeaderText="koeranimi"
                    SortExpression="koeranimi" />
               <asp:BoundField DataField="synniaasta"
                HeaderText="synniaasta" SortExpression="synniaasta" />
            </Columns>
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

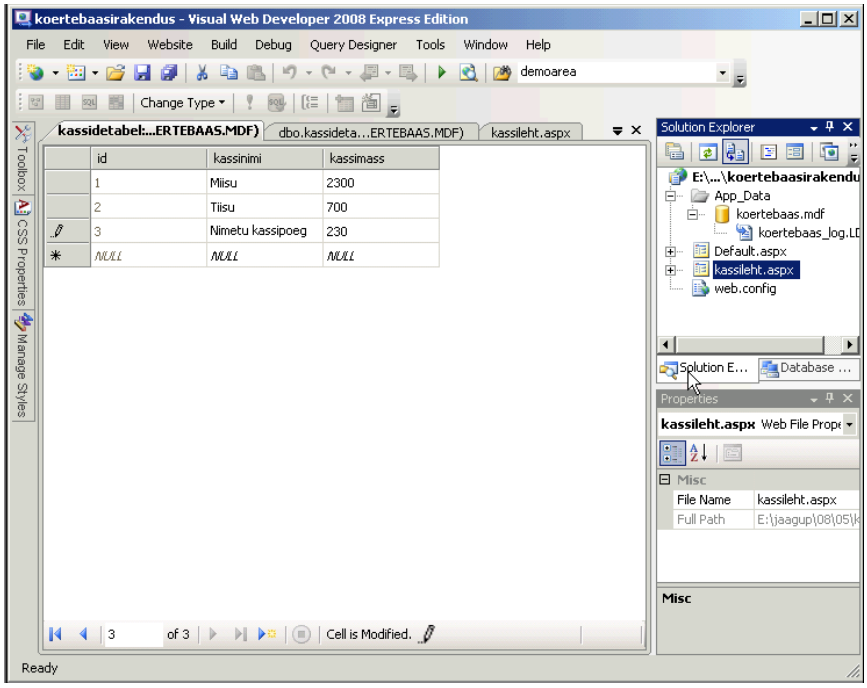
http://video.msn.com/video.aspx?vid=89746fe3-ab23-41d3-9963-df0fac04df52

Ülesandeid

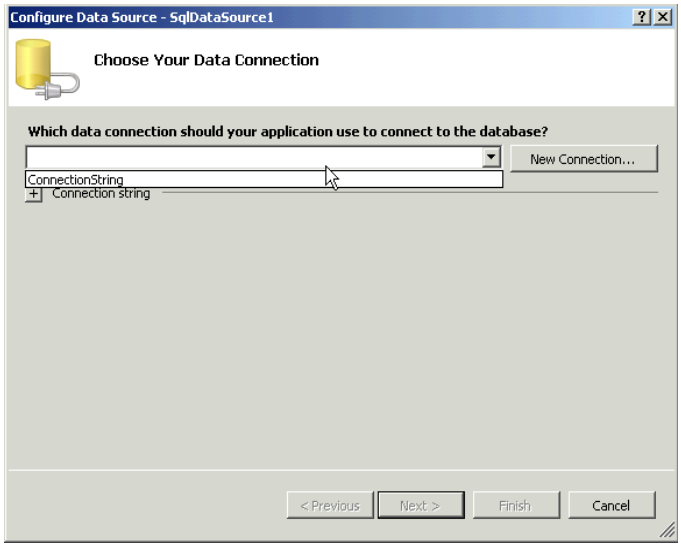
- Tee näide sarnaselt läbi, ainult, et koerte asemel võta inimeste nimed, pikkused ja massid. Veendu, et sortida saab iga tulba järgi.

- Muuda lehe SQL-lauset nõnda, et näha oleksid vaid 170st sentimeetrist lühemad inimesed (SELECT * FROM inimesed WHERE pikkus < 170)

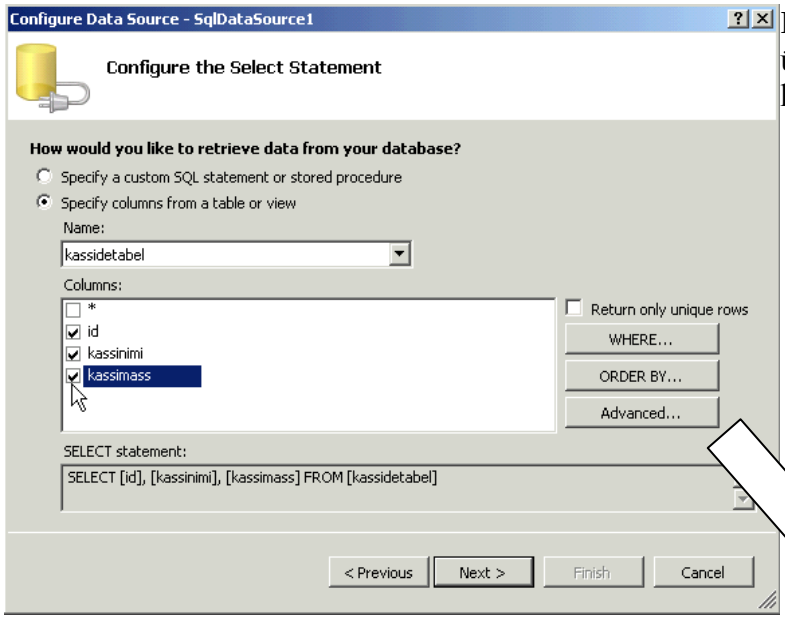
Andmete muutmine ja lisamine veebilehelt



Et töötav koerteleht jääks ilusti hilisema tutvumise jaoks alles, teeme uue sarnase andmevaatamismooduse kasside tarbeks ning püüame seda hiljem mõnevõrra võimalusterohkemaks teha. Alustuseks ikka aspx-leht ning omaette andmetabel

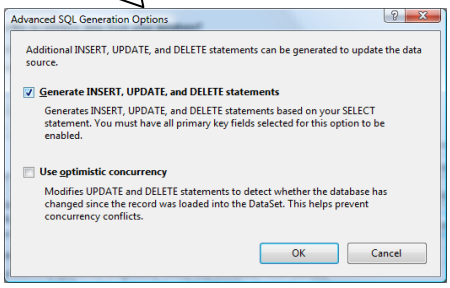


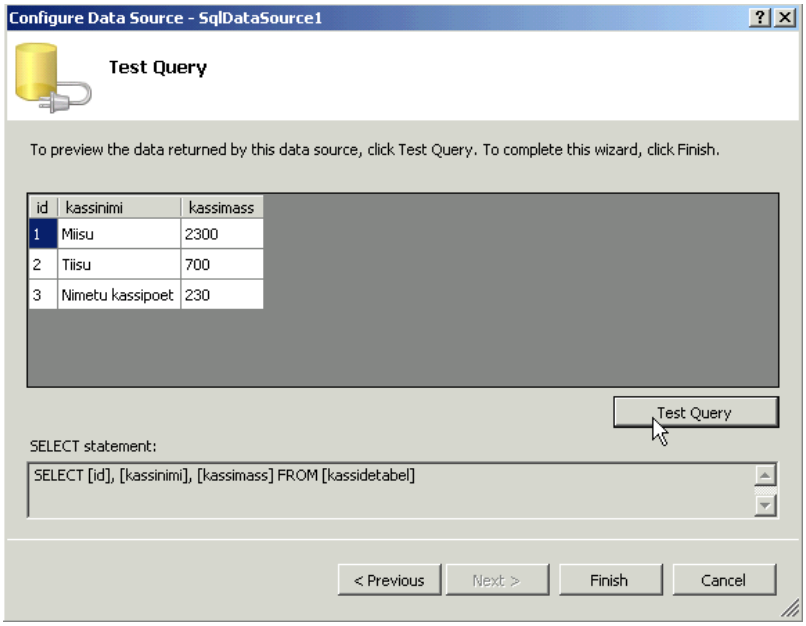
Kuna kasutame sama andmebaasi, saab andmeallika juures kasutada sama ConnectionStringi



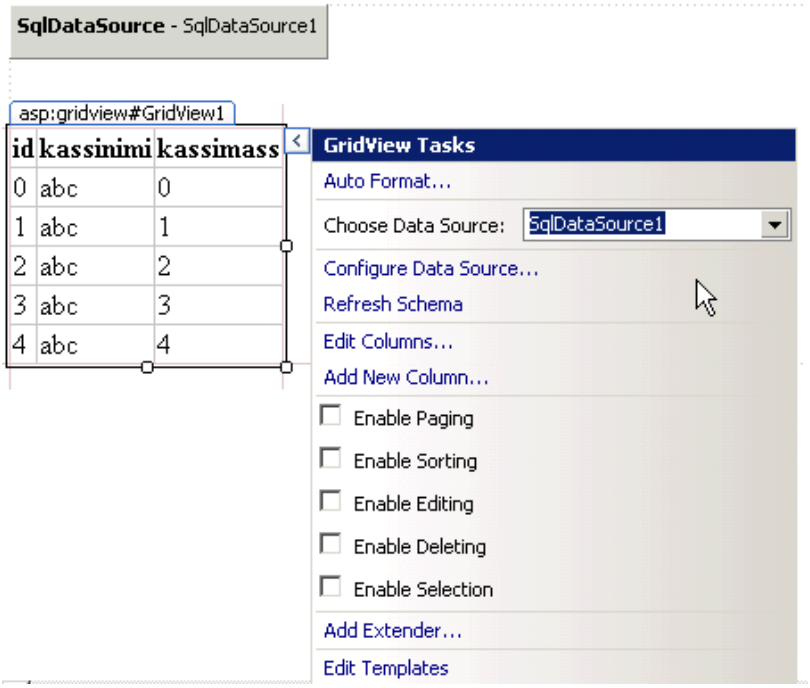
Et SQL-lause oleks ülevaatlikum, selleks märgin kõik tulbad eraldi.

Lisaks Advanced-nupu alt linnuke sooviga, et genereeritaks lisaks SELECT´ile ka INSERT, UPDATE ja DELETE-laused, ehk siis käsud, mille abil andmeid lisada, muuta ja kustutada. Neid lauseid lubatakse genereerida vaid juhul, kui andmebaasitabelis on primaarvõti määratud.





Jällegi väike test veendumaks, et andmed liiguvad



Ning edasi lehele juba GridView

Kuna andmeallikas rohkem SQL lauseid genereeritud, siis pakutakse ka rohkem võimalusi linnukeste panekuks.

	id	kassinimi	kassimass
Edit Delete	1	Miisu	2300
Edit Delete	2	Tiisu	700
Edit Delete	3	Nimetu kassipoet	230

Kui muutmis- ja kustutusõigus märgitud, siis võib lehel asuda vastavaid toiminguid tegema.

	id	kassinimi	kassimass
Edit Delete	1	Miisu	2300
Edit Delete	2	Tiisu	700
Update Cancel	3	Nimetu kassipoet	230

Valides muutmise, saab olemasoleva teksti asemele sobilikuma kirjutada ning vajutada, et nüüd on paras aeg muutused salvestada.

	id	kassinimi	kassimass
Edit Delete	1	Miisu	2300
Edit Delete	2	Tiisu	700
Edit Delete	3	Nimetu kassipoeg	230

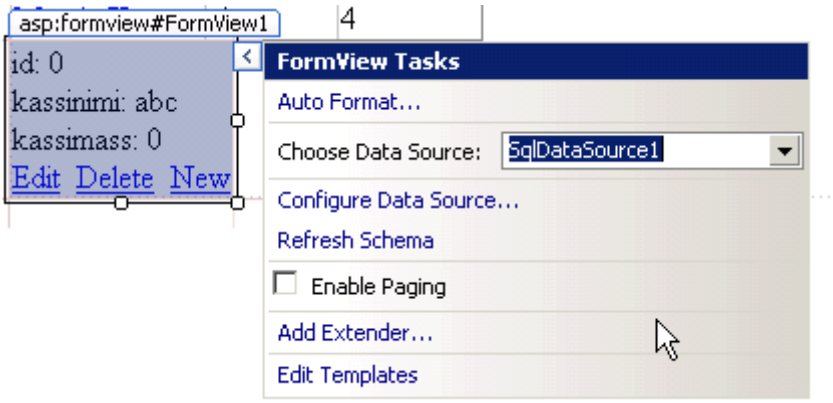
ning veenduda, et nad muudetuna ka veebilehel nähtavad on.

Soovides rakendust maakeelseks muuta, tasub koodist GridView seest üles otsida CommandField ning määrata tekstidele sobiv tõlge.

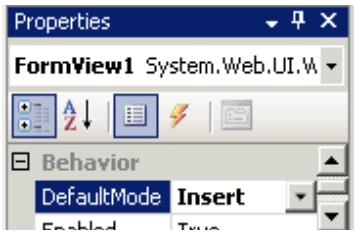
```
<asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
  DeleteText="Kustuta" EditText="Muuda"
  UpdateText="Salvesta muutused" CancelText="Katkesta"/>
```

Andmete lisamine

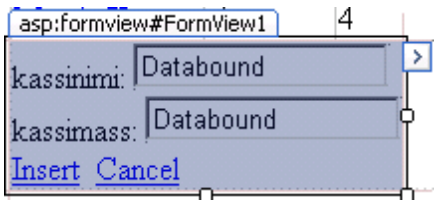
Andmete lisamiseks lihtsaimad abilised on FormView ja DetailsView.



FormView tuleb sarnaselt eelmisele Toolboxist lehe peale sikutada. Kuna andsime oma andmeallikale ka andmete lisamise oskuse, siis saame kasutada sama andmeallikat.



Omaduste (Properties) alt tasub valida avanemiskujuks (DefaultMode) Insert



ning selle peale muudetakse komponendi kuju juba sisestuse jaoks sobivaks.

	kassinimi	kassimass
Muuda Kustuta	Miisu	2300
Muuda Kustuta	Nimetu armas kassipoeg	235

kassinimi: Liisu
kassimass: 2000
Insert Cancel

Avades lehekülje veebilehitsejas, saab tekkinud vormi kaudu uusi andmeid sisse kirjutada

	kassinimi	kassimass
Muuda Kustuta	Miisu	2300
Muuda Kustuta	Nimetu armas kassipoeg	235
Muuda Kustuta	Liisu	2000

kassinimi:
kassimass:
Insert Cancel

ning rõõmustada selle üle, et need ka salvestuvad.

	kassinimi	kassimass
Muuda Kustuta	Miisu	2300
Muuda Kustuta	Nimetu armas kassipoeg	235
Muuda Kustuta	Liisu	2000

kassinimi:
kassimass:
Lisa Katkesta

Taas mõningased täiendused lähtekoodis asendades inglisekeelseid sõnu eestikeelsetega ning juba näebki leht ilusam välja.

Et aga FormView satub generaator põhjalikumalt lahti kirjutama kui GridView, siis tuleb muutusi teha ka rohkem kui ühes kohas. Sellest aga varsti allpool.

Lähtekood

Mitmelgi moel on siinne leht koeralehega sarnane. Märkimist väärivad aga lisandused/muutused.

Kuna andmeallika loomisel palusime, et loodaks laused ka muutmise ja kustutamise tarbeks, siis on nad nüüd olemas. SELECT-lause enamvähem sarnane nagu enne. Juurde on tulnud aga kustutamise (DELETE), lisamise (INSERT) ja muutmise (UPDATE) laused. Tasub neile lausetele lähemalt peale vaadata, sest need neli SQL-lauset on vajalikud enamiku andmebaasiga seotud rakenduste juures. Lausetest leiab salapäraseid @-märgiga sõnu. Näiteks kustutamise tarbeks loodud lause näeb välja

```
DELETE FROM [kassidetabel] WHERE [id] = @id.
```

Too tagumine @id on parameeter, mis tuleb kusagilt rakenduse seest saada. Altpoolt piiludes näeb, et selline parameeter on eraldi välja toodud.

```
            <DeleteParameters>
                <asp:Parameter Name="id" Type="Int32" />
            </DeleteParameters>
```

ja määratud, et tüübiks on Int32 ehk siis täisarv. Samuti leiab selle GridView juurest. Selle järgi teatakse, et kui kasutaja vajutab kustutusviidet, siis saadetakse andmebaasile käsklus, et selle id-ga rida tuleb kassidetabelist ära kustutada.

Sarnaselt määratakse parameetrid ka teiste lausete puhul.

Lisamiseks

```
INSERT INTO [kassidetabel] ([kassinimi], [kassimass])
  VALUES (@kassinimi, @kassimass)
```

ehk siis kõigepealt tabeli nimi kuhu lisatakse, seejärel sulgudes tulpade nimed kuhu lisatakse ja pärast sõna VALUES need väärtused, mis sinna tegelikult lähevad. Et neil on @-märgid ees, siis saab tegelikud väärtused taas töö käigus ehk parameetritena sisse anda.

Muutmiseks

```
UPDATE [kassidetabel] SET [kassinimi] = @kassinimi,
   [kassimass] = @kassimass WHERE [id] = @id
```

määratakse kõigepealt tabeli nimi. Edasi hakatakse ükshaaval määrama, milline tulp millise uue väärtuse saab. Nagu INSERT-lause puhul, nii ka siin ei anta id-le uut väärtust. Lisamises id puudub sootuks, see pannakse automaatselt. Muutmise juures näitab id, millist tabeli rida muuta.

Siin rakenduses käib kogu suhtlus GridView kaudu, aga keerukamal juhul võivad andmed tulla hoopis mitmest sisestusväljast, teiselt lehelt või näiteks programmikoodi kaudu.

FormView näeb lähtekoodis mõnikord välja üllatavalt pikk: kogu kujundus on pikalt välja kirjutatud ning seda annab hiljem soovi korral märgatavalt sättida. Kui pikemalt peale vaadata, siis paistab, et koodi saab eraldada mitmeks üsna iseseisvaks osaks.

```
        <asp:FormView ID="FormView1" runat="server" DataKeyNames="id"
            DataSourceID="SqlDataSource1" DefaultMode="Insert">
            <EditItemTemplate>
            </EditItemTemplate>
```

```
        <InsertItemTemplate>
        </InsertItemTemplate>
        <ItemTemplate>
        </ItemTemplate>
    </asp:FormView>
```

Ehk siis FormView sees on eraldi mallid ehk Templated andmete muutmiseks, lisamiseks ja kustutamiseks. Kuna me praeguses rakenduses kasutame FormViewd ainult andmete lisamiseks, siis tegelikult teised on kasutud. Aga generaator genereeris nad ikka igaks juhuks kõik – mine tea mis programmeerijal pärast võib pähe tulla. Ja olemasoleva koodi vahele millegi automaatselt juurde panek võib juba hoopis keerukaks osutuda.

Lisamislõik on peaaegu tavaline viisakas HTML. Vähemalt sellena ta lõpuks veebilehitsejasse näidatakse. Kui eelnevat lehepilti siinse koodiga võrrelda, siis paistab, et kõik on olemas. Teade kassinimi koos kooloniga on ilusti tekstina nähtav. Element asp:TextBox genereeritakse HTMLi vastavaks elemendiks. Sõna Bind tekstikasti omaduse Text sees näitab, et selle tekstivälja (mille nimeks on kassinimiTextBox) tekst seotakse andmeallika (ning sealtkaudu ühtlasi ka andmetabeli) tulbaga, mille nimeks kassinimi. Praegu kasutatakse seda seost andmete lisamisel. Hiljem aga muutmise puhul seesama Bind aitab väärtusi andmetabelist veebilehele ja tagasi liigutada.

 nagu tähendab HTMLis reavahetust veebilehel (tavaline reavahetus koodis seda ei põhjusta). Märgend   (non-breaking space) paneb sobivasse kohta tühiku.

Lõppu siis lisamise ja lisamiskatkestuse nupud. Element asp:LinkButton näeb välja nagu viide, aga tema külge saab programmitoiminguid siduda. Siinsel juhul siis on need toimetused peidus FormView sees. Kuna viimane on seotud SqlDataSource1-ga, siis lisamisnupu peale võetakse andmed tekstiväljadest, saadetakse andmeallika parameetritesse ning sealt edasi INSERT-lause abil andmebaasi.

```
<InsertItemTemplate>
    kassinimi:
    <asp:TextBox ID="kassinimiTextBox" runat="server"
        Text='<%# Bind("kassinimi") %>' />
    <br />
    kassimass:
    <asp:TextBox ID="kassimassTextBox" runat="server"
        Text='<%# Bind("kassimass") %>' />
    <br />
    <asp:LinkButton ID="InsertButton"
        runat="server" CausesValidation="True"
        CommandName="Insert" Text="Lisa" />
     
    <asp:LinkButton ID="InsertCancelButton" runat="server"
        CausesValidation="False"
        CommandName="Cancel" Text="Katkesta" />
</InsertItemTemplate>
```

Ning lõpetuseks lähtekood tervikuna.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="kassileht.aspx.cs"
Inherits="kassileht" %>
<!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>Kasside leht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            DeleteCommand="DELETE FROM [kassidetabel] WHERE [id] = @id"
            InsertCommand="INSERT INTO [kassidetabel]
              ([kassinimi], [kassimass]) VALUES (@kassinimi, @kassimass)"
            SelectCommand="SELECT [id], [kassinimi], [kassimass]
               FROM [kassidetabel]"
            UpdateCommand="UPDATE [kassidetabel]
                SET [kassinimi] = @kassinimi, [kassimass] = @kassimass
                WHERE [id] = @id">
            <DeleteParameters>
                <asp:Parameter Name="id" Type="Int32" />
            </DeleteParameters>
            <UpdateParameters>
                <asp:Parameter Name="kassinimi" Type="String" />
                <asp:Parameter Name="kassimass" Type="Int32" />
                <asp:Parameter Name="id" Type="Int32" />
            </UpdateParameters>
            <InsertParameters>
                <asp:Parameter Name="kassinimi" Type="String" />
                <asp:Parameter Name="kassimass" Type="Int32" />
            </InsertParameters>
        </asp:SqlDataSource>
        <br />
        <asp:GridView ID="GridView1"
            runat="server" AutoGenerateColumns="False"
            DataKeyNames="id" DataSourceID="SqlDataSource1">
            <Columns>
                <asp:CommandField ShowDeleteButton="True"
                  ShowEditButton="True" DeleteText="Kustuta"
                  EditText="Muuda" UpdateText="Salvesta muutused"
                  CancelText="Katkesta"/>
                <asp:BoundField DataField="id" HeaderText="id"
                  InsertVisible="False" ReadOnly="True"
                  SortExpression="id" Visible="false" />
                <asp:BoundField DataField="kassinimi"
                  HeaderText="kassinimi" SortExpression="kassinimi" />
                <asp:BoundField DataField="kassimass"
                  HeaderText="kassimass" SortExpression="kassimass" />
            </Columns>
        </asp:GridView>

        <asp:FormView ID="FormView1" runat="server" DataKeyNames="id"
            DataSourceID="SqlDataSource1" DefaultMode="Insert">
            <EditItemTemplate>
                id:
                <asp:Label ID="idLabel1" runat="server"
                    Text='<%# Eval("id") %>' />
                <br />
```

```
            kassinimi:
            <asp:TextBox ID="kassinimiTextBox" runat="server"
                Text='<%# Bind("kassinimi") %>' />
            <br />
            kassimass:
            <asp:TextBox ID="kassimassTextBox" runat="server"
                Text='<%# Bind("kassimass") %>' />
            <br />
            <asp:LinkButton ID="UpdateButton" runat="server"
                CausesValidation="True"
                CommandName="Update" Text="Salvesta muutused" />
             
            <asp:LinkButton ID="UpdateCancelButton" runat="server"
                CausesValidation="False" CommandName="Cancel"
                Text="Katkesta" />
        </EditItemTemplate>
        <InsertItemTemplate>
            kassinimi:
            <asp:TextBox ID="kassinimiTextBox" runat="server"
                Text='<%# Bind("kassinimi") %>' />
            <br />
            kassimass:
            <asp:TextBox ID="kassimassTextBox" runat="server"
                Text='<%# Bind("kassimass") %>' />
            <br />
            <asp:LinkButton ID="InsertButton" runat="server"
                CausesValidation="True"
                CommandName="Insert" Text="Lisa" />
             
            <asp:LinkButton ID="InsertCancelButton" runat="server"
                CausesValidation="False" CommandName="Cancel"
                Text="Katkesta" />
        </InsertItemTemplate>
        <ItemTemplate>
            id:
            <asp:Label ID="idLabel" runat="server"
              Text='<%# Eval("id") %>' />
            <br />
            kassinimi:
            <asp:Label ID="kassinimiLabel" runat="server"
               Text='<%# Bind("kassinimi") %>' />
            <br />
            kassimass:
            <asp:Label ID="kassimassLabel" runat="server"
               Text='<%# Bind("kassimass") %>' />
            <br />
            <asp:LinkButton ID="EditButton" runat="server"
                CausesValidation="False"
                CommandName="Edit" Text="Muuda" />
             
            <asp:LinkButton ID="DeleteButton" runat="server"
                CausesValidation="False"
                CommandName="Delete" Text="Kustuta" />
             
            <asp:LinkButton ID="NewButton" runat="server"
                CausesValidation="False"
                CommandName="New" Text="Lisa" />
        </ItemTemplate>
    </asp:FormView>
</div>
</form>
```

```
</body>
</html>
```

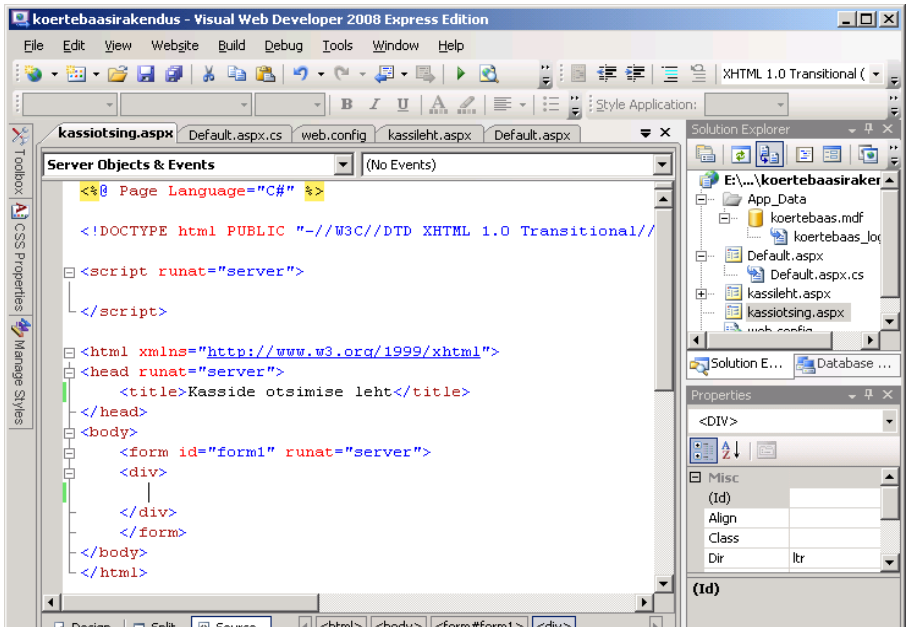
http://video.msn.com/video.aspx?vid=e8670aed-904d-4e81-8e23-2ee583f52655

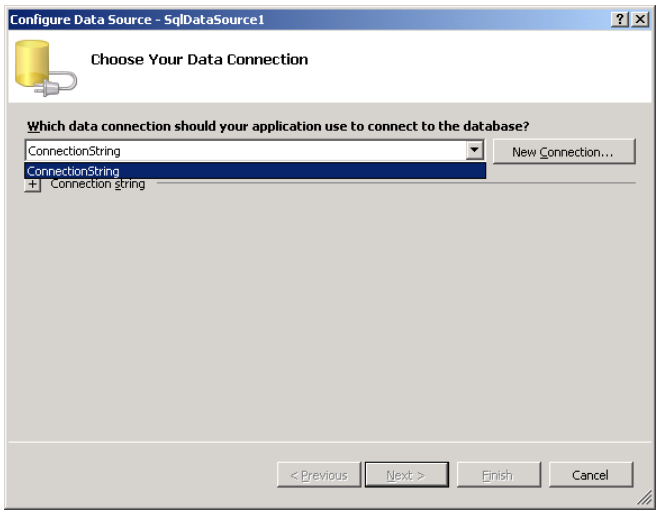
Ülesandeid

- Tee kasside näide otsast lõpuni läbi, veendu, et toimib
- Loo leht matkale inimeste registreerimiseks. Igaüks saab end lisada (eesnimi, telefon, elektronpost), lisatud andmed on vaid vaatamiseks.
- Kui oli eelnevalt tehtud matka tutvustav lehestik, siis seo lisamisvorm selle lehestikuga. Kui mitte, koosta eraldi paar lehte matka marsruudi ja päevakava tutvustuseks. Kasuta lehtedel ühist kujundusfaili.
- Lisa eraldi administraatorileht, kus saab vigaseid sisestusi kustutada ja muuta, kuid mitte lisada. Administraatorilehele pääseb ligi vaid failinime teades – otsest viidet sinna teistelt lehtedelt pole.
- Koosta veebilehestik paari Põhja-Eestis oleva joa tutvustusega. Kasutajad saavad lehtede vahele liikuda ning iga joa juures tekkinud muljete kohta kommentaare lisada. Kommentaarid on vastava joa lehel nähtavad. Administraatorile tulevad lehel ette kõik kommentaarid. Ta saab sealt ebasoovitavana tunduvaid kustutada.

Otsing

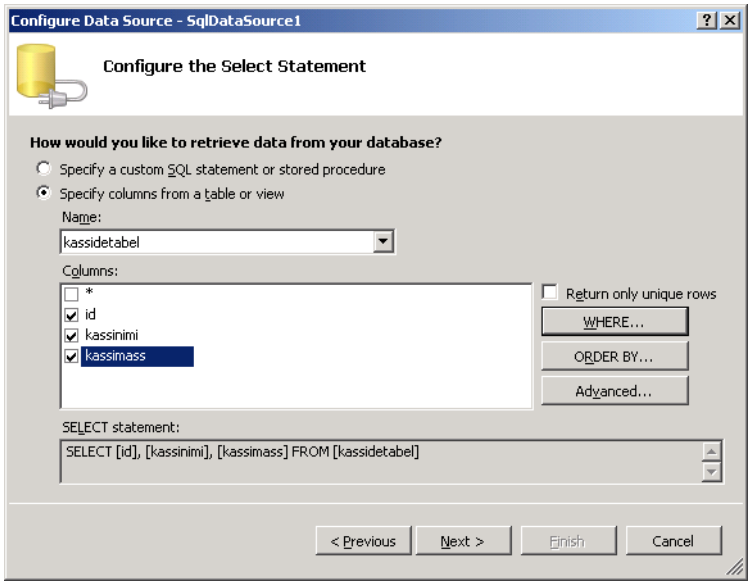
Andmebaasipõhine veebilehestik osutub vajalikuks näiteks juhul, kui tuleb leida vaid osa andmeid suurema hulga seast. Mõnikord piisab otsimisest sisukorrast, mõnikord aitab väike Javaskriptijupp mured lahendada. Aga kui andmeid on ikka tuhandeid, siis on serveripoolne abi tarvilik. Siin näitena teeme eelnevalt kokkupandud kassiloetelule juurde otsingu. Alustuseks uus eraldi ASP.NETi veebivorm ehk aspx leht. Kuna otsest programmikoodi tarvis pole, siis eraldi cs-koodifaili pole juurde vaja.



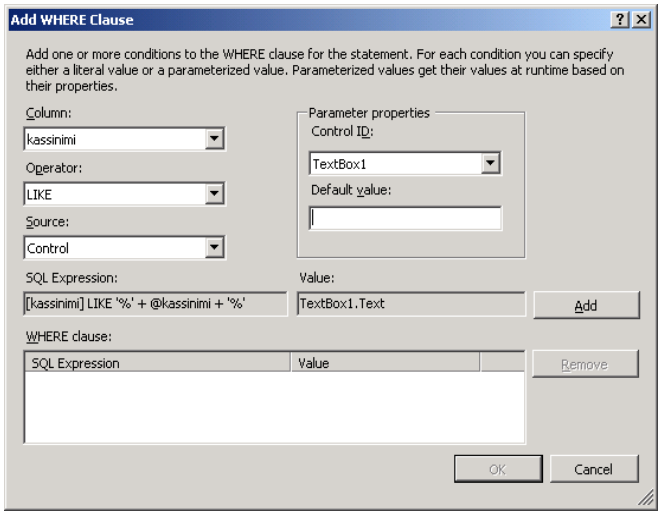


Et otsisõna kuhugi sisestada oleks, selleks sikutame veebilehele tekstikasti, milleks pannakse automaatne nimi Textbox1.

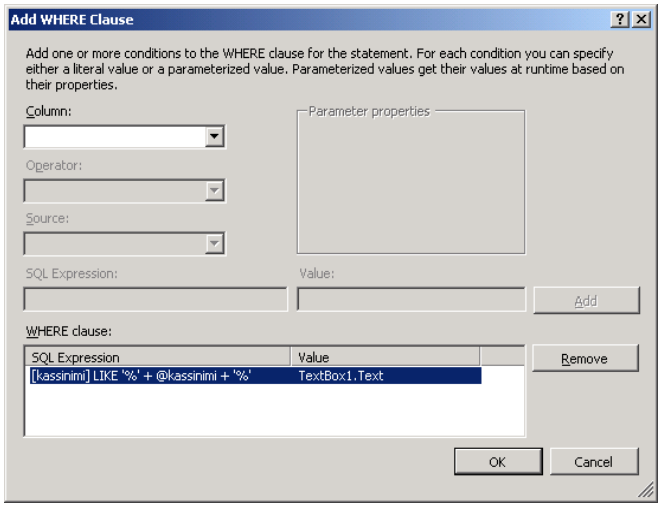
Andmebaasi juurde saab kasutada juba loodud ühendust, sest tegemist sama baasiga.



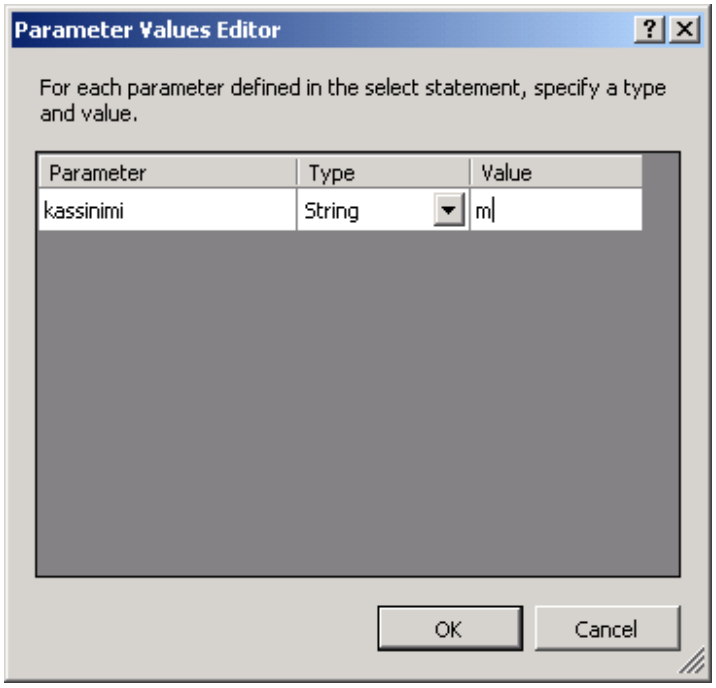
Andmete vaatamiseks jällegi kõigepealt „tavaline“ SELECT-lause



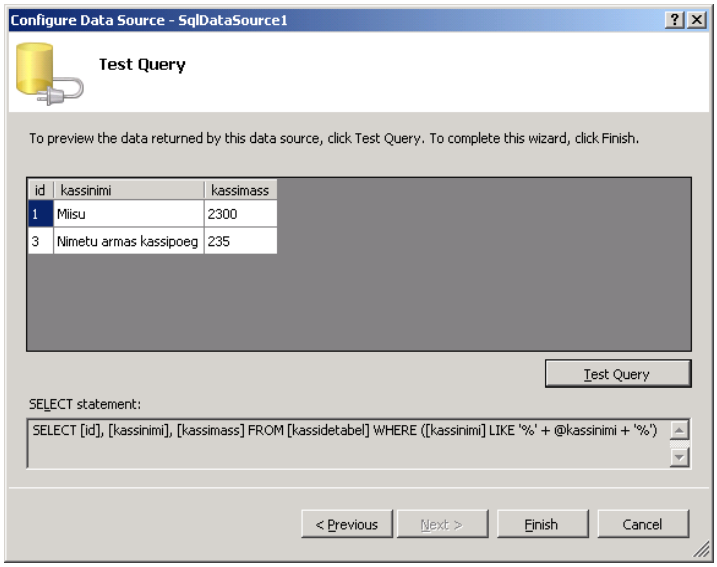
Otsingu puhul tasub eraldi WHERE-osas asuda piiranguid sättima. Avanevas aknas on selle jaoks loodud abivahend. Nime järgi otsingul tuleb tingimusega seotud tulbaks kassinimi. Operaator LIKE võimaldab leida selliseid nimesid, kus otsitav täht või lõik sees. Kust see lõik saadakse, määratakse järgnevast rippmenüüst. Veebilehel olevad elemendid (näiteks tekstiväli) käivad nimetuse alla Control. Valime lehel oleva ainukese tekstivälja ehk kasti nimega Textbox1. Alla tekib niimoodi automaatselt genereeritav SQL.



Pärast Add-nupule vajutamist jäetakse lisand alla ritta meelde.

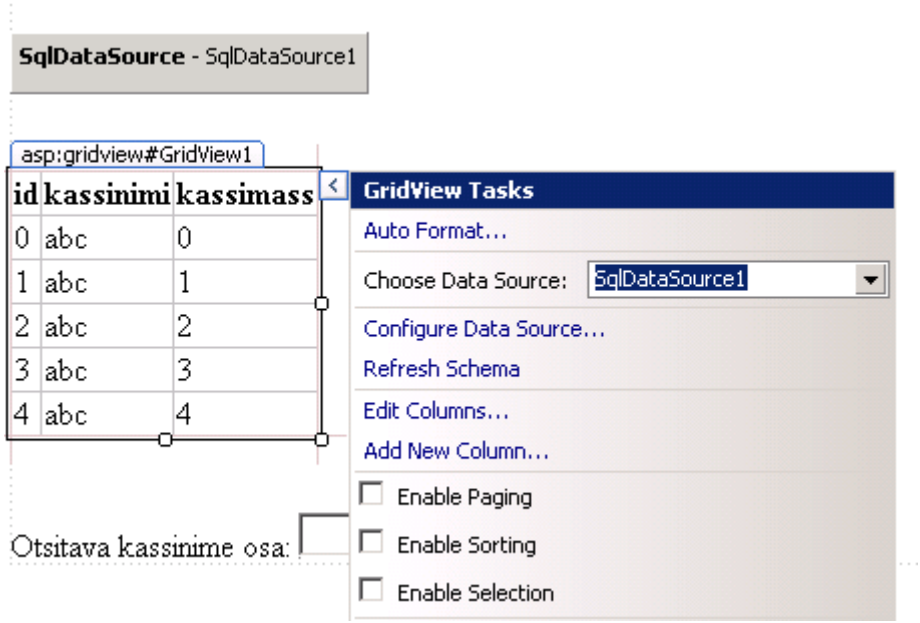


Testimise juures tasub loodud piirang järele proovida. Püüame näha kasse, kelle nimes sisaldub m-täht.

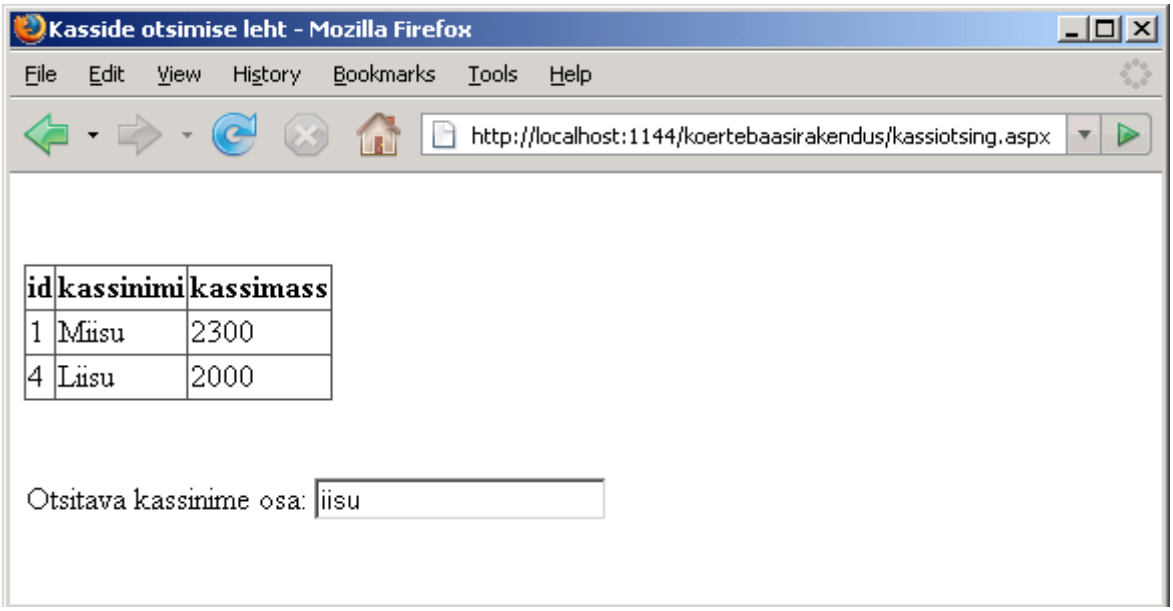


Ning need nimed tulidki nähtavale.

Nüüd on andmeallikas olemas ja seadistatud. Vaja veel abivahend, kust kaudu andmed ekraanile kuvada. Tuttav GridView saab sellega täiesti hakkama kui talle lihtsalt me ainuke ehk sobiv allikas määrata.



Veebilehelt otsime sedakorda kõiki kassinimesid, kelle nimi sisaldab lõiku „iisu“. Ning nagu selgub, võibki neid vaadata.

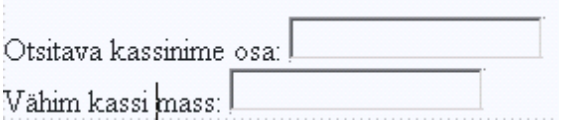


Graafiliselt programm kokku lohistada on tore küll. Aga vahetevahel on kasulik, kui ka koodipoolelt ülevaade säilib – hiljem nii kergem vigu otsida ja muutusi teha. Allpool siis andmeallika kood. ConnectionString ikka sama. SELECT-lausele on juurde tulnud piirang WHERE ([kassinimi] LIKE '%' + @kassinimi + '%'). Teades, et protsendimärk tähistab suvalist sümbolite jada, saab tingimuse lahti seletada nõnda, et kassinimi peab olema kujul hulk suvalisi sümboleid (ülakomade vahel protsent), siis parameetrina antud kassinime osa ning edasi taas suvaliste sümbolite hulk.

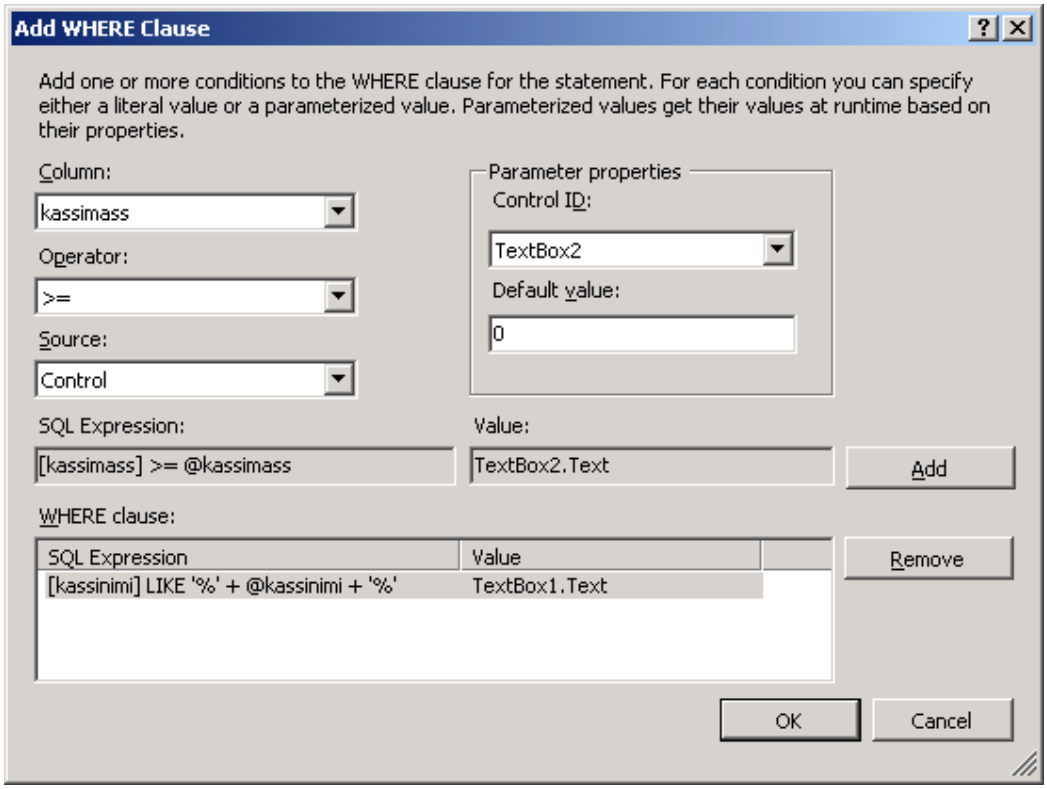
Eraldi asp:SqlDataSource alamelemendina on plokk SelectParameters. ControlParameter'i kaudu antakse teada, et SQL-lauses @-märgiga kassinime väärtus saadakse tegelikult elemendi TextBox1 väljalt Text. Nõnda seotaksegi otsad kokku ja kasutajal võimalik andmeid otsida.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    SelectCommand="SELECT [id], [kassinimi], [kassimass]
       FROM [kassidetabel]
       WHERE ([kassinimi] LIKE '%' + @kassinimi + '%')">
    <SelectParameters>
        <asp:ControlParameter ControlID="TextBox1" Name="kassinimi"
            PropertyName="Text"  Type="String" />
    </SelectParameters>
</asp:SqlDataSource>
```

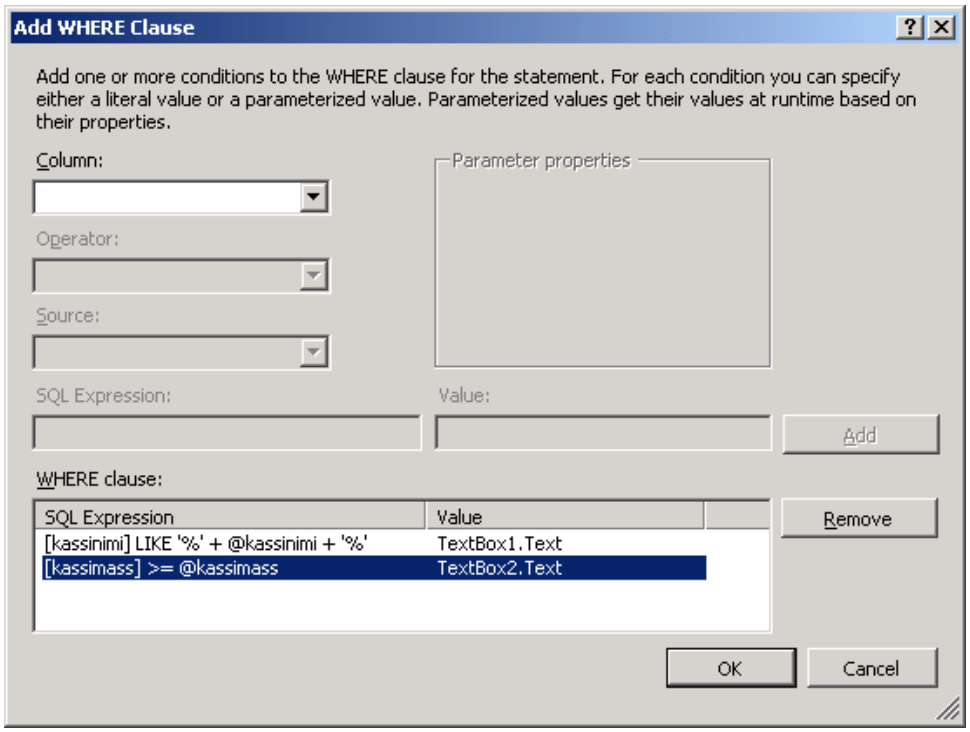
Mõnikord aga tahetakse otsida rohkem kui ühe tunnuse järgi. Või kord ühe ja kord teise järgi. Või mõnikord ka kõiki andmeid näha. Selleks teeme lehele teise sisestusvälja juurde, sedakorda kassi massi tarbeks.



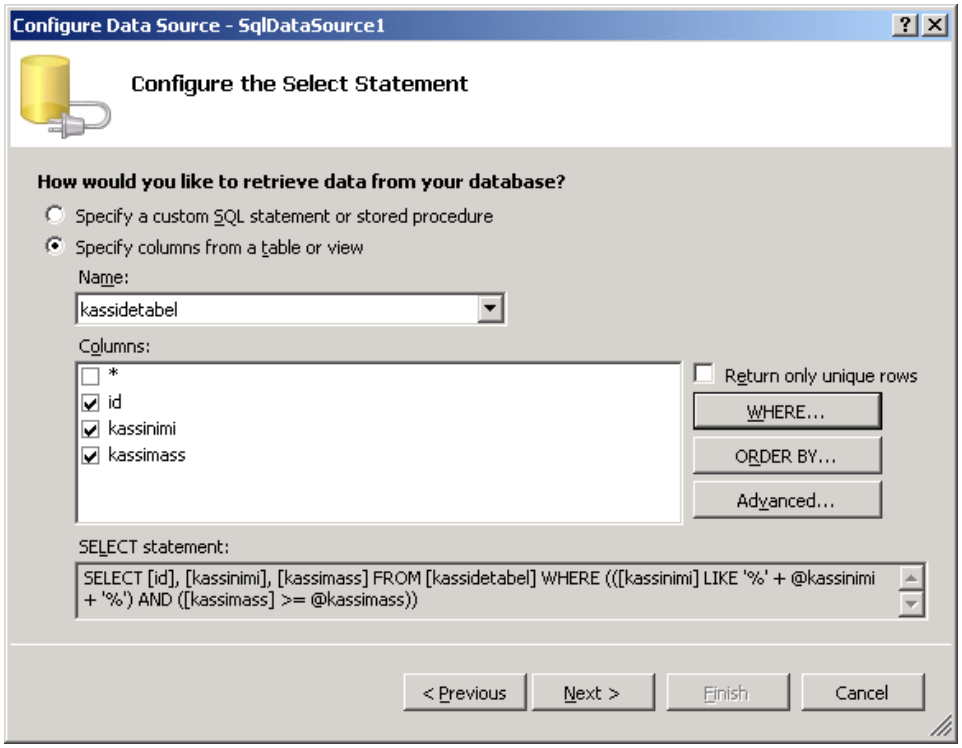
Edasi lisame andmeallika juurde uue tingimuse. Ning vaikimisi väärtuseks paneme 0. Ehk kui lahtrisse otsingulahtrisse andmeid ei kirjutata, siis näidatakse kõiki kasse, kelle mass on suurem või võrdne nulliga.



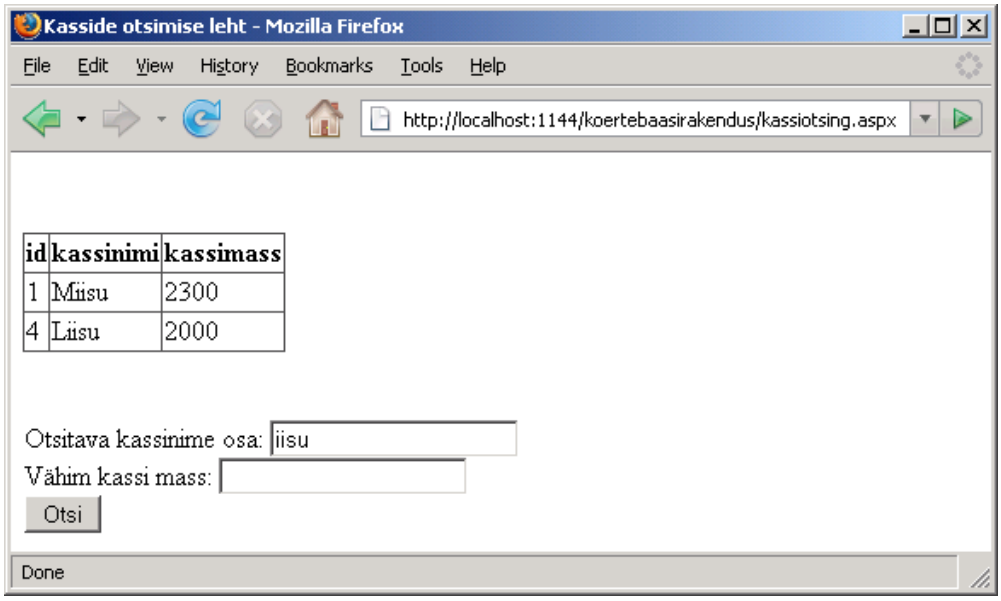
Ja ongi ilusti kaks piirangut kõrvuti.



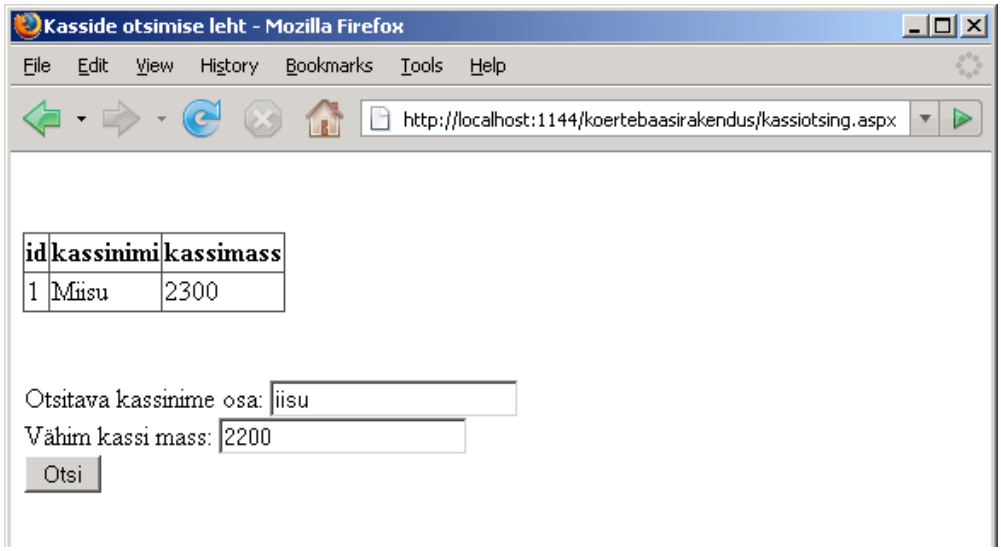
Samuti näha pikk SQL päring, kus kirjas nii tulpade nimed kui piirangud.



Kui massi ei piirata, näeb lehel kõiki sobiva nimeosaga kasse.



Piiramise korral aga ainult neid, kes filtrist läbi lähevad.



Et ka tekstide puhul sarnane omadus kehtiks, et tühja teksti puhul kõiki kasse näidataks, selleks tuleb kassinime parameetri juurde lisada omadus

```
ConvertEmptyStringToNull="false"
```

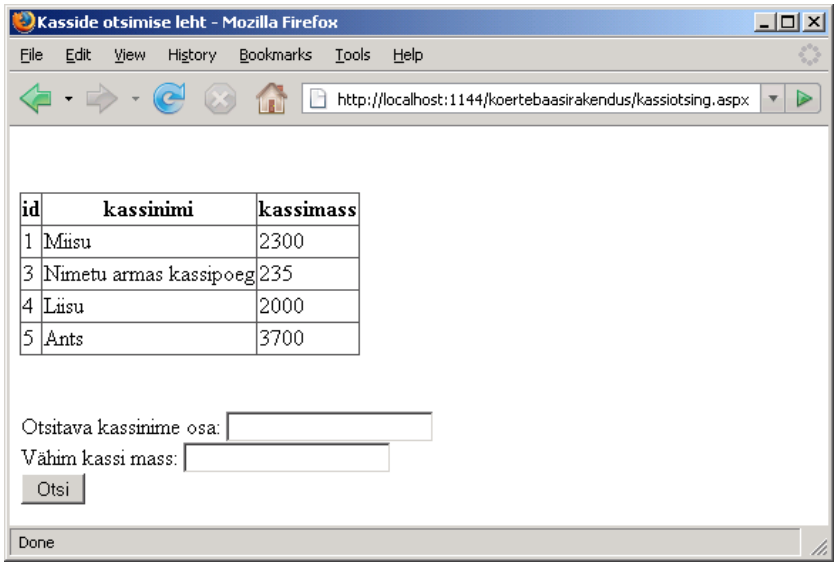
ehk siis

```
   <asp:ControlParameter ControlID="TextBox1" Name="kassinimi"
    PropertyName="Text" Type="String" ConvertEmptyStringToNull="false"/>
```

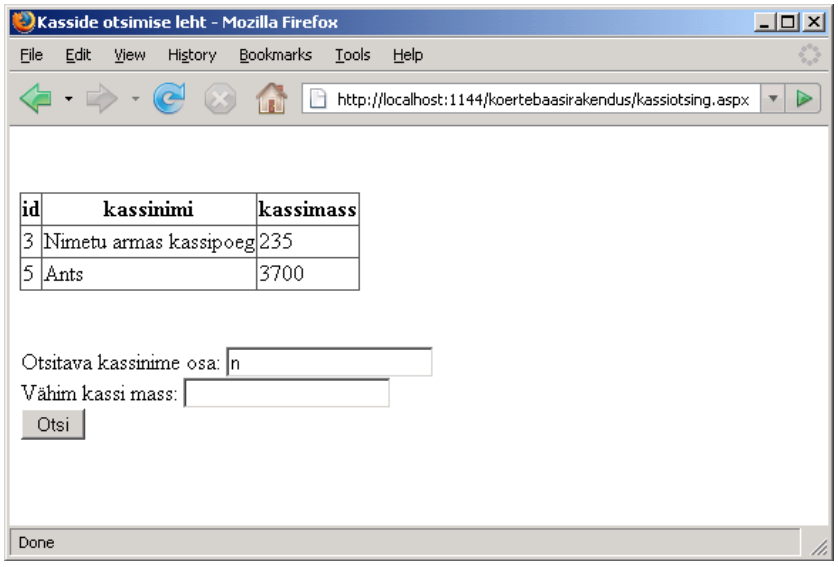
Muul juhul kiputakse tühi lahter muudetama eriliseks NULL-väärtuseks ning sellele vastavat nime ei leita. Kui aga tühi tekst jääb viisakalt tühjaks tekstiks ning protsendimärgid ütlevad, et selle ees ja taga võib olla suvaline hulk suvalisi sümboleid, siis tulemusena näidatakse tühja teksti puhul kõiki kasse. Ehk siis kogu andmeallika kood:

```
         <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT [id], [kassinimi], [kassimass] FROM
[kassidetabel] WHERE (([kassinimi] LIKE '%' + @kassinimi + '%') AND
([kassimass] &gt;= @kassimass))">
            <SelectParameters>
                <asp:ControlParameter ControlID="TextBox1"
                    Name="kassinimi" PropertyName="Text"
                    Type="String" ConvertEmptyStringToNull="false"/>
                <asp:ControlParameter ControlID="TextBox2"
                    DefaultValue="0" Name="kassimass"
                    PropertyName="Text" Type="Int32" />
            </SelectParameters>
        </asp:SqlDataSource>
```

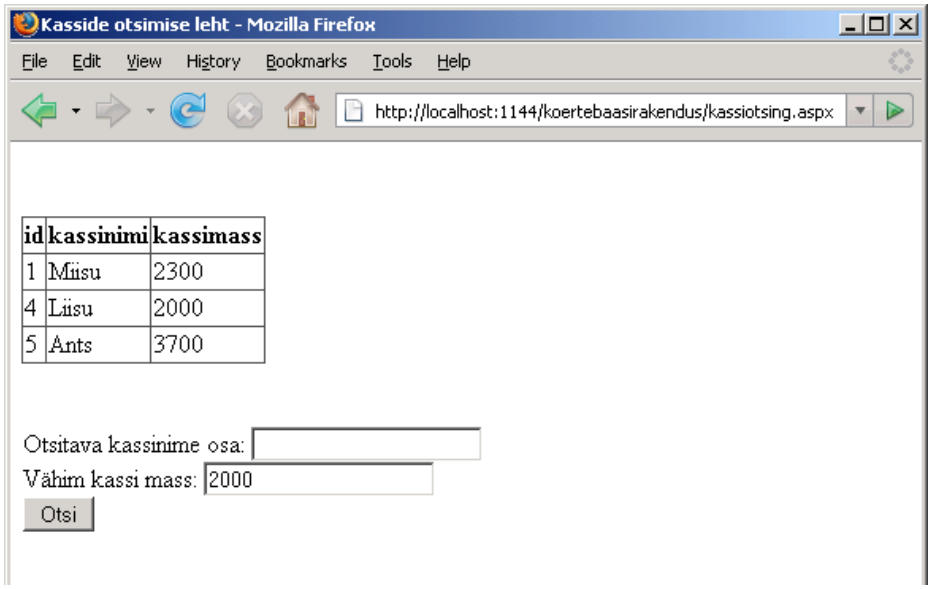
Ning ka veebilehel kõik kassid ilusti nähtaval.



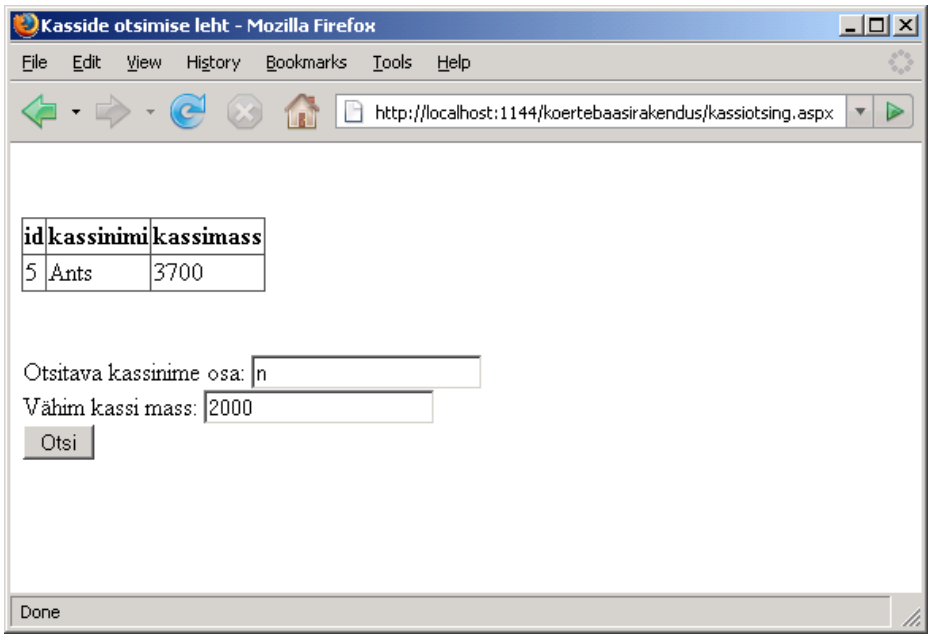
Piirates aga nähtavaks ainult n-i sisaldavad nimed, näeb neid



Ning kui massi alampiiriks panna kaks kilo, siis näha ainult pontsakamaid isendeid.



Ning töötavad ka mõlemad piirangud koos.



Ülesandeid

- Otsi üles või koosta uuesti matkale registreerumise vorm, kuhu inimesed saavad omi nimesid ja kontaktandmeid salvestada.

- Loo võimalus inimeste otsimiseks mitmesuguste andmete järgi: näiteks eesnimi, perekonnanimi, sünniaasta (või nende vahemik), sugu.

- Loo eraldi lehekülg, kus näidatakse ainult pärast aastat 2000 sündinud inimesi (vajalik muuta SQL-lauset, tekstivälja ega parameetrit pole vaja).

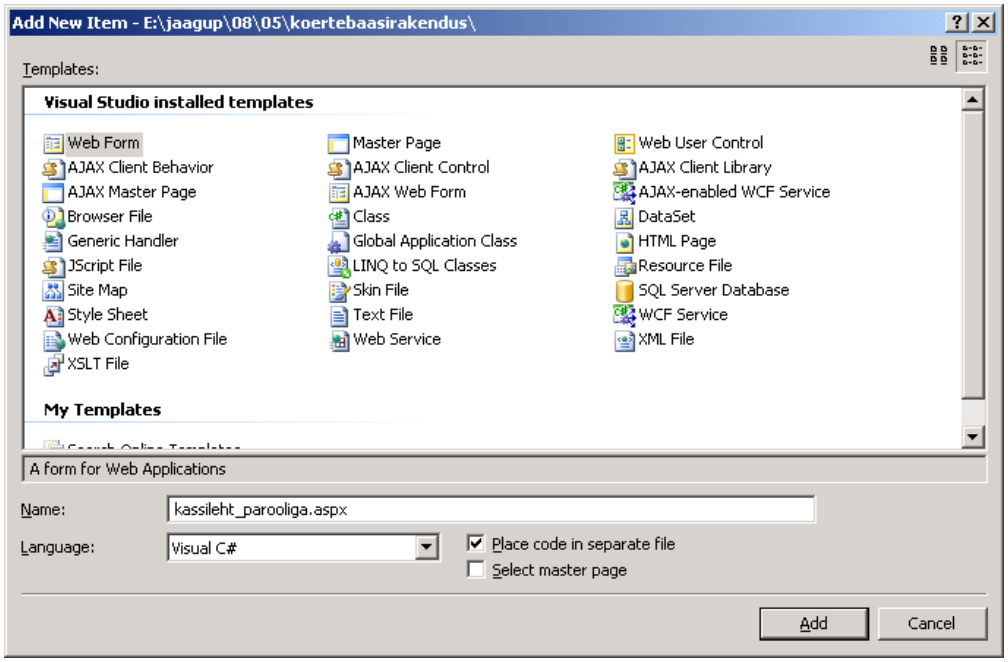
- Võimalda otsingul sugu valida rippmenüüst (leia, milline rippmenüü (DropDownList) omadus sobib siduda SQL-lausega).

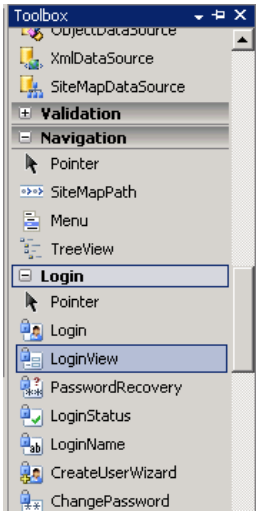
Lehe kaitsmine parooliga

Avalik ligipääs on lihtne ja hea, aga millegipärast ikka ei suudeta veel kõiki inimesi usaldada. Või ka mõnikord usaldatakse, aga pannakse ikka sahtlile või majale lukk ette – et kogemata ei tekiks kiusatust näiteks lapsel nõudepesuvahendit juua või naabrimehel öisel ajal uksega eksida. Ja ka iseenese eest on vahel kasulik mõned asjad luku taha panna – et tasub enne mitu korda järele mõelda, kas ikka on midagi õrna või ohtlikku vaja niisama kätte võtta.

Sarnane lugu ka veebilehtede juures. Kui andmete muutmise ja parandamise tarbeks tuleb eraldi sisse logida, siis pole karta, et kogemata andmeid vaadates mõne hiireklõpsu peale miskit paigast ära läheb.

Parooliga kaitstud andmete leht on leht nagu iga teinegi. Lehe loomisel on aga kasulik märkida, et soovime eraldi koodifaili: sinna saab hiljem paigutada kasutaja andmete sobivuse kontrolli.



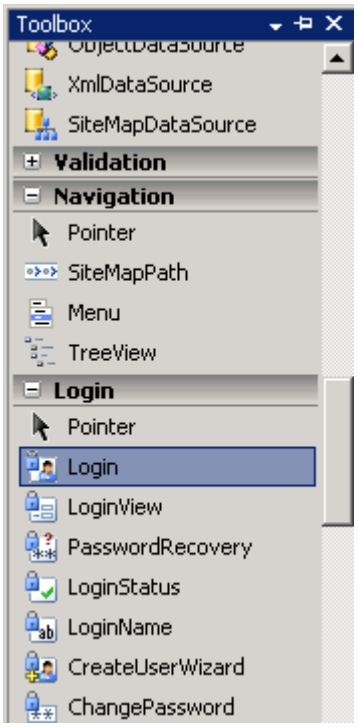


Kui osa lehe sisust on parooliga kaitstud, siis tuleb määrata, mida näidata anonüümsele ehk sisselogimata kasutajale ning mida sisenenud kasutajale. Sellise valiku jaoks on loodud element nimega LoginView

Nagu aimata võib, on elemendi sees kaks suuremat plokki: AnonymousTemplate ning LoggedInTemplate.

```
<asp:LoginView ID="LoginView1" runat="server">
  <AnonymousTemplate>
  </AnonymousTemplate>
  <LoggedInTemplate>
  </LoggedInTemplate>
</asp:LoginView>
```

Korraga mõlemat ei näidata. Sisu tuleb vastavalt sellele, kummas seisundis kasutaja on.

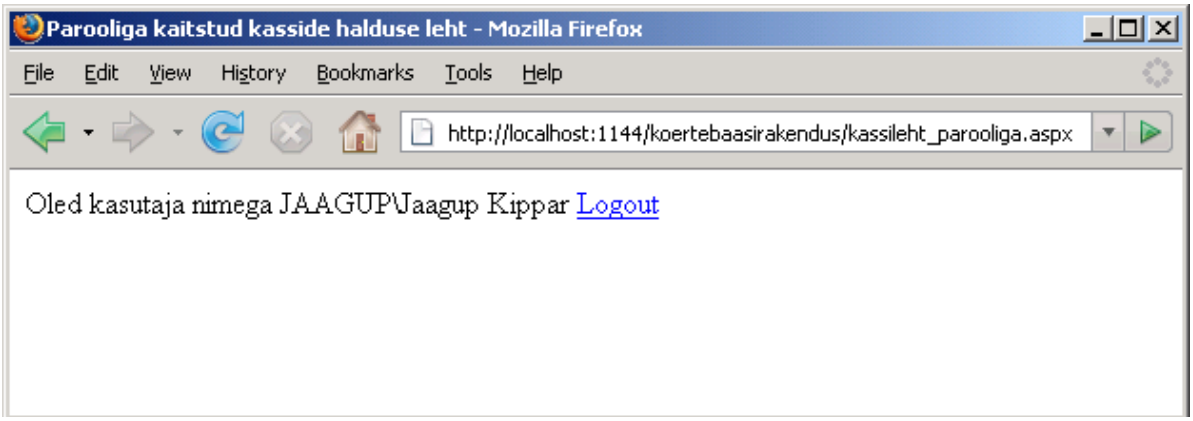


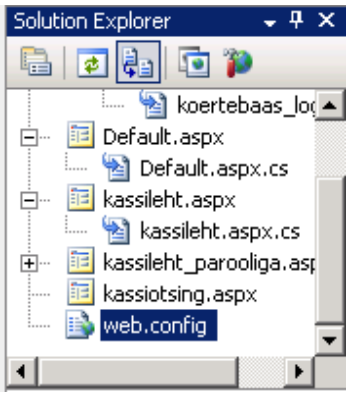
Anonüümsele kasutajale on tõenäoliselt viisakas pakkuda sissemeldimisvõimalust. Selle jaoks on olemas standardne element nimega Login, kus küljes kasutajanime ja parooli sisestamise kohad. Samuti paik andmete sobivuse kontrolliks koodiga.

Sisselogitule on viisakas teatada, kes ta on. Selleks sobib kontroll nimega LoginName. Et taas välja pääseks, siis aitab LoginStatus, miss sisselogitule vaikimisi pakub võimaluse „logout“.

```
<asp:LoginView ID="LoginView1" runat="server">
  <AnonymousTemplate>
      Tere. Oled tundmatu kasutaja.
      <asp:Login ID="Login1" runat="server">
      </asp:Login>
  </AnonymousTemplate>
  <LoggedInTemplate>
     Oled kasutaja nimega
      <asp:LoginName ID="LoginName1" runat="server" />
      <asp:LoginStatus ID="LoginStatus1" runat="server" />
  </LoggedInTemplate>
</asp:LoginView>
```

Kui rõõmsasti leht käivitada, siis teatatakse, et olen kasutaja nimega JAAGUP\Jaagup Kippar. Nuputan, et kus ta küll seda võis teada. Aga siis meenub, et olen vastava nimega masinasse selle tunnusega sisse loginud. Ehk siis kui pole määratud teisiti, kasutatakse ASP.NETi juures Windowsi kasutajaid ning nende õigusi. Nii et juhul, kui koolis/asutuses on juba välja mõeldud korralik Windowsi kasutajate/gruppide süsteem ning tahetakse ka veebis samu tunnuseid ja õigusi kasutada, on sealtkaudu toimetamine mugav. Kasutajanimed ja paroolid jäävad samaks, lisada vaid vaja lehed, mis sobivaid andmeid serveerivad.





Mõnikord aga ei pruugi veebikasutajad ja kohalikud kasutajad kattuda. Et veebirakenduses õnnestuks kohalikke kasutajaid teha, selleks tuleb muudatus teha failis nimega web.config.

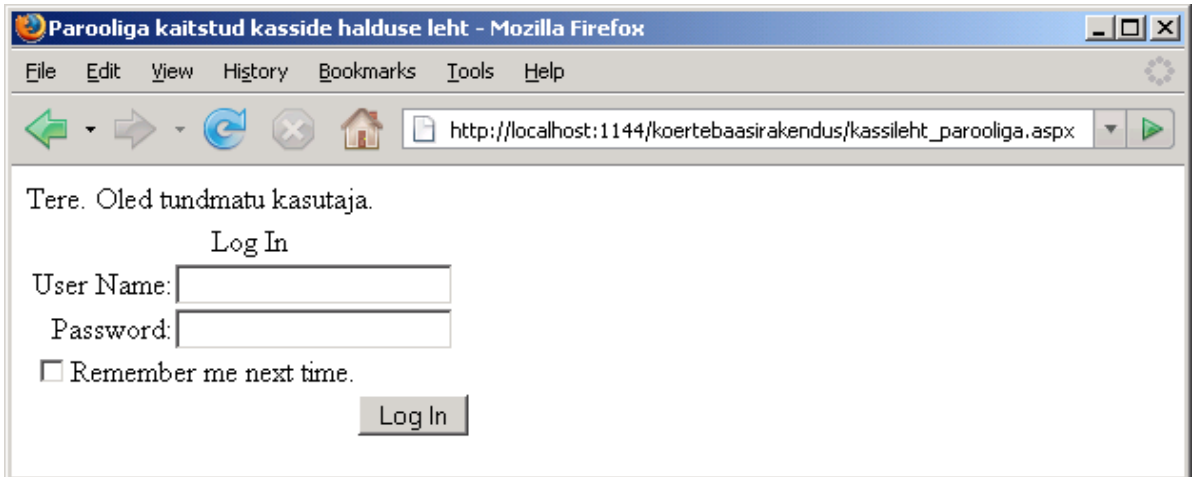
Mõningase otsimise peale peaks sealt leidma rea

```
<authentication mode="Windows" />
```

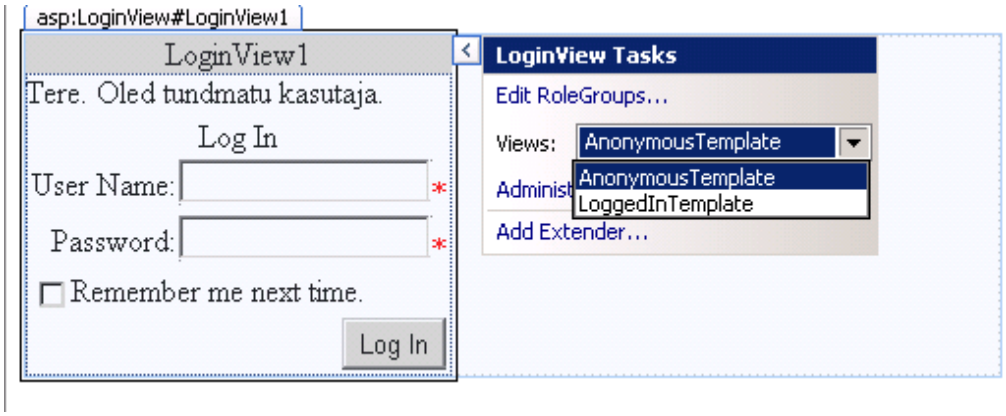
mis tasuks asendada reaga

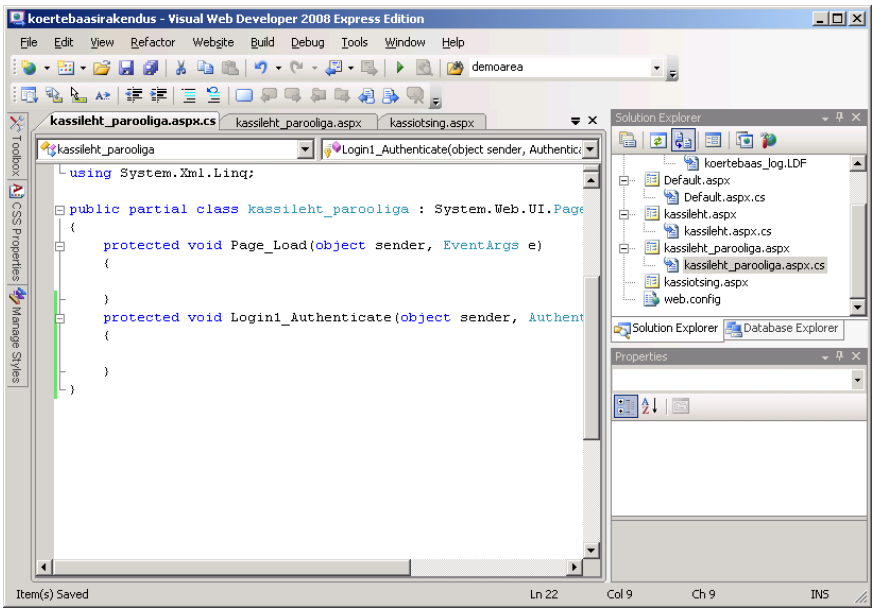
```
<authentication mode="Forms" />
```

ehk siis Windowsi kasutajate põhine autentimine asendatakse vormist sisestatud andmete põhjal autentimisega. Kui nüüd loodud leht avada, siis teatatakse, et tegemist on tundmatu kasutajaga – pole ju veel kuhugi kasutajanime ja parooli löödud. Ja ega sellest löömisest poleks ju ka kasu veel – sest kusagil pole kirjas, mis vastava nime ja parooliga peale hakata.



Sisenemistunnuste kontrolliks üks mugav viis on registreerida programmikood Login-kontrolli sisestusnupu külge. Tasub disainivaates LoginView puhul ette manada AnonymousTemplate mood koos seal sees oleva Login elemendiga. LoginView puhul saab rippmenüüst valida, kumba malli parajasti kasutatakse.





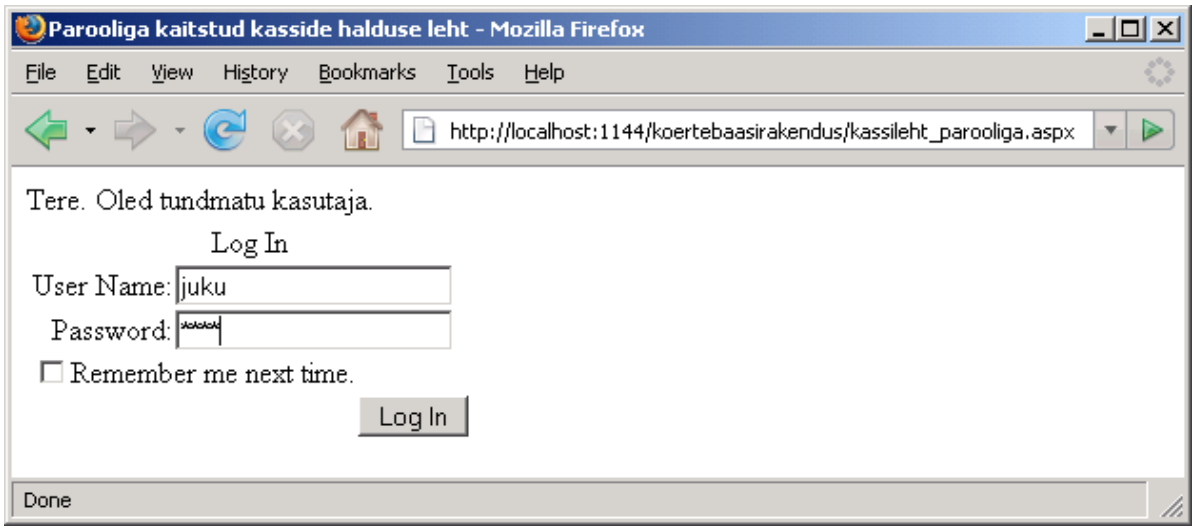
Edasi topeltklõps sisenemisnupul tekitab koodifaili sisse funktsiooni nimega Login1_Authenticate ning kirjutab ka nupu juurde kuuluvasse ossa, et nupuvajutusel tuleb vastav funktsioon käivitada.

Tekkinud kohale tasub siis paigutada koodijupp kontrollimaks, kas sisestatud andmed sobivad. Nagu näha, tuli funktsioonile ehk alamprogrammile kaks parameetrit. Esimene (sender, tüübist object) näitab, millisele elemendile vajutati, ehk siis mis on sündmuse saatjaks/allikaks. Teine (nimega e, tüübist AuthenticateEventArgs ehk siis autentimisega kaasas käivad argumendid) on mõeldud kohaks, kuhu koodiga teada anda, kas andmed sobisid.

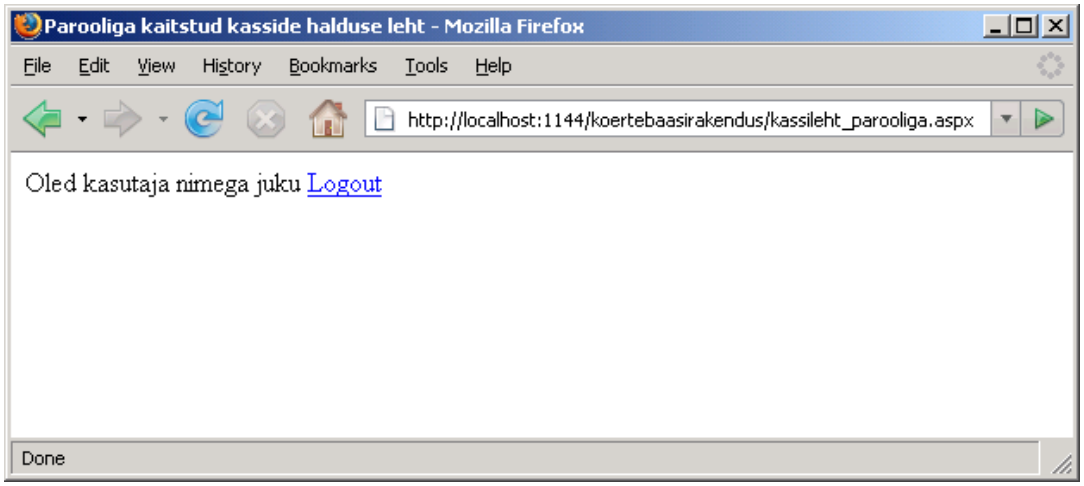
Esialgu piirdume lihtsa kontrolliga. Et allikaobjektist saaks mugavasti andmed kätte, selleks püüame ta muutujasse tüübist Login, nimega sisestusvorm. Nii nagu arve ja tekste saab meelde jätta, nii kehtib see ka muude ligipääsetavate elementide kohta. Objektimuutujast küsime eraldi väljadena välja kasutajanime ja parooli ning võrdleme neid juku ja kalaga. Tingimusele järgi käitumiseks on C# keeles if-käsklus. Tingimus pannakse ümarsulgude vahele. Võrdlemiseks kasutatakse kahte võrdusmärki. Kui soovitakse, et mitu tingimust oleks tõese väärtuse jaoks korraga täidetud, siis pannakse tingimuste vahele kaks &-märki. Ja tõese väärtuse puhul tehtav toiming kirjutatakse loogeliste sulgude {} vahele. Sedakorda määratakse siis saabunud argumendi omadus Authenticated tõeseks, et ASP.NETi serveerija teaks arvestada, et sisestatud nimega kasutaja on sisse loginud.

```
protected void Login1_Authenticate(
           object sender, AuthenticateEventArgs e)
{
    Login sisestusvorm = (Login)sender;
    if (sisestusvorm.UserName == "juku" &&
        sisestusvorm.Password == "kala") {
           e.Authenticated = true;
    }
}
```

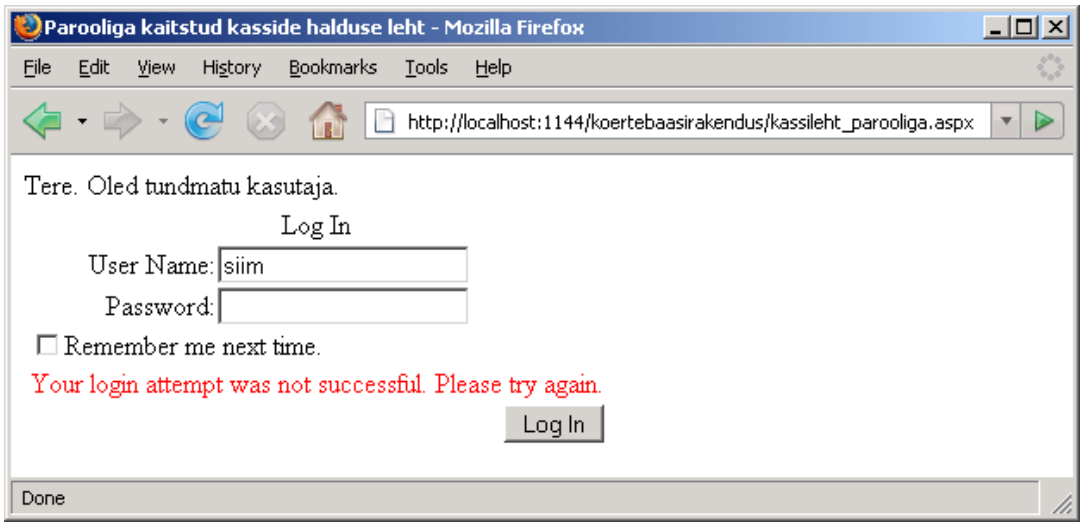
Kirjutades sisestusväljadesse juku ja kala



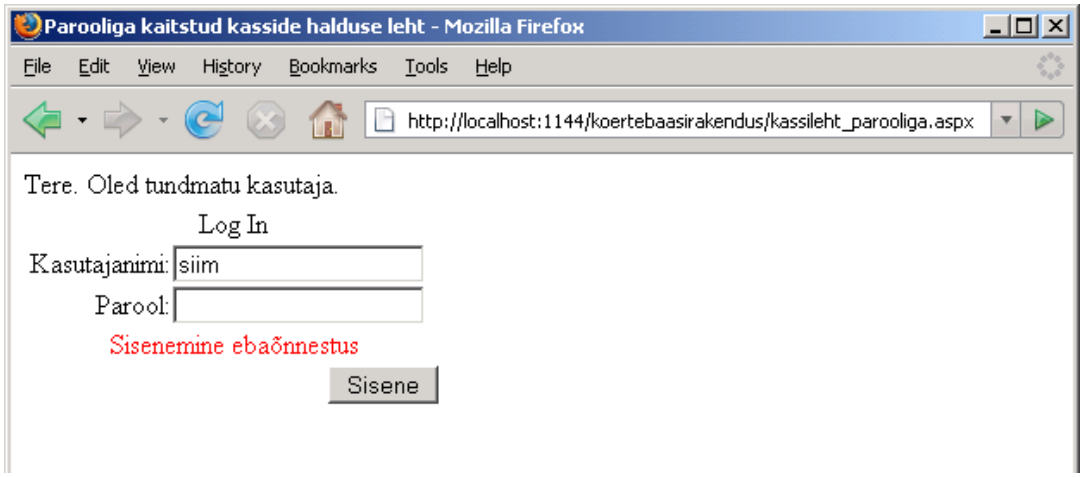
saab kasutaja ilusti veebilehele „sisse“.

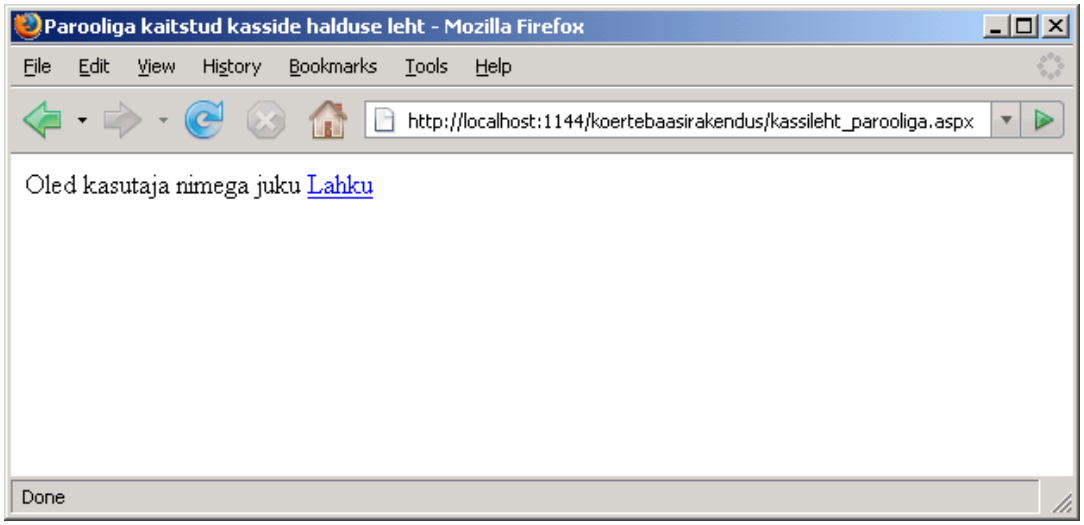


Edasi Logout-käsklusele vajutades pääseb välja. Ning kui tundmatu kasutajanimega püütakse siseneda, siis ei lasta sisse.



Veidi tõlketööd sisenemiselemendi juures ning võib juba maakeelset teksti imetleda.

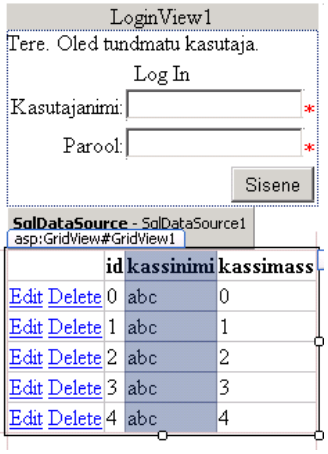




Tavajuhul ei soovi me kasutaja meeldejätmist arvuti taga. Ning kui omadus DisplayRememberMe määrata vääraks, siis saabki sellest kastikesest lahti. Sarnaselt saab LoginStatus elemendile määrata näidatava teksti ning ongi kõik silma alla jääv tõlgitud.

```
<asp:LoginView ID="LoginView1" runat="server">
  <AnonymousTemplate>
      Tere. Oled tundmatu kasutaja.
      <asp:Login ID="Login1" runat="server"
        onauthenticate="Login1_Authenticate"
        LoginButtonText="Sisene"
        UserNameLabelText="Kasutajanimi:"
        PasswordLabelText="Parool:"
        DisplayRememberMe="false"
        FailureText="Sisenemine ebaõnnestus">
      </asp:Login>
  </AnonymousTemplate>
  <LoggedInTemplate>
     Oled kasutaja nimega
      <asp:LoginName ID="LoginName1" runat="server" />
      <asp:LoginStatus ID="LoginStatus1" runat="server"
            LogoutText="Lahku" />
  </LoggedInTemplate>
```

```
        </asp:LoginView>
```



See oli nüüd ainult sisselogimise pool. Logimine vajalik aga enamasti selleks, et mingit ressurssi parooliga kaitsta. Siinsel lehel sobib kaitstavaks näiteks kasside andmete muutmis- ja kustutusõigus.

Lihtsam moodus on vastav paranduskomplekt ehk andmeallikas+GridView eelnevalt valmis teha ning siis vastav koodilõik LoginView LoggedInTemplate sisse kopeerida.

Vaikimisi genereerimise puhul tuleb GridView päris pikk, sest üksikute tulpade nimed kirjutatakse eraldi välja ning automaatselt ei lubata neid genereerida. Tahtes aga koodi lühemaks saada, võib atribuudi AutoGenerateColumns GridView seest välja võtta või tõeseks muuta ning siis osatakse juba otse andmeallikast veerge võtta. Lisaveeruks jääb vaid CommandField, mille sees nähtavad kustutamis- ja muutmisnupp.

```
<asp:GridView ID="GridView1" runat="server"
            DataKeyNames="id" DataSourceID="SqlDataSource1">
            <Columns>
                <asp:CommandField ShowDeleteButton="True"
                                  ShowEditButton="True" />
            </Columns>
 </asp:GridView>
```

Jõudsime lehestikuga siis nõnda kaugele, et esialgu saavad kõik lehe avajad ka kasside tabelis muutusi teha. Ehk siis nii andmeallikas kui ka vaade on väljapool (tagapool) LoginView'd nagu alloleval pildil näha. SqlDataSource on plussmärgiga kokku tõmmatud ja hall, kuna nii on seda skeemil kergem näha. Soovides muutmisõigust aga piirata, tuleb mõlemad tõsta ülespoole, ainult sisseloginud kasutajale määratud piirkonda.

```
            </asp:Login>
        </AnonymousTemplate>
        <LoggedInTemplate>
            Oled kasutaja nimega
            <asp:LoginName ID="LoginName1" runat="server" /
            <asp:LoginStatus ID="LoginStatus1" runat="serve
        </LoggedInTemplate>
    </asp:LoginView>
    <asp:SqlDataSource ID="SqlDataSource1" ...>...</asp:S
    <asp:GridView ID="GridView1" runat="server"
        DataKeyNames="id" DataSourceID="SqlDataSource1">
        <Columns>
            <asp:CommandField ShowDeleteButton="True" Sho
        </Columns>
```

Pärast tõstmist näeb kood järgnev välja – ehk siis andmeallikas koos tabeliga on LoggedInTemplate sees.

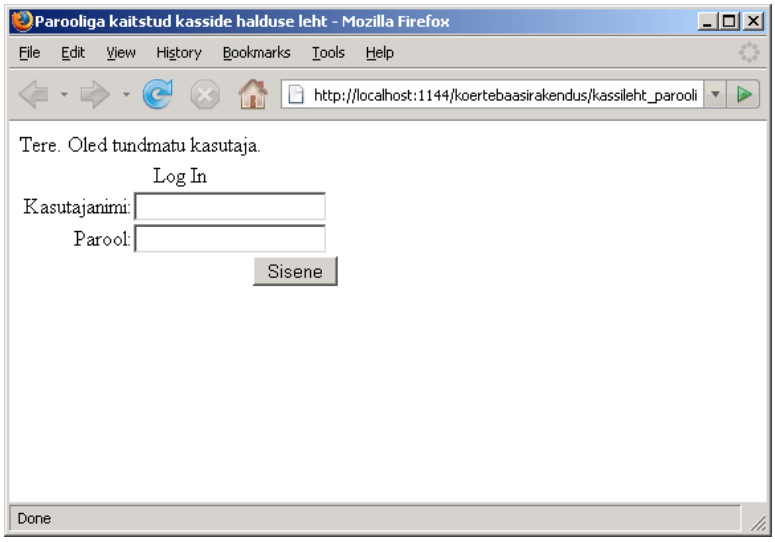
```
        </AnonymousTemplate>
        <LoggedInTemplate>
            Oled kasutaja nimega
            <asp:LoginName ID="LoginName1" runat="server" /
            <asp:LoginStatus ID="LoginStatus1" runat="serve
            <asp:SqlDataSource ID="SqlDataSource1" ...>...<
            <asp:GridView ID="GridView1" runat="server" Dat
                <Columns>
                    <asp:CommandField ShowDeleteButton="Tru
                </Columns>
            </asp:GridView>
        </LoggedInTemplate>
    </asp:LoginView>
```

Ning parooliga kaitstud aspx-lehe lähtekood tervikuna.

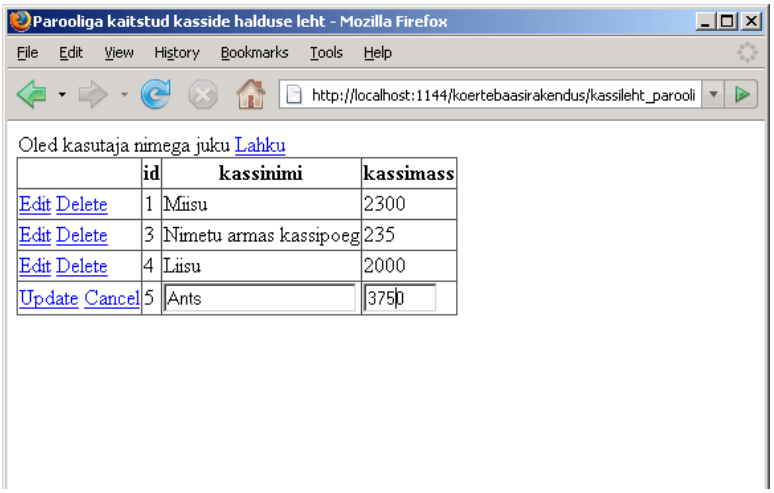
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="kassileht_parooliga.aspx.cs" Inherits="kassileht_parooliga" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Parooliga kaitstud kasside halduse leht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:LoginView ID="LoginView1" runat="server">
          <AnonymousTemplate>
              Tere. Oled tundmatu kasutaja.
              <asp:Login ID="Login1" runat="server"
                onauthenticate="Login1_Authenticate"
                LoginButtonText="Sisene" UserNameLabelText="Kasutajanimi:"
                PasswordLabelText="Parool:"
                DisplayRememberMe="false" FailureText="Sisenemine ebaõnnestus">
              </asp:Login>
          </AnonymousTemplate>
          <LoggedInTemplate>
             Oled kasutaja nimega
              <asp:LoginName ID="LoginName1" runat="server" />
```

```
                <asp:LoginStatus ID="LoginStatus1" runat="server"
                     LogoutText="Lahku" />
                <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                     ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
                    DeleteCommand="DELETE FROM [kassidetabel] WHERE [id] = @id"
                    InsertCommand="INSERT INTO [kassidetabel]
                                                    ([kassinimi], [kassimass])
                                   VALUES (@kassinimi, @kassimass)"
                    SelectCommand="SELECT [id], [kassinimi], [kassimass]
                                   FROM [kassidetabel]"
                    UpdateCommand="UPDATE [kassidetabel]
                                SET [kassinimi] = @kassinimi, [kassimass] = @kassimass
                                WHERE [id] = @id">
                    <DeleteParameters>
                        <asp:Parameter Name="id" Type="Int32" />
                    </DeleteParameters>
                    <UpdateParameters>
                        <asp:Parameter Name="kassinimi" Type="String" />
                        <asp:Parameter Name="kassimass" Type="Int32" />
                        <asp:Parameter Name="id" Type="Int32" />
                    </UpdateParameters>
                    <InsertParameters>
                        <asp:Parameter Name="kassinimi" Type="String" />
                        <asp:Parameter Name="kassimass" Type="Int32" />
                    </InsertParameters>
                </asp:SqlDataSource>
                <asp:GridView ID="GridView1" runat="server"
                              DataKeyNames="id" DataSourceID="SqlDataSource1">
                    <Columns>
                        <asp:CommandField ShowDeleteButton="True"
                            ShowEditButton="True" />
                    </Columns>
                </asp:GridView>
            </LoggedInTemplate>
        </asp:LoginView>
    </div>
    </form>
</body>
</html>
```

Nüüd võib rahus rakendusse siseneda vaid lubatud tunnustega tegelane



Ja vaid tema pääseb andmeid muutma ja kustutama.



Ülesandeid

- Loo või otsi üles matkale registreerumise vorm ja tabel. Eraldi administraatorilehel on võimalik andmeid kustutada ja parandada.

- Kaitse administraatorileht kasutajanime ja parooliga, nii et sinna igaüks ligi ei pääse.

Andmed mitmes tabelis

Andmeid on aegade jooksul arvutis hoitud mitut moodi. 1980ndatest aastatest alates aga on enamik suuremaid andmekogumeid traditsiooniliselt paigutatud tabelitel põhinevasse ehk relatsioonilisse andmebaasi. 2000ndatest alates tulevad jõuliselt juurde objektorienteeritud ning XMLi põhised andmebaasisüsteemid. Aga andmestikud on visad muutuma ning 2008ndal aastal on veel vähemasti kolmveerand arvutis hoitavatest andmetest ikka andmetabelites – nende jaoks on nii palju häid hoidmis-, otsingu- ja kiirendusmooduseid loodud, et muudel süsteemidel on raske konkureerida. Ning miks peakski muutma hästi töötavaid asju. Üheks arengusuunaks on veel moodus saada sarnaselt andmeid kätte väga mitmesugustest allikatest. Microsoftil on selle tarbeks loodud eraldi päringukeel nimega LINQ. Aga kuhu areng viib, eks seda näitab aeg.

Tabelite loomine ja sidumine

Tabelitepõhises andmebaasisüsteemis on kõik andmed tabelis. Nii „tavalised“ andmed nagu kellegi eesnimi. Kui ka mitmesugused seosed – et kes mida ostis või kes kus käis. Samuti peetakse relatsioonilise andmebaasi loomise juures silmas, et andmeid võimalusel ei korrataks. Ehk siis kui näiteks poebaasis on kirjas ostetav kaup koos oma andmetega (nimetus, hind, tootja) ning ostja (eesnimi, perekonnanimi, telefon, aadress), siis ostu juurde ei kirjutata mitte kõiki seitset omadust, vaid jäetakse lihtsalt meelde, et sellise koodiga ostja ostis vastava koodiga toote. Ning alles andmete väljatrüki juures kasutajaliideses trükitakse need andmed ühte ritta, kui see peaks vajalik olema. Nõnda on ühel ostjal alati ja ainult üks aadress ning pole muret, et aadressi muutumise korral vana tootega seotud teated valele aadressile võiksid minna.

Püüame võimalikult lihtsalt alustada. Pärast ühe tabeliga rakenduse koostamist on järgmine samm kahe andmetabeliga rakenduse loomine. Kaks tabelit läheb vaja üldjuhul oludes, kus andmestikul on küljes tunnus, mis võib sarnasena olla mitmel elemendil. Näiteks inimeste loetelu puhul võivad mitu neist olla sündinud samas linnas. Ning linnal võib olla lisaks linnanimele juures ka geograafilised koordinaadid ja rahvaarv. Sellisel juhul ei kirjutata linna nime inimese juurde, vaid pigem tehakse eraldi linnade tabel, kus igal real kirjas ühe linna andmed (kaasaarvatud unikaalne kood) ning inimeste tabelisse pannakse iga inimese sünnilinna juurde kirja lihtsalt vastava linna kood. Eks siis edasise programmi juures juba vaadatakse, kas on vaja linna kohta küsida nime, asukohta või näiteks linnapea nime.

	id	kassinimi	kassimass
▶	1	Miisu	2300
	3	Nimetu armas ka…	235
	4	Liisu	2000
	5	Ants	3750
*	NULL	NULL	NULL

Olemasolevat näidet edasi arendades tuletame meelde, et meil on praeguseks olemas kasside tabel.

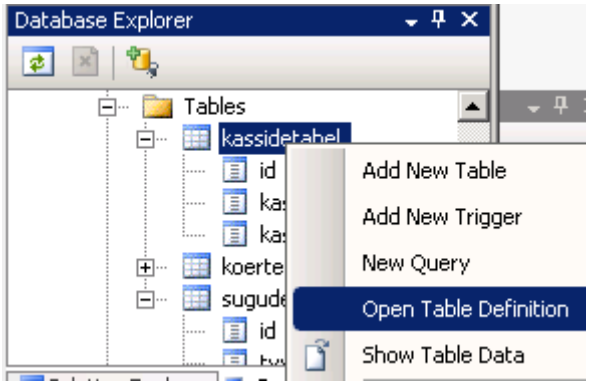
	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	tyyp	nvarchar(50)	☐
▶	kirjeldus	nvarchar(50)	☐
			☐

Iga kassi juurde on võimalik lisada sootunnus. Kusjuures keerukuse huvides jätame soo kohta nii tüübi kui kirjelduse – siis rakenduses võimalik valida, mida parajasti vaja kasutada.

	id	tyyp	kirjeldus
	1	Teadmata	Sugu teadmata
	2	Isane	Fertiilne isane
	3	Emane	Fertiilne emane
	4	Kohiisane	Suguvõimetu isane
	5	Kohiemane	Suguvõimetu em…
▶*	NULL	NULL	NULL

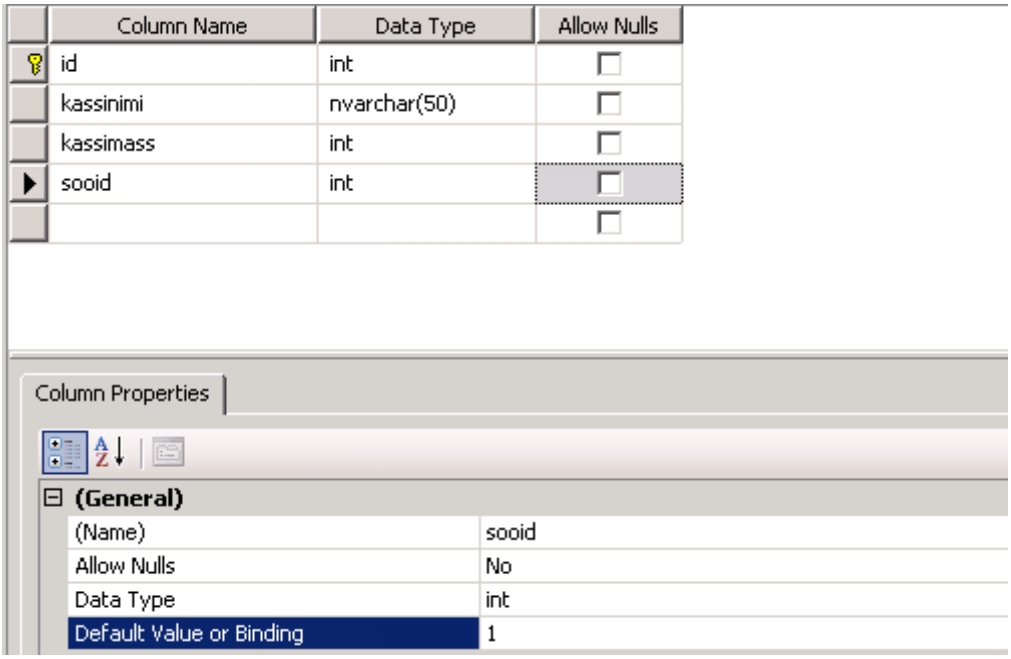
Ning edasi mõned andmed sisse.

Esialgu pole kassid tabelis kuidagi sootunnustega seotud. Viimased on lihtsalt eraldiseisvalt kirjeldatud, aga neid ei kasutata kusagil. Tüüpiliseks tabelite sidumise mooduseks on põhitabelisse koodiga tulba lisamine. Ehk siis praegusel juhul sobib kasside tabelisse kassi soo jaoks eraldi tulp.



Tabeli tulpade halduseks tuleb Database Exploreri juures parema hiireklahviga avada tabeli definitsioon ehk kirjeldus.

Sinna siis lisame kassi soo idendifitseerimiseks soo id ehk kokkukirjutatuna tulba nimega sooid. Kuna kassidel siiamaani veel sugu pole, id-tulpa pole aga viisakas tühjaks jätta, sel juhul tuleb siia panna algul vaikimisi väärtus. Nagu ülalt paistab, vastab koodile 1 kirjeldus „sugu teadmata“, mis võiks kõigile algul sobida.

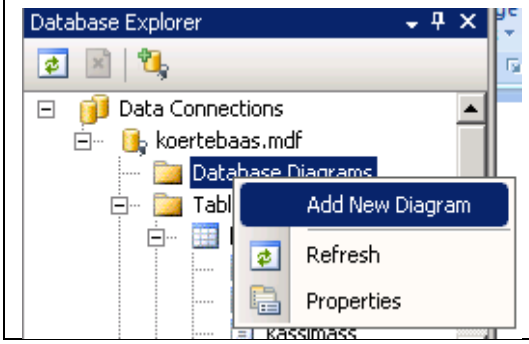


Et kui loodud tabelit vaadata, siis tekkis kassidele juurde tulp nimega sooid ja kõigil on sealseks koodiks 1, mis siis sugude tabelist järgi vaadatuna annab teada, et nende kasside sugu on veel teadmata.

	id	kassinimi	kassimass	sooid
▶	1	Miisu	2300	1
	3	Nimetu armas ka…	235	1
	4	Liisu	2000	1
	5	Ants	3750	1
*	NULL	NULL	NULL	NULL

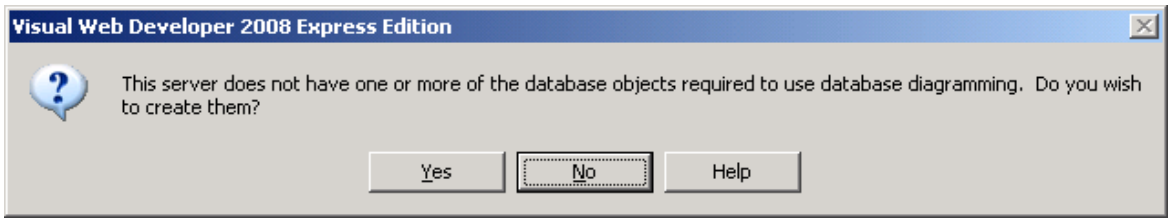
Nüüd on juba arv kasside tabeli juures küljes, aga seda teame ainult meie oma peas. Iseenesest sellest programmi loomiseks piisab, aga kindlam on, kui arvuti kontrollib, et me kogemata vigaseid koode sisse ei kirjuta. Koodile 1 vastab tüüp „Teadmata“ ning sellega on igati korras. Kui aga keegi kirjutaks kogemata soo id-ks mõne tunduvalt suurema arvu, siis ei osataks sellele kuidagi seletust anda.

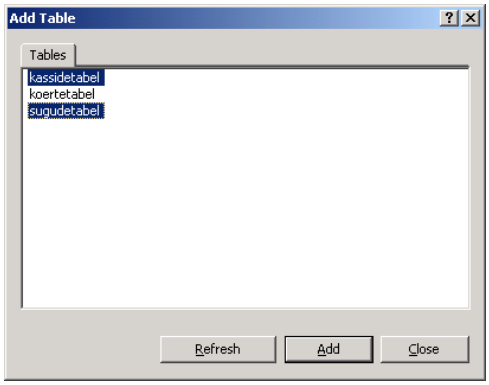
Üks mugav moodus seoste kirjapanekuks on diagrammide kaudu.



Kõigepealt parema hiireklahviga andmebaasihalduri juurest teade, et soovime uut diagrammi.

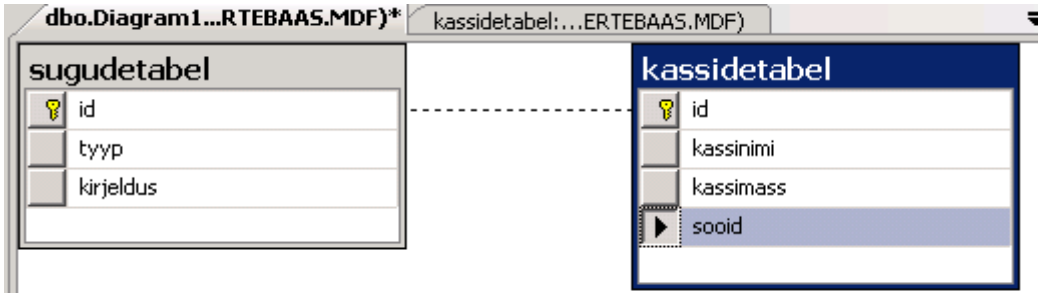
Arvuti igaks juhuks küsib üle, et kas ikka soovin siia veebirakenduse juurde omale diagramme luua. Edasiliikumiseks tuleb vastata jaatavalt.



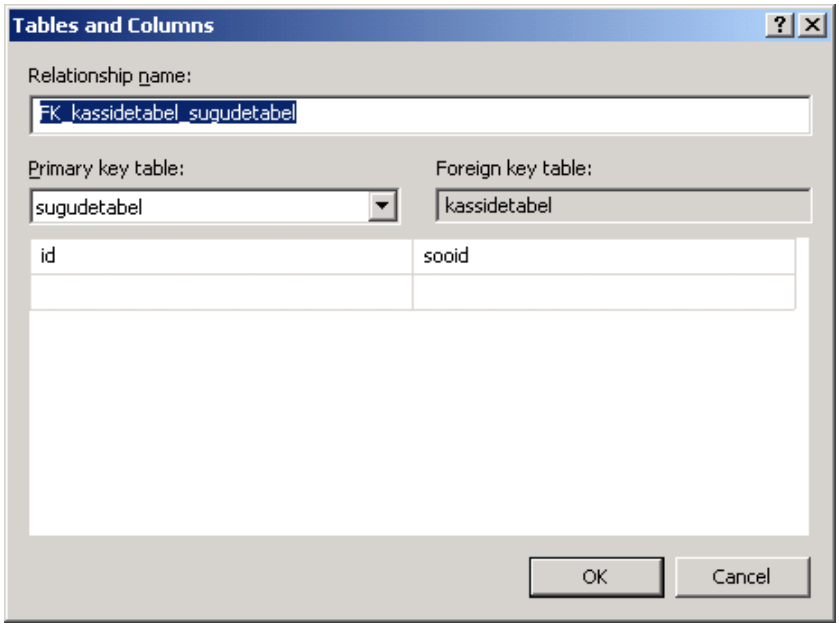


Siis lisan loetellu tabelid, mida siduda tahan. Ehk siis kasside ja sugude tabeli.

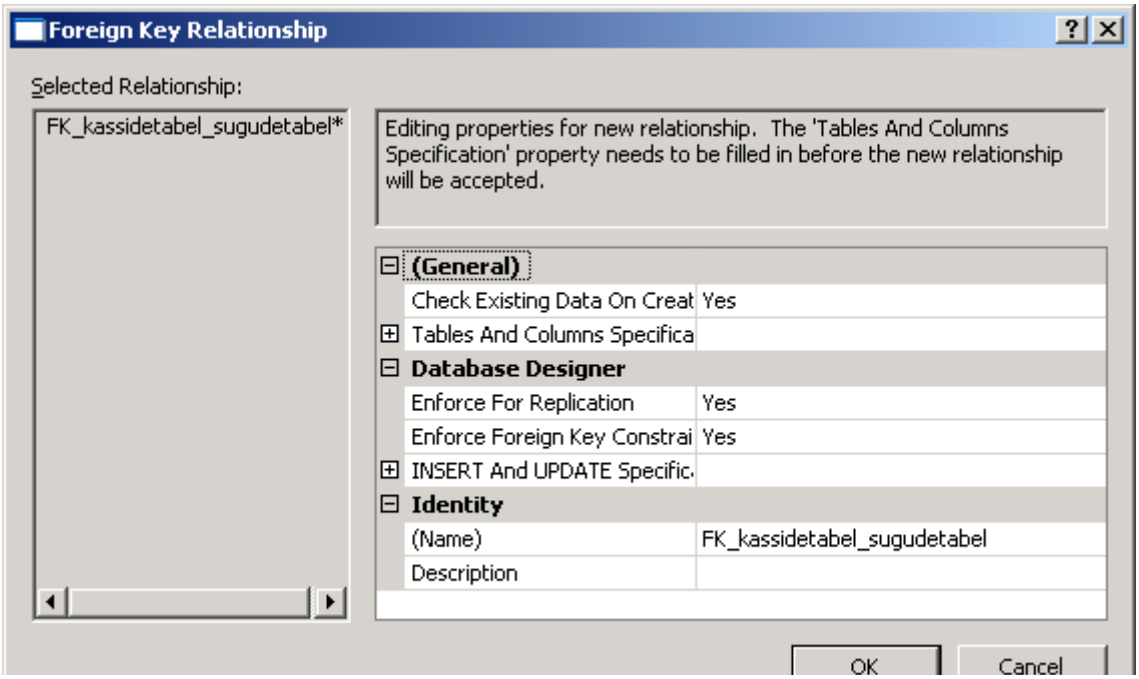
Seose loomiseks lohistan kassidetabeli tulba sooid sugudetabeli tulba id peale.



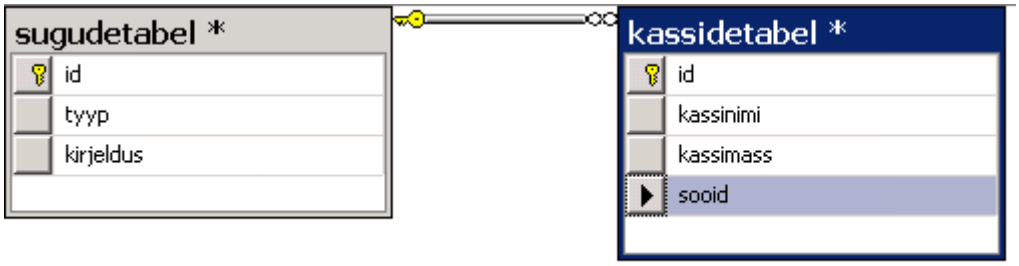
Selle peale küsitakse järgi, et kas kindlasti tahan nõnda, et kasside tabeli tulp sooid näitaks sugude tabeli tulba id peale. Ehk siis nõnda, et sugudetabeli tulp id on selles seoses primaarvõtmeks kuhu näidatakse. Ning kassidetabeli tulp sooid on selles seoses võõrvõtmeks (foreign key), mis näitab. Ning mille puhul siis kontrollitakse, et sooid väärtusteks saaksid olla vaid sellised arvud, mis ka sugudetabeli id-tulbas olemas.



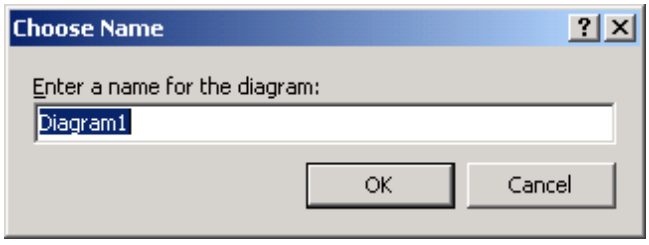
Edasi tuleb hulk seadistusi selle kohta, mida täpselt kontrollitakse. Aga esimeses lähenduses võib need muutmata jätta.



Kui nüüd seos märgitud, tekib joonisele võtme ja lõpmatusemärgiga seosejoon. Lõpmatusemärk kasside poolel tähendab, et igast soost võib olla piiramatu arv kasse (see tähendab 0, 1 või rohkem kassi). Iga kassi juures märgitud soonumber peab aga sugudetabelis olemas olema. Ametlikult nimetatakse sellist „üks mitmele“. Ehk siis ühele reale sugude tabelist võib vastata suvaline hulk ridu tabelis, kus selle soo peale näidatakse.

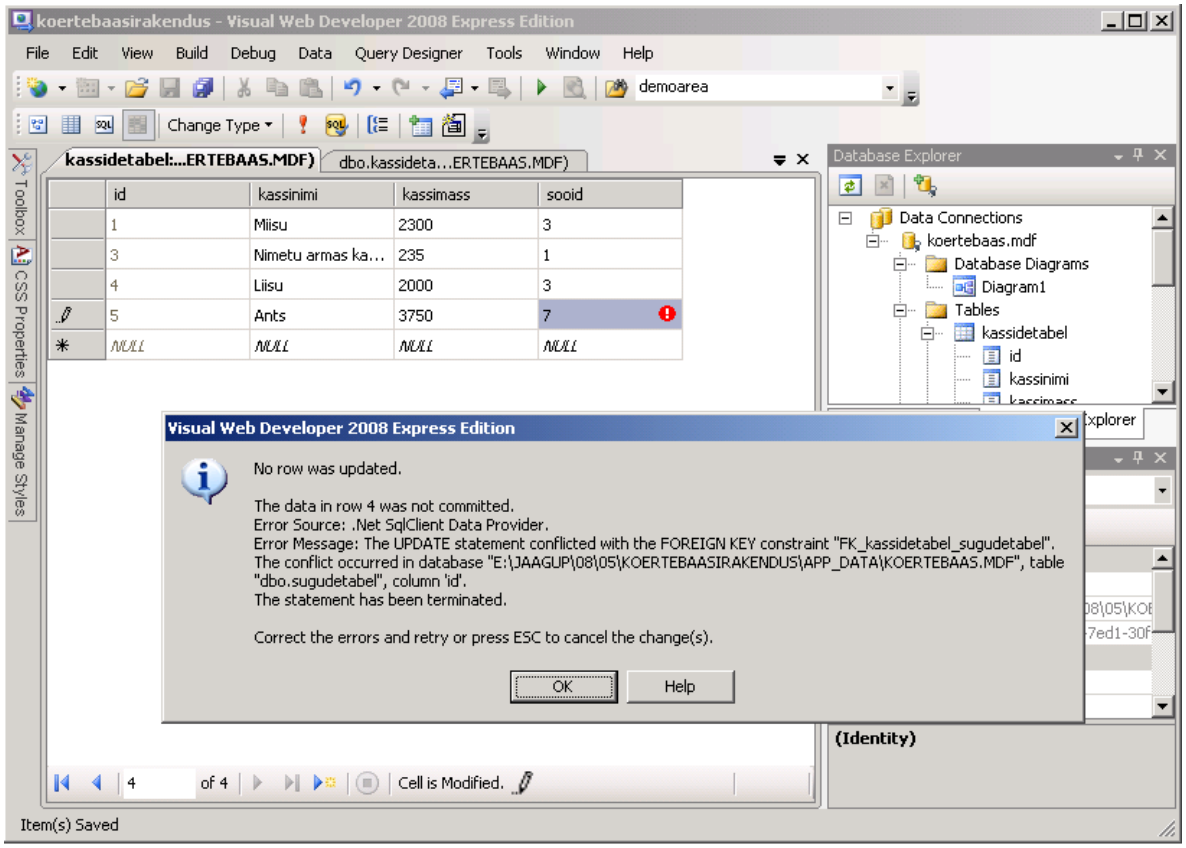


Salvestamise juures pakutakse diagrammile nime. See võib ju sellisena jääda.



Nüüd tasub asuda kasside sugude määramise juurde. Ehk siis sobiv arv sooid tulpa kirjutada. Praegu peab kirjutaja arvudega piirduma, hiljem kasutajaliidese juures on toimetus võimalik mugavamaks teha, et ka kohe näha on, kes millisest soost. Aga kui tabeli algust meenutada, siis 1 tähendas teadmata sugu, 2 isast ja 3 emast. Kassipoeg esiotsa teadmata, Miisu ja Liisu

võiksid emased olla. Põnevuse mõttes paneme Antsule vigase numbri ja vaatame, mis juhtub. Antakse viisakas veateade – väärtus 7 sugude tabelist puudu ning seega ei tohi sellist numbrit ka Antsu soo koodiks kirjutada.



Kui aga Ants määrata number kahe alla ehk isaseks, siis võeti kõik ilusti vastu.

	id	kassinimi	kassimass	sooid
	1	Miisu	2300	3
	3	Nimetu armas ka...	235	1
	4	Liisu	2000	3
	5	Ants	3750	2
▶*	NULL	NULL	NULL	NULL

Ülesandeid

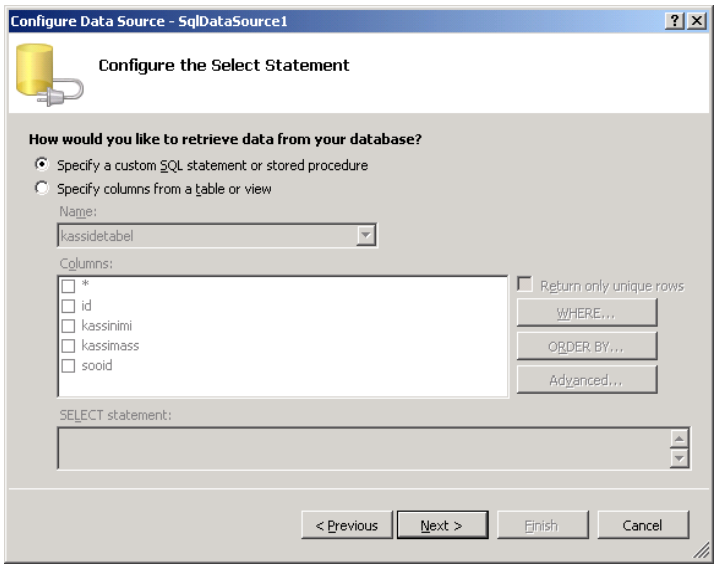
Koosta tabel linnade andmetega (id, linnanimi, rahvaarv), lisa mõned andmed.

Koosta inimeste tabel (eesnimi, telefon, linnakood). Määra linnakood võõrvõtmeks näitama linnade tabeli id-tulbale.

Sisesta andmeid inimeste tabelisse. Veendu, et vigast linnakoodi ei lubata sisestada.

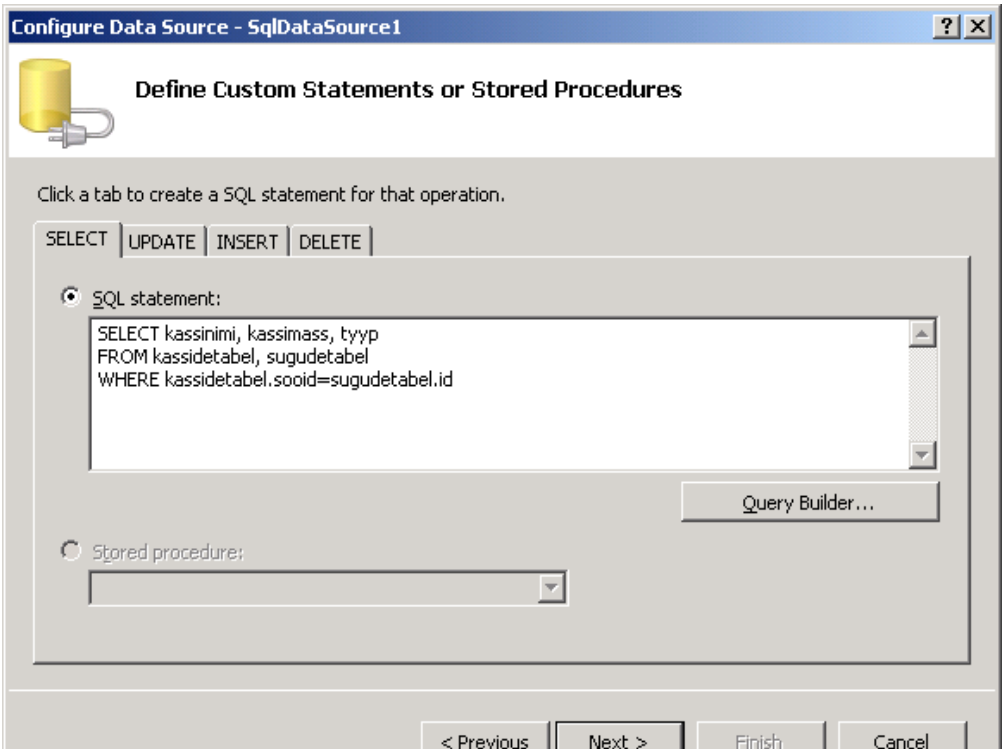
Andmete vaatamine

Ühest tabelist andmete nägemiseks piisas paarist hiireklõpsust andmeallika tegemisel. Mitme seotud tabeli puhul on ka põhimõtteliselt võimalik andmed hiire ja võluri abil nähtavale saada, aga otse SQL-päringu kirjutamine on tõenäoliselt lihtsam.

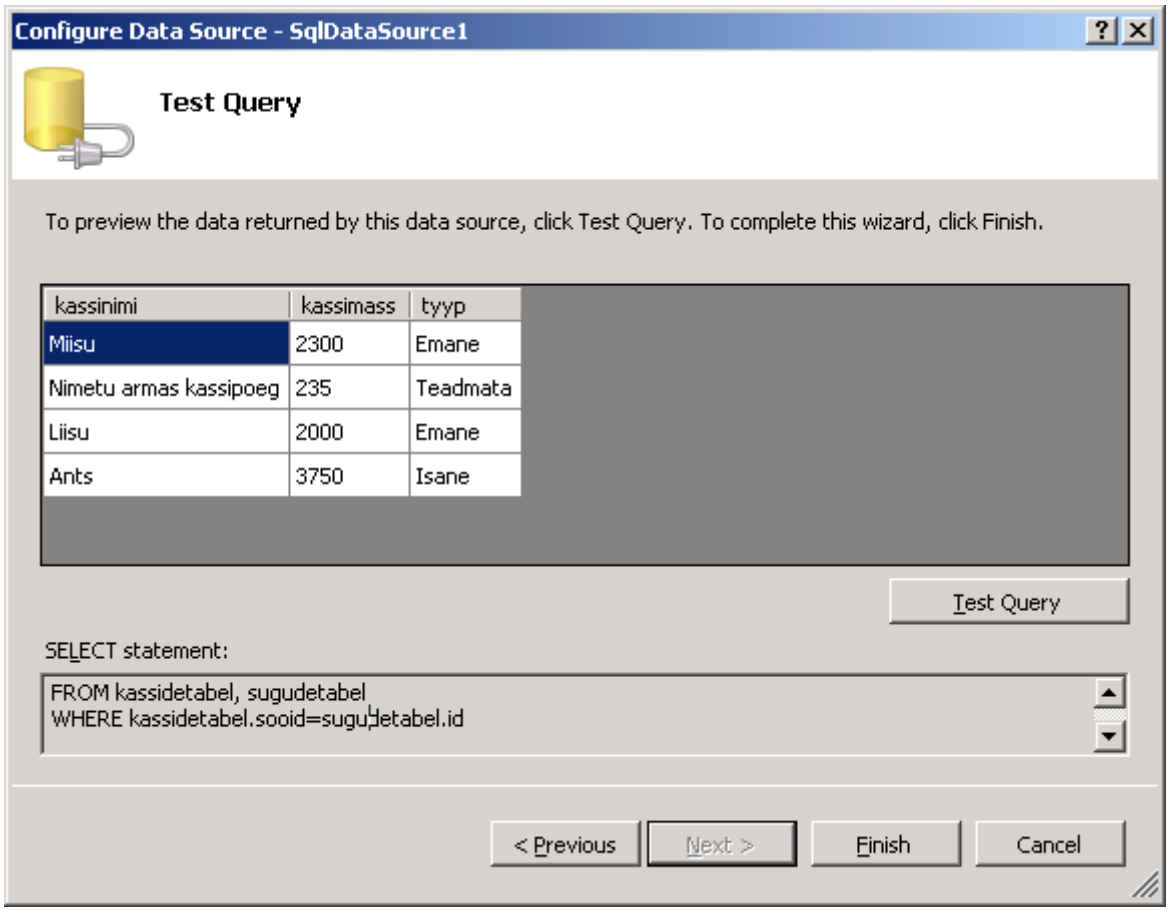


Põhiliseks konksuks päringu juures on, et kasutajale oleks viisakas näidata mitte kassi soo koodi, vaid sugu sõnana. Selleks tuleb aga teisest tabelist koodile vastav sõna leida. Õnneks on SQL-keele juures sellistele olukordadele mõeldud ning lause täiesti kirjutatav. Kõigepealt SELECT-sõna järel nende tulpade loetelu, mida soovime näha. Ehk siis kassinimi, kassimass ja tyyp. Kui eri tabelites on tulbad eraldi nimedega, siis pole tabeli nime lisamine kohustuslik. Muidu saaks kirjutada ka kujul kassidetabel.kassinimi.

FROM-ossa tuleb päringus kasutatavate tabelite loetelu. Ehk siis praegu kassidetabel ja sugudetabel. Ning WHERE-osas tuleb määrata, et millised tulbad kokku käivad. Ehk siis kassidetabeli tulp sooid ning sugudetabeli tulp id peavad võrdsete väärtustega olema, et sellist kahest tabelist tulnud andmete kombinatsiooni oleks põhjust näidata.



Igasugu päringud tasub enne kasutusse laskmist järele kontrollida. Ehk siis järgnevalt on kasulik testimisnuppu vajutada. Nagu näha, tuli sugu sõnana ilusti vastava kassi andmete taha.

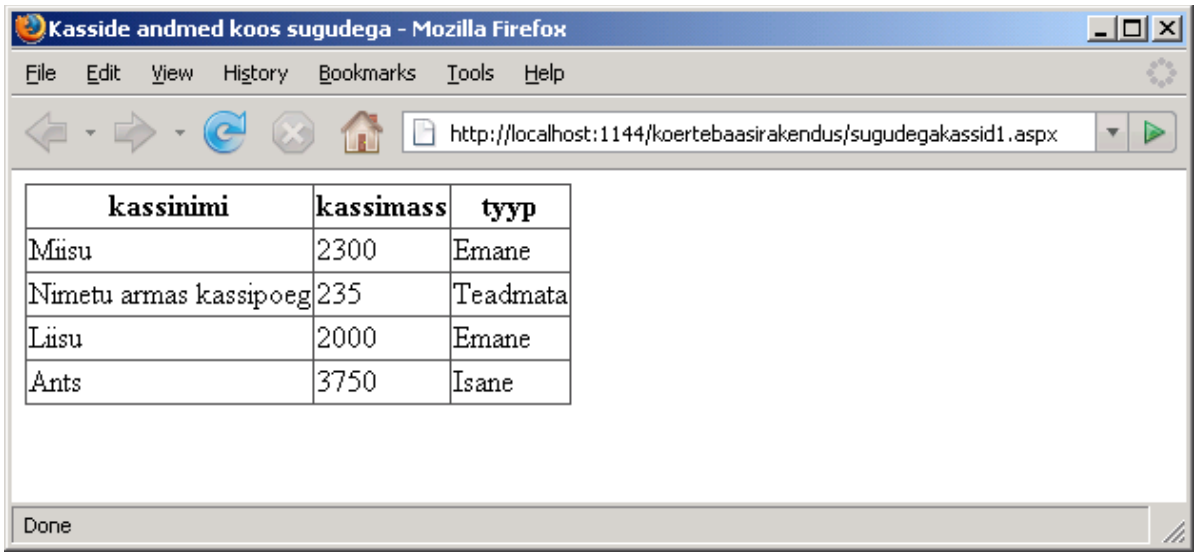


Veebilehe lähtekoodi piiludes paistab, et kõik eelnev on ilusti ka sinna kirja pandud. Ühendustekst võetakse sarnaselt nagu eelnevate näidete puhul. Ning juures vaid SelectCommand, ehk eelnevalt loodud lause, millega andmed tabelist kätte saab.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT kassinimi, kassimass, tyyp
FROM kassidetabel, sugudetabel
WHERE kassidetabel.sooid=sugudetabel.id"></asp:SqlDataSource>
```

Juurde veel GridView andmete näitamiseks ning seal määrata, millise id-ga andmeallikast andmed võetakse ning võibki loetelu veebilehel imetleda.

```
    <asp:GridView ID="GridView1" runat="server"
DataSourceID="SqlDataSource1">
    </asp:GridView>
```



kassinimi	kassimass	tyyp
Miisu	2300	Emane
Nimetu armas kassipoeg	235	Teadmata
Liisu	2000	Emane
Ants	3750	Isane

Lähtekood siis tervikuna järgmine

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Kasside andmed koos sugudega</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT kassinimi, kassimass, tyyp
FROM kassidetabel, sugudetabel
```

```
WHERE kassidetabel.sooid=sugudetabel.id"></asp:SqlDataSource>

    <asp:GridView ID="GridView1" runat="server"
                  DataSourceID="SqlDataSource1">
    </asp:GridView>

    </div>
    </form>
</body>
</html>
```

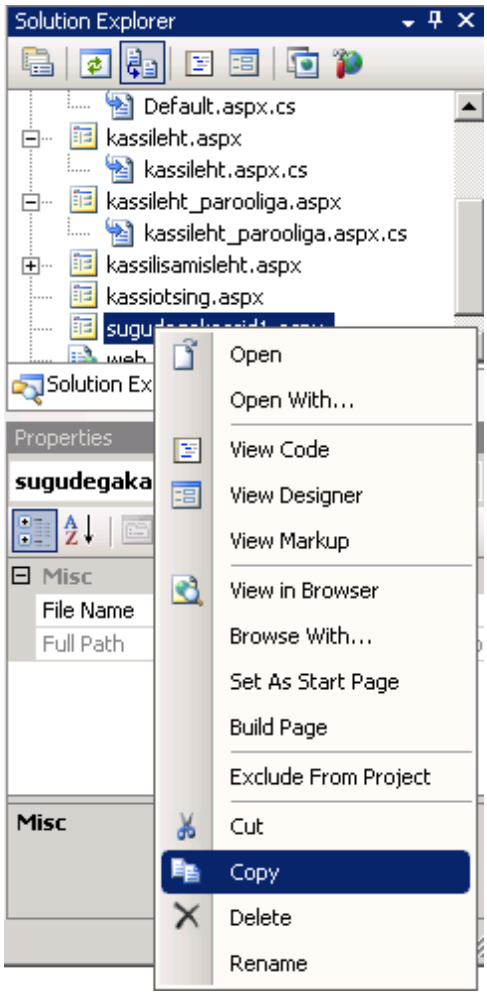
Sellega siis mitmesse tabelisse seotud andmete vaatamise osa õnnelikult valmis.

Ülesandeid

- Näita sarnaselt veebi eelnevalt loodud inimeste tabeli ja linnade tabeli andmed. Nõnda, et iga inimese juures oli kirjas tema linna kood. Veebis on aga inimese nime kõrval tema linna nimi.

- Lisa päringu WHERE ossa juurde tingimus, et näidataks vaid neid inimesi, kes elavad linnades, mille rahvaarv on väiksem kui 100000 inimest (kahe tingimuse vahele sõna AND).

Andmete lisamine



Mis töötab, seda ära puutu. Ehk siis olemasolevat targemaks ehitama hakates on mõistlik lehest koopia teha. Nagu ikka – Copy-käsuga mällu ning ülevalt rakenduse nime juurest Pastega tagasi. Ja arusaadav nimi ka juurde – siin näiteks sugudegakassid2.aspx.

Siinse rakenduse puhul on lisamise juures konksuks, et tegelikult tahame andmeid saada vaid kasside tabelisse. Aga kassi soo kohta peaks sinna minema kood, samas kui inimesel oleks mugavam valida soo nimetus. Nüüd tuleb tähelepanelikult eelmisest lisamisrakendusest sobivad kohad maha kirjutada, õigetesse kohtadesse aga muudatused teha ning märgatav osa esialgu sootuks tähelepanuta jätta.

Andmete lisamiseks vaja andmeallikasse juurde kirjutada INSERT-lause lisatavate tulpade jaoks. Kassi id-d ei lisata, see tuleb automaatselt. Küll aga on kassi soo jaoks olemas tulp. Andmeallika kaudu lisades pannakse lisatavate väärtuste kohale parameetrid. Automaatselt genereerides tuleksid need tulpadega samade nimedega. Siin aga meelega paneme erinevad, et näeks, kus tegu tulbaga, kus parameetriga.

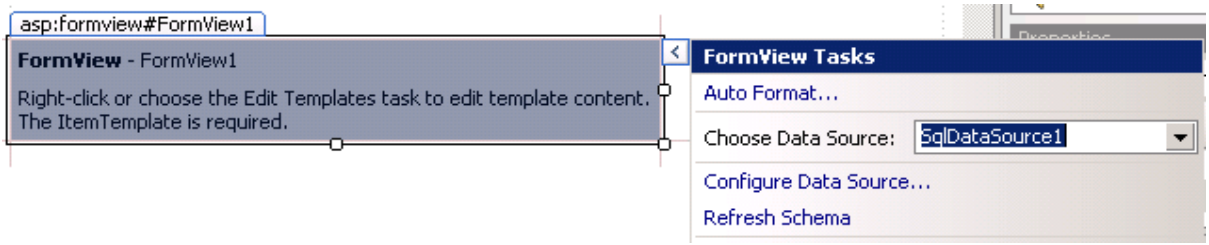
```
InsertCommand="INSERT INTO kassidetabel (kassinimi, kassimass, sooid)
  VALUES (@prkassinimi, @prkassimass, @prkassisugu)"
```

Andmeallika kood siis tervikuna:

```
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT kassinimi, kassimass, tyyp
FROM kassidetabel, sugudetabel
WHERE kassidetabel.sooid=sugudetabel.id"
InsertCommand="INSERT INTO kassidetabel (kassinimi, kassimass, sooid)
  VALUES (@prkassinimi, @prkassimass, @prkassisugu)"
>
```

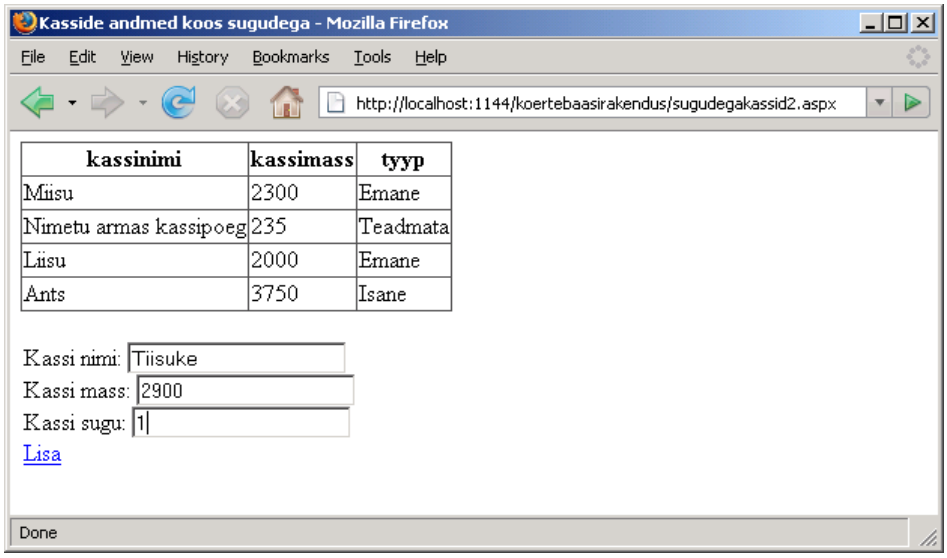
```
  <InsertParameters>
      <asp:Parameter Name="prkassinimi" Type="String" />
      <asp:Parameter Name="prkassimass" Type="Int32" />
      <asp:Parameter Name="prkassisugu" Type="Int32" />
  </InsertParameters>
</asp:SqlDataSource>
```

Andmete lisamiseks sobib tuttav FormView – seal võimalik ise edasi määrata, kuidas mida kujutatakse. Veebilehele lohistatuna saab valida küll andmeallika, kuid muu kujundus tuleb koodis ise kirjutada.

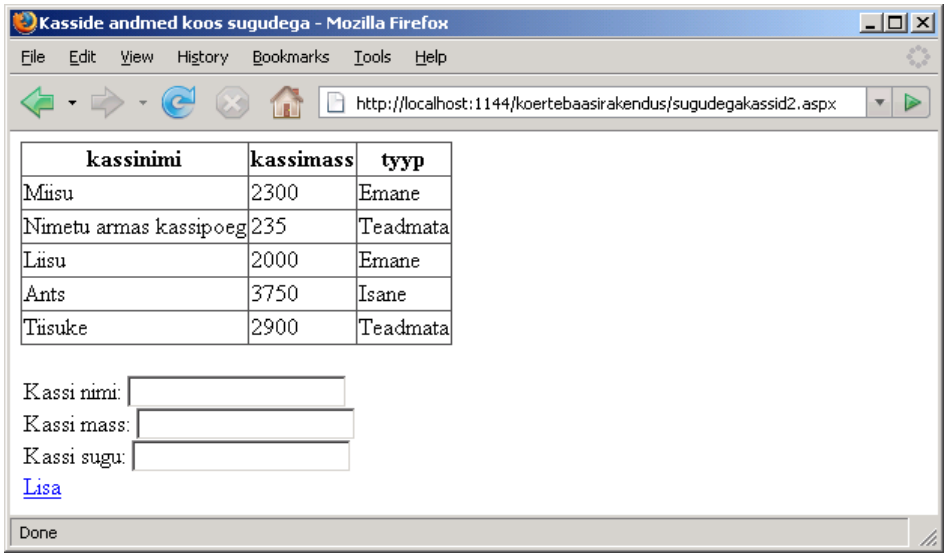


Kui vaikimisi moodiks on „Insert“, siis lisamiselemendiks piisab vaid InsertItemTemplatest. Esialgu märgime vaid tekstikastidega andmete sisestamise kohad. Ning CommandFieldi kaudu nupp, kust sisestamine käivitada.

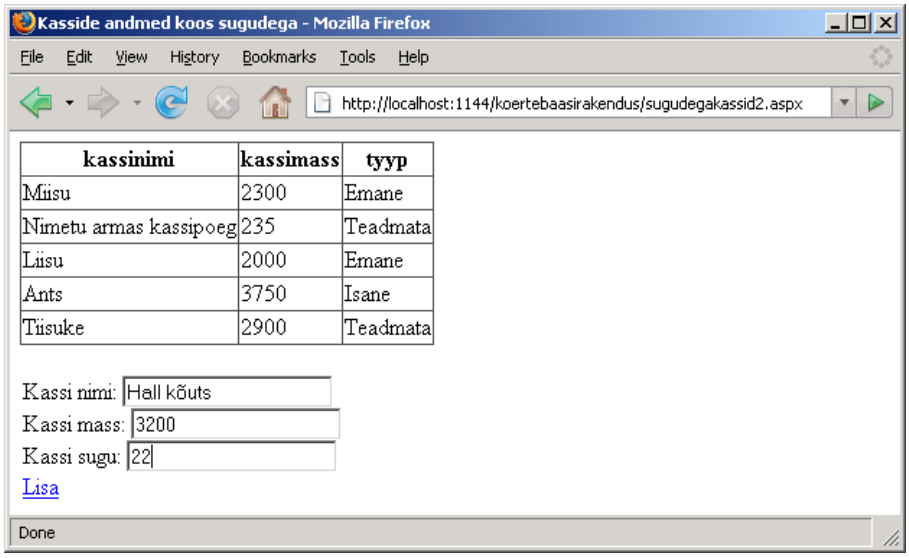
```
<asp:FormView ID="FormView1" runat="server"
    DataSourceID="SqlDataSource1" DefaultMode="Insert">
    <InsertItemTemplate>
        Kassi nimi:
        <asp:TextBox ID="knimi" runat="server"
             Text='<%# Bind("prkassinimi") %>' /><br />
        Kassi mass:
        <asp:TextBox ID="kmass" runat="server"
             Text='<%# Bind("prkassimass") %>' /><br />
        Kassi sugu:
        <asp:TextBox ID="ksugu" runat="server"
             Text='<%# Bind("prkassisugu") %>' /><br />
        <asp:LinkButton ID="InsertButton" runat="server"
            CausesValidation="True" CommandName="Insert"
            Text="Lisa" />
    </InsertItemTemplate>
</asp:FormView>
```



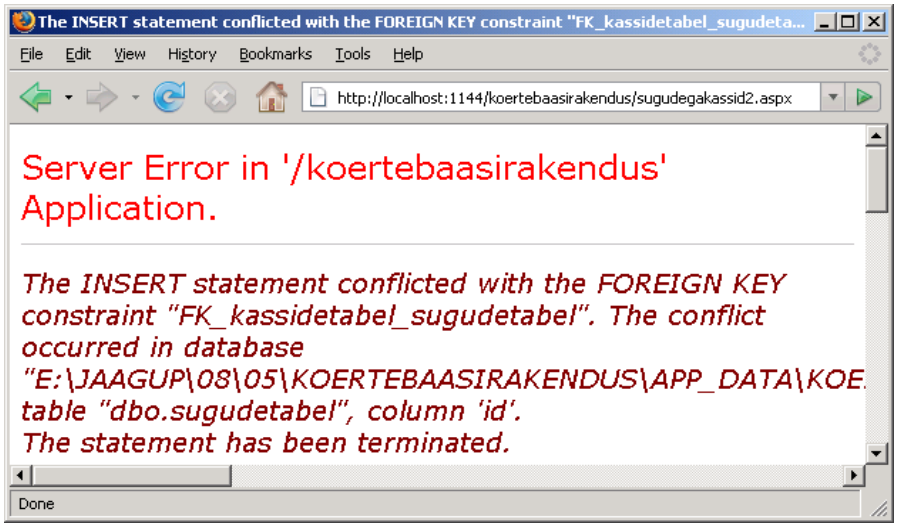
Edasi võibki veebist andmeid sisse kirjutada. Esialgu tuleb soo kood küll veel arvuna määrata.



Nagu näha, koodile 1 vastas soo tüüp „teadmata“ ja andmed läksid kirja.

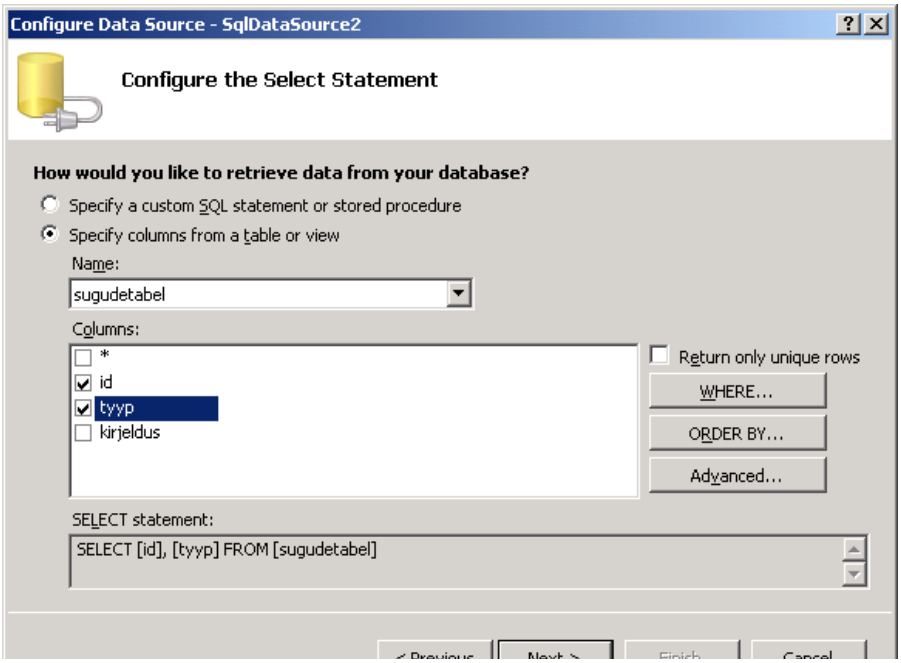


Tekstivälja sisestust aga kohe ei kontrollita, nii ei sega miski sinna esialgu suvalisi suuri arve kirjutamast.



Kuna aga andmebaasi siseselt kontrollitakse võõrvõtit, siis vigased andmed siiski tabelisse ei lähe, salvestamisel antakse veateade. Esmane sisestusmoodus nõnda olemas – kuigi selleks polnuks veel vaja oma FormView sisu kirjutada – sarnase loo olime juba eelnevalt tööle saanud.

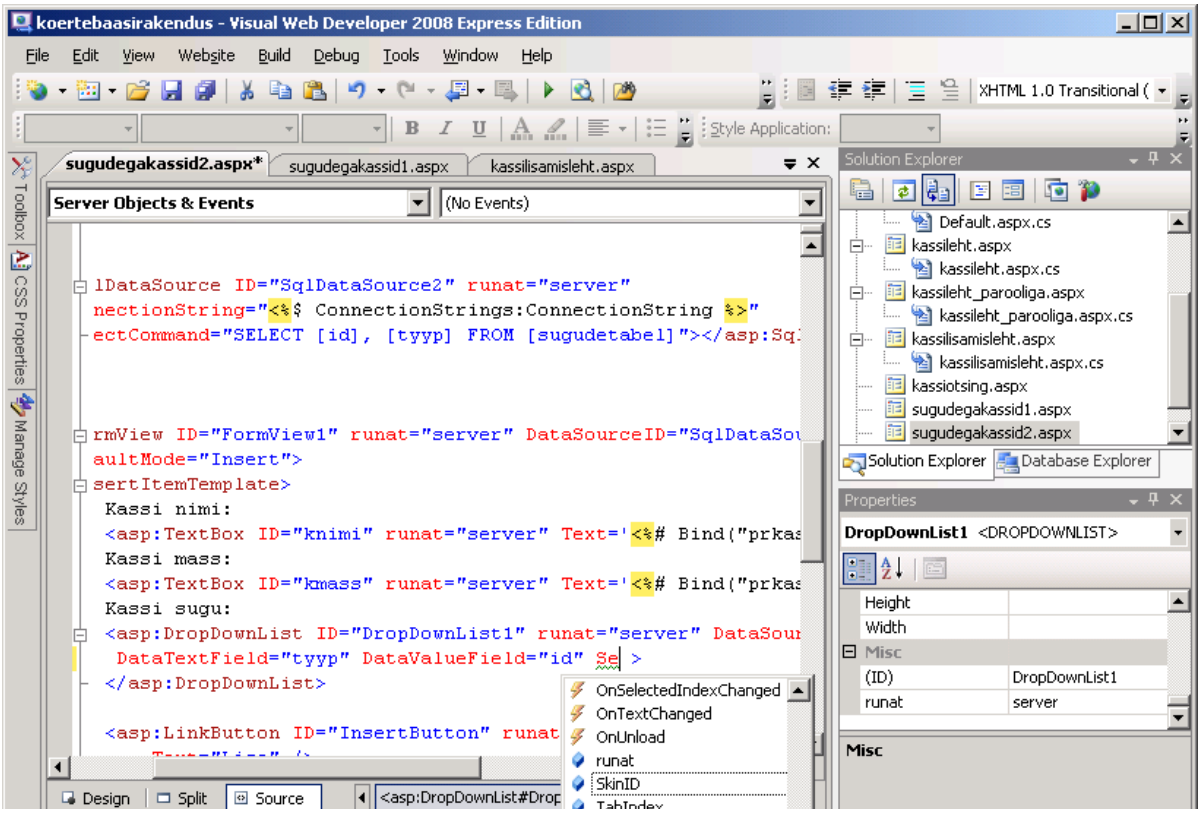
Kasutajale tunduvalt mugavam on aga valida sugu juba olemasolevate valikute hulgast. Selleks tasub andmeallika kaudu küsida tabelist sugude andmed ning neid hiljem sobiva elemendi abil näidata. Tabelist küsime välja id ja tüübi väärtused. Teise näitamiseks ning esimese sisestustabelisse kirjutamiseks.



Valiku peale genereeritud andmeallika kood näeb välja järgmine:

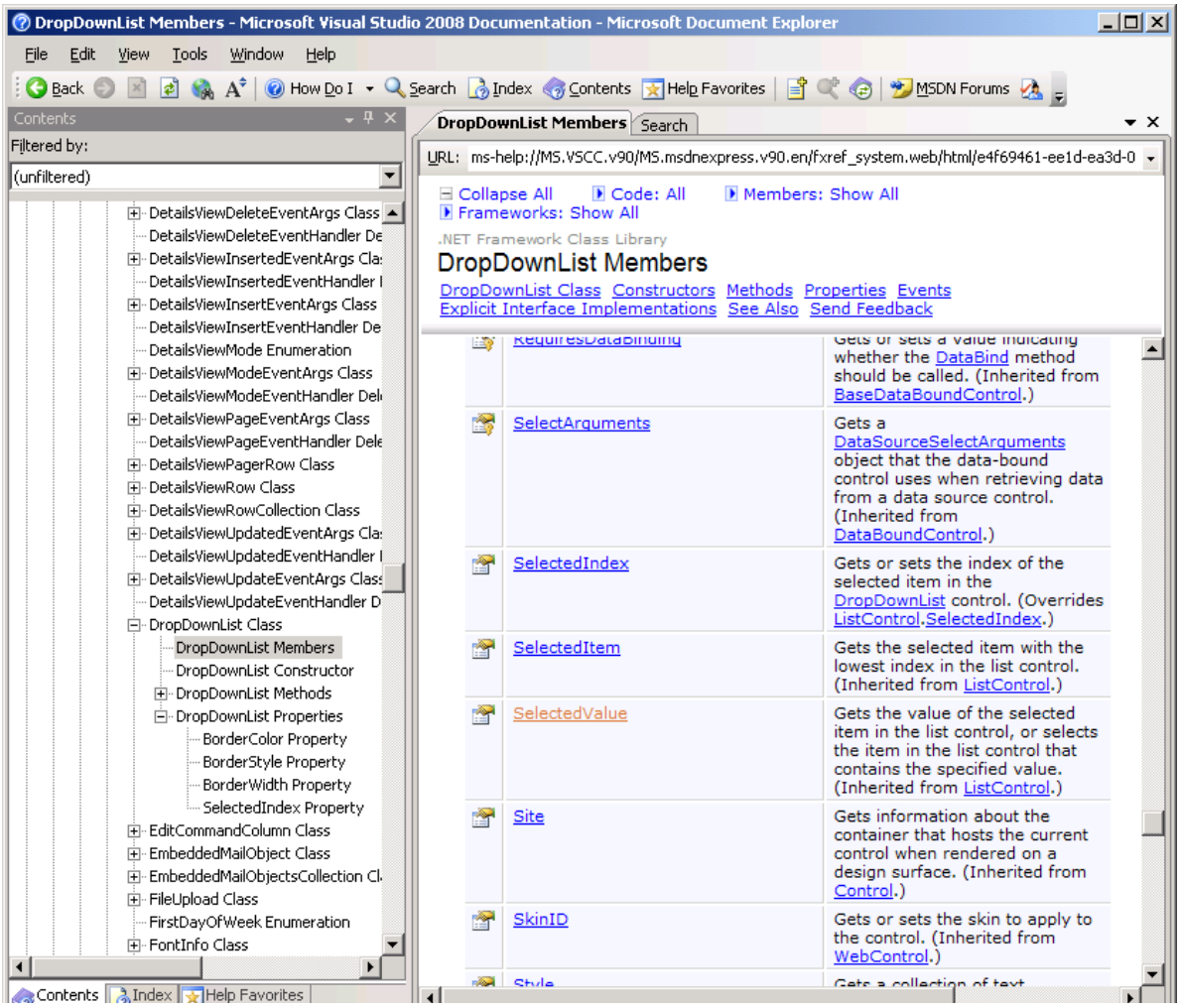
```
        <asp:SqlDataSource ID="SqlDataSource2" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
            SelectCommand="SELECT [id], [tyyp] FROM
[sugudetabel]"></asp:SqlDataSource>
```

Loetelus olevate andmete näitamiseks on mitmesuguseid elemente. Siin kasutame DropDownListi, aga mõnevõrra sarnased on ListBox, RadioButtonList, samuti CheckBoxList ja ka BulletedList. Et kui leht koos, võib mõnikord proovida kujundust muuta elementi asendades. Rippmenüüle märgime andmeallikaks SqlDataSource2. Näidatavate sõnade tulbaks (DataTextField) sobib tyyp ning tabelisse saadetava koodi tulbaks (DataValueField) id. Veidi raskem on selgeks teha, millise rippmenüü omaduse väärtus Bindiga andmeallika külge saadetakse ja sealtkaudu lisamisel tabelisse juhatakse. Loogiline tunduks, et vastav väärtus võiks rippmenüüst kuidagi kergesti kättesaadav olla nagu mõnes teises kohas ja keskkonnas. Kui aga pakutavaid omadusi keskkonna IntelliSense abil vaadata, siis ühtki Selected... omadust ei paista olema ja muud ka ei vihja sellisele võimalusele.



Sellegipoolest ei tasu meelt heita. Sarnaste graafiliste koodiarenduskeskkondade puhul on küllalt sageli tavaks, et keegi kurdab, et „naabril läks roheliseks aga mul ei läinud“. Mis ei tähenda, et kummalgi midagi viga peaks olema. Visual Studio on mitme teise sarnase arenduskeskkonnaga võrreldes küll suhteliselt töö- ja lollikindel, aga ega kõike ei suuda temagi. Kindlamat teavet omaduste kohta saab ametlikust abiinfost. Olgu see siis kohapeale installeeritud või veebi kaudu vaadatav. Help -> Contents -> .NET Framework SDK -> .NET Framework Class Library ning System.Web.UI.WebControls alt leiab DropDownListi (vajadusel leiab selle ka otsingu kaudu). Seal peaks ametlikult kõik vastavad omadused kirjas olema.

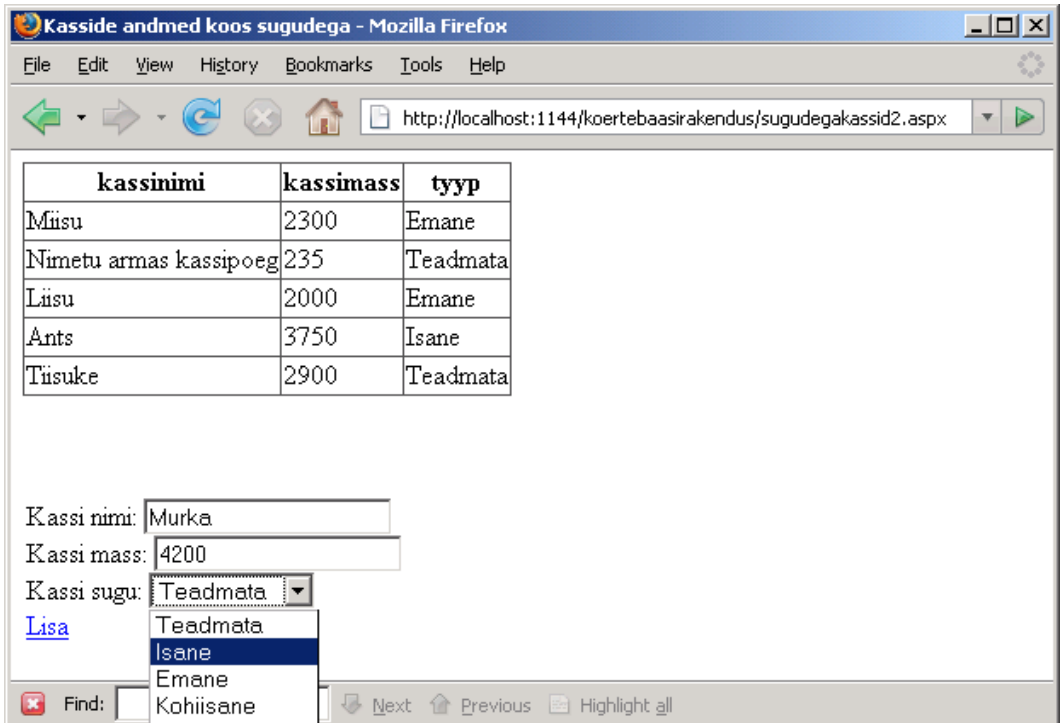
Veidi otsimist ning paistabki omadus nimega SelectedValue, mis just meil tarvilik on, et andmed tabelisse saata.



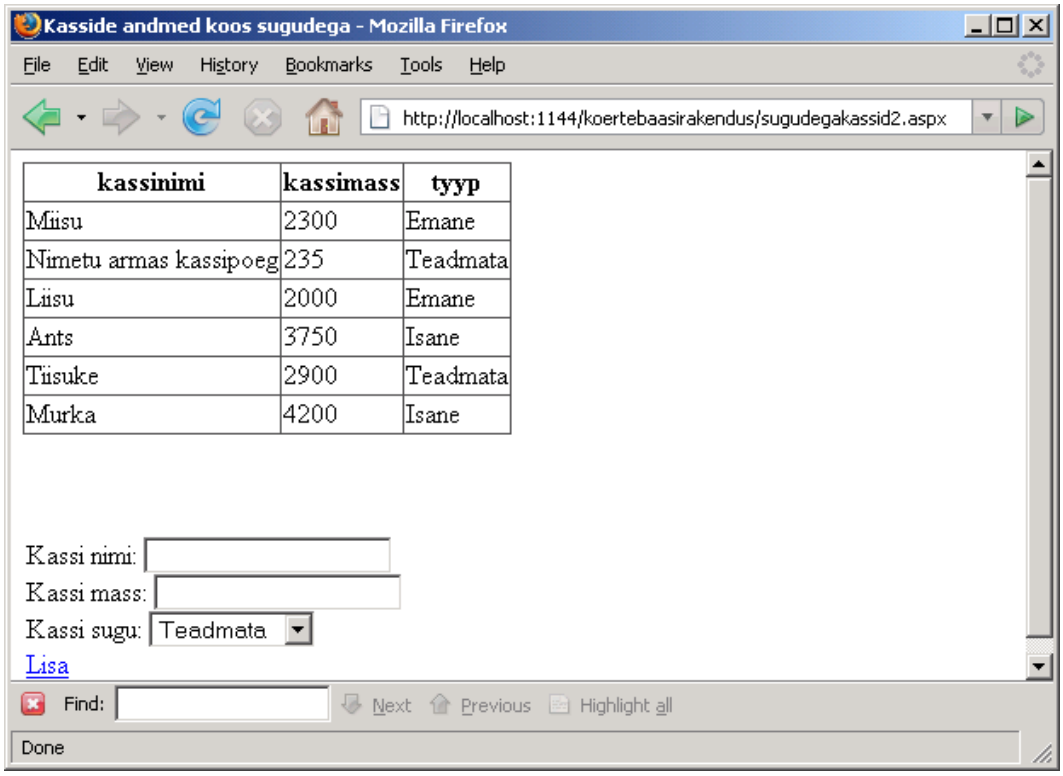
Tolle SelectedValue saab siis Bindiga andmeallika (SqlDataSource1 ehk kasside tabeliga suhtleja) külge siduda. Rippmenüüst valitud väärtus läheb siis sisestusparameetri `prkassisugu` kohale.

```
Kassi sugu:
<asp:DropDownList ID="DropDownList1" runat="server"
     DataSourceID="SqlDataSource2"
 DataTextField="tyyp" DataValueField="id"
     SelectedValue='<%# Bind("prkassisugu") %>' >
</asp:DropDownList>
```

Ja võibki veebist rippmenüü kaudu andmeid lisada



Ning rõõmustada, et need ka kohale jõuavad.



Ülesanne

- Loo eelnevalt koostatud linnade ja inimeste tabeli juurde ka sarnane lisamisvõimalus, kus inimese lisamisel saab tema sünnilinna valida rippmenüüst.

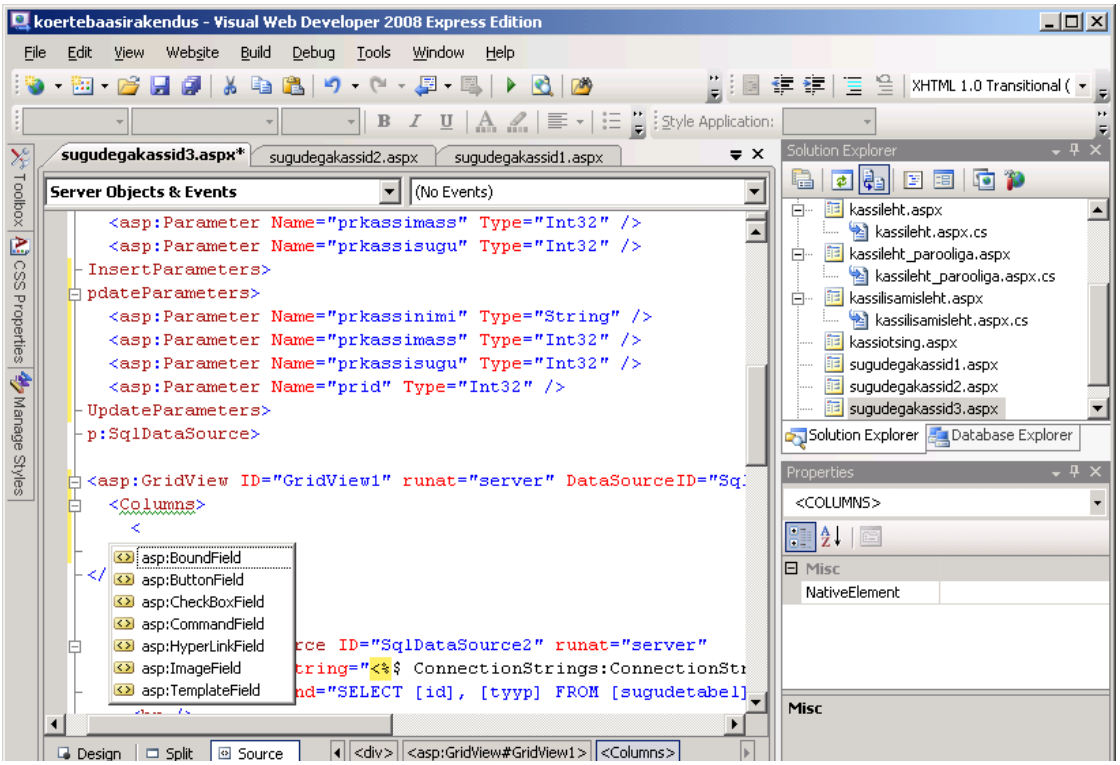
Andmete muutmine

Uute kasside lisamisega saame nüüd hakkama ning sugu õnnestub ka sinna kõrvale valida. Kui aga peaks olema midagi valesti läinud, siis esialgu ei aita muud, kui „käsitsi“ vastav rida andmebaasis kustutada või parandada. Aga nagu mujal, nii ka mitme tabeli puhul saab selle mugavamaks teha. Vaja vaid veidi nuputamist sinna juurde.

Kui andmete näitamise GridViewle lihtsalt muutmisnupp külge panna, siis kurdetakse, et me andmeallikas ei oska andmeid muuta. Ning kui lihtsalt veel muutmislause (UPDATE) lisada, siis saab muuta küll, aga jällegi tuleks asuda soo koode ise arvuna kirjutama. Võimalik, aga tülikas. Ehk saab paremini. Olemasolevast töötavast failist taas koopia ning võib toimetama asuda.

Seega tuleb mõlemat komponenti veidi ümber teha. FormView juures andmete sisestamisel kirjutasime nagunii terve HTMLi eraldi valmis. GridView puhul aga piirdusime seni vaid võimalusega, et näidati välja kõik andmed, mis andmeallikast tulid ning sellistena nagu nad tulid. Kui aga soovime näha sugu sõnana samas muutes tema koodi, siis nii lihtsalt ei saa.

Et veerge automaatselt ei genereeritaks, selleks tuleb GridViewle anda omaduse `AutoGenerateColumns` väärtuseks `false`. Ja samas et mingeid tulpi ikka näha oleks, tuleb siis nood ükshaaval kirja panna. Nagu jooniselt näha, on väljade tüüpe seitse, igaühe nimi räägib iseenda eest. Lihtsaimal juhul andmeallika tulba ja tabeli tulba sidumiseks sobib BoundField. Tahtes aga tulpa panna veebiviidet, aitab `HyperLinkField` või pildi näitamiseks kõlbab `ImageField`. Lihtsaimaks sidumiseks aga peavad SELECT lausest tulevate tulpade nimed ning pärast UPDATE abil minevad muutusparameetrite nimed olema samade väärtustega. Seetõttu kui INSERT lause juures võisime parameetrimuutujatele rahus tähed pr ette kirjutada, siis siin ei saa. Joonisel oleval kujul tekib käivitamise juures muresid.



Kui aga SELECT´i tulbad ja UPDATE parameetrid panna samade nimedega, siis all GridViews võib iga vastava tulba kohta lihtsalt öelda, et see on BoundField ning saabki andmeid rahus veebi kaudu muuta. SELECT lauses on id- tulbale ette pandud tabeli nimi, sest vastava nimega tulp leidub nii kasside kui sugude tabelis. Et oleks võimalik sugu näha ja muuta, selleks on küsitud välja nii vaatamiseks `tyyp` kui ka muutmiseks `sooid`. Ja et kellelgi ei tuleks pähe asuda sõnana olevat sugu muutma (mida ta nagunii teha ei saa), siis tabeli juures on vastavale tulbale lisatud omadus `ReadOnly`.

UPDATE lause puhul siis korjatakse uued väärtused parameetrite juurest kokku ja omistatakse vastavatele tulpadele. Kusjuures piirajaks, millisele andmereale need väärtused omistada on rea id-number. Andmetabeli juures on tollele identifikaatorile antud omadus `Visible="false"` ehk siis pole tavakasutajale nähtav ja vajalik. Samas arvuti jaoks on hädavajalik id kaudu identifitseerida, millise reaga tegemist on.

```
UPDATE kassidetabel SET kassinimi=@kassinimi, kassimass=@kassimass,
sooid=@sooid WHERE id=@id
```

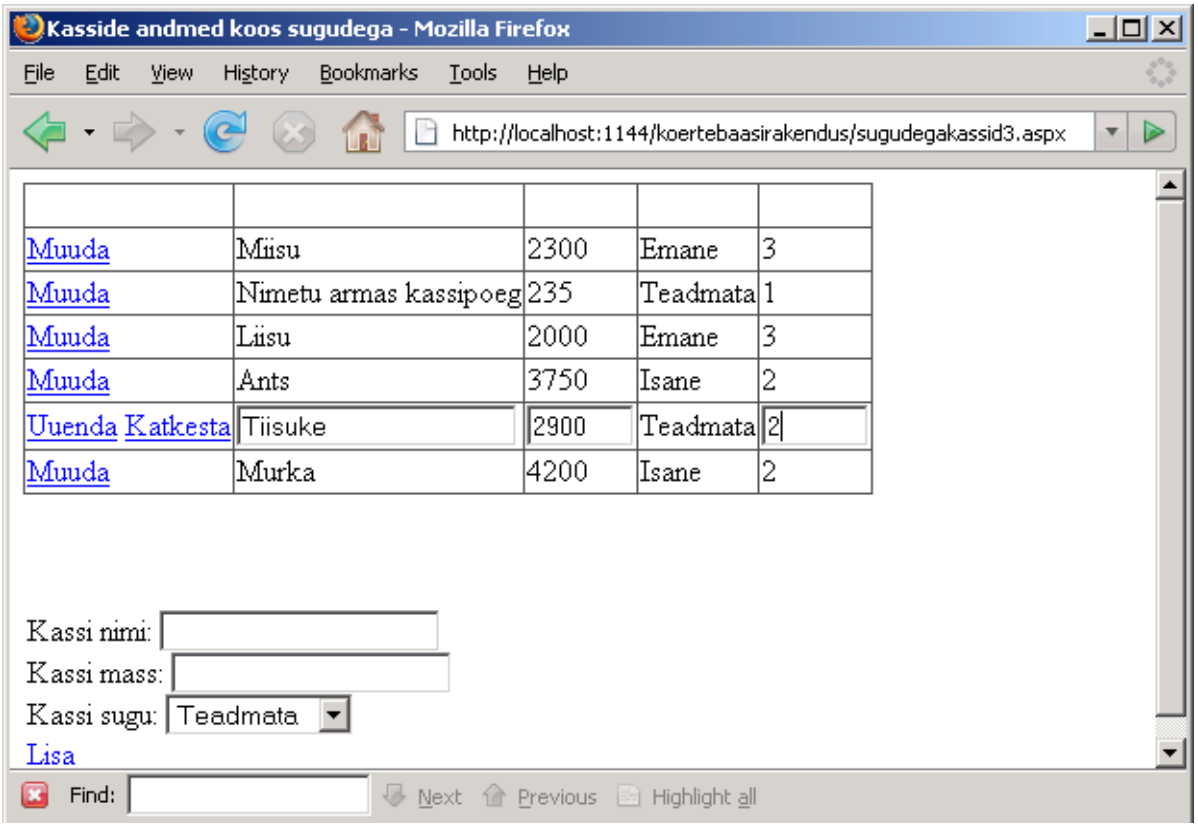
Ning liigutatavate väärtustega parameetrid tuleb siis andmeallika vastavas tsoonis kõik ükshaaval üles lugeda.

```
<UpdateParameters>
    <asp:Parameter Name="kassinimi" Type="String" />
    <asp:Parameter Name="kassimass" Type="Int32" />
    <asp:Parameter Name="sooid" Type="Int32" />
    <asp:Parameter Name="id" Type="Int32" />
</UpdateParameters>
```

Järgnevalt muutmisvõimelise veebilehe tekitav koodilõik tervikuna.

```
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT kassidetabel.id, kassinimi, kassimass, tyyp, sooid
FROM kassidetabel, sugudetabel
WHERE kassidetabel.sooid=sugudetabel.id"
InsertCommand="INSERT INTO kassidetabel (kassinimi, kassimass, sooid)
  VALUES (@prkassinimi, @prkassimass, @prkassisugu)"
UpdateCommand="UPDATE kassidetabel SET kassinimi=@kassinimi,
kassimass=@kassimass, sooid=@sooid WHERE id=@id"
>
  <InsertParameters>
      <asp:Parameter Name="prkassinimi" Type="String" />
      <asp:Parameter Name="prkassimass" Type="Int32" />
      <asp:Parameter Name="prkassisugu" Type="Int32" />
  </InsertParameters>
  <UpdateParameters>
      <asp:Parameter Name="kassinimi" Type="String" />
      <asp:Parameter Name="kassimass" Type="Int32" />
      <asp:Parameter Name="sooid" Type="Int32" />
      <asp:Parameter Name="id" Type="Int32" />
  </UpdateParameters>
</asp:SqlDataSource>

    <asp:GridView ID="GridView1" runat="server"
DataSourceID="SqlDataSource1" AutoGenerateColumns="false"
      DataKeyNames="id">
      <Columns>
        <asp:CommandField ShowEditButton="true" EditText="Muuda"
UpdateText="Uuenda" CancelText="Katkesta" />
        <asp:BoundField DataField="id" Visible="false" />
        <asp:BoundField DataField="kassinimi"   />
        <asp:BoundField DataField="kassimass" />
        <asp:BoundField DataField="tyyp" ReadOnly="true"/>
        <asp:BoundField DataField="sooid" />
      </Columns>
    </asp:GridView>
```

Joonise järgi näeb, et selline leht töötab. Vahetan soo koodi ära ühelt kahele



ning ongi näha, et Tiisuke on isane.

Muuda	Miisu	2300	Emane	3
Muuda	Nimetu armas kassipoeg	235	Teadmata	1
Muuda	Liisu	2000	Emane	3
Muuda	Ants	3750	Isane	2
Muuda	Tiisuke	2900	Isane	2
Muuda	Murka	4200	Isane	2

Samas selline numbrite kirjutamine on ikka tülikas. Kui sai andmete lisamise juures sellest lahti, saab ka siin. Ainult koodi jälle mõnevõrra juures.

Kui piisab automaatsest sidumisest, siis on BoundField igati kasulik – tekstiväljas saan uue väärtuse sisse kirjutada ning võin tulemusega rahul olla. Nüüd aga andmete vaatamisel piisab tavalisest tekstist soo nägemiseks, andmete muutmiseks aga oleks hea kasutada rippmenüüd. Sellise mitmekülgsema tulba loomiseks on välja mõeldud TemplateField. Selle sees tavaline ItemTemplate määrab, kuidas näeb lahter välja vaatamise ajal. EditItemTemplate aga hoolitseb kujunduse eest muutmisel ehk pärast vastava nupu vajutamist.

Lihtsa teksti näitamiseks sobib silt ehk komponent tüübist asp:Label. Bind-käsuga seome tulbaga tyyp ning võibki rahus sõnana vaadata, millisest soost kass on.

```
<ItemTemplate>
   <asp:Label ID="Silt1" runat="server"
              Text='<%# Bind("tyyp") %>' />
</ItemTemplate>
```

Muutmise puhul aga taas vajalik rippmenüü. Kasutada saab sama eelnevalt loodud SqlDataSource2-nimelist andmeallikat. Abiinfost leitud SelectedValue nimeline omadus võimaldab algul sobiva sõna ette keerata. Samuti selle sama väärtuse kaudu jõuab uus valitud väärtus UpdateParameetrite kaudu andmeallikasse ja sealt edasi tabelisse.

```
<EditItemTemplate>
   <asp:DropDownList ID="DropDownList2" runat="server"
                     DataSourceID="SqlDataSource2"
       DataTextField="tyyp" DataValueField="id"
       SelectedValue='<%# Bind("sooid") %>'>
   </asp:DropDownList>
</EditItemTemplate>
```

Kogu tabeli kood tervikuna

```
<asp:GridView ID="GridView1" runat="server"
  DataSourceID="SqlDataSource1" AutoGenerateColumns="false"
  DataKeyNames="id">
  <Columns>
    <asp:CommandField ShowEditButton="true" EditText="Muuda"
      UpdateText="Uuenda" CancelText="Katkesta" />
    <asp:BoundField DataField="id" Visible="false" />
    <asp:BoundField DataField="kassinimi"   />
    <asp:BoundField DataField="kassimass" />
    <asp:TemplateField>
     <ItemTemplate>
      <asp:Label ID="Silt1" runat="server" Text='<%# Bind("tyyp") %>'/>
     </ItemTemplate>
     <EditItemTemplate>
        <asp:DropDownList ID="DropDownList2" runat="server"
            DataSourceID="SqlDataSource2" DataTextField="tyyp"
            DataValueField="id" SelectedValue='<%# Bind("sooid") %>'>
        </asp:DropDownList>
     </EditItemTemplate>
    </asp:TemplateField>
  </Columns>
</asp:GridView>
```

Pilt soo muutmise kohta

Muuda	Miisu	2300	Emane
Muuda	Nimetu armas kassipoeg	235	Teadmata
Muuda	Liisu	2000	Emane
Muuda	Ants	3750	Isane
Uuenda Katkesta	Tiisuke	2900	Isane
Muuda	Murka	4200	Teadmata Isane Emane Kohiisane

ning võib veenduda, et Tiisuke on vanaks ja rahulikuks kõutsiks tehtud.

Muuda	Miisu	2300	Emane
Muuda	Nimetu armas kassipoeg	235	Teadmata
Muuda	Liisu	2000	Emane
Muuda	Ants	3750	Isane
Muuda	Tiisuke	2900	Kohiisane
Muuda	Murka	4200	Isane

Ülesandeid

Katseta sünnilinna muutmise võimalus läbi inimeste ja linnade tabelite peal

Autode kirbuturg

- Koosta tabel automarkide tarbeks. Loo leht andmete sisestamiseks.
- Koosta tabel müüdavate autode tarbeks. Automargi saab valida olemasolevate hulgast. Lisatakse väljalaskeaasta, kommentaar, soovitav hind.
- Koosta leht andmete lisamiseks. Samuti soovitud tunnuse (mark, aasta, hind) järgi otsimiseks.
- Koosta tabel autoosade nimetuste tarbeks, loo leht andmete sisestamiseks.
- Täienda autode tabelit nõnda, et üheks tulbaks tuleks juurde müüdava autoosa kood. Vaikimisi väärtuseks on variandi „terve auto“ kood.
- Võimalda administraatorilehel parandada olemasolevaid sisestatud väärtusi.

- Täienda otsingulehte nõnda, et saaks ka määrata, millise margi millist osa otsitakse.

- Kaitse administraatorileht parooliga.

Veebi kopeerimine

ASP.NET veebirakendused on lihtsalt teisaldatavad. Nende liigutamiseks uude serverisse on vaja kopeerida kõik veebi jaoks vajalikud failid uude asukohta ning ongi veeb uues kohas.

Selle lihtsa tegevuse veelgi lihtsamaks muutmiseks on Visual Sudiosse lisatud veebi kopeerimise vahend, mille leiate menüüst „Website/Copy Web Site…“

Veebi kopeerimise vahend võimaldab teil veebi kopeerida nelja erinevat tüüpi sihtkohta. Ühenduse loomiseks oma veebi sihtkohaga vajutage nupule Connect ning valige veebi sihtkoht:

- File System – veebi asukoht on ligipääsetav failisüsteemi kaudu (nagu teeks windowsis copy/paste teise kausta)

- Local IIS – sama arvuti IIS Serveris[2] paiknev veebisait

- FTP Site – FTP[3] ligipääsuga veebisait. Üheks selliseks veebisaidiks on nt.ENETA[4] hostingukeskkond. Kui soovite oma veebi ENETA hostingusse ülesse riputada siis küsige oma õpetaja käest, kuidas te sinna ligi saate ja millistel tingimustel ning kui kaua saab seal veebi hoida

- Remote Site – HTTP[5] või HTTPS[6] abil ligipääsetav veeb. Ligipääs toimub üle tavalise veebikanali nagu ka veebi vaatamine, kuid lisaks peab teil olema veebi kirjutamise

---

[2] IIS – Internet Information Service so Windowsi teenus, mis võimaldab veebi serveerimist nii sisevõrku kui Internetti. Rohkem infot leheküljel 481

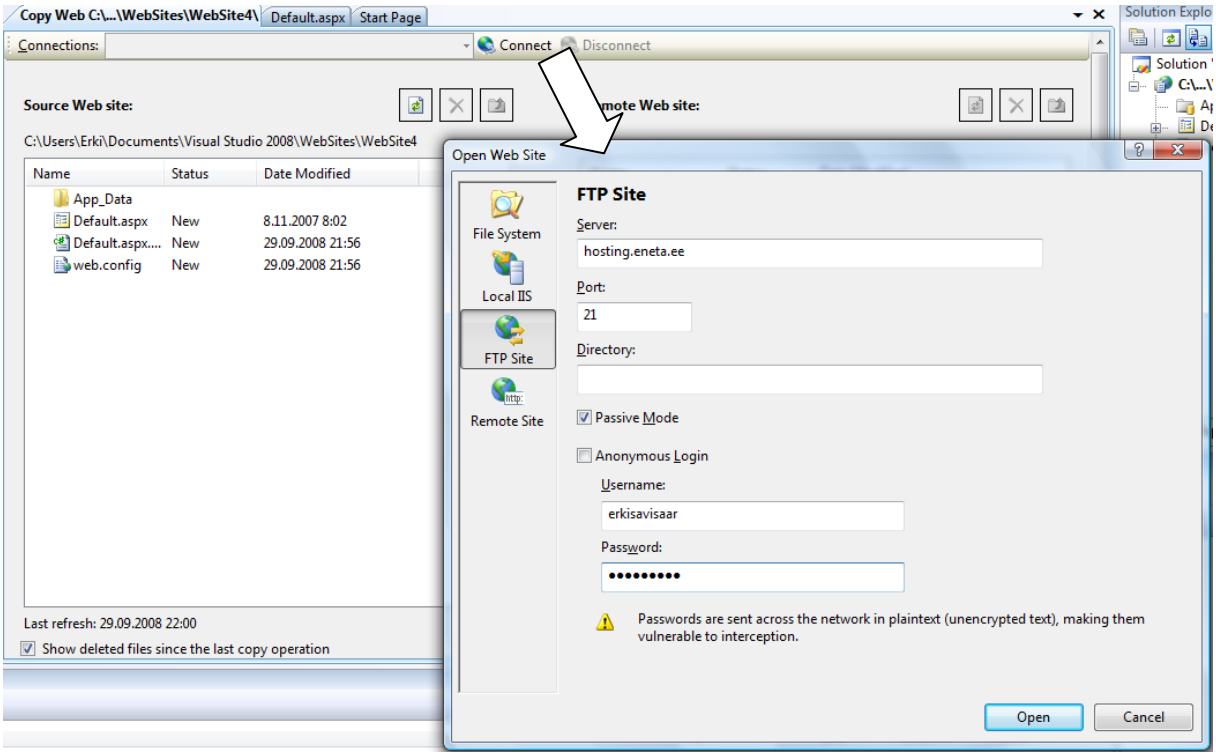
[3] FTP – File Transferee Protocol

[4] ENETA – Eesti .NET Assotsiatsioon – rohkem infot veebist http://www.eneta.ee

[5] HTTP – Hyper Text Transfeere Protocol – standard veebilehtede edastamiseks üle Interneti

[6] HTTPS – HTTP turvaline versioon, kus andmevahetus serveri ja kliendi (brauseri) vahel on krüpteeritud.

õigus. Üheks heaks näiteks selles vallas on oma veebi kopeerimine SharePoint7 portaali.



Antud näites luuakse FTP ühendus hosting.eneta.ee saidis asuva veebiga.

Kui ühendus loodud saate asuda failide kopeerimise ja sünkroniseerimise juurde. Kopeerimist lihtsustava lisainfona näidatakse teile, milliseid faile on kummalgi pool muudetud/kustutatud peale viimast sünkroniseerimist. Seda infot näidatakse nii Status veerus oleva kirjeldusega kui ka vastavate ikoonidega faili ees.

Teil on võimalus kõiki faile ja kaustu nii kopeerida kui ka kustutada.

Failide ja kaustade kustutamiseks valige failid ja kaustad, mida soovite eemaldada ning vajutage failivaliku peal asuvale punase ristiga tähistatud Delete nupule

Failide ja kaustade kopeerimiseks valige failid ja kasutad, mida soovite kopeerida ning vajutage kahe failivaliku vahel olevale noolele. Noolevalikuid on kolm:

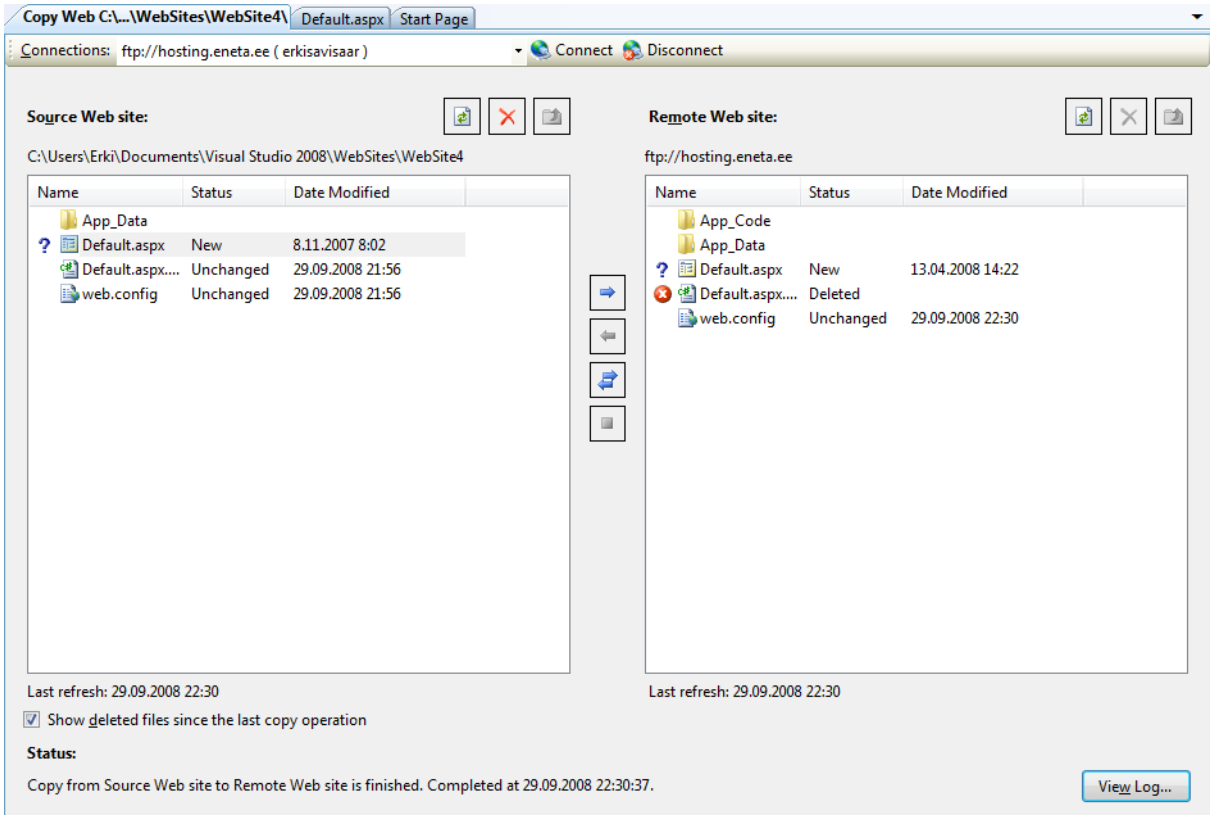
- Nool paremale kopeerib veebisaidile

---

[7] SharePoint – Microsoft Office SharePoint Server ja Microsoft Windows SharePoint Services on keskkonnad portaalide ehitamiseks. Neid kasutatakse enamasti intranetis töökorralduse ja failihalduse lahendamiseks. Rohkem infot veebist http://www.microsoft.com/sharepoint

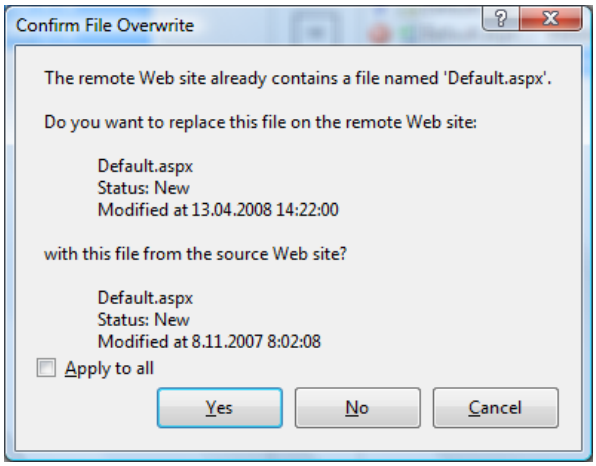
- Nool vasakule kopeerin veebisaidilt

- Mõlemas suunas nooled teevad mõlemad operatsioonid korraga e. kopeerivad vasakul valitud failid/kasutad veebi ning kopeerivad paremal valitud failid veebist arvutisse.



Kui fail, mida kopeerite on sihtkohas juba olemas esitatakse teile küsimus, kas kirjutada sihtkohas asuv fail üle või mitte.

Märgistades märkeruudu „Apply to all“ rakendatakse teie poolt antavat vastust kõigile analoogsetele küsimustele.



Kui veeb on suur ja/või ühendus veebiga on aeglane võib see kopeerimise protsess üksjagu kaua aega võtta.

Edasijõudnutele

Programmi koodi paigutamine eraldi faili

Kuigi veebirakendus lubab kirjutada koodi ja lehekülje märgistuse samasse faili, on enamasti mõistlik need kaks eraldada. Põhjus seisneb selles, et üldjuhul ei ole programmeerijad head disainerid ja ka vastupidi: hea disainer ei ole hea programmeerija. Jagades rakenduses lehekülje disaini ja koodi eraldi, saavad mõlemad tegelda just sellega, mis kõige paremini välja tuleb.

Seega kui teeme lihtsat veebirakendust, tuleks luua tegelikult kaks faili: default.aspx ja default.aspx.cs.

Faili default.aspx sisu võiks olla järgmine:

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!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>Minu esimene leht</title>
</head>
<body>
    <form id="form1" runat="server">
        Tere maailm
    </form>
</body>
</html>
```

Võrreldes esimese näitega on Page elemendile lisandunud kaks atribuuti.

- CodeFile atribuut – ütleb kompilaatorile, millisesse faili on salvestatud lehega seotud kood.

- Inherits atribuut – ütleb, millisesse klassi koodifailis on kood paigutatud

Lisaks .aspx failile tuleb nüüd luua ka page elemendis kirjeldatud koodifail (kuna kirjutame C# keeles peaks see olema .cs laiendiga), koos vastava klassiga. Kuna hetkel, meie lehel mingit erilist dünaamikat ei ole sisaldab loodav default.cs fail vaid klassi kirjeldust:

```
public partial class _Default : System.Web.UI.Page {}
```

Nagu ülal olevast näites näha pärinevad kõik ASP.NET lehed System.Web.UI.Page klassist, kus on realiseeritud raamistik, mis vajalik lehe genereerimiseks ning kontrollide kasutamiseks.

Tegelikult on üsna uskumatu, kui palju on võimalik ära teha deklaratiivselt ilma koodi e. .cs faili poole pöördumata. Samas on enam kui selge, et kui funktsionaalsus läheb natukenegi keerukamaks, tuleb ka programselt üht-teist korda saata.

Näiteks sellel samal lihtsal lehel on võimalik kirjutada kogu tekst dünaamiliselt koodi abil. Selleks eemaldame teksti .aspx failist:

```
<%@ Page Language="C#"  CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!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 />
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="lblTere" runat="server" />
    </form>
</body>
</html>
```

Ning lisame selle koodi abil:

```
using System;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Page.Title = "Minu esimene leht";
        lblTere.Text = "Tere maailm!";
    }
}
```

Page_Load on sündmus, mis tekib iga kord lehekülje laadimisel. Selleks, et uurida genereeritava asp.net lehe omadusi saame ära kasutada Page objekti.

Staatilise teksti kirjutamiseks on kõige mugavam kasutada lipikuid (asp:Label). See võimaldab hiljem lihtsa vaevaga teksti kujundada. Iga objekt, mille poole soovite koodis pöörduda peab omama unikaalset ID' d! Teksti saab lipikule lisada läbi Text atribuudi. Lipikud genereeruvad enamasti SPAN html tagideks.

Läbi kontrollide teksti kirjutamisel on üks oluline eelis võrreldes staatilise tekstiga – kontrollidel olevat teksti on lihtne koodis kasutada ning seda on võimalik lihtsa vaevaga lokaliseerida! Seega on alati soovitav kirjutada teksti veebilehele kas asp:label või mõne muu serveri kontrolli kaudu.

Ülesandeid

* Installeeri Visual Web Developer või leia enesele ligipääs IIS serverile

* Koosta ja käivita tervitav leht

* Määra lehel olev tervitus programmijupi abil.

* Küsi inimeselt nimi ja nupuvajutuse peale tervita teda, kopeerides tekst tekstiväljast sildile

Seadistamine (Web.config)

Igal veebirakendusel peaks olema ka oma seadistuste fail. ASP.NET rakenduse seadistust on võimalik seadistada üsna mitmest kohast. Alustades raamistiku üldseadistustest machine.config (paikneb .NET raamistiku programmifailide hulgas), seejärel veebisaidi seadistus e. web.config fail juurveebis ning lõpetades veebsaidi juurkataloogis oleva web.config failiga. Lisaks on võimalik erinevatele alamkaustadele eriseadistuse andmiseks võimalik lisada igasse kausta oma web.config fail.

Lõpuks jäävad kehtima need määrangud, mis on leheküljele kõige lähemal! Kuna masina seadistusele ja veebisaidi seadistusele meil enamasti ligipääsu ei ole siis püüame kogu vajaliku seadistuse ära teha veebirakenduse juurkataloogis oleva web.config faili abil.

.config failid on XML failid, milles antakse lisainformatsiooni nii rakendusele endale kui ka .NET raamistikule ja kompilaatorile.

Kõik väärtused/muutujad, mida soovite, et programmi kasutaja/administraator saaks hiljem lihtsalt muuta tuleks sisestada web.config faili!

Esialgu võiks web.config fail näha välja järgmine:

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true" />
    </system.web>
</configuration>
```

Selle seadistusega ütleme, et meil on kõik nii nagu serveris ikka, kuid me soovime oma rakenduse juures näha lisainfot silumiseks. Kui programm saab valmis, peaks panema debug atribuudi väärtuseks ”false”. Silumisinfo olemasolul kuvatakse teile vigade tekkimisel konkreetsed koodiread.

ASP.NET veebi seadistus on mitmekihiline ning esmased seadistused teie veebisaidile võetakse masina (veebiserveri) seadistustest. Seejärel lisandub juurveebi seadistus ning alles seejärel jõutakse teie veebi seadistusteni.

Lisaks oma veebi juurkataloogis olevale seadistusele võite teatud seadeid muuta ka alamkaustades. Selleks on teil kaks võimalust:

- Lisada alamkasuta eraldi web.confog fail, kus on vastavad seaded ümber tehtud

- Lisada veebi juurkataloogis olevasse web.config faili location element, ning näidata ära, mida soovite ümber teha. Sellekohast näidet saad vaadata leheküljel 445.

Rakenduse jälgimine (Trace)

Nii ASPi kui ka PHP suur puudus on see, et programmi silumiseks või vea leidmiseks vajaliku metainfo kuvamine on väga keeruline. Enamasti lõppeb see metainfo kirjutamisega otse lehele, rikkudes sellega ära lehekülje üldise väljanägemise.

ASP.NETil on selle tarbeks märksa mugavam vahend – Trace. Selleks, et jälgida ühel lehel toimuvat on kõige lihtsam Page elemendis sisse lülitada Trace atribuut.

```
<%@ Page Language="C#"
CodeFile="Default.aspx.cs"
Inherits="_Default" ,
Trace ="true" %>
```

Selle ühe väikse atribuudi lisamisega lisatakse lehekülje lõppu terve hulk informatsiooni alustades sellest, mis moodi seda lehekülge küsiti ning kirjeldades ära kogu lehekülje loomise protseduuri (koos lehekülje struktuuriga) ning lõpetades loeteludega erinevatest muutujatest (sessioon, programm, server jne).

Tere maailm!

Request Details

Session Id:	0bvavz3bkoddjm2kttuwvi55	**Request Type:**	GET
Time of Request:	27.11.2006 20:24:38	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0,00139794303465943	0,001398
aspx.page	Begin Init	0,00239415903417693	0,000996
aspx.page	End Init	0,00262323842834774	0,000229
aspx.page	Begin InitComplete	0,00281153051574991	0,000188
aspx.page	End InitComplete	0,00299954323803724	0,000188
aspx.page	Begin PreLoad	0,00318504167429101	0,000185
aspx.page	End PreLoad	0,00337640677795642	0,000191
aspx.page	Begin Load	0,00356218457932503	0,000186
aspx.page	End Load	0,00416980370410206	0,000608
aspx.page	Begin LoadComplete	0,00437988627046175	0,000210
aspx.page	End LoadComplete	0,00456817835786392	0,000188
aspx.page	Begin PreRender	0,00475926409641449	0,000191
aspx.page	End PreRender	0,00495593713726186	0,000197
aspx.page	Begin PreRenderComplete	0,00514255303397499	0,000187
aspx.page	End PreRenderComplete	0,00532749273999908	0,000185
aspx.page	Begin SaveState	0,0666025989336634	0,061275
aspx.page	End SaveState	0,094853116806745	0,028251
aspx.page	Begin SaveStateComplete	0,0950472755615588	0,000194
aspx.page	End SaveStateComplete	0,0952199232025299	0,000173
aspx.page	Begin Render	0,0953816756040223	0,000162
aspx.page	End Render	0,0991489141776399	0,003767

Control Tree

Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
__Page	ASP.default_aspx	569	0	0
ctl02	System.Web.UI.LiteralControl	171	0	0
ctl00	System.Web.UI.HtmlControls.HtmlHead	50	0	0
ctl01	System.Web.UI.HtmlControls.HtmlTitle	37	0	0
ctl03	System.Web.UI.LiteralControl	14	0	0
form1	System.Web.UI.HtmlControls.HtmlForm	314	0	0
ctl04	System.Web.UI.LiteralControl	10	0	0
lblTere	System.Web.UI.WebControls.Label	38	36	0
ctl05	System.Web.UI.LiteralControl	6	0	0
ctl06	System.Web.UI.LiteralControl	20	0	0

Session State

Session Key	Type	Value

Application State

Application Key	Type	Value

Request Cookies Collection

Name	Value	Size

Response Cookies Collection

Name	Value	Size

Headers Collection

Name	Value
Connection	Keep-Alive
Accept	*/*
Accept-Encoding	gzip, deflate
Accept-Language	et
Host	localhost:50505
User-Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; .NET CLR 3.0.04320)
UA-CPU	x86

Response Headers Collection

Name	Value
X-AspNet-Version	2.0.50727
Cache-Control	private
Content-Type	text/html

Form Collection

Name	Value

Querystring Collection

Name	Value

Server Variables

Name	Value
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:*/* HTTP_ACCEPT_ENCODING:gzip, deflate HTTP_ACCEPT_LANGUAGE:et HTTP_HOST:localhost:50505 HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; .NET CLR 3.0.04320) HTTP_UA_CPU:x86
ALL_RAW	Connection: Keep-Alive Accept: */* Accept-Encoding: gzip, deflate Accept-Language: et Host: localhost:50505 User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; .NET CLR 3.0.04320) UA-CPU: x86
APPL_MD_PATH	
APPL_PHYSICAL_PATH	C:\Users\Erki\Documents\Visual Studio 2005\WebSites\MS_Proj2\
AUTH_TYPE	NTLM
AUTH_USER	MP\Erki
AUTH_PASSWORD	
LOGON_USER	MP\Erki
REMOTE_USER	MP\Erki
CERT_COOKIE	
CERT_FLAGS	
CERT_ISSUER	
CERT_KEYSIZE	
CERT_SECRETKEYSIZE	
CERT_SERIALNUMBER	
CERT_SERVER_ISSUER	
CERT_SERVER_SUBJECT	
CERT_SUBJECT	
CONTENT_LENGTH	0
CONTENT_TYPE	
GATEWAY_INTERFACE	
HTTPS	
HTTPS_KEYSIZE	
HTTPS_SECRETKEYSIZE	
HTTPS_SERVER_ISSUER	
HTTPS_SERVER_SUBJECT	
INSTANCE_ID	
INSTANCE_META_PATH	
LOCAL_ADDR	127.0.0.1
PATH_INFO	/MS_Proj2/Default.aspx
PATH_TRANSLATED	C:\Users\Erki\Documents\Visual Studio 2005\WebSites\MS_Proj2\Default.aspx
QUERY_STRING	
REMOTE_ADDR	127.0.0.1
REMOTE_HOST	127.0.0.1
REMOTE_PORT	
REQUEST_METHOD	GET
SCRIPT_NAME	/MS_Proj2/Default.aspx
SERVER_NAME	localhost
SERVER_PORT	50505
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	
URL	/MS_Proj2/Default.aspx
HTTP_CONNECTION	Keep-Alive
HTTP_ACCEPT	*/*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	et
HTTP_HOST	localhost:50505
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; .NET CLR 3.0.04320)

Loomulikult on võimalik sinna abiinfosse lisada ka oma märkuseid. Jälitusinfo kuvamiseks ja uurimiseks on Trace klass. Jälitus infot kirjutada saab kahe meetodiga:

- Trace.Write – lihtsalt jutt jälitusinfosse
- Trace.Warn – hoiatus (kuvatakse jälitusinfot vaadates punasena)

Lisame näiteks pisut jälitusinfot oma default.cs faili:

```
using System;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Trace.Write("Alustan Page_Load meetodiga ...");
        Page.Title = "Minu esimene leht";
        lblTere.Text = "Tere maailm!";
        Trace.Warn("Meetod lõpetatud");
    }
}
```

Tulemuseks on kaks uut rida Traces, Begin Load ja End Load sündmuste vahel:

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0,000109511125017286	0,000110
aspx.page	Begin Init	0,000281041305529055	0,000172
aspx.page	End Init	0,000458996883682144	0,000178
aspx.page	Begin InitComplete	0,000622146110748712	0,000163
aspx.page	End InitComplete	0,000785295337815281	0,000163
aspx.page	Begin PreLoad	0,000947327104422489	0,000162
aspx.page	End PreLoad	0,00139542874862587	0,000448
aspx.page	Begin Load	0,00156640019890796	0,000171
	Alustan Page_Load meetodiga ...	0,00239108601791568	0,000825
	Meetod lõpetatud	0,00273694003008762	0,000346
aspx.page	End Load	0,00290986703617359	0,000173
aspx.page	Begin LoadComplete	0,00307357499346984	0,000164
aspx.page	End LoadComplete	0,00347809567975818	0,000405
aspx.page	Begin PreRender	0,00364152427193959	0,000163
aspx.page	End PreRender	0,00381528937337008	0,000174
aspx.page	Begin PreRenderComplete	0,00398067352135537	0,000165
aspx.page	End PreRenderComplete	0,00427149260590382	0,000291
aspx.page	Begin SaveState	0,00469668631069033	0,000425
aspx.page	End SaveState	0,00545767688351453	0,000761
aspx.page	Begin SaveStateComplete	0,00562725150822241	0,000170
aspx.page	End SaveStateComplete	0,00579012137017414	0,000163
aspx.page	Begin Render	0,00595215313678135	0,000162
aspx.page	End Render	0,00651647066875818	0,000564

Kui soovite mingi sündmuse ilmnemisel natukene põhjalikumat analüüsi teha siis on kasulik enne analüüsi alustamist kontrollida, kas jälitus on üldse sisse lülitatud. Selleks saame uurida Trace objekti IsEnabled omadust. Väga kasulik nt Try ... Catch konstruktsioonides:

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        object o = null;
        Trace.Write("Alustan Page_Load meetodiga ...");
```

```
            Trace.Warn("Tekitan vea", o.ToString());
            Page.Title = "Minu esimene leht";
            lblTere.Text = "Tere maailm!";
        }
        catch (Exception ex)
        {
            if (Trace.IsEnabled)
            {
                Trace.Warn("error", "Tekkis tõrge", ex);
                // TODO: Siia tuleks teha analüüs, miks viga tekkis
            }
        }
}
```

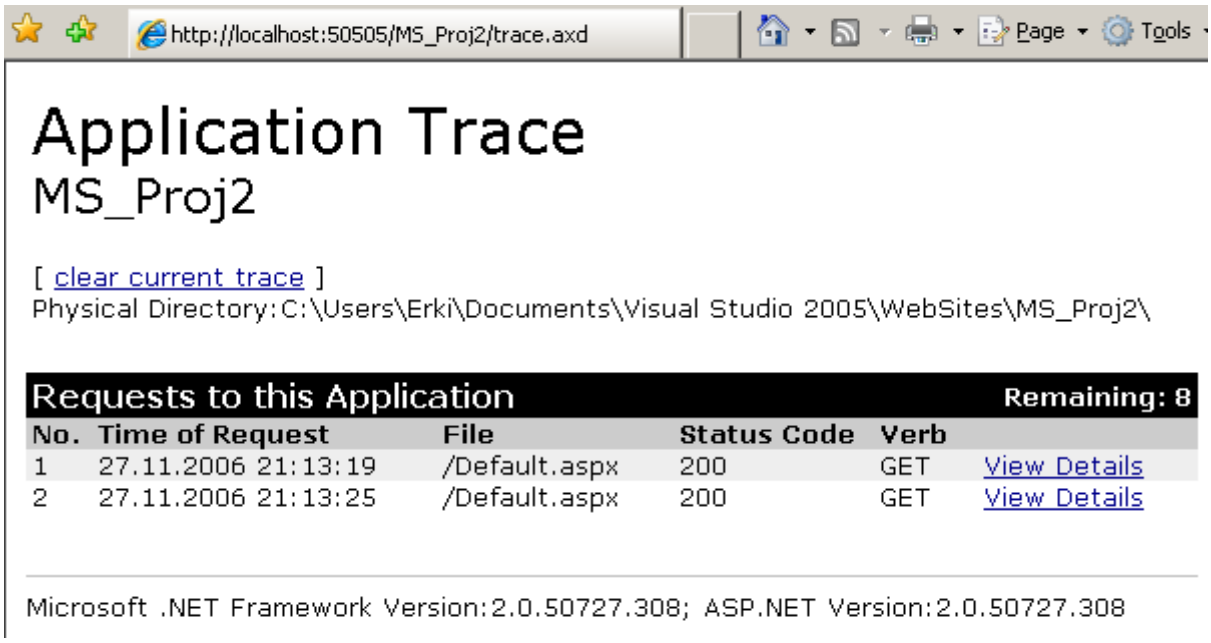
Tulemuseks trükitakse detailne vea kirjeldus jälitusinfo sisse:

aspx.pageBegin Load		0,00154963829201756	0,000240
	Alustan Page_Load meetodiga ...	0,00295763847081123	0,001408
error	Tekkis tõrge Object reference not set to an instance of an object. at _Default.Page_Load(Object sender, EventArgs e) in c:\Users\Erki\Documents\Visual Studio 2005\WebSites\MS_Proj2\Default.aspx.cs:line 11	0,00346412742401618	0,000506
aspx.pageEnd Load		0,0401011860445951	0,036637

Lisaks lehekülje põhisele sisse-välja lülitamisele on võimalik jälitusinfot kontrollida ka keskselt. Selleks tuleb vastav viide panna web.config faili system.web elemendi alamelemendiga trace:

```
<trace enabled="true" />
```

Nüüd ei kuvata enam infot mitte iga lehekülje lõpus vaid virtuaalse veebilehe trce.axd abil veebirakenduse juurkataloogis:



Klõpsates iga päringu järel oleval View Details lingil kuvatakse meile info, mis varem oli lisatud lehekülje lõppu.

Trace elemendil on lisaks enable atribuudile veel teisigi mis võimaldavad selle info kogumist paremini korraldada. Näiteks võime öelda, et

- jälitusinfo on nähtav ainult konsoolil (localhost) ja üle veebi seda ei serveerita localOnly = „true” (default)

- nähtavale jäävad kõige viimased päringud mostRecent = „true”. Vaikimisi näidatakse esimest x päringut!

- Sorteerime päringud ajalisse järjekorda traceMode=”SortByTime”

- Jätame meelde 100 päringut (10 asemel) requestLimit=”100”

- Korjame jälitusinfo trace.axd sisse mitte ei pane lehekülgede lõppu pageOutput=”false”

```
<trace enabled="true"
       localOnly="true"
       mostRecent="true"
       traceMode="SortByTime"
       requestLimit="100"
       pageOutput="false"/>
```

PS! Jälitusinfot hoitakse mälus ning see kustub koos rakenduse mälust eemaldamisega.

Ülesandeid

* Lülita lehel sisse Trace, tutvu saada oleva teabega

* Trüki lehe koodis trace-teateid, loe neid.

* Vaata trace-infot nii lehe alt kui eraldi lehelt.

Vigade haldamine

Ükskõik kui väga me ka ei püüaks ikkagi esineb programmis vigu, olgu nad siis tingitud asjaolude kokkusattumusest, programmeerijapoolsest lohakusest või kasutaja pahatahtlikkusest.

.NET raamistik koos ASP.NETiga pakub vigade haldamiseks üsna mitmekihilist süsteemi. Loomulikult tuleks vead kõrvaldada nii vara kui võimalik, kuid alati tuleb arvestada

asjaoludega, et kõike pole võimalik ette ennustada ja midagi võib jääda kahe silma vahele! Seega tuleks igal kihil pakkuda mingit lahendust vigade haldamiseks.

Kui Te jätate mõne vea kinni püüdmata siis teeb seda lõpuks ASP.NET ise, kuid nagu arvata võite, ei pruugi sealt tulevad veateated kasutajatele meeldida.

Rakenduse veebist eemaldamine

Kui rakenduses ilmnevad suuremad tõrked või on vaja teha hooldustöid on kasulik veebirakendus ajutiselt „maha võtta" e. muuta kasutajatele kättesaamatuks. ASPi ajal oli ainukeseks arvestatavaks meetodiks, kas rakenduse ümbernimetamine või veebiteenuse peatamine. ASP.NET pakub aga väga mugavat alternatiivi – nimelt tuleb rakenduse maha võtmiseks lisada veebirakenduse juurkataloogi fail app_offline.htm. Fail peab kindlasti olema htm mitte html laiendiga! Faili sisse kirjutage teade, mida soovite saidi külastajatele anda.

Näiteks võiks kirjutada midagi järgmist:

```
<!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>
    <meta content="text/html; charset=utf-8" http-equiv="content-type" />
    <title>Teenus ajutiselt suletud</title>
</head>
<body>
<h1>Teenus ajutiselt suletud!</h1>
<p>
Vabandame, seoses hooldustöödega ei ole veebirakendus hetkel
kättesaadav.<br />
Püüame probleemi kõrvaldada nii kiiresti kui võimalik.</p>
<p>Ootame teid peatselt tagasi!</p>
<!-- Selleks, et kõik ka IEs korrektselt toimiks peab lehekülje maht olema
vähemalt 512B -->
</body>
</html>
```

Kui soovite rakenduse taas käima panna siis võib selle app_offline.htm faili ära kustutada või ümber nimetada.

Failide veebist väljaarvamiseks võib neile kirjutada .exclude laiendi. Sellise laiendiga faile veebi ei serveerita. Tulemuseks on HTTP error 403 (Page not served).

Globaalne veakontroll

Globaalne veakontroll on viimane kontrollliin enne raamistikku ning võimaldab reageerida vigadele vastavalt HTTP veakoodidele. Veakontrolli sisselülitamiseks tuleb web.configi system.web sektsiooni lisada customErrors element. Kõige lihtsama lahenduse jaoks on vaja moodustada üks lehekülg, mis hakkab kas viga programselt haldama või kuvab kasutajale mingi viisaka teate.

customErrors elemendil on kaks atribuuti:

- mode näita, kuidas vea puhul käituda (On – näidata kõigile oma veateadet, Off – näidata kõigile süsteemset veateadet, RemoteOnly - näidata konsoolilt pöördujatele süsteemset veateadet ja teistele kohandatud veateadet[8].

- defaultRedirect näitab, kuhu kasutaja suunata, kui midagi konkreetsemat pole öeldud

Nt järgnev rida web.configis ütleb, et kui tekib mingi tõrge tuleb kasutaja suunata lehele error.html.

```
<customErrors mode="On" defaultRedirect="~/vead/error.html" />
```

Kui soovite teatud HTTP vigade puhul anda mingi konkreetsema teate siis saate lisada customErrors elemendi alla täiendavad error elemendid.

Näiteks järgmise seadistusega ütleme, et kui tuleb HTTP viga 404 suuname kasutaja lehele error404.html, kui tuleb HTTP error 500 suuname kasutaja lehele error500.html, kui tuleb mingi muu viga suuname kasutaja lehele error.html.

```
<customErrors mode="On" defaultRedirect="~/vead/error.html">
  <error statusCode="404" redirect="~/vead/error404.html"/>
  <error statusCode="500" redirect="~/vead/error500.html"/>
</customErrors>
```

Keskne veakontroll

Globaalsetele sündmustele saab reageerida globaalses koodifailis global.asax. Veakontrolli tarbeks on seal meetod Application_Error, mille sisse võite kirjutada oma vea haldamise protseduuri.

Application_Error meetod käivitatakse, kui programmis tekib viga, mida leheküljel ära ei lahendata. Tegemist on viimase programse võimalusega tekkinud vea viisakaks lahendamiseks.

```
Vea haldamiseks tuleb ära kasutada Server objekt.
Server.GetLastError() annab tagasi Exception tüüpi objekti viimase
veaga. Sealt saate teada missuguse veaga oli tegemist.
Peale seda kui olete pakkunud veale mingi viisaka lahenduse saate
öelda, et viga on lahendatud kutsudes välja Server.ClearError()
meetodi.
```

---

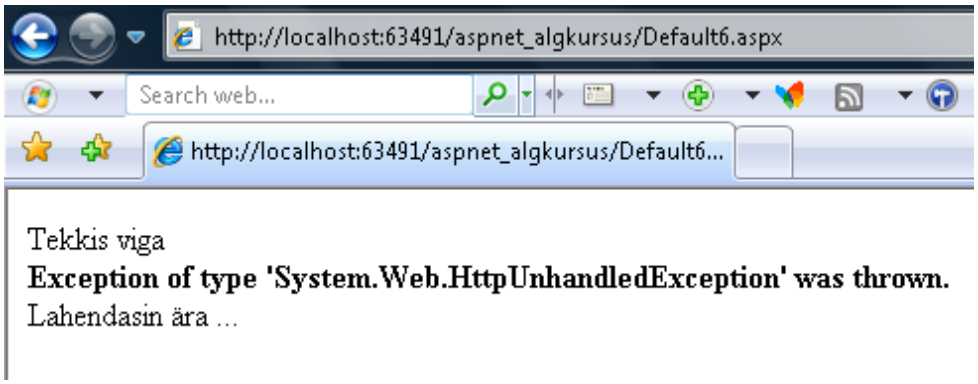
[8] Kui soovite veateateid väljastada vaid konkreetsele IP aadressile siis tuleb selle koha peal kasutada pisut kavalamat protseduuri. Täpsemad juhised leiate mitmetest foorumitest või MSDNist (http://msdn2.microsoft.com/en-us/library/aa479319.aspx)

Nt moodustame järgmise global.asax faili:

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Error(object sender, EventArgs e)
    {
        Response.Write("Tekkis viga<br /><b>");
        Response.Write(Server.GetLastError().Message);
        Server.ClearError();
        Response.Write("</b><br />Lahendasin ära ...");
    }
</script>
```

Kui nüüd mõni leht tekitab vea, mida kohapeal ära ei lahendata (nt all olev protseduur) siis käivitub keskne veakontroll ning ASP.NETi süsteemset veateadet kasutajale ei näidata.

```
using System;
public partial class Default6 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        object o = null;
        Page.Title = "Globaalse veakontrolli test";
        lblTere.Text = o.ToString();
    }
}
```



Veakontroll lehel

ASP.NET leheküljele leheküljekeskse veakontrolli paigaldamiseks tuleb leheküljele lisada Page_Error meetod. Page_Error meetod on mõeldud olukordade jaoks kus viga ei ole võimalik meetodite siseselt kinni püüda ja lahendada või kui soovite pakkuda lehekülje lõikes keskset veakontrolli.

Vea uurimiseks on võimalik kasutada taas (nagu ka veakontrollis global.asax failis) Server objekti.

```
protected void Page_Error(object sender, EventArgs e) {
```

```
    Response.Write("Tekkis viga<br /><b>");
    Response.Write(Server.GetLastError().Message);
    Server.ClearError();
    Response.Write("</b><br />Lahendasin lokaalselt ära ...");
}
```

Page_Error meetod on ainult System.Web.UI.Page klassist pärinevatel lehtedel. Pealehel (master page vt lk 420) sellist meetodit ei ole! Kui soovite sellist leheküljekeskset veakontrolli kasutada kõigil lehtedel on võimalik aspx lehe tarbeks moodustada oma baasklass, mis pärineb System.Web.UI.Page klassist ning lahendada Page_Error meetod selle sees.

Konstruktsioon näeks siis välja järgmine:

- lisame uue klassifaili LehePohi.cs. See fail oleks kasulik paigutada veebi juurikas olevasse App_Code kausta, siis on ta kõikjalt ühte moodi hästi kasutatav. Sinna faili tekitame oma lehtede tarbeks baasklassi:

```
using System;
public abstract class LehePohi : System.Web.UI.Page
{
    protected void Page_Error(object sender, EventArgs e)
    {
        Response.Write("Tekkis viga<br /><b>");
        Response.Write(Server.GetLastError().Message);
        Server.ClearError();
        Response.Write("</b><br />Lahendasin lokaalselt ära ...");
    }
}
```

- Kui seejärel pärime oma lehe LehePohi klassist saame sellega kaasa ka lahenduse Page_Error meetodile.:

```
using System;
public partial class Default7 : LehePohi
{
    protected void Page_Load(object sender, EventArgs e)
    {
        object o = null;
        Page.Title = "Globaalse veakontrolli test";
        lblTere.Text = o.ToString();
    }
}
```

- Kui mõnel lehel on vaja personaalset lähenemist veakontrollile on võimalik Page_Error meetod kas üle kirjutada või pärida see leht otse System.Web.UI.Page klassist

Veakontroll koodis

Kuigi ASP.NET pakub mitmeid võimalusi nii globaalseks kui ka lokaalseks veakontrolliks on kõige parem kui suudate vigu ennetada või siis lahendada nad kohe tekkimise hetkel. Meetodi sees vigadele reageerimiseks saab kasutada Try ... Catch konstruktsiooni, mis võimaldab väga mugavalt vigu hallata.

Ülesandeid

* Lisa rakenduse juurkataloogi fail app_offiline.htm. Veendu, et rakendus pole veebist kättesaadav. Eemalda faili ja kontrolli, et kõik taas toimiks.

* Loo vea puhul näitamiseks eraldi leht. Tekita lehel veaolukord. Veendu, et kasutaja suunataks sinna.

* Koosta eri vealehed lehe puudumise (404) ning muude vigade kohta. Katseta toimimist

AJAX

AJAX - Asyncronious Java And Xml on raamistik, mis võimaldab asünkroonset andmevahetust kliendi e. veebisirvija ja serveri vahel. Kui AJAX-it ei kasuta siis iga serveri poole pöördumisega renderdatakse terve lehekülg uuesti ning laetakse terve lehekülg uuesti ka veebisirvijasse. Seda protsessi on veebilehte kasutades hästi näha sest iga klõpsu peale veebivormil lehekülg korraks vilksatab ning laetakse uuesti. Kasutades ajaksit värskendatakse lehekülge osaliselt. See saavutatakse tänu ASP.NET ja JavaScripti koostööle, kus andmevahetus käib XML vormingus. Java võimaldab lehekülje osalist muutmist kasutades DHTMLi ning ASP.NETi on lisatud võimalus lehekülje osaliseks renderdamiseks.

AJAX on sisse ehitatud .NET raamistik 3.5e ning saadaval ka eraldi .NET raamistik 2.0 tarbeks.

AJAXi kasutamine

Kõik AJAXiga seotud vahendid paiknevad Visual Studio Tööriistades (Toolbox) AJAX Extension kategoorias.

AJAXi kasutamiseks tuleb igale lehele lisada üks ja ainult üks ScriptManager. ScriptManageri ülesandeks on kasutaja browserisse vajalike JavasCriptide üleslaadimine ning andmevahetuse korraldamine veebisirvija ja serveri vahel.

```
<asp:ScriptManager ID="sm" runat="server" />
```

Kõik osad, veebivormist, mis vajavad osalist uuendamist tuleb paigutada UpdatePanel plokkidesse. Selliseid elemente võib lehel olla mitu ning iga UpdatePanel plokki on võimalik värskendada vastavalt vajadusele.

Värskendamise ajal on võimalik kuvada informatiivseid teated UpdateProgress elemendi abil.

Timer element võimaldab sooritada ajastatud tegevusi.

Järgnevalt väike näide veebilehest, mis kasutab AJAX funktsionaalsust GridViews olevate andmete filtreerimiseks.

```
<asp:ScriptManager ID="sm" runat="server" />
<p>Staatus:
<asp:UpdatePanel ID="u1" runat="server">
    <ContentTemplate>
        <asp:DropDownList ID="ddl1" runat="server"
                                                AutoPostBack="True">
            <asp:ListItem Value="True">Tehtud</asp:ListItem>
            <asp:ListItem Selected="True" Value="False">
                                              Tegemata</asp:ListItem>
        </asp:DropDownList>
    </ContentTemplate>
</asp:UpdatePanel></p>
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>Uuendan andmeid .....</ProgressTemplate>
</asp:UpdateProgress>
<asp:UpdatePanel ID="u2" runat="server">
    <ContentTemplate>
       <asp:GridView ID="GridView1" runat="server"
            DataKeyNames="Kood" DataSourceID="sds1"
            AllowSorting="True" >
            <Columns>
                <asp:CommandField ShowEditButton="True" />
            </Columns>
        </asp:GridView>
<asp:SqlDataSource ID="sds1" runat="server"
     ConnectionString="<%$ ConnectionStrings:toodConn %>"
     SelectCommand="SELECT [Kood], [Nimi], [Tehtud] FROM
                            [tooderegister] WHERE Tehtud = @Tehtud"
    UpdateCommand="UPDATE [tooderegister]
        SET [Nimi] = @Nimi, [Tehtud] = @Tehtud WHERE [Kood] = @Kood">
<SelectParameters>
    <asp:ControlParameter ControlID="ddl1" Name="Tehtud"
         PropertyName="SelectedValue" Type="Boolean" />
    </SelectParameters>
    <UpdateParameters>
         <asp:Parameter Name="Nimi" Type="String" />
         <asp:Parameter Name="Tehtud" Type="Boolean" />
         <asp:Parameter Name="Kood" Type="Int32" />
    </UpdateParameters>
</asp:SqlDataSource>
</ContentTemplate>
</asp:UpdatePanel>
```

AJAX Control Toolkit

Kui standardsetest vahenditest jääb väheseks siis saate lisaks laadida endale AJAX Control Toolkit'i. http://www.asp.net/ajax/ajaxcontroltoolkit

Juhendi Control Toolkiti paigaldamiseks leiate aadressilt
http://www.asp.net/AJAX/AjaxControlToolkit/Samples/Walkthrough/Setup.aspx

Samal leheküljel on ka näiteid kõigi Toolkiti koosseisu kuuluvate elementide kasutamise kohta.

Lokaliseerimine

Tänapäeva globaliseeruvas keskkonnas ei ole mõeldav, et veebirakenduse kasutajaliides on loodud vaid ühes keeles. On iseenesest mõistetav, et kasutaja saab ise valida endale sobiva keele ning veelgi enam võiks rakendus ka esmase keelevaliku teha vastavalt kasutaja eelnevatele seadistustele.

Seega, kui hakkame looma oma veebirakendust, püüame selle teha kohe nii, et tegemist oleks mitmekeelse rakendusega.

Rakenduse lokaliseerimiseks kasutatakse ASP.NETis ressursifaile (.resx), mis tuleb koondada leheküljega samal tasemel olevatesse alamkaustadesse:

- App_GlobalResources
- App_LocalResources

App_GlobalResources kaustas paiknevad rakenduse globaalsed ressursid e. väärtused, mis rakenduse piires ei muutu.

App_LocalResources kaustas paiknevad lehekülje jaoks lokaalsed ressursid e. väärtused, mis on seotud vaid ühe leheküljega. Sinna kausta tuleb iga lehe ning keele jaoks teha eraldi ressursi .resx fail. Kõige lihtsam oleks ressursi fail lisada läbi Visual Studio - siis kirjutatakse kogu päise info ära automaatselt.

Loomulikult on võimalik lokaliseeritud ressursse hoida ka kusagil mujal, kuid selle jaoks tuleb ümber kirjutada lokaliseerimise teenusepakkuja (Provider). Kuidas seda korraldada saate vaadata materjali lõpus olevast lisast lk 469.

Lokaalsed ressursid

Samas võib ressursifaile ka käsitsi teha. Näiteks lisame faili Default.aspx.resx. .resx fail on XML fail, mille juurelemendiks on <root>. Järgnevalt tuleks kirjeldada faili versioon:

```
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>2.0</value>
</resheader>
<resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
```

Kui päis moodustatud, lisame kõik vajalikud väärtused. Lisaks tekstilistele väärtustele on tegelikult võimalik lokaliseerida ka kõiki teisi väärtuseid näiteks suurus, värv jne

```
<data name="Label1.Text">
  <value>Esimese labeli tekst Eesti keeles</value>
</data>
<data name="Label1.ForeColor">
  <value>Blue</value>
</data>
<data name="Label2.Text">
  <value>Teise labeli tekst Eesti keeles</value>
</data>
<data name="Page.Title" >
  <value>Esimene lehekülg</value>
</data>
```

Kui eestikeelse tekstiga ressursifail on valmis, tuleks korrata sama toimingut ka kõigi teiste vajalike keelte jaoks. Kõigi järgnevate keelte ressursifailide nimedes peab sisalduma ka keel, mille kohta ressursse lisate. Kirjutades failide nimed sellisel kujul, vaatab .NET raamistik kõigepealt vajaliku tunnusega keelefaili, kui seda ei leia (või ei leia sellest otsitavat ressurssi), pöördutakse ilma keele laiendita ressursifaili poole. Seega ilma laiendita ressursifailis PEAVAD olema kõik kasutatavad ressursid kirjeldatud! Muul juhul ei pruugita neid ka olemasolevatest keelefailidest leida.

Näiteks lisame inglise keele tarbeks faili Default.aspx.en.resx

```
<root>
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>2.0</value>
</resheader>
<resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
```

```
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<data name="Label1.Text">
  <value>First label in English</value>
</data>
<data name="Label1.ForeColor">
  <value>Red</value>
</data>
<data name="Label2.Text">
  <value>Second label in English</value>
</data>
<data name="Page.Title" >
  <value>First Page</value>
</data>
</root>
```

Loomulikult tuleb ka Default.aspx failis mainida, et ta võib ressursse otsima hakata. Selleks tuleb päisesse (Page tag) lisada atribuudid Culture="Auto" UICulture="Auto".

```
Kui on soov lokaliseerida terve veebirakenduse, võib selle info
kirjutada ka konfiguratsioonifaili lisades system.web elemendi alla
alamelemendi globalization:
<globalization culture="auto" uiCulture="auto"/>
```

Culture ja UICulture vahe seisneb selles, et Culture määrab ära kuupäevade ja numbrite kujunduse ning UICulture määrab ära ressursifailide kasutuse.

PS! Jälgige tähesuuruseid: aspx failis algavad Culture ja UICulture suurte tähtedega, web.config failis aga väikeste tähtedega!

Lisaks tuleb ära märgistada kõik kontrollid, mis on plaanis lokaliseerida. Selleks tuleb kontrolli alustava tagi sisse lisada atribuut meta:resourcekey, mille väärtus peab vastama ressursifailis kirjeldatule. Antud juhul soovime lokaliseerida lehekülje pealkirja, seega peaks lisama meta atribuudi Page elemendile.

Page element näeb nüüdseks välja järgmine:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" meta:resourcekey="Page" Culture="Auto" UICulture="Auto"
%>
```

Lisaks mõnele lehekülje määrangule soovime lokaliseerida kahe labeli Label1 ja Label2 omadusi. Nimetatud lipikud näevad välja järgnevad:

```
<asp:Label ID="Label1" runat="server" meta:resourcekey="Label1" /><br />
<asp:Label ID="Label2" runat="server" meta:resourcekey="Label2" />
```

Nüüd jääb muidugi küsimus, kuidas raamistik aru saab, millist keelt kasutada? Selleks on kaks võimalust. Esmalt on võimalik käsitsi muuta Culture ja UICulture atribuute. Samas kui kultuuri atribuudid on seatud automaatse valiku peale, valib raamistik keele vastavalt sellele, mis määratud veebisirvijas. Internet Exploreris on see info kirjas Tools/Internet Options/General/Languages vormil.

Lokaalseid ressursse on võimalik kasutada ka programselt. Lokaliseerimisega seotud klasside kasutamiseks on kasulik öelda, et soovite kasutada System.Globalization nimeruumi.

```
using System.Globalization;
```

Ressursside lugemiseks on kõige mugavam kasutada meetodit GetLocalResourceObject, mis vajab parameetrina ressursi nime. Näiteks muudame programselt Label1 lipiku sisu:

```
Label1.Text = GetLocalResourceObject("Label1.Text").ToString();
```

Globaalsed ressursid

Globaalseteks nimetatakse ressursse, mis on samad terves rakenduses. Globaalse ressursi lisamiseks tuleb lisada sobiva nimega ressursifail kausta App_GlobalResources. Lisame näiteks ressursifaili asjad.resx ning iga vajaliku keele jaoks keele tähisega ressursifail asjad.en.resx, asjad.ru.resx jne.

```
<root>
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>2.0</value>
</resheader>
<resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<data name="jutt">
  <value>Mingi jutt</value>
</data>
</root>
```

Globaalse ressursi kasutamiseks tuleb teada nii ressurssi sisaldava faili nime kui ka ressursi nime. Kui see info teada, võime ressurssi kasutada ükskõik millise serveri kontrolli atribuudina. Näiteks lipiku teksti võime globaalsest ressursist lugeda järgmiselt:

```
<asp:Label ID="Lbl1" runat="server" Text="<%$ Resources:asjad, jutt %>" />
```

Kui on soov globaalset ressurssi lugeda programselt, saate kasutada GetGlobalResourceObject meetodit:

```
Lbl1.Text = GetGlobalResourceObject("asjad", "jutt").ToString();
```

Ressursse, nii lokaalseid kui ka globaalseid, on võimalik lisaks .resx failidele hoida ka andmebaasis või ükskõik millises teises andmehoidlas. Selleks tuleb ümber kirjutada lokaliseerimist pakkuv teenusepakkuja e. provider. Keda see võimalus huvitab, võib otsida sellekohaseid materjale nii otsingumootoritest kui ka erinevatest foorumitest. Ühe võimaliku lahenduse leiate ka lisast lk 469 „Ressursside hoidmine SQL Serveris".

Programselt keele muutmine

Selleks, et muuta keelt programselt, oleks esmalt vajalik tekitada mingi mehhanism keele valimiseks. Üks variant on kasutada ripploendit, kõige parem koht sellele ripploendile oleks pealeht (master page), kuid võime selle lisada ka igale .aspx lehele. Siinses näites kirjutame ripploendisse elemendid staatiliselt, kuid neid võiks lugeda ka andmebaasist.

```
<asp:DropDownList runat="server" ID="ddlKeel" AutoPostBack="true"
    OnSelectedIndexChanged="ddlKeel_SelectedIndexChanged"
    OnPreRender="ddlKeel_PreRender" >
    <asp:ListItem Text="Eesti" Value="et" />
    <asp:ListItem Text="English" Value="en" />
</asp:DropDownList>
```

Et ripploend hakkaks keelt vahetama, tuleb ära kasutada SelectedIndexChanged sündmus. Valitud keelt oleks hetkel kõige parem hoida sessiooni muutujas. Selleks, et raamistik saaks aru, et midagi muutus, palume lehekülje uuesti laadida:

```
protected void ddlKeel_SelectedIndexChanged(object sender, EventArgs e)
{
  Session["keel"] = ddlKeel.SelectedValue;
  Server.Transfer(Request.Url.PathAndQuery, false);
}
```

Server.Transfer teeb ümbersuunamise serveri tasemel e. kasutajale märkamatult.

Kuna lehe laadimine katkestatakse üsna varases faasis, ei õnnestu raamistikul kasutaja tehtud valikut salvestada. Seega tuleb kasutaja valik ise taastada. Selle koha peal aitab meid PreRender sündmus:

```
protected void ddlKeel_PreRender(object sender, EventArgs e)
{
    object oKeel = Session["keel"];
    if (oKeel != null)
        ddlKeel.SelectedValue = (string)oKeel;
}
```

Hetkel oleme loomulikult tegelenud vaid kasutaja soovide meelde jätmisega. Nüüd tuleks kirjutada ka meetod nende soovide realiseerimiseks. Keele muudatus on selline, mis tuleb paika panna lehe renderdamise varases faasis, sest sisuliselt kõik sõltub keelest. Selle tarbeks pakub raamistik 2.0 meile leheküljel meetodit InitializeCulture:

```
protected override void InitializeCulture()
{
    object oKeel = Session["keel"];
    if (oKeel != null)
    { // kui sessioonis on määratud, millist keelt kasutada
        string sKeel = oKeel.ToString().ToUpper();
        if (sKeel !=
            Thread.CurrentThread.CurrentUICulture.ToString().ToUpper())
        { // kui kehtiv keel ei ole sama mis valitud keel
            Thread.CurrentThread.CurrentUICulture =
                CultureInfo.CreateSpecificCulture(sKeel);
        }
    }
    else { // kui sessioonis keeleinfot veel ei ole
        Session["keel"] = Request.UserLanguages[0];
        Thread.CurrentThread.CurrentUICulture =
            new CultureInfo(Request.UserLanguages[0]);
    }
    Thread.CurrentThread.CurrentCulture =
        CultureInfo.CreateSpecificCulture(
                                Thread.CurrentThread.CurrentUICulture.Name);
}
```

Kuna meetod on kirjeldatud ka System.Web.UI.Page klassis, peame selle üle kirjutama sellest ka märksõna override meetodi kirjelduses. Edasi kontrollime, kas sessioonimuutujasse on kasutaja soov salvestatud. Kui ei, siis kasutame veebisirvija määranguid.

Ülesandeid

* Loo tervitusega leht. Tervituse tekst võetakse keelefailist

* Määra ressursifailis lisaks tervituse tekstile ka tervituse taustavärv

* Küsi andmeid programmi abil globaalsete ja lokaalsete ressursside failidest

Master Pages

Pealehed (Master Pages) on mõeldud rakenduses ühtse kujunduse või programsete elementide paigutuse hoidmiseks. Pealehed võimaldavad vältida või vähendada koodi dubleerimist ja lihtsustavad oluliselt rakenduse disaini ühtsustamist ning ka hilisemat haldamist. Lõppkasutajale on pealehed täiesti nähtamatud st leht, kus olete kasutanud pealehti, näeb veebisirvijas välja täpselt samasugune, kui leht, kus neid pealehti kasutatud ei ole. Tegemist on vahendiga, mis on mõeldud arendajatele produktiivsuse tõstmiseks.

Pealeht erineb tavalisest ASP.NET lehest selle poolest, et tema laiendiks on .master, lisaks muudele elementidele kirjeldatakse temas, kuhu võib panna programseid elemente ning kood pärineb klassist System.Web.UI.MasterPage.

Järgnevalt loomegi ühe lihtsa pealehe tavaline.master, kus näitame, kuhu tekivad leheküljе päis, menüü ning üks piirkond, kuhu aspx lehed saavad oma funktsionaalsuse paigutada.

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="tavaline.master.cs" Inherits="tavaline" %>
<!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>Minu esimene pealeht</title>
</head>
<body>
    <form id="form1" runat="server">
    <table>
        <tr>
            <td colspan = "2" >
                Siia tuleb lehekülje päis
            </td>
        </tr>
        <tr >
            <td>
                Siia ehitame menüü
            </td>
            <td>
                <asp:contentplaceholder id="sisu" runat="server" />
            </td>
        </tr>
    </table>
    </form>
</body>
</html>
```

Kuna hetkel sellel lehel mingit programset funktsionaalsust ei ole, siis on koodifail tavaline.master.cs sisuliselt tühi.

```
using System;
public partial class tavaline : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e) {}
}
```

Kui soovime, et meie default.aspx kasutaks seda pealehte, tuleb teha paar muudatust: lisada Page elemendi atribuutidesse MasterPageFile atribuut, nii et Page element näeb välja järgmine:

```
<%@ Page Language="C#" MasterPageFile="~/tavaline.master"
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
Title="Minu teine leht" %>
```

Viidates ASP.NET rakendustes teistele failidele, kasutatakse tihtipeale faili absoluutset aadressi rakenduse kausta suhtes. Rakenduse juurkaustale viitamiseks kasutatakse tildet „~” e. kui me vaatame ülal olevat näidet, siis paikneb fail tavaline.master rakenduse juurkataloogis.

Lisaks muudatusele lehekülje Page elemendi atribuutides, tuleb ära kustutada kõik, mis asub form elemendist väljaspool, kaasaarvatud form elemendi alustavad ja lõpetavad tagid. Selle asemele tulevad piirkonnad pealehelt. Seega näeb ülejäänud default.aspx välja järgmine:

```
<asp:Content ID="Content1" ContentPlaceHolderID="sisu" Runat="Server">
<asp:Label ID="Label1" runat="server" />
</asp:Content>
```

Lehekülje koodis mingeid muudatusi teha ei tule.

Kui soovite kasutada mõnda pealehel olevat elementi või omadust aspx lehel, siis leiate lehe küljes oleva pealehe Page.Master omaduse alt. Samas pole see omadus otse kasutatav. See tähendab, kui näiteks teie pealehel on omadus jutt, siis ei ole võimalik seda muuta näiteks sellise lausega Page.Master.jutt = „mingi jutt”. Selleks, et saada ligi pealehe omadustele ja sealolevatele elementidele, tuleb Master all olev objekt teisendada õigesse tüüpi. Seega kui teie pealehe taga oleva klassi nimi on MinuPealeht, siis tuleks seal oleva omaduse jutt muutmiseks kirjutada järgnevalt:

```
((MinuPealeht)Master).jutt = ”mingi jutt”;
```

http://video.msn.com/video.aspx?vid=daa5ce9b-8c0f-40c8-adbf-e71784c4d9cd

Elemendid lehel

Lehekülje loomisel on meil võimalus kogu töö teha ära kasutades ASPist tuntud Response.Write meetodit, genereerides rea kaupa kliendile mineva HTMLi, kuid saame kasutada ka erinevaid kontrolle.

Elemendid võimaldavad läheneda leheküljele objekt-orienteeritud põhimõtteid kasutades. Elemente, mida kasutada, on lugematul hulgal. Suur hulk neist on kaasas koos .NET raamistikuga, väga palju saab tõmmata Internetist (osad on tasuta, osad tasulised) ning loomulikult on neid võimalik ka ise kokku panna.

Loomulikult ei jõua me siinkohal vaadata kõiki võimalikke kontrolle väga detailselt, kuid püüame anda ülevaate olulisemast ning sellest, mida kus ja kuidas kasutada.

Standardsed serveri elemendid

Alustame standardsete ASP.NET elementidega. ASP.NET Server elemendid on põhilised abivahendid veebirakenduste loomisel. Serveri elementidel on loogiline ja lihtsalt kasutatav objektmudel ning lisaks sellele oskab raamistik neid sõltuvalt kliendi veebisirvija võimekusest renderdada erinevateks HTML tagideks.

Standardsed ASP.NET kontrollid tunnete ära sellejärgi, et nende nime eesliide on asp:. Kõigil kontrollidel peab kindlasti olema täidetud:

- runat=”server” atribuut, mis näitab, et kontroll tuleb renderdada serveris
- ID atribuut, mis on selle kontrolli unikaalne tunnus. ID kasutatakse kontrolli poole pöördumiseks nii märgistuses kui ka koodis. Kõik ID peavad lehekülje piires olema erinevad!

Järgnevalt standardsete serverielementidele lühikirjeldused koos näidetega:

AdRotator	`<asp:AdRotator runat="server" ID="adrLogod" AdvertisementFile="~/App_Data/sisu.xml" />` Näitab ühe kaupa, juhuslikus järjekorras andmeballikast leitud pilte. Kasulik nt banneri loomisel
BulletedList	`<asp:BulletedList ID="BulletedList1" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" />` Täppidega loend andmebaasis olevatest andmetest
Button	`<asp:Button ID="bNupp" runat="server" OnClick="bNupp_Click" Text="Vajuta mind" />` Nupud on mingite kasutaja poolt algatatavate tegevuste tegemiseks. Nupuvajutusele reageerimiseks tuleb realiseerida OnClick sündmus.
Calendar	`<asp:Calendar ID="cKalender" runat="server" />` Tekitab kalendri, mille pealt saab kasutaja valida kuupäeva. Kuupäeva kasutamiseks tuleb pöörduda kalendri `SelectedDate` atribuudi juurde
CheckBox	`<asp:CheckBox ID="cbYx" runat="server" Text="Vali mind" />`

	Tekitab märkeruudu, mille väärtust saate edaspidi kasutada läbi `Checked` atribuudi.
CheckBoxList	`<asp:CheckBoxList ID="cblValikud" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood" />` Tekitab märkeloetelu ruutude loetelu, kust on kasutajal võimalik valida mitmeid. Esimese valiku leidmiseks saate kasutada `SelectedIndex`, `SelectedItem` või `SelectedValue` atribuute. Kõigi valikute leidmiseks tuleb läbi käia Items kollektsioon ning kontrollida iga valiku `Checked` atribuuti.
DropDownList	`<asp:DropDownList ID="ddlValikud" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood" />` Ripploend andmeallikast leitud andmetest. Valitud väärtuse leidmiseks saab kasutada `SelectedIndex`, `SelectedItem` või `SelectedValue` atribuute.
FileUpload	`<asp:FileUpload ID="fuFail" runat="server" />` Võimaldab faile ülesse laadida. Kasutaja poolt valitud faili leiate `PostedFile` atribuudi alt. Faili saab kätte andmevoona `PostedFile.InputStream` atribuudist.
HiddenField	`<asp:HiddenField ID="hfPeidetud" runat="server" />` Peidetud väli, kus on võimalik hoida teksti, mida kasutajale näidata ei soovi.
HyperLink	`<asp:HyperLink ID="hlLink" runat="server" NavigateUrl="http://www.savisaar.ee" Text="koduleht" />` Hüperlink, kus lingina võib toimida nii tekst kui ka pilt.
Image	`<asp:Image ID="iPilt" runat="server" ImageUrl="~/images/Sample.jpg" />` Pilt, mis renderdub HTMLi IMG tagiks.
ImageButton	`<asp:ImageButton ID="ibNupp" runat="server"`

```
ImageUrl="~/images/Sample.jpg" OnClick="bNupp_Click" />
```

Pilt, mis reageerib hiire klikile e. töötab nagu tavaline nupp.

ImageMap

```
<asp:ImageMap ID="imJooksja" runat="server"
   ImageUrl="~/images/Sample.jpg">
   <asp:CircleHotSpot X="184" Y="38" Radius="20"
        PostBackValue="pea" HotSpotMode="PostBack" />
   <asp:RectangleHotSpot Left="160" Top="90" Right="283"
        Bottom="212" PostBackValue="parempool"
        HotSpotMode="PostBack" />
   <asp:PolygonHotSpot PostBackValue="jooksja"
        Coordinates="0,0,105,0,108,68,150,102,150,
        111,135,116,85,112,64,18,49,41,61,139,107,175,103,
        194,51,202,0,85" HotSpotMode="PostBack" />
</asp:ImageMap>
```

ImageMap tekitab pildi, millel on erinevaid piirkondi, kuhu kasutaja saab klõpsata. Klõpsatavate piirkondade tekitamiseks tuleb ImageMap elemendi sisse panna HotSpot alamelemendid. CircleHotSpot tekitab ringi kujulise piirkonna, RectangleHotSpot tekitab ristkülikukujulise piirkonna, PolygonHotSpot tekitab hulknurgakujulise piirkonna.

Kui keegi klikib mingil piirkonnal siis on võimalik, kas suunata nii uuele aadressile kui ka programselt klikki hallata. Programse haldamise jaoks tuleb ära realiseerida Click sündmus, kus on võimalik vaadelda parameetriga kaasa antud PostBackValue omadust

---

Label

```
<asp:Label ID="lblJutt" runat="server" Text="Mingi tekst" />
```

Label on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.

Labeli peale saab panna staatilist teksti ja ka lihtsamaid HTMLi tage kuid pole võimalik lisada dünaamiliselt serveri kontrolle.

	Label renderdub veebisirvijas SPAN tagiks.
LinkButton	`<asp:LinkButton ID="lbNupp" runat="server" Text="Vajuta mind" OnClick="bNupp_Click" />` Hüperlink, mis töötab nupuna e. kui hiirega klõpsata ei minda mitte uuele lehel vaid tekitatakse Click sündmus.
ListBox	`<asp:ListBox ID="lbLoend" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood"/>` Loend väärtustest, kust sõltuvalt seadistusest on võimalik valida, kas üks või mitu väärtust e. sõltuvalt seadistusest käitub analoogselt RadioButtonList'iga või CheckBoxList'iga. Kõige olulisem erinevus seisneb selles, et pikk loetelu on võimalik kokku suruda väga väiksele alale.
Literal	`<asp:Literal ID="lKoht" runat="server" />` Literal on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder. Literal erineb Label elemendist selle poolest, et ta ei lisa dünaamilise sisu ümber mingit lisamärgistust. Sellest tulenevalt ei ole võimalik sisu stiilidega kujundada.
Localize	`<asp:Localize ID="lzKoht" runat="server" />` Erinevalt Literal kontrollist võimaldab Localize kontroll reserveerida koha lokaliseeritud sisu tarbeks. Muus osas on need kaks kontrolli täiesti ühesugused. Erinevalt Label kontrollist ei lisa Localize sisu ümber mingit märgistust ega võimalda ka sisu kujundamist.

MultiView	`<asp:MultiView ID="mvVaade" runat="Server"`
	`ActiveViewIndex="0">`
	`<asp:View ID="v1" runat="server">`
	`Mingid asjad, mida näidata esimeses vaates`

```
    </asp:View>

    <asp:View ID="v2" runat="server">

       Mingid teistsugused asjad teises vaates näitamiseks

    </asp:View>

</asp:MultiView>
```

MultiView võimaldab teha leheküljel oleva info vaatamiseks erinevaid vaateid. Vaateid saab vahetada saab valida nii märgistuses kui ka programselt ActiveViewIndex atribuudi abil. Sisuliselt võimaldab MultiView integreerida ühele ASP.NET lehele mitmeid analoogseid veebilehti.

Panel

```
<asp:Panel ID="pKoht" runat="server" />
```

Panel on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.

Erinevalt Label ja Literal kontrollidest on võimalik sinna sisse paigutada ka teisi serveri kontrolle.

Panel kontroll paneb kogu dünaamilise sisu kas div või table elemendi sisse.

PlaceHolder

```
<asp:PlaceHolder ID="phKoht" runat="server" />
```

PlaceHolder on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.

PlaceHolder võimaldab analoogselt Panel kontrolliga lisada dünaamilisi teisi serveri kontrolle kuid erinevalt Panel elemendist mingit lisamärgistust nende ümber ei lisata.

RadioButton

```
<asp:RadioButton ID="rb1" runat="server" GroupName="g1"
Text="Valik1" />
```

Raadionupp võimaldab teha ühe valiku ette antud loetelust. Kui lehel on mitmeid loetelusid mitu on võimalik raadionupud jagada gruppidesse.

RadioButtonList

```
<asp:RadioButtonList ID="rblValikud" runat="server"
```

```
DataSourceID="xmlDSLoend" DataTextField="tekst"
DataValueField="kood" />
```

RadioButtonList võimaldab andmebaasis olevatest andmetest tekitada raadionuppudega loendi, kust saab valida ühe valiku.

Substitution

```
<asp:Substitution ID="sDynKoht" runat="server"
MethodName="DynSisu" />
```

Substitution kontroll märgistab ära koha, mida puhverdatud (cached) lehel dünaamiliselt uuendatakse. Dünaamilise sisu peab tekitama staatiline, meetod, mis tagastab stringi.

Table

```
<asp:Table ID="t1" runat="Server" >

   <asp:TableHeaderRow ID="th" runat="server">

      <asp:TableHeaderCell id="th1" runat="server">

         A veerg

      </asp:TableHeaderCell>

      <asp:TableHeaderCell id="th2" runat="server">

         B veerg

      </asp:TableHeaderCell>

   </asp:TableHeaderRow>

   <asp:TableRow ID="tr1" runat="server">

      <asp:TableCell ID="a1" runat="server">

         Lahter A1

      </asp:TableCell>

      <asp:TableCell ID="b1" runat="server">

         Lahter B1

      </asp:TableCell>

   </asp:TableRow>
```

```
   <asp:TableRow ID="tr2" runat="server">
      <asp:TableCell ID="a2" runat="server">
         Lahter A2
      </asp:TableCell>
      <asp:TableCell ID="b2" runat="server">
         Lahter B2
      </asp:TableCell>
   </asp:TableRow>
</asp:Table>
```

Table kontroll võimaldab tekitada tabeli, mida on võimalik väga lihtsalt programselt hallata. Eriti kasulik on see struktuur juhul kui soovite tabelit moodustada koodi abil, mitte lehekülje märgistuses.

TextBox

```
<asp:TextBox ID="txtInf" runat="server" />
```

TextBox on kast, kuhu kasutaja saab sisestada mingit teksti. Tekstikaste on võimalik teha nii ühe kui ka mitmerealisi ning peidetud tekstiga paroolikaste. Sisestatud teksti saab hiljem kätte Text atribuudi alt.

View

```
<asp:View ID="v1" runat="server">
   Mingid asjad, mida näidata esimeses vaates
</asp:View>
```

View kontrolle saab lisada ainult MultiView kontrolli sisse. Tegemist on ühega paljudest vaadetest MultiView sees.

Wizard

```
<asp:Wizard ID="wVolur" runat="server" ActiveStepIndex="1">
   <WizardSteps>
      <asp:WizardStep runat="server" Title="Samm 1">
         Mingi esimese sammu sisu
```

```
        </asp:WizardStep>

        <asp:WizardStep runat="server" Title="Samm 2">

           mingi teise sammu sisu

        </asp:WizardStep>

    </WizardSteps>

</asp:Wizard>
```

Võlur on sarnane MultiView kontrolliga võimaldades luua samadest andmetest mitmeid vaateid. Erinevalt MultiView'st on võluris ette määratud liikumise loogika ning info paigutus.

Võlureid on võimalik kasutada nii töövoogude (WorkFlow) kui ka pikemate vormide juures, kus on võimalik infot jagada mitmele lehele.

Võlur on mõeldud suuremahuliste sisestuste haldamiseks, näiteks küsitlused. Selle asemel, et kõik väljad koondada ühele lehele, võimaldab võlur tekitada mitu lehekülge andmete sisestamiseks, mille vahel võib kasutaja liikuda vastavalt vajadusele, kas lehtede järjekorras või vabal valikul.

Xml

```
<asp:Xml ID="Xml1" runat="server"
DocumentSource="~/App_Data/sisu.xml" />
```

Xml kontroll võimaldab veebilehel kuvada XML andmeid ja seda nii toorel kujul kui ka teisendatud kujul.

Loe lisaks ka XMLi peatükki.

Dünaamiliselt sisu tekitamisel on soovitus eelistada Literal või Localize kontrolli Labelile ning PlaceHolder kontrolli Panel kontrollile.

Programselt hallatavad HTML elemendid

Kui HTML elemendile lisada atribuut runat="server" siis on HTML elemendid programmis kasutatavad analoogsed Serveri elementidega. Erinevus võrreldes serveri elementidega seisneb selles, et nende objektmudel on moodustatud DHTMLi baasil ning nende haldamine ei ole nii lihtne kui serveri elementide puhul. Samuti ei oska .NET neile pakkuda alternatiivseid renderdusi sõltuvalt veebisirvija võimekusest.

Sisendi kontrollimise elemendid (Validators)

Kõige rohkem tööajal tekkivatest vigadest ja ka rakenduses olevatest turvaaukudest on tingitud kasutajate poolt tehtavatest ootamatutest sisestustest. Validation kontrollid on mõeldud kasutaja poolt sisestatud info kontrollimiseks ning on väga olulised vormide ehitamisel.

Kontrollimine käib enamasti kahes etapis:

- Kontrollitakse, kliendi poolel ehk veebisirvijas, javascripti abil, kas andmed on korrektsed. See võimaldab anda kasutajale märksa kiiremaid vastuseid kui tehes sama kontrolli serveris.

- Kontrollitakse serveris, kas kõik on korrektne. Kaitseb teie veebirakendust vigaste sisestuste eest. Kui kasutaja väga tahab, siis võib ta JavaScripti välja lülitada ning sellisel juhul kliendipoolne kontroll ei toimi. Teile on aga oluline, et andmed oleksid korrektsed ennem kui Te hakkate neid edasi töötlema. Seega on serveri poole kontroll väga oluline. Enamgi veel - kliendi poolsest kontrollist on võimalik loobuda aga serveris tuleb alati kontrollida, muidu muutub loodav veebrakendus väga ebastabiilseks ning ohtlikuks.

Kõigil validaatoritel on mõned olulised omadused, mis peaks olema alati määratud:

- Display – kas kontrolli positsioonil reserveeritakse veateate jaoks ruum või mitte

- ControlToValidate – kontroll, mida validaatoriga valideerida

- ErrorMessage – täispikk veateade. Näidatakse juhul kui kontroll ebaõnnestub

- SetFocusOnError – kas vea puhul muudetakse vea põhjustanud kontroll aktiivseks

- Text – lühike veakirjeldus. Näidatakse juhul kui kontroll ebaõnnestub

- ValidationGrupp – juhul kui samal lehel on mitu iseseisvat andmekogumit on võimalik ka kontrollid jagada gruppidesse ning kontrollida iga gruppi eraldi.

RequiredFieldValidator	Kontrollib, et väli oleks täidetud.
RangeValidator	Tunneb erinevaid andmetüüpe ning kontrollib, et sisestatud väärtus oleks etteantud vahemikus.
RegularExpressionValidator	Kontrollib sisestuse sobivust regulaaravalisega
CompareValidator	Võrdleb sisestust teise kontrolli sisuga. Kasulik nt paroolivahetuse vormi juures, kus tuleb uut parooli sisestada

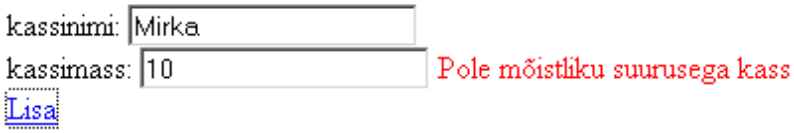
	kaks korda
CustomValidator	Ise tehtud validaator. Kontrollib täpselt nii nagu ise soovid. Validaatorit ehitades tuleb kindlasti realiseerida ServerValidate sündmus ning kui soovite ka veebisirvijas kontrolli rakendada siis JavaScripti abil ka kliendi poolne kontroll. JavaScripti funktsiooni nime saate sisestada ClientValidationFunction atribuudile.
ValidationSummary	Kokkuvõte vigadest. Kui lehel on ValidationSummary kontroll siis validaatori positsioonil näidatakse lühikest veakirjeldust (Text atribuut) ning ValidationSummary näitab pikka vea kirjeldust (ErrorMessage atribuut).

Selgitavaks näiteks siia juurde ülalpool andmebaasipõhise rakendusena kasside lisamislehe juurde kuuluv kontroll, et kassinimi oleks olemas ning kassi mass kassile sobilikus vahemikus.

```
<InsertItemTemplate>
    kassinimi:
    <asp:TextBox ID="kassinimiTextBox" runat="server"
        Text='<%# Bind("kassinimi") %>' />
        <asp:RequiredFieldValidator
             ControlToValidate="kassinimiTextBox"
             ErrorMessage="Kassi nimi puudub"
             runat="server"
             ID="nimeolemasolukontrollija"/>
       <br />
       kassimass:
       <asp:TextBox ID="kassimassTextBox" runat="server"
           Text='<%# Bind("kassimass") %>' />
       <asp:RangeValidator ID="RangeValidator1" runat="server"
            ControlToValidate="kassimassTextBox" MinimumValue="100"
            MaximumValue="10000"
            ErrorMessage="Pole mõistliku suurusega kass"/>
    <br />
    <asp:LinkButton ID="InsertButton"
        runat="server" CausesValidation="True"
        CommandName="Insert" Text="Lisa" />
</InsertItemTemplate>
```

Nagu näha, ei saa lisamata jätta kassi nime.

Samuti peab olema kassi mass mõistlikus vahemikus nagu ülalt koodist MinimumValue ja MaximumValue vahelised piirid määravad.



Ülesandeid

* Katseta mitmesuguseid kontrolle.

* Püüa nende väärtusi programmiga lugeda ja kätte saada.

* Kontrolli validaatori abil, et tekstiväli oleks täidetud

Navigeerimiselemendid

Navigeerimiseks mõeldud serveri kontrollid pakuvad mugavaid veebirakenduses navigeerimise võimalusi. Navigaatoreid on kokku kolm.

SiteMapPath

```
<asp:SiteMapPath ID="smpKoht" runat="server"/>
```

SiteMapPath näitab kasutajale loogilist paiknemist veebirakenduse struktuuris. Paiknemist näidatakse analoogselt URLile või failisüsteemis kataloogiteekonnale.

SiteMap kasutamiseks tuleb veebi juurkataloogi lisada web.sitemap fail, milles kirjeldate ära veebi loogilise struktuuri. Web.sitemap on XML fail, mis vastab SiteMap-File skeemile (schema). Faili sisu võiks siis välja näha midagi sellist:

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

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-
File-1.0" >

  <siteMapNode url="~/Default.aspx" title="Avaleht">

    <siteMapNode url="~/Default1.aspx" title="1. katse" />
```

```
    <siteMapNode url="~/Default2.aspx" title="2. katse" >
      <siteMapNode url="~/Default3.aspx" title="variant 1" />
      <siteMapNode url="~/Default4.aspx" title="variant 2" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

Sitemap failis peab ära kirjeldama kõikide lehekülgede loogilise positsiooni, kuhu kasutaja võib sattuda. Vajadusel saab seal pakkuda isegi erinevaid lahendusi vastavalt URLis nähtavatele parameetritele.

Kuna veebisirvija Back nupu programne haldamine on väga keeruline siis kasutatakse tagasi liikumise võimaldamiseks enamasti SiteMapPath kontrollil baseeruvat struktuuri. Mõnikord kutsutakse sellist navigeerimisvahendit ka leivapuruks (kunstmuinasjutu järgi, kus leivapuru järgi püüti koju tagasi minna).

Menu

```
<asp:Menu ID="menyy" runat="server">
    <Items>
        <asp:MenuItem Text="Avaleht" />
        <asp:MenuItem Text="Mingid asjad" >
            <asp:MenuItem Text="Variant 1" />
            <asp:MenuItem Text="Variant 2" />
        </asp:MenuItem>
        <asp:MenuItem Text="Veel midagi" />
    </Items>
</asp:Menu>
```

Menüü võimaldab kasutajal lihtsalt leida veebirakenduses olevat informatsiooni. Menüü sisu on võimalik sisestada nii staatiliselt (nagu on näha ülalolevast näitest), kui ka lugeda andmebaasist.

TreeView

```
<asp:TreeView ID="TreeView1" runat="server">
  <Nodes>
    <asp:TreeNode Text="Avaleht" />
    <asp:TreeNode Text="Mingid asjad" >
      <asp:TreeNode Text="Variant 1" />
      <asp:TreeNode Text="Variant 2" />
    </asp:TreeNode>
    <asp:TreeNode Text="Veel midagi" />
  </Nodes>
</asp:TreeView>
```

TreeView võimaldab vaadelda hierarhilisi andmeid puu kujul. Puud on võimalik moodustada nii staatiliselt (nagu näha ülal olevast näitest) kui ka moodustada dünaamiliselt andmebaasis olevate andmete järgi.

Ülesandeid

* Koosta menüüga pealeht (master) ning sealt viited mitmele alamlehele

* Lisa lehestikus asukohta näitav SiteMap

* Paiguta lehestiku lehed mõttes puukujulisse struktuuri. Aita navigeerimisel kasutajat TreeView-ga

Omaloodud elemendid (UserControl)

Lisaks pealehtedele saab programmiloogikat jagada ka läbi isetehtud graafilise kasutajaliidese elementide (User Controls) e. kasutaja kontrollide (KK). KK on ASP.NET leht ASP.NET lehe sees. KK abil saab kapseldada mingite graafilise liidese elementide kooslust ning funktsionaalsust.

Näiteks lisame oma veebilehele lisaks sildile veel tekstikasti ja nupu ning ütleme, et me soovime reageerida sündmusele, kui keegi klikib sellel nupul. Peale seda väikest täiendust näeb default.aspx content elemendi sisu välja järgmine:

```
<asp:TextBox ID="txtKast" runat="server" /><br />
```

```
<asp:Button ID="bOK" runat="server" Text="Klõpsa siia" OnClick="bOK_Click"
/><br />
<asp:Label ID="lblInf" runat="server" />
```

Nüüd tuleb teha ka väikesed täiendused lehekülje koodis default.aspx.cs. Esmalt muudame ära Page_Load sündmuse, kus ütleme, et kui sellele leheküljele tullakse 1. korda, siis tuleb labeli sisu ära kustutada. Kui sama sessiooni ajal tullakse lehele 2st, 3ndat jne korda, siis ei tehta midagi. Lehekülje objekti küljes olev omadus IsPostBack on true, kui ei olda lehel esimest korda.

```
protected void Page_Load(object sender, EventArgs e)
{
if (!Page.IsPostBack)
lblInf.Text = "";
}
```

Lisaks peame tekitama protseduuri nupu klikile reageerimiseks. Kui keegi klikib nupule, siis muudame ära labeli teksti vastavalt tekstikastis olevale jutule.

```
protected void bOK_Click(object sender, EventArgs e)
{
lblInf.Text = "Sa kirjutasid tekstikast: " + txtKast.Text;
}
```

Kui me nüüd avastame, et sellist tekstikasti, labeli ja nupu interaktiivset kooslust läheb meil ka mujal vaja, siis on mõistlik see funktsionaalsus realiseerida KK abil. KK lehed e. .ascx lehed on oma ülesehituselt väga sarnased .aspx lehtedega seega on üsna tavaline, et esmalt realiseeritakse mingi funktsionaalsus .aspx lehel ning seejärel teisendatakse see leht KK-ks.

KK-l on võrreldes .aspx lehega 4 erinevust:

- faililaiend on .ascx
- Page element on asendatud Control elemendiga, millel on peaaegu samad atribuudid.
- kuna tegemist on osaga lehest, siis ei saa seal kasutada ei form elemente ega content elemente
- KK taga olev klass päritakse System.Web.UI.UserControl klassist

Seega võiks meie kolme komponendiga kk.ascx näha välja järgmine:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="kk.ascx.cs"
Inherits="kk" %>
<asp:TextBox ID="txtKast" runat="server" /><br />
<asp:Button ID="bOK" runat="server" Text="Klõpsa siia" OnClick="bOK_Click"
/><br />
```

```
<asp:Label ID="lblInf" runat="server" />
```

Ning kood selle KK juures (failis kk.ascx.cs) näeks välja selline:

```
using System;
using System.Web.UI;
public partial class kk : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
            lblInf.Text = "";
    }
    protected void bOK_Click(object sender, EventArgs e)
    {
        lblInf.Text = "Sa kirjutasid tekstikast: " + txtKast.Text;
    }
}
```

Selleks, et oma värskelt loodud KK'd kasutada, peame selle paigutama mõnele aspx lehel. Kuna tegemist ei ole raamistikku kuuluva elemendiga, tuleb see esmalt registreerida. Selleks lisame lehekülje algusesse peale Page elementi ning enne igasuguseid teisi elemente Register elemendi, milles ütleme, millises failis meie KK paikneb, ning kuidas me soovime temale viidata.

```
<%@ Register Src="kk.ascx" TagName="kk" TagPrefix="kool" %>
```

Ning kohas, kus see KK peab paiknema, lisame vastava kirjeldusega elemendi:

```
<kool:kk ID="Kk1" runat="server" />
```

Antud juhul võiks selleks kohaks olla Content element default.aspx lehel. Kuna kogu eelnev funktsionaalsus sai KK'sse üle viidud, siis võiks kogu ülejäänud sisu Content elemendi seest ära kustutada. Ühtlasi tuleks kustutada ka kogu kood default.aspx.cs seest e. alles jääb vaid tühi klass.

```
public partial class _Default : System.Web.UI.Page{}
```

Sama KK'd võib ühel ja samal lehel kasutada mitmes eksemplaris ning samuti võime selle KK registreerida nii mitmele lehele kui soovime.

Seega võiks kokkuvõtteks öelda, et läbi pealehtede MasterPage ühtlustame üldise elementide paigutuse aspx lehtedel. KK abil kapseldame kasutajaliidese elementide kooslust.

Ülesandeid

* Kasuta näites loodud kasutajakontrolli oma lehel

* Loo kontroll kahe arvu korrutamiseks. Katseta.

Veebilehtede kujundamine kasutades nägusid (Themes)

Üldiselt on tuntud tõde, et hea programmeerija ei ole tavaliselt hea disainer ning hea disainer ei ole enamasti hea programmeerija. Luues aga veebirakendust, on vaja, et hästi oleks tehtud nii disain kui ka funktsionaalsus. See tähendab aga seda, et ühe veebilehega tegelevad korraga kaks inimest. Juba ASP.NET esimeses versioonis oli võimalik kood ja kujundus üksteisest eraldada (aspx fail kujunduse ja paigutuse jaoks, .cs fail koodi jaoks), kuid sellel lähenemisel oli kaks nõrka kohta:

- Läbi aspx faili ei saa kujundada dünaamiliselt tekitatud objekte

- Ühtse kujunduse hoidmine on raske, kuna kujundust tuleb igas aspx failis iga kontrolli juures uuesti korrata.

Üks lahendus on kasutada vana head stiilifaili .css, kuid raamistiku poolt genereeritud HTML objektide stiiliga kujundamine on üsna raske. Raskeks teeb selle asjaolu, et genereeritud struktuurid võivad tulla väga keerulised ning aeg-ajalt on väga raske ennustada, milliseks HTML objektiks üks või teine kontroll renderdub.

Lahendusena neile probleemidele pakub ASP.NET 2.0 välja näod e. Theme'd. Ühe näo alla koondatakse hulk kujundusi, mida on mugav tervele lehele peale panna. Nägu koosneb kahte liiki kujundustest:

- nahad (skin), on mõeldud serveri kontrollide kujundamiseks

- stiilid (css), on mõeldud HTML tagide kujundamiseks

Kõik näod tuleb paigutada veebi juurikas asuvasse App_Themes kausta. Sinna tuleb teha alamkaust iga vajaliku näo jaoks. Loodud alamkaustadesse saate hakata paigutama .skin ja .css faile.

Loome nt näo nimega DefKujundus ning selle alla naha nimega nahk.skin. Naha sisse saate kopeerida .aspx failist kogu kujunduse. Näiteks on meil vaja kujundust tabelivaatele ja kahele erinevale kalendrile. Sellisel juhul võiks nt nahk.skin näha välja järgmine:

```
<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >
    <AlternatingRowStyle BackColor="Blue" />
</asp:GridView>
<asp:Calendar runat="server" BackColor="White" BorderColor="Black"
        BorderStyle="Solid" CellSpacing="1" Font-Names="Verdana"
        Font-Size="9pt" ForeColor="Black" Height="250px"
```

```
        NextPrevFormat="ShortMonth" Width="330px">

    <SelectedDayStyle BackColor="#333399" ForeColor="White" />
    <TodayDayStyle BackColor="#999999" ForeColor="White" />
    <DayStyle BackColor="#CCCCCC" />
    <OtherMonthDayStyle ForeColor="#999999" />
    <NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="White" />
    <DayHeaderStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"
        Height="8pt" />
    <TitleStyle BackColor="#333399" BorderStyle="Solid" Font-Bold="True"
        Font-Size="12pt" ForeColor="White" Height="12pt" />
</asp:Calendar>
<asp:Calendar runat="server" SkinId="punaneKalender" BackColor="White"
    BorderColor="Black" BorderStyle="Solid" CellSpacing="1"
    Font-Names="Verdana" Font-Size="9pt" ForeColor="Black" Height="250px"
    NextPrevFormat="ShortMonth" Width="330px">

    <SelectedDayStyle BackColor="#333399" ForeColor="White" />
    <TodayDayStyle BackColor="#999999" ForeColor="White" />
    <DayStyle BackColor="#CCCCCC" />
    <OtherMonthDayStyle ForeColor="#999999" />
    <NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="White" />
    <DayHeaderStyle Font-Bold="True" Font-Size="8pt" ForeColor="red"
        Height="8pt" />
    <TitleStyle BackColor="#333399" BorderStyle="Solid" Font-Bold="True"
        Font-Size="12pt" ForeColor="White" Height="12pt" />
</asp:Calendar>
```

Nagu näha, on nahas olevad serveri elemendid peaaegu samasugused kui .aspx lehel. Ainukeseks erinevuseks on see, et kogu funktsionaalne osa on eemaldatud, so igasugused meetodid sündmustele reageerimiseks jne. Kujundusi saame teha kahte moodi:

- peakujundus – serveri kontroll, millel puudub kujunduse ID
- spetsiaalkujundus – serveri kontroll, millele on lisatud atribuut SkinID. Vastavad SkinIDd peavad olema samad nii aspx failis kui ka .skin failis.

Kui me soovime seda nahka kasutada oma .aspx lehe kujundamisel, siis tuleb meil:

- lisada lehekülje päisesse (Page element) atribuut Theme=”DefKujundus”
- kustutada kõigilt serveri kontrollidelt kõik kujundusega seonduv
- kontrollidele, mis vajavad spetsiaalkujundust, lisame atribuudi SkinID

Tulemusena on ilus ja lihtne aspx fail, kus on ainult funktsionaalsust puudutavad määrangud.

```
<%@ Page Language="C#" MasterPageFile="~/tavaline.master"
        AutoEventWireup="true" CodeFile="Default.aspx.cs"
        Inherits="_Default" Title="Minu teine leht" Theme="DefKujundus"%>
<asp:Content ID="Content3" ContentPlaceHolderID="cphSisu" Runat="Server">
    <asp:TextBox ID="txtAasta" runat="server" AutoPostBack="True"
```

```
        OnTextChanged="txtAasta_TextChanged"></asp:TextBox><br />
    <asp:Calendar ID="Cal1" runat="server" />
    <asp:Calendar ID="Cal2" SkinID="punaneKalender" runat="server" />
    <asp:SqlDataSource ID="andmeallikas" runat="server"
        ConnectionString="<%$ ConnectionStrings:yhendusTxt %>"
        ProviderName="<%$ ConnectionStrings:yhendusTxt.ProviderName %>"
        SelectCommand="SELECT TOP 100 ToodeID, Nimi, Hind FROM Toode" />
    <asp:GridView ID="gwToode" runat="server" AutoGenerateColumns="true"
        DataSourceID="andmeallikas" />
</asp:Content>
```

Ülesandeid

* Lisa Themes kausta kujundus SinineKujundus, milles määratakse nupu (asp:Button) värv siniseks.

* Katseta seda oma lehel, määrates näo (Theme) lehe algul

* Lisa teine kujunduskaust PunaneKujundus, kus nupu värv määratakse punaseks

* Veendu, et kujunduse vahetamisel muutub nupu värv.

* Püüa kujundust muuta programmi abil (Page.Theme="PunaneKujundus"). Seda on võimalik teha lehe preInit meetodi juures.

Väärtuste tööaegne meelespidamine

Muutuvate väärtuste meeles pidamiseks on lugematul hulgal erinevaid võimalusi alustades lokaalsetest muutujatest ning lõpetades andmebaasiga. Järgnevalt vaatleme vahendeid, mida pakub ASP.NET

Veebi globaalsed muutujad (Application variables)

Läbi Application klassi on võimalik jagada infot kõikjale terves veebirakenduses: see info on saadaval kõigil lehtedel terves rakenduses sõltumata sellest, milline kasutaja neid vaatab.

Application klassi väärtuse salvestamine käib järgmiselt:

```
Application[”ProgrammiMuutujaNimi”] = ”Mingi väärtus”;
```

Lugemine käib loomulikult vastupidises omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole seda muutujat veel tekitatud või ei ole sinna veel ühtegi väärtust salvestatud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

```
Object oAppVar = Application[”ProgrammiMuutujaNimi”];
if (oAppVar != null)
      Label1.Text = (string)oAppVar;
```

Sellised globaalsed muutujad on kasulikud jooksva statistika nt aktiivsete külastajate arvu jms tarbeks.

PS! Kõiki globaalseid muutujaid hoitakse arvuti mälus e. nendes muutujates ei tohiks hoida suuri andmehulki nagu nt tabelid jms

Andmete puhverdamine (Cache variable)

Application klass ei ole hea koht suurte andmehulkade hoidmiseks, kuna see raiskab palju arvuti mälu.

Cache klassi kaudu on samuti võimalik jagada infot kõikjale terves veebirakenduses: see info on saadaval kõigil lehtedel terves rakenduses sõltumata sellest, milline kasutaja neid vaatab. Info säilib Caches kuni arvutil jätkub ressurssi info hoidmiseks või kuni teie poolt määratud ajani.

Kõige lihtsam meetod Cache väärtuse salvestamiseks on järgmine:

```
Cache[”CacheMuutujaNimi”] = ”Mingi väärtus”;
```

Sellisel kujul salvestatud info säilib mälus kuni ressurssi jätkub, st kui arvuti vaba mälu hulk muutub kriitiliseks, visatakse vähem kasutatud andmed mälust välja. Kui soovite neid andmeid veel kasutada, tuleb andmed uuesti mällu laadida.

Mälust kustutamise aega saame ka ette määrata. Selleks on kaks võimalust: absoluutne aegumise aeg ja dünaamiliselt muutuv aeg.

Absoluutse aegumise aja kasutamiseks tuleb väärtuse Cache salvestamiseks kasutada Cache.Add või Cache.Insert meetodit.

Järgnevalt lisame info cache' i täpselt kümneks minutiks:

```
Cache.Insert("CacheMuutujaNimi", "Mingi väärtus", null,
DateTime.Now.AddMinutes(10),
System.Web.Caching.Cache.NoSlidingExpiration);
```

Selles näites salvestame infot cache' i nii, et info säilib 10 minutit viimasest kasutamisest: esmalt säilib info 10 minutiks salvestamisest, kui keegi selle aja jooksul infot kasutab, säilib see veel 10 min jne. Kui keegi pole 10 minuti jooksul infot kasutanud, info kustub.

```
Cache.Insert("CacheMuutujaNimi", "Mingi väärtus", null,
System.Web.Caching.Cache.NoAbsoluteExpiration,
TimeSpan.FromMinutes(10));
```

Lugemisel tuleb arvestada, et cache’ is olev info võib kaduda ning tuleks alati mõelda välja kaks sündmuste käiku: juhuks kui info on cache' is ja juhuks kui ei ole.

```
Object oCacheVar = Cache[”CacheMuutujaNimi”];
if (oCacheVar != null){
      // info on Caches ning kasutame seda
      Label1.Text = (string)oCacheVar;
} else {
      // infot ei ole Caches seega otsime info kusagilt mujalt
      // lisaks kasutamisele salvestame selle ka Cachesse
      String jutt = ”Mingi väärtus”;
      Cache[”CacheMuutujaNimi”] = jutt;
      Label1.Text = jutt;
}
```

Veebilehtede puhverdamine (OutputCache atribute)

Lisaks oma info puhverdamisele on võimalik puhvrisse salvestada ka renderdatud veebilehti. Veebilehtede puhverdamine tõstab oluliselt veebirakenduse jõudlust, kuna raamistik ei pea neid lehti uuesti genereerima, vaid saab kasutada varem loodut. Samas tuleb selle meetodiga olla äärmiselt ettevaatlik, kuna puhverdamine tähendab, et uut infot lehele ei tule. Seega ei tohiks seda kasutada kiiresti muutuvat infot sisaldavate lehtede peal.

Puhverdamiseks tuleb veebilehe .aspx faili või ka kasutaja kontrolli ascx algusesse lisada OutputCache element, kus näitate ära, kui kaua tuleb infot Caches hoida ning kuidas seda hoitakse

Näide 1: Hoiame infot puhvris 100 sekundit ning info on sama sõltumata parameetritest, millega lehe poole pöördutakse.

```
<%@ OutputCache Duration="100" VaryByParam="none" %>
```

Näide 2: Hoiame infot puhvris 100 sekundit ning infost on üks eksemplar iga pöördunud veebisirvija ja vormil olevate väärtuste koha st. kui üks kasutaja kasutab IE’d ja teine Mozillat, siis salvestatakse puhvrisse üks eksemplar mõlemale sirvijale saadetud lehest. Kui nt IE kasutaja muudab vormil infot või lisab päringuteksti ?id=2, siis salvestatakse lehest uus koopia vastavalt kasutatud parameetritele.

```
<%@ OutputCache Duration="100" VaryByParam="*" VaryByCustom="Browser" %>
```

Seansi muutujad (Session variables)

Läbi Session klassi on võimalik jagada infot kõikjale ühe seansi (sessiooni) piires. See info on saadaval kõigil lehtedel, kuhu kasutaja antud külastuse vältel satub. Seega on Session klass hea kasutaja staatust puudutava info hoidmiseks.

Session klassi väärtuse salvestamine käib järgmiselt:

```
Session[”SessiooniMuutujaNimi”] = ”Mingi väärtus”;
```

Lugemine käib loomulikult vastupidises järjestus omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole te seda muutujat veel tekitanud või ei ole sinna veel ühtegi väärtust salvestanud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

```
Object oSessVar = Session[”SessiooniMuutujaNimi”];
if (oSessVar != null)
      Label1.Text = (string)oSessVar;
```

Lehelkülje seisund (ViewState variables)

Läbi ViewState klassi on võimalik säilitada infot ühe lehekülje piires. ViewState klassi on mõtet salvestada infot, mida on vaja programmis pruukida, kuid mida ei soovi veebilehel kasutajale näidata. Sellist asja on võimalik korraldada kahte moodi:

- Kasutada hidden välju (input type=”hidden”)
- Kasutada ViewState klassi

ViewState klassil on peidetud välja ees kaks eelist:

- ise ei pea mingit välja tegema ja haldama
- Info on kergelt krüpteeritud, st kui kasutaja vaatab lehe lähtekoodi (View\Source), siis ta ei loe sealt midagi välja.

ViewState klassi väärtuse salvestamine käib järgmiselt:

```
ViewState[”LeheMuutujaNimi”] = ”Mingi väärtus”;
```

Lugemine käib loomulikult vastupidises järjestus omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole te seda muutujat veel tekitanud või ei ole sinna veel ühtegi väärtust salvestanud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

```
Object oVwVar = ViewState[”LeheMuutujaNimi”];
if (oVwVar != null)
      Label1.Text = (string)oVwVar;
```

Ülesandeid

* Pane iga salvestusmeetodi (Application, Cache, Session, ViewState) juurde muutuja, mille väärtust suurendatakse igal pöördumisel. Vaata tulemusi ning püüa kindlaks teha sarnasused ja erinevused.

Veebisaidi turvamine

Kuna maailm on muutunud väga vaenulikuks, oleme sunnitud ka kõik veebilahendused ehitama üles nii, et ilma ennast tutvustamata seal palju teha ei saaks.

Üldjuhul on kõigil veebisaitidel mitu turvataset. Näiteks on olemas mingi avalik vaade ning kohad, kuhu saavad ligi ainult valitud kasutajad. Lisaks sellele võivad ka tuvastatud kasutajad olla erinevate õigustega nt Admin, Müügimees, Ostja. Kõigi nende elementaarsete kuid keeruliste protseduuride jaoks pakub ASP.NET alates versioonist 2.0 välja terve hulga serverikontrolle ning rollihalduri.

Kuna kogu kasutajatehalduse ja ning tuvastamise teema on üsna mahukas ja keeruline siis ei maksa ise jalgratast leiutada ning tasub maksimaalselt ära kasutada ASP.NETi poolt pakutavaid lihtsustavaid vahendeid.

Üldised seadistused

Alustuseks tuleks konfiguratsioonifailis öelda, et me soovime kasutajaid autentida ning autentimata kasutajad sisse ei saa. Selleks lisame system.web elemendi alla kaks uut elementi:

- authentication – ütleb, kuidas plaanime kasutajaid tuvastada
- authorization – ütleb, kes saab sisse

Autentimiseks kasutame vormipõhist autentimist, mis võimaldab meil kirjutada oma metoodika kasutajate autentimiseks:

```
<authentication mode="Forms">
      <forms name="Projekt" loginUrl="~/logon.aspx" />
</authentication>
```

Järgmiseks tuleb anda ligipääs kõigile kasutajatele/kasutajagruppidele keda soovite oma veebile ligi lasta ning keelata ära ligipääs ülejäänutele. Ligipääsu andmiseks või keelamiseks tuleb autorization elemendi sisse lisada allow ja deny alamelemendid vastavalt ligipääsu andmiseks ja keelamiseks.

Kasutajale ligipääsu andmiseks või keelamiseks tuleb lisada allow elemendi sisse atribuut users. Näiteks:

```
<allow users="mati"/>
```

Annab ligipääsu kasutajale mati. Lisaks sellele on võimalik kasutada ka kahte erimärki * (tärn) ning ? (küsimärk). Tärniga tähistatakse kõiki kasutajaid ning küsimärgiga tähistatakse anonüümseid ehk autentimata kasutajaid. Kuni te autoriseerimisreegleid ei ole kehtestanud kehtivad vaikimise määratud autoriseerimisreeglid, mis annavad ligipääsu kõigile kasutajatele:

```
<allow users="*"/>
```

Kõige lihtsamaks muudatuseks autoriseerimisreeglites oleks keelata ligipääs kõigile anonüümsetele kasutajatele lubades kõik ülejäänud (kes on oma kasutajanime ja parooli sisestanud) veebile ligi:

```
<authorization >
      <allow users="*"/>
      <deny users="?"/>
</authorization>
```

See tekitab muidugi olukorra, kus ilma paroolita ei saa külastaja mitte ühelegi lehele ligi. Kui soovite osaliselt lubada ka anonüümset ligipääsu, tuleks kas:

- kõik turvamist vajavad lehed tõsta teise kausta ja lubada juurkausta kõik ning seada täiendavad piirangud alamkaustadele

- või öelda, et nt default.aspx'i me ei kaitse

Mõlemal juhul tuleb lisada konfiguratsiooni eraldi location element, millega anname teatud asukohale erineva seadistuse. Hetkel läheme seda teed, et lubame kõigil vaadata default.aspx faili.

```
<location path="default.aspx">
    <system.web>
      <authorization >
        <allow users="*"/>
        <deny users="?"/>
      </authorization>
    </system.web>
</location>
```

Nüüd tuleks valmis teha kasutajate autentimise vorm logon.aspx. ASP.NET alates versioonist 2.0 on tore selle poolest, et kasutaja tuvastamise vormi ei pea ise moodustama, selle jaoks on spetsiaalne server element nimega logon. Seega näeb meie login.aspx välja järgmine:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="logon.aspx.cs"
```

```
         Inherits="logon" %>
<!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>Kasutaja tuvastamine</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Login ID="Login1" runat="server"
                                        OnAuthenticate="TuvastaKasutaja" />
    </form>
</body>
</html>
```

Koodifailis peame realiseerima kasutajate tuvastamise protseduuri. Hetkel lahendame selle ülilihtsalt. Tegelikkuses peaks olema tegemist mingi andmebaasipõhise kasutaja tuvastamisega.

```
public partial class logon : System.Web.UI.Page
{
    protected void TuvastaKasutaja(object sender, AuthenticateEventArgs e)
    {
        e.Authenticated = Login1.UserName.ToUpper() == "MATI" &&
                          Login1.Password == "pwd";
    }
}
```

Ülesandeid

* Kaitse parooliga terve kataloog

* Küsi lubatud kasutajanimed ja paroolid andmebaasist

Kasutajate haldus

Kasutajatunnuste haldamiseks pakub ASP.NET välja mitmeid põnevaid elemente, mis kõik töötavad MembershipProvider'i peal. Vaikimisi kasutab kõnealune teenusepakkuja ühte konkreetset SQL serveri andmebaasi, kuid seda funktsionaalsust on võimalik ümber ehitada.

Kui teil on veebiserveris piisavalt õigusi siis võite rahus kasutada Microsofti poolset lahendust kasutajateinfo hoidmiseks. Selle info tekitamiseks ja haldamiseks on teil võimalik kasutada WebSite menüüst käsklust „ASP.NET Configuration", mille abil saate muuta ära nii kasutajate tuvastamise mehhanismi kui ka teha uusi kasutajakontosid ja hallata olemasolevaid kasutajakontosid. Kõik need määrangud paigutatakse teie veebi App_Data kasuta ASPNETDB nimelisse andmebaasi.

Selle info asukoht on määratud masina seadistustes e. kaustas %windir%\Microsoft.NET\Framework\v2.0.50727\CONFIG asuvas failis machine.config. Asukoht on määratud SQL serveri asukohaga, ühendustekstis LocalSqlServer

```
<connectionStrings>
  <add name="LocalSqlServer"
       connectionString="data source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User
Instance=true"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Kui te ei saa oma veebis sellist lähenemist andmebaasile kasutada siis võite kogu selle ASP.NET tarkuse ümber suunata oma andmebaasi. Selleks tuleb teil:

- Oma veebi seadistustes muuta ära vaikimisi määratud andmebaasi asukoht.
- Lisada oma andmebaasi vajalikud tabelid, vaated ja protseduurid

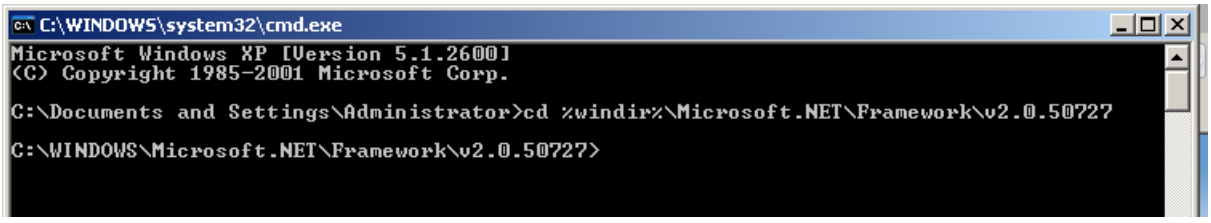
Vaikimisi määratud andmebaasi asukoha muutmiseks peate eemaldama süsteemse ühendusteksti ning lisama sama nimega oma ühendusteksti.

```
  <connectionStrings>
    <remove name="LocalSqlServer" />
    <add name="LocalSqlServer"
         connectionString="Data Source=localhost\sqlexpress;Initial
Catalog=omabaas;Integrated Security=True"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Loomulikult eeldab ASP.NET, et see andmebaas on ka vastavalt ette valmistatud – nii, et seal oleksid olemas kõik vajalikud tabelid, vaated ja protseduurid. Nende andmebaasiobjektide tekitamiseks on .NET raamistikuga kaasas üks programm, mis paikneb kaustas %windir%\Microsoft.NET\Framework\v2.0.50727 ning kannab nime aspnet_regsql.exe. Sellel programmil on olemas nii graafiline kui ka käsurea liides. Soovitan kasutada käsurea liidest, sest see võimaldab anda programmile rohkem infot selle kohta, milliseid tabeleid on vaja tekitada. Graafiline liines (avaneb topeltklõpsuga) tekitab teie andmebaasi kõik tabelid kõigi võimalike teenuste tarbeks.

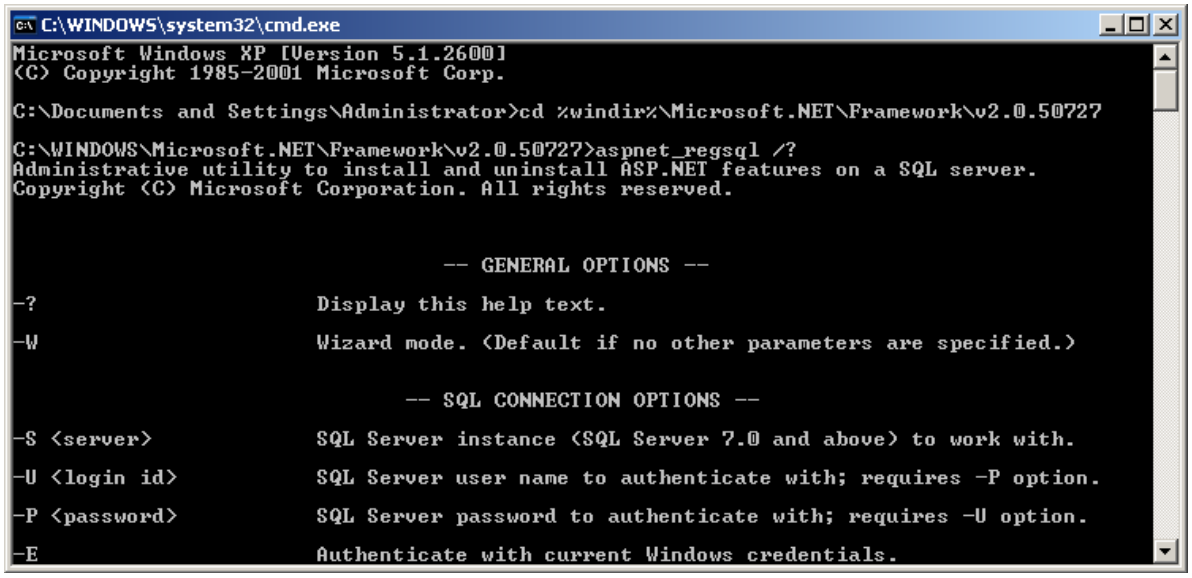
Programmi käivitamiseks käsurealt avage Windowsi Command Prompt. See paikneb start menüüs Programs\Accessories\Command Promt.

Seejärel navigeerige õigesse kausta kasutades käsku cd <kaustanimi>



Nüüd saate programmi käivitada, kuid enne oleks mõistlik lugeda kuidas seda programmi kasutada. Selleks kirjutage käsureale <programminimi> /?

See käsk kuvab teile lühiõpetuse, käsu kasutamiseks.



Sellest õpetusest tuleks leida, kuidas saab ühendust andmebaasiga ning kuidas määrata paigaldatavaid komponente. Järgnevalt on toodud väike tõlgitud väljavõte sellest õpetusest:

```
-S <server>  SQL Server, kus paikneb teie andmebaas
-E           Kasutame SQLiga suhtlemisel Windowsi autentimist
-A all|m|r|p|c|w  Milliseid funktsioone soovime paigaldada
          all: Kõik funktsioonid
          m: Kasutajate haldus
          r: Rollide haldus
          p: Profiilid
          c: Isikupärastamine
          w: veebisündmused
-R all|m|r|p|c|w  Milliseid funktsioone soovime eemaldada.
-d <database>  Millisesse andmebaasi funktsioonide jaoks vajalik paigaldada
```

Ülaltoodust tulenevalt saaksime oma andmebaasi paigaldada kasutajatehalduse järgmise käsureaga:

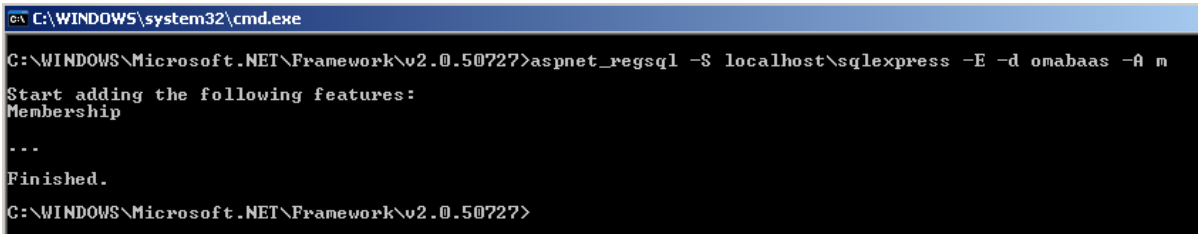
```
aspnet_regsql -S localhost\sqlexpress -E -d omabaas -A m
```

Kui soovite paigaldada rollihaldust kirjutate vastavalt

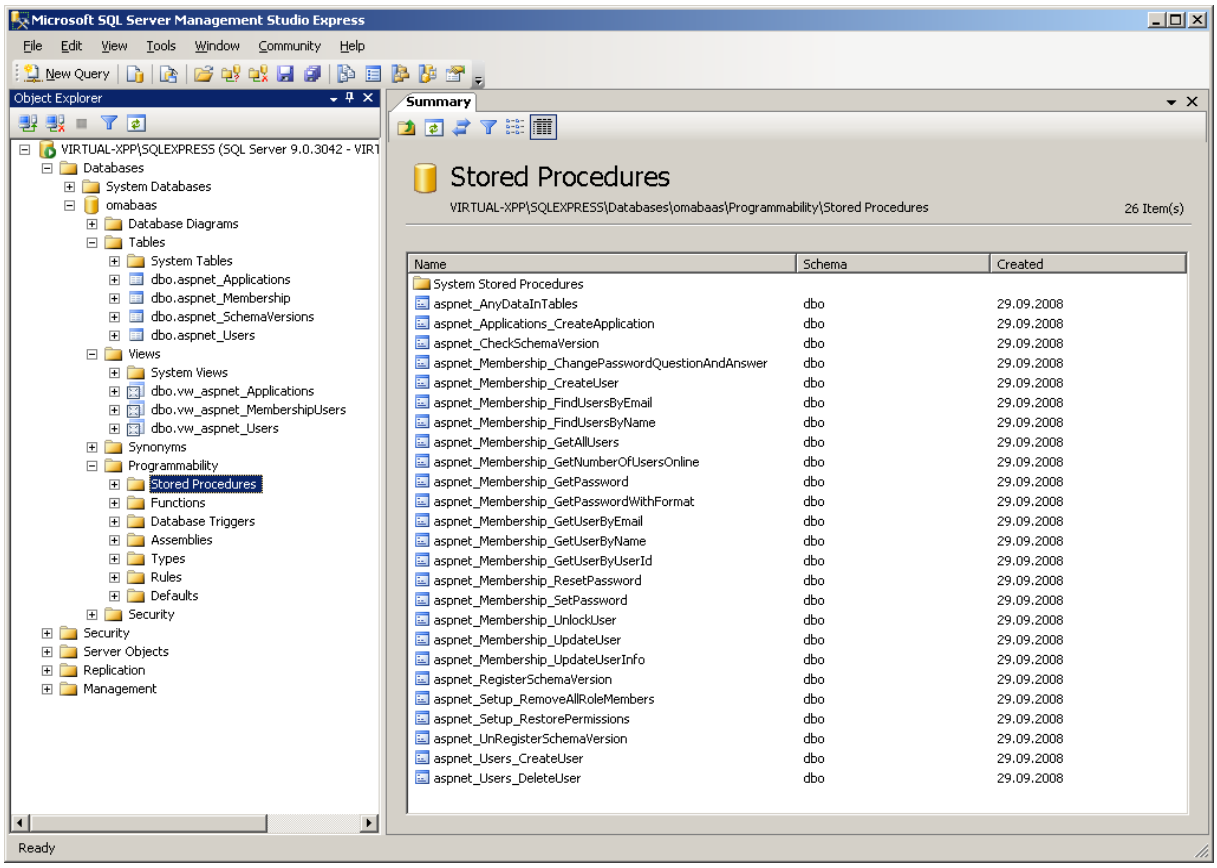
```
aspnet_regsql -S localhost\sqlexpress -E -d omabaas -A r
```

Kui soovite paigaldada nii kasutajate kui ka rollihaldust siis saate seda teha

```
aspnet_regsql -S localhost\sqlexpress -E -d omabaas -A mr
```



Kui kõik õnnestub, siis loetletakse ülesse, millised funktsioonid said paigaldatud. Ning võite vaadata oma andmebaasi, kuhu tekkis terve hulk uusi tabeleid vaateid ja protseduure.



Tekitatud objektide hulk on päris suur. Vaataks neist üle olulisemad:

Tabel: dbo.aspnet_applications – teie programmid/veebid, mis seda teenust kasutavad

Tabel: dbo.aspnet_membership – kasutajate andmed

Tabel: dbo.aspnet_users – kasutajate nimed

Vaade: dbo.vw_aspnet_membershipusers – kasutajate andmed koos nimedega

Üldiselt on nii, et ei ole soovitav nende tabelite sisu otse muuta, sest selle tulemusena võib muutuda see info ASP.NETile kasutamiskõlbmatuks. Selle asemel saate infot vaadata ja muuta kasutades ASP.NET poolt pakutavat membership klassi.

Hea uudis on aga see, et kõik ASP.NETiga kaasas olevad, kasutajate haldusega tegelevad serveri elemendid töötavad ☺ ning nende kasutamiseks ei pea te enam ühtegi koodirida kirjutama. Lihtsalt lohistate teid huvitava elemendi oma veebilehele sobivasse kohta ning see töötab.

Elemendi nimi	Kirjeldus
Login	Vorm kasutaja autentimiseks
LoginView	Vaade vastavalt kasutaja õigustele
PasswordRecovery	Vorm kasutajale oma parooli meeldetuletuse saamiseks Kuna parool saadetakse meiliga siis selle elemendi töölepanek eeldab, et teie kasutada on mõni meiliserver (SMTP teenus)
LoginStatus	Lipik, mis näitab, kas kasutaja on autenditud või mitte
LoginName	Lipik, mis näitab kasutajanime
CreateUserWizard	Võlur uute kasutajate loomiseks
ChangePassword	Vorm parooli vahetamiseks

Lisaks sellele saate kasutajaid hallata läbi VisualStudios asuva configureerimislehel: WebSite\ASP.NET Configuration

Loomulikult surub selline lähenemine peale Microsofti nägemuse kasutajate haldamise funktsionaalsusest ning andmete paigutamisest. Kui soovite päris oma lähenemist siis tuleb teil luua oma teenusepakkuja, mis realiseerib kõik vajalikud funktsioonid just teile vajalikul moel. Näide selle kohta on leheküljel 473.

Rollide kasutamine ligipääsu kontrollimisel

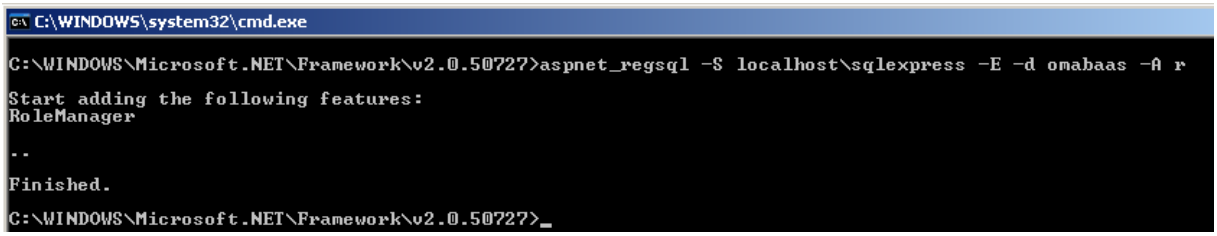
Kuni veebil on kasutajaid vähe on kasutajanime järgi õiguste andmine mõeldav. Kuid kui kasutajaid tekib palju või kasutajad vahelduvad tihti muutub sellise süsteemi haldamine äärmiselt vaevarikkaks ning märksa lihtsam on ligipääsu anda vastavalt sellele, milliseid rolle antud kasutaja peab teie veebis täitma.

Selleks, et kasutada rollipõhist ligipääsu tuleb iga kasutaja juures määrata, milliseid rolle kasutaja täidab. Lisaks sellele, kui hoiate kõiki andmeid oma andmebaasis peate täiendama ka

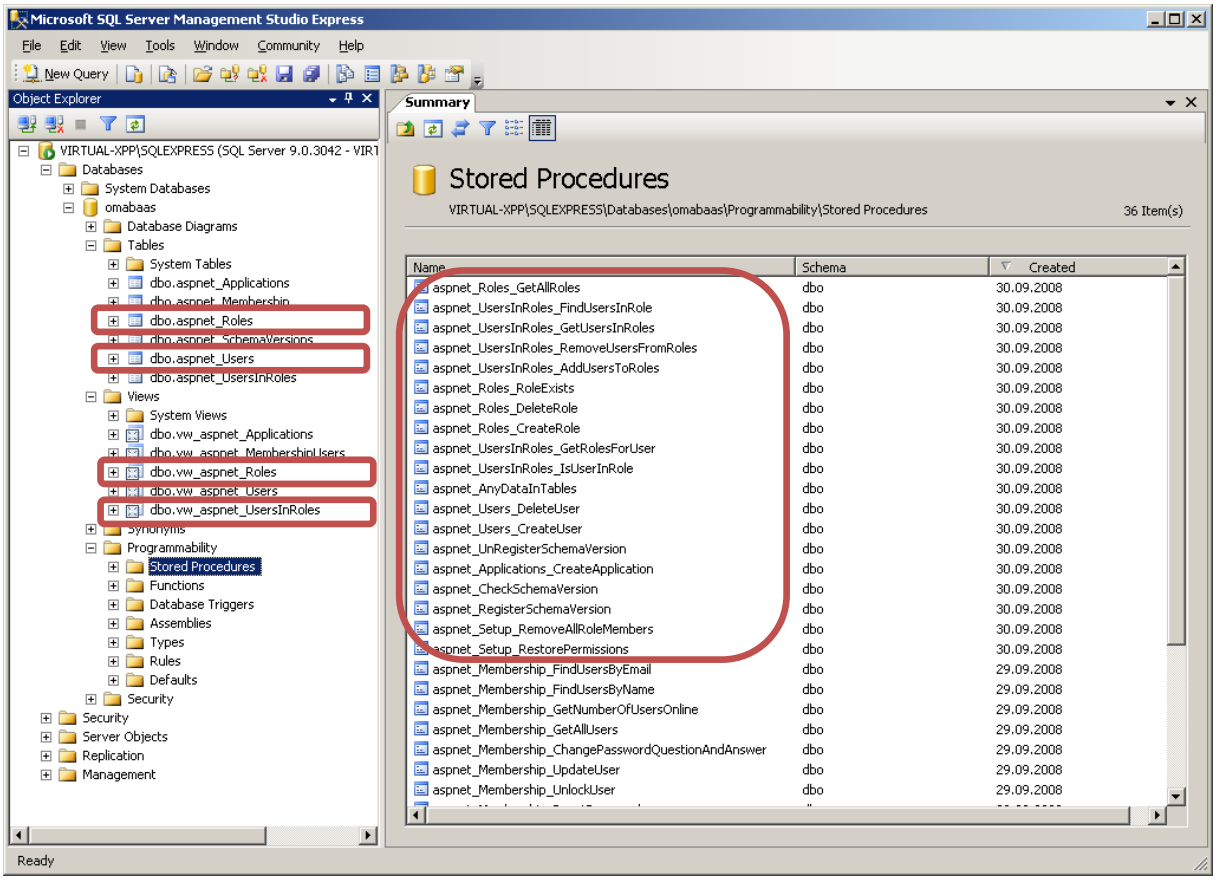
oma andmebaasi rollihalduseks vajalike tabelite, vaadete ja protseduuridega või siis kirjutama oma teenusepakkuja rollihalduseks (vaata lk 475).

Rollihalduseks vajalike andmebaasiobjektide lisamiseks saate kasutada taas aspnet_regsql.exe programmi.

```
aspnet_regsql -S localhost\sqlexpress -E -d omabaas -A r
```



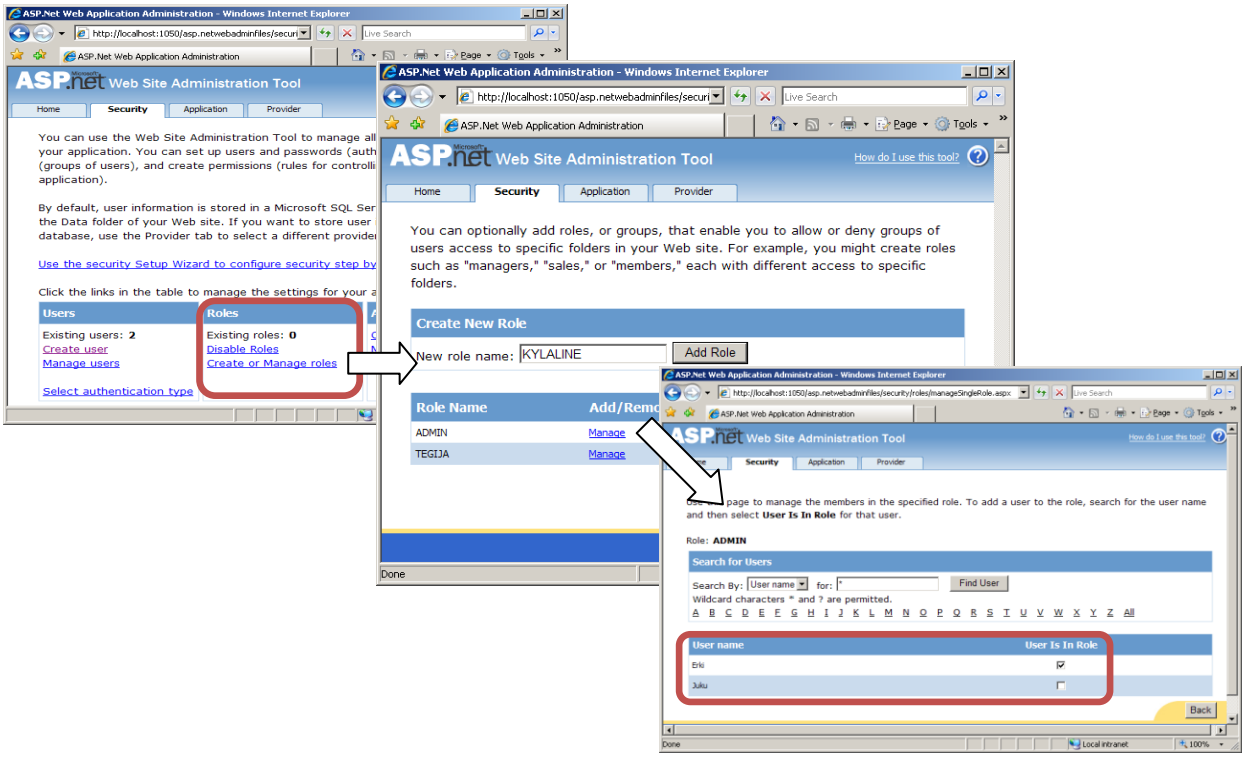
Selle tulemusena lisatakse andmebaasi mõned uued tabelid, vaated ja protseduurid.



Pale struktuuri loomist on vaja ASP.NETile kuulutada, et kasutate rollihaldust e. teha väike muudatus web.config failis lisades system.web elemendi alla elemendi rolemanager:

```
<roleManager enabled="true" />
```

Peale seda saate tekitada kõik vajalikud rollid ning määrata kasutajate rollikuuluvuse. Nende tegevuste tarbeks saate luua oma halduslehe, kasutades Roles klassi poolt pakutavaid meetodeid või kasutada veebi konfigureerimisutiliiti WebSite\ASP.NET Configuration



Õiguste jagamiseks saate kasutada web.config faili autorization elemendi alamelemente allow ja deny, määrates seekord users atribuudi asemel roles atribuudi. Näiteks anname ligipääsu kõigile kasutajatele, kes kuuluvad ADMIN rolli ning keelame ligipääsu kõigile teistele.

```
<authorization >
  <allow roles="ADMIN"/>
  <deny users="*"/>
</authorization>
```

Andmetega manipuleerimine

Andmetega manipuleerimiseks ASP.NET keskkonnas on alates versioonist 3.5 kaks võimalust:

- Kasutada vanamoelist lähenemist ADO.NET vahendite abil
- Kasutada uut ja põnevat lähenemist LINQ abil

Nende kahe meetodi erinevus seisneb andmebaasiga suhtlemise metoodikas. Kui kõik suhtlusreeglid on kokkulepitud siis selle andmeallika kasutamine andmeid kuvavate elementide peal käib mõlemal juhul ühte moodi.

Andmete kasutamine ADO.NET abil

Selleks, et andmetele ligi pääseda, tuleb luua ühendus andmebaasiga. Ühenduse loomiseks on vaja andmebaasi. Andmebaasile viitamiseks kasutatakse ühendusteksti (ConnectionString). Kuna peale rakenduse valmimist võib administraator tõsta andmebaasi mõnda teise serverisse või anda andmebaasile teise nime või muuta turvameetmeid, peab ühendustekst olema administraatorile ehk rakenduse hooldajale lihtsasti kättesaadav. Seega ei saa ühendusteksti kirjutada tekstina otse koodi. Parim koht selle hoidmiseks on konfiguratsioonifail web.config. ASP.NET konfiguratsioonis on loodud isegi eraldi konteinerelement ühendustekstide hoidmiseks. configuration elemendi alamelemendi nimeks on connectionStrings.

Ühendustekst nimega yhendusTxt võiks välja näha midagi sarnast järgnevaga:

```
<connectionStrings>
<add name="yhendusTxt" connectionString="packet size=4096;uid=KASUTAJANIMI;
pwd=PAROOL;data source=dotnet;persist security info=False;initial
catalog=KASUTAJANIMI" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Kui on kasutada rohkem kui üks andmebaas, võib siia ühendustekste juurde lisada. Suurema turvalisuse saavutamiseks on võimalik ka teatud osade sh connectionStrings krüpteerimine konfiguratsiooni failis.

Ühendusteksti kasutamiseks tuleb koodis sellele viidata. Selleks on kaks võimalust.

Esiteks võime seda teha aspx lehel andmeallika kirjelduses.

```
<asp:SqlDataSource id="andmeallikas" runat="server"
ProviderName="System.Data.SqlClient" ConnectionString='<%#
ConfigurationManager.ConnectionStrings("yhendusTxt").ConnectionString %>'
SelectCommand="SELECT TOP 100 ToodeID, Nimi, Hind FROM toode"/>
```

Sellest pikast lausest on olemas ka lühendatud kuju:

```
<asp:sqldatasource id="andmeallikas " runat="server"
ProviderName="System.Data.SqlClient" ConnectionString="<%$
ConnectionStrings:yhendusTxt %>" SelectCommand="SELECT TOP 100 ToodeID,
Nimi, Hind FROM toode"/>
```

Saadud andmete näitamiseks võimalik kasutada nt tabeli vaadet (GridView).

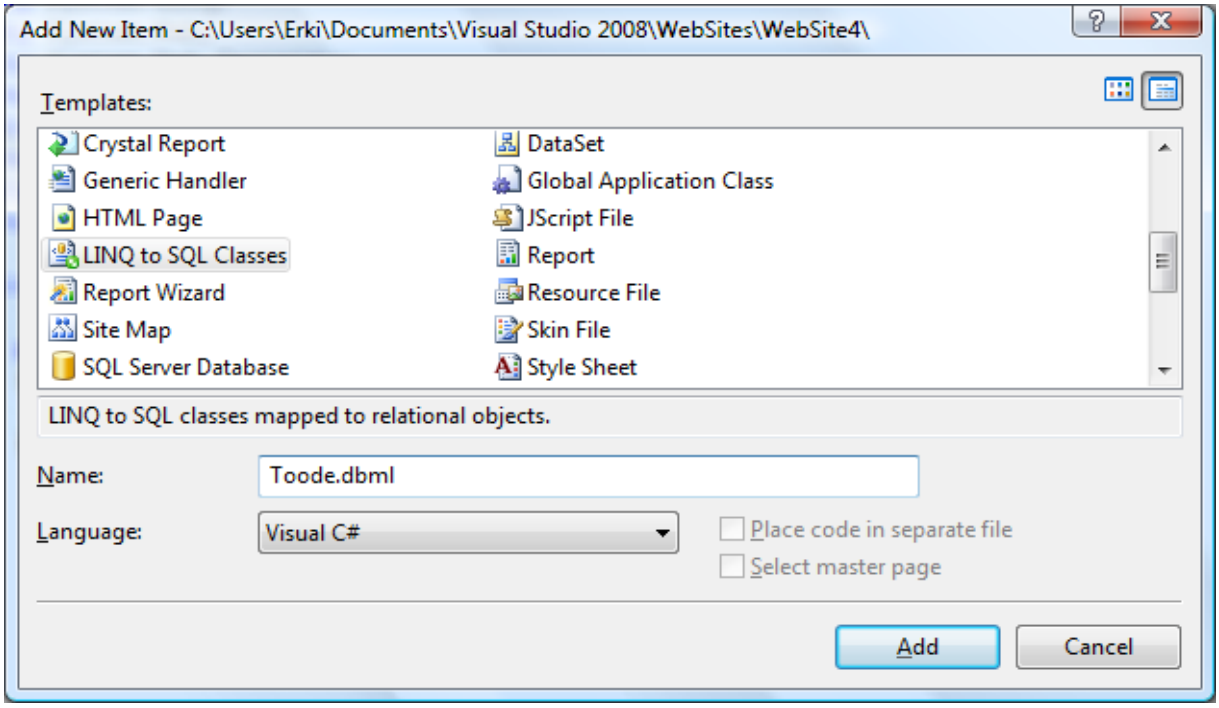
```
<asp:GridView ID="gw" runat="server" AutoGenerateColumns="True"
DataSourceID="andmeallikas" EmptyDataText="Pole midagi näidata" />
```

Nagu näete, on võimalik lugeda andmebaasist andmeid tabelivaatesse ilma ridagi koodi kirjutamata. Kui kasutate veebilehe loomiseks graafilisi abivahendeid nagu VisualStudio, genereeritakse teile ka eelpool kirjeldatud read aspx failidesse.

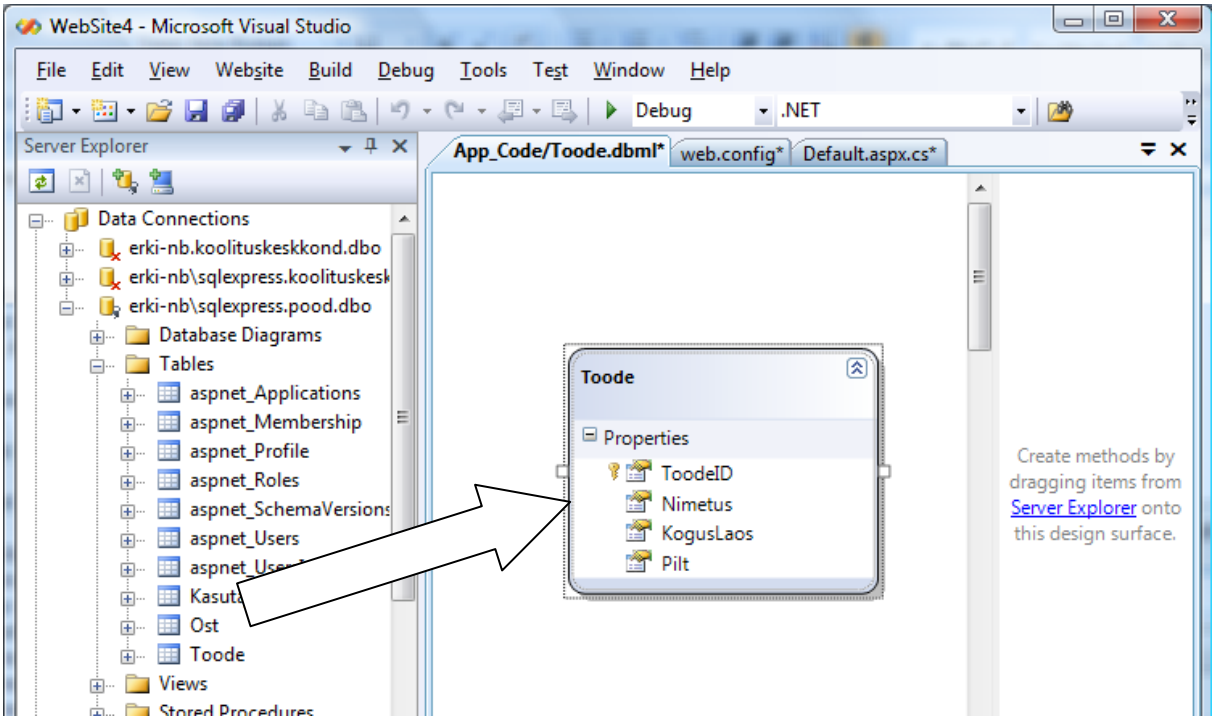
Andmete kasutamine LINQ abil

Andmete leidmiseks LINQ abil tuleb esmalt luua App_Code kausta „LINQ to SQL“ klass, mis hakkab korraldama andmevahetust veebirakenduse ja andmebaasi vahel.

Selleks lisage oma veebi App_Code kausta uus item, mis on „LINQ to SQL Classes“ tüüpi.

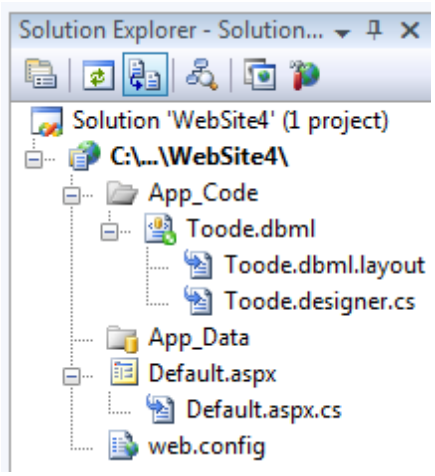


Avaneb LINQ to SQL klassi disainerivaade, kus saate lohistada kõik teile olulised tabelid, koos nendevaheliste seostega LINQ to SQL klassi. Näiteks haarame ServerExplorerist oma andmebaasis paikneva toodete tabeli ning lohistame disainerisse:



Vajadusel saate lohistada sellele skeemile tabeleid juurde ning määrata keerukamaid meetodeid nii andmete leidmiseks kui muutmiseks. Samas kui teile piisab sellisest lihtsustatud ligipääsust siis võite selle klassi ära salvestada ja hakata teda kasutama andmetega manipuleerimisel.

Selles disainerivaates näete te selle klassi graafilist kuju. Klassi tegelik kood paikneb selle disainifaili taba olevas koodifailis <klassinimi>.designer.cs



Lisaks on seal ka fail <klassinimi>.dbml.layout, milles on XML kujul see sama graafiline vaade teie loodud klassile.

Visates kiire pilgu loodud koodile võib selles välja lugeda, et tegemist on maskeeringuga, kus klass Toode maskeerib ära andebaasis oleva tabeli Toode, koos kõigi selle tabeli omaduste ja muutmisvõimalustega ning klass ToodeDataContext on andmeallikas LINQ päringute tegemiseks, mis kasutab seda maskeeritud Toode tabelit.

```
#pragma warning disable 1591
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1433
//
//     Changes to this file may cause incorrect behavior and will be lost
//     if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

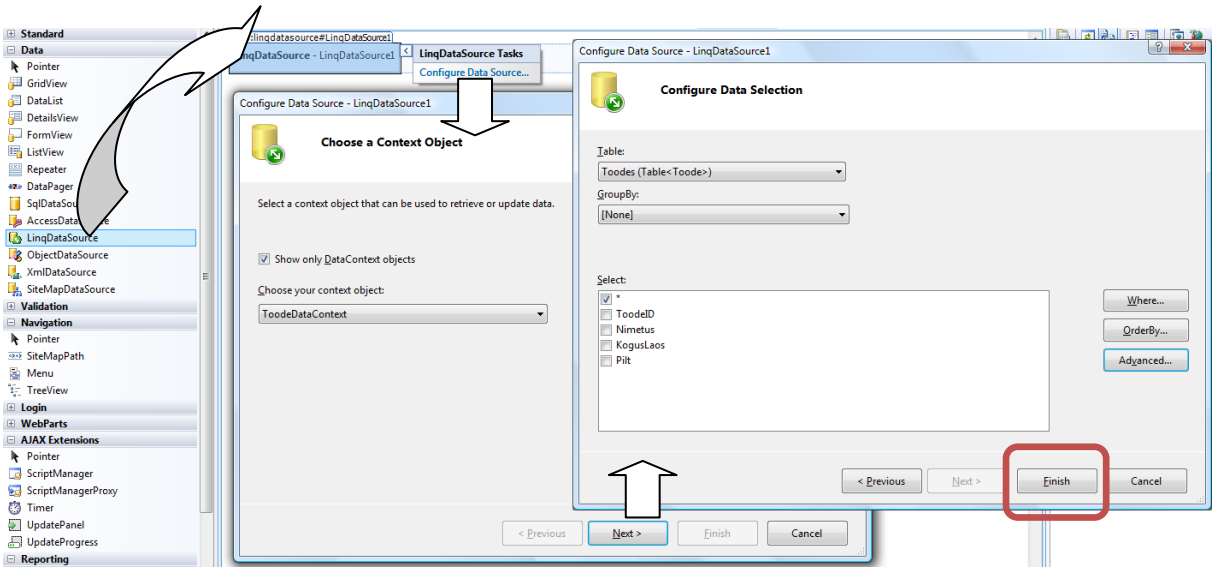
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;

[System.Data.Linq.Mapping.DatabaseAttribute(Name="pood")]
public partial class ToodeDataContext : System.Data.Linq.DataContext
{

	private static System.Data.Linq.Mapping.MappingSource mappingSource = new
AttributeMappingSource();

  #region Extensibility Method Definitions
  partial void OnCreated();
  partial void InsertToode(Toode instance);
  partial void UpdateToode(Toode instance);
  partial void DeleteToode(Toode instance);
  #endregion

	public ToodeDataContext() :

	base(global::System.Configuration.ConfigurationManager.ConnectionStrings["poo
dConnectionString"].ConnectionString, mappingSource)
	{
		OnCreated();
	}

	public ToodeDataContext(string connection) :
			base(connection, mappingSource)
	{
		OnCreated();
	}

	public ToodeDataContext(System.Data.IDbConnection connection) :
			base(connection, mappingSource)
	{
		OnCreated();
	}

	public ToodeDataContext(string connection,
System.Data.Linq.Mapping.MappingSource mappingSource) :
			base(connection, mappingSource)
	{
		OnCreated();
	}

	public ToodeDataContext(System.Data.IDbConnection connection,
System.Data.Linq.Mapping.MappingSource mappingSource) :
			base(connection, mappingSource)
	{
		OnCreated();
	}

	public System.Data.Linq.Table<Toode> Toodes
	{
		get
		{
			return this.GetTable<Toode>();
		}
	}
}
```

```
[Table(Name="dbo.Toode")]
public partial class Toode : INotifyPropertyChanging, INotifyPropertyChanged
{

      private static PropertyChangingEventArgs emptyChangingEventArgs = new
PropertyChangingEventArgs(String.Empty);

      private int _ToodeID;

      private string _Nimetus;

      private int _KogusLaos;

      private System.Data.Linq.Binary _Pilt;

    #region Extensibility Method Definitions
    partial void OnLoaded();
    partial void OnValidate(System.Data.Linq.ChangeAction action);
    partial void OnCreated();
    partial void OnToodeIDChanging(int value);
    partial void OnToodeIDChanged();
    partial void OnNimetusChanging(string value);
    partial void OnNimetusChanged();
    partial void OnKogusLaosChanging(int value);
    partial void OnKogusLaosChanged();
    partial void OnPiltChanging(System.Data.Linq.Binary value);
    partial void OnPiltChanged();
    #endregion

      public Toode()
      {
            OnCreated();
      }

      [Column(Storage="_ToodeID", AutoSync=AutoSync.OnInsert, DbType="Int NOT NULL
IDENTITY", IsPrimaryKey=true, IsDbGenerated=true)]
      public int ToodeID
      {
            get
            {
                  return this._ToodeID;
            }
            set
            {
                  if ((this._ToodeID != value))
                  {
                        this.OnToodeIDChanging(value);
                        this.SendPropertyChanging();
                        this._ToodeID = value;
                        this.SendPropertyChanged("ToodeID");
                        this.OnToodeIDChanged();
                  }
            }
      }

      [Column(Storage="_Nimetus", DbType="NVarChar(50) NOT NULL", CanBeNull=false)]
      public string Nimetus
      {
            get
            {
                  return this._Nimetus;
            }
            set
            {
                  if ((this._Nimetus != value))
                  {
                        this.OnNimetusChanging(value);
                        this.SendPropertyChanging();
```

```
                        this._Nimetus = value;
                        this.SendPropertyChanged("Nimetus");
                        this.OnNimetusChanged();
                    }
                }
            }

            [Column(Storage="_KogusLaos", DbType="Int NOT NULL")]
            public int KogusLaos
            {
                get
                {
                    return this._KogusLaos;
                }
                set
                {
                    if ((this._KogusLaos != value))
                    {
                        this.OnKogusLaosChanging(value);
                        this.SendPropertyChanging();
                        this._KogusLaos = value;
                        this.SendPropertyChanged("KogusLaos");
                        this.OnKogusLaosChanged();
                    }
                }
            }

            [Column(Storage="_Pilt", DbType="VarBinary(MAX)",
UpdateCheck=UpdateCheck.Never)]
            public System.Data.Linq.Binary Pilt
            {
                get
                {
                    return this._Pilt;
                }
                set
                {
                    if ((this._Pilt != value))
                    {
                        this.OnPiltChanging(value);
                        this.SendPropertyChanging();
                        this._Pilt = value;
                        this.SendPropertyChanged("Pilt");
                        this.OnPiltChanged();
                    }
                }
            }

            public event PropertyChangingEventHandler PropertyChanging;

            public event PropertyChangedEventHandler PropertyChanged;

            protected virtual void SendPropertyChanging()
            {
                if ((this.PropertyChanging != null))
                {
                    this.PropertyChanging(this, emptyChangingEventArgs);
                }
            }

            protected virtual void SendPropertyChanged(String propertyName)
            {
                if ((this.PropertyChanged != null))
                {
                    this.PropertyChanged(this, new
PropertyChangedEventArgs(propertyName));
                }
            }
```

```
}
#pragma warning restore 1591
```

Loodud klassi kasutamiseks andmeallikana tuleb lisada veebilehele LinqDatasource. Lohistades vastava elemendi Toolbox'ilt oma veebilehele, ning seadistades teda läbi seadistusvõluri, mis paikneb Configure Data Source käsu all.



Kui peale seda protsessi vaatate oma veebilehe koodi siis leiate, et tulemuseks element asp:LinqDataSource, mis seab ContextTypeName atribuudi abil kokku loodud andmeallika ning LINQ to SQL klassi:

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
     ContextTypeName="ToodeDataContext" TableName="Toodes">
</asp:LinqDataSource>
```

Andmeallik kasutamiseks GridView peal tuleb vaid viidata loodud andmeallikale:

```
<asp:GridView ID="GridView1" runat="server"
              AutoGenerateColumns="True"
              DataKeyNames="ToodeID"
              DataSourceID="LinqDataSource1" />
```

Kokkuvõtteks võiks öelda, et LINQ kasutamine vajab esmalt pisut rohkem ettevalmistust, kui traditsiooniline ADO.NET, kuid kui ettevalmistused tehtud (LINQ to SQL klassi loomise teel) on selle kasutamine oluliselt lihtsam kui seda on ADO.NET kasutamine. Lisaks sellele on loodud ToodeDataContext kasutatav ka programmelt e. C# koodi abil. See võimaldab teil luua nt programme, mis vaatavad selle toodete tabeli rea kaupa üle ning teevad vastavalt vajadusele seal muudatusi jne.

Samuti on hiljem selgub, et andmete poole pöördumist või nendega manipuleerimist on vaja täiustada siis saate seda teha ühes keskses kohas (LINQ to SQL klassis) ning tehtud muudatustega arvestatakse kõigil veebilehtedel, mis neid andmeid kasutasid. Kasutades traditsioonilist ADO.NET lähenemist peate te muudatusi tegema eraldi igal lehel, mis andmetega tegeleb!

Andmete kuvamine lihtsate loetelude abil

Lihtsateks loeteludeks nimetatakse kasutajaliidese elemente, mis võimaldavad vaid ühe väärtuse loetelu.

Lihtsad loetelud on:

- AdRotator – võimaldab kuvada bannereid. Iga kord kui kasutaja tuleb lehele, valitakse juhuslikult uus pilt kuvamiseks
- BulletedList – täppidega loetelu
- DropDownList – ripploend
- ListBox – kerimisribadega loetelu kastis
- RadioButtonList – raadionuppudega loetelu, kust saab valida vaid ühe väärtuse
- CheckBoxList – valikukastidega loetelu, kust saab valida mitmeid väärtuseid

Selleks, et lihtsaid loetelusid kasutada, tuleb määrata ära DataSource või DataSourceID omadus, millega näitate, kust tulevad andmed.

Lisaks on soovitav täita väljad DataTextField, mis näitab milline väli andmetest kuvatakse kasutajale ning DataValueField e milline väli on objekti väärtuseks, kui kasutaja midagi valib.

Kõik loetelus olevad elemendid on programselt saadaval läbi Items kollektsiooni ning kasutaja poolt tehtud valiku saate teada läbi SelectedIndex omaduse. Kui soovite igale kasutaja valikule reageerida, võite teha protseduuri sündmuse SelectedIndexChanged sündmuse tarbeks.

Järgnevalt tekitame toodetest täppidega loetelu:

```
<asp:BulletedList ID="loetelu " runat="server" DataSourceID="andmeallikas"
DataTextField="Nimi" />
```

Andmete kuvamine keerukate loetelude abil

Keerukateks loeteludeks nimetatakse kasutajaliidese elemente, mis andmete kuvamiseks tekitavad hulga alamelemente.

Keerulised loetelud on:

- GridView – võimaldab näidata andmeid tabeli kujul, pakkudes lihtsaid vahendeid andmete sorteerimiseks, filtreerimiseks ja muutmiseks.

- DetailsView – võimaldab andmeid kirje kaupa vaadata, muuta ja juurde lisada. Andmed kuvatakse tabeli kujul, väljad üksteise all.

- FormView – sisuliselt nagu DetailsView, kuid andmete kuvamise loogika tuleb läbi põhjade (Template) ise paika panna.

- Repeater – Kõige abstraktsem loetelu, võimaldab põhjade abil leida andmetele just sellise kuju nagu teil vaja.

Hierarhiliste andmete kuvamine

Hierarhilised loetelud on mõeldud hierarhiliste andmete näitamiseks nagu XML failis olevad andmed ja vanema – lapse suhted. Hierarhilised loetelud on:

- TreeView – andmete esitus puu kujul

- Menu – menüü rakenduses navigeerimiseks. Võib olla nii puu kujul kui ka puu kujul.

WebParts

Siiani oleme kogu veebi kujunduse ise valmis teinud, kuid lisaks sellele on võimalik veebileht ehitada üles ka nii, et erinevate osade paigutuse saab valida kasutaja ise. Me võime seda funktsionaalsust lubada kõigile, või piirata nt administraatorite grupile. Oluline on see, et me saame lihtsate vahenditega anda kasutajale võimaluse veebi väljanägemise muutmiseks. WebPart'id on kontrollide kogumik, mis seda funktsionaalsust võimaldab.

WebPart'ide struktuur on kolme kihiline: isikupärastamine, kasutajaliidese struktuuri komponendid ja kasutajaliidese komponendid ise.

Isikupärastamine on WebPart'ide struktuuri alustalaks. See võimaldab kasutajatel muuta – isikupärastada – WebPartide paigutust, väljanägemist ja käitumist. Tehtud muudatused on püsivad st nad jäävad kehtima ka siis kui kasutaja käib vahepeal teistel lehtedel või paneb üldse veebisirvija kinni.

Struktuuri elemendid baseeruvad isikupärastamisel ning pakuvad baasstruktuuri ja teenused kõigile WebPartidele. Üks struktuuri element, mis peab olema igal WebParte kasutaval leheküljel on WebPartManager. Kuigi see kontroll ei ole kunagi nähtaval on tal üks kriitiline ülesanne – koordineerida kõigi WebPartide olekut leheküljel – ta haldab kõiki tsoone, kus on võimalik WebParte hoida ning seda, millised WebPart'id ühes või teises on olemas ning milline on nende olek. Lisaks sellele koordineerib ta ka WebPartide vahelist suhtlemist.

Teine struktuurielement, mis peab alati olemas olema on tsoon e. koht kuhu on võimalik WebParte paigutada.

Järgnevalt ülevaade WebPart'idega seotud kontrollidest:

WebPartManager	Haldab lehel WebPart kontrolle. Igal lehel on vaja ühte ja ainult ühte WebPartManager'i
CatalogZone	Sisaldab CatalogPart kontrolle. Seda kasutatakse kasutajale WebPartide kataloogi loomiseks
EditorZone	Sisaldab EditorPart kontrolle. Selles tsoonis saab kasutaja WebParte muuta
WebPartZone	Üldine WebPartide paigutus lehel. Selliseid tsoone võib lehel olla rohkem kui üks.
ConnectionsZone	Sisaldab WebPartConnection kontrolle ning pakub kasutajaliidest ühenduste haldamiseks.
WebPart	Kasutajaliidese element, mille paigutuse üle saab kasutaja otsustada
CatalogPart	Sisaldab loetelu WebPartidest, mida kasutaja saab lehele lisada
WebPartConnection	Tekitab ühenduse kahe WebParti vahele. Ühest neist WebPartidest saab klient ja teisest andmete pakkuja
EditorPart	Lubab kasutajal muuta WebParti põhiomadusi

Veebiteenused

Veebiteenus on protseduur, mis asub kauges serveris. Sisuliselt on tegemist tavalise klassiga, mille mõned meetodid on märgistatud kui veebist kasutatavad meetodid.

Veebiteenused on uus põlvkond protseduuri kaugkutseid, mida varem on püütud realiseerida COM, COM+ ja RPC abil. Veebiteenused võimaldavad lihtsalt korraldada rakenduste ja organisatsioonide vahelist andmevahetust, kuna andmevahetus käib XML kujul SOAP kirjadega ning ei ole vahet, kes ja kuidas need kirjad kokku paneb, peaasi, et nad on korrektses vormingus.

Veebiteenuste tegemine

Veebiteenuste faililaiendiks on .asmx ning selle faili sisu on ülimalt lihtne. Lisame näiteks faili Service.asmx. Selle faili sees tuleks öelda, et tegemist on veebiteenusega, mis keeles selle teenuse koodi plaanime kirjutada, kuhu (mis faili ja klassi) kood on salvestatud:

```
<%@ WebService Language="C#" CodeBehind="~/Service.cs" Class="Service" %>
```

Koodi keerukus sõltub loomulikult teenuse funktsionaalsusest. Siinsel juhul teeme ühe hästi lihtsa teenuse, millel on vaid üks meetod „Tere", mis välja kutsudes ütleb „Teretulemast veebiteenuste maailma!"

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
[WebService(Namespace = "http://tempuri.org/")]
public class Service : System.Web.Services.WebService
{
    [WebMethod]
    public string Tere() {
        return "Teretulemast veebiteenuste maailma!";
    }

}
```

Kuna tegemist on XML veebiteenusega, tuleb meil ära määrata nimeruum. Niisama katsetamiseks võib kasutada tempuri.org'i, kuid kui on omal mõni domeen kasutada, mida ise ka kontrollite, siis võite seda nime kasutada.

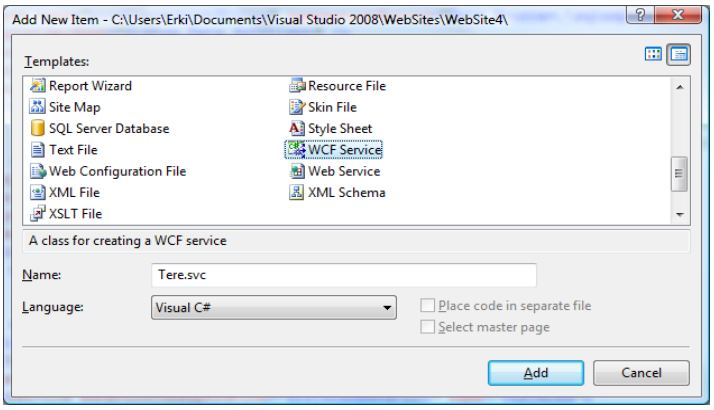
Veebiteenuse klass tuleb pärida klassist System.Web.Services.WebService, mis tagab kogu vajaliku funktsionaalsuse.

Kõik meetodid, mida tohib veebist välja kutsuda, tuleb märgistada [WebMethod] atribuudiga.

Veebiteenuse seadistamine käib web.config faili abil nagu ASP.NET rakendusel, kogu ülejäänud programmeerimine on täpselt nagu tavalisel klassil.

WCF teenuste tegemine

WCF – Windows Communication Foundation on uus meetod programmidevahelise infovahetuse korraldamiseks. Selliseid teenuseid saab teha alates raamistikust 3.0. Uue WCF teenuse loomiseks lisage oma veebi uus objekt, mis on „WCF Service“ tüüpi:



Selle tulemusena tekitatakse teile kaks faili. <teenusenimi>.svc, mille sisu on väga minimalistlik:

```
<%@ ServiceHost Language="C#" Debug="true" Service="Tere"
CodeBehind="~/App_Code/Tere.cs" %>
```

Ning selle teenuse nägu (paigutatakse App_Code kausta) <teenusenimi>.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

// NOTE: If you change the interface name "ITere" here, you must
also update the reference to "ITere" in Web.config.
[ServiceContract]
public interface ITere
{
      [OperationContract]
      void DoWork();
}
```

Nägu on selles, et tutvustada oma teenust selle teenuse kasutajatele. Lisaks sellele peate selle näo järgi tegema ka ühe klassi, mis hakkab reaalset teenust pakkuma! Nägu on sisuliselt leping teenusepakkuja ja teenuse tarbija vahel.

Loome näiteks veebiteenuse, mis oskab teenusekasutajat tervitada. Seda nii anonüümselt kui ka nimeliselt. Selleks teen esmalt väikesed muudatused selles teenuse näos. Ütleme, et meie teenusel saab olema kaks tervitamise meetodit. Üks neist vajab parameetrit ja teine mitte ning mõlemad tagastavad tervituse teksti kujul:

```
[ServiceContract()]
public interface IGeometryService
{
 [OperationContract]
 string Tervita();
 [OperationContract]
 string Tervita(string nimi);
}
```

Järgmiseks loome klassi, mis annab sellele näole funktsionaalsuse:

Veebiteenuse kasutamine

Veebiteenust saab kasutada igas .NET rakenduses ning on variant seda kasutada ka varem loodud programmeerimisvahenditega nagu nt VBA (Visual Basic for Applications).

.NET rakenduses kasutamiseks tuleb teha viide veebiteenusele. Kõige mugavam on seda teha VisualStudioga valides menüüst Website\WebReference, näitate ära teenuse aadressi nt http://localhost:52876/VeebiTeenus/Service.asmx ning klõpsate nupule AddReference.

Samas võib seda protsessi teha ka ilma VisualStudio abita.

Selleks tuleb esmalt lisada oma veebikausta alamkaust App_WebReferences, mille alla teete teenuse nimega alamkaust. Alamkausta nimi peaks olema selline, mida teil on hiljem mugav koodis kasutada nt TereTeenus ning sinna alla tulevad kõik ülejäänud failid, mida teenuse kasutamiseks vaja.

Esmalt on teil vaja teenuse päris nimega wsdl faili. Meie näite puhul oleks selleks Service.wsdl. Tegemist on XML failiga, mis räägib raamistikule, kuidas selle teenusega tuleb käituda. Selle faili sisu saate kopeerida teenuse juurest pannes asmx faili järgi võtme ?wsdl nt

http://localhost:52876/VeebiTeenus/Service.asmx?wsdl

Meie lihtsa teenuse puhul genereerub sealt järgmine XML:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
```

```
            targetNamespace="http://tempuri.org/">
    <s:element name="Tere">
      <s:complexType />
    </s:element>
    <s:element name="TereResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="TereResult"
                     type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
<wsdl:message name="TereSoapIn">
  <wsdl:part name="parameters" element="tns:Tere" />
</wsdl:message>
<wsdl:message name="TereSoapOut">
  <wsdl:part name="parameters" element="tns:TereResponse" />
</wsdl:message>
<wsdl:portType name="ServiceSoap">
  <wsdl:operation name="Tere">
    <wsdl:input message="tns:TereSoapIn" />
    <wsdl:output message="tns:TereSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="Tere">
    <soap:operation soapAction="http://tempuri.org/Tere"
                    style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="Tere">
    <soap12:operation soapAction="http://tempuri.org/Tere"
                      style="document" />
    <wsdl:input>
      <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="Service">
  <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
    <soap:address
            location="http://localhost:52876/VeebiTeenus/Service.asmx" />
  </wsdl:port>
  <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
    <soap12:address
            location="http://localhost:52876/VeebiTeenus/Service.asmx" />
  </wsdl:port>
```

```
  </wsdl:service>
</wsdl:definitions>
```

Teiseks on vaja lisada veebiteenuse nimega .disco fail. See fail räägib raamistikule, kuidas seda teenust edaspidi leida. Nagu .wsdl fail on ka .disco XML fail ning selle faili sisu saate teenuse juurest, lisades .asmx järgi võtme ?disco. Meie näite puhul oleks selleks

http://localhost:52876/VeebiTeenus/Service.asmx?disco ning sealt avaneks järgmine XML:

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://localhost:52876/VeebiTeenus/Service.asmx?wsdl"
               docRef="http://localhost:52876/VeebiTeenus/Service.asmx"
               xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://localhost:52876/VeebiTeenus/Service.asmx"
               xmlns:q1="http://tempuri.org/" binding="q1:ServiceSoap"
               xmlns="http://schemas.xmlsoap.org/disco/soap/" />
  <soap address="http://localhost:52876/VeebiTeenus/Service.asmx"
        xmlns:q2="http://tempuri.org/" binding="q2:ServiceSoap12"
        xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

Kolmandaks tuleb luua teenuse nimega .discomap fail, mis räägib, miks te need kaks eelmist faili tegite. Sinna tuleb lisada kaks elementi: esimene ütleb, kuhu faili panite kasutamise info e. lepingu so wsdl fail; ning teine ütleb, kus on info teenuse leidmiseks so. disco fail. Nagu kõik eelmised, on ka .discomap XML fail ning meie näite puhul võiks tema sisu olla järgmine:

```
<?xml version="1.0" encoding="utf-8"?>
<DiscoveryClientResultsFile
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Results>
    <DiscoveryClientResult
            referenceType="System.Web.Services.Discovery.ContractReference"
            url="http://localhost:52876/VeebiTeenus/Service.asmx?wsdl"
            filename="Service.wsdl" />
    <DiscoveryClientResult
   referenceType="System.Web.Services.Discovery.DiscoveryDocumentReference"
            url="http://localhost:52876/VeebiTeenus/Service.asmx?disco"
            filename="Service.disco" />
  </Results>
</DiscoveryClientResultsFile>
```

Sellega on viide loodud. Järgmiseks tuleb seda teenust kasutada. Selleks loome kohas, kus soovime teenust kasutada, selle teenuse instantsi ning kutsume välja sobiva meetodi:

```
TereTeenus.Service srv = new TereTeenus.Service();
Label1.Text = srv.Tere();
```

Nii lihtsalt see lähebki. Loodetavasti tekib rohkelt mõtteid, kus selliseid teenuseid oleks võimalik kasutada. Mõelge, kui tore oleks järgmine stsenaarium: loote veebipoodi ning teil on vaja teada, kui palju klient peab tellitud toodete pealt makse maksma. Selle välja selgitamiseks kasutate maksuameti poolt pakutavat maksuarvutusteenust. Seejärel on vaja teada, kui palju maksab tellitud toodete kliendile viimine: selleks kasutate mõne kullerfirma hinnapäringuteenust jne. Nii on võimalik väga lihtsa vaevaga pakkuda oma rakenduses väga keerulist funktsionaalsust.

PS! Ka Google otsing on saadaval veebiteenusena ;)

IIS

IIS teenus on olemas kõigil Windowsi serveritel ning Ärikasutuse tööjaama Windowsidel nagu (Windows 2000 Pro, Windows XP Pro, Vista Business jne).

Lisad

Järgnevalt mõned pikemad koodijupid, mida võib suurema ja dünaamilisema rakenduse loomisel vaja minna.

Ressursside hoidmine SQL Serveris – Resource Provider

Järgnev kood kirjutab üle mootori, mis tegeleb ressursside lugemisega. See võimaldab ressursse hoida .resx failide asemel SQL serveri andmebaasis.

Selleks, et allolev näide tööle läheks, tuleb:

- Lisada konfiguratsiooni ühendustekst, mis ühendab ressursse hoidva andmebaasiga. Ühendusteksti nimi omistage konstandile connStrName.

```
Lisada konfiguratsiooni element
<globalization culture="auto" uiCulture="auto"
resourceProviderFactoryType="OmaProjekt.SqlResourceProviderFactory" />
```

Loomulikult on võimalik allolev klass kirjutada märksa keerukamaks, nii et seda oskaks kasutada ka VisualStudio ressursside võlur ning samuti on võimalik lisada funktsionaalsus puuduvate ressursside automaatseks lisamiseks.

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Globalization;
using System.Resources;
using System.Text;
using System.Web;
```

```
using System.Web.Compilation;
namespace OmaProjekt
{
  public sealed class SqlResourceProviderFactory : ResourceProviderFactory
  {
    const string connStrName = "yhendusTxt";
    public override IResourceProvider
                             CreateGlobalResourceProvider(string classKey)
    {
      return new SqlResourceProvider(null, classKey);
    }
    public override IResourceProvider
                           CreateLocalResourceProvider(string virtualPath)
    {
      virtualPath = System.IO.Path.GetFileName(virtualPath);
      return new SqlResourceProvider(virtualPath, null);
    }
    private sealed class SqlResourceProvider : IResourceProvider
    {
      private string _virtualPath;
      private string _className;
      private IDictionary _resourceCache;
      private static object CultureNeutralKey = new object();
      public SqlResourceProvider(string virtualPath, string className)
      {
        _virtualPath = virtualPath;
        _className = className;
      }
      private IDictionary GetResourceCache(string cultureName)
      {
        object cultureKey;
        if (cultureName != null)
        {
          cultureKey = cultureName;
        }
        else
        {
          cultureKey = CultureNeutralKey;
        }
        if (_resourceCache == null)
        {
          _resourceCache = new ListDictionary();
        }
        IDictionary resourceDict =
                                 _resourceCache[cultureKey] as IDictionary;
        if (resourceDict == null)
        {
          resourceDict = SqlResourceHelper.GetResources(_virtualPath,
                                   _className, cultureName, false, null);
          _resourceCache[cultureKey] = resourceDict;
        }
        return resourceDict;
      }
      object IResourceProvider.GetObject(string resourceKey,
                                                    CultureInfo culture)
      {
        string cultureName = null;
        if (culture != null)
        {
          cultureName = culture.Name;
        }
        else
        {
          cultureName = CultureInfo.CurrentUICulture.Name;
        }
        object value = GetResourceCache(cultureName)[resourceKey];
        if (value == null)
        {
```

```
      // vajaliku kultuuri ressurss on puudu, kasutame ilma
      // kultuuriinfota ressurssi
      HttpContext.Current.Trace.Warn("Puuduv ressurss",
        "resourceKey=" + resourceKey +
        ";virtualPath=" + _virtualPath +
        ";className=" + _className +
        ";cultureName=" + cultureName);
      value = GetResourceCache(null)[resourceKey];
    }
    if (value == null)
    {
      // Vajalikku ressurssi pole üldse olemas
      HttpContext.Current.Trace.Warn("Puuduv ressurss",
        "resourceKey=" + resourceKey +
        ";virtualPath=" + _virtualPath +
        ";className=" + _className);
    }
    return value;
  }
  IResourceReader IResourceProvider.ResourceReader
  {
    get
    {
      return new SqlResourceReader(GetResourceCache(null));
    }
  }
}
private sealed class SqlResourceReader : IResourceReader
{
  private IDictionary _resources;
  public SqlResourceReader(IDictionary resources)
  {
    _resources = resources;
  }
  IDictionaryEnumerator IResourceReader.GetEnumerator()
  {
    return _resources.GetEnumerator();
  }
  void IResourceReader.Close()
  {
  }
  IEnumerator IEnumerable.GetEnumerator()
  {
    return _resources.GetEnumerator();
  }
  void IDisposable.Dispose()
  {
  }
}
internal static class SqlResourceHelper
{
  public static IDictionary GetResources(string virtualPath,
                string className, string cultureName, bool designMode,
                IServiceProvider serviceProvider)
  {
    SqlConnection con = new SqlConnection(
                           System.Configuration.ConfigurationManager.
                          ConnectionStrings[connStrName].ToString());
    SqlCommand com = new SqlCommand();
    if (!String.IsNullOrEmpty(virtualPath))
    {
      // Lokaalsete ressursside küsimine
      if (string.IsNullOrEmpty(cultureName))
      {
        // ilma kultuuriinfota ressursid
        com.CommandType = CommandType.Text;
        com.CommandText = "select resource_name, resource_value" +
                  " from ASPNET_GLOBALIZATION_RESOURCES" +
```

```
                " where resource_object = @virtual_path" +
                " and culture_name is null";
      com.Parameters.AddWithValue("@virtual_path", virtualPath);
    }
    else
    {
      com.CommandType = CommandType.Text;
      com.CommandText = "select resource_name, resource_value" +
                " from ASPNET_GLOBALIZATION_RESOURCES " +
                "where resource_object = @virtual_path " +
                "and culture_name = @culture_name ";
      com.Parameters.AddWithValue("@virtual_path", virtualPath);
      com.Parameters.AddWithValue("@culture_name", cultureName);
    }
  }
  else if (!String.IsNullOrEmpty(className))
  {
    // Globaalsete ressursside küsimine
    if (string.IsNullOrEmpty(cultureName))
    {
      // ilma kultuuriinfota ressursid
      com.CommandType = CommandType.Text;
      com.CommandText = "select resource_name, resource_value" +
                " from ASPNET_GLOBALIZATION_RESOURCES " +
                "where resource_object = @class_name" +
                " and culture_name is null";
      com.Parameters.AddWithValue("@class_name", className);
    }
    else
    {
      com.CommandType = CommandType.Text;
      com.CommandText = "select resource_name, resource_value " +
                "from ASPNET_GLOBALIZATION_RESOURCES where " +
                "resource_object = @class_name and" +
                " culture_name = @culture_name ";
      com.Parameters.AddWithValue("@class_name", className);
      com.Parameters.AddWithValue("@culture_name", cultureName);
    }
  }
  else
  {
    // Probleem lähteandmetega
    throw new Exception("SqlResourceHelper.GetResources()" +
        " - puuduvad parameetrid virtualPath või className");
  }
  ListDictionary resources = new ListDictionary();
  try
  {
    com.Connection = con;
    con.Open();
    SqlDataReader sdr =
                  com.ExecuteReader(CommandBehavior.CloseConnection);
    while (sdr.Read())
    {
      string rn = sdr.GetString(sdr.GetOrdinal("resource_name"));
      string rv = sdr.GetString(sdr.GetOrdinal("resource_value"));
      resources.Add(rn, rv);
    }
  }
  catch (Exception e)
  {
    throw new Exception(e.Message, e);
  }
  finally
  {
    if (con.State == ConnectionState.Open)
    {
      con.Close();
```

```
          }
        }
        return resources;
      }
    }
  }
}
```

Kasutajatunnuste hoidmine andmebaasis - MembershipProvider

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web;
using System.Web.Security;
public class KasutajaProvider : MembershipProvider
{
    #region abstraktsed meetodid ja propertid, mis tuleb kindlasti realiseerida
    public override string ApplicationName
    {
        get
        {
            throw new Exception("The method or operation is not implemented.");
        }
        set
        {
            throw new Exception("The method or operation is not implemented.");
        }
    }
    public override bool ChangePassword(string username, string oldPassword, string
newPassword)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override bool ChangePasswordQuestionAndAnswer(string username, string
password, string newPasswordQuestion, string newPasswordAnswer)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override MembershipUser CreateUser(string username, string password,
string email, string passwordQuestion, string passwordAnswer, bool isApproved,
object providerUserKey, out MembershipCreateStatus status)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override bool DeleteUser(string username, bool deleteAllRelatedData)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override MembershipUserCollection GetAllUsers(int pageIndex, int
pageSize, out int totalRecords)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override int GetNumberOfUsersOnline()
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override string GetPassword(string username, string answer)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override MembershipUser GetUser(object providerUserKey, bool
userIsOnline)
```

```
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override MembershipUser GetUser(string username, bool userIsOnline)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override string GetUserNameByEmail(string email)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override bool EnablePasswordReset
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override bool EnablePasswordRetrieval
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override MembershipUserCollection FindUsersByEmail(string emailToMatch,
int pageIndex, int pageSize, out int totalRecords)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override MembershipUserCollection FindUsersByName(string
usernameToMatch, int pageIndex, int pageSize, out int totalRecords)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override int MaxInvalidPasswordAttempts
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override int MinRequiredNonAlphanumericCharacters
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override int MinRequiredPasswordLength
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override int PasswordAttemptWindow
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override MembershipPasswordFormat PasswordFormat
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override string PasswordStrengthRegularExpression
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override bool RequiresQuestionAndAnswer
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override bool RequiresUniqueEmail
    {
        get { throw new Exception("The method or operation is not implemented."); }
    }
    public override string ResetPassword(string username, string answer)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override bool UnlockUser(string userName)
    {
        throw new Exception("The method or operation is not implemented.");
```

```
    }
    public override void UpdateUser(MembershipUser user)
    {
        throw new Exception("The method or operation is not implemented.");
    }
    public override bool ValidateUser(string username, string password)
    {
        SqlCommand cmd = new SqlCommand("TuvastaKasutaja_proc",
            new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng));
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.Add("@username", SqlDbType.NVarChar, 50).Value = username;
        cmd.Parameters.Add("@password", SqlDbType.NVarChar, 50).Value = password;
        cmd.Parameters.Add("@KasutajaID", SqlDbType.Int).Direction =
ParameterDirection.Output;
        cmd.Connection.Open();
        cmd.ExecuteNonQuery();
        cmd.Connection.Close();
        bool KasutajaOK = false;
        if (cmd.Parameters["@KasutajaID"].Value != System.DBNull.Value)
        {
            HttpContext.Current.Session["ActiveInimeneID"] =
cmd.Parameters["@KasutajaID"].Value.ToString();
            KasutajaOK = true;
        }
        return KasutajaOK;
    }
    #endregion
}
```

Kasutajagruppide hoidmine andmebaasis – RoleProvider

```
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web;
using System.Web.Security;
public class SqlRoleProvider : RoleProvider
{
    public override void AddUsersToRoles(string[] usernames, string[] roleNames)
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "AddUserToRole_proc";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.Add("@UserName", SqlDbType.NVarChar, 50);
        cmd.Parameters.Add("@RoleName", SqlDbType.VarChar, 50);
        cmd.Connection.Open();
        foreach (string user in usernames)
        {
            cmd.Parameters["@UserName"].Value = user;
            foreach (string role in roleNames)
            {
                cmd.Parameters["@RoleName"].Value = role;
                cmd.ExecuteNonQuery();
            }
        }
        cmd.Connection.Close();
    }
    public override string ApplicationName
```

```
    {
        get
        {
            return "Tallinna Täiskasvanute Gümnaasiumi testimiskeskus";
        }
        set
        {
            throw new Exception("The method or operation is not implemented.");
        }
    }
    public override bool IsUserInRole(string username, string roleName)
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "IsUserInRole_proc";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.AddWithValue("@UserName", username);
        cmd.Parameters.AddWithValue("@RoleName", roleName);
        cmd.Parameters.Add("@OK", SqlDbType.Bit).Direction =
ParameterDirection.Output;
        cmd.Connection.Open();
        cmd.ExecuteNonQuery();
        cmd.Connection.Close();
        return Convert.ToBoolean(cmd.Parameters["@OK"].Value);
    }
    public override string[] GetRolesForUser(string username)
    {
        string[] rollid = new string[1] { "ADMINISTRATORS" };
        return rollid;
    }
    public override void CreateRole(string roleName)
    {
        if (HttpContext.Current.User.IsInRole("ADMINISTRATORS"))
        {
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "CreateRole_proc";
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
            cmd.Parameters.AddWithValue("@RoleName", roleName);
            cmd.Connection.Open();
            cmd.ExecuteNonQuery();
            cmd.Connection.Close();
        }
    }
    public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
    {
        bool tulemus = false;
        if (HttpContext.Current.User.IsInRole("ADMINISTRATORS"))
        {
            SqlCommand cmd = new SqlCommand();
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
            cmd.Parameters.AddWithValue("@RoleName", roleName);
            cmd.Connection.Open();
            if (throwOnPopulatedRole)
            {
                cmd.CommandText = "RolePopulated_proc";
                cmd.Parameters.AddWithValue("@Tulemus",
throwOnPopulatedRole).Direction = ParameterDirection.Output;
                cmd.ExecuteNonQuery();
                if (Convert.ToBoolean(cmd.Parameters["@Tulemus"].Value))
                {
```

```
                    throw new Exception("Ei saa kustutada rolli, kuhu kuuluvad
kasutajad!");
                }
                else {
                    cmd.Parameters.Remove(cmd.Parameters["@Tulemus"]);
                    cmd.CommandText = "DeleteRole_proc";

                    cmd.ExecuteNonQuery();
                    tulemus = true;
                }
            }
            else
            {
                cmd.Parameters.Remove(cmd.Parameters["@Tulemus"]);
                cmd.CommandText = "DeleteRole_proc";
                cmd.ExecuteNonQuery();
                tulemus = true;
            }
            cmd.Connection.Close();
        }
        return tulemus;
    }
    public override bool RoleExists(string roleName)
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "RoleExists_proc";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.AddWithValue("@RoleName", roleName);
        cmd.Parameters.Add("@Tulemus", SqlDbType.Bit).Direction =
ParameterDirection.Output;
        cmd.Connection.Open();
        cmd.ExecuteNonQuery();
        cmd.Connection.Close();
        return Convert.ToBoolean(cmd.Parameters["@Tulemus"].Value);
    }
    public override void RemoveUsersFromRoles(string[] usernames, string[]
roleNames)
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "RemoveUserFromRole_proc";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.Add("@UserName", SqlDbType.NVarChar, 50);
        cmd.Parameters.Add("@RoleName", SqlDbType.VarChar, 50);
        cmd.Connection.Open();
        foreach (string user in usernames)
        {
            cmd.Parameters["@UserName"].Value = user;
            foreach (string role in roleNames)
            {
                cmd.Parameters["@RoleName"].Value = role;
                cmd.ExecuteNonQuery();
            }
        }
        cmd.Connection.Close();
    }
    public override string[] GetUsersInRole(string roleName)
    {
        ArrayList kasutajad = new ArrayList();
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "GetUsersInRole";
        cmd.CommandType = CommandType.StoredProcedure;
```

```
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.AddWithValue("@RoleName", roleName);
        cmd.Connection.Open();
        SqlDataReader lugeja = cmd.ExecuteReader();
        while (lugeja.Read()) {
            kasutajad.Add(lugeja["KNimi"].ToString());
        }
        lugeja.Close();
        cmd.Connection.Close();
        return (string[])kasutajad.ToArray(typeof(string));
    }
    public override string[] GetAllRoles()
    {
        ArrayList rollid = new ArrayList();
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "GetAllRoles";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Connection.Open();
        SqlDataReader lugeja = cmd.ExecuteReader();
        while (lugeja.Read())
        {
            rollid.Add(lugeja["Nimi"].ToString());
        }
        lugeja.Close();
        cmd.Connection.Close();
        return (string[])rollid.ToArray(typeof(string));
    }
    public override string[] FindUsersInRole(string roleName, string
usernameToMatch)
    {
        ArrayList kasutajad = new ArrayList();
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "FindUsersInRole_proc";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = new
SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionStri
ng);
        cmd.Parameters.AddWithValue("@RoleName", roleName);
        cmd.Parameters.AddWithValue("@usernameToMatch", usernameToMatch);
        cmd.Connection.Open();
        SqlDataReader lugeja = cmd.ExecuteReader();
        while (lugeja.Read())
        {
            kasutajad.Add(lugeja["KNimi"].ToString());
        }
        lugeja.Close();
        cmd.Connection.Close();
        return (string[])kasutajad.ToArray(typeof(string));
    }
}
```