ბ. ღვინეფაძე

# WEB-დაპროგრამება

# WEB 2.0, XML, AJAX

“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

# WEB-დაპროგრამება

# WEB 2.0, XML, AJAX

დამტკიცებულია
სახელმძღვანელოდ სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2013

შაკ 681.3.06

განხილულია ინტერნეტით სარგებლობის სრულყოფის მიზნით შემუშავებული ახალი კონცეფციის WEB 2.0-ის წანამძღვრები, ასევე, ამ მიმართულებით სამუშაოების შესასრულებლად განკუთვნილი თანამედროვე საშუალებები: XML და AJAX.

სახელმძღვანელო დაწერილია ინფორმატიკის ფაკულტეტზე მაგისტრატურის სასწავლო პროგრამაში შემავალი საგნის სილაბუსის მოთხოვნების გათვალისწინებით. მისით სარგებლობა შეუძლიათ შესაბამისი საკითხებით დაინტერესებულ სხვა პირებსაც.

რეცენზენტები: სრ. პროფ. გ. სურგულაძე,
სრ. პროფ. თ. სუხიაშვილი



ISBN 978-9941-20-229-5

http://www.gtu.ge/publishinghouse/

შესავალი

# WEB 2.0

დღეს ინტერნეტი ისე ფართოდ შემოიჭრა ადამიანების ცხოვრებაში, რომ სპეციალისტებმა დააყენეს საკითხი, მისი რესურსებით მომსახურებას მიეცეს რაც შეიძლება მასობრივი, ეფექტიანი და ამავე დროს სტანდარტული (აქ იგულისხმება მასთან ურთიერთობის წესების ადვილად ათვისების უზრუნველყოფა) სახე. ასეთი მიდგომის სახელდებისათვის ბოლო ხანებში სპეციალური ტერმინებიც შემოიღეს – *პროგრამული უზრუნველყოფა, როგორც სერვისი* (Saas - software as a service) და ასეთი პოეტურიც კი – *ღრუბლისებრი გამოთვლები* (cloud computing). სწორედ დროის ასეთი მოთხოვნების გასათვალისწინებლად იქნა ბოლო წლებში შემუშავებული WEB 2.0-ის სახელით ცნობილი ახალი კონცეფცია ინტერნეტისათვის.

ცნება WEB 2.0-ისათვის ნათლია გახლავთ ამერიკელი ტიმ ო'რეილი – აღნიშნული ტერმინი პირველად მის მიერ იქნა გამოყენებული 2005 წლის ოქტომბერში გამოქვეყნებულ სტატიაში.

ო'რეილმა WEB 2.0 ტერმინის არსი ასე განმარტა:

WEB 2.0 არის კომპიუტერული სისტემების დაპროექტების ისეთი მეთოდი, რომელიც ორიენტირებულია ქსელურ ურთიერთქმედებებზე შემდეგი გათვლით – რაც მეტია პროექტის (გამოყენების) მომხმარებელი, მით უფრო ნათლად უნდა გამოიკვეთოს ამ პროექტის უპირატესობა მანამდე არსებულ სისტემებთან შედარებით.

ამრიგად, აქ საუბარია საკითხისადმი გლობალურ – პრინციპული სახის, შეიძლება ითქვას, ფილოსოფიურ მიდგომაზე და არა საიტების დამუშავების პროცესის ეფექტიანობის ასამაღლებლად რაიმე კონკრეტული სახის რეკომენდაციების გაცემაზე.

ახალი კონცეფციის შემუშავების საჭიროება განაპირობა მანამდე არსებული მიდგომის (მას, შესაბამისად, WEB 1.0 კონცეფცია დაარქვეს) კრახმა, რომელზე დაყრდნობითაც უმსხვილესი ინტერნეტ-კომპანიები

ცდილობდნენ მთელი ეს უზარმაზარი ინტერნეტ-სივრცე ცენტრალიზებული, მკაცრად განსაზღვრული წესებითა და ასეთივე შეზღუდვების შემოღებით ემართათ. ამგვარი მიდგომის მარცხი ნათელი გახადა შემდეგი სიტუაციის წარმოშობამ – ინტერნეტ-სივრცის ცენტრალიზებული წესებით მართვის მოსურნე ინტერნეტ-კომპანიებმა ვერ მოახერხეს ბაზრის დაპყრობა, ვერ გაუწიეს კონკურენცია უამრავ მცირე მასშტაბის კომპანიას, რომელთაც ინტერნეტი აავსეს "წესების გარეშე" მოქმედი წვრილ-წვრილი საიტებით (რასაც ხატოვნად შემდგომ "გრძელი კუდის" კონცეფცია ეწოდა).

ჩამოვთვალოთ ის ძირითადი ტენდენციები (რეკომენდაციები), რომელთა ბაზაზე, WEB 2.0 კონცეფციის მიხედვით, უნდა მოხდეს საპროექტო გადაწყვეტილებების მიღება:

- ახალ პროექტთან (დანართთან) მუშაობა არ უნდა მოითხოვდეს მომხმარებლისაგან რაიმე დამატებითი ზომების მიღებას, ანუ როგორც აპარატურის, ისე პროგრამული ნაწილის ამ თუ იმ სახით წინასწარ მომზადებას – მასთან სამუშაოდ სავსებით საკმარისი უნდა იყოს მხოლოდ ბროუზერის შესაძლებლობანი.
- დანართები უნდა უზრუნველყოფდეს შემდეგი სახის WEB-სერვისს – მათ უნდა შეეძლოთ ერთმანეთთან "საერთო ენის" გამონახვა, რაც გულისხმობს ერთმანეთის შესაძლებლობების გამოყენებას.
- კონცეფცია იყენებს ე.წ. Mash – up მიდგომასაც – საჭირო WEB-სერვისის შექმნა შესაძლებელი უნდა იყოს რამდენიმე სხვა WEB-სერვისის ინტეგრაციის შედეგად.
- სურვილის შემთხვევაში შესაძლებელი უნდა იყოს მომხმარებლებს შორის დიალოგის უზრუნველყოფა, ე.წ. სოციალიზაცია – კოლექტიური ინტელექტისათვის ასპარეზის მიცემა. WEB 2.0 კონცეფციის მიხედვით შექმნილი დანართები

ხშირად უბრალოდ შუამავლის როლში გამოდის მომხმარებლებს შორის, რომლებიც თვითონ განსაზღვრავენ მათთვის საინტერესო კონტენტს.

- იზრდება ტეგების როლი. მათი მეშვეობით ხდება კონტენტის ცალკეული უბნების დახარისხება-წვდომა და რელევანტური ინფორმაციის მოსაძიებლად საჭირო გარემოს შექმნა.

ზემოთ მოყვანილი ჩამონათვალი არ გახლავთ მოთხოვნების სრული სია – აქ ყურადღება გამახვილებულია მხოლოდ WEB 2.0 კონცეფციისათვის დამახასიათებელ ძირითად ტენდენციებზე.

აღვნიშნავთ, რომ WEB 2.0 კონცეფცია ინარჩუნებს ყველა იმ საღი აზრის მოთხოვნასაც, რომელთა გათვალისწინება რეკომენდებული იყო WEB-დაპროგრამების "ადრეულ ეპოქებშიც".

მოვიყვანოთ ამ მოთხოვნების სიაც:

- საიტების სტრუქტურის, დიზაინის იმგვარად სრულყოფა, რომ რაც შესაძლებელია მეტად გაადვილდეს უმთავრესი ინფორმაციის მოძიება, მომხმარებელთან ინტერფეისი იყოს მაქსიმალურად მეგობრული, საიტთან ურთიერთობის ფორმატები მომხმარებელს უნდა მოაგონებდეს უკვე ნაცნობ გარემოს, სიტუაციებს, რათა ზედმეტად ორიგინალურმა გადაწყვეტებმა მას მოცემულ საიტთან მუშაობის სურვილი არ გაუქროს.
- საიტი ბროუზერში მაქსიმალურად სწრაფად უნდა იტვირთებოდეს.
- საიტი, უპირველეს ყოვლისა, ოპტიმიზებული უნდა იყოს ძირითადი ამოცანის გადაწყვეტაზე.
- საიტის დიზაინი და შინაარსი ერთმანეთთან ორგანულად უნდა იყოს დაკავშირებული.
- ინფორმაცია უნდა იყოს, რამდენადაც შესაძლებელია, ადვილად აღქმადი და გრამატიკულად გამართული, ყოველგვარი ზედმეტი ელემენტებით გადატვირთვის გარეშე.

შემჩნეულია, რომ საიტსაც “ტანსაცმლის (აქ გაფორმების) მიხედვით ხვდებიან”.

დასასრულ, ვიტყვით, რომ WEB 2.0 კონცეფციის არსს მოკლედ სპეციალისტები ასეც განმარტავენ:

ამ კონცეფციის მიხედვით შექმნილი საიტი შესაძლებლობას უნდა იძლეოდეს, რომ მისი კონტენტი მთლიანად თუ არა, უმეტესწილად მაინც მომხმარებლის სურვილების მიხედვით განისაზღვრებოდეს.

მომდევნო თავებში განხილული იქნება WEB 2.0 კონცეფციის რეალიზებისათვის განკუთვნილი ისეთი თანამედროვე ტექნოლოგიები, როგორებიცაა: AJAX და XML .

# AJAX ტექნოლოგია

AJAX ტექნოლოგია წარმოადგენს შემდგომ ნაბიჯს კლიენტის მხარეზე მოქმედი დინამიკური WEB-გამოყენებების შექმნის სფეროში, რომელთა დამუშავება, დაყენება და შესრულება ხორციელდება უფრო სწრაფად და უკეთესად მის გარეშე შექმნილებთან შედარებით. ამასთან ერთად, უმჯობესდება მომხმარებელთან ინტერაქტიურობის ხარისხიც (ტრადიციული HTML-ფორმებისაგან განსხვავებით).

AJAX ტექნოლოგია ემყარება JavaScript ენისა და HTTP მოთხოვნების სიმბიოზს, ოღონდ JavaScript-ის სცენარების შესრულება ხდება ასინქრონულად, ფონურ რეჟიმში, ამასთან, სერვერიდან ინფორმაციის გადმოსაგზავნად მეტწილად იყენებენ XML ენის შესაძლებლობებს (საერთოდ კი, დასაშვებია ამ მიზნით ნებისმიერი სხვა ფორმატის, მათ შორის ტექსტის, გამოყენებაც).

ზემოთ აღნიშნული თავისებურებებიდან გამომდინარე, AJAX ტექნოლოგიის არსის განსამარტავად ამგვარ “ფორმულასაც” კი იყენებენ:

**AJAX = ასინქრონული JavaScript + XML**

შემდეგ, AJAX ტექნოლოგიის თავისებურებას წარმოადგენს ის გარემოებაც, რომ აქ JavaScript ენას მნიშვნელოვანი როლი ეკისრება ბროუზერს და სერვერს შორის ურთიერთობის დამყარებაში. AJAX–ის მიერ JavaScript გამოიყენება მონაცემების გასაცვლელად ბროუზერსა და Web-სერვერს შორის. სწორედ ამ პროცესში იკვეთება სცენარების ასინქრონულად შესრულების დადებითი მხარე:

HTTP მოთხოვნებზე დაყრდნობით, ფონურ რეჟიმში ბროუზერსა და Web-სერვერს შორის შედარებით მცირე მოცულობის ინფორმაციის გაცვლა მიმდინარეობს WEB-ფურცლის გადატვირთვის გარეშე, რაც მნიშვნელოვნად ამაღლებს პროცესის სისწრაფეს.

ხაზგასასმელია ის გარემოებაც, რომ აღნიშნული ტექნოლოგიის გამოყენებაზე მხოლოდ ბროუზერია პასუხისმგებელი, ანუ WEB-სერვერის მხარეს არ მოითხოვება რაიმე დამატებითი ქმედებების ჩატარება. ამასთან ერთად, ბროუზერების (მხედველობაში გვაქვს თანამედროვე ბროუზერები) პროგრამული უზრუნველყოფისათვისაც საჭირო არ არის ახალი კომპონენტების შემუშავება, რადგან AJAX იყენებს ქვემოთ ჩამოთვლილი ტექნოლოგიების მხოლოდ ღია სტანდარტებს:

JavaScript,

XML,

HTML,

CSS.

ზემოთ ჩამოთვლილი სტანდარტები უცხო ელემენტებს არ წარმოადგენს ყველა ძირითადი თანამედროვე ბროუზერისათვის და კომპიუტერული პლატფორმისათვის.

Web-ტექნოლოგიებმა უკვე დაამტკიცეს, რომ მათი გამოყენება შესაძლებელია არა მხოლოდ ინტერნეტისათვის, არამედ – ტრადიციული, სამაგიდო კომპიუტერებისათვის განკუთვნილი პროგრამული სისტემების შემუშავების დროსაც. თუმცა ასეთ შემთხვევაში, როგორც წესი, მოითხოვება “სუფთა” Web-ტექნოლოგიები გამდიდრდეს დამატებითი

შესაძლებლობებითაც. სწორედ ამ მიზნის მიღწევას ემსახურება AJAX-ტექნოლოგია.

## AJAX-ის გამოყენების მაგალითი

ქვემოთ ვაჩვენოთ AJAX-ის დახმარებით შექმნილი სცენარის მაგალითი (მოითხოვება HTML-ფორმის ტექსტურ ველში გვარის აკრეფვის დაწყებისთანავე გამოყენებამ შემოგვთავაზოს შესაძლო ვარიანტები):

```
<form>
  გვარი:
  <input type="text" id="txt1"
  onkeyup="showHint(this.value)">
</form>
  <p>ეს გვარი? <span id="txtHint"></span></p>
```

ჩანს, რომ "txt1"-თი იდენტიფიცირებულ ტექსტურ ველში პირველი ასოს შეტანისთანავე, ასევე ყოველი მომდევნოს აკრეფვისას (კლავიშის აშვების მომენტში) ხდება showHint() ფუნქციის გამოძახება, რომელიც მიმართავს სერვერს და იქიდან მიღებული მონაცემებით (ამ შემთხვევაში გვარით) განსაზღვრავს span ელემენტის მნიშვნელობას.

აღვნიშნოთ, რომ JavaScript-ენაზე დაწერილი **showHint()** ფუნქცია უნდა განთავსდეს WEB-ფურცლის HTML-კოდის HEAD უბანში:

```
function showHint(str)
{
if (str.length==0)
{
document.getElementById("txtHint").innerHTML=""
return
}
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
```

```
{
alert ("მოცემულ ბროუზერს არ შეუძლია HTTP-მოთხოვნების
      შესრულება")
return
}
var url="gethint.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}
```

განვიხილოთ, თუ რა ოპერაციებს ახორციელებს ზემოთ მოყვანილი showHint() ფუნქცია:

- უწინარეს ყოვლისა, ხდება შემდეგი პირობის შემოწმება (str.length > 0) – შეტანილია თუ არა ტექსტურ ველში რაიმე ინფორმაცია;
- თუ ეს პირობა შესრულებულია, ფუნქცია ცდილობს, შექმნას xmlHttp სახელწოდების მქონე, სერვერთან xml ენაზე ურთიერთობისათვის განკუთვნილი ობიექტის ეგზემპლარი (თუკი ბროუზერი ამის საშუალებას იძლევა);
- თუ პასუხი უარყოფითია, დისპლეიზე ვკითხულობთ შესაბამის შეტყობინებას, დადებითი პასუხის შემთხვევაში კი განისაზღვრება სერვერზე გასაგზავნი ფაილის მისამართი და სახელი (URL);
- აღნიშნულ URL-ს ემატება q პარამეტრი, რომლის მნიშვნელობაც განისაზღვრება ტექსტური ველის ახალი შემცველობით;

- იმ მიზნით, რომ შეტყობინების მიღებისას სერვერმა გადაგზავნილი ფაილის ძებნა არ განახორციელოს კეშ-მეხსიერებაში, URL-ს დაემატება შემთხვევითი რიცხვების გენერატორის მიერ გენერირებული რაიმე რიცხვი;
- xmlHttp ობიექტის ეგზემპლარის რაიმე ცვლილებისას ხდება stateChanged ფუნქციის გამოძახება (იხ. ქვემოთ);
- xmlHttp მზადდება სერვერზე გადაგზავნისათვის (მას გაღებისას გადაეცემა ზემოთ ფორმირებული URL პარამეტრი);
- სერვერზე იგზავნება შესაბამისი შეტყობინება.

აქვე შევნიშნოთ, რომ თუ Enter-ზე ხელის დაჭერისას შეტანის ველი ცარიელი აღმოჩნდება, ფუნქცია მასში განათავსებს txtHint ტექსტს.

როგორც ზემოთ აღვნიშნეთ, stateChanged ფუნქციის შესრულებაზე გაშვება ხდება xmlHttp ობიექტის ეგზემპლარის მდგომარეობის ყოველი ცვლილებისას. იმ შემთხვევაში, როცა ეს მდგომარეობა მოინიშნება კოდით “4" ან “complete”-თი, txtHint ველში ჩაიწერება სერვერიდან გადმოგზავნილი პასუხი.

*განვიხილოთ* ***stateChanged()*** *ფუნქციის დანიშნულება. მისი კოდი გახლავთ:*

```
function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}
```

ხაზი უნდა გაესვას შემდეგ გარემოებას:

**AJAX-გამოყენებები სრულდება მხოლოდ ისეთ ბროუზერებში, რომლებშიც ჩადებულია XML-ტექნოლოგიის სრული მხარდაჭერის შესაძლებლობა.**

ზემოთ მოყვანილი მაგალითში ვხედავთ, რომ ხდება GetXmlHttpObject ფუნქციის გამოძახება და მოცემული ობიექტის შესატყვისი ეგზემპლარის შექმნა. აღნიშნული ფუნქციას აქვს შემდეგი სახე:

```
function GetXmlHttpObject(handler)
{
 var objXMLHttp=null
 if (window.XMLHttpRequest)
 {
  objXMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
  objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
 return objXMLHttp
}
```

მოვიყვანოთ მწყობრში განხილული მასალა და ქვემოთ წარმოვადგინოთ ამ მაგალითის მთლიანი კოდი.

საწყისი HTML ფაილი შეიცავს მარტივ HTML ფორმას და დაყრდნობას HTML ფაილზე:

```
<html>
<head>
<script src="clienthint.js"></script>
</head>
<body>
<form>
სახელი:
<input type="text" id="txt1"
onkeyup="showHint(this.value)">
</form>
```

```
<p>ამ სახელს ირჩევთ: <span id="txtHint"></span></p>
</body>
</html>
```

ზედა ფაილიდან გამოძახებული clienthint.js ფაილის კოდია:

```
var xmlHttp
function showHint(str)
{
if (str.length==0)
{
document.getElementById("txtHint").innerHTML=""
return
}
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
{
alert ("HTTP მოთხოვნების შესრულებას ეს ბროუზერი ვერ
ახერხებს")
return
}
var url="gethint.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
```

```
document.getElementById("txtHint").innerHTML=xmlHttp.responseText
}
}

function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
{
objXMLHttp=new XMLHttpRequest()
}
else if (window.ActiveXObject)
{
objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
}
return objXMLHttp
}
```

## AJAX-ის სერვერული ფურცლები ASP-ისა და PHP-ისთვის

ამთავითვე შევნიშნოთ, AJAX-ის გამოყენებისათვის სპეციალური სერვერის არსებობა საჭირო არ გახლავთ. AJAX-ტექნოლოგიაზე ორიენტირებული WEB-ფურცლების დამუშავება შეუძლია ინტერნეტში მომუშავე ნებისმიერ სერვერს, მაგალითად, IIS-ს. ზემო მაგალითში ამ სერვერის მეშვეობით ხდება JavaScript-სცენარის მიერ გამოძახებული მარტივი სახის მქონე gethint.asp დასახელების ფაილის კოდის დამუშავება და კლიენტის მოთხოვნის შესრულება – მასივიდან ამორჩეული შესაბამისი სახელების მისთვის დაბრუნება.

gethint.asp ფაილისათვის კოდი დაწერილია VBScript-ზე და აქვს სახე:

```
<%
dim a(30)
'შევავსოთ სახელების მასივი:
a(1)="Anna"
a(2)="Brittany"
a(3)="Cinderella"
a(4)="Diana"
a(5)="Eva"
a(6)="Fiona"
a(7)="Gunda"
a(8)="Hege"
a(9)="Inga"
a(10)="Johanna"
a(11)="Kitty"
a(12)="Linda"
a(13)="Nina"
a(14)="Ophelia"
a(15)="Petunia"
a(16)="Amanda"
a(17)="Raquel"
a(18)="Cindy"
a(19)="Doris"
a(20)="Eve"
a(21)="Evita"
a(22)="Sunniva"
a(23)="Tove"
a(24)="Unni"
a(25)="Violet"
a(26)="Liza"
a(27)="Elizabeth"
a(28)="Ellen"
```

```
a(29)="Wenche"
a(30)="Vicky"
'URL-იდან გამოვაცალკევოთ q პარამეტრი:
q=ucase(request.querystring("q"))
'თუ მასივის სიგრძე q>0,  გავეცნოთ რეკომენდაციებს:
if len(q)>0 then
  hint=""
  for i=1 to 30
    if q=ucase(mid(a(i),1,len(q))) then
      if hint="" then
        hint=a(i)
      else
        hint=hint & " , " & a(i)
      end if
    end if
  next
end if
'გამოვიტანოთ რეკომენდაციებს დათვალიერების შედეგი:
if hint="" then
  response.write("ძიება უშედეგოდ დამთავრდა")
else
  response.write(hint)
end if
%>
```

ამჯერად კი განვიხილოთ იმავე ამოცანის გადაწყვეტის მაგალითი PHP-ტექნოლოგიაზე დაყრდნობით.

პირველ ყოვლისა, აღვნიშნოთ, რომ აუცილებელია "clienthint.js" ფაილში URL ცვლადის მნიშვნელობა "gethint.asp" შევცვალოთ "gethint.php"-ით.

კოდს ექნება შემდეგი სახე:

```
<?php
```

// შევავსოთ სახელების მასივი:

```
$a[]="Anna";
$a[]="Brittany";
$a[]="Cinderella";
$a[]="Diana";
$a[]="Eva";
$a[]="Fiona";
$a[]="Gunda";
$a[]="Hege";
$a[]="Inga";
$a[]="Johanna";
$a[]="Kitty";
$a[]="Linda";
$a[]="Nina";
$a[]="Ophelia";
$a[]="Petunia";
$a[]="Amanda";
$a[]="Raquel";
$a[]="Cindy";
$a[]="Doris";
$a[]="Eve";
$a[]="Evita";
$a[]="Sunniva";
$a[]="Tove";
$a[]="Unni";
$a[]="Violet";
$a[]="Liza";
$a[]="Elizabeth";
$a[]="Ellen";
$a[]="Wenche";
$a[]="Vicky";
```

```
// URL-იდან გამოვაცალკევოთ q პარამეტრი:
$q=$_GET["q"];
// თუ მასივის სიგრძე q>0,  გავეცნოთ რეკომენდაციებს:
if (strlen($q) > 0)
{
  $hint="";
  for($i=0; $i<count($a); $i++)
  {
  if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
    {
    if ($hint=="")
      {
      $hint=$a[$i];
      }
    else
      {
      $hint=$hint." , ".$a[$i];
      }
    }
  }
}
if ($hint == "")
{
$response="ძიება უშედეგოდ დამთავრდა";
}
else
{
$response=$hint;   // შესაბამისი მნიშვნელობების მინიჭება
}
echo $response;  // პასუხის გამოტანა
?>
```

## AJAX-ის მონაცემთა ბაზასთან დაკავშირების მაგალითი

დავუშვათ მოითხოვება, WEB-ფურცელზე გამოტანილი იქნეს მონაცემთა ბაზაში განთავსებული კლიენტების სია, ხოლო რომელიმე მათგანის არჩევისას – ამ კლიენტის შესახებ ბაზაში არსებული ინფორმაცია.

ქვემოთ მოყვანილი, ამოცანის შესატყვისი HTML-კოდი შეიცავს მარტივ HTML-ფორმას და JavaScript-ის სცენარზე დაყრდნობას:

```
<html>
<head>
  <script src="selectcustomer.js"></script>
</head>
<body>
  <form>
  აირჩიეთ შემკვეთი:
  <select name="customers" onchange="showCustomer(this.value)">
    <option value="ALFKI">Alfreds Futterkiste
    <option value="NORTS ">North/South
    <option value="WOLZA">Wolski Zajazd
  </select>
 </form>
 <p>
   <div id="txtHint"><b>აქ გამოვა ინფორმაცია შემკვეთის შესახებ.</b></div>
 </p>
</body>
</html>
```

ჩანს, რომ “customers” ჩამოშლად სიაში მონაცემების (შემკვეთის) ყოველი არჩევისას, ანუ "onchange" ხდომილებისას, გამოიძახება "showCustomer()" ფუნქცია, რომლის შესრულების შედეგად "txtHint"

სახელის მქონე div ელემენტი შეივსება WEB-სერვერიდან გადმოგზავნილი ინფორმაციით.

რაც შეეხება JavaScript-ის სცენარს, იგი ინახება selectcustomer.js ფაილში და ასეთი შემცველობისაა:

```
var xmlHttp
function showCustomer(str)
{
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
 {
  alert ("HTTP მოთხოვნების შესრულებას ეს ბროუზერი ვერ ახერხებს")
  return
 }
var url="getcustomer.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
 var objXMLHttp=null
```

```
 if (window.XMLHttpRequest)
 {
 objXMLHttp=new XMLHttpRequest()
 }
 else if (window.ActiveXObject)
 {
 objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
return objXMLHttp
}
```

ზემოთ მოყვანილი კოდიდან ჩანს, რომ JavaScript-ის სცენარი, თავის მხრივ, გამოიძახებს სერვერზე განთავსებულ getcustomer.asp ფაილს. ეს asp-ფაილი მუშავდება ინტერნეტის საინფორმაციო სერვერის (IIS) მიერ. ქვემოთ მოყვანილ მაგალითში იგი შეიცავს VBScript ენაზე დაწერილ კოდს. აქვე უნდა შევნიშნოთ, რომ შესაძლებელია ამ კოდის გადაწერა PHP და ნებისმიერ სხვა სერვერულ ენაზეც.

მოცემულ კოდში SQL ბრძანებების მეშვეობით ხდება მონაცემთა ბაზიდან ინფორმაციის ამოკრეფვა და მათი ეკრანზე გამოტანა HTML ცხრილის სახით:

```
sql="SELECT * FROM CUSTOMERS WHERE CUSTOMERID="
sql=sql & request.querystring("q")

set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
rs.Open sql, conn

response.write("<table>")
do until rs.EOF
for each x in rs.Fields
```

```
response.write("<tr><td><b>" & x.name & "</b></td>")
response.write("<td>" & x.value & "</td></tr>")
next
rs.MoveNext
loop

response.write("</table>")
```

## AJAX-ის მეშვეობით XML ფაილიდან მონაცემების ამორჩევის მაგალითი

ქვემოთ მოყვანილ მაგალითში WEB-ფურცელზე გამოდის მონაცემთა ბაზაში განთავსებული მუსიკოს-შემსრულებელთა სია. რომელიმე მათგანის არჩევისას ეკრანზე გამოიტანება ინფორმაცია მისი ნაწარმოებების შემცველი კომპაქტ-დისკოს შესახებაც:

```
<html>
<head>
  <script src="selectcd.js"></script>
</head>
<body>
<form>
აირჩიეთ კომპაქტ-დისკო:
<select name="cds" onchange="showCD(this.value)">
  <option value="Bob Dylan">Bob Dylan</option>
  <option value="Bonnie Tyler">Bonnie Tyler</option>
  <option value="Dolly Parton">Dolly Parton</option>
</select>
</form>
<p>
<div id="txtHint"><b>აქ გამოდის ინფორმაცია კომპაქტ-დისკოს
შესახებ.</b></div>
</p>
```

```
</body>
</html>
```

"cds" სიაში ელემენტის არჩევისას ანუ "onchange" ხდომილობისას გამოიძახება "showCD" ფუნქცია. ამ ფუნქციის კოდი მოთავსებულია "selectcd.js" ფაილში:

```
var xmlHttp

function showCD(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
{
alert ("HTTP მოთხოვნების შესრულებას ბროუზერი ვერ ახერხებს")
return
}
var url="getcd.asp"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
document.getElementById("txtHint").innerHTML=xmlHttp.responseText
}
}

function GetXmlHttpObject()
{
```

```
var objXMLHttp=null
if (window.XMLHttpRequest)
{
objXMLHttp=new XMLHttpRequest()
}
else if (window.ActiveXObject)
{
objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
}
return objXMLHttp
}
```

ზემოთ მოყვანილი კოდიდან ჩანს, რომ JavaScript-ის სცენარი, თავის მხრივ, გამოიძახებს სერვერზე განთავსებულ getcd.asp ფაილს. ეს asp-ფაილი მუშავდება ინტერნეტის საინფორმაციო სერვერის (IIS) მიერ. ქვემოთ მოყვანილ მაგალითში იგი შეიცავს VBScript ენაზე დაწერილ კოდს. შევნიშნოთ, რომ შესაძლებელია ამ კოდის გადაწერა PHP და ნებისმიერ სხვა სერვერულ ენაზეც.

მოცემული კოდით ხდება XML ფაილის დამუშავება და შედეგების HTML კოდის სახით ეკრანზე გამოტანა:

```
q=request.querystring("q")

set xmlDoc=Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load(Server.MapPath("cd_catalog.xml"))

set nodes=xmlDoc.selectNodes("CATALOG/CD[ARTIST='" & q & "']")

for each x in nodes
for each y in x.childnodes
response.write("<b>" & y.nodename & ":</b> ")
response.write(y.text)
response.write("<br />")
```

```
next
next
```

## AJAX-ისთვის XMLHttpRequest ობიექტის გამოყენება

AJAX ტექნოლოგიით გამოყენების საქმეში უმნიშვნელოვანეს როლს ასრულებს JavaScript ენაში არსებული XMLHttpRequest ობიექტი, მით უფრო, როცა საუბარია WEB 2.0 მიდგომაზე.

გავეცნოთ ამ ობიექტს, მის თვისებებსა და მეთოდებს.

ამთავითვე შევნიშნოთ, რომ სხვადასხვა ბროუზერი XMLHttpRequest ობიექტის შესაქმნელად განსხვავებულ გზებს მიმართავს:

კერძოდ, Internet Explorer ბროუზერი მიზნის მისაღწევად იყენებს ActiveXObject საშუალებას, დანარჩენები ბროუზერები კი სარგებლობენ JavaScript ენის არსენალში მყოფი XMLHttpRequest ობიექტით.

შესაბამისად, ობიექტის შექმნისას საჭირო ხდება კოდში ამ პრობლემის გათვალისწინება-გადაჭრა, რაც შემდეგნაირად ხდება:

```
var XMLHttp=null


if (window.XMLHttpRequest)
 {
 XMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
 XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
```

ამრიგად, ამ ობიექტის გამოყენების მიზნით, თავდაპირველად იქმნება XMLHttp ცვლადი, რომელსაც ეძლევა null მნიშვნელობა. შემდეგ მოწმდება, შესაძლებელია თუ არა ბროუზერის მოცემული ვერსიისათვის

window.XMLHttpRequest ობიექტით სარგებლობა (ასეთი რამ დასაშვებია ისეთი თანამედროვე ბროუზერებისათვის, როგორებიცაა: Firefox, Mozilla და Opera). თუ ეს ასეა, მაშინ იქმნება ამ ობიექტის ეგზემპლარი, ხოლო საწინააღმდეგო შემთხვევაში მოწმდება მოცემული ბროზერისათვის window.ActiveXObject ობიექტის გამოყენების შესაძლებლობა. პირობის დაკმაყოფილებისას (რაც ხდება Internet Explorer 5.5 და უფრო მაღალი დონის ვერსიებისათვის) ფორმირდება შესაბამისი ობიექტის ეგზემპლარი:

```
XMLHttp=new ActiveXObject().
```

მოვიყვანოთ სხვა მაგალითიც, რომელშიც გამოყენებული იქნება XMLHttpRequest ობიექტის უფრო ახალი, სწრაფქმედი ვერსია. მაგრამ თუ ბროუზერს ამ სიახლით ("Msxml2.XMLHTTP" ობიექტით) სარგებლობა არ შეუძლია, მაშინ მიმართვა ხდება Microsoft.XMLHTTP ობიექტისადმი:

```
var XMLHttp=null
try
{
XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")
}
catch(e)
 {
 try
 {
 XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
}
if (XMLHttp==null)
{
XMLHttp=new XMLHttpRequest()
}
```

ზემოთ მოყვანილ მაგალითშიც თავდაპირველად ფორმირდება null მნიშვნელობის მქონე XMLHttp ცვლადი. შემდეგ Internet Explorer 6-სა და მომდევნო ვერსიებისათვის ხორციელდება ობიექტის ეგზემპლარის შექმნის მცდელობა:

```
XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")
```

თუ ეს ქმედება შეცდომას იწვევს, მაშინ გამოიყენება ჩვენ მიერ ადრე განხილული მიდგომა, გათვალისწინებული მოძველებული ვერსიის Internet Explorer 5.5 ბროუზერისათვის:

```
XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
```

შემდეგ, თუ XMLHttp-ის მნიშვნელობა კვლავ null გახლავთ, პროგრამა ცდილობს ობიექტი შექმნას სტანდარტული გზით:

```
XMLHttp=new XMLHttpRequest()
```

მოკლედ XMLHttpRequest-ის სხვა მეთოდების შესახებაც:

*open()* მეთოდით ფორმირდება Web-ისადმი მოთხოვნა,

*send()* მეთოდით ხდება სერვერისათვის მოთხოვნის გაგზავნა,

*abort()* მეთოდით კი – ამ მოთხოვნის გაუქმება.

### XMLHttpRequest ობიექტის თვისებები:

XMLHttpRequest ობიექტის თვისებებიდან გამოვარჩევთ ორ მათგანს:

პირველი გახლავთ **readyState თვისება**. იგი განსაზღვრავს XMLHttpRequest ობიექტის მიმდინარე მდგომარეობას.

ქვემოთ მოყვანილია readyState თვისების შესაძლო მნიშვნელობები:

| მდგომარეობა | აღწერა |
|---|---|
| 0 | მოთხოვნა არ არის ინიციალიზებული |
| 1 | მოთხოვნა ფორმირებულია |
| 2 | მოთხოვნა გაიგზავნა |

| | |
|---|---|
| 3 | მოთხოვნა მუშავდება |
| 4 | მოთხოვნა შესრულდა |

- readyState=0 – ცვლადს აღნიშნული მნიშვნელობა მიენიჭება XMLHttpRequest ობიექტის შექმნის შემდეგ. ამ მნიშვნელობას იგი ინარჩუნებს open() მეთოდის გამოძახებამდე;
- readyState=1 – ამ მნიშვნელობას ცვლადი იღებს open() მეთოდის გამოძახების შემდეგ;
- readyState=2 – მნიშვნელობა თვისებას ეძლევა ხდება send() მეთოდის გამოძახების შემდეგ;
- readyState=3 – ბროუზერი სერვერს დაუკავშირდა, მაგრამ ჯერ სერვერიდან პასუხის მიღების პროცესი არ დასრულებულა;
- readyState=4 – დასრულდა სერვერიდან პასუხის მიღება.

სხვადასხვა ბროუზერი სხვადასხვაგვარად რეაგირებს აღწერილ სიტუაციებზე. მაგალითად, ზოგი მათგანი არ იტყობინება “0” და “1” მდგომარეობების შესახებ.

აქვე უნდა აღინიშნოს, რომ AJAX ინტერესდება მხოლოდ ბოლო მდგომარეობით, ანუ სიტუაციით, როდესაც სერვერიდან მონაცემების გადმოგზავნა დამთავრდა და უკვე შესაძლებელია მათი დანიშნულებისამებრ გამოყენება.

რაც შეეხება XMLHttpRequest ობიექტის სხვა, responseText თვისებას, მისი დანიშნულება არის სერვერიდან გადმოგზავნილი ტექსტის შენახვა.

# XML

## შესავალი

თავიდან იყო SGML (Standardized General Markup Language) – სტანდარტიზებული განზოგადებული ენა, საფუძველი მონიშვნათა ისეთი ენებისთვის, როგორიცაა, მაგალითად, HTML (Hypertext Markup Language), ანუ ჰიპერტექსტის მონიშვნის (გაწყობის, დაფორმატების) ენა.

HTML-ენაზე დაწერილ კოდს აანალიზებს ბროუზერი და ეკრანზე გამოაქვს შესაბამისი დაფორმატების მქონე ჰიპერტექსტი. დაფორმატების სახე კი მოცემული ტექსტური ფრაგმენტისთვის განისაზღვრება მისი გარემომცველი ტეგებით.

XML (Extensible Markup Language) განიმარტება, როგორც მონიშვნის გაფართოებული, ანუ მეტი შესაძლებლობების მქონე ენა. ეს ენაც SGML-ზეა დაფუძნებული და მონაცემების კოდირებას ასევე ტეგების დახმარებით ახდენს, მაგრამ, HTML-ისაგან განსხვავებით, XML-ში ტეგები ასახავენ არა მონაცემების დაფორმატების წესებს, არამედ მათ სტრუქტურას.

მოვიყვანოთ XML-ენაზე XML-დოკუმენტის შექმნის მაგალითი (შინაარსობრივად ეს დოკუმენტი წარმოადგენს შეკვეთის ბლანკს):

```
<Order OrderNo=”1234”>
   <OrderDate>2001-01-01</OrderDate>
   <Customer>Graeme Malcome</Customer>
   <Item>
        <ProductID>1</ ProductID>
        <Quantity>2</ Quantity>
   </Item>
   <Item>
        <ProductID>4</ ProductID>
        <Quantity>1</ Quantity>
    </Item>
</Order>
```

ვხედავთ, რომ დოკუმენტი არ შეიცავს დაფორმატების ინსტრუქციებს. მონაცემები მოთავსებულია, ფაქტობრივად, ნებისმიერი სახით შერჩეულ ტეგებს შორის. შესაბამისად, XML-ანალიზატორს (HTML-ანალიზატორისაგან განსხვავებით) არ სჭირდება, ერკვეოდეს თითოეული ტეგის რაობაში. სწორედ ამ თავისებურების გამო XML-ის დასახელებაში ფიგურირებს სიტყვა extensible (გაფართოებული).

ზემოთ მოყვანილი XML-დოკუმენტის მაგალითზე განვიხილოთ, თუ როგორ შეიძლება განისაზღვროს დოკუმენტის სტრუქტურა. ისევე, როგორც HTML-ენაში, აქაც გამოიყენება ელემენტები და ატრიბუტები. ამასთან, ელემენტები, თავის მხრივ, შეიძლება შეიცავდნენ სხვა ელემენტებსაც. კერძოდ, მოცემულ შემთხვევაში XML-ფრაგმენტი წარმოადგენს იერარქიული სტრუქტურის Order-ელემენტს 1234 მნიშვნელობის მქონე OrderNo-ატრიბუტითა და შემდეგი შიგთავსით:

- OrderDate ჩადგმული ელემენტი (მისი მნიშვნელობაა 2001-01-01);
- Customer ჩადგმული ელემენტი (Graeme Malcolm მნიშვნელობით);
- ორი ჩადგმული ელემენტი Item, რომლებიც, თავის მხრივ, შეიცავენ ProductID და Quantity ელემენტებს (შესაბამისი მნიშვნელობებით).

*შენიშვნა: XML-ში ატრიბუტების მნიშვნელობა თავსდება ბრჭყალებში (ორმაგი, ცალმაგი). ამრიგად, დასაშვებია ვარიანტები:*

<Order OrderNo=”5555”>

<Order OrderNo=’5555’>

*შენიშვნა: HTML-ისგან განსხვავებით, საწყის ტეგს აუცილებლად უნდა მოსდევდეს საბოლოო ტეგი, ხოლო თუ მათ შორის მნიშვნელობა არ გვხვდება (ანუ საქმე გვაქვს ცარიელ ელემენტთან), დასაშვებია (სასურველიცაა) ელემენტის გაფორმების კომპაქტური ვარიანტის გამოყენება. მაგალითად, ვწერთ*

*<MiddleInitial/> კონსტრუქციას*

*ნაცვლად ჩვენთვის კარგად ნაცნობი დაფორმატების ხერხისა:*

*< MiddleInitial > </ MiddleInitial >*

## მოთხოვნები XML-დოკუმენტისადმი

XML-დოკუმენტები შედგენილი უნდა იქნეს **კორექტულად** და **სწორად**.

XML-ის თვალსაზრისით, დოკუმენტი კორექტულია, თუ XML-ანალიზატორს შეუძლია მისი ინტერპრეტირება. ეს კი მაშინ მოხდება, როცა დოკუმენტში ყოველი ელემენტი (თავისი მნიშვნელობითა და ატრიბუტებით) გარკვეული წესების დაცვით იქნება ფორმირებული.

კორექტულობა დოკუმენტის ვარგისიანობისთვის გახლავთ აუცილებელი, მაგრამ არასაკმარისი პირობა. საქმე ისაა, რომ "კორექტული დოკუმენტი" ავტომატურად არ ნიშნავს "სწორ დოკუმენტს" (აქ შეიძლება ანალოგიად მოვიყვანოთ ჩვეულებრივი წინადადებისთვის სინტაქსისა და სემანტიკის ურთიერთმიმართების საკითხი).

ამრიგად, კორექტულობის გარდა, აუცილებელია, დაკმაყოფილდეს დოკუმენტის სისწორის პირობა – იგი უნდა შეიცავდეს მთელ იმ ინფორმაციას, რომელიც მოითხოვება მოცემული კლასის დოკუმენტებისაგან.

მაგალითად, ცხადია, ქორწინების დოკუმენტი მხოლოდ მაშინ იქნება სრულფასოვანი, როცა იგი მხოლოდ ერთი პირის შესახებ არ შეიცავს ინფორმაციას.

XML-დოკუმენტის სისწორე მოწმდება ბროუზერის მიერ დოკუმენტის შიგთავსის შეჯერებით შესაბამისი კლასის დოკუმენტების სპეციფი-კატორთან.

სპეციფიკატორიც დოკუმენტია, ოღონდ DTD ტიპის (Document Type Definition – დოკუმენტის ტიპის განსაზღვრა) ან ე.წ. **სქემის** სახით წარმოდგენილი. დოკუმენტის სისწორის შემოწმების საკითხებს დაწვრილებით ქვემოთ გავეცნობით, ამჯერად კი დავუბრუნდეთ კორექტული დოკუმენტის შექმნის ამოცანას.

შევქმნათ კორექტული XML-დოკუმენტი. ამგვარი სახის დოკუმენტის შედგენა ხდება ქვემოთ ჩამოთვლილი წესების მიხედვით:

- მასში აუცილებლად უნდა ფიგურირებდეს ფესვური ელემენტი (რომელიც, ბუნებრივია, მოიცავს დოკუმენტის ყველა სხვა ელემენტს);
- ელემენტი შემოსაზღვრული უნდა იყოს ორივე ტეგით – გაღება-დახურვის. (ეს მოთხოვნა ზემოთაც აღვნიშნეთ);
- XML-ტეგებში განირჩევა დიდი და პატარა ასოები (HTML-ისგან განსხვავებით);
- ელემენტები აიგება იერარქიული სტრუქტურის მიხედვით.

## ინსტრუქციები, კომენტარები

მონაცემების გარდა, XML-დოკუმენტი შეიძლება შეიცავდეს ინსტრუქციებს XML-ანალიზატორისთვის. მათი მეშვეობით ანალიზატორს ეცნობება XML-სპეციფიკაციის ვერსია, რომელიც იქნა გათვალისწინებული მოცემულ XML-დოკუმენტზე მუშაობისას. ინსტრუქციაშივე შეიძლება მიეთითოს სტილის ცხრილიც (ანუ სტილების მაფორმირებელი ცხრილი) და სხვ.

აღნიშნული ინსტრუქციები სპეციალური სახის ტეგში ჩაისმება:

<? *ინსტრუქცია* ?>

მოვიყვანოთ იმ ინსტრუქციის მაგალითი, რომელსაც ფესვური ელემენტის წინ განათავსებენ:

```
<?xml version=”1.0”?>
<Order OrderNo=”1234”>
    <OrderDate>2001-01-01</OrderDate>
    <Customer>Graeme Malcome</Customer>
    <Item>
        <ProductID>1</ ProductID>
        <Quantity>2</ Quantity>
    </Item>
    <Item>
        <ProductID>4</ ProductID>
```

```
        <Quantity>1</ Quantity>
    </Item>
</Order>
```

ისევე, როგორც დაპროგრამების სხვა ენები, XML-იც იყენებს კომენტარებს პროგრამისტების მიერ თავიანთვის ან სხვა პროგრამისტებისთვის ინფორმაციის მისაწოდებლად. XML-ში ამ მიზნით გამოიყენება შემდეგი კონსტრუქცია:

```
<!- კომენტარი ->
```

გავიხსენოთ, რომ ასეთივე სახის კომენტარს იყენებდა HTML-იც:

```
<?xml version="1.0"?>
    <!- კომენტარი ->
<Order OrderNo="1234">
    <OrderDate>2001-01-01</OrderDate>
    <Customer>Graeme Malcome</Customer>
     <!- კომენტარი ->
    <Item>
        <ProductID>1</ ProductID>
        <Quantity>2</ Quantity>
    </Item>
    <Item>
        <ProductID>4</ ProductID>
        <Quantity>1</ Quantity>
    </Item>
</Order>
```

## სახელების სივრცე

**სახელების სივრცე** განიმარტება, როგორც მოცემულ სიმრავლეში ამა თუ იმ წესით ერთმანეთთან დაკავშირებული (ამასთან, ამ სიმრავლეში უნიკალური) სახელების, ტერმინების და იდენტიფიკატორების ერთობლიობა.

სახელების სივრცის შემოტანა-გამოყენება განპირობებულია შემდეგი გარემოებით:

შესაძლოა, WEB-დოკუმენტში (და, ცხადია, არა მარტო მასში) ფიგურირებდეს ერთნაირი სახელის, მაგრამ განსხვავებული დანიშნულების (შესაბამისად, განსხვავებული ტიპის მონაცემების შემცველი) ელემენტები.

მაგალითად, XML-დოკუმენტში წიგნის მაღაზიაში წიგნებზე მოთხოვნას შეიძლება ასეთი სახე ჰქონდეს:

```
<?xml version="1.0"?>
<BookOrder OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer>
      <Title>Mr.</Title>
      <FirstName>Graeme</FirstName>
      <LastName>Malcolm</LastName>
   </Customer>
   <Book>
      <Title>Treasure Island</Title>
      <Author>Robert Louis Stevenson</Author>
   </Book>
</BookOrder>
```

ვხედავთ, რომ დოკუმენტი შეიცავს სხვადასხვა დანიშნულების მქონე ორ Title ელემენტს. ერთი მათგანია მიმართვა მყიდველისადმი, მეორე – წიგნის სახელწოდება.

გაურკვევლობის თავიდან ასაცილებლად ასეთ შემთხვევაში XML იყენებს ე.წ. **სახელების სივრცეებს,** რომელთა არსი შემდეგია:

სახელების სივრცის მეშვეობით ხდება XML-დოკუმენტის ელემენტების და ატრიბუტების დაკავშირება (მიბმა) რაიმე უნივერსალურ იდენტიფიკატორთან – URI-სთან. აქვე აღვნიშნავთ, რომ Universal Resource Indentifier ანუ **რესურსის უნივერსალური მაჩვენებელი** შეიძლება

წარმოადგენდეს URL-ს, მაგრამ შესაძლოა, იგი იყოს რაიმე სხვა უნიკალური იდენტიფიკატორიც (ე.ი არ წარმოადგენდეს რესურსს ინტერნეტიდან).

სახელების სივრცე შესაძლებელია ნებისმიერ ელემენტში გამოცხადდეს.

ამ მიზნით XML-ში გამოიყენება xmlns ატრიბუტი. ბუნებრივია, რომ ამ ატრიბუტის მნიშვნელობა დუმილით ვრცელდება კვალიფიცირებული ელემენტის შემადგენელ ქვეელემენტებზეც.

წიგნებზე მოთხოვნა შემდეგნაირად დავაზუსტოთ:

```
<?xml version="1.0"?>
<BookOrder OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer xmlns="http://www.northwindtraders.com/customer">
      <Title>Mr.</Title>
      <FirstName>Graeme</FirstName>
      <LastName>Malcolm</LastName>
   </Customer>
   <Book xmlns="http://www.northwindtraders.com/book">
      <Title>Treasure Island</Title>
      <Author>Robert Louis Stevenson</Author>
   </Book>
</BookOrder>
```

ერთნაირი სახელების მქონე ელემენტების პრობლემა შემდეგნაირად გადაწყდა:

Customer და შესაბამისად, მასში ჩადგმული ელემენტები, მაშასადამე, Title ელემენტიც, მიეკუთვნება ინტერნეტის ერთ-ერთ მისამართზე, მაგალითად, http://www.northwindtraders.com/customer-ზე არსებულ სახელების სივრცეს, ხოლო Book ელემენტი და, ცხადია, მასში ჩადგმული ასევე Title სახელის მქონე სხვა ელემენტი

http://www.northwindtraders.com/book მისამართზე ფიქსირებულ სახელების სივრცეს.

ვხედავთ, რომ სახელთა სივრცის ამგვარი წესით გამოცხადება მთლად მოხერხებული არ არის, განსაკუთრებით მაშინ, როცა მას უკავშირდება ბევრი ელემენტი და ატრიბუტი. ასეთ შემთხვევებში მიმართავენ საკითხის ალტერნატიულ გადაწყვეტას – ფესვურ ელემენტში ცხადდება ყველა ასეთი სივრცე. ერთ-ერთი მათგანი დუმილით გაითვალისწინება პრეფიქსით მოუნიშნავი ელემენტებისა და ატრიბუტებისათვის, დანარჩენი სივრცეებისთვის კი გამოცხადდება აბრევიატურები, რომელთა მეშვეობითაც მოინიშნება საჭირო ელემენტები და ატრიბუტები (აბრევიატურა მათი სახელის წინ პრეფიქსის როლში მოგვევლინება).

რადგანაც აბრევიატურა სულ რამდენიმე სიმბოლოსაგან შედგება, კოდის უკეთ და სწრაფად აღსაქმელად მას წაუმძღვარებენ ქვეელემენტებს იმ შემთხვევაშიც კი, როცა XML-ანალიზატორს ასეთი გადაწყვეტის გარეშეც შეუძლია ვითარებაში გარკვევა.

მოვიყვანოთ მაგალითი:

```
<?xml version="1.0"?>
<BookOrder xmlns="http://www.northwindtraders.com/order"
   xmlns:cust="http://www.northwindtraders.com/customer"
   xmlns:book="http://www.northwindtraders.com/book"
      OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <cust:Customer>
      <cust:Title>Mr.</cust:Title>
      <cust:FirstName>Graeme</cust:FirstName>
      <cust:LastName>Malcolm</cust:LastName>
   </cust:Customer>
   <book:Book>
      <book:Title>Treasure Island</book:Title>
      <book:Author>Robert Louis Stevenson</book:Author>
```

```
  </book:Book>
</BookOrder>
```

დოკუმენტი ეყრდნობა სამ სახელთა სივრცეს:

- http://www.nothwindtraders.com/order – დუმილით;
- http://www.nothwindtraders.com/customer – აღნიშნული სივრცისთვის გამოცხადდა cust აბრევიატურა;
- http://www.nothwindtradesr.com/book – ამ სივრცისთვის კი – book აბრევიატურა.

დოკუმენტის ელემენტებისა და ატრიბუტების წინ დასმულია სათანადო პრეფიქსები, ხოლო ელემენტები და ატრიბუტები, რომელთა სახელწოდებებში პრეფიქსი არ ფიგურირებს (მოცემულ მაგალითში ესენია: BookOrder, OrderNo და OrderDate), მიეკუთვნება დუმილით განსაზღვრულ სახელების სივრცეს.

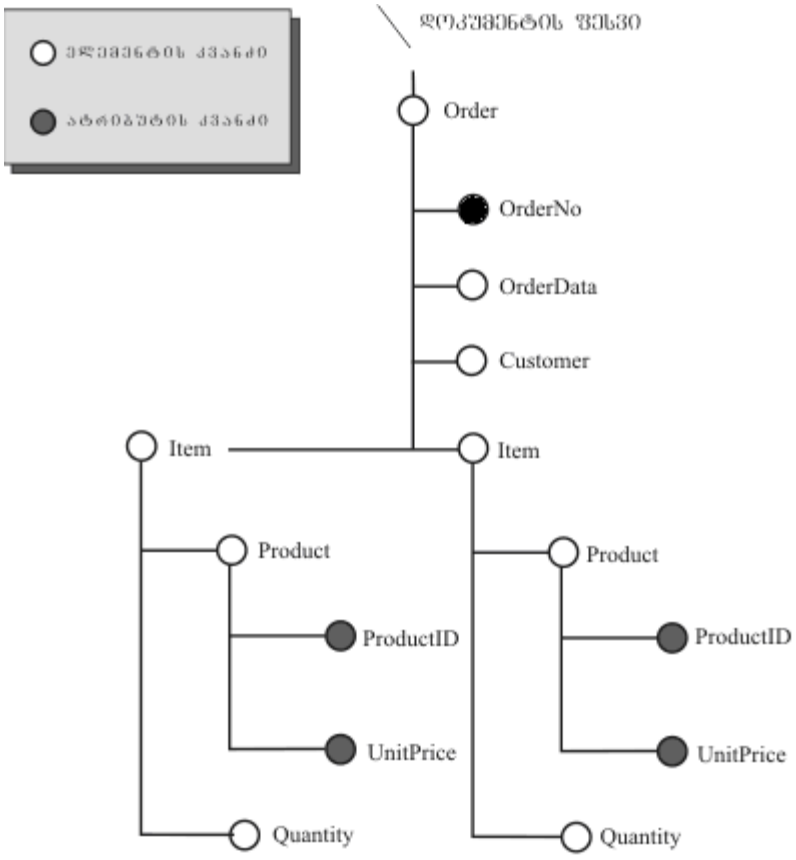
## მოგზაურობა XML-დოკუმენტის სამყაროში

XML-დოკუმენტს ამუშავებს რაიმე პროგრამა (გამოყენება). ცხადია, ინფორმაციის მისაღებად პროგრამას უნდა შეეძლოს დოკუმენტის სხვადასხვა ფრაგმენტის მოძიება, ანუ მოგზაურობა ელემენტებისა და ატრიბუტების სივრცეში, რათა გაიგოს და გამოიყენოს მათი მნიშვნელობები. ამ საქმეში პროგრამას ეხმარებიან ე.წ **Xpath-გამოსახულებები**. თურმე შესაძლებელია დოკუმენტი წარმოდგენილ იქნეს, როგორც კვანძებისგან შემდგარი ხე და მაშინ Xpath-გამოსახულებით შეიძლება დასამუშავებელ კვანძზე მითითება. საკითხის არსში გასარკვევად განვიხილოთ მარტივი XML-დოკუმენტი:

```
<?xml version="1.0"?>
<Order OrderNo="1234">
  <OrderDate>2001-01-01</OrderDate>
  <Customer>Graeme Malcolm</Customer>
  <Item>
```

```
    <Product ProductID="1" UnitPrice="18">Chai</Product>
    <Quantity>2</Quantity>
  </Item>
  <Item>
    <Product ProductID="2" UnitPrice="19">Chang</Product>
    <Quantity>1</Quantity>
  </Item>
</Order>
```

მოცემული დოკუმენტი შეიძლება წარმოვადგინოთ, როგორც კვანძთა ხე:




*ნახ.1.*

## კვანძის ადგილმდებარეობამდე გზის ჩვენება

XML-ში ამ მიზნით შეიძლება გამოყენებული იქნეს ორი სახის სინტაქსი: **შემოკლებული** და **სრული**.

*ჩვენ განვიხილავთ შემოკლებული სახის სინტაქსს.*

ელემენტები და ატრიბუტები დოკუმენტში სხვადასხვა დონეზე არიან განლაგებული. თვით დოკუმენტის დონე (ე.წ. ფესვური დონე) "/" სიმბოლოთი აღინიშნება. თუ საჭიროა ამ დონიდან Order-ელემენტზე (ხის თვალსაზრისით, Order-ელემენტის კვანძზე) გადასვლა, ვიყენებთ Xpath-ის შემდეგ გამოსახულებას:

/order

განვაგრძოთ მოგზაურობა. გადავინაცვლოთ Customer ელემენტისკენ. ამ კვანძის არჩევა მოხდება შესაბამისი Xpath-გამოსახულებით:

/Order/Customer

ამა თუ იმ ელემენტში არსებულ ატრიბუტთან შესაღწევად შემოკლებული სინტაქსი @ სიმბოლოს იყენებს. მოვიყვანოთ მაგალითი:

/Order/@OrderNo

რაც შეეხება ამ თუ იმ კვანძისთვის მოცემული სახელის მქონე ყველა შვილობილ კვანძთან შეღწევას, იგი შემდეგი წესით ხორციელდება:

/Order//Product

ხოლო მოცემული კვანძისთვის ნებისმიერი სახელის მქონე ყველა შვილობილ კვანძთან შესაღწევად ასეთ გამოსახულებას გამოვიყენებთ:

/Order/*

ის რაც ზემოთ განვიხილეთ, გახლდათ ელემენტთან (ატრიბუტთან) შესაღწევად აბსოლუტური გზის მაჩვენებელი Xpath-გამოსახულების სინტაქსი.

ახლა გავეცნოთ კვანძამდე ფარდობითი გზის მაჩვენებელ Xpath-გამოსახულებებს. სინტაქსი ამ შემთხვევაში გულისხმობს, რომ რაიმე კვანძი უკვე არჩეულია, ანუ მიმდინარე გახლავთ. გზა ახალ კვანძამდე Xpath-გამოსახულებაში იგება მიმდინარე კვანძის მდებარეობის დუმილით გათვალისწინების შედეგად.

მაგალითად, თუ დოკუმენტში მიმდინარე კვანძს წარმოადგენს პირველი Item ელემენტი, შემდეგი Xpath-გამოსახულება

Quantity

მიგვიყვანს შესაბამის ელემენტთან, ხოლო Xpath-გამოსახულება

Product/@PtoductID

მიგვიყვანს შესაბამის ატრიბუტთან (მაშასადამე, პროგრამას შეეძლება ამ ატრიბუტის მნიშვნელობის გაგება).

მოცემული კვანძიდან დოკუმენტის დასაწყისში (ანუ ფესვურ კვანძში) ერთი ნახტომით გადასაადგილებლად გამოიყენება ორი წერტილი. აქედან კი შესაძლებელია ქვემოთ დაშვებაც. მოვიყვანოთ მაგალითი:

.. /@OrderNo

ზოგჯერ პროგრამას შეიძლება დასჭირდეს გარკვევა – რომელია მიმდინარე კვანძი? ამ მიზნის მისაღწევად სინტაქსი გვთავაზობს ერთადერთი წერტილის გამოყენებას.

## გზის ჩვენებისას კრიტერიუმების გამოყენება

როგორც ვნახეთ, გზის ჩვენებით შეიძლება გავიდეთ არა მარტო ერთ კვანძზე, არამედ მათ ჯგუფზეც. XML იძლევა ამ ჯგუფიდან რაიმე კრიტერიუმის მიხედვით კვანძების შემდგომი შერჩევის შესაძლებლობასაც. სინტაქსი მოითხოვს კრიტერიუმი-გამოსახულება კვადრატულ ფრჩხილში იქნეს ჩასმული.

შემდეგი Xpath-გამოსახულება გვიბრუნებს ყველა ისეთ Product-ელემენტს, რომლებისთვისაც UnitPrice-ატრიბუტის მნიშვნელობა 18 ერთეულს აჭარბებს:

/Order/Item/Product[@UnitPrice>18]

აქ გზა UnitPrice კვანძამდე დუმილით განისაზღვრება. ცხადია, შესაძლებელია მასში სხვა გზის მითითებაც (როგორც წესი, მიმართავენ ფარდობითი გზის ჩვენების ხერხს):

/Order/Item[Product/@ProductID=1]

ეს Xpath-გამოსახულება აბრუნებს ისეთ Item-ელემენტებს, რომელთა შვილობილი Product-ელემენტებისთვის ProductID-ატრიბუტის მნიშვნელობა ‘1’-ის ტოლია.

## სტილის XSL-ცხრილები

მართალია, XML ენა შესანიშნავ საშუალებას წარმოადგენს პროგრამებს თუ ორგანიზაციებს შორის გასაცვლელი მონაცემების ასაღწერად, მაგრამ ე.წ. ბიზნეს-პროცესების გარკვეულ ეტაპზე საჭირო ხდება ამ მონაცემების გადაყვანა სხვა ფორმატში მათი შემდგომი დამუშავებისა თუ ეკრანზე ასახვისათვის. უპირველეს ყოვლისა, ეს გახლავთ HTML-ფორმატი.

სწორედ, XML-მონაცემების HTML-ფორმატში გადასაყვანად შეიქმნა XSL ენა. შემდეგ კი გაირკვა, რომ შესაძლებელია XML-მონაცემების ნებისმიერ ფორმატში გადაყვანაც და შეიქმნა XML-ის გაუმჯობესებული ვერსია – XSLT (T სიმბოლო ტრანსფორმაციას აღნიშნავს).

### XSL-დოკუმენტები

XSL-დოკუმენტი თვითონაც XML-დოკუმენტს წარმოადგენს, რომელშიც XSL-ბრძანებების საფუძველზე განისაზღვრება **სტილის ცხრილები.** ამ ცხრილებზე დაყრდნობით, XML-ანალიზატორი საწყისი XML-დოკუმენტიდან აფორმირებს გამოსაყვან ტექსტს, ანუ ქმნის საბოლოო დოკუმენტს.

იმ მიზნით, რომ XML-ანალიზატორმა შეძლოს XSL-დოკუმენტის ბრძანებების გაგება, დოკუმენტის ფესვურ ელემენტში აცხადებენ სახელების სივრცეს, ჩვეულებრივ, xsl-პრეფიქსით, თუმცა ხშირად გამოიყენება xslt-პრეფიქსიც. ეს პრეფიქსები ელემენტებს აკავშირებენ შემდეგ **სახელთა სივრცეებთან:**

XSL - http://www.w3.org/TR/WD-xsl

XSLT - http://www.w3.org/1999/XSL/Transform

დავუშვათ, მოცემულია რაიმე XML-დოკუმენტი:

```
<?xml version="1.0"?>
<Order OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer>Graeme Malcolm</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="18">Chai</Product>
      <Quantity>2</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="19">Chang</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

*(აიგება ნახ. №1-ის ბაზაზე)*

შევქმნათ XSL-დოკუმენტი. მის ფესვურ ელემენტად, ჩვეულებრივ, ირჩევენ stylesheet-ს. იგი შეიცავს ერთ ან რამდენიმე template ელემენტს, რომელთა გავლენის არე ვრცელდება საწყის დოკუმენტში შემავალ კონკრეტულ XML-მონაცემებზე.

რადგანაც XSL-დოკუმენტი იგივე XML-დოკუმენტია, ბუნებრივია, მას მოეთხოვება, აკმაყოფილებდეს XML დოკუმენტის კორექტულობისადმი ყველა მოთხოვნას.

მოვიყვანოთ უმარტივესი XSL-დოკუმენტის მაგალითი, რომლის მეშვეობითაც შეიძლება დამუშავდეს ზემოთ მოყვანილი შეკვეთის ამსახველი საწყისი XML-დოკუმენტი:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="/">
      <HTML>
```

```
        <HEAD>
          <TITLE>Northwind Web Page</TITLE>
        </HEAD>
        <BODY>
          <P>Customer Order</P>
        </BODY>
      </HTML>
    </xsl:template>
</xsl:stylesheet>
```

ამ დოკუმენტში ერთადერთი XSLT ტიპის შაბლონია განსაზღვრული XML-დოკუმენტის ფესვისთვის და მასში შემავალი ყველა ელემენტისათვის. თვით შაბლონი მხოლოდ HTML-ტეგებისგან შედგება და, ფაქტობრივად, არავითარ ისეთ ოპერაციას არ ასრულებს, რისთვისაც საერთოდ იყენებენ სტილის XSL-ცხრილებს. ამ შაბლონს საწყისი XML-დოკუმენტიდან ბროუზერის ეკრანზე გამოჰყავს მხოლოდ მის მიერვე წინასწარ ზუსტად განსაზღვრული HTML-დოკუმენტი. საერთოდ კი, XSL-ცხრილით შესაძლებელია საწყისი XML-დოკუმენტიდან მონაცემების ამორჩევა-გარდაქმნა და საბოლოო დოკუმენტში დამატება, რისთვისაც შაბლონში გამოყენებული უნდა იქნეს XSL-ბრძანებები. გავეცნოთ ამ ბრძანებებს.

### Value-of ბრძანება

ამ ბრძანების საშუალებით ხდება XML-კოდიდან კონკრეტული ელემენტებისა და ატრიბუტების მნიშვნელობების ამოკრება და შედეგებში გადაგზავნა. Value-of ბრძანება იყენებს select ატრიბუტს, რომლის მნიშვნელობა წარმოადგენს Xpath- გამოსახულებას საჭირო მონაცემებთან შესაღწევად.

ყოველივე ზემოთქმულის გათვალისწინებით, შესაძლებელი ხდება, შაბლონს დაევალოს XML-ბლოკიდან მონაცემთა გარკვეული ნაწილის ამოკრება:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Web Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
            <P>Order No: <xsl:value-of select="Order/@OrderNo"/></P>
            <P>Date: <xsl:value-of select="Order/OrderDate"/></P>
            <P>Customer: <xsl:value-of select="Order/Customer"/></P>
         </BODY>
      </HTML>
    </xsl:template>
</xsl:stylesheet>
```

მივაქციოთ ყურადღება: Xpath-გამოსახულებაში გზა ფარდობითია - იგი აითვლება შაბლონის match-ატრიბუტის მნიშვნელობიდან. მოცემულ შემთხვევაში ეს მნიშვნელობაა “/”, რის გამოც იგი მისამართში აღარ ფიგურირებს:

მართლაც, გვაქვს select=”order/. . . “

და არა select=”/order/. . . “

თუკი ამ დოკუმენტით დამუშავდება საწყისი XML-დოკუმენტი, მიიღება შემდეგი HTML-კოდი:

```
<HTML>
   <HEAD>
      <TITLE>Northwind Web Page</TITLE>
   </HEAD>
   <BODY>
```

```
    <P>Customer Order</P>
    <P>Order No: 1234</P>
    <P>Date: 2001-01-01</P>
    <P>Customer: Graeme Malcolm</P>
  </BODY>
</HTML>
```

### For-each ბრძანება

XML-დოკუმენტში ერთნაირი ტიპის არსთა ჩამოთვლისათვის ხშირად გამოიყენება ერთსახელა ელემენტები. მაგალითად, ზემოთ განხილული შეკვეთის ბლანკი შეიცავს ორ Item-ელემენტს, რომლებიც მიუთითებენ ორ შეკვეთილ საქონელზე. სწორედ ამგვარი კვანძების ჩასათვალიერებლად შეიძლება გამოვიყენოთ for-each ბრძანება. ცხადია, select-ატრიბუტის გამოყენება ამ შემთხვევაშიც არის საჭირო. მოვიყვანოთ ამ ბრძანების გამოყენების მაგალითი:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Northwind Web Page</TITLE>
      </HEAD>
      <BODY>
        <P>Customer Order</P>
        <P>Order No: <xsl:value-of select="Order/@OrderNo"/></P>
        <P>Date: <xsl:value-of select="Order/OrderDate"/></P>
        <P>Customer: <xsl:value-of select="Order/Customer"/></P>
        <TABLE Border="0">
          <TR>
```

```
            <TD>ProductID</TD>
            <TD>Product Name</TD>
            <TD>Price</TD>
            <TD>Quantity Ordered</TD>
          </TR>
          <xsl:for-each select="Order/Item">
            <TR>
              <TD><xsl:value-of select="Product/@ProductID"/>
              </TD>
              <TD><xsl:value-of select="Product"/></TD>
              <TD><xsl:value-of select="Product/@UnitPrice"/>
              </TD>
              <TD><xsl:value-of select="Quantity"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

როგორც ვხედავთ, წინა დოკუმენტში ჩაემატა ბლოკი, რომელიც უზრუნველყოფს ეკრანზე ცხრილის (Table) ასახვას. ამ ბლოკის დასაწყისში ხისტი სქემის მიხედვით (ე.ი. არ გამოიყენება xml-მონაცემები) ცხრილისთვის ფორმირდება სათაურის სტრიქონი. შემდეგ კი for-each ბრძანება ქმნის მონაცემების შემცველ სტრიქონებს იმავე ცხრილისთვის. თითოეული სტრიქონის ფორმირება ხდება select-ატრიბუტის მნიშვნელობაში მითითებული “Order/Item” ელემენტიდან ამოღებული ინფორმაციის მეშვეობით. მოცემულ შემთხვევაში, ცხადია, სტრიქონისთვის შეირჩევა ცხრილის სათაურის უბნის (ProductID, ProductName, Price,

Quantity) შესატყვისი ინფორმაცია, ანუ ProductID-ატრიბუტის, Product-ელემენტის, UnitPrice-ატრიბუტის და Quantity-ელემენტის მნიშვნელობები.

For-each ბრძანება იმუშავებს იმდენჯერ, რამდენი Item-ელემენტიც იქნება საწყის XML-დოკუმენტში, ანუ მოცემულ შემთხვევაში ორჯერ და, მაშასადამე, ცხრილისთვის შეიქმნება მონაცემთა შემცველი ორი სტრიქონი.

ამ XSL-დოკუმენტით საწყისი XML-დოკუმენტის დამუშავება მოგვცემს შემდეგი სახის საბოლოო XML-დოკუმენტს:

```
<HTML>
  <HEAD>
    <TITLE>Northwind Web Page</TITLE>
  </HEAD>
  <BODY>
    <P>Customer Order</P>
    <P>Order No: 1234</P>
    <P>Date: 2001-01-01</P>
    <P>Customer: Graeme Malcolm</P>
    <TABLE Border="0">
      <TR>
        <TD>ProductID</TD>
        <TD>Product Name</TD>
        <TD>Price</TD>
        <TD>Quantity Ordered</TD>
      </TR>
      <TR>
        <TD>1</TD>
        <TD>Chai</TD>
        <TD>18</TD>
        <TD>2</TD>
      </TR>
```

```
        <TR>
          <TD>2</TD>
          <TD>Chang</TD>
          <TD>19</TD>
          <TD>1</TD>
        </TR>
      </TABLE>
   </BODY>
</HTML>
```

### ატრიბუტების შექმნა **attribute** ბრძანების მეშვეობით

ზემოთ განხილულ მაგალითებში საბოლოო XML-დოკუმენტში მონაცემების ფორმირება ხდებოდა მხოლოდ ელემენტების სახით. არცთუ იშვიათად მოითხოვება, რომ საბოლოო დოკუმენტში ატრიბუტიც შევქმნათ, რომლის მნიშვნელობა აიღება საწყისი XML-დოკუმენტიდან.

მაგალითად, დავუშვათ საჭიროა, თითოეული დასახელების საქონლისათვის განისაზღვროს ჰიპერკავშირი, რომლის მეშვეობითაც სხვა Web-ფურცელს გადაეცემა ProductID-პარამეტრი, საქონლის შესახებ დაწვრილებითი ინფორმაციის ასახვის მიზნით. HTML-დოკუმენტში უნდა შეიქმნეს A-ელემენტი href-ატრიბუტით. სწორედ აქ დაგვჭირდება attribute ბრძანების დახმარება:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Web Page</TITLE>
         </HEAD>
         <BODY>
```

```
<P>Customer Order</P>
<P>Order No: <xsl:value-of select="Order/@OrderNo"/></P>
<P>Date: <xsl:value-of select="Order/OrderDate"/></P>
<P>Customer: <xsl:value-of select="Order/Customer"/></P>
<TABLE Border="0">
  <TR>
    <TD>ProductID</TD>
    <TD>Product Name</TD>
    <TD>Price</TD>
    <TD>Quantity Ordered</TD>
  </TR>
  <xsl:for-each select="Order/Item">
    <TR>
      <TD><xsl:value-of select="Product/@ProductID"/>
      </TD>
      <TD>
        <A>
          <xsl:attribute name="HREF">
           Products.asp?ProductID=<xsl:value-of
              select="Product/@ProductID"/>
           </xsl:attribute>
           <xsl:value-of select="Product"/>
        </A>
      </TD>
      <TD><xsl:value-of select="Product/@UnitPrice"/>
      </TD>
      <TD><xsl:value-of select="Quantity"/></TD>
    </TR>
  </xsl:for-each>
</TABLE>
</BODY>
```

```
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

ზემოთ განხილული შეკვეთის XML-ბლანკისთვის ამ XSL-ცხრილის გამოყენება მოგვცემს შემდეგ HTML-კოდს:

```
<HTML>
  <HEAD>
    <TITLE>Northwind Web Page</TITLE>
  </HEAD>
  <BODY>
    <P>Customer Order</P>
    <P>Order No: 1234</P>
    <P>Date: 2001-01-01</P>
    <P>Customer: Graeme Malcolm</P>
    <TABLE Border="0">
      <TR>
        <TD>ProductID</TD>
        <TD>Product Name</TD>
        <TD>Price</TD>
        <TD>Quantity Ordered</TD>
      </TR>
      <TR>
        <TD>1</TD>
        <TD><A HREF="Products.asp?ProductID=1">Chai</A></TD>
        <TD>18</TD>
        <TD>2</TD>
      </TR>
      <TR>
        <TD>2</TD>
        <TD><A HREF="Products.asp?ProductID=2">Chang</A></TD>
```

```
            <TD>19</TD>
            <TD>1</TD>
          </TR>
        </TABLE>
    </BODY>
</HTML>
```

## ერთ XSL-დოკუმენტში რამდენიმე შაბლონის გამოყენება

აქამდე განხილული XSL-დოკუმენტები მხოლოდ ერთადერთ შაბლონს შეიცავდა, რომელიც გამოიყენებოდა XML-დოკუმენტის ფესვური ელემენტისათვის.

XSL უშვებს XSL-დოკუმენტში რამდენიმე შაბლონის გამოყენების შესაძლებლობასაც.

ასეთი მიდგომის ღირსებებია:

- დოკუმენტის ნაწილების წარმოდგენის ლოგიკაში მეტი თავისუფლება, რაც ამარტივებს სტილის ცხრილების გაწყობისა და მოდიფიცირების პროცესებს;
- იქმნება შესაძლებლობა, XPath-გამოსახულებებით სხვადასხვაგვარად დაფორმატდეს XML-მონაცემები მათი მნიშვნელობების მიხედვით.

აღნიშნული მიდგომის განსახორციელებლად, ჩვეულებრივ, ჯერ ქმნიან ზედა დონის შაბლონს, რომელიც გამიზნულია დოკუმენტის, როგორც ერთი მთლიანობის, დასამუშავებლად.

ყველა დონის შაბლონი სტილის ცხრილში ჩაერთვება apply-templates ბრძანების მეშვეობით.

დამატებითი შაბლონების გამოძახება შესაძლებელია ნებისმიერ წერტილში.

ზედა დონის შაბლონი მხოლოდ იმ მონაცემებს ამუშავებს, რომლებიც დამატებითი შაბლონების მოქმედების ზონის მიღმა აღმოჩნდებიან.

ქვემოთ მოყვანილ მაგალითში სტილის ცხრილში გამოიყენება 4 შაბლონი:

1. ზედა დონის შაბლონი – გამოიყენება დოკუმენტის ფესვისათვის;
2. შაბლონი იმ Product-ელემენტებისათვის, რომელთათვისაც UnitPrice-ატრიბუტის მნიშვნელობა 18-ს აღემატება;
3. შაბლონი დანარჩენი Product-ელემენტებისათვის;
4. შაბლონი Quantity-ელემენტებისათვის.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Northwind Web Page</TITLE>
         </HEAD>
         <BODY>
            <P>Customer Order</P>
            <P>Order No: <xsl:value-of select="Order/@OrderNo"/></P>
            <P>Date: <xsl:value-of select="Order/OrderDate"/></P>
            <P>Customer: <xsl:value-of select="Order/Customer"/></P>
            <TABLE Border="0">
               <TR>
                  <TD>ProductID</TD>
                  <TD>Product Name</TD>
                  <TD>Price</TD>
                  <TD>Quantity Ordered</TD>
```

```
            </TR>
            <xsl:for-each select="Order/Item">
               <TR>
                  <xsl:apply-templates/>
               </TR>
            </xsl:for-each>
         </TABLE>
      </BODY>
   </HTML>
</xsl:template>

<xsl:template match="Product[@UnitPrice > 18]">
   <TD><xsl:value-of select="@ProductID"/></TD>
   <TD>
      <A>
         <xsl:attribute name="HREF">
         Products.asp?ProductID=<xsl:value-of select="@ProductID"/>
         </xsl:attribute>
         <xsl:value-of select="."/>
      </A>
   </TD>
   <TD>
      <FONT color="red">
         <xsl:value-of select="@UnitPrice"/>
      </FONT>
   </TD>
</xsl:template>

<xsl:template match="Product">
   <TD><xsl:value-of select="@ProductID"/></TD>
   <TD>
      <A>
         <xsl:attribute name="HREF">
```

```
            Products.asp?ProductID=<xsl:value-of select="@ProductID"/>
            </xsl:attribute>
            <xsl:value-of select="."/>
          </A>
        </TD>
        <TD><xsl:value-of select="@UnitPrice"/></TD>
    </xsl:template>

    <xsl:template match="Quantity">
        <TD><xsl:value-of select="."/></TD>
    </xsl:template>

</xsl:stylesheet>
```

ამ XSL-ცხრილის გამოყენება ზემოთ განხილული შეკვეთის XML-ბლანკისთვის მოგვცემს შემდეგ HTML-კოდს:

```
<HTML>
    <HEAD>
        <TITLE>Northwind Web Page</TITLE>
    </HEAD>
    <BODY>
        <P>Customer Order</P>
        <P>Order No: 1234</P>
        <P>Date: 2001-01-01</P>
        <P>Customer: Graeme Malcolm</P>
        <TABLE Border="0">
            <TR>
                <TD>ProductID</TD>
                <TD>Product Name</TD>
                <TD>Price</TD>
                <TD>Quantity Ordered</TD>
            </TR>
            <TR>
```

```
            <TD>1</TD>
            <TD><A HREF="Products.asp?ProductID=1">Chai</A></TD>
            <TD>18</TD>
            <TD>2</TD>
          </TR>
          <TR>
            <TD>2</TD>
            <TD><A HREF="Products.asp?ProductID=2">Chang</A></TD>
            <TD><FONT color="red">19</FONT></TD>
            <TD>1</TD>
          </TR>
        </TABLE>
    </BODY>
</HTML>
```

## სტილის ცხრილების გამოყენება

სტილის ცხრილების XML-დოკუმენტისთვის გამოყენება XML-ანალიზატორის ფუნქცია გახლავთ.

XML-ანალიზატორს სათანადო შეტყობინება მიეწოდება დასამუშავებელი XML-დოკუმენტიდან, რისთვისაც XML-დოკუმენტში უნდა გავითვალისწინოთ xml-stylesheet-ინსტრუქცია (მასში კი განვათავსებთ ინფორმაციას საჭირო ცხრილის შესახებ):

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Order.xsl"?>
<Order OrderNo="1234">
    <OrderDate>2001-01-01</OrderDate>
    <Customer>Graeme Malcolm</Customer>
    <Item>
        <Product ProductID="1" UnitPrice="18">Chai</Product>
```

```
    <Quantity>2</Quantity>
  </Item>
  <Item>
    <Product ProductID="2" UnitPrice="19">Chang</Product>
    <Quantity>1</Quantity>
  </Item>
</Order>
```

სხვადასხვა ანალიზატორში სტილის ცხრილების გამოყენება განსხვავებულად ხდება. Microsoft იყენებს საკუთარ მიდგომას, რომლის რეალიზება ხდება ქვემოთ მოყვანილი კოდით:

```
Dim objXML
Dim objXSL
Dim strResult

‘Load the XML document.
Set objXML = CreateObject("Microsoft.XMLDom")
objXML.Async = False
objXML.Load "c:\Order.xml"

‘Load the XSL style sheet.
Set objXSL = CreateObject("Microsoft.XMLDom")
objXSL.Async = False
objXSL.Load "c:\Order.xsl"

‘Apply the style sheet.
strResult = objXML.transformNode(objXSL)
```

# მონაცემთა XML-სქემები

XML-ტექნოლოგიაში კიდევ ერთი მნიშვნელოვანი ნაბიჯი გადაიდგა წინ XML-სქემების შემოღებით. მათი მეშვეობით ხდება ინფორმაციის გამცვლელ პუნქტებს, ვთქვათ, კომპანიებს შორის გადასაცემი დოკუმენტების სისწორის უფრო დეტალური შემოწმება. მაგალითად, კომპანიები შეიძლება წინასწარ შეთანხმდნენ, დოკუმენტში ინფორმაციის რომელი ელემენტები უნდა ფიგურირებდეს აუცილებლად, მაქსიმუმ რამდენი ეგზემპლარის სახით და ა.შ. ამ მხრივ, სქემები გაცილებით მოქნილი ინსტრუმენტია, ვიდრე Document Type Definition (DTD) სახელით ცნობილი ტექნოლოგია.

სახელმძღვანელოში ჩვენ გავეცნობით World Wide Web Consortium (W3C)-ის მიერ შემოთავაზებულ რეკომენდაციას აღნიშნული პრობლემის გადასაწყვეტად. იგი წარმოადგენს XML-Data სინტაქსის ქვესიმრავლეს და მისი სახელწოდებაა:

XML-Data Reduced (XDR).

## XDR-სქემის შექმნა

XDR-სქემა წარმოადგენს XML-დოკუმენტს, რომელშიც გამოცხადებული ელემენტები და ატრიბუტები შეიძლება გამოვიყენოთ ამ სქემაზე დაფუძნებულ XML-დოკუმენტებში.

ამასთან, სქემაში, როგორც წესი, მიუთითებენ მონაცემთა ელემენტების და ატრიბუტების ტიპებს, დოკუმენტებში მათი განლაგების მიმდევრობას, მნიშვნელობების სიგრძის დიაპაზონს *(მინიმალურ და მაქსიმალურ სიგრძეებს),* ეგზემპლარების მინიმალურ და მაქსიმალურ რაოდენობებს.

სქემაში დასაშვებია, აგრეთვე, ვუჩვენოთ, შეიძლება თუ არა დოკუმენტში ფიგურირებდეს ისეთი ელემენტები და ატრიბუტები, რომლებიც სქემით არ არის განსაზღვრული.

სქემები ეფუძნება XML-Data მონაცემთა სივრცეს, ანუ, სწორედ, ეს სივრცე განსაზღვრავს ატრიბუტებისა და ელემენტების გამოცხადების წესებს – სინტაქსს.

სქემის უმაღლესი დონის ელემენტია Schema. ეს ელემენტი შეიცავს არააუცილებელ name ატრიბუტს. მინიმალურ სქემას შემდეგი სახე აქვს:

```
<?xml version="1.0"?>
<Schema name="orderschema"
   xmlns="urn:schemas-microsoft-com:xml-data">
</Schema>
```

სქემას რომ რაიმე სასარგებლო ფუნქცია შევასრულებინოთ, ცხადია, მასში უნდა გამოვაცხადოთ ელემენტები და ატრიბუტები, რაც მიიღწევა შემდეგი საკვანძო სიტყვების მეშვეობით:

ElemetType და AttributeType.

ელემენტების გამოცხადება ხდება სქემის ზედა დონეზე.

ატრიბუტები გლობალური ტიპის ცვლადებია. ამასთან, მოცემული ატრიბუტი შეიძლება გამოყენებულ იქნეს როგორც ერთადერთ, ასევე რამდენიმე ელემენტში.

მას შემდეგ, რაც გამოცხადდება ელემენტი და/ან ატრიბუტი, მათი ეგზემპლარების გამოსაცხადებლად მიმართავენ შემდეგ საკვანძო სიტყვებს:

element და attribute.

ხაზგასმით აღვნიშნავთ, რომ XML-კოდში განირჩევა დიდი და პატარა ასოები.

ვაჩვენოთ სქემის მაგალითი XML-შეკვეთის ბლანკის განსაზღვრისათვის. სქემა აცხადებს, რომელ ატრიბუტებს და ელემენტებს შეიძლება შეიცავდეს ეს ბლანკი:

```
<?xml version="1.0"?>
<Schema name="orderschema"
   xmlns="urn:schemas-microsoft-com:xml-data">
```

```
  <ElementType name="OrderDate"/>
  <ElementType name="Customer"/>
  <ElementType name="Product">
    <AttributeType name="ProductID"/>
    <AttributeType name="UnitPrice"/>
    <attribute type="ProductID"/>
    <attribute type="UnitPrice"/>
  </ElementType>
  <ElementType name="Quantity"/>
  <ElementType name="Item">
    <element type="Product"/>
    <element type="Quantity"/>
  </ElementType>
  <ElementType name="Order">
    <AttributeType name="OrderNo"/>
    <attribute type="OrderNo"/>
    <element type="OrderDate"/>
    <element type="Customer"/>
    <element type="Item"/>
  </ElementType>
</Schema>
```

ამ სქემაში პირველი ორი ელემენტის გამოცხადება მარტივად ხდება, რაც შეეხება მესამე ელემენტს – Product-ს, იგი უფრო რთული აგებულებისაა – შეიცავს ორი ატრიბუტისა და მათი თითო-თითო ეგზემპლარის გამოცხადებებს.

შემდეგ გამოცხადებულია მომდევნო ელემენტები: Quantity და Item. უკანასკნელში, თავის მხრივ, ხდება Product და Quantity ელემენტების ეგზემპლარების გამოცხადება *(element საკვანძო სიტყვებით).*

დაბოლოს, სქემაში ცხადდება Order ელემენტი. იგი შეიცავს OrderNo ატრიბუტის განსაზღვრებას, ამავე ატრიბუტის ეგზემპლარის

განსაზღვრებას და შვილობილი ელემენტების OrderDate, Customer და Item-ის ეგზემპლარებს.

მოცემულ XML-სქემაზე დაფუძნებულ შეკვეთის XML-ბლანკს შეიძლება ასეთი სახე ჰქონდეს:

```
<?xml version="1.0"?>
<Order OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer>Graeme Malcolm</Customer>
   <Item>
      <Product ProductID="1" UnitPrice="18">Chai</Product>
      <Quantity>2</Quantity>
   </Item>
   <Item>
      <Product ProductID="2" UnitPrice="19">Chang</Product>
      <Quantity>1</Quantity>
   </Item>
</Order>
```

## შემცველობის ღია და დახურული მოდელები

დუმილით, დასაშვებია სქემაზე დაფუძნებულ დოკუმენტში სხვა, სქემაში არარსებული ელემენტების და ატრიბუტების ჩართვაც. მაგალითად, შესაძლებელია ზემოთ მოყვანილ შეკვეთის ბლანკში ჩავამატოთ DeliveryDate ელემენტი, რომელიც აქამდე სქემაში არ ფიგურირებდა:

```
<?xml version="1.0"?>
<Order OrderNo="1234">
   <OrderDate>2001-01-01</OrderDate>
   <Customer>Graeme Malcolm</Customer>
   <DeliveryDate>2001-02-02</DeliveryDate>
   <Item>
```

```
    <Product ProductID="1" UnitPrice="18">Chai</Product>
    <Quantity>2</Quantity>
  </Item>
  <Item>
    <Product ProductID="2" UnitPrice="19">Chang</Product>
    <Quantity>1</Quantity>
  </Item>
</Order>
```

მაგრამ თუკი გვსურს, „თვითშემოქმედება“ აიკრძალოს, მაშინ სქემაში უნდა გავითვალისწინოთ model ატრიბუტი, „closed“ მნიშვნელობით.

*შენიშვნა: როცა ატრიბუტს არ მივუთითებთ, იგი იგულისხმება და ავტომატურად „open“ მნიშვნელობას ღებულობს.*

აკრძალვა უნდა განვახორციელოთ თითოეული ელემენტისთვის. მათში აღარ დაიშვება დამატებითი ქვეელემენტების და ატრიბუტების შეტანა.

დახურულად ვაქციოთ ზემოთ განხილული სქემის ყველა ელემენტის მოდელი:

```
<?xml version="1.0"?>
<Schema name="orderschema"
  xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="OrderDate" model="closed"/>
  <ElementType name="Customer" model="closed"/>

  <ElementType name="Product" model="closed">
    <AttributeType name="ProductID"/>
    <AttributeType name="UnitPrice"/>
    <attribute type="ProductID"/>
    <attribute type="UnitPrice"/>
  </ElementType>
```

```
  <ElementType name="Quantity" model="closed"/>
  <ElementType name="Item" model="closed">
    <element type="Product"/>
    <element type="Quantity"/>
  </ElementType>
  <ElementType name="Order" model="closed">
    <AttributeType name="OrderNo"/>
    <attribute type="OrderNo"/>
    <element type="OrderDate"/>
    <element type="Customer"/>
    <element type="Item"/>
  </ElementType>
</Schema>
```

## ელემენტის შემცველობის შეზღუდვა

ვიცით, რომ ელემენტი შეიძლება შეიცავდეს როგორც ტექსტურ მნიშვნელობას, ასევე – ჩადგმულ ელემენტებსაც.

დასაშვებია ელემენტის შემცველობაზე შეზღუდვების შემოღება, რისთვისაც გამოიყენება content ატრიბუტი შემდეგი შესაძლო მნიშვნელობებით:

- textOnly – დასაშვებია მხოლოდ ტექსტი;
- eltOnly – დასაშვებია მხოლოდ ჩადგმული ელემენტები;
- mixed – დაიშვება ტექსტიც და ჩადგმული ელემენტებიც;
- empty – ელემენტი ცარიელი უნდა იყოს.

ვაჩვენოთ ისეთი სქემის შექმნის მაგალითი, რომელზეც დაფუძნებულ XML-დოკუმენტში OrderDate, Customer და Quantity ელემენტები შეიცავენ მხოლოდ ტექსტს, ხოლო Item ელემენტი – მხოლოდ ჩადგმულ ელემენტებს.

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

<Schema name="orderschema"
   xmlns="urn:schemas-microsoft-com:xml-data">

   <ElementType name="OrderDate" model="closed" content="textOnly"/>
   <ElementType name="Customer" model="closed" content="textOnly"/>

   <ElementType name="Product" model="closed" content="mixed">
      <AttributeType name="ProductID"/>
      <AttributeType name="UnitPrice"/>
      <attribute type="ProductID"/>
      <attribute type="UnitPrice"/>
   </ElementType>

   <ElementType name="Quantity" model="closed" content="textOnly"/>
   <ElementType name="Item" model="closed" content="eltOnly">
      <element type="Product"/>
      <element type="Quantity"/>
   </ElementType>

   <ElementType name="Order" model="closed" content="mixed">
      <AttributeType name="OrderNo"/>
      <attribute type="OrderNo"/>
      <element type="OrderDate"/>
      <element type="Customer"/>
      <element type="Item"/>
   </ElementType>
</Schema>
```

## მონაცემთა ელემენტების რიცხვის შეზღუდვა

სქემაში შეიძლება განისაზღვროს დოკუმენტისათვის აუცილებელი მონაცემები, აგრეთვე ელემენტების ეგზემპლარების მინიმალური და მაქსიმალური რიცხვები. გავეცნოთ ამ და ზოგიერთ სხვა შესაძლებლობას.

### ატრიბუტების იძულებითი ჩასმა

მოცემული ატრიბუტი ელემენტში მხოლოდ ერთხელ შეიძლება გამოვიყენოთ, ამიტომ საჭირო არ არის მისი ეგზემპლარისთვის მინიმალურ და მაქსიმალურ რაოდენობათა მითითება. რაც შეეხება დოკუმენტში ატრიბუტის აუცილებლად ჩართვის უზრუნველყოფას, ეს ხდება ატრიბუტის გამოცხადების AttributeType ტეგში required ატრიბუტის ჩართვით და მისთვის „Yes“ მნიშვნელობის მიცემით. შედეგად, თუ მოცემული ატრიბუტი XML-დოკუმენტში ნაჩვენები არ იქნა, ანალიზატორის მიერ ეს დოკუმენტი არასწორად ჩაითვლება.

ატრიბუტის ეგზემპლარის (და არა ატრიბუტის) გამოცხადების attribute ტეგში დასაშვებია default ატრიბუტის ჩვენებაც, მისთვის შესაბამისი მნიშვნელობის განსაზღვრით. ეს მნიშვნელობა გამოყენებული იქნება მაშინ, თუ დოკუმენტში გამოვტოვებთ სქემაში მოცემულ ატრიბუტს.

### ელემენტთა ეგზემპლარების რიცხვის შეზღუდვა

იმისათვის, რომ დოკუმენტში გარანტირებული იყოს ელემენტის არსებობა, მისი ეგზემპლარის გამოცხადების ტეგში არააუცილებელ minOccurs ატრიბუტს მივცემთ ერთის ტოლ მნიშვნელობას, ხოლო maxOccurs-ისთვის შესაბამისი მნიშვნელობის განსაზღვრით კი დავადგენთ დოკუმენტში ელემენტის ეგზემპლარების მაქსიმალურ რიცხვს.

ქვემოთ ნაჩვენებია აღნიშნული ატრიბუტების სქემაში გამოყენების მაგალითები:

```
<?xml version="1.0"?>
<Schema name="orderschema"
   xmlns="urn:schemas-microsoft-com:xml-data">
   <ElementType name="OrderDate" model="closed" content="textOnly"/>
   <ElementType name="Customer" model="closed" content="textOnly"/>
```

```
  <ElementType name="Product" model="closed" content="mixed">
    <AttributeType name="ProductID" required="yes"/>
    <AttributeType name="UnitPrice"/>
    <attribute type="ProductID"/>
    <attribute type="UnitPrice" default="10.00"/>
  </ElementType>

  <ElementType name="Quantity" model="closed" content="textOnly"/>
  <ElementType name="Item" model="closed" content="eltOnly">
    <element type="Product" minOccurs="1" maxOccurs="1"/>
    <element type="Quantity" minOccurs="1" maxOccurs="1"/>
  </ElementType>

  <ElementType name="Order" model="closed" content="mixed">
    <AttributeType name="OrderNo" required="yes"/>
    <attribute type="OrderNo"/>
    <element type="OrderDate" minOccurs="1" maxOccurs="1"/>
    <element type="Customer" minOccurs="1" maxOccurs="1"/>
    <element type="Item" minOccurs="1" maxOccurs="*"/>
  </ElementType>
</Schema>
```

ჩამოვწეროთ ქაღალდზე, რომელ ელემენტებს როგორი სახის შეზღუდვები დაედოთ და შევადაროთ მიღებული შედეგი ქვემოთ ჩამოწერილ სიას (იხ. ცხრილი A).

***ცხრილი A***

1. Order ელემენტის OrderNo ატრიბუტი და Product ელემენტის ProductAID ატრიბუტი აუცილებელი სახისაა;
2. UnitPrice ატრიბუტისათვის დუმილით გაითვალისწინება 10.00 მნიშვნელობა;
3. Item ელემენტი შეიძლება შეიცავდეს Product და Quantity ელემენტების მხოლოდ თითო-თითო ეგზემპლარს;

4. Order ელემენტი შეიძლება შეიცავდეს Orderdate და Customer ელემენტების მხოლოდ თითო-თითო ეგზემპლარს;
5. დოკუმენტში უნდა ფიგურირებდეს ერთი Item ელემენტი მაინც. ამასთან, დასაშვებია მისი მრავალჯერ გამეორებაც.

## მონაცემთა ტიპების ჩვენება

დოკუმენტებში შეცდომების რიცხვის შესამცირებლად სქემები დამატებით ხერხებსაც გთავაზობენ მონაცემთა ტიპების წინასწარი განსაზღვრისა და შემდეგ მათი შემოწმების გზით.

მონაცემთა ტიპების ჩვენება ხდება datatypes სახელთა სივრცეზე დაყრდნობით.

ვაჩვენოთ ზოგიერთი ასეთი ტიპის (date, string, fixed, int) გამოყენების მაგალითები:

```
<?xml version="1.0"?>
<Schema name="orderschema"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">

   <ElementType name="OrderDate" model="closed" content="textOnly"
      dt:type="date"/>
   <ElementType name="Customer" model="closed" content="textOnly"
      dt:type="string"/>

   <ElementType name="Product" model="closed" content="mixed">
      <AttributeType name="ProductID" required="yes" dt:type="int"/>
      <AttributeType name="UnitPrice" dt:type="fixed.14.4"/>
      <attribute type="ProductID"/>
      <attribute type="UnitPrice" default="10.00"/>
   </ElementType>

   <ElementType name="Quantity" model="closed" content="textOnly"
      dt:type="int"/>
```

```
  <ElementType name="Item" model="closed" content="eltOnly">
    <element type="Product" minOccurs="1" maxOccurs="1"/>
    <element type="Quantity" minOccurs="1" maxOccurs="1"/>
  </ElementType>

  <ElementType name="Order" model="closed" content="mixed">
    <AttributeType name="OrderNo" required="yes" dt:type="int"/>
    <attribute type="OrderNo"/>
    <element type="OrderDate" minOccurs="1" maxOccurs="1"/>
    <element type="Customer" minOccurs="1" maxOccurs="1"/>
    <element type="Item" minOccurs="1" maxOccurs="*"/>
  </ElementType>
</Schema>
```

შევნიშნოთ, რომ UnitPrice-ისთვის განისაზღვრება ფიქსირებული ათობითი რიცხვი (14 ნიშნით ათობით წერტილამდე და 4-მდე ნიშნით მის შემდეგ).

## XML დოკუმენტის შემოწმება

იმისათვის, რათა უზრუნველვყოთ დოკუმენტის შემოწმება სქემის მიხედვით, დოკუმენტის სახელთა სივრცეში უნდა მივუთითოთ X-schema საკვანძო სიტყვა შესაბამისი მნიშვნელობის ჩვენებით.

ქვემოთ მოყვანილი დოკუმენტი ეყრდნობა Orderschema.xml სქემას:

```
<?xml version="1.0"?>
<Order OrderNo="1234" xmlns="x-schema:Orderschema.xml">
  <OrderDate>2001-01-01</OrderDate>
  <Customer>Graeme Malcolm</Customer>
  <Item>
    <Product ProductID="1" UnitPrice="18">Chai</Product>
    <Quantity>2</Quantity>
  </Item>
```

```
    <Item>
        <Product ProductID="2" UnitPrice="19">Chang</Product>
        <Quantity>1</Quantity>
    </Item>
</Order>
```

# ლიტერატურა


1. გ. ღვინეფაძე. WEB-დაპროგრამება. HTML5. სახელმძღვანელო. თბილისი. “ტექნიკური უნივერსიტეტი”. 2013. ISBN http://gtu.ge/books.php/
2. http://www.intuit.ru/department/internet/html5fwcd/0/
3. http://www.intuit.ru/department/internet/jsbasics/13/1.html
4. გ. ღვინეფაძე. WEB-დაპროგრამება. Javascript. სახელმძღვანელო. თბილისი. “ტექნიკური უნივერსიტეტი”. 2009. ISBN 99940-14-80-3. http://gtu.ge/books.php/ 681.3(06) /203
5. http://www.webmasterwiki.ru/AJAX
6. WEB-ტექნოლოგიების სტანდარტების საიტი http://www.w3schools.com


# შინაარსი


WEB 2.0 ---------- 3

AJAX ტექნოლოგია ---------- 6

XML ---------- 28

მონაცემთა XML-სქემები ---------- 56

*ლიტერატურა* ---------- 67