გ. ღვინეფაძე

WEB-დაპროგრამება

III ნაწილი

X M L

“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

WEB-დაპროგრამება

III ნაწილი

X M L

დამტკიცებულია
სტუ-ს
სასწავლო-მეთოდური
საბჭოს მიერ

თბილისი
2007

უაკ 681.3.06

განხილულია ჰიპერტექსტების გაწყობისათვის განკუთვნილი სპეციალიზებული ენა XML. განკუთვნილია ინფორმატიკის სპეციალობათა (22.02, 22.01, 0719.08) შემსწავლელი სტუდენტებისა და ამ საკითხით დაინტერესებული პირებისთვის.

რეცენზენტები: პროფ. ო. ნატროშვილი,
ასოც. პროფ. თ. სუხიაშვილი

ISBN

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 ფაქტიურად ნებისმიერად არჩეულ 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 აუცილებელი, მაგრამ არასაკმარისი პირობა გახლავთ. საქმე ისაა, რომ “კორექტული დოკუმენტი” ავტომატურად არ ნიშნავს “სწორ დოკუმენტს” (აქ შეიძლება ანალოგიად მოვიყვანოთ ჩვეულებრივი

წინადადებისთვის სინტაქსისა და სემანტიკის ურთიერთმიმართების საკითხი).

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

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

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

სპეციფიკატორიც დოკუმენტია, ოღონდ DTD-ტიპისა (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 დოკუმენტი უნდა შედგეს ქვემოთ ჩამოთვლილი წესების მიხედვით:

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

ინსტრუქციები, კომენტარები

მონაცემების გარდა, XML-დოკუმენტი შეიძლება შეიცავდეს ინსტრუქციებს XML-ანალიზატორისთვის. მათი მეშვეობით ანალიზატორს ეცნობება XML-სპეციფიკაციის ვერსია, რომელიც იქნა გათვალისწინებული XML-დოკუმენტზე მუშაობისას. ინსტრუქციაშივე შეიძლება მიეთითოს სტილების ცხრილიც და სხვ. ეს ინსტრუქციები სპეციალური სახის ტეგში ჩაისმება:

<? *ინსტრუქცია* ?>

მოვიყვანოთ ინსტრუქციის გამოყენების მაგალითი, რომელიც განთავსებულია ფესვური ელემენტის წინ:

```
<?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

შესაძლოა, XML-დოკუმენტში ფიგურირებდეს ერთნაირი სახელის, მაგრამ განსხვავებული დანიშნულების (შესაბამისად, განსხვავებული ტიპის მონაცემების შემცველი) ელემენტები. მაგალითად, 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 დასახელება.

გაურკვევლობის თავიდან ასაცილებლად ასეთ შემთხვევაში XML იყენებს ე.წ. სახელების სივრცეებს.

სახელების სივრცის არსი შემდეგში მდგომარეობს – მისი მეშვეობით ხდება XML-დოკუმენტის ელემენტების და ატრიბუტების დაკავშირება (მიბმა) რაიმე უნივერსალურ იდენტიფიკატორთან – URI-სთან. Universal Resource Indentifier, ანუ რესურსის უნივერსალური მაჩვენებელი შეიძლება წარმოადგენდეს URL-ს, მაგრამ შესაძლოა, იგი გახლდეთ რაიმე სხვა უნიკალური იდენტიფიკატორიც (ე.ი არ წარმოადგენდეს რესურსს ინტერნეტიდან).

სახელების სივრცე შესაძლებელია ნებისმიერ ელემენტში გამოცხადდეს.

ამ მიზნით 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

http://www.northwindtraders.com/customer

არსებულ სახელების სივრცეს, ხოლო Book ელემენტი და, ცხადია, მასში ჩადგმული ასევე title სახელის მქონე სხვა ელემენტი

http://www.northwindtraders.com/book

სახელების სივრცეს.

ვხედავთ, რომ სახელთა სივრცეების ამგვარი წესით გამოცხადება მთლად მოხერხებული არ გახლავთ, განსაკუთრებით მაშინ, როცა მათთან კავშირდება 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 წინ წარუმძღვანებენ 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 დასახელებებში პრეფიქსი არ ფიგურირებს (ესენია: BookOrder, OrderNo და OrderDate), მიეკუთვნება დუმილით განსაზღვრულ სახელების სივრცეს.

ამ თემის უფრო ღრმად შესწავლამდე საჭიროა გავეცნოთ საკითხს, როგორ ხდება XML-დოკუმენტის დამუშავება.

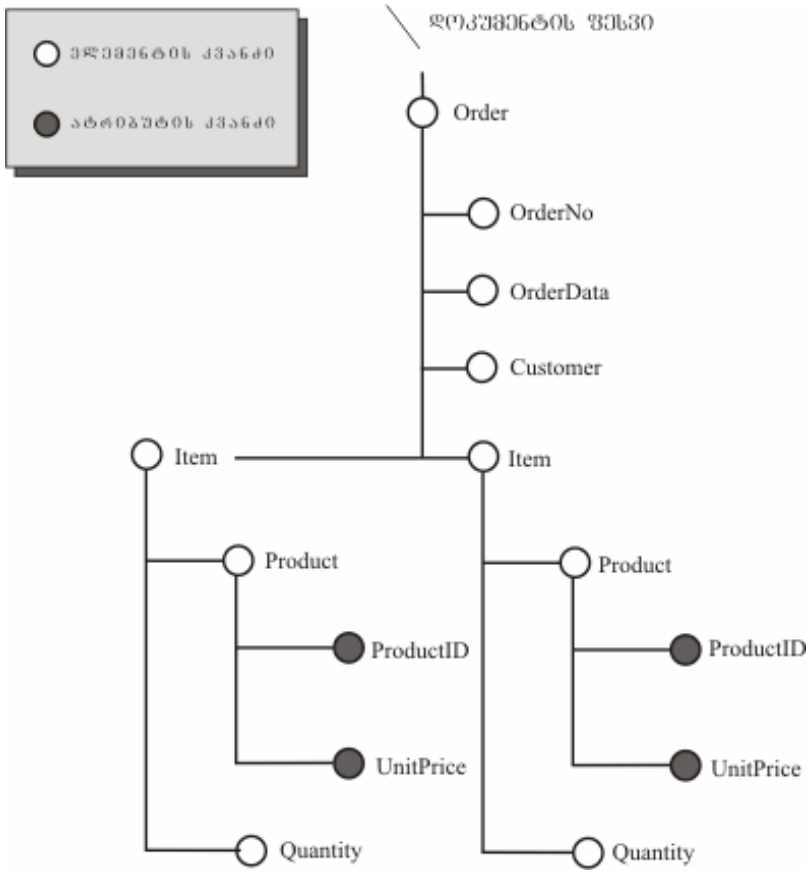
მოგზაურობა XML-დოკუმენტის სამყაროში

XML-დოკუმენტს ამუშავებს რაიმე პროგრამა (გამოყენება). ცხადია, ინფორმაციის მისაღებად პროგრამას უნდა შეეძლოს დოკუმენტის სხვადასხვა ფრაგმენტების მოძიება, ანუ მოგზაურობა ელემენტებისა და ატრიბუტების სივრცეში, რათა გაიგოს და გამოიყენოს მათი მნიშვნელობები. ამ საქმეში პროგრამას ეხმარება 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 გზების ჩვენება

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

ჩვენ განვიხილავთ შემოკლებული სახის სინტაქსისს.

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 სინტაქსისი იყენებს @ სიმბოლოს. მოვიყვანოთ მაგალითი:

/Order/@OrderNo

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

/Order//Product

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

/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 იგება მიმდინარე კვანძის მდებარეობის დუმილით გათვალისწინების შედეგად Xpath-გამოსახულებაში.

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. ოვიყვანოთ მაგალითი:

.. /@OrderNo

ზოგჯერ პროგრამას შეიძლება დასჭირდეს, გაარკვიოს, რომელია მიმდინარე კვანძი. ამ მიზნით სინტაქსისი 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. სინტაქსისი მოითხოვს, რომ კრიტერიუმი-გამოსახულება კვადრატულ ფრჩხილში იქნეს ჩასმული.

შემდეგი Xpath-გამოსახულება გვიბრუნებს ყველა ისეთ Product-ელემენტს, რომლებისთვისაც UnitPrice-ატრიბუტის მნიშვნელობა 18 ერთეულს არ 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 ტოლი გახლავთ.

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

XSL-დოკუმენტები

XSL-დოკუმენტი თვითონაც XML-დოკუმენტს წარმოადგენს, რომელშიც XSL-ბრძანებების საფუძველზე განისაზღვრება სტილების 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)

ახლა კი 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, რომ აკმაყოფილებდეს დოკუმენტის კორექტულობისადმი ყველა მოთხოვნას. მოვიყვანოთ უმარტივესი XSL-დოკუმენტის მაგალითი, რომელშიც განსაზღვრული სტილების XSL-ცხრილის საშუალებით 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 ფაქტიურად იგი არავითარ ისეთ ოპერაციას არ ასრულებს, რისთვისაც საერთოდ იყენებენ სტილების XSL-ცხრილებს. ამ შაბლონს საწყისი XML-დოკუმენტიდან ბროუზერის ეკრანზე გამოჰყავს მხოლოდ მის მიერვე წინასწარ ზუსტად განსაზღვრული HTML-დოკუმენტი. საერთოდ კი, 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 დაეკისროს XML-ბლოკიდან მონაცემთა გარკვეული ნაწილის ამოღება:

```
<?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 “l”, 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 ხშირად გამოიყენება ერთსახელა ელემენტები ერთნაირი ტიპის არსთა ჩამოთვლისათვის. მაგალითად, ზემოთ განხილული შეკვეთის ბლანკი ორ Item-ელემენტს შეიცავს, 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) ფორმირდება ცხრილისთვის 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 შეიცავდნენ, 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 სტილების 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 სტილების ცხრილში ჩაერთვება apply-template ბრძანების მეშვეობით.

დამატებითი შაბლონების გამოძახება ნებისმიერ წერტილშია შესაძლებელი.

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

ქვემოთ მოყვანილ მაგალითში სტილების ცხრილში 4 შაბლონი გამოიყენება:

1. ზედა დონის შაბლონი - გამოიყენება დოკუმენტის ფესვისათვის;
2. შაბლონი იმ Product-ელემენტებისათვის, რომელთათვის UnitPrice-ატრიბუტის მნიშვნელობა 18-ს არ 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>
```

სტილების ცხრილების გამოყენება

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

XML-ანალიზატორისთვის სათანადო შეტყობინების მიწოდება ხორციელდება დასამუშავებელი XML-დოკუმენტიდან, რისთვისაც XML-დოკუმენტში უნდა გავითვალისწინოთ xml-stylesheet-ინსტრუქცია (მასში კი ვუჩვენებთ 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 ანალიზატორებში სტილების 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 წინასწარ შეიძლება შეთანხმდნენ, დოკუმენტში ინფორმაციის რომელი ელემენტები უნდა ფიგურირებდეს აუცილებლად, მაქსიმუმ რამდენი ეგზემპლიარის სახით და ა.შ. ამ მხრივ, სქემები გაცილებით მოქნილი ინსტრუმენტია, ვიდრე Document Type Definition (DTD) სახელით ცნობილი ძველი ტექნოლოგია.

სახელმძღვანელობაში ჩვენ გავეცნობით World Wide Web Consortium (W3C)-ის მიერ შემოთავაზებულ რეკომენდაციას აღნიშნული პრობლემის გადასაწყვეტად. იგი წარმოადგენს XML-Data სინტაქსის (რომლის დახვეწაზეც მუშაობა გრძელდება) 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), ეგზემპლიარების 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>
```

სქემამ რომ რაიმე სასარგებლო ფუნქცია შეასრულოს, ცხადია, მასში უნდა გამოვაცხადოთ ელემენტები და ატრიბუტები, რაც ხდება შემდეგი საკვანძო სიტყვების მეშვეობით:

ElemetType და AttributeType.

ელემენტების გამოცხადება ხდება სქემის ზედა დონეზე, ატრიბუტები გლობალური ტიპის ცვლადებია. ამასთან, ერთი ატრიბუტი შეიძლება გამოყენებულ იქნეს ერთადერთ ან რამდენიმე ელემენტში.

მას შემდეგ, რაც გამოცხადდება ელემენტი და/ან ატრიბუტი, მათი ეგზემპლიარების 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 ეგზემპლიარის გამოცხადებებს.

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

დაბოლოს, სქემაში ცხადდება Order ელემენტი. იგი შეიცავს OrderNo ატრიბუტის განსაზღვრებას, ამავე ატრიბუტის ეგზემპლიარის განსაზღვრებას და შვილობილი ელემენტების OrderDate, Customer და Item-ის ეგზემპლიარებს.

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 სქემაში არ ფიგურირდება:

```
<?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 ხდება 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 ელემენტებს.

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 ეგზემპლიარების მინიმალური და მაქსიმალური რიცხვები. გავეცნოთ ამ და ზოგიერთ სხვა შესაძლებლობას.

ატრიბუტების იძულებითი ჩასმა

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

ატრიბუტის ეგზემპლიარის (და არა ატრიბუტის) გამოცხადების attribute ტეგში დასაშვებია default ატრიბუტის ჩვენებაც, მისთვის შესაბამისი მნიშვნელობის მინიჭებით. ეს მნიშვნელობა გამოყენებული იქნება მაშინ, როცა დოკუმენტში გამოიტოვება სქემაში განსაზღვრული ატრიბუტი.

ელემენტთა ეგზემპლიარების რიცხვის შეზღუდვა

იმისათვის, რომ დოკუმენტში გარანტირებული იყოს ელემენტის არსებობა, მისი ეგზემპლიარის გამოცხადების ტეგში არააუცილებელ minOccurs ატრიბუტს მივანიჭებთ ერთის ტოლ მნიშვნელობას, ხოლო maxOccurs-სთვის შესაბამისი მნიშვნელობის განსაზღვრით კი დავადგენთ დოკუმენტში ელემენტის ეგზემპლიარების მაქსიმალურ რიცხვს.

ქვემოთ ნაჩვენებია სქემაში აღნიშნული ატრიბუტების 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 რა და რა სახის შეზღუდვები დაედოთ და შევადაროთ მიღებული შედეგი ქვემოთ ჩამოწერილ სიას (იხ. ცხრილი A).

ცხრილი A

1. Order ელემენტის OrderNo ატრიბუტი და Product ელემენტის ProduAID ატრიბუტი აუცილებელი სახისაა;
2. UnitPrice ატრიბუტისათვის დუმილით გაითვალისწინება 10.00 მნიშვნელობა;
3. Item ელემენტი შეიძლება შეიცავდეს Orderdate და Quantity ელემენტების მხოლოდ თითო-თითო ეგზემპლიარს;
4. Order ელემენტი შეიძლება შეიცავდეს Orderdate და Customer ელემენტების მხოლოდ თითო-თითო ეგზემპლიარს;
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-სთვის განისაზღვრება ფიქსირებული ათობითი რიცხვი 14 ნიშნით ათობით წერტილამდე და 4-მდე ნიშნით მის შემდეგ.

XML - დოკუმენტის შემოწმება

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

ქვემოთ მოყვანილი დაკუმენტი 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. Грэм Малкольм. Программирование для Microsoft SQL Server 2000 с использованием XML, Москва , 2002.

2. Учебный курс «Компьютерные сети» , Microsoft Press. Санкт-Петербург, 1999.

შინაარსი:

XML