**Andmebaasipõhiste veebirakenduste arendamine Microsoft Visual Studio   
ja SQL Server’i baasil**

**SQL ja andmed**

Tallinn  
2011

**Sisukord**

[SQLi keel 4](#_Toc294015946)

[Microsoft SQL Server 2008 R2 5](#_Toc294015947)

[SQL Server 2008 R2 perekond 5](#_Toc294015948)

[SQL Server Express Edition 6](#_Toc294015949)

[Põhivõimalused 16](#_Toc294015950)

[Töö alustamine 16](#_Toc294015951)

[Andmebaasi loomine 18](#_Toc294015952)

[Lihtsamad päringud 26](#_Toc294015953)

[Tabelite vahelised seosed 39](#_Toc294015954)

[Tabelite ühendamine päringutes 41](#_Toc294015955)

[Edasijõudnutele 44](#_Toc294015956)

[Pikemad päringud 44](#_Toc294015957)

[Keerukamad seosed tabelite vahel 57](#_Toc294015958)

[Alampäringud 66](#_Toc294015959)

[Lisavõimalused 72](#_Toc294015960)

[XML andmete kasutamine 94](#_Toc294015961)

[Varukoopia 102](#_Toc294015962)

[Andmetele ligipääs ADO.NET 104](#_Toc294015963)

[Andmeallika külge ühendumine 107](#_Toc294015964)

[Töötamine andmebaasiga ühendatud keskkonnas 110](#_Toc294015965)

[XxxCommand 111](#_Toc294015966)

[Parameetrite kasutamine 112](#_Toc294015967)

[Ridade lugemine väljundist (DataReader) 113](#_Toc294015968)

[Transaktsioonid 114](#_Toc294015969)

[Töötamine ühenduseta keskkonnas (DataSets) 115](#_Toc294015970)

[Olemasolevate andmete põhjal DataSeti loomine 118](#_Toc294015971)

[XML 124](#_Toc294015972)

[XML’i kirjutamise reeglid 125](#_Toc294015973)

[Reeglid 125](#_Toc294015974)

[XML’i elemendid 126](#_Toc294015975)

[Atribuudid 127](#_Toc294015976)

[XHTML 128](#_Toc294015977)

[Nimeruum 129](#_Toc294015978)

[XML’i valideerimine 130](#_Toc294015979)

[XML skeemid 130](#_Toc294015980)

[XMLi kasutamine SqlServeris 132](#_Toc294015981)

[XMLi genereerimine relatsioonilistest andmetest 132](#_Toc294015982)

[XML andmetüübi kasutamine 135](#_Toc294015983)

[XML andmete kasutamine .NET raamistikus 139](#_Toc294015984)

[XMLi parsimine 139](#_Toc294015985)

[XMLi valideerimine 143](#_Toc294015986)

[XMLi salvestamine 145](#_Toc294015987)

[Kokkuvõte 147](#_Toc294015988)

# 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 R2 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 R2

SQL Server 2008 R2 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. | http://www.microsoft.com/global/sqlserver/2008/en/us/PublishingImages/diag-sql2008-lg.gif |

### SQL Server 2008 R2 perekond

SQL Server 2008 R2 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 [www.microsoft.com/sql](http://www.microsoft.com/sql)
* Compact Edition – spetsiaalne SQL Serveri versioon mobiilsete seadmete tarbeks

Täpsem info erinevate SQL Serveri perekonnaliikmete kohta leiate aadressidelt www.microsoft.com/sql

### SQL Server Express Edition

SQL Server Express Edition on mõeldud SQL Serveriga tutvumiseks ning ka ärikasutuseks väikeste lahenduste puhul. Express versiooni saab tõmmata aadressilt [www.microsoft.com/express/Database/](http://www.microsoft.com/express/Database/)

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 10 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 R2 Express Edition on toetatud Windows XP, Windows 2003, Windows Vista, Windows 2008 ja Windows 7 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:

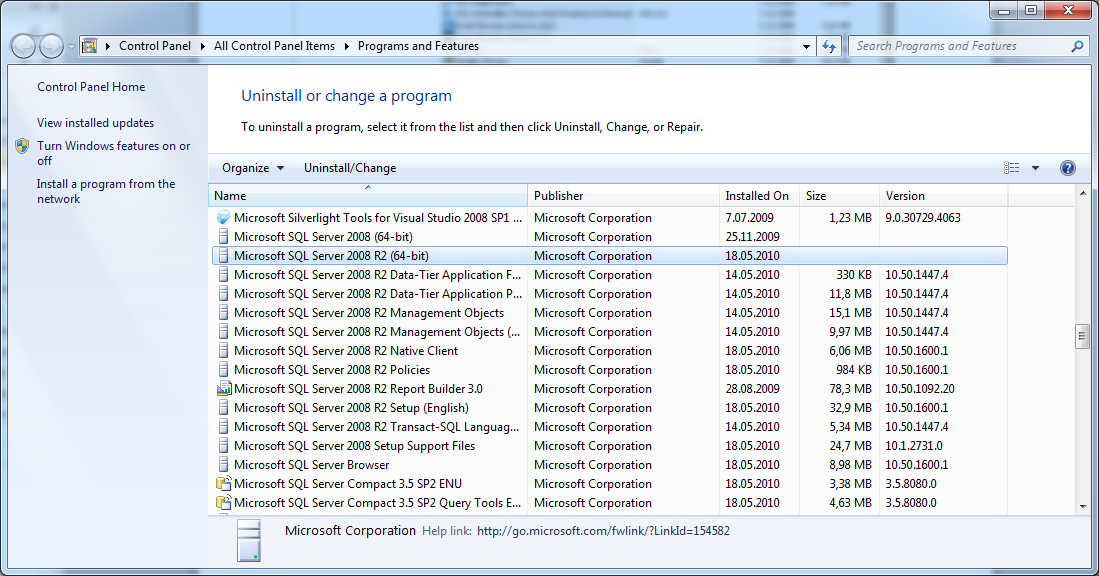
* Tuleb installeerida .NET raamistik 4.0
* Veenduge, et teie masinas oleks Windows Installer 4.5. <http://support.microsoft.com/kb/942288>
* Kui paigaldate SQL Serveri koos haldusvahenditega siis tuleb installeerida ka Windows PowerShell 2.0. [www.microsoft.com/powershell](http://www.microsoft.com/powershell)
* Tuleb eemaldada kõik varasemad SQL Server Beta ning CTP versioonid
* Installida SQL Server 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.

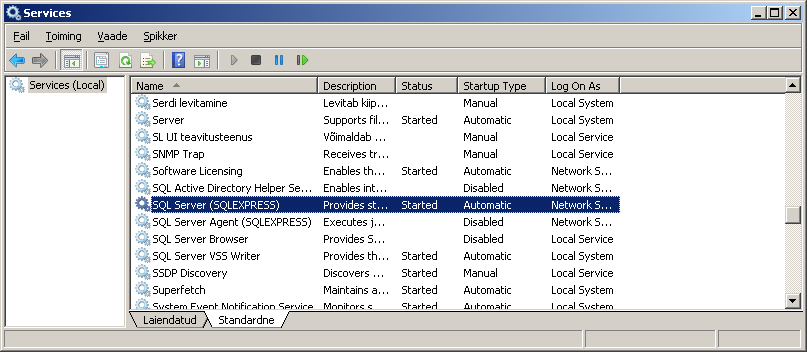
#### Installeerimine

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.



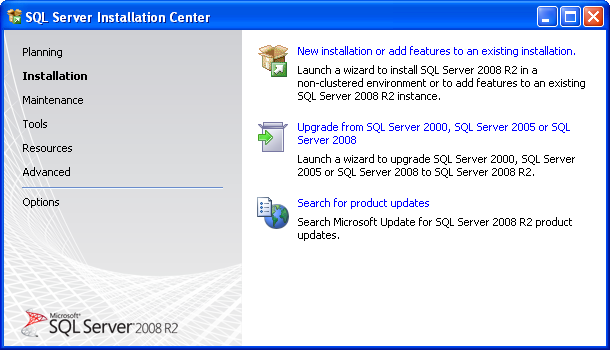
Ning lisaks sellele ka Windowsi teenuste alt.



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 installation or add features to an existing installation või juhul kui olete eelnevalt juba mõne varasema SQL Serveri Expressi versiooni paigaldanud ning soovite versiooni uuendada siis Upgrade from SQL Server 2000, SQL Server 2005 or SQL Server 2008.

****

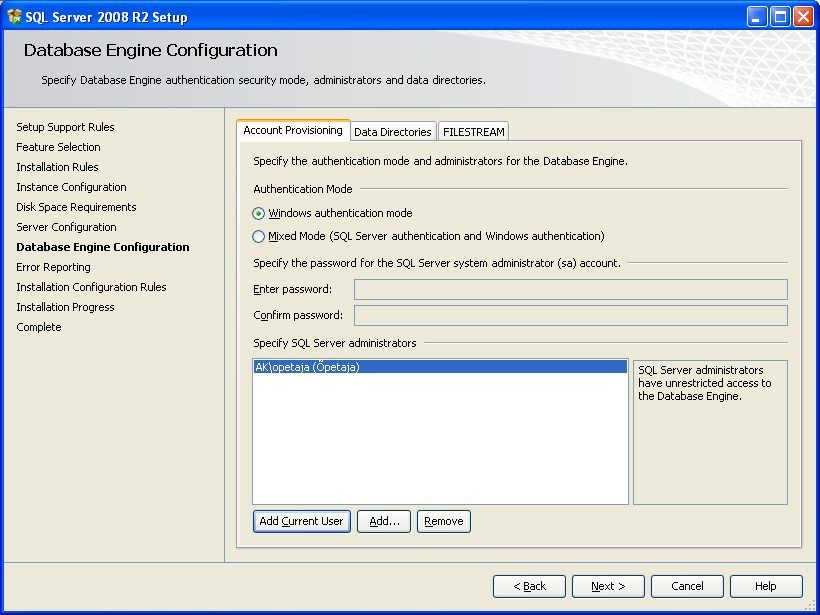
Peale seda, kui installiprogramm on endale vajalikud lisavidinad ära installeerinud saate valida, millised komponendid paigaldada. Sõltuvalt käivitatud SQL installeri versioonist võib komponentide valik olla erinev, kuid kindlasti on vaja peale panna „Database Engine Service“ ning soovituslik oleks paigadada ka haldusvahend „Management Tools“.

Peale sobivate komponentide valimist tuleb teil otsustada kuhu SQL server installeerida ja kas loote SQLi default instansi – SQL teenuse mille poole pöördute masina nime järgi või nimelise instance-i – SQL teenuse, mille poole pöördute nimega masinimi\instancenimi. Default instance saab igas masinas olla vaid üks kuid nimelisi võib olla mitu e. võite soovikorral oma masinasse paigaldada nt 10 SQL teenust. Samas aknas loetletakse teiele ette ka kõik olemasolevad SQL instanced.

Selles materjalis tehtud näited on kõik loodud SQL nimelise instance SQLExpress baasil.

Liikudes järgmisele lehele saate valida, millise konto all SQL Server töötab. SQL Server Database Engine pange esialgu tööle NT AUTHORITY\SYSTEM nime all. Selle sama vormi teisel vahelehel Collation on kirjas serveris kasutatav tähestik. Soovitav oleks kasutada eesti tähestikku. See võimaldab kõik meil kasutatavad täpitähed järjestada korrektselt.

Liikudes edasi küsitakse, kellest saab selle SQL serveri adminsitraator. Vaadake, et selles loetelus oleks kindlasti olemas kas teie nimi või siis mõni kasutajagrupp kuhu te kuulute. Oma kasutajakonto lisamiseks administraatorite nimistusse vajutage nupule „Add Current User“.



|  |  |
| --- | --- |
| Seejärel next lõpuni ning oodake kuni kõik on edukalt paigaldatud.  Edukust peab kinnitama lõpus teade „installation completed successfully“ . |  |

#### 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.

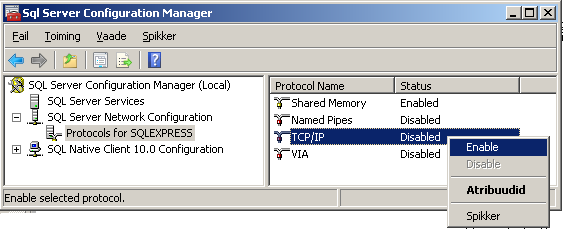
Ligipääsu jagamine serverile käib kolmel erineval tasemel:

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

##### Ligipääsukanal üle võrgu

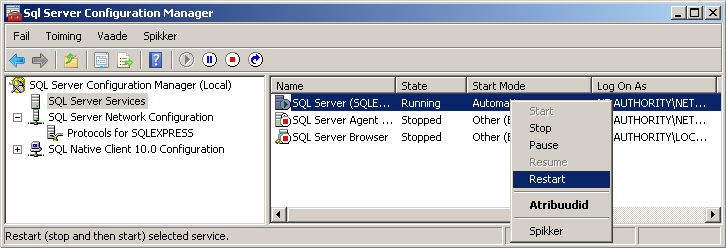
|  |  |
| --- | --- |
|  | Ligipääsukanalite haldamiseks saate kasutada SQL Server Configuration Manageri koosesisus olevat SQL Server Network Configuration utiliiti. |

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.

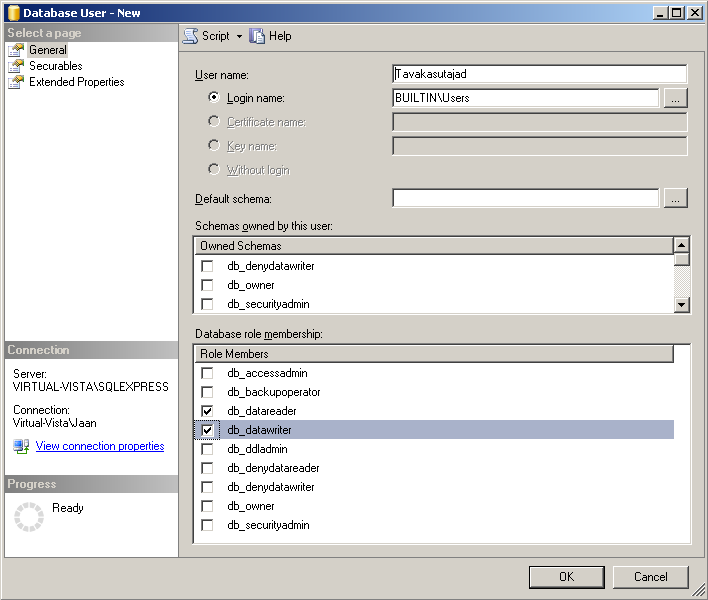
##### Ligipääs andmebaasile

|  |  |
| --- | --- |
| 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… |  |

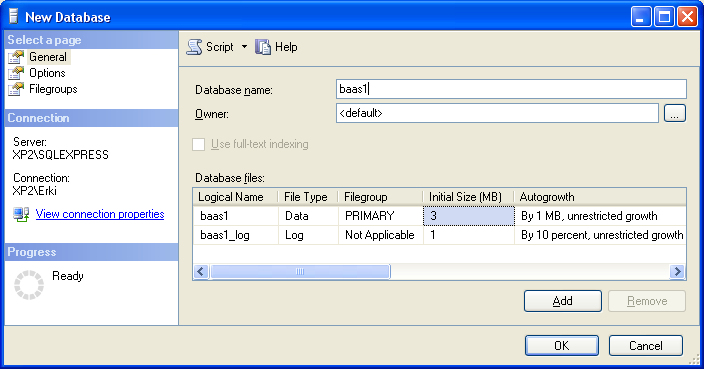
Baasi loomise juures küsitakse kõigepealt andmebaasi nime. Saagu selleks baas1. Baasile saab hulga omadusi määrata, mis suuremate ja tihedasti kasutatavate baaside administreerimisel on küllalt tähtsad.. Path teatab, kuhu kettale ja kataloogi andmed salvestatakse. Initial Size kaudu öeldakse algne baasi 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])

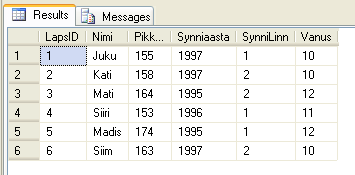
VALUES ('Siim', 163, 1997, 2)

Avades tabeli vaatamiseks või kirjutades SQL Lause kujul:

SELECT \*

FROM dbo.Laps\_tbl

Saame tulemuseks tabeli, milles 6 rida:



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

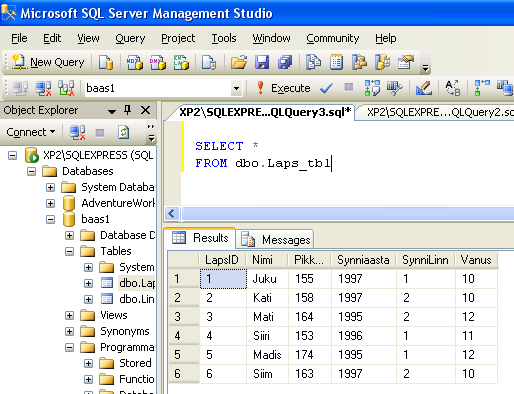
Enne, kui andmebaasi loomisega edasi läheme vaatame, kuidas oleks võimalik juba sisestatud andmeid vaadata ja uurida.

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

SELECT \*

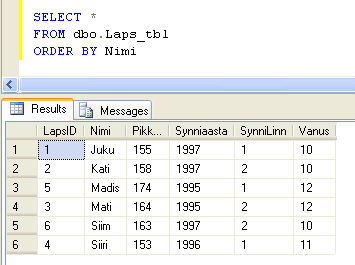
FROM dbo.Laps\_tbl

Tulemusena joonistub rakenduse allserva kogu tabelitäis andmeid koos tulpade nimedega.



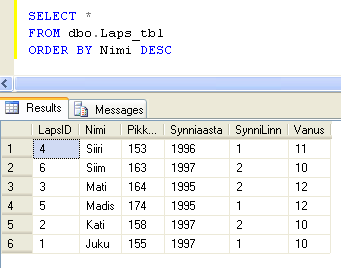
Sellist päringut kasutatakse vaid erandolukordades ning väga väikeste tabelite juures. Reaalses tööolukorras tuleks kindlasti loetleda ülesse kõik väljad, mida soovite vaadata ning seada piirangud ridade arvule!

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

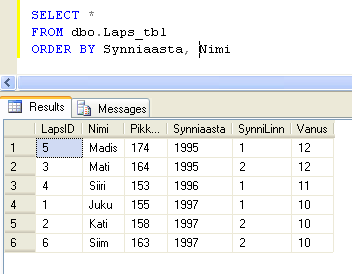


Ja tulevadki andmed nimede järgi sordituna. Et esimeses tulbas olevad id-d näevad juhuslikult segi paisatutena välja, see on täiesti loomulik. Kui sorditakse nime järgi, siis tõstetakse read niimoodi ümber, et eesnimed lähevad tähestikulisse järjekorda. Iga rea andmed aga kuuluvad endiselt kokku. Nii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

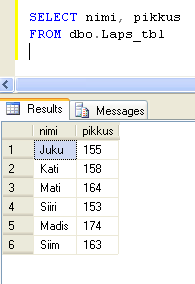
Tahtes sorteerimisjärjekorra muuta vastupidiseks, tuleb tulba nimele lisada tähed DESC (sõnast descending). Ja ongi Siiri esimene ja Juku viimane.



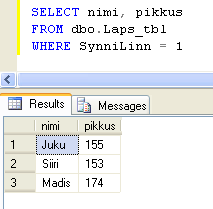
Järjestust määravaid tulpi võib olla mitu. Sellisel juhul tuleb ORDER BY järgi loetleda väljad tähtsuse järgi. Nt võttes sorteerimise aluseks sünniaasta ning seejärel nime saame, et sõnniaastad on sorteeritud kasvavasse järjekorda ning kui samal aastal on sündinud mitu last on nad sorteeritud nimede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



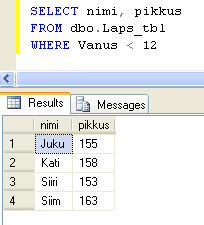
Sugugi alati pole andmete juures vaja kõiki tulpasid näha. Kui soovin vaadata vaid nime ja pikkust võin selle info panna ka SELECT lausesse, loetledes kõik vajalikud väljad SELECT´ i järel.



Samuti saab seada piirangu ridade näitamise suhtes. Siin vaid lapsed, kelle sünnilinn on Tallinn e. linna kood on 1



Või vaatame, millised lapsed on nooremad kui 12:

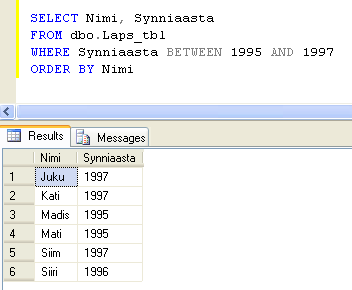


Kui tulemusse tekivad korduvad read saame nendest vabaneda kasutades DISTINCT märksõna. Näiteks soovime välja selgitada milliste sünniaastatega lapsed meil tabelis on? Kui kirjutame SELECT´i ilma DISCTINCT märksõnata saame loetelu, kui kõigi laste sünniaastad, kui lisame DISTINCTI saame kõik erinevad sünniaastad:

|  |  |
| --- | --- |
|  |  |

Võime piiranguid seada ka numbrivahemike järgi. Selleks on kaks võimalust:

* Kasutada BETWEEN operaatorit
* Kombineerida kaks võrratust AND operaatoriga



Sama tulemuse saaksime ka järgmise SQL lausega:

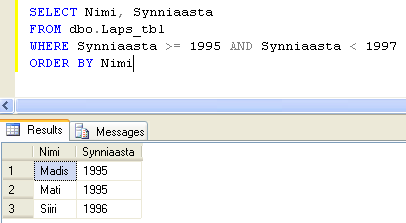
SELECT Nimi, Synniaasta

FROM dbo.Laps\_tbl

WHERE Synniaasta >= 1995 AND Synniaasta <= 1997

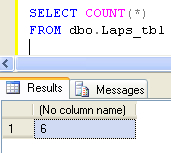
ORDER BY Nimi

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:

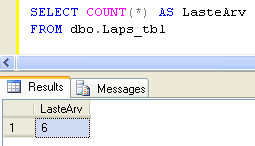


#### Agregaatfunktsioonid

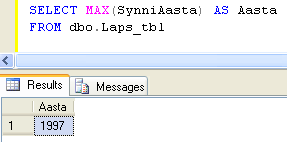
Lihtsamad kokkuarvutamised saab samuti SQL-keele abil ära teha, nende jaoks pole vaja täiendavate programmide abi vaja otsida. Alljärgnevalt toodud funktsioonide töö tulemuseks on siinsetel juhtudel terve tabeli kohta vaid üks arv. Aga eks see summa, suurima, vähima, koguse või keskmise leidmisel nii olegi. Tahtes päringust välja tulnud ridu kokku lugeda, aitab funktsioon COUNT(\*). Saame teada, et praegu nimekirjas olevaid lapsi on 6.



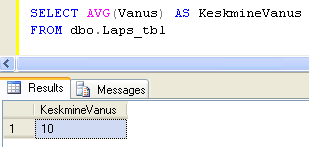
Nagu näeme puudub saadud tulemusel pealkiri. Kui soovime SELECT´is loetletud välju ümber nimetada või neile nimesid anda tuleb kasutada AS märksõna mille järgi saate kirjutada sobiva nime.



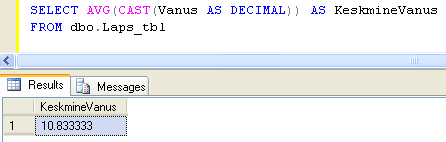
Suurima sünniaasta saab kätte käsuga MAX. Sarnaselt töötab ka MIN.



Või leiame tabelis olevate laste keskmise vanuse kasutades AVG funktsiooni (sõnast average)

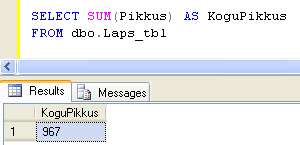


Nagu näeme on tulemus täpselt sama täpne kui lähteandmed. Kui soovime andmetüüpi arvutuste käigus muuta tuleb lähteandmed teisendada sobivale kujule ning sooritada tehte pärast teisendust. Näiteks kui teisendame esmalt kõik vanused reaalarvudeks ning arvutame seejärel keskmise saame hoopis põnevama tulemuse:



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.



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

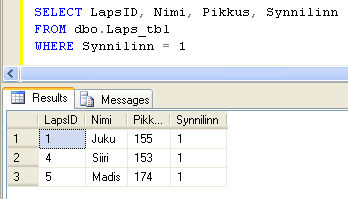
Selles lauses on näha ka kui oluline on unikaalne kood. Kui koode ei oleks ning me püüaksime muuta nime baasile e. Nimi = ’Mati’ siis muudetaks kõigi Matide pikkused!

|  |  |
| --- | --- |
| Enne: | Pärast: |

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:



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: | Pärast: |

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 | Pärast |

#### 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 | Pärast |

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)

|  |  |
| --- | --- |
|  |  |

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)

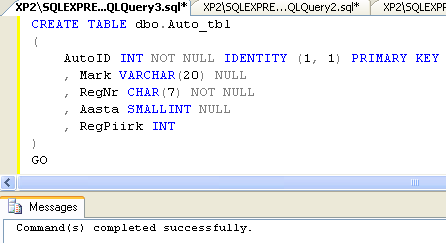
* Loo autode tabel, kus igaühe kohta on kirjas mark, registrinumber ja tootmisaasta ning registripiirkond
* Sisesta järgmised autod:

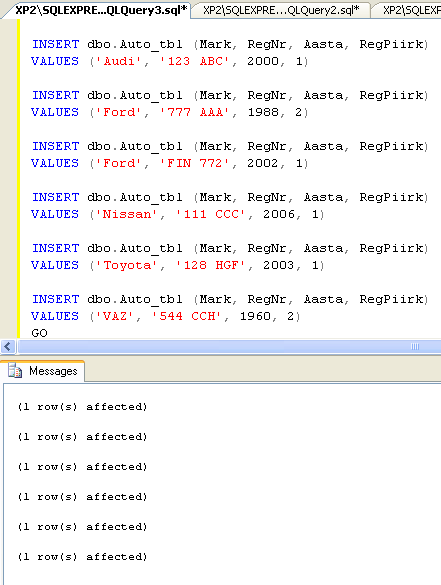
| Mark | RegNr | Aasta | RegPiirk |
| --- | --- | --- | --- |
| Audi | 123 ABC | 2000 | 1 |
| Ford | 777 AAA | 1988 | 2 |
| Ford | FIN 772 | 2002 | 1 |
| Nissan | 111 CCC | 2006 | 1 |
| Toyota | 128 HGF | 2003 | 1 |
| VAZ | 544 CCH | 1960 | 2 |

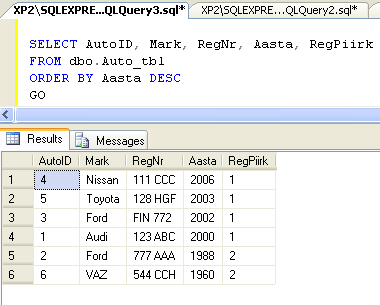
* Järjesta autod tootmisaasta järgi kahanevasse järjekorda
* Väljasta kõik erinevad margid
* Väljasta enne 1993. aastat toodetud autode registrinumbrid
* Väljasta enne 1993. aastat toodetud autode registrinumbrid tähestiku järjekorras
* Väljasta autode kõige varasem väljalaskeaasta (MIN)
* Muuda registrinumbrit 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)

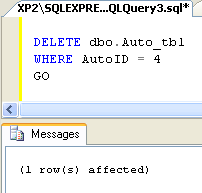


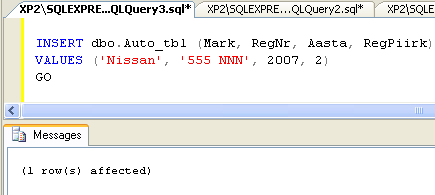


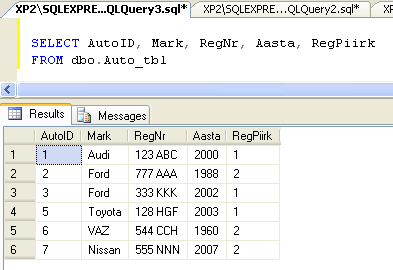


|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
|  |  |







### 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 |
| 1 | Miisu | 5 |
| 2 | Pauka | 7 |

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 |
| 5 | Siiri | 153 | 1996 | 1 | Miisu | 5 |
| 7 | Siim | 163 | 1997 | 2 | Pauka | 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 |
| 5 | Siiri | 153 | 1996 | 1 | Miisu | 5 |
| 7 | Siim | 163 | 1997 | 2 | Pauka | 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 |
| 7 | Siim | 163 | 1997 | 2 | Pauka | 7 |
| 5 | Siiri | 153 | 1996 | 1 | Miisu | 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 |
| 1 | Miisu | 5 | 5 | Siiri | 153 | 1996 |
| 2 | Pauka | 7 | 7 | Siim | 163 | 1997 |

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

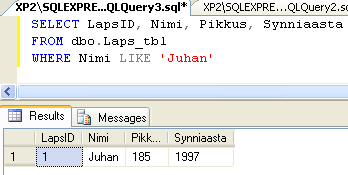
Siim Pauka

## Edasijõudnutele

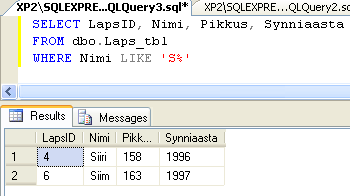
### Pikemad päringud

#### LIKE

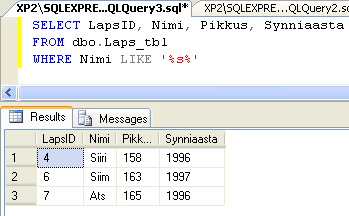
Tekstidest otsimise juures aitab sobivat vastet leida võrdlus LIKE. Kui sinna anda ette lihtsalt tekst, siis käitub LIKE võrdusmärgina:



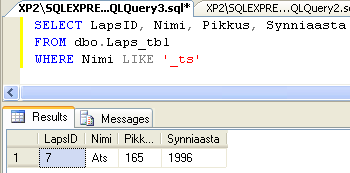
Käskluse põhivõlu seisneb aga võimaluses otsida metamärkide järgi. Alljoon \_ tähistab ühte suvalist sümbolit ning protsendimärk % suvalist arvu suvalisi sümboleid. Näiteks S% vastab kõigile tekstidele, mis algavad S-iga.



Ning %s% vastab kõigile tekstidele, mis sisaldavad s-i.

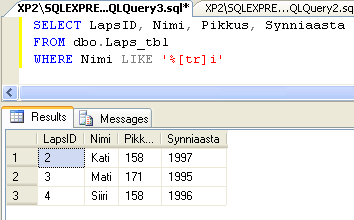


Alljoon, nagu öeldud, vastab vaid ühele sümbolile.



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.



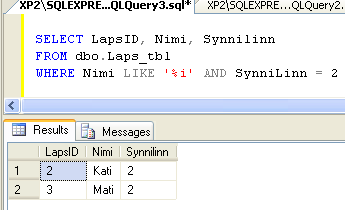
LIKE võtab arvutilt küllalt palju mõtlemisaega – seetõttu ei soovitata suuremate tabelite juures vastavat käsklust ilmaasjata pruukida. Samas aga on see küllalt mugav vahend sobivate nimede ja koodide leidmiseks, nii et mõõduka kasutamise juures on ta täiesti omal kohal.

Pikemate, nö täistekstide juures, on SQL-serveril omad käsud CONTAINS ning FREETEXT, mille abil võimalik omale sobivaid andmeid kätte saada.

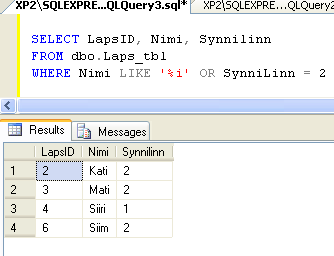
Kuna täisteksti otsinguid ei tee mitte SQL Server vaid Windowsi Indekseerimise ja otsimise teenus siis vajab nende funktsioonide kasutamine administraatori poolset andmete ettevalmistust ning hoolitust.

#### 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)

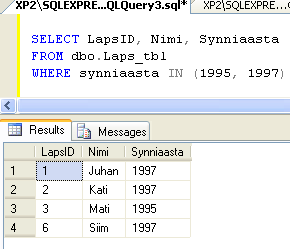


Edasi kõik i-lõpulised nimed pluss veel lisaks kõik, kes sündinud Tartus. Nimesid korduvalt siiski ei näidata. Kuigi Siiri ja Mari vastavad mõlemale tingimusele, on nad nimekirjas siiski ainult ühe korra.



#### 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

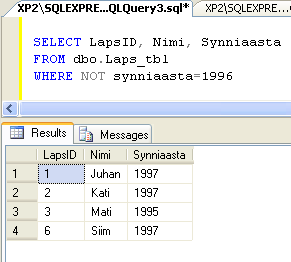
FROM dbo.Laps\_tbl

WHERE synniaasta = 1995 OR synniaasta = 1997

Kuigi tulemus on sama ja SQL Server käsitleb neid päringuid ühtemoodi on IN operaatori kasutamine ülevaatlikum.

#### NOT

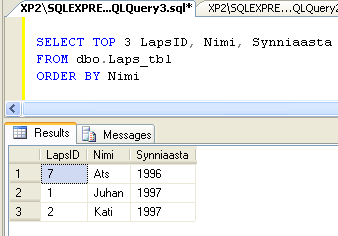
NOT pöörab tulemuse ümber. Ehk siis kõik need lapsed, kes ei ole sündinud aastal 1996.



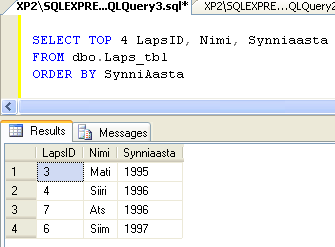
Võimaluse korral soovitatakse NOTi mitte pruukida, sest enamasti peab sel juhul andmebaasimootor vaatama läbi tabeli kõik read, mis on suurte andmemahtude juures küllalt suur töö. Aga kui muidu läbi ei saa, eks siis peab ikka selle sõna kirjutama.

#### TOP, päringu algusosa

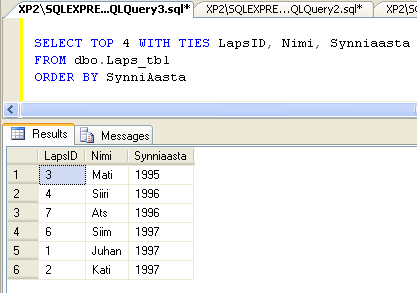
Lihtsalt andmetest ülevaate saamiseks ei ole vaja sageli kõike näha. Samuti, kui soovime viite kiiremat jooksjat või viite vanemat autot, siis on mugav, kui päring kohe annabki meile soovitu kätte, mitte ei pea hakkama ise pead vaevama, kuidas soovitud kohast andmeid võtma hakata. Laste tabelist tähestiku järjekorras kolm esimest nime näiteks saab kätte nii.



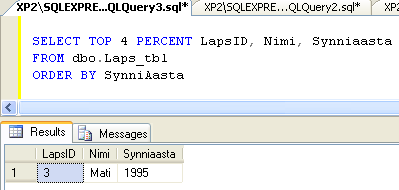
Tahtes saada neli vanemat last, tuleb andmed sorteerida sünniaasta järgi.



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

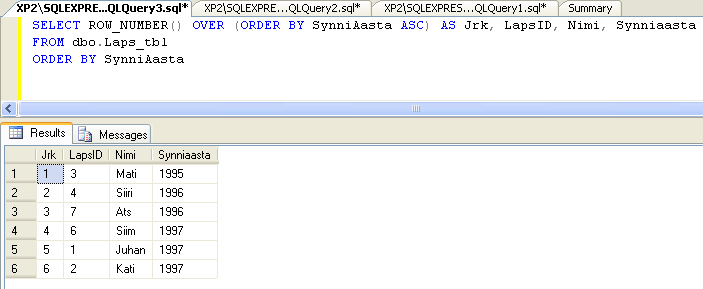
FROM lapsed

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.



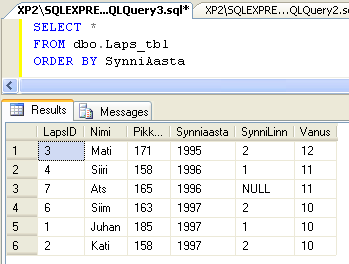
Süntaks on siis järgmine: ROW\_NUMBER() OVER (partitsioon) st OVER märksõna järgi sulgudes tuleb öelda, mis moodi on read nummerdatud. Antud näites nummerdatakse sünniaastate järgi kasvavasse järjekorda.

#### Grupeerimine

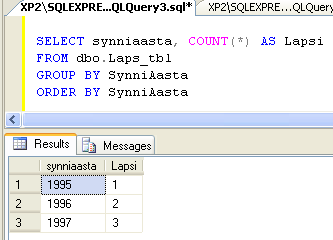
Eelnevalt uurisime agregaatfunktsioone suurima, vähima, keskmise, summa ja koguse leidmiseks. Nad on kogu tabeli kohta head abilised. WHERE-tingimuse abil saab filtreerida sobiva tunnuse väärtuse alusel read välja ning siis nende põhjal kokkuvõtteid teha. Näiteks leida kõikide nende laste keskmise pikkuse, 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.

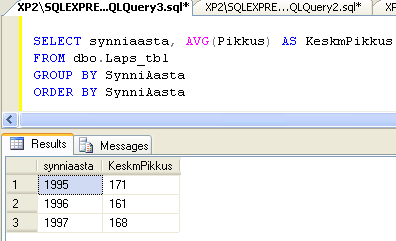
Nüüd siis mõned näited ja seletused, et siinse kirjutise lugejad ei peaks sama pikka ja okkalist teed läbi käima. Algul meeldetuletuseks laste andmed, et oleks näha, mida ja kuidas grupeeritakse.



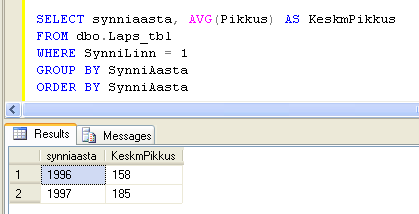
Tahtes iga aasta kohta teada, mitu last meie nimekirjast vastaval aastal sündinud on, aitab järgnev lause. COUNT(\*) loeb kokku plokis olevad read. Et päringu lõpus on GROUP BY synniaasta, siis loetakse iga erinev sünniaasta omaette plokiks. Tahtes sünniaastat ka ennast näha, tuleb ka see SELECT´i järele 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.



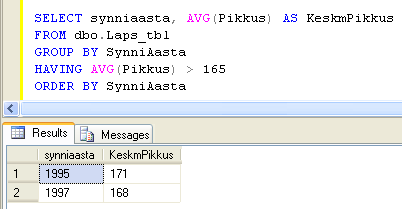
Sarnaselt nagu võib ridu kokku lugeda, saab ka teisi grupeerimisfunktsioone kasutada. Siin leitakse iga sünniaasta kohta sealsete laste keskmine pikkus.



Kui rakendame päringule piiranguid WHERE abil siis esmalt filtreeritakse lähteandmed ning alles peale seda hakatakse gruppe looma:

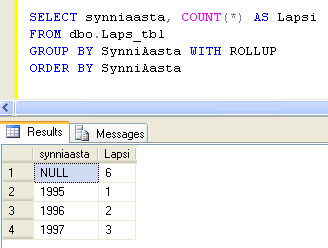


Kui soovime juba grupeeritud tulemusele piiranguid seada siis saame kasutada HAVING lauseosa. Näiteks soovime moodustada korvpallimeeskonda ning tahame teada, milliste vanusegruppide keskmine pikkus on üle 165 cm



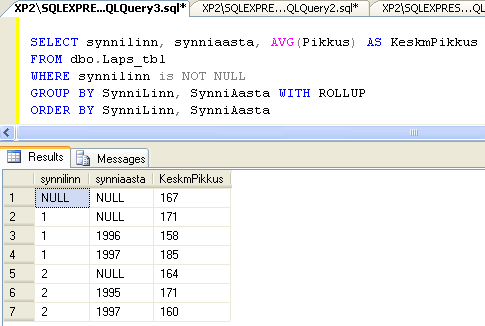
#### ROLLUP, gruppide koondinfo

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.

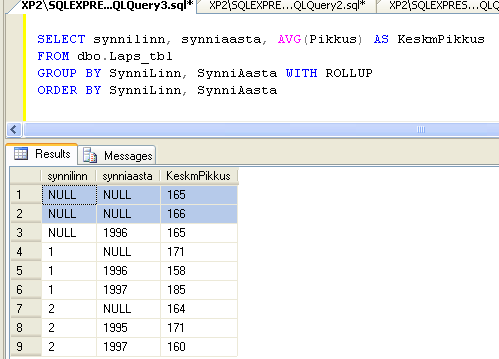


Kui grupeeritavaid tulpasid on rohkem, siis saab ka sellist lisastatistikat rohkem välja lugeda. Järgnevas näites grupeeriti lapsed sünnilinna ja sünniaasta järgi. See tähendab, et ühte gruppi sattunuksid nad vaid juhul, kui nad sündinuksid samal aastal ja oleksid ühepikkused. Iga muu kombinatsioon annab uue grupi. Nii see loend siis ka tuleb, kui algusest lugema hakata.

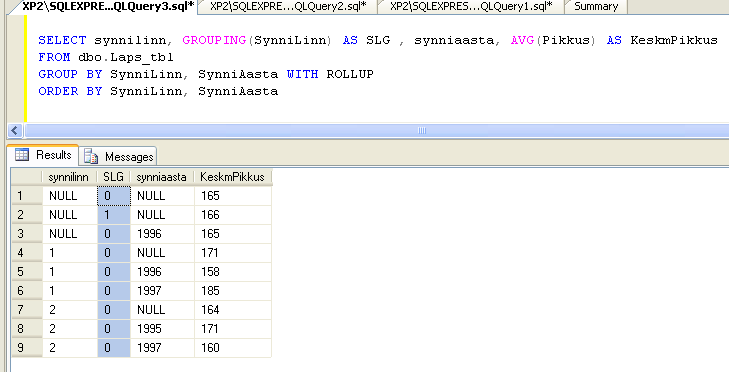
Kõigepealt esimeses reas teatatakse kõigi laste keskmine pikkus 167. Seejärel järgmises reas on esimese linna e. Tallinna laste keskmine pikkus 171. Seejärel tulevad keskmised pikkused Tallinna lastel, kes sündinud aastatel 1996 ja 1997. Viiendas reas on Tartu laste keskmine pikkus ning see järel aastatel 1995 ja 1997 sündinud Tartu laste keskmised pikkused.



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:



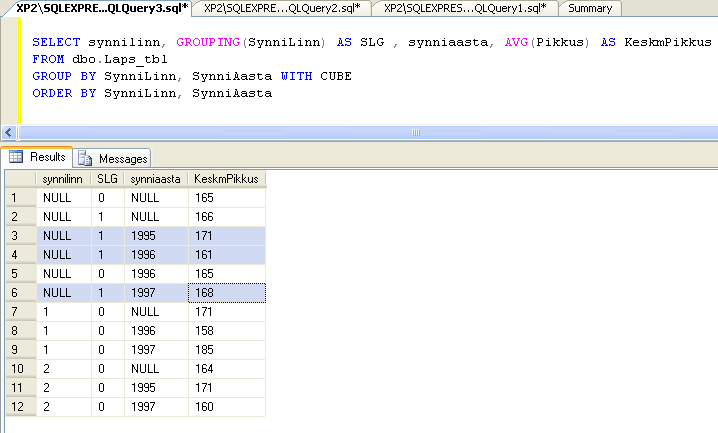
Tekib küsimus kus on kõigi laste keskmine ning kus on teadmata sünnilinnaga laste keskmine pikkus. Selle probleemi lahendamiseks saame probleemsele väljale rakendada GROUPING funktsiooni:



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.



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

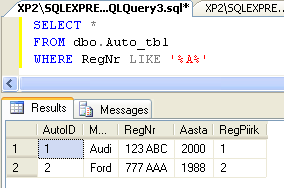
Väljasta iga margi kohta, mitu eksemplari seda on.

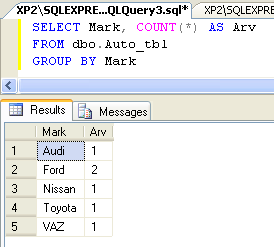
Väljasta iga margi ja väljalaskeaasta komplekti kohta, mitu seda on.

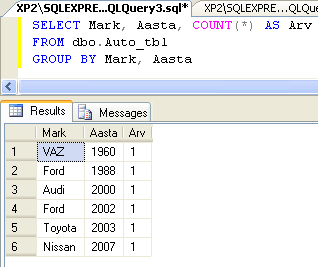
Väljasta iga margi kohta keskmine väljalaskeaasta

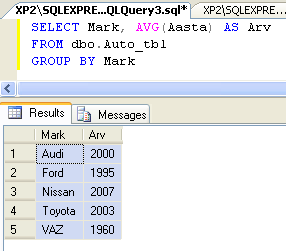
Väljasta iga margi kohta suurima ja vähima väljalaskeaasta vahe.

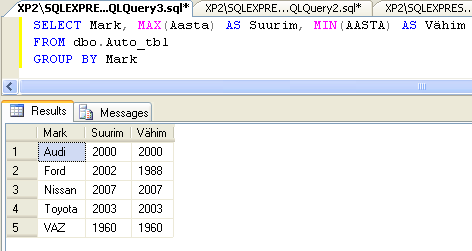
Katseta ROLLUP ja CUBE lisainfo võimalusi eelmiste päringute juures











### 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

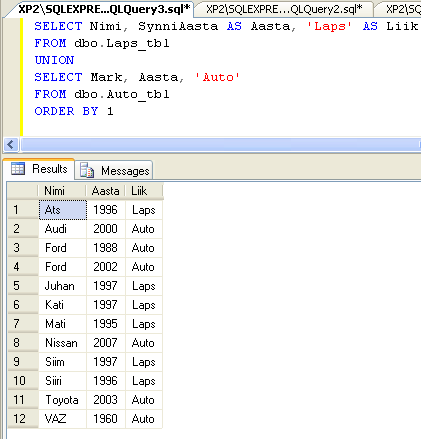
Väljasta kõik autod koos Linna nimetustega, kus nad registreeritud

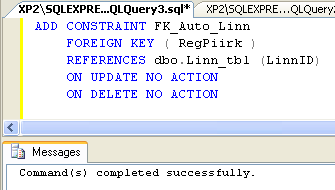
Väljasta autode arv grupeerituna Linnade kaupa

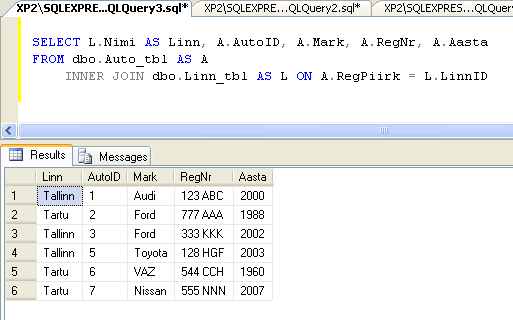
Lisa Linnade tabelisse uus linn

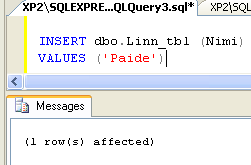
Väljasta LEFT JOINi abil kõik linnad koos neis registreeritud autodega. Registreeritud autodeta maakonnast väljasta vaid nimi.

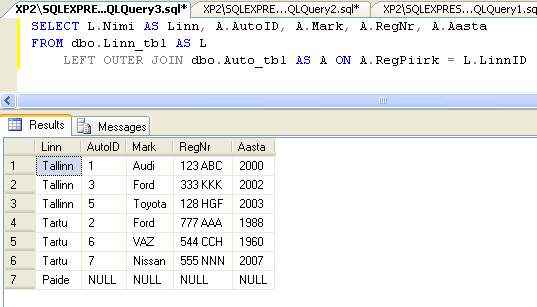
Kingi igale lapsele auto, mille margiks on FORD, väljalaske aasta on lapse sünniaasta ning numbrimärk moodustub lapse pikkus + esimesed 3 tähte lapse nimest.

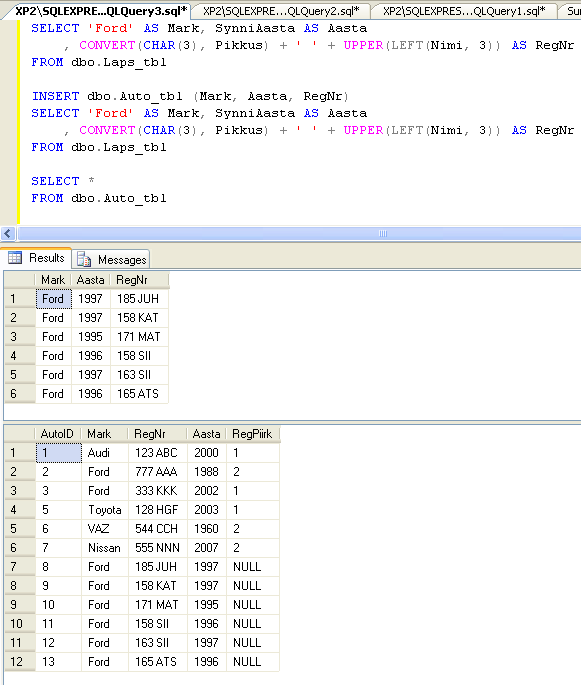












### Alampäringud

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

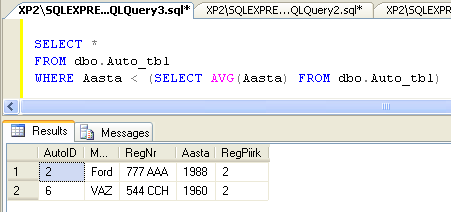
Siim 1997 163

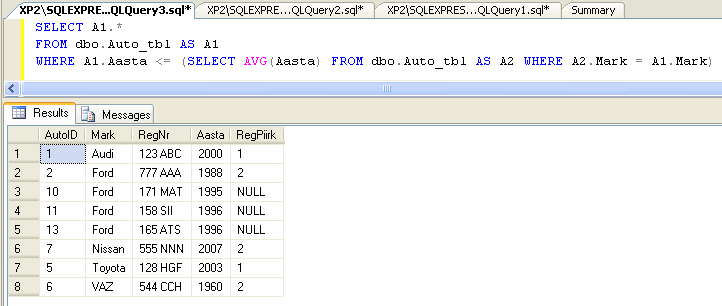
#### Harjutused (Alampäringud)

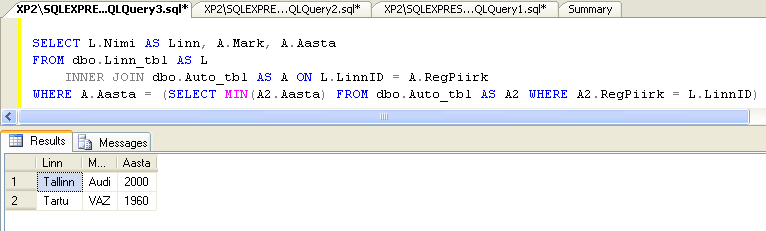
Teata alampäringu abil kõik keskmisest vanemad autod

Teata alampäringu abil markide kaupa kõik selle margi keskmisest vanemad autod

Teata iga linna kohta selles linnas liikuva kõige vanema sõiduki mark/margid



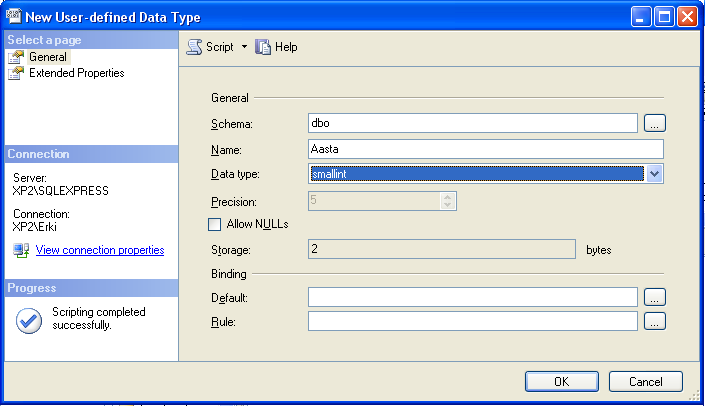




### Lisavõimalused

#### 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:

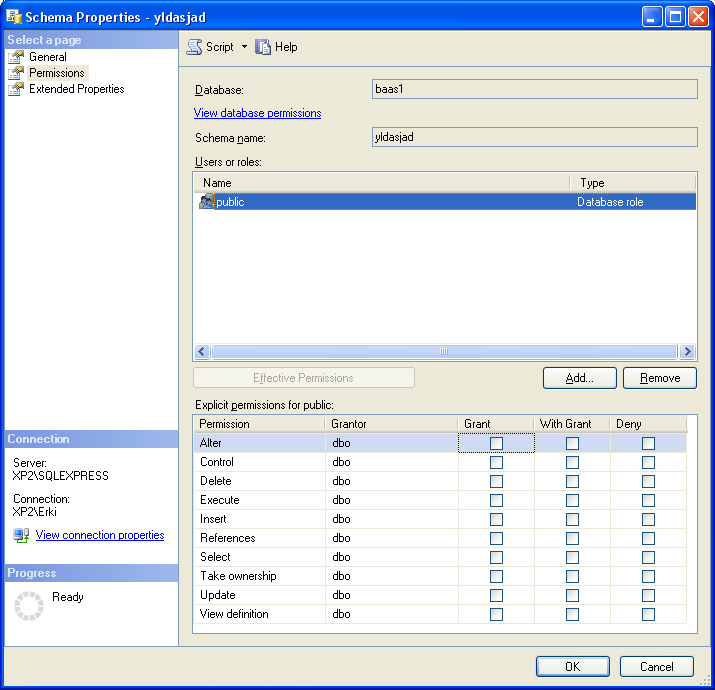
CREATE USER [tegijad]

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)

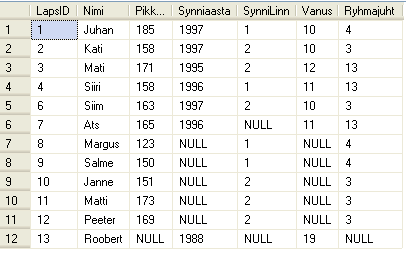
SET @UusLapsID = @@identity

UPDATE dbo.Laps\_tbl

SET Ryhmajuht = @UusLapsID

WHERE Ryhmajuht IS NULL AND LapsID <> @UusLapsID

Kui kõik tehtud on meie tabel järgmine:



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

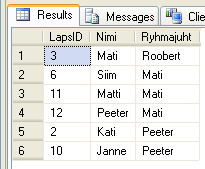
INNER JOIN Alluvad AS A ON L.RyhmaJuht = A.LapsID

)

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



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



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

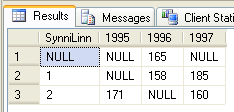
FROM dbo.Laps\_tbl ) AS data

PIVOT (

AVG (Pikkus)

FOR Synniaasta IN ([1995], [1996], [1997])

) AS piv



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

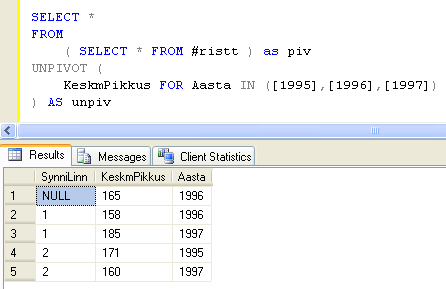
FROM dbo.Laps\_tbl ) AS data

PIVOT (

AVG (Pikkus)

FOR Synniaasta IN ([1995], [1996], [1997])

) AS piv



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)

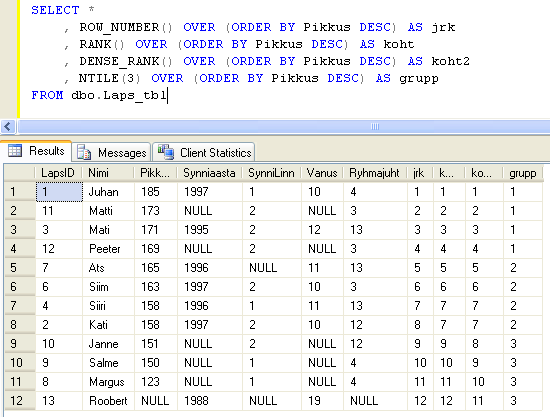
#### Nummerdamised

ROW\_NUMBER() – nummerdab kõik read tulemuses

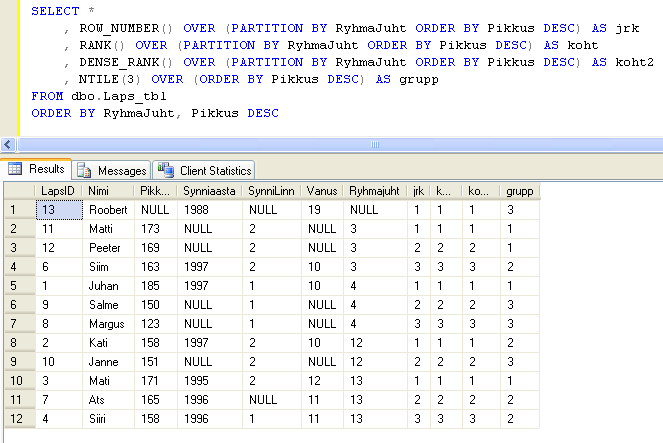
RANK() – ütleb, mitmes on loetelus antud väärtus

DENSE\_RANK() – analoogne RANK funktsiooniga, kuid ei jäta numbreid vahele

NTILE(n) – jagab tulemuse N grupiks



Kõiki nummerdamise funktsioone on võimalik kasutada ka üle gruppide jagades andmed sobiva tunnuse alusel partitsioonideks:



#### 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)

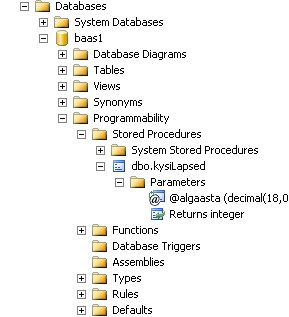
AS

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.

|  |  |
| --- | --- |
| SQL Server 2000 | SQL Server 2005 |
| 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" />

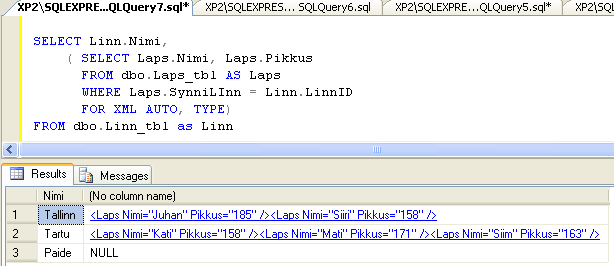
<Laps Nimi="Siim" Pikkus="163" />

</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

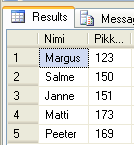
EXEC sp\_xml\_preparedocument @hdoc OUTPUT, @doc

SELECT \*

FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')

WITH (Nimi VARCHAR(40), Pikkus SMALLINT)

EXEC sp\_xml\_removedocument @hdoc



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 \*

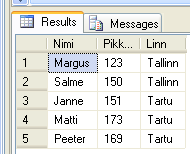
FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')

WITH ( Nimi VARCHAR(40)

, Pikkus SMALLINT

, Linn VARCHAR(20) '../@Nimi')

EXEC sp\_xml\_removedocument @hdoc



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

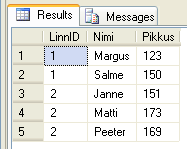
FROM OPENXML(@hdoc, '/lapsed/Linn/Laps')

WITH ( Nimi VARCHAR(40)

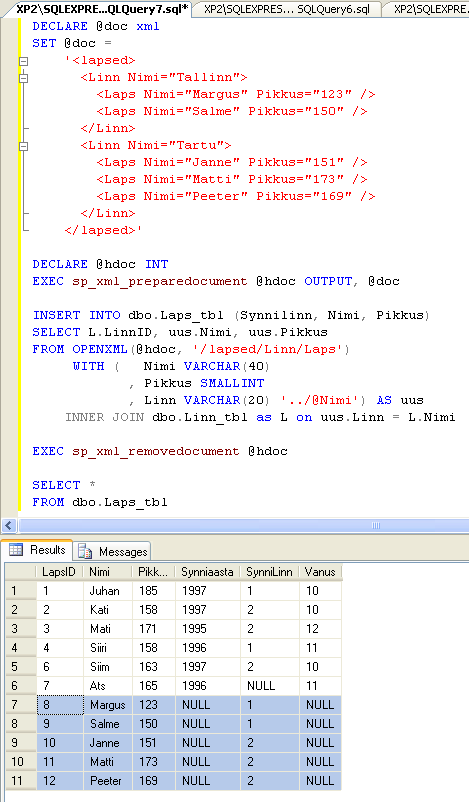
, Pikkus SMALLINT

, Linn VARCHAR(20) '../@Nimi') AS uus

INNER JOIN dbo.Linn\_tbl as L on uus.Linn = L.Nimi



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.

# 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:

SqlDataReader

SqlCommand

SqlConnection

SQL Server

* 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 | DataSet  SqlDataAdapter  SqlConnection  SQL Server |

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 tagaAsta 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.

Andmeallikas

XxxConnection

XxxCommand (XxxParameter)

XxxDataReader/XmlReader

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| Andmeallikas – DataAdapter – DataTable  Fill ->  Update <- | 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

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 <b><i></b></i> , siis XMLis sellist asja olla ei tohi ehk korrektne oleks <b><i></i></b>

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]](#footnote-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).

## 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 | TellitudToode!2! ToodeID | TellitudToode!2! Toode | TellitudToode!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

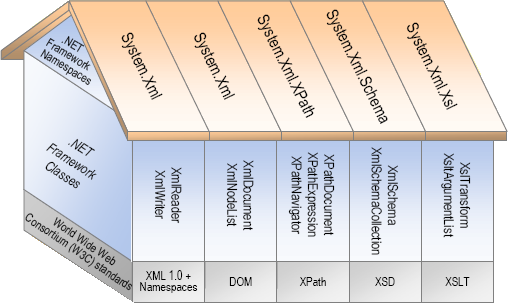
|  |  |
| --- | --- |
| kood | Xmljutt |
| 1 | <toode toode="kala" hind="55.0000" /><toode toode="õun" hind="10.0000" /> |

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");

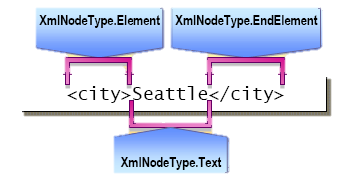
Kui lugeja on olemas, saame XMLi oksa (node) kaupa lugema hakata. Enamasti on seda kasulik teha mingi korduslausega:

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

## 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. Kui aga andmeid on vaja mõne kaugema kohaga vahetada, siis sobib selleks vahepealseks kandjaks XML. Samuti tasub selle vormingu abi peale mõelda juhul, kui andmed juba loomulikuna paiknevad puukujuliselt. Ning kui kumbki vahend veel soovitud tulemuseni ei vii, siis tasub abi otsida .NETi suurest klasside- ja funktsioonide kogumikust, mille kaudu võiks juba päris paljudesse kohtadesse sobivad lahendused kokku panna. Kes püüab kõigest väest, saab üle igast mäest.

1. tühiruum on tühikud, tabulaatorid, Enterid [↑](#footnote-ref-1)