ბ. ღვინეფაძე

WEB-დაპროგრამება

XML, მონაცემთა ბაზები და ინტერნეტი

“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

XML, მონაცემთა ბაზები და ინტერნეტი

დამტკიცებულია დამხმარე
სახელმძღვანელოდ სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2007

უაკ 681.3.06

წიგნი წარმოადგენს პრაქტიკულ სახელმძღვანელოს SQL და XML-ის ბაზაზე ბიზნეს-გამოყენებების შექმნისა და ინტერნეტში განთავსებისთვის.

იგი განკუთვნილია ინფორმატიკის 22.02, 22.01 და 0719.08 სპეციალობათა შემსწავლელი სტუდენტებისა და ამ საკითხით დაინტერესებული პირებისთვის.

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

ISBN 99940-40-19-7

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 <u>კორექტულად</u> და <u>სწორად</u>.

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 სპეციფიკატორთან.

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 სტილების ცხრილიც 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

დასაშვებია, 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 მათგანი არის მიმართვა მყიდველისადმი, მეორე კი - წიგნის დასახელება.

XML ასეთ შემთხვევებში გაურკვევლობის თავიდან ასაცილებლად 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 იყოს რაიმე სხვა უნიკალური იდენტიფიკატორიც, ანუ არ წარმოადგენდეს რესურსს ინტერნეტიდან.

სახელების სივრცე ნებისმიერ ელემენტში შეიძლება გამოცხადდეს.

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

დავაზუსტოთ წიგნებზე მოთხოვნა შემდეგი სახით:

```
<?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 სივრცესთან ბევრი ელემენტი და ატრიბუტი კავშირდება. ასეთ შემთხვევებში მიმართავენ საკითხის ალტერნატიულ გადაწყვეტას:

ყველა ასეთი სივრცე გამოცხადდება ფესვურ ელემენტში; ერთ-ერთი მათგანი დუმილით გაითვალისწინება პრეფიქსით მოუნიშნავი ელემენტებისა და ატრიბუტებისათვის, ხოლო 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 ქვეელემენტებს წინ წარუმძღვანებენ 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 - ეს სივრცე გამოიყენება დუმილით;
- http://www.nothwindtraders.com/customer სივრცისთვის გამოცხადებულია cust აბრევიატურა;
- http://www.nothwindtradesr.com/book სივრცისთვის გამოცხადებულია 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.

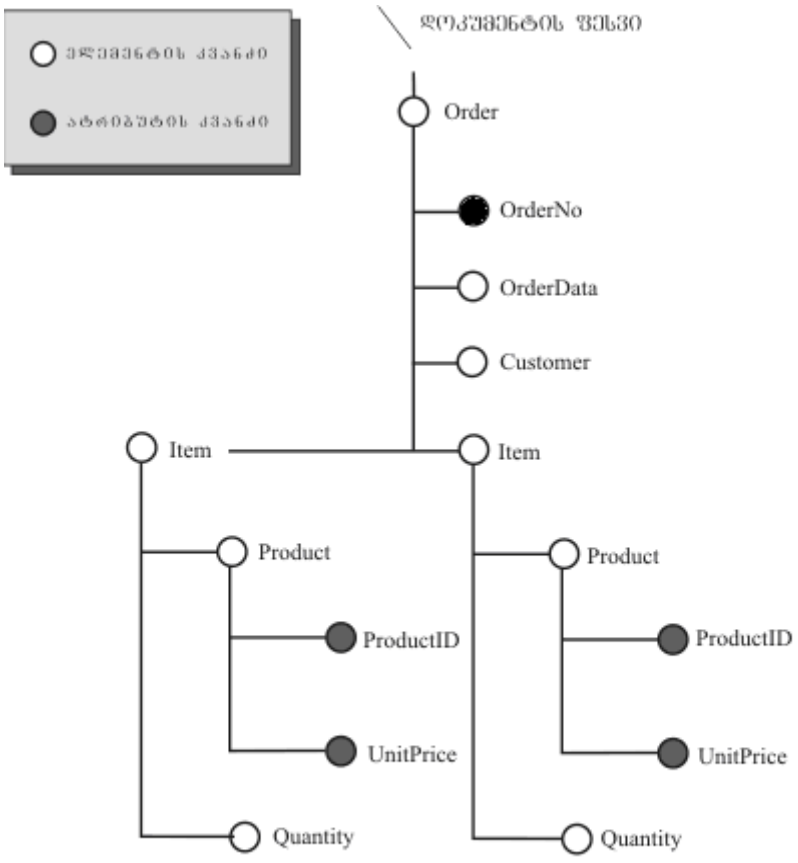
ამ თემის უფრო ღრმად შესწავლამდე საჭიროა გავეცნოთ საკითხს, თუ როგორ ხდება XML-დოკუმენტის დამუშავება.

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 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 გზების 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 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 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, გაარკვიოს, რომელია მიმდინარე კვანძი. ამ მიზნით 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, რომ 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 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 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 ტრანს-ფორმაციას 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)

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 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 სტილების 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 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) 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 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-templates 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, 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>
```

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 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.

<u>ხაზგასმით აღვნიშნავთ, რომ XML-კოდში განირჩევა დიდი და პატარა ასოები.</u>

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 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 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 გახლავთ მისი ეგზემპლარისთვის მინიმალურ და მაქსიმალურ რაოდენობათა მითითება. რაც შეეხება დოკუმენტში ატრიბუტის აუცილებლად ჩართვის უზრუნველყოფას, ეს მიიღწევა ატრიბუტის გამოცხადების AttributeType ტეგში required ატრიბუტის ჩართვით და მისთვის „Yes“ მნიშვნელობის მინიჭებით. 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 მინიჭებით. ეს მნიშვნელობა გამოყენებული იქნება მაშინ, თუ დოკუმენტში გამოვტოვებთ სქემაში განსაზღვრულ ატრიბუტს.

ელემენტთა ეგზემპლარების რიცხვის შეზღუდვა

იმისათვის, რომ დოკუმენტში გარანტირებული იყოს ელემენტის არსებობა, მისი ეგზემპლარის გამოცხადების ტეგში არააუცილებელ minOccurs ატრიბუტს მივანიჭებთ ერთის ტოლ მნიშვნელობას, ხოლო maxOccurs-სთვის 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 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 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-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>
```

XML, SQL და ინტერნეტი

ბოლო ხანებში XML გაბატონებულ ტექნოლოგიად იქცა სხვადასხვა პარტნიორებს შორის ინტერნეტის მეშვეობით ინფორმაციის გაცვლისათვის. XML მოგვევლინა ერთგვარი ესპერანტოს როლში, რომელიც უკვე არსებულ სისტემებს (გამოყენებებს) უადვილებს როგორც ერთმანეთს შორის ურთიერთობას, ასევე ახალ სისტემებთან ინტეგრაციას. ამასთან, მნიშვნელობა არა აქვს სისტემის მიერ გამოყენებულ აპარატურულ პლატფორმას.

XML-ის პოპულარობამ განაპირობა შემდეგი ტენდენცია —– მონაცემთა ბაზების მართვის სისტემებში ჩადებულ იქნეს ამ ენის გამოყენების შესაძლებლობანი, ე.ი. მიენიჭოთ მათ XML-ფუნქციონალობა.

XML-ის შესაძლებლობებს განვიხილავთ ჩვენთვის უკვე კარგად ნაცნობი SQL Server მონაცემთა ბაზის გამოყენებებს შორის კავშირის დამყარების მაგალითებზე.

დღეისათვის სულ უფრო და უფრო ხშირად გამოითქმება აზრი, რომ ინტერნეტი საწარმოებს, ორგანიზაციებს და მათ კლიენტებს შორის ურთიერთობის ძირითადი საშუალებაა ერთმანეთისთვის ე.წ. ბიზნეს-მონაცემების გადასაგზავნად. ამ მონაცემების შესანახად, როგორც წესი, იყენებენ მონაცემთა რელაციურ ბაზებს, რომელთა შორის ერთ-ერთი ყველაზე პოპულარული სისტემაა Microsoft SQL Server. მასში შენახულ ინფორმაციასთან მუშაობისთვის კი გამოიყენება მოთხოვნების სტრუქტურიზებული ენა SQL.

მონაცემები ახასიათებს რეალურ არსებს (საქონელი, მომხმარებლები). ეს არსები წარმოდგება კლასების სახით. კლასი განსაზღვრავს არსისთვის დამახასიათებელ თვისებებს. მათი მნიშვნელობების დაკონკრეტება ხდება ობიექტებში - არსთა ეგზემპლარებში. არსთა ეგზემპლარების სახით წარმოდგენილ

თვისებათა ერთობლიობის მნიშვნელობები ინახება მონაცემთა ცხრილებში.

მონაცემთა ბაზების ცხრილებს შორის მყარდება კავშირები ცხრილებში არსებული საერთო საკვანძო ველების მეშვეობით.

შედეგად მიიღება მოცემული საწარმოს (ორგანიზაციის) ლოგიკური ბიზნეს-მოდელი.

რადგანაც სხვადასხვა ორგანიზაციები სხვადასხვა ლოგიკურ ბიზნეს-მოდელებს იყენებს, დგება საკითხი - მოიძებნოს მათთვის "საერთო ენა".

სწორედ, ამ პრობლემის გადაწყვეტას ემსახურება XML-ტექნოლოგია.

თუ მონაცემთა რელაციური ბაზა ბიზნეს-მონაცემთა ეფექტიანი შენახვისათვის არის განკუთვნილი, XML-ის დანიშნულებაა ასევე ეფექტიანად უზრუნველყოს მათ შორის მონაცემების გაცვლა. ამასთან, იგი არ არის დამოკიდებული აპარატურულ პლატფორმაზე, ოპერაციულ სისტემაზე, დაპროგრამების ენაზე და ა. შ.

ბიზნეს-არსების წარმოდგენა XML-ის მეშვეობით

ვიცით, რომ რელაციურ ბაზებში არსის თვისებებს შეეთანადება ცხრილის ველები. XML-დოკუმენტებში კი ეს თვისებები შეიძლება წარმოდგეს ატრიბუტების, ელემენტების მნიშვნელობების და ჩადგმული ელემენტების სახით.

მოვიყვანოთ მარტივი რელაციური ცხრილის მაგალითი:

ცხრილი Customers

CustID	Name	Phone
1001	Graeme	555 111222
1002	Rose	555 222111

ეს ცხრილი XML-დოკუმენტის სახით შემდეგნაირად წარმოგვიდგება:

```
<Customers>
    <Customer CustID='1001' Name='Graeme' Phone='555
111222'/>
    <Customer CustID='1002' Name='Rose' Phone='555
222111'/>
</Customers>
```

აქ მონაცემთა ბაზის ცხრილს შეესაბამება Customers სახელის მქონე XML-დოკუმენტი, რომლის ორი ელემენტის

მახასიათებლები წარმოდგენილია ე.წ. ატრიბუტზე ორიენტირებულ მანერაში.

დასაშვებია ცხრილის წარმოდგენა ცენტრულ-ელემენტური (ელემენტებზე ორიენტირებული) სახითაც:

```
<Customers>
    <Customer>
        <CustID>1001</CustID>
        <Name>Graeme</Name>
        <Phone>555 111222</Phone>
     </Customer>
    <Customer>
        <CustID>1002</CustID>
        <Name>Rose</Name>
        <Phone>555 222111</Phone>
    </Customer>
</Customers>
```

ამ შემთხვევაში ცხრილის ყოველი ველი წარმოგვიდგება, როგორც ჩადგმული ელემენტი ფესვური ელემენტის მიმართ.

სავსებით დასაშვებია შერეული ვარიანტის გამოყენებაც:

```
<Customers>
    <Customer CustID='1001'>
        Graeme
        <Phone>555 111222</Phone>
    </Customer>
    <Customer CustID='1002'>
        Rose
        <Phone>555 222111</Phone>
    </Customer>
</Customers>
```

მოცემული ცხრილისთვის კონკრეტული XML-კოდის არჩევა, როგორც იტყვიან, გემოვნების საქმეა. საერთოდ კი, უნდა ითქვას, რომ ატრიბუტი ელემენტისთვის (და, შესაბამისად, ობიექტისთვის) მხოლოდ ერთი მნიშვნელობის მინიჭების საშუალებას იძლევა, ხოლო ჩაშენებული ელემენტები - რამდენიმესი (მაშინაც, როცა მათი არსებობა მხოლოდ პოტენციურადაა მოსალოდნელი).

მოვიყვანოთ მაგალითი. წინა ფრაგმენტში, ცხადია, კონკრეტულ კლიენტს მხოლოდ ერთი იდენტიფიკატორი შეიძლება გააჩნდეს, ტელეფონის ნომერი კი - რამდენიმე:

```
<Customers>
    <Customer CustID='1001'>
        Graeme
```

```
        <Phone>555 111222</Phone>
        <Phone>555 111333</Phone>
    </Customer>
    <Customer CustID='1002'>
        Rose
        <Phone>555 222111</Phone>
    </Customer>
</Customers>
```

XML-ის მეშვეობით კავშირების წარმოდგენა

რელაციური ბაზები საშუალებას იძლევა არსებს შორის წარმოდგენილ იქნეს რელაციური კავშირები. მაგალითად, არსი შეკვეთა (order) შეიძლება შეიცავდეს რამდენიმე არსს საქონელი (item):

Orders

OrderNo	Date	Customer
1235	01/01/2001	1001
1236	01/01/2001	1002

Items

ItemNo	OrderNo	ProductID	Price	Quantity
1	1235	1432	12.99	2
2	1235	1678	11.49	1
3	1236	1432	12.99	3

ყველაზე ხშირად ასეთი მონაცემების წარმოდგენა შემდეგი სახის XML-დოკუმენტის მეშვეობით ხდება:

```
<Orders>
    <Order OrderNo='1235' Date='01/01/2001'
Customer='1001'>
        <Item ProductID='1432' Price='12.99'
Quantity='2'/>
        <Item ProductID='1678' Price='11.49'
Quantity='1'/>
    </Order>
    <Order OrderNo='1236' Date='01/01/2001'
Customer='1002'>
        <Item ProductID='1432' Price='12.99'
Quantity='3'/>
    </Order>
</Orders>
```

მაგრამ, როცა გადასაცემი ინფორმაციის მოცულობა ძალიან დიდია, განმეორებადი ელემენტების გამოსარიცხად იყენებენ XML-DATA სქემებს, რომლებიც ამ მიზნით სპეციალური ტიპის ატრიბუტების გამოყენების საშუალებას იძლევა (ეს ტიპებია ID, IDREF და IDREFS). მაგალითად, დავუშვათ, მომწოდებელს უნდა გადაეცეს დოკუმენტი-კატალოგი, რომელშიც საქონელი დახარისხებულია კატეგორიების მიხედვით.

ვისარგებლოთ შემდეგი სქემით:

```
<Schema name='catalogschema'
    xmlns='urn:schemas-microsoft-com:xml-data'
    xmlns:dt='urn:schemas-microsoft-com:datatypes'>
    <ElementType name='Category' model='closed'>
        <AttributeType name='CategoryID' dt:type='id'/>
        <AttributeType name='CategoryName'
dt:type='string'/>
        <attribute type='CategoryID'/>
        <attribute type='CategoryName'/>
    </ElementType>
    <ElementType name='Product' model='closed'>
        <AttributeType name='ProductID' dt:type='i4'/>
        <AttributeType name='ProductName'
dt:type='string'/>
        <AttributeType name='Category' dt:type='idref'/>
        <attribute type='ProductID'/>
        <attribute type='ProductName'/>
        <attribute type='Category'/>
    </ElementType>
    <ElementType name='Catalog' content='eltOnly'
model='closed'>
        <element type='Category' maxOccurs='*'/>
        <element type='Product' maxOccurs='*'/>
    </ElementType>
</Schema>
```

ამ სქემაში განსაზღვრულია Category და Product ელემენტები (თავისი ატრიბუტებით), განსაზღვრულია აგრეთვე Catalog ელემენტი, რომელიც შეიძლება მოიცავდეს ამ ელემენტებს.

აღნიშნული სქემის გამოყენებით კატალოგის მონაცემები შეიძლება წარმოვადგინოთ ამგვარი XML-დოკუმენტის სახით:

```
<Catalog xmlns='x-schema:catalogschema.xml'>
    <Category CategoryID='1' CategoryName='Games'/>
    <Category CategoryID='2'
CategoryName='Educational'/>
    <Product ProductID='131' ProductName='TicTacToe'
Category='1'/>
```

```
    <Product ProductID='1432' ProductName='Chess'
Category='1'/>
    <Product ProductID='1678' ProductName='Spelling'
Category='2'/>
</Catalog>
```

რადგანაც CategoryID ატრიბუტი (Category ელემენტისთვის) სქემაში განსაზღვრულია, როგორც ID ტიპის, ხოლო Category ატრიბუტი (Product ელემენტისთვის), როგორც IDREF ტიპის, პროდუქტებს და კატეგორიებს შორის შეიძლება დამყარდეს კავშირი.

შევნიშნოთ, რომ IDREFS ტიპი შესაძლებლობას იძლევა პროდუქტი სიის სახით ჩამოთვლილ რამდენიმე კატეგორიას მივაკუთვნოთ. სიის წევრები ერთმანეთისგან წერტილ-მძიმით უნდა გამოიყოს.

შემდგომ, ჩვენ განვიხილავთ ვირტუალური Nortwind კომპანიის ბიზნეს-პროცესების გადატანის საშუალებებს WEB-სივრცეში XML-ის მეშვეობით.

კომპანია სხვა კომპანიებს აწვდის სხვადასხვა სახის პროდუქტებს მთელი მსოფლიოს მასშტაბით. მათი თანამშრომლების, პროდუქტების, კლიენტების, მათი შეკვეთებისა და ამ შეკვეთების შესრულების შესახებ ინფორმაცია ინახება SQL SERVER-ის მონაცემთა ბაზაში.

XML-მონაცემების მიღება Transact-SQL ოპერატორების მეშვეობით

ვუჩვენოთ, როგორ ხდება Nortwind ბაზიდან SELECT ოპერატორის მეშვეობით შეკვეთების შესახებ ინფორმაციის მიღება, ამ SELECT ოპერატორის ფუნქციონალობის გაფართოების შედეგად XML-ანგარიშების ფორმირება და კლიენტებისათვის მათი გადაგზავნა ინტერნეტით.

ოპერატორი SELECT... FOR XML

სტანდარტული SELECT ოპერატორისთვის FOR XML კონსტრუქციის დამატება SQL სერვერს ამცნობს, რომ შედეგები მან უნდა დააბრუნოს არა ჩანაწერების კრებულთა სახით, არამედ როგორც XML-ნაკადი. აქ დასაბრუნებელი XML-მონაცემების ფორმატის ჩვენებისათვის საჭიროა რეჟიმის მითითებაც:

RAW, AUTO ან EXPLICIT

SELECT... FOR XML ოპერატორის სინტაქსი შემდეგი სახისაა:

```
SELECT select_list
FROM table_source
WHERE search_condition
FOR XML AUTO | RAW | EXPLICIT [, XMLDATA] [,
ELEMENTS] [, BINARY BASE64]
```

შევნიშნოთ, რომ ოპერატორი აბრუნებს მხოლოდ XML-ფრაგმენტს. მისი გადაქცევა კორექტულ XML-დოკუმენტად შესაძლებელია მხოლოდ ფესვური ელემენტის დამატებით. მოვიყვანოთ XML-ფრაგმენტისა და მისი კორექტულ XML-დოკუმენტად გარდაქმნის მაგალითები:

```
<OrderItem OrderID="10248" ProductID="11" Quantity="12"/>
<OrderItem OrderID="10249" ProductID="42" Quantity="10"/>

<Invoice>
    <OrderItem OrderID="10248" ProductID="11"
Quantity="12"/>
    <OrderItem OrderID="10249" ProductID="42"
Quantity="10"/>
</Invoice>
```

ოპერატორის შესწავლა უნდა დავიწყოთ ყველაზე მარტივ რეჟიმთან გაცნობით.

RAW რეჟიმის გამოყენება

RAW რეჟიმში მოთხოვნილი ინფორმაცია გვიბრუნდება row ელემენტების კრებულის სახით. თითოეული ელემენტის ტეგში ატრიბუტთა და მათი მნიშვნელობების სახით განთავსდება დასაბრუნებელი ველებიდან ამოკრეფილი ინფორმაცია.

მოვიყვანოთ მოთხოვნისა და მისი შესრულების შედეგის მაგალითები:

```
SELECT OrderID, ProductID, UnitPrice, Quantity
FROM [Order Details]
WHERE OrderID = 10248
FOR XML RAW

<row OrderID="10248" ProductID="11" UnitPrice="14"
Quantity="12"/>
<row OrderID="10248" ProductID="42" UnitPrice="9.8"
Quantity="10"/>
```

```
<row OrderID="10248" ProductID="72" UnitPrice="34.8"
Quantity="5"/>
```

ვხედავთ, რომ მოცემულ რეჟიმში მხოლოდ ცარიელი ელემენტები გვიბრუნდება. ისინი არ შეიცავენ არც მნიშვნელობებს და არც ჩადგმულ ელემენტებს. ბაზიდან ამოკრეფილი მთელი ინფორმაცია ატრიბუტებშია განთავსებული.

გავიხსენოთ, ინფორმაციის ასეთ ორგანიზებას ჩვენ ვუწოდეთ ატრიბუტზე ორიენტირებული განლაგება.

JOIN ოპერატორის გამოყენება

ატრიბუტზე ორიენტირებული სახით გვიბრუნდება ინფორმაცია RAW რეჟიმში JOIN ოპერატორის გამოყენების შემთხვევაშიც.

მოვიყვანოთ JOIN ოპერატორის დახმარებით ისეთი ანგარიშის ფორმირების მაგალითი, რომელშიც ფიგურირებს ინფორმაცია შეკვეთის თარიღისა (Orders ცხრილიდან) და შეკვეთილი პროდუქტების (Orders Details ცხრილიდან) შესახებ.

ინფორმაცია ამოიკრიფება ზემოაღნიშნული ცხრილებიდან, რომელთა დაკავშირება ხდება JOIN ოპერატორით.

ქვემოთ მოყვანილი მოთხოვნა გვიბრუნებს შემდეგ XML-ფრაგმენტს:

```
SELECT Orders.OrderID, OrderDate, ProductID,
UnitPrice, Quantity
FROM Orders JOIN [Order Details]
ON Orders.OrderID = [Order Details].OrderID
WHERE Orders.OrderID = 10248
FOR XML RAW

<row OrderID="10248" OrderDate="1996-07-04T00:00:00"
ProductID="11"
    UnitPrice="14" Quantity="12"/>
<row OrderID="10248" OrderDate="1996-07-04T00:00:00"
ProductID="42"
    UnitPrice="9.8" Quantity="10"/>
<row OrderID="10248" OrderDate="1996-07-04T00:00:00"
ProductID="72"
    UnitPrice="34.8" Quantity="5"/>
```

ატრიბუტების სახელდება ველების ფსევდონიმებით

ცხრილებიდან წამოღებულ ინფორმაციას ავტომატურად უნარჩუნდება ველის სახელი, მაგრამ შესაძლებელია სიტუაციის შეცვლა ველებისთვის ფსევდონიმების გათვალისწინებით. RAW რეჟიმში დასაშვებია ატრიბუტებისა და გამოთვლადი ველებისთვის სახელების გადარქმევა, row ელემენტებისთვის კი სახელის შეცვლა აკრძალულია.

მოვიყვანოთ ამგვარი მოთხოვნის მაგალითი:

```
SELECT OrderID InvoiceNo,
       SUM(Quantity) TotalItems
FROM [Order Details]
WHERE OrderID = 10248
GROUP BY OrderID
FOR XML RAW
```

იგი გვიბრუნებს შემდეგ XML-ფრაგმენტს:

```
<row InvoiceNo="10248" TotalItems="27"/>
```

AUTO რეჟიმის გამოყენება

აღნიშნული რეჟიმი აფართოებს XML-კოდის მართვის შესაძლებლობებს. ამ რეჟიმში, დუმილით, row ელემენტს შესაბამისი ცხრილის სახელი ენიჭება:

```
SELECT OrderID, CustomerID
FROM Orders
WHERE OrderID = 10248
FOR XML AUTO
```

შედეგი ასე გამოისახება:

```
<Orders OrderID="10248" CustomerID="VINET"/>
```

თუ ცხრილის დასახელებაში "ჰარი" ფიგურირებს, იგი სპეცსიმბოლოებით შეიცვლება. მაგალითად:

```
SELECT OrderID, ProductID, UnitPrice, Quantity
FROM [Order Details]
WHERE OrderID = 10248
FOR XML AUTO
```

მოთხოვნა დააბრუნებს ასეთ XML-ფრაგმენტს:

```
<Order_x0020_Details OrderID="10248" ProductID="11"
UnitPrice="14"
    Quantity="12"/>
<Order_x0020_Details OrderID="10248" ProductID="42"
UnitPrice="9.8"
    Quantity="10"/>
<Order_x0020_Details OrderID="10248" ProductID="72"
UnitPrice="34.8"
    Quantity="5"/>
```

AUTO-რეჟიმში დასაშვებია ფსევდონიმების გამოყენება, როგორც ველების, ასევე ცხრილების დონეზეც. მოვიყვანოთ მაგალითი:

```
SELECT OrderID InvoiceNo,
       ProductID,
       UnitPrice Price,
       Quantity
FROM [Order Details] Item
WHERE OrderID = 10248
FOR XML AUTO
```

შედეგში, ვხედავთ, შეცვლილია როგორც ზოგიერთი ატრიბუტის, ასევე ელემენტის სახელი, ცხრილისა და ველების ფსევდონიმებიდან გამომდინარე:

```
<Item InvoiceNo="10248" ProductID="11" Price="14"
Quantity="12"/>
<Item InvoiceNo="10248" ProductID="42" Price="9.8"
Quantity="10"/>
<Item InvoiceNo="10248" ProductID="72" Price="34.8"
Quantity="5"/>
```

AUTO რეჟიმში JOIN ოპერატორის გამოყენება

მოთხოვნები AUTO რეჟიმში JOIN ოპერატორის ჩართვით სხვაგვარად ხორციელდება, ვიდრე RAW რეჟიმში. კერძოდ, მიერთებული ცხრილის ველების ერთობლიობა ფესვურ ელემენტში ჩადგმული ელემენტების სახით წარმოგვიდგება:

```
SELECT Invoice.OrderID InvoiceNo,
       OrderDate,
       ProductID,
       UnitPrice Price,
       Quantity
FROM Orders Invoice JOIN [Order Details] Item
```

```
ON Invoice.OrderID = Item.OrderID
WHERE Invoice.OrderID = 10248
FOR XML AUTO
```

მიღებული შედეგი მნიშვნელოვნად განსხვავდება RAW რეჟიმში განხორციელებული ანალოგიური მოთხოვნისაგან:

```
<Invoice InvoiceNo="10248" OrderDate="1996-07-
04T00:00:00">
    <Item ProductID="11" Price="14" Quantity="12"/>
    <Item ProductID="42" Price="9.8" Quantity="10"/>
    <Item ProductID="72" Price="34.8" Quantity="5"/>
</Invoice>
```

ELEMENTS პარამეტრის გამოყენება

ზემოთ მოყვანილ მოთხოვნას დავუმატოთ FOR XML ოპერატორში ერთადერთი პარამეტრი ELEMENTS:

```
SELECT Invoice.OrderID InvoiceNo,
       OrderDate,
       ProductID,
       UnitPrice Price,
       Quantity
FROM Orders Invoice JOIN [Order Details] Item
ON Invoice.OrderID = Item.OrderID
WHERE Invoice.OrderID = 10248
FOR XML AUTO, ELEMENTS
```

ვნახავთ, რომ მიიღება ელემენტზე ორიენტირებული XML-კოდი – ორივე ცხრილიდან ყველა ველი XML-დოკუმენტში ჩადგმული ელემენტების სახით დაგვიბრუნდება:

```
<Invoice>
    <InvoiceNo>10248</InvoiceNo>
    <OrderDate>1996-07-04T00:00:00</OrderDate>
    <Item>
        <ProductID>11</ProductID>
        <Price>14</Price>
        <Quantity>12</Quantity>
    </Item>
    <Item>
        <ProductID>42</ProductID>
        <Price>9.8</Price>
        <Quantity>10</Quantity>
    </Item>
    <Item>
```

```
        <ProductID>72</ProductID>
        <Price>34.8</Price>
        <Quantity>5</Quantity>
    </Item>
</Invoice>
```

ამრიგად, ELEMENTS პარამეტრი ზღვრულ ვარიანტს გვთავაზობს – ყოველი ველი მხოლოდ ელემენტის სახით გვიბრუნდება.

მართალია, AUTO რეჟიმი, RAW-ისგან განსხვავებით, უფრო მოქნილია, მაგრამ მისი მეშვეობით ვერ ხერხდება ველებისთვის აგრეგაციის ფუნქციების გამოყენება. ამის გამო, როცა XML-დოკუმენტში მოითხოვება, ვთქვათ, ჯამური მნიშვნელობების გამოთვლა, საჭიროა ვისარგებლოთ RAW რეჟიმით.

EXPLICIT რეჟიმის გამოყენება

EXPLICIT რეჟიმი, ადრე განხილულებთან შედარებით, უფრო რთულ სინტაქსს იყენებს, რათა უზრუნველყოს XML-კოდის მართვის გაცილებით ფართო შესაძლებლობები.

დავუშვათ, მოითხოვება შემდეგი XML-ფრაგმენტის ფორმირება:

```
<Item InvoiceNo=OrderID>ProductID</Item>
<Item InvoiceNo=OrderID>ProductID</Item>
```

სანამ შესაბამის მოთხოვნას დაწერდნენ, ჯერ აგებენ საჭირო XML-ფრაგმენტის შესაბამის ე.წ. უნივერსალურ ცხრილს, რომელშიც ფრაგმენტებში წარმოდგენილი თითოეული ელემენტისთვის გათვალისწინებულია:

Tag ველი *(XML-ფრაგმენტის ტეგის ნომერი)*

Parent ველი *(განკუთვნილია იერარქიაში ელემენტების დონის განსაზღვრისათვის)*

შემდეგ მოდის ველები: ელემენტის მნიშვნელობის შესანახად, ელემენტის ატრიბუტების მნიშვნელობების შესანახად.

ამ ველების სახელები (უნივერსალური ცხრილისთვის) შემდეგი პარამეტრული სახით ფორმირდება:

ტეგის-სახელი! ტეგის-ნომერი!
ტეგის-სახელი! ტეგის-ნომერი! ატრიბუტის-სახელი! დირექტივა

შევნიშნოთ, რომ “ტეგის სახელი” და “ტეგის ნომერი” პარამეტრების ერთობლივი გამოყენება საშუალებას იძლევა, ცალსახად მოვახდინოთ ერთსახელა ელემენტის დაკვალი-

ფიცირება, როდესაც ისინი იერარქიულად სხვადასხვა ადგილზე იმყოფებიან.

ყოველივე ზემოთქმულის გათვალისწინებით, მოცემული XML-ფრაგმენტისთვის უნივერსალური ცხრილი ამგვარი სახის იქნება:

Tag	Parent	Item!1	Item!1!InvoiceNo
1	NULL	ProductID	OrderID
1	NULL	ProductID	OrderID
...	...	...	...

შევნიშნოთ, რომ ორივე ელემენტისთვის Tag ველის მნიშვნელობა განისაზღვრება, როგორც "1"-ის ტოლი, იერარქიული დონის შესაბამისად. რადგანაც ამ ფრაგმენტის ელემენტებს მშობელი ელემენტი არ გააჩნია, Parent ველმა მიიღო NULL მნიშვნელობა.

ქვემოთ მოყვანილი Transaqt-SQL ოპერატორი აღნიშნულ ცხრილს ქმნის Order Details ცხრილზე დაყრდნობით:

```
SELECT   1 AS Tag,
NULL AS Parent,
ProductID AS [Item!1],
OrderID AS [Item!1!InvoiceNo]
FROM [Order Details]
WHERE OrderID = 10248
```

აღვნიშნოთ, რომ პირველი ორი ველის - Tag და Parent-ის მნიშვნელობები ოპერატორში პირდაპირ არის განსაზღვრული. შევნიშნოთ, რომ დასაშვებია მოთხოვნის ფორმირება AS საკვანძო სიტყვის გარეშეც:

```
SELECT   1  Tag,
 NULL Parent,
 ............
```

XML-დოკუმენტის შესაქმნელად კი საჭიროა FOR XML EXPLICIT საკვანძო სიტყვების დამატება. შედეგს ექნება სახე:

```
<Item InvoiceNo="10248">11</Item>
<Item InvoiceNo="10248">42</Item>
<Item InvoiceNo="10248">72</Item>
```

EXPLICIT რეჟიმში გამოყენებული დირექტივები

მივაქციოთ ყურადღება უნივერსალურ ცხრილში ველის სახელის მეოთხე პარამეტრს - დირექტივას. იგი უზრუნველყოფს მონაცემების წარმოდგენის მრავალფეროვნებას.

FOR XML EXPLICIT სახის მოთხოვნებში შეიძლება გამოვიყენოთ შემდეგი დირექტივები:

- element - ახდენს ველის მონაცემების კოდირებას და წარმოადგენს ველს XML-ფრაგმენტში ჩადგმული ელემენტის სახით;
- xml - იგივეს აკეთებს კოდირების გარეშე;
- hide - მონაცემთა ელემენტი ფიგურირებს მხოლოდ უნივერსალურ ცხრილში – XML-ფრაგმენტში იგი არ აისახება;
- xmltext - მიმდინარე ელემენტს უმატებს ცხრილის გარეშე, ე.წ. გადავსების ველში არსებულ XML-მონაცემებს;
- cdata - XML-ფრაგმენტში ცხრილის ველი წარმოდგება CDATA განყოფილების სახით;
- ID, IDREF და IDREFS – გამოიყენება XMLDATA პარამეტრთან ერთად ჩადგმული სქემის დასაბრუნებლად ID, IDREF ან IDREFS ტიპის ატრიბუტებით. ამ დირექტივებით იქმნება კავშირები სხვადასხვა დოკუმენტების ელემენტებს შორის.

ჩადგმული ელემენტის მიღება element და xml დირექტივებით

element დირექტივა სხვებთან შედარებით უფრო ხშირად გამოიყენება. მისი მეშვეობით ველი ჩადგმულ ელემენტად იქცევა.

ვაჩვენოთ სასურველი XML-ფრაგმენტის მისაღებად მისი გამოყენების მაგალითი. ვთქვათ, გვესაჭიროება ბაზაში არსებული ინფორმაცია შემდეგი XML-ფრაგმენტის სახით წარმოვადგინოთ:

```
<Item InvoiceNo=OrderID>
    ProductID
    <Price>UnitPrice</Price>
</Item>
<Item InvoiceNo=OrderID>
    ProductID
    <Price>UnitPrice</Price>
</Item>
```

შესაბამის უნივერსალურ ცხრილს ექნება სახე:

Tag	Parent	Item!1	Item!1!InvoiceNo	Item!1!Price!element
1	NULL	ProductID	OrderID	UnitPrice
1	NULL	ProductID	OrderID	UnitPrice
...	...	...	...	...

ამ უნივერსალური ცხრილის ბაზაზე შემდეგი Transact-SQL ოპერატორი:

```
SELECT   1 AS Tag,
NULL AS Parent,
ProductID AS [Item!1],
OrderID AS [Item!1!InvoiceNo],
UnitPrice AS [Item!1!Price!element]
FROM [Order Details]
WHERE OrderID = 10248
FOR XML EXPLICIT
```

მოგვცემს მონაცემთა ბაზიდან რაიმე კონკრეტული სახის XML-ფრაგმენტს:

```
<Item InvoiceNo="10248">
    11
    <Price>14</Price>
</Item>
<Item InvoiceNo="10248">
    42
    <Price>9.8</Price>
</Item>
<Item InvoiceNo="10248">
    72
    <Price>34.8</Price>
</Item>
```

აღვნიშნოთ, რომ დირექტივა element ახდენს ველის მონაცემების კოდირებას. მაგალითად, თუ ველის მნიშვნელობა განსაზღვრულია როგორც >5, XML-ფრაგმენტში იგი წარმოგვიდგება >5 კოდის სახით.

ამრიგად, xml და element დირექტივები საშუალებას იძლევიან, მივიღოთ ისეთი XML-ფრაგმენტები, რომლებშიც მონაცემების ნაწილი წარმოგვიდგება ატრიბუტების, ნაწილი კი ელემენტების სახით, ანუ ფრაგმენტის კონფიგურაციას ექნება ატრიბუტ-ელემენტ-ორიენტირებული სახე.

სანამ სხვა დირექტივებს გავეცნობოდეთ, შევისწავლოთ, თუ როგორ ხდება EXPLICIT რეჟიმში მონაცემების მიღება რამდენიმე ცხრილიდან.

EXPLICIT რეჟიმში მონაცემების მიღება რამდენიმე დაკავშირებული ცხრილიდან.

დავუშვათ, გვსურს XML-ფრაგმენტში ფიგურირებდეს შეკვეთილი პროდუქტების დასახელებებიც.

ვწერთ მოთხოვნას, რომელშიც კავშირდება Order Details და Products ცხრილები:

```
SELECT   1 AS Tag,
NULL AS Parent,
ProductName AS [Item!1],
OrderID AS [Item!1!InvoiceNo],
OD.UnitPrice AS [Item!1!Price!element]
FROM [Order Details] OD JOIN Products P
ON OD.ProductID = P.ProductID
WHERE OrderID = 10248
FOR XML EXPLICIT
```

შეკვეთა გვიბრუნებს შემდეგ XML-ფრაგმენტს:

```
<Item InvoiceNo="10248">
    Queso Cabrales
    <Price>14</Price>
</Item>
<Item InvoiceNo="10248">
    Singaporean Hokkien Fried Mee
    <Price>9.8</Price>
</Item>
<Item InvoiceNo="10248">
    Mozzarella di Giovanni
    <Price>34.8</Price>
</Item>
```

ამ მაგალითში Order Details ცხრილის გარე გასაღების ველი JOIN ოპერატორის მეშვეობით მონაცემებით შეიცვალა დაკავშირებული Products ცხრილიდან.

ვხედავთ, რომ ეს ოპერაცია საკმაოდ მარტივად და წინა რეჟიმების (AUTO, RAW) ანალოგიურად განხორციელდა. მაგრამ ზოგჯერ საჭიროა თითოეული პროდუქტისთვის მისი მომცველი შეკვეთის შესახებ უფრო რთული, იერარქიული სახით წარმოდგენილი სასათაურო მონაცემების მიღება.

ასეთ შემთხვევაში საჭირო ხდება კიდევ ერთი ცხრილის აგება:

```
SELECT   1 AS Tag,
NULL AS Parent,
ProductName AS [Item!1],
O.OrderID AS [Item!1!InvoiceNo],
OrderDate AS [Item!1!Date],
OD.UnitPrice AS [Item!1!Price!element]
FROM Orders O
JOIN [Order Details] OD ON O.OrderID = OD.OrderID
JOIN Products P ON OD.ProductID = P.ProductID
WHERE O.OrderID= 10248
FOR XML EXPLICIT
```

შედეგს ექნება ასეთი სახე:

```
<Item InvoiceNo="10248" Date="1996-07-04T00:00:00">
    Queso Cabrales
    <Price>14</Price>
</Item>
<Item InvoiceNo="10248" Date="1996-07-04T00:00:00">
    Singaporean Hokkien Fried Mee
    <Price>9.8</Price>
</Item>
<Item InvoiceNo="10248" Date="1996-07-04T00:00:00">
    Mozzarella di Giovanni
    <Price>34.8</Price>
</Item>
```

ვხედავთ, რომ მოთხოვნა აბრუნებს კონკრეტულ შეკვეთასთან დაკავშირებული პროდუქტების სიას. მაგრამ, უნდა აღინიშნოს, რომ აქ უხერხულობას ქმნის თითოეული პროდუქტისთვის შეკვეთის შესახებ ინფორმაციის აღმნიშვნელი ე.წ. სასათაურო მონაცემების არსებობა.

სჯობს ეს მონაცემები ერთ Invoice ელემენტში დაჯგუფდეს. გაუმჯობესებული XML-სტრუქტურა შემდეგი სახით წარმოგვიდგება:

```
<Invoice InvoiceNo="10248" Date="1996-07-
04T00:00:00">
    <Item Product="Queso Cabrales">
        <Price>14</Price>
    </Item>
    <Item Product="Singaporean Hokkien Fried Mee">
        <Price>9.8</Price>
    </Item>
    <Item Product="Mozzarella di Giovanni">
```

```
        <Price>34.8</Price>
    </Item>
</Invoice>
```

ამ XML-სტრუქტურის მონაცემებით შევსების მიზნით უნივერსალური ცხრილის აგება შედარებით რთული საქმეა. პირველ რიგში, აღვნიშნოთ, რომ საჭირო XML-ფრაგმენტში Products ცხრილს შეეთანადება <Invoice> ტეგი, ხოლო Order Details ცხრილს - <Item>. ამ ელემენტებისთვის, ცხადია, განსხვავებული მნიშვნელობები უნდა მიიღოს Tag და Parent პარამეტრებმა. მაგალითად, <Invoice> ელემენტის Tag პარამეტრმა შეიძლება მიიღოს “1”-ის ტოლი მნიშვნელობა, ხოლო <Item> ელემენტის ანალოგიურმა პარამეტრმა - “2”.

ადვილი შესამჩნევია, რომ საჭირო XML-ფრაგმენტის მისაღებად ერთ მოთხოვნაში უნდა მოხდეს ორი მოთხოვნის გაერთიანება. მოთხოვნების შედეგების გაერთიანებას ემსახურება UNION ALL საკვანძო სიტყვები, რომელთა მეშვეობით ხდება განმეორებადი ჩანაწერების ამოგდება.

უნივერსალური ცხრილი შემდეგი სახით წარმოგვიდგება:

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date	Item!2!Product	Item!2!Price!element
1	NULL	InvoiceNo	OrderDate	NULL	NULL
2	1	InvoiceNo	NULL	ProductName	UnitPrice
2	1	InvoiceNo	NULL	ProductName	UnitPrice
1	NULL	InvoiceNo	OrderDate	NULL	NULL
2	1	InvoiceNo	NULL	ProductName	UnitPrice
...	...	...	...	...	...

ჯერ განვიხილოთ პირველი მოთხოვნის აგების საკითხი, რომლის რეალიზებამ უნდა შეავსოს Invoice ელემენტები. ამ მიზნით გამოიყენება უნივერსალური ცხრილის პირველი და მეოთხე სტრიქონები (მივაქციოთ ყურადღება, რომ Product და Price ველების მნიშვნელობაა NULL).

მეორე მოთხოვნა ავსებს Item ელემენტებს, რისთვისაც გამოიყენება უნივერსალური ცხრილის დანარჩენი სტრიქონები.

შევნიშნოთ, რომ რადგანაც ხდება მოთხოვნათა შედეგების გაერთიანება, ორივე მოთხოვნაში უნდა ფიგურირებდეს ერთნაირი სვეტები. ამ სვეტების ჩვენება აუცილებელია მაშინაც კი, როცა მოთხოვნა შესაბამის შედეგებს არ აბრუნებს. მაგალითად, პირველ მოთხოვნაში საჭიროა Product და Price ველების გათვალისწინებაც (მათ ენიჭება NULL მნიშვნელობა).

საბოლოოდ, პირველ მოთხოვნას ექნება სახე:

```
SELECT   1 AS Tag,
         NULL AS Parent,
         OrderID AS [Invoice!1!InvoiceNo],

         OrderDate AS [Invoice!1!Date],
         NULL AS [Item!2!Product],
         NULL AS [Item!2!Price!element]
FROM Orders
WHERE OrderID = 10248
```

მეორე მოთხოვნა (რომელიც Item ელემენტებს ავსებს) შედარებით რთული სახისაა:

ჯერ ერთი, მითითებული უნდა იქნეს, რომ Item ელემენტის იერარქიული დონე უფრო დაბალია და იგი Invoice ელემენტის შვილობილი ელემენტი გახლავთ. ამ მიზნით, Tag პარამეტრს ვანიჭებთ “2”-ის, ხოლო Parent-ს - “1”-ის ტოლ მნიშვნელობებს.

შემდეგ, უნდა ვაჩვენოთ, რომ მონაცემების მიღება ხდება Orders, Products და Order Details ცხრილებიდან. მაშასადამე, დაგვჭირდება ცხრილების ორი შეერთება და აქაც ველების ზუსტად ის მიმდევრობა უნდა გამოვიყენოთ, რომელსაც მივმართეთ პირველი მოთხოვნის ფორმირებისას. ამიტომ მოთხოვნაში ვითვალისწინებთ OrderID სვეტსაც, რომლის მეშვეობით ერთმანეთს შეეთანადება Order და Order Details ცხრილების მნიშვნელობები, ხოლო OrderDate ველს კი ვანიჭებთ NULL მნიშვნელობას.

მეორე მოთხოვნა ასეთ სახეს მიიღებს:

```
SELECT  2,
        1,
        O.OrderID,
        NULL,
        P.ProductName,
        OD.UnitPrice
FROM Orders O JOIN [Order Details] OD
ON O.OrderID = OD.OrderID
JOIN Products P
ON OD.ProductID = P.ProductID
WHERE O.OrderID = 10248
```

მოთხოვნები უნდა გაერთიანდეს UNION ALL ოპერატორის მეშვეობით, ხოლო ORDER BY განყოფილება უზრუნველყოფს XML-ელემენტების შეთანადების სისწორეს.

საბოლოოდ, მოთხოვნას ექნება სახე:

```
SELECT  1 AS Tag,
        NULL AS Parent,
```

```
        OrderID AS [Invoice!1!InvoiceNo],
        OrderDate AS [Invoice!1!Date],
        NULL AS [Item!2!Product],
        NULL AS [Item!2!Price!element]
FROM Orders
WHERE OrderID = 10248
UNION ALL
SELECT  2,
       1,
        O.OrderID,
        NULL,
        P.ProductName,
        OD.UnitPrice
FROM Orders O JOIN [Order Details] OD
ON O.OrderID = OD.OrderID
JOIN Products P
ON OD.ProductID = P.ProductID
WHERE O.OrderID = 10248
ORDER BY [Invoice!1!InvoiceNo], [Item!2!Product]
FOR XML EXPLICIT
```

აღვნიშნოთ, რომ EXPLICIT რეჟიმში ყოველი დამატებითი ცხრილიდან ინფორმაციის ამოსაღებად საკმარისია კოდში UNION ALL ოპერატორით ახალი მოთხოვნის დამატება, რათა შევსებულ იქნეს ამ ცხრილების შესაბამისი ტეგები.

ცხადია, მანამდე უნდა ავაგოთ საჭირო უნივერსალური ცხრილი.

hide დირექტივის მეშვეობით მონაცემების დახარისხება

hide დირექტივით დაკვალიფიცირებული ველები XML-ფრაგმენტში არ აისახება. ამ, ერთი შეხედვით, უცნაურ შესაძლებლობას იყენებენ მაშინ, როცა ORDER BY-თი მოწესრიგებული მონაცემების ეკრანზე გამოყვანისას საჭიროდ არ მიაჩნიათ შედეგებში იმ ველის მნიშვნელობების ასახვა, რომლის მიხედვითაც ჩატარდა დახარისხება.

ჩვეულებრივ, მოთხოვნებში hide ოპერატორის გამოყენება საჭირო არ გახლავთ. მაგრამ, როცა ვიყენებთ UNION ALL ოპერატორს, სინტაქსი მოითხოვს, რომ ORDER BY განყოფილებაში მითითებული ველები (დახარისხება რამდენიმე ველის მნიშვნელობების მიხედვით შეიძლება ხდებოდეს) აუცილებლად ნაჩვენები იყოს SELECT ოპერატორის ამორჩევის სიაში.

ქვემოთ მოყვანილი მოთხოვნა აბრუნებს თარიღის მიხედვით დახარისხებულ, კლიენტისათვის წარსადგენ ანგარიშებს, მაგრამ თვით თარიღის ველი შედეგში არ აისახება:

```
SELECT   1 AS Tag,
NULL AS Parent,
CustomerID AS [Invoice!1!Customer],
OrderID AS [Invoice!1!InvoiceNo],
OrderDate AS [Invoice!1!Date!hide],
NULL AS [Item!2!Product],
NULL AS [Item!2!Price!element]
FROM Orders
WHERE CustomerID = 'VINET'
UNION ALL
SELECT   2,
         1,
O.CustomerID,
O.OrderID,
O.OrderDate,
P.ProductName,
OD.UnitPrice
FROM Orders O JOIN [Order Details ] OD
ON O.OrderID = OD.OrderID
JOIN Products P
ON OD.ProductID = P.ProductID
WHERE O.OrderID = 10248
ORDER BY [Invoice!1!InvoiceNo], [Item!2!Product]
FOR XML EXPLICIT
```

მართლაც, ვრწმუნდებით, რომ თარიღის მიხედვით მოწესრიგებულ ანგარიშებში თვით ეს თარიღის ველი არ ფიგურირებს:

```
<Invoice Customer="VINET" InvoiceNo="10248">
    <Item Product="Mozzarella di Giovanni">
        <Price>34.8</Price>
    </Item>
    <Item Product="Queso Cabrales">
        <Price>14</Price>
    </Item>
    <Item Product="Singaporean Hokkien Fried Mee">
        <Price>9.8</Price>
    </Item>
</Invoice>
<Invoice Customer="VINET" InvoiceNo="10274">
    <Item Product="Flotemysost">
        <Price>17.2</Price>
```

```
    </Item>
    <Item Product="Mozzarella di Giovanni">
        <Price>27.8</Price>
    </Item>
</Invoice>
```

XML-მნიშვნელობების მიღება xmltext დირექტივის მეშვეობით

ვთქვათ, კლიენტს სურს, განაახლოს თავის შესახებ ინფორმაცია NortWind კომპანიის ბაზაში. გამორიცხული არ არის, რომ მან დოკუმენტში "ზედმეტი" მონაცემებიც შეიტანოს, რომელთათვისაც ბაზაში ველები არ გაითვალისწინება. ქვემოთ მოყვანილ XML-დოკუმენტში ასეთი მონაცემები წარმოდგენილია WEB-ელემენტის სახით:

```
<Customerdetails>
    <CustomerID>AROUT</CustomerID>
    <CompanyName>Around the Horn</CompanyName>
    <ContactName>Thomas Hardy</ContactName>
    <ContactTitle>Sales Representative</ContactTitle>
    <Address>120 Hanover Sq.</Address>
    <City>London</City>
    <Region>Europe</Region>
    <PostalCode>WA1 1DP</PostalCode>
    <Country>UK</Country>
    <Phone>(171) 555-7788</Phone>
    <Fax>(171) 555-6750</Fax>
    <Web email="sales@aroundhorn.co.uk"
        site="www.aroundhorn.co.uk">
</Customerdetails>
```

კლიენტის მიერ გადმოგზავნილი ინფორმაცია ბაზაში შეინახება Customers ცხრილში. ცხრილის ე.წ. გადავსების ველში შესაძლებელია ზედმეტი მონაცემების მოთავსებაც.

შემდეგ, თუ გვესაჭიროება გადავსების ველიდან XML-მონაცემების მიღება, გამოვიყენებთ xmltext დირექტივას. ამასთან, თუ მოთხოვნის ტექსტში გავითვალისწინებთ ატრიბუტის სახელსაც, მონაცემები გადმოგვეცემა ჩადგმული ელემენტის სახით (მოცემული სახელით). წინააღმდეგ შემთხვევაში ისინი მშობელი ელემენტის ატრიბუტებად დაგვიბრუნდება.

განვიხილოთ ორივე მაგალითი. ჯერ მივუთითოთ ატრიბუტის სახელი (overflow):

```
SELECT  1 AS Tag,
        NULL AS Parent,
```

```
        companyname AS [customer!1!companyname],
        phone AS [customer!1!phone],
        overflow AS [customer!1!overflow!xmltext]
FROM Customers
WHERE CustomerID = 'AROUT'
FOR XML EXPLICIT
```

მოთხოვნა დაგვიბრუნებს შემდეგ XML-ფრაგმენტს:

```
<customer companyname="Around the Horn" phone="(171)
555-7788">
    <overflow email="sales@aroundhorn.co.uk"
        site="www.aroundhorn.co.uk"/>
</customer>
```

ახლა გამოვტოვოთ ატრიბუტის სახელი:

```
SELECT 1 AS Tag,
       NULL AS Parent,
       companyname AS [customer!1!companyname],
       phone AS [customer!1!phone],
       overflow AS [customer!1!!xmltext]
FROM Customers
WHERE CustomerID = 'AROUT'
FOR XML EXPLICIT
```

შედეგი ასეთი იქნება:

```
<customer companyname="Around the Horn"
       phone="(171) 555-7788"
    email="sales@aroundhorn.co.uk"
    site="www.aroundhorn.co.uk"/>
```

CDATA განყოფილებების მიღება
cdata დირექტივის მეშვეობით

ზოგჯერ XML-დოკუმენტები შეიცავს ისეთ მონაცემებს, რომლებიც, თავისთავად, საკვანძო სიტყვებს წარმოადგენს. იმ მიზნით, რომ ასეთი ინფორმაცია XML-ანალიზატორის მიერ აღქმული იქნეს, როგორც ჩვეულებრივი ტექსტი, ქმნიან CDATA განყოფილებებს.

CDATA განყოფილებებში განთავსებული ინფორმაცია XML-ანალიზატორის მიერ არ დამუშავდება.

ასეთი სახის ინფორმაციის ცხრილიდან CDATA განყოფილებაში გადმოსაწერად იყენებენ cdata დირექტივას.

დავიმახსოვროთ, რომ აქ ატრიბუტის სახელის მითითება დაუშვებელია.

შევქმნათ მოთხოვნა, რომელიც CDATA განყოფილებაში განათავსებს კლიენტის ტელეფონის ნომერს:

```
SELECT 1 AS Tag,
       NULL AS Parent,
       companyname AS [customer!1!companyname],
       phone AS [customer!1!!cdata]
FROM customers
WHERE CustomerID = 'AROUT'
FOR XML EXPLICIT
```

ამ მოთხოვნის შესრულების შედეგი იქნება:

```
<customer companyname="Around the Horn">
    <![CDATA[(171) 555-7788]]>
</customer>
```

ID, IDREF, IDREFS დირექტივებისა და XMLDATA პარამეტრის გამოყენება

უკვე ვიცით - ID, IDREF, IDREFS დირექტივები შეიძლება გამოყენებულ იქნეს რთული სახის მონაცემებთან მუშაობისთვის და მათი მეშვეობით შესაძლებელია დოკუმენტში აღმოიფხვრას ინფორმაციის დუბლირება. ამ მიზნის მისაღწევად საჭირო ხდება XML-დოკუმენტისთვის სქემის გამოყენება. სქემის კონცეფციას ჩვენ უკვე გავეცანით, მაგრამ მოცემულ შემთხვევაში საქმე გვექნება XML-დოკუმენტში ჩადგმულ სქემასთან. XMLDATA პარამეტრის გამოყენებით XML-დოკუმენტი, ჩადგმული სქემით, შეიძლება შეიქმნეს სამივე - RAW, AUTO და EXPLICIT - რეჟიმის მოთხოვნებით. მაგრამ ბოლო რეჟიმში, თუ ზემოთაღნიშნული დირექტივებიც იქნება გამოყენებული, განისაზღვრება XML-დოკუმენტის რელაციური ველებიც.

მაგალითად, დავუშვათ, გვჭირდება კონკრეტული კლიენტისთვის წაყენებული ანგარიშების სიის ფორმირება. ნაცვლად იმისა, რომ თითოეულ ანგარიშში მოვახდინოთ პროდუქტების შესახებ მონაცემების დუბლირება, შესაძლებელია დოკუმენტში განვათავსოთ ცალ-ცალკე სია აღნიშნული პროდუქტებისთვის და ID/IDREF კავშირით მოვახდინოთ პროდუქტების მიბმა შეკვეთებთან.

შემდეგი მოთხოვნა ეძებს ჩვენთვის საჭირო მონაცემებს:

```
SELECT 1 AS Tag,
       NULL AS Parent,
       ProductID AS [Product!1!ProductID!id],
       ProductName AS [Product!1!Name],
       NULL AS [Order!2!OrderID],
       NULL AS [Order!2!ProductNo!idref]
FROM Products
UNION ALL
SELECT 2,
       NULL,
       NULL,
       NULL,
       OrderID,
       ProductID
FROM [Order Details]
ORDER BY [Order!2!OrderID]
FOR XML EXPLICIT, XMLDATA
```

მოთხოვნის რეალიზება ასეთ შედეგს მოგვცემს:

```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-
com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="Product" content="mixed"
model="open">
        <AttributeType name="ProductID"
dt:type="id"/>
        <AttributeType name="Name" dt:type="string"/>
        <attribute type="ProductID"/>
        <attribute type="Name"/>
    </ElementType>
    <ElementType name="Order" content="mixed"
model="open">
        <AttributeType name="OrderID" dt:type="i4"/>
        <AttributeType name="ProductNo"
dt:type="idref"/>
        <attribute type="OrderID"/>
        <attribute type="ProductNo"/>
    </ElementType>
</Schema>
<Product xmlns="x-schema:#Schema1" ProductID="1"
Name="Chai"/>
<Product xmlns="x-schema:#Schema1" ProductID="2"
Name="Chang"/>
<Product xmlns="x-schema:#Schema1" ProductID="3"
Name="Aniseed Syrup"/>
```

```
<Order xmlns="x-schema:#Schema1" OrderID="10248"
ProductNo="11"/>
<Order xmlns="x-schema:#Schema1" OrderID="10248"
ProductNo="42"/>
<Order xmlns="x-schema:#Schema1" OrderID="10249"
ProductNo="72"/>
```

წარმოდგენილი XML-ფრაგმენტი შეიცავს ჩადგმულ სქემას, რომელიც განსაზღვრავს დოკუმენტის ელემენტებსა და ატრიბუტებს. იმ ველებს, რომლებისთვისაც EXPLICIT რეჟიმში განხორციელებულ მოთხოვნაში მითითებულია ID და IDREF დირექტივები, ენიშნებათ ID და IDREF-მონაცემთა XML-ტიპები.

ეს ველები კავშირს ამყარებენ ProductId ატრიბუტსა და Product ელემენტს, ასევე - ProductNo ატრიბუტსა და Order ელემენტს შორის.

XML-მონაცემებთან შეღწევა ADO-ს მეშვეობით

ჩვენ განვიხილეთ SQL Server ბაზებიდან XML-ფრაგმენტების სახით მონაცემების მიღების საკითხი. მაგრამ ბიზნეს-გამოყენებების შესაქმნელად საჭიროა ვისწავლოთ, თუ როგორ ხდება სერვერთან მიერთება, მისთვის FOR XML მოთხოვნების გადაცემა და შედეგების დაბრუნება.

აღნიშნული მიზნის განსახორციელებლად ძალიან ხშირად იყენებენ ქვემოთ აღწერილ გზას, რომელსაც Microsoft Active Data Object ანუ შემოკლებით ADO-მიდგომას უწოდებენ. ADO წარმოადგენს იმ კომპონენტების ბიბლიოთეკას, რომლებიც აგებულია API OLE DB სტანდარტული ინტერფეისის ბაზაზე, იგი ეყრდნობა COM ტექნოლოგიას და იძლევა ნებისმიერ წყაროში არსებულ ნებისმიერ მონაცემებთან შეღწევის საშუალებას. (SQL Server უზრუნველყოფილია ADO რომელიმე ვერსიით).

ADO-ს ბიბლიოთეკა მონაცემებთან შესაღწევად 5 ძირითად ობიექტს გვთავაზობს. ესენია:

- Connection,
- Command,
- Recordset,
- Record,
- Stream.

მონაცემთა ბაზებთან მომუშავენი უფრო ხშირად პირველ სამ ტიპს მიმართავენ.

უმეტეს შემთხვევაში Connection ობიექტი, რომელიც მონაცემების წყაროსთან ქსელურ შეერთებას უზრუნველყოფს, უშუალოდ არ გამოიყენება - ADO მას არაცხადად ქმნის Command, Recordset და Record ობიექტების მეშვეობით მონაცემისადმი მიმართვისას.

რაც შეეხება Command ობიექტს, რომლითაც მონაცემების წყაროსთან მუშაობა წარმოებს, იგი ყოველთვის იქმნება (ცხადად ან არაცხადად) მონაცემებთან ADO-ს მეშვეობით მიმართვისას. დასაბრუნებელი მონაცემები მიეთითება მოთხოვნასა ან ბრძანებაში. რელაციურ ბაზებში მონაცემებისადმი მიმართვა შეიძლება განხორციელდეს Transact-SQL ოპერატორების ან შენახვადი პროცედურების მეშვეობით. Command ობიექტს უმეტესწილად უშუალოდ ქმნიან, თუმცა შესაძლებელია მისი არაცხადი სახით გამოყენებაც Connection, Recordset ან Record ობიექტებისთვის.

ხაზი გავუსვათ, რომ ობიექტი Recordset მუშაობას წარმართავს მონაცემთა კრებულებთან, ხოლო Record გვიბრუნებს ველების მხოლოდ ერთ რიგს. (ეს შეიძლება იყოს მონაცემთა ბაზიდან წამოღებული ველების მნიშვნელობების ერთობლიობა, ფაილური სისტემის რომელიმე საქაღალდის შემცველობა ან რაიმე სხვა სახის მონაცემთა კრებული).

Stream ობიექტი წარმოადგენს ტექსტური ან ორობითი მონაცემების ნაკადს. მაგალითად, ეს შეიძლება იყოს იმ ფაილის შემცველობა, რომელსაც ეყრდნობა Record ობიექტი.

იმ შემთხვევაში, როცა გვსურს ADO-ს მეშვეობით FOR XML მოთხოვნა განვახორციელოთ, ამის შესახებ უნდა ვამცნოთ SQL SERVER-ს და გადავცეთ მას მოთხოვნა XML-შაბლონ-დოკუმენტის სახით.

შევნიშნოთ, რომ ყოველი შაბლონი ეყრდნობა Microsoft XML-SQL სახელთა სივრცეს. ამასთან, მოთხოვნის ამსახველი ფესვური ელემენტი გვიბრუნდება XML-დოკუმენტის ფესვურ ელემენტად.

ვაჩვენოთ FOR XML მოთხოვნის მაგალითი. XML-SQL სახელთა სივრცეზე მიუთითებს <sql:query> ტეგი:

```
<Invoice xmlns:sql='urn:schemas-microsoft-com:xml-
sql'>
    <sql:query>
        SELECT SalesRecord.OrderID InvoiceNo,
               SalesRecord.OrderDate,
               LineItem.ProductID,
               LineItem.UnitPrice,
               LineItem.Quantity
```

```
        FROM Orders SalesRecord
        JOIN [Order Details] LineItem
        ON SalesRecord.OrderID = LineItem.OrderID
        WHERE SalesRecord.OrderID = 10248
        FOR XML AUTO
    </sql:query>
</Invoice>
```

მოთხოვნის შესრულების შედეგად მიიღება შემდეგი XML–დოკუმენტი:

```
<?xml version='1.0'?>
<Invoice xmlns:sql="urn:schemas-microsoft-com:xml-
sql">
    <SalesRecord InvoiceNo="10248" OrderDate="1996-
07-04T00:00:00">
        <LineItem ProductID="11" UnitPrice="14"
Quantity="12"/>
        <LineItem ProductID="42" UnitPrice="9.8"
Quantity="10"/>
        <LineItem ProductID="72" UnitPrice="34.8"
Quantity="5"/>
    </SalesRecord>
</Invoice>
```

აღვნიშნოთ, რომ ერთ შაბლონში შეიძლება გაერთიანდეს რამდენიმე მოთხოვნა, რომელთა შესრულების შედეგები განლაგდება ფესვურ ელემენტში.

მაგალითად, შემდეგი შაბლონი გვიბრუნებს მოცემულ შეკვეთაში მოთხოვნილი საქონლის სიას და მათ ჯამურ ღირებულებას.

```
<Invoice xmlns:sql='urn:schemas-microsoft-com:xml-
sql'>
    <sql:query>
        SELECT SalesRecord.OrderID InvoiceNo,
               SalesRecord.OrderDate,
               LineItem.ProductID,
               LineItem.UnitPrice,
               LineItem.Quantity
        FROM Orders SalesRecord
        JOIN [Order Details] LineItem
        ON SalesRecord.OrderID = LineItem.OrderID
        WHERE SalesRecord.OrderID = 10248
        FOR XML AUTO
    </sql:query>
    <sql:query>
```

```
        SELECT Sum(UnitPrice) TotalPrice
        FROM [Order Details]
       WHERE OrderID = 10248
        FOR XML RAW
    </sql:query>
</Invoice>
```

ორივე მოთხოვნის შედეგები გაერთიანებულია ერთ XML–დოკუმენტში:

```
<?xml version='1.0'?>
<Invoice xmlns:sql="urn:schemas-microsoft-com:xml-
sql">
    <SalesRecord InvoiceNo="10248" OrderDate="1996-
07-04T00:00:00">
        <LineItem ProductID="11" UnitPrice="14"
Quantity="12"/>
        <LineItem ProductID="42" UnitPrice="9.8"
Quantity="10"/>
        <LineItem ProductID="72" UnitPrice="34.8"
Quantity="5"/>
    </SalesRecord>
    <row TotalPrice="58.6"/>
</Invoice>
```

შევნიშნოთ, რომ რადგანაც მეორე მოთხოვნა შედგენილ მნიშვნელობას გამოთვლის, აქ დაუშვებელია AUTO რეჟიმის გამოყენება.

შენახვადი პროცედურების გამოყენება

ნაცვლად FOR XML მოთხოვნის უშუალოდ მოთხოვნის შაბლონში ჩართვისა, შესაძლებელია მონაცემთა ბაზაში განვათავსოთ ამ FOR XML მოთხოვნის შემცველი პროცედურა.

მართლაც, შესაძლებელია პირველი მოთხოვნა შემდეგი პროცედურის სახით წარმოვადგინოთ:

```
CREATE PROC GetInvoice @orderno int
AS
SELECT SalesRecord.OrderID InvoiceNo,
       SalesRecord.OrderDate,
       LineItem.ProductID,
       LineItem.UnitPrice,
       LineItem.Quantity
FROM Orders SalesRecord
JOIN [Order Details] LineItem
```

```
ON SalesRecord.OrderID = LineItem.OrderID
WHERE SalesRecord.OrderID = @orderno
FOR XML AUTO
```

შენახვად პროცედურას შემდეგი სახით გამოვიძახებთ მოცემული ანგარიშის შესახებ ინფორმაციის მისაღებად:

```
<Invoice xmlns:sql='urn:schemas-microsoft-com:xml-
sql'>
    <sql:query>EXEC GetInvoice 10248</sql:query>
</Invoice>
```

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.

3. Ребекка М. Риордан. Программирование в SQL Server 2000. Шаг за шагом. Москва, 2002.

შინაარსი

XML