Универзитет у Београду
Математички факултет

Никола Станковић

Поређење SQL/XML и XQuery упитних језика за рад са XML типом података у РСУБП DB2

Мастер рад

Ментор: проф. др Ненад Митић

Београд,
2017.

Ментор:

проф. др Ненад Митић
Математички факултет
Универзитет у Београду

Чланови комисије:

проф. др Гордана Павловић Лажетић
Математички факултет
Универзитет у Београду

проф. др Саша Малков
Математички факултет
Универзитет у Београду

Датум одбране:

Садржај

Глава 1

Увод

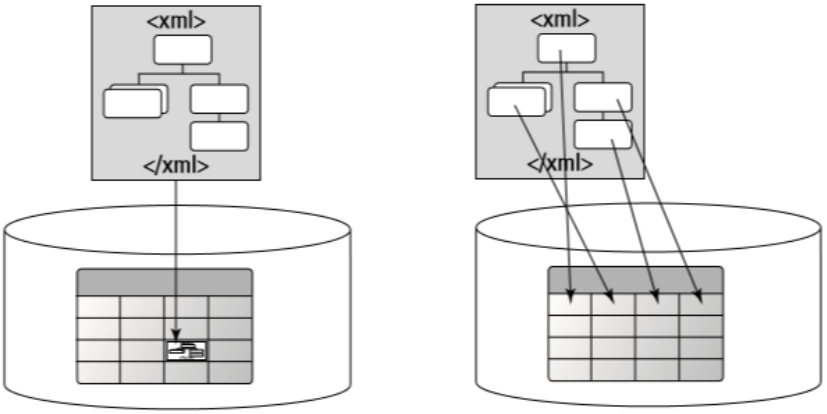
XML, the e**X**tensible **M**arkup **L**anguage, је стандардан формат за размену информација између различитих система, апликација и организација. Појавио се пре тачно две деценије и постао је један од најчешће коришћених формата за чување и размену података. Организације имају стандардизоване XML схеме (енг. XML schemas) које им помажу да промовишу и поједноставе размену података. Такође, дају им могућност да и даље развијају дате схеме и да прате промене у пословним потребама организације.

XML користи ознаке (енг. tags) да дефинише елементе и атрибуте који садрже податке. Ознаке служе да опишу значење елемената података док њихово угњеждење описује хијерархијске односе између елемената података. Из тог разлога, XML је формат података код којег су подаци и метаподаци (енг. metadata) уско повезани. Такође, у случају потребе, нове ознаке се могу лако додати. Ова проширивост дозвољава XML-у да се лако прилагоди сталном развијању пословних потреба.

Разне организације које су одлучиле да прихвате XML како би лакше размењивале информације, имале су потребу како за брзим складиштењем велике количине података тако и за брзим добијањем жељених података. На ефикасно решење које би задовољило тражене услове морало се чекати неко време.

Једно од првих решења за складиштење података било је чување докумената са XML подацима. Наиме, ако сваки слог (енг. record) сачувамо као посебан XML документ, у случају да затреба информација из слога мора се претражити садржај сваког документа посебно. Са све већом количином података од овог приступа се одустало из очигледних разлога.

Временом јавили су се додатни захтеви као што су на пример потреба за заштитом података, опоравком као и за омогућавањем паралелности у раду. Као логично решење указале су се релационе базе података које су већ поседовале све тражене функционалности [1].



Слика 1.1: Пуњење и сецкање

Пре него што је IBM представио решење за директно складиштење и обраду XML података у оквиру релационих база података чување комплетних XML података у једном пољу релационе базе података се показало као најједноставније решење. Овај процес је још познат и под појмом пуњење (енг. stuffing). С обзиром да могу да се користе све могућности које пружају релационе базе података овај приступ остварује много боље перформансе али и даље остаје проблем преузимања и претраге информација које се налазе унутар XML података. Да би се поправиле перформансе преузимања тражених информација из XML података дошло се до идеје пресликавања XML података у табеле релационих база података. Овај процес је познат и под појмом сецкања (енг. shredding). Овакав приступ омогућава доста бржу претрагу и преузимање специфичних информација које се налазе у склопу XML података. Због процеса разлагања (енг. decomposing) XML података и њиховог смештања у поља релационих база података сада је потребно више времена за уношење XML података у базу података али то је цена која је свесно плаћена зарад бржег добијања само делова XML података који су потребни у датом тренутку.

1.1 Анатомија XML документа

На следећем примеру биће објашњени најважнији делови XML документа:

```
<?xml version="1.0" encoding="UTF-8" ?>
<zaposleni xmlns="http://posample.org" id="1005">
        <ime>Aleksandar Petrovic</ime>
        <adresa drzava="Srbija">
                <grad>Beograd</grad>
                <ulica>Ruzveltova 31</ulica>
        </adresa>
        <telefon tip="poslovni">060-425-443</telefon>
        <telefon tip="privatni">064-546-6121</telefon>
<!-- ovo je komentar -->
</zaposleni>
```

Пример 1.1: Пример XML документа

Прва линија XML документа садржи опциону XML декларацију која у овом случају показује да документ користи XML 1.0 стандард, који се и најчешће користи. Такође, у првој линији документа се, поред стандарда, може декларисати и тип кодирања (UTF-8 у примеру 1.1). XML документ се састоји од елемената и њихових атрибута. Сваки елемент се састоји од почетне ознаке (енг. start tag) и завршне ознаке (енг. end tag). Веома је важно да свака почетна ознака неког елемента има и одговарајућу завршну ознаку. Пример почетне и завршне ознаке у склопу XML документа може се видети у трећој линији претходног примера и ознаке, као такве, дефинишу један XML елемент (елемент **име**). Скуп карактера који се налази између почетне и завршне ознаке представља вредност или садржај елемента. Елемент може садржати неколико других елемената. У примеру, елемент **адреса** састоји се од елемената **град** и **улица**. Угњеждење елемената помаже да се уочи однос између елемената. Такође, елементи се могу појавити и више од једног пута као што је у примеру 1.1 случај са елементом **телефон**. Елементи могу садржати један или више атрибута унутар почетне ознаке и њихова функција је да дају додатне информације о елементу. Атрибути се састоје од имена атрибута, знака једнакости и вредности атрибута који се наводи између наводника. У примеру, елемент **телефон** садржи атрибут **тип** који даје информације о томе да ли је дати број телефона пословни или приватни. Да би XML документ био добро обликован (енг. well-formed) мора поседовати један

основни елемент (енг. root element). Основни елемент садржи све остале елементе у XML документу. У претходном примеру основни елемент је **запослени** који такође поседује неколико атрибута. Атрибут **xmlns** је резервисани атрибут и служи за декларисање именског простора (енг. namespace). Имена XML елемената и атрибута могу се састојати од слова, цифара и одређених специјалних карактера (на пример, доња црта) док имена ознака не смеју да садрже бланко карактере, почињу са цифром, интерпункцијским карактерима или са речју xml. Такође је битно и у којем редоследу се елементи појављују у XML документу док редослед атрибута у почетној ознаци елемента није битан.

Глава 2

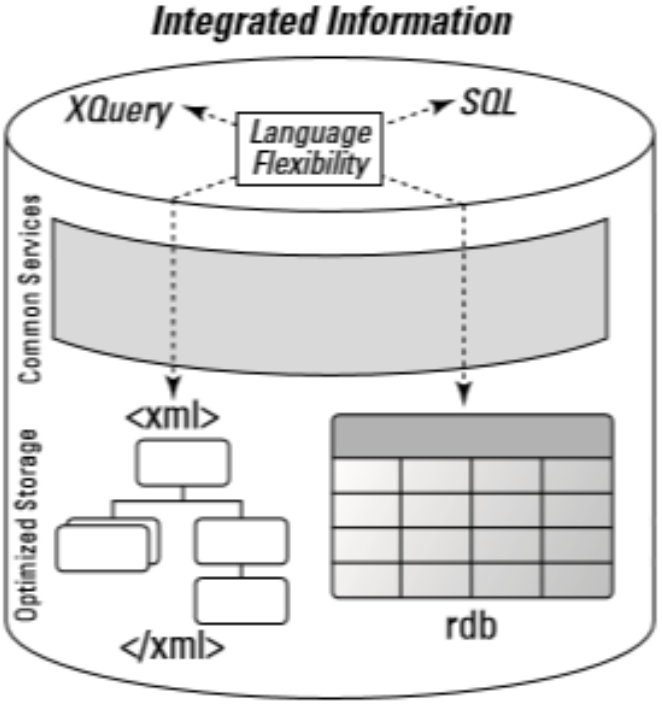
DB2 pureXML

Иницијални приступи за складиштење XML података била су више привремена решења користећи већ постојећу инфраструктуру. Оно што је било потребно је инфраструктура која је уско повезана са XML подацима, која има могућност да директно складишти XML податке и омогући лакши рад са њима. Овај захтев су испуниле XML базе података. Оне се могу поделити у две категорије:

- **XML-only базе података** су базе података које складиште само XML податке. Погодне су када се ради искључиво са XML подацима и не користи се паралелно други тип информација. Међутим, овакав приступ постаје све ређи из разлога што организације не желе да се ограниче на коришћење само XML података већ им је циљ да користе што већи број различитих типова података.
- **Релационе базе података које подржавају директно складиштење XML података**. Овакво решење нуди побољшане перформансе и сматра се знатно квалитетнијим.

DB2 од верзије 9 уводи могућност за директно складиштење и обраду XML података. Ово IBM-ово решење назива се pureXML и у њему се XML подаци посматрају као посебан структуиран тип који садржи колекцију XML елемената и атрибута и као такви се чувају у бази. Такво решење омогућава брзо уписивање у базу података као и брзо добијање тражених информација. Овакав приступ се доста разликује од решења других произвођача база података који нуде могућност рада са XML подацима. Неки од њих складиште XML податке као низ карактера (енг. string) а

неки као бинарне објекте. Код таквих решења, када се XML не чува у оригиналном формату, највише трпе перформансе. Из тог разлога IBM се фокусирао на развијање решења које ће подржавати XML у свом изворном облику. Због таквих захтева, IBM није покушавао да постојећу инфраструктуру релационих базе података прилагоди XML-у већ је развијао потпуно нову могућност за рад са XML-ом која ће радити заједно са постојећом инфраструктуром.



Слика 2.1: pureXML

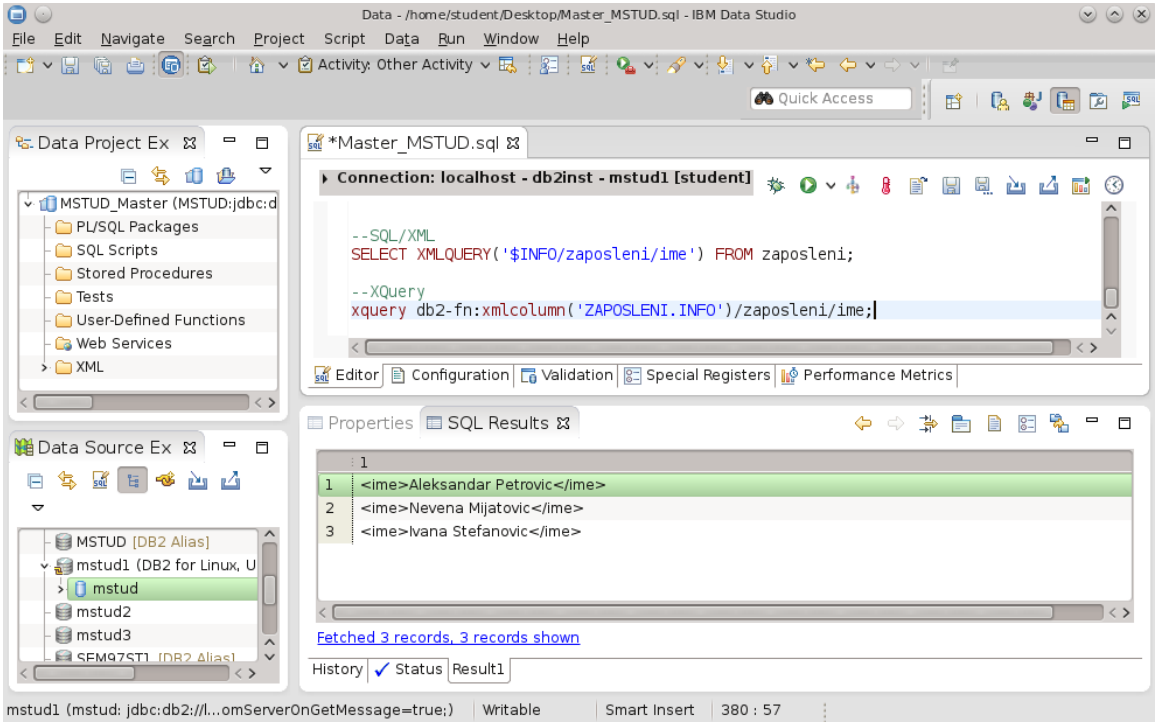
На слици 2.1 је приказано IBM-ово решење које на најнижем нивоу чине складиште оптимизовано за рад са XML подацима као и потпуно независно складиште оптимизовано за рад са релационим подацима. На нивоу изнад њих постоји заједнички сервис који омогућава језичку флексибилност приликом коришћења коначног производа. Другим речима, можемо користити SQL за приступ подацима као и XQuery[1] уколико желимо да извучемо одређене податке из базе. Пошто код оваквог приступа није потребно вршити никакве трансформације XML података приликом њиховог уношења у базу података овај корак је брз и олакшава живот администраторима и члановима развојног тима. Друга, и највећа, предност је брзина којом се извлаче информације из базе података. Пошто се XML подаци чувају у изворном облику нема потребе за парсирањем података из базе што знатно убрзава цео процес. Званичан

[1]XQuery - програмаски језик за који се често каже да за XML податке представља оно што језик SQL представља за релационе базе података. Спада у групу функционалних програмских језика.

IBM-ов интерфејс за рад са стандардним SQL изразима као и са XML подацима је Data Studio.

Језгро pureXML-а је тип података XML који је у стандарду SQL:2003 додат међу доступне типове података. Корисници базе података могу да формирају табеле које садрже један или више атрибута XML типа, а свака вредност тог атрибута може да буде добро дефинисан XML документ или NULL. Табела која садржи један или више атрибута XML типа може садржати и атрибуте осталих типова као што су INTEGER, VARCHAR, DATE или неки други тип. Такође, могуће је формирати табелу која садржи само један атрибут XML типа.

DB2 не чува XML податке као текст типа LOB и не конвертује унети XML у релациони формат већ се XML документа чувају у парсираном хијерархијском формату. Сваки XML документ се парсира само једном, оног тренутка када се први пут уноси у атрибут табеле XML типа. Максимална величина XML документа је 2GB. Стандардне SQL наредбе као што су *insert*, *delete* и *update* могу се применити искључиво на целе XML документе. Такође, као и стандардни релациони подаци и XML подаци се налазе у склопу *backup*, *restore* и *recovery* операција. Што се тиче услужних програма базе података (енг. database utilities) као што су LOAD, UNLOAD, IMPORT, EXPORT и остали, сви они подржавају и XML податке.



Слика 2.2: Data Studio - интегрисано окружење

PureXML не зависи од XML схеме тј. коришћење XML схеме је опционо. XML схема се може користити као нека врста ограничења за XML документа али у DB2 није обавезно да постоји уколико желимо да формирамо атрибут табеле XML типа или да унесемо неки XML документ. Уколико ипак желимо да потврдимо исправност XML документа онда се XML схема користи на нивоу документа а не на нивоу атрибута. DB2 не захтева да сва документа из атрибута табела XML типа припадају истој XML схеми али се уз помоћ окидача (енг. triggers) и то може омогућити. Треба напоменути да перформансе неће трпети уколико у DB2 уносимо XML документа без провере исправности.

Да бисмо извршили упите над XML подацима користимо XPath или SQL/XML док у DB2 можемо користити и XQuery. SQL/XML стандард дозвољава да XPath и XQuery изрази буду у склопу SQL наредби тако да и XML и релациони подаци могу бити претражени заједно у једном упиту. Такође, могућа су и спајања на основу XML атрибута као и спајања између релационих и XML атрибута.

Постоји SQL/XML функција XMLTABLE која се може користити да претражује XML податке и враћа вредност у релационом формату. Такође, постоје и SQL/XML функције које раде супротно тј. претражују податке у релационим табелама и формирају XML документ са вредностима резултата тог упита. У наставку рада биће детаљније објашњене неке од наведених функција које DB2 подржава [2]:

- агрегатне функције (LISTAGG, XMLAGG, XMLGROUP, AVG, COUNT...)
- скаларне функције (LTRIM, NVL, SUBSTR, XMLATTRIBUTES, XMLCONCAT, XMLQUERY, XMLELEMENT, XSLTRANSFORM...)
- табеларне функције (XMLTABLE)
- кориснички дефинисане функције

Слично као и код релационих података, а да би се постигле добре перформансе приликом претраживања XML података, DB2 дозвољава да се формирају XML индекси (енг. indexes) на одређеним XML елементима и атрибутима. Индексирају се они елементи и атрибути који се често користе као услови за спајање као и они над којима се често врше упити. Уколико упит претражује и релационе и XML податке, DB2 користи комбинацију релационих и XML индекса да би проценио тај упит и изабрао најјефтиније решење.

2.1 Операције над XML подацима

Најједноставније операције које се могу радити над XML подацима су операције које укључују рад са целим XML документом. То могу бити уметање (енг. insert), брисање (енг. delete) и издвајање (енг. retrieval) целих XML докумената. Приликом уметања врши се конверзија XML документа у интерни DB2 XML формат док се приликом издвајања врши конверзија у супротном смеру. Ова конверзија, у оба смера, захтева од DB2 да води рачуна о XML заглављима, белинама (енг. whitespace) и специјалним карактерима. На срећу крајњег корисника, DB2 ово аутоматски извршава али корисник ипак има могућност да прилагоди понашање приликом конверзије сопственим потребама.

Додавање XML документа у атрибут табеле XML типа релационе базе података врши се помоћу SQL наредбе *insert*. Такође, могу се још користити *load* и *import* услужне алатке. У случају да документ већ постоји у табели, могуће је изменити га коришћењем SQL наредбе *update*. Систем за управљање релационим базама података DB2 подржава само додавање добро дефинисаног XML документа у атрибут табеле XML типа. Добро дефинисан XML документ је документ који синтаксно испуњава све услове а о томе је више било речи у одељку 1.1. Улогу да провери да ли је приликом додавања у табелу XML документ добро дефинисан, има XML рашчлањивач (енг. parser) кога РСУБП DB2 позива приликом извршавања *insert* наредбе. У случају да XML документ није добро дефинисан, рашчлањивач га не може успешно обрадити и DB2 ће одбацити такав документ. Уколико постоји потреба да се у РСУБП DB2 ипак убаци XML документ који није добро дефинисан, то се може урадити додавањем тог документа у табелу која има атрибут табеле типа CLOB или типа VARCHAR. Приликом сваког додавања, учитавања или ажурирања XML документа може се потврдити (енг. validate) исправност документа коришћењем XML схеме али треба нагласити да је та провера опциона.

Основни језик за претраживање XML података је XPath који је подскуп XQuery језика. XPath омогућава основне фукнције за пролажење кроз XML документ као и за добијање вредности података из XML документа. Добро разумевање XPath-а и његовог модела података је најважнија ствар за претраживање XML података у РСУБП DB2. XQuery додаје XPath језику додатне изразе и језичке конструкције као и могућност да подржава формирање сложенијих упита за претраживање XML података. XQuery и XPath су стандардизовани језици од стране WWW Конзорцијума

(World Wide Web Consortium, W3C) и стандард SQL:2006 укључује функције које омогућавају коришћење наведених језика у оквиру SQL наредби.

Коришћење XPath и XQuery језика није једини начин да се извршавају упити над XML подацима. То могу бити и:

- чист SQL
- SQL/XML: SQL са уграђеним XPath језиком
- SQL/XML: SQL са уграђеним XQuery језиком
- самосталан XQuery језик
- XQuery са уграђеним SQL језиком

Чист SQL је користан уколико је потребно вратити резултате на нивоу целог документа као и код операција као што су додавање, брисање или ажурирање целог документа. Издвајање подскупа докумената мора бити извршена по неком не-XML атрибуту из исте табеле.

SQL наредбе са уграђеним XPath језиком омогућавају веома широку функционалност. Могу се користити за претрагу целог документа, ефикасно извршити агрегацију (енг. aggregation) и груписање (енг. grouping) или за резултат вратити само сегмент документа. Такође, дозвољено је спајање на основу XML атрибута или спајање преко XML и релационих података. Већини апликација је овај приступ довољно добар.

SQL наредбе са уграђеним XQuery језиком нуде највећу функционалност због свог богатства које XQuery језик има односу на XPath језик. XQuery омогућава напредан концепт као што је директно формирање XML елемената и коришћење условних израза (енг. conditional expressions). Ипак треба имати у виду да XPath може надокнадити одређене XQuery функционалности, које му недостају, уз помоћ SQL функционалности. На пример, *if-then-else* изрази које поседује XQuery могу се заменити са SQL-овим *case* изразима тако да се добију исти резултати.

Самосталан XQuery језик је добра опција уколико је обрада и извршавање упита над XML подацима све што је потребно. Уколико, на пример, мигрирамо податке из XML-only базе података у релациону базу података DB2 препоручљиво је наставити са коришћењем самосталног XQuery језика.

XQuery са уграђеним SQL језиком је добар избор уколико се жели искористити предност коју омогућавају индекси. Такође, XQuery са уграђеним SQL језиком може покретати спољашње (енг. external) и кориснички дефинисане функције (енг. user-defined function) над XML подацима. Ипак, упити са груписањем и агрегацијом су обично ефикаснији у SQL/XML [3].

2.2 Разлика XML података и релационих података

Да би упоредили податке из XML документа и податке из релационе базе података биће узет једноставан пример приказан на слици 2.3. Нека се табела састоји од шест атрибута где сваки од атрибута има унапред дефинисан тип података. Оваква табела има строго дефинисану структуру где свака врста има идентичан формат тако да није могуће да једна врста у табели има више атрибута од неке друге врсте. Такође, сваки атрибут дефинисан је искључиво са једним типом података и није могуће да има више типова података или да буде без типа. Овако строга правила нам дају доста информација о подацима који ће се налазити у табели и омогућавају веома брз приступ тим подацима.

Подаци из XML документа на десној страни слике 2.3 су идентични релационим подацима из табеле. Претпоставимо да постоји нови запослени чије податке (име, град, улица, итд.) је потребно додати у табелу релационе базе података. Уколико се међу његовим подацима налази и податак о атрибуту **улица**, дужине од на пример 57 карактера, убацивање таквог податка у овако дефинисану табелу релационе базе података није могуће јер је овај атрибут унапред дефинисан као тип података са максималном дужином од 50 карактера. Због тога, процес креирања новог запосленог у табели релационе базе података ће се завршити грешком.

Због флексибилности XML схеме, податак са називом улице од 57 карактера може бити обрађен без грешака. Ова флексибилност је пожељна у смислу да дозвољава тренутно чување података док проблем може представљати то што нема никакве контроле колику дужину може имати вредност елемента **улица**. Опционо, може се изабрати таква XML схема да се XML документ формира по строгим правилима, идентичним као релациона табела са претходне слике 2.3. Тако да на кориснику остаје да у зависности од потребе апликације изабере флексибилност XML схеме.

```
CREATE TABLE zaposleni(
    id INTEGER,
    ime VARCHAR(40),
    grad VARCHAR(30),
    ulica VARCHAR(50),
    email VARCHAR(50),
    telefon VARCHAR(20));
```

```
<zaposleni id="1003">
  <ime>Dragan Milosevic</ime>
  <adresa>
      <grad>Beograd</grad>
      <ulica>Beogradska 25</ulica>
  </adresa>
  <telefon>064-325-223</telefon>
</zaposleni>
```

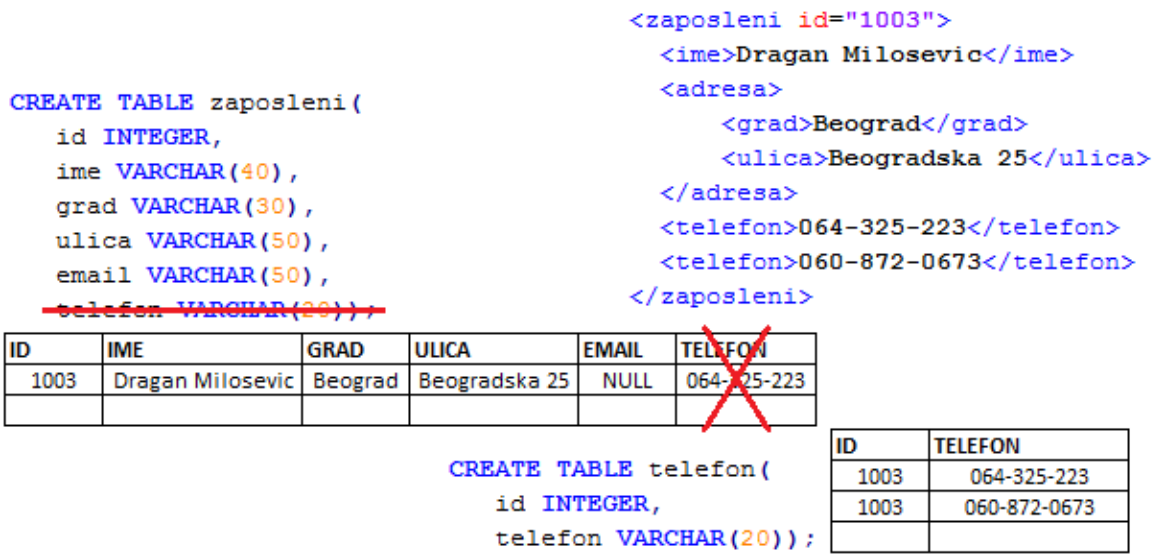
ID	IME	GRAD	ULICA	EMAIL	TELEFON
1003	Dragan Milosevic	Beograd	Beogradska 25	NULL	064-325-223

Слика 2.3: Релациона табела (лево) и XML документ (десно)

У релационој табели на слици 2.3 можемо приметити да је вредност за атрибут табеле **email** постављена на **NULL** док је у XML документу елемент **email** једноставно изостављен уколико запослени не поседује електронску пошту. Овакви, опциони, елементи у XML документу су још један вид флексибилности. Уколико у склопу података о запосленом добијемо и име његовог асистента нећемо имати могућност да ту информацију сачувамо у горе дефинисаној табели релационе базе података док уз опциони елемент **асистент** XML схеме та информација ће бити сачувана у склопу XML документа.

Следећи вид флексибилности може се видети на примеру када неко од запослених поседује два или више телефона (слика 2.4). У XML документу се то једноставно разрешава појављивањем елемента **телефон** два пута за тог запосленог док је у том случају у релационој бази података потребна нормализација табеле **запослени** што представља значајну промену схеме.

Флексибилност XML-а има за последицу да за претраживање XML података може бити утрошено више времена него за претраживање уколико би ти исти подаци били сачувани у релационој форми. Разлог за то је што схема није увек фиксна и информације о структури XML података се добијају тек приликом покретања. Код релационог модела података ситуација је другачија, флексибилност је доста ограничена али то је цена која мора да се плати зарад веома високих перформанси које се постижу захваљујући строгој и унапред дефинисаној схеми [4].



Слика 2.4: Промена схеме, релациона табела (лево) и XML документ (десно)

Најзначајније карактеристике као и разлике између релационих и XML података су следеће:

- Релациони подаци
  1. Промена схеме је захтеван посао
  2. Користи се NULL за информације које недостају
  3. Унапред дефинисана схема и метаподаци
  4. Унапред дефинисан број атрибута у табели
  5. Идентичан тип података за све вредности атрибута табеле
- XML подаци
  1. Промена схеме је доста једноставнија
  2. Опциони елементи и атрибути могу бити изостављени
  3. Променљива схема и метаподаци
  4. Променљив број елемената и атрибута у документу
  5. Тип података није обавезан и може бити променљив

Глава 3

XPath

3.1 Увод

XPath језик је направљен са сврхом како би се крајњем кориснику олакшала претрага XML документа као и проналажење потребних података. Разлог је што није могуће на једноставан начин пронаћи потребан елемент, односно чвор који представља тај елемент, онда корисник нема могућност да брзо обради добијене податке у XML документима. XPath језик доживљава XML документ као скуп чворова у структури стабла где постоји више различитих типова чворова. Ти типови могу бити елементи, атрибути, текстуални чворови, чворови именског простора или чворови документа. Тако да је XPath изузетно моћан и користан језик који омогућава кориснику да на једноставан начин извршава жељене операције над XML документима. Најновија верзија XPath језика је 3.1 која је од марта 2017. године и препоручена од стране WWW Конзорцијума.

Што се структуре XPath језика тиче, модел података представља најважнији део језика. Од верзије језика 2.0 јединствен модел података деле три језика [5]:

- XPath
- XSLT[1] 2.0
- XQuery

[1]Extensible Stylesheet Language Transformations (XSLT) - језик који се користи за превођење XML документа у други XML документ или у документ неког другог формата, на пример HTML, PDF, обичан текст, итд.

У моделу података, секвенца се дефинише као колекција од нула или више чланова. Члан може бити атомична вредност или чвор. Секвенца која не садржи ниједан члан назива се празна секвенца док се секвенца са једним чланом назива јединична секвенца (енг. singleton). Важно је напоменути да секвенце никада не могу бити угњеждене и уколико је од ( ), 1 и (2, 3) потребно направити једну секвенцу, та резултујућа секвенца ће бити облика (1, 2, 3).

Изрази путање

Изрази путање представљају један од битнијих делова XPath језика и ти изрази омогућавају кориснику да се креће кроз XML документ како би дошао до потребних података. Изрази путање могу бити формирани уз помоћ кључних речи, симбола или операнада. Најчешћи облик израза путање се састоји од навођења назива елемената из XML документа раздвојених оператором коса црта као што је приказано у примеру 3.1.

```
/zaposleni/ime
```

Пример 3.1: Облик израза путање

У самом изразу путање, поред наведених назива елемената из XML документа, могу се користити и разне функције. XPath језик садржи библиотеку функција која је направљена како би се кориснику олакшала обрада и претрага XML података. Поменута библиотека функција се састоји од нумеричких функција, функција за рад са нискама карактера, функција за конверзију типа података као и функција за рад са читавим XML документом. Неке од функција које се налазе у библиотеци су:

- функције за рад са нискама карактера - concat(), substring(), string-length()
- нумеричке функције - sum(), round(), floor()
- функције за конверзију типа података - string(), number(), boolean()

Оператори

Слично другим програмским језицима, XPath такође има дефинисан скуп оператора који је могуће користити у изразима [6]. Списак најкоришћенијих оператора у XPath језику дат је у табели 3.1.

Фамилија оператора	**Листа доступних оператора**
аритметички оператори	$+$, $-$, $*$, *mod*, *div*, *idiv*
општи оператори поређења	$=$, $!=$, $<$, $>$, $<=$, $>=$
оператори поређења вредности	*eq*, *ne*, *lt*, *gt*, *le*, *ge*
оператори поређења чворова	$<<$, $>>$, *is*
Буловски оператори	*and*, *or*
оператори поређења секвенци	*union*, *intersect*, *except*

Табела 3.1: Доступни оператори у XPath језику

3.2 Извршавање XPath упита над XML подацима

Концепт коришћења путање за навигацију кроз хијерархијску структуру је одавно познат концепт. Прави пример за то је систем датотека (енг. file system). На било ком рачунару са Linux или UNIX оперативним системом, постоји основни директоријум (енг. root directory) који има своје поддиректоријуме а сваки од тих поддиректоријума има сопствене поддиректоријуме и тако даље. Пример путање на UNIX систему датотека је */home/users/student/xml*. Путања се састоји од навођења имена директоријума раздвојених карактером коса црта. Команда *cd /home/users/student/xml* отвара наведени директоријум и то постаје текући директоријум корисника. Команда *cd ..* се користи за кретање кроз директоријуме и позиционира корисника у родитељски директоријум у односу на тренутни директоријум. XPath користи сличан принцип приликом приступања елементима и атрибутима у XML документу. Користи се карактер коса црта за раздвајање два елемента док се две тачке користе за приступ родитељском елементу. За разлику од система датотека, XML елементи могу имати више потомака (енг. child elements) са истим именом.

Основни елемент у примеру 1.1 XML документа је елемент **запослени**. Директан потомак основног елемента је елемент **име**. Пример 3.2 показује како се из XML документа, користећи XPath, може добити списак имена.

```
XPath:  /zaposleni/ime
Output: <ime>Aleksandar Petrovic</ime>
```

Пример 3.2: Добијање имена запослених из XML документа

Треба истаћи да постоји разлика између великих и малих слова приликом навођења елемената у XPath изразу и да се ће приликом погрешног навођења имена елемента за резултат добити празан излаз (пример 3.3).

```
XPath:  /Zaposleni/Ime
Output:
```

Пример 3.3: Навођење погрешних назива елемената

Уколико постоји потреба да се добије излаз без XML ознака у резултату (пример 3.2 враћа име запосленог са све почетном и завршном ознаком **име**) потребно је експлицитно нагласити да је потребан искључиво текст траженог елемента.

```
XPath:  /zaposleni/ime/text()
Output: Aleksandar Petrovic
```

Пример 3.4: Враћање вредности елемента без ознака

Треба нагласити да text() није функција и да никада нема аргументе. Исти резултат као у примеру 3.4 се може постићи помоћу функција string() или data(). У примеру 3.5 функција data() рачуна вредност прослеђеног аргумента и враћа атомичну вредност уместо текста наведеног елемента као што је случај када се користи text().

```
XPath:  /zaposleni/data(ime)
Output: Aleksandar Petrovic
```

Пример 3.5: Коришћење функције data()

Разлика између text() и функције data() (или функције string()) постаје видљива када се примени на елементе који имају потомке као што је елемент **адреса** у примеру 1.1.

Упит из примера 3.6 као резултат враћа празну секвенцу пошто не постоји текст чвор (енг. node) који је директан потомак елемента **адреса** док пример 3.7 враћа вредности свих елемената који су потомци елемента **адреса**.

```
XPath:  /zaposleni/adresa/text()
Output:
```

Пример 3.6: Покушај враћања текстуалне вредности елемента који има потомке

```
XPath:  /zaposleni/string(adresa)
Output: BeogradRuzveltova 31
```

Пример 3.7: Коришћење функције string()

У овом случају, резултат се добија као дописивање вредности свих директних потомака. Приликом дописивања не додаје се размак између вредности два потомка.

Уколико XPath израз не користи неку од претходно наведених функција а показује на елемент који има потомке, као резултат ће се добити елемент са свим потомцима. Овакво понашање, као у примеру 3.8, даје могућност да се издвоје читави фрагменти из XML документа.

```
XPath:  /zaposleni/adresa
Output: <adresa drzava="Srbija">
            <grad>Beograd</grad>
            <ulica>Ruzveltova 31</ulica>
        </adresa>
```

Пример 3.8: XPath пример који враћа део XML документа

Такође, у претходном примеру се може приметити да елемент **адреса** поред два потомка има и атрибут **држава**. Како би се вредност атрибута вратила као резултат XPath израза потребно је користити карактер **@**.

```
XPath:  /zaposleni/adresa/data(@drzava)
Output: Srbija
```

Пример 3.9: Добијање вредности атрибута

Коришћење функције data() или string() обавезно је у XPath изразу уколико се жели добити вредност атрибута. Уколико се изостави функција data() и покуша добити вредност атрибута уз помоћ израза */zaposleni/adresa/@drzava*, XPath ће вратити грешку.

3.2.1 Специјални карактери и специјалне ниске

XPath омогућава коришћење и специјалних карактера. Специјалан карактер ***** се користи као замена за било који елемент док ниска карактера **@*** служи као замена за било који атрибут. XPath израз у примеру 3.10 користи специјалан карактер * да врати све елементе који су директни потомци елемента **адреса**.

```
XPath:  /zaposleni/adresa/*
Output: <grad>Beograd</grad>
        <ulica>Ruzveltova 31</ulica>
```

Пример 3.10: XPath пример са специјалним карактером

У примеру 3.11 специјалан карактер * је на месту директног потомка основног елемента **запослени** тако да се претражују сви елементи којима је елемент **запослени** родитељ а који имају за директног потомка елемент **град**.

```
XPath:  /zaposleni/*/grad
Output: <grad>Beograd</grad>
```

Пример 3.11: XPath пример са специјалним карактером

Као што је наведено, специјална ниска @* користи се за упаривање са било којим атрибутом XML документа. У примеру 3.12 је наведен упит који као резултат враћа атрибут било ког елемента који је директан потомак основног елемента **запослени**.

```
XPath:  /zaposleni/*/data(@*)
Output: Srbija
        poslovni
        privatni
```

Пример 3.12: Пример упита са специјалним ниском у XPath-у

Још једна специјална ниска која чини XPath изразе уопштенијим је дупла коса црта (//). Са овом специјалном ниском може се приступити потомку на било ком нивоу у XML документу. Да би се илустровао пример употребе ове специјалне ниске, потребно је проширити пример 1.1 XML документа са елементом **асистент** који има два потомка, елемент **име** и елемент **телефон**.

```
<zaposleni id="1005">
    <ime>Aleksandar Petrovic</ime>
    <adresa drzava="Srbija">
        <grad>Beograd</grad>
        <ulica>Ruzveltova 31</ulica>
    </adresa>
    <telefon tip="poslovni">060-425-443</telefon>
    <telefon tip="privatni">064-546-6121</telefon>
```

```
    <asistent>
        <ime>Stevan Markovic</ime>
        <telefon tip="poslovni">063-114-5531</telefon>
    </asistent>
</zaposleni>
```

Пример 3.13: Пример XML документа

XPath израз из примера 3.14 тражи сва појављивања елемента са ознаком **име** који је потомак (не обавезно директан већ на било ком нивоу) основног елемента **запослени**.

```
XPath:  /zaposleni//ime
Output: <ime>Aleksandar Petrovic</ime>
        <ime>Stevan Markovic</ime>
```

Пример 3.14: XPath пример са специјалним ниском

Коришћење ове специјалне ниске има предности и мане. Предност је да се доста лако могу наћи сва појављивања одређеног елемента у XML документу као и да је омогућено да се без прецизног знања на ком се нивоу налази елемент дође до тог података. Мана је да се за резултат може добити доста више података уколико се не користи пажљиво. Такође, ту је и питање перформанси пошто коришћење ове специјалне ниске захтева обраду доста већег дела XML документа да би се добили резултати.

3.2.2 Предикати

Претходни примери XPath израза нису имали дефинисане услове који би филтрирали податке који се враћају у резултату а таква могућност је од велике користи приликом свакодневног коришћења. У XPath језику услови за филтрирање (предикати) се наводе у угластим заградама и могу се наћи у било ком делу XPath израза. Да би се на примерима боље приказао рад са условима који филтрирају податке, XML документ ће бити проширен са подацима за још једног запосленог.

```
<zaposleni id="1005">
    <ime>Aleksandar Petrovic</ime>
    <adresa drzava="Srbija">
        <grad>Beograd</grad>
```

```
        <ulica>Ruzveltova 31</ulica>
    </adresa>
    <telefon tip="poslovni">060-425-443</telefon>
    <telefon tip="privatni">064-546-6121</telefon>
    <asistent>
        <ime>Stevan Markovic</ime>
        <telefon tip="poslovni">063-114-5531</telefon>
    </asistent>
</zaposleni>
<zaposleni id="1006">
    <ime>Nevena Mijatovic</ime>
    <adresa drzava="Srbija">
        <grad>Nis</grad>
        <ulica>Kneza Milosa 77b</ulica>
    </adresa>
    <telefon tip="poslovni">066-335-2243</telefon>
    <telefon tip="privatni">060-116-7713</telefon>
    <asistent>
        <ime>Ana Jeremic</ime>
        <telefon tip="poslovni">062-334-3155</telefon>
    </asistent>
</zaposleni>
```

Пример 3.15: Пример XML документа

Пример 3.16 показује како се може добити име запосленог којем је id=1005. Прецизније, тражи се сваки основни елемент **запослени** који има атрибут **ид** а онда се проверава да ли је тај атрибут једнак 1005. Уколико се пронађе такав унос у XML документу враћа се текстуални приказ имена тог запосленог.

```
XPath:  /zaposleni[@id=1005]/ime/text()
Output: Aleksandar Petrovic
```

Пример 3.16: Коришћење услова за филтрирање резултата

Треба напоменути да у једном XPath изразу може постојати више услова као и то да сваки услов мора ићи после наведеног елемента односно, не може се навести услов одмах после косе црте.

Поред поређења вредности нумеричког типа које је илустровано у претходном примеру, могу се поредити и алфанумеричке ниске. У случају да је услов био [@id="1005"] тада би се атрибут **ид** запосленог поредио са ниском карактера "1005" а не са нумеричком вредношћу 1005. Пример таквог поређења је издвајање имена свих запослених који су пријављени да станују у Нишу као што је наведено у примеру 3.17.

```
XPath:  /zaposleni[adresa/grad="Nis"]/ime/text()
Output: Nevena Mijatovic
```

Пример 3.17: Коришћење услова за филтрирање резултата

Услов [*adresa/grad="Nis"*] проверава да ли за сваки основни елемент **запослени** постоји потомак **адреса** који даље има сопствени потомак **град** чија је вредност ниска карактера "Ниш". Уколико постоје такви уноси у XML документу враћају се текстуалне вредности елемента **име**. У XPath изразу одмах после услова се налази елемент **име** и он се посматра у односу на елемент пре услова (**запослени**) а не у односу на коришћене елементе у оквиру угластих заграда.

Такође, треба имати у виду да услов у угластим заградама обично никада не почиње са карактером коса црта пошто би то значило да се услов не односи на тренутни елемент иза ког је наведен већ да је то основни елемент у XML документу. Уколико у XPath изразу дефинишемо услов без коришћења угластих заграда (као у примеру 3.18) то ће се посматрати као Буловски израз и резултат тог упита ће бити тачно уколико постоји бар један запослени са именом "Невена Мијатовић", или ће у супротном резултат тог израза бити нетачно.

```
XPath:  /zaposleni/ime="Nevena Mijatovic"
Output: true
```

Пример 3.18: Буловски израз

До сада у овом поглављу било је речи о вредносним и структурним предикатима. Вредносни предикати пореде елемент или атрибут из XML документа са вредношћу неког броја или ниске карактера. С друге стране, структурне предикате не занима вредност већ структура XML документа и проверава се постојање траженог елемента или атрибута у XML документу. Поред две наведене врсте предиката, постоји и трећа, позициони предикати. Они се могу користити да се из документа добију резултати на основу позиције на којој се налазе у документу.

Позициони предикат у XPath изразу се дефинише као цео број у угластим заградама што се може видети у примеру 3.19 и означава да се као резултат врате сви бројеви телефона који се налазе на другом месту у документу за сваког запосленог.

```
XPath:  /zaposleni/telefon[2]
Output: <telefon tip="privatni">064-546-6121</telefon>
        <telefon tip="privatni">060-116-7713</telefon>
```

Пример 3.19: XPath израз са позиционим предикатом

Уколико се жели видети последње додати број телефона за сваког запосленог, у позиционом предикату може се користити функција last() уместо конкретног броја.

3.2.3 Егзистенцијална семантика

Приликом коришћења XPath језика, егзистенцијална семантика је нешто што се примењује приликом сваког враћања резултата. Конкретно, егзистенцијална семантика значи да је довољно да постоји бар једно поклапање у услову да би се цео услов вредновао као тачан. На пример, уколико имамо две секвенце (2, 3, 7) и (1, 2, 4), поређење (2, 3, 7) = (1, 2, 4) ће бити вредновано као тачно пошто постоји бар један члан прве секвенце који је једнак бар једном члану друге секвенце. У овом примеру то је члан са вредношћу 2. Оно што може изгледати чудно на први поглед, поређење (2, 3, 7) != (1, 2, 4) ће се такође вредновати као тачно јер у првој секвенци постоји бар један члан који није једнак бар једном члану из друге секвенце и то је у егзистенцијалној семантици довољан услов да цео исказ буде вредноване као тачан (primer 3.20).

```
XPath:  /zaposleni[adresa/grad != ("Nis", "Beograd")]/ime/text()
Output: Aleksandar Petrovic
        Nevena Mijatovic
```

Пример 3.20: Егзистенцијална семантика

Још један од примера егзистенцијалне семантике наведен је у примеру 3.21. Услов се вреднује као тачан уколико елемент **адреса**, који је директан потомак основног елемента **запослени**, поседује атрибут **држава**. Уколико је тај услов испуњен као резултат се враћају имена свих запослених.

```
XPath:  /zaposleni[adresa/@drzava]/ime/text()
Output: Aleksandar Petrovic
        Nevena Mijatovic
```

Пример 3.21: Егзистенцијална семантика

3.2.4 Логички оператори

Слично SQL језику и XPath омогућава коришћење логичких оператора као што су **and** ili **or** као и функције not() која негира Буловску вредност прослеђеног аргумента. Пример XPath израза са условом који користи један од логичких оператора дат је у примеру 3.22. Овај пример враћа име запосленог чији је приватни број телефона 060-425-443 али такав запослени не постоји у XML документу пошто је то Александров пословни број из примера 3.15.

```
XPath:  /zaposleni[telefon="060-425-443" and telefon/@tip="privatni"]/ime
   /text()
Output: Aleksandar Petrovic
```

Пример 3.22: Коришћење логичког оператора

Поставља се питање, како се онда не добија празан резултат? Одговор је у томе што овако написан услов значи "да ли постоји запослени чији је број телефона 060-425-443" и "да ли тај запослени има приватни телефон"? Пошто је за запосленог Александра одговор на оба питања потврдан, егзистенцијална семантика коју користи XPath, враћа његово име као резултат иако то није оно што је требало овај израз да постигне. Да би се овај проблем решио, потребно је да буде назначено да баш тражени број телефона мора бити и приватни. То се може постићи уколико се модификује постојећи израз користећи угњеждене угласте заграде у услову као у наведеном примеру 3.23.

```
XPath:  /zaposleni[telefon [text()="060-425-443" and @tip="privatni"] ]/
   ime/text()
Output:
```

Пример 3.23: Коришћење угњеждених угластих заграда

Спољашње угласте заграде говоре да као резултат треба вратити име запосленог само уколико постоји одређени елемент **телефон**, који је директан потомак основног елемента **запослени** и који испуњава додатне услове а који су наведени у унутрашњим угластим заградама. Услов у унутрашњим угластим заградама наводи да је потребно да број телефона буде "060-425-443" као и да је то приватни телефон. Као што се може видети у резултату претходног примера такав запослени не постоји.

Као што је раније споменуто, функција not() негира вредност прослеђеног аргумента а пример 3.24 најбоље објашњава то понашање.

```
XPath:  /zaposleni[not(telefon = "060-425-443")]/ime/text()
Output: Nevena Mijatovic
```

Пример 3.24: Употреба функције not()

Потребно је наћи име запосленог који не поседује телефон са бројем "060-425-443". За овакав услов не може се искористити није једнако (енг. not equal) јер, као што је већ раније наведено, егзистенцијална семантика, са тако дефинисаним условом, тражи запосленог који поседује бар један телефон чији број није "060-425-443". Зато, потребно је пронаћи запосленог који поседује бар један телефон са бројем "060-425-443" и то проследити у функцију not() као аргумент.

3.2.5 Унија и формирање секвенци

У претходним примерима вредности које су се добијале у резултатима обично су биле јединственог типа. Другим речима, враћала се вредност једног елемента, било да је то име запосленог, његова држава, бројеви телефона и слично. Некада то није довољно и потребно је вратити више различитих елемената и то се може постићи коришћењем оператора унија, који се може користити у XPath изразу као кључна реч *union* или као карактер вертикална црта (енг. pipe).

```
XPath:  /zaposleni/adresa/(grad|ulica)
Output: <grad>Beograd</grad>
        <ulica>Ruzveltova 31</ulica>
        <grad>Nis</grad>
        <ulica>Kneza Milosa 77b</ulica>
```

Пример 3.25: Коришћење оператора унија

Поред уније постоје још и оператори пресека (енг. intersect) и разлике (енг. except). Секвенце могу да се формирају коришћењем карактера зарез. Добијени резултат је исти као и коришћењем оператора уније уз разлику да унија искључује дупликате из резултата док то није случај са формирањем секвенце помоћу оператора зарез. Тако пример 3.26 XPath израза враћа исти резултат као и пример 3.25

```
XPath:  /zaposleni/adresa/(grad,ulica)
Output: <grad>Beograd</grad>
        <ulica>Ruzveltova 31</ulica>
        <grad>Nis</grad>
        <ulica>Kneza Milosa 77b</ulica>
```

Пример 3.26: Формирање секвенце оператором зарез

Оператор зарез дозвољава коришћење атомичних вредности приликом конструкције секвенце док унија за аргументе мора имати елементе или атрибуте.

3.2.6 Функције

У претходним примерима наведене су неке од функција, као што су data(), string(), not() које се могу користити у XPath изразима. Наравно, то нису све функције које се могу користити пошто XPath поседује велики број уграђених функција. Неке од функција које се могу користити су: count(), sum(), contains(), substring(), upper-case(), concat(), итд. [7] Неки од примера XPath израза који користе уграђене функције су:

```
XPath:  /zaposleni/count(telefon)
Output: 2
        2
```

Пример 3.27: Коришћење функције count()

```
XPath:  /zaposleni/concat(ime, " - ", adresa/grad)
Output: Aleksandar Petrovic - Beograd
        Nevena Mijatovic - Nis
```

Пример 3.28: Коришћење функције concat()

Глава 4

XQuery

4.1 Увод

Програмски језик XQuery спада у групу функционалних програмских језика. Креирао га је, 2007. године, Конзорцијум за управљање Web-ом (енг. World Wide Web Consortium, W3C) како би се задовољила потреба корисника за брзим изменама над великом количином XML података као и давање додатних могућности корисницима приликом извршавања упита над XML подацима. Верзија 1.0 XQuery језика се може посматрати као проширење верзије 2.0 XPath језика. Уколико је израз синтаксно исправан у XPath 2.0 и XQuery 1.0 језику, крајњи корисник ће као коначан резултат добити исту вредност у оба језика. За програмски језик XQuery често се каже да за XML податке представља оно што језик SQL представља за релационе базе података. Структура XML података доста је флексибилнија од структуре релационих података па одатле потреба да се дефинише један нови језик који ће моћи да одговори на такве захтеве.

Најважнији део XQuery упита су изрази. Изрази формирају део XQuery упита који се назива тело упита (енг. query body) и у наставку поглавља ће детаљније бити објашњена улога израза и који све типови израза постоје. Пре тела XQuery упита може постојати део који се назива пролог (енг. prolog). То је део упита у којем се наводи низ декларација које служе да дефинишу окружење у којем ће се тај упит извршавати док се тело упита састоји од израза помоћу којих ће се добити резултати упита. Овај део упита може бити сачињен од већег броја израза који су повезани одређеним операторима или кључним речима.

DB2 сервер проверава да ли упит почиње са кључном речју *xquery*, и уколико је то случај, то говори DB2 преводиоцу да ће у наставку бити наведен XQuery упит и да треба обратити пажњу на мала и велика слова пошто у XQuery језику постоји разлика између њих. Што се пријављивања грешака тиче, то је разрешено на исти начин као и у SQL језику, помоћу SQLCODE и SQLSTATE исписа док се, за разлику од SQL-а, упозорења не враћају.

Као што ће детаљније бити објашњено у следећем поглављу (Поглавље 5) SQL језик је довољно добар и може послужити као прихватљиво решење за рад са XML подацима али ипак XQuery језику би требало дати предност у следећим случајевима:

- ажурирање постојећих XML података
- претрага XML документа уколико се не зна његова структура и на ком се нивоу хијерархијске структуре налази жељени елемент
- потребно је извршити структурну трансформацију над XML подацима

XQuery функције

Подаци који се појављују у крајњим резултатима добијају се тако што XQuery позива функције које издвајају XML податке из DB2 табеле. Функције које XQuery упит може користи како би се добили тражени XML подаци су **db2-fn:sqlquery** и **db2-fn:xmlcolumn**.

Функција db2-fn:xmlcolumn као аргумент узима знаковни литерал (енг. string literal) помоћу којег идентификује табелу у бази података која садржи атрибут табеле XML типа и као резултат враћа низ XML вредности које се налазе у том атрибуту. У примеру 4.1 функција db2-fn:xmlcolumn издваја из табеле **запослени** све податке који се налазе у атрибуту табеле **инфо**. Над тако добијеним подацима приказују се само имена запослених као коначан приказ резултата кориснику.

```
XQuery:
xquery db2-fn:xmlcolumn('ZAPOSLENI.INFO')/zaposleni/ime;

Output:
```

```
1
------------------------------
<ime>Aleksandar Petrovic</ime>
<ime>Nevena Mijatovic</ime>
<ime>Ivana Stefanovic</ime>
```

Пример 4.1: Коришћење функције db2-fn:xmlcolumn

Пошто се не користи додатан услов приликом претраге, ова функција дозвољава кориснику да као резултат врати све вредности података из атрибута табеле XML типа. Треба нагласити да корисник, приликом навођења литерала као аргумента функције, мора обратити пажњу на употребу малих и великих слова пошто их XQuery функције разликују.

Функција db2-fn:sqlquery као аргумент узима ниску карактера која представља подупит и враћа XML секвенцу која је настала конкатенацијом XML вредности које су добијене подупитом. У резултујућем скупу који враћа наведени подупит морају бити подаци из атрибута табеле XML типа. Пример 4.2 користи функцију db2-fn:sqlquery за добијање резултата за разлику од претходног примера који је користио функцију db2-fn:xmlcolumn. Разлика је што се помоћу SQL упита, који је наведен као аргумент db2-fn:sqlquery функције, као међурезултат добијају сви подаци само о оном запосленом чији је **ид** једнак 1007. Тек над тако добијеним резултатом формира се коначан резултат који се приказује крајњем кориснику а то је, у овом случају, само име запосленог.

```
XQuery:
xquery db2-fn:sqlquery("SELECT info
                        FROM zaposleni
                        WHERE id = 1007")/zaposleni/ime;

Output:
1
---------------------------
<ime>Ivana Stefanovic</ime>
```

Пример 4.2: Коришћење функције db2-fn:sqlquery

XML именски простор

XML именски простор (енг. namespace) је колекција имена која се идентификују помоћу именског простора јединственог идентификатора ресурса (енг. Universal Resource Identifier, URI). Постојање именског простора је начин да се квалификују имена која се користе за типове података, елементе, атрибуте и функције у оквиру XQuery језика. Име које је квалификовано префиксом именског простора назива се квалификовано име (енг. qualified name, QName).

Квалификовано име се састоји из два дела. Први део је префикс именског простора и тај део је опциони док је други део локално име. Та два дела именског простора су раздвојена са две тачке. Префикс именског простора, уколико је наведен, повезује се са јединственим идентификатором ресурса и креира његову скраћену верзију. Током обраде упита, XQuery проширује квалификовано име и разрешава јединствени идентификатор ресурса који је повезан са префиксом именског простора. Два квалификована имена могу бити идентична само уколико се њихове проширене верзије поклапају, односно уколико им је и јединствени идентификатор ресурса и локално име идентично. То такође значи да два квалификована имена могу бити једнака и уколико имају различите префиксе именског простора али под условом да су оба префикса повезана на исти јединствени идентификатор ресурса. Уколико се префикс именског простора не наведе, пошто је то опциони део квалификованог имена, локално име се повезује са подразумеваним елементом именског простора.

У DB2 постоји одређени број префикса именских простора који су унапред декларисани [8]. Списак тих префикса је приказан у табели 4.1.

Префикс	**Јединствени идентификатор ресурса (URI)**
xml	`http://www.w3.org/XML/1998/namespace`
xs	`http://www.w3.org/2001/XMLSchema`
xsi	`http://www.w3.org/2001/XMLSchema-instance`
fn	`http://www.w3.org/2005/xpath-functions`
xdt	`http://www.w3.org/2005/xpath-datatypes`
db2-fn	`http://www.ibm.com/xmlns/prod/db2/functions`

Табела 4.1: Унапред декларисани префикси именских простора

Као што је неведено у примеру 4.3 додатни префикси именских простора могу бити декларисани не неколико начина:

1. декларисањем именског простора у прологу упита
2. декларисањем подразумеваног именског простора у прологу упита
3. декларисањем у склопу атрибута приликом конструкције елемента

```
(1) declare namespace ns1 = "http://namespacesample.org";

(2) declare default element namespace "http://namespacesample.org";

(3) <ns2:grad xmlns:ns2="http://namespacesample.org">Beograd</ns2:grad>
```

Пример 4.3: Декларисање додатних префикса именских простора

Предекларисан префикс именског простора може бити редефинисан са новим декларисањем у прологу упита док префикс именског простора који је декларисан у прологу упита може бити редефинисан декларисањем у склопу атрибута приликом конструкције елемента [9].

4.2 Изрази

Као што је већ споменуто у уводу овог поглавља, изрази представљају најважнији део при формирању XQuery упита. Тело упита састоји се од једног или већег броја израза помоћу којих се добијају коначни резултати. Како је XPath подскуп XQuery језика део израза је споменут и у XPath поглављу (Поглавље 3). Неки од најважнијих XQuery израза су: [10]

- **Изрази путање** (енг. Path expressions) - користе се за проналажење елемената и атрибута у склопу XML документа. Изрази путање у XQuery језику засновани су на синтакси језика XPath 2.0 детаљније објашњених у поглављу 3.
- **Изрази поређења** (енг. Comparison expressions) - служе за поређење две вредности. У XQuery језику постоје три врсте израза поређења: опште, вредносно и поређење на нивоу чвора. Две вредности се могу поредити само ако су истог типа или уколико је тип једног операнда подтип другог. Оператори који се користе код вредносног типа поређења су **eq**, **ne**, **lt**, **le**, **gt** и **ge** (детаљније у табели 4.2). Резултат вредносног поређења може бити Буловска вредност, празна секвенца или грешка.

Оператор	**Функција оператора**
eq	Враћа тачно уколико је прва вредност једнака другој вредности
ne	Враћа тачно уколико је прва вредност није једнака другој вредности
lt	Враћа тачно уколико је прва вредност мања од друге вредности
le	Враћа тачно уколико је прва вредност мања или једнака другој вредности
gt	Враћа тачно уколико је прва вредност већа од друге вредности
ge	Враћа тачно уколико је прва вредност већа или једнака другој вредности

Табела 4.2: Вредносно поређење - списак оператора

Опште поређење покушава да утврди да ли, у две секвенце било које дужине, постоји бар по један члан који задовољава услов поређења. Оператори у овој врсти поређења су =, ! =, <, <=, > и >= (детаљније о коришћеним операторима у табели 4.3). Резултат општег поређења је Буловска вредност или грешка.

Последња врста поређења је поређење на нивоу чвора. Чворови се пореде како би се утврдило да ли један чвор претходи, односно следи, други чвор у документу или да ли су та два чвора идентична. Оператори који се користе су <<, >> и **is** (детаљније у табели 4.4) док резултат таквог поређења може бити Буловска вредност, празна секвенца или грешка.

- **Изрази секвенци** (енг. Sequence expressions) - могу се користити за конструкцију нових секвенци или комбинацију постојећих. Приликом формирања секвенце користи се оператор зарез или оператор опсега. Уколико се користи оператор зарез, једна секвенца је дефинисана са два или више операнда раздвојена зарезом. Секвенца никада не може садржати другу секвенцу, већ ће дужина резултујуће секвенце бити сума дужина свих улазних секвенци. На пример, ако израз комбинује пет секвенци чије су дужине два $(3, 8)$, један (2), нула (), три $(9, 10, 1)$, један (5) дужина резултујуће секвенце ће бити седам $(3, 8, 2, 9, 10, 1, 5)$ односно сума дужина свих наведених секвенци.

  Секвенца формирана оператором опсега састоји се од два целобројна операнда раздвојена оператором **to**. Резултат формирања секвенце на тај начин је секвенца која садржи узастопне целе бројева из опсега наведених операндима. На пример, резултат израза (3 to 7) је $(3, 4, 5, 6, 7)$. У случају да је први операнд већи од другог или уколико је први операнд празна секвенца крајњи резултат израза је празна секвенца. Такође, могуће је и комбиновати ова два начина

Оператор	**Функција оператора**
=	Враћа тачно уколико је бар једна од вредности из прве секвенце једнака бар једној од вредности из друге секвенце
! =	Враћа тачно уколико је бар једна од вредности из прве секвенце није једнака бар једној од вредности из друге секвенце
<	Враћа тачно уколико је бар једна од вредности из прве секвенце мања од бар једне од вредности из друге секвенце
<=	Враћа тачно уколико је бар једна од вредности из прве секвенце мања или једнака бар једној од вредности из друге секвенце
>	Враћа тачно уколико је бар једна од вредности из прве секвенце већа од бар једне од вредности из друге секвенце
>=	Враћа тачно уколико је бар једна од вредности из прве секвенце већа или једнака бар једној од вредности из друге секвенце

Табела 4.3: Опште поређење - списак оператора

и као један операнд приликом креирања секвенце користити секвенцу са оператором опсега. Односно, израз $(10, 1 \text{ to } 4)$ је синстаксно исправан и резултат оваквог израза је $(10, 1, 2, 3, 4)$.

- **Логички изрази** (енг. Logical expressions) - користе операторе *and* или *or* за добијање резултата који могу бити искључиво Буловске вредности (тачно или нетачно) или грешка.

- **Аритметички изрази** (енг. Arithmetic expressions) - XQuery језик поседује аритметичке операторе за сабирање $(+)$, одузимање $(-)$, множење $(*)$, дељење $(/)$, целобројно дељење $(idiv)$ и остатак при дељењу (mod). Резултат аритметичког израза може бити нумеричка вредност, празна секвенца или грешка. Када се операнди преведу у атомичне вредности, резултат је празна секвенца уколико је један од преведених операнада празна секвенца док се грешка добија уколико се преведени операнд не може конвертовати у тип **xs:double** или садржи више од једне вредности.

- **Изрази трансформисања** (енг. Transform expressions) - употребљавају се уколико је потребно трансформисати или ажурирати постојећи XML документ. Детаљније о изразима трансформисања и ажурирању XML докумената у Поглављу 7.

- **Изрази конверзије** (енг. Cast expressions) - омогућавају да се крајња вредност преведе у други тип података и састоје се од два операнда, улазног израза и

Оператор	**Функција оператора**
is	Враћа тачно уколико су два чвора која се пореде идентична
<<	Враћа тачно уколико у документу чвор из првог операнда претходи чвору који је наведен као други операнд
>>	Враћа тачно уколико у документу чвор из првог операнда следи чвор који је наведен као други операнд

Табела 4.4: Поређење на нивоу чвора - списак оператора

цильног типа у који се жели превести резултат. Улазни израз се атомизацијом[1] преводи у атомичну вредност или у празну секвенцу након чега израз конверзије покушава да формира нову вредност цильног типа. Може се десити да цильни тип није компатибилан са вредношћу добијеном од улазног израза због чега се као резултат добија грешка. Грешка за резултат се може добити и уколико је резулат атомизације секвенца од две или више атомичне вредности. Опционо, знак питања на крају израза оставља могућност да се за улазни израз може користити и празна секвенца.

- **Условни изрази** (енг. Conditional expressions) - користе наредбу **if-then-else** како би се утврдило који се израз извршава у зависности од Буловске вредности наведеног услова. Уколико је резултат услова **тачно** онда се извршава израз наведен у грани **then** у супротном, извршава се израз из гране **else**.

Поред до сада обрађених израза, у овом поглављу биће наведена још два типа али ће се они, због своје обимности, детаљније обрадити у засебним секцијама. Споменути изрази су:

- **FLWOR изрази** (енг. FLWOR expressions) - омогућавају пролазак кроз секвенце и повезивање променљивих са међурезултатима упита. Корисни су за спајање и комбиновање два или више XML документа, или сортирање резултата.
- **Конструктор изрази** (енг. Constructor expressions) - XQuery конструктори могу се искористити за формирање XML елемената или атрибута како би се формирао нови XML документ у оквиру упита.

[1] атомизација (енг. atomization) - користи се када је у изразима потребна секвенца атомичних вредности и представља процес превођења секвенце чланова у секвенцу атомичних вредности.

4.3 FLWOR изрази

Изрази који нуде кориснику највише могућности и вероватно једни од најмоћнијих и најкоришћенијих израза у оквиру XQuery језика су FLWOR изрази. Оно што FLWOR изрази представљају за XML податке може се упоредити са SELECT-FROM-WHERE наредбом над релационим подацима из SQL језика. Назив FLWOR је настао од кључних речи *for*, *let*, *where*, *order by* и *return*. У наставку ове секције биће дат детаљан опис синтаксе FLWOR израза.

4.3.1 Синтакса FLWOR израза

У систему за управљање релационим базама података DB2 на почетку XQuery израза наводи се кључна реч **xquery** која указује да ће у наставку бити наведен XQuery а не SQL упит. Након те кључне речи наводи се наредба **for**. Она се састоји од кључне речи *for*, променљиве, кључне речи *in* и израза. Наредба **for** пролази кроз секвенцу чланова одређеним изразом *expression1* из примера 4.4. Наредба **let** не пролази кроз све чланове већ додељује променљивој *$var2* целу секвенцу одређеном изразом *expression2*. Сваки FLWOR израз мора имати бар једну од ове две наредбе. Наредбе **where** и **order by** су опционе и имају исту функцију као и у SQL упиту, односно филтрирају и сортирају резултујући скуп који се касније враћа као крајњи резултат кориснику кроз наредбу **return**.

```
xquery
for $var1 in expression1
let $var2 := expression2
where expression3
order by expression4 [ascending|descending]
return expression3
```

Пример 4.4: Синтакса FLWOR израза

Приликом дефинисања имена променљиве у XQuery језику обавезно је навођење долар карактера ($). Такође треба нагласити да једна или више белина између долар карактера и почетка имена променљиве не представља проблем XQuery преводиоцу. Односно, уколико су променљиве дефинисане као $ $var1$ и $ $var1$ оне су синтаксно исправне и идентичне али се због читљивости самог израза препоручује да се непотребне белине изоставе.

Наредбе for и let

Као што је већ наведено, сваки FLWOR израз мора имати бар једну **for** или **let** наредбу. Те две наредбе формирају нове промењљиве које се могу искористити у другим наредбама FLWOR израза. Пошто обе нарадбе додељују вредности промењљивама, кроз пример 4.5 биће детаљније разјашњено на који начин то раде и које су разлике. Како би се пример што више поједноставио, не користе се изрази помоћу којих би се добиле секвенце које садрже вредности XML елемената добијених из атрибута табеле XML типа већ је формирана секвенца која садржи атомичне вредности 5, 2 и 8. Први FLWOR израз у примеру 4.5 користи наредбу **for** како би прошао кроз све чланове наведене секвенце. У првој итерацији, промењљивој $\$i$ додељује се вредност 5, у другој итерацији вредност 2 а у трећој итерацији вредност 8. Приликом сваке итерације наредба **return** креира елемент **output** чија је вредност, вредност промењљиве $\$i$ у тренутној итерацији. Тако да овакав FLWOR израз као резултат враћа онолики број редова колико има чланова улазне секвенце, у овом случају три.

```
(1) XQuery:
xquery for $i in (5,2,8) return <output>{$i}</output>;

Output:
<output>5</output>
<output>2</output>
<output>8</output>

(2) XQuery:
xquery let $j := (5,2,8) return <output>{$j}</output>;

Output:
<output>5 2 8</output>
```

Пример 4.5: Разлика између наредби for и let

Други FLWOR израз, у примеру 4.5, користи наредбу **let** и идентичну секвенцу са атомичним вредностима. Супротно понашању наредбе **for**, наредба **let** не пролази кроз секвенцу већ промељивој $\$j$ додељује читаву секвенцу. Тако да наредба **return** за резултат креира само један елемент **output** који садржи све вредности из секвенце.

Ограничење у смислу максималног броја употребљених наредби **for** или **let** у FLWOR изразу не постоји. Пример 4.6 показује понашање и резултате када се користе две наредбе **for**. Може се повући паралела између понашања угњеждених наредби **for** и угњеждене *for* петље из програмских језика. Прва наредба **for** пролази спољашњу секвенцу (5,2,8) док друга пролази кроз унутрашњу секвенцу (a,b) и за сваку итерацију спољашње секвенце пролази се кроз целу унутрашњу секвенцу. Последица тога је да ће се као резултат добити Декартов производ улазних секвенци. Овакво понашање је слично Декартовом производу две табеле у SQL упиту (оператору CROSS JOIN у SQL-u).

```
XQuery:
xquery
for $i in (5,2,8)
    for $j in ("a","b")
return <output>{$i,$j}</output>;

Output:
<output>5 a</output>
<output>5 b</output>
<output>2 a</output>
<output>2 b</output>
<output>8 a</output>
<output>8 b</output>
```

Пример 4.6: Коришћење две наредбе for

Пример комбиноване употребе наредби **for** и **let** у FLWOR изразу дат је у примеру 4.7. Како наредба **let** не пролази кроз секвенцу тако и њено коришћење заједно са наредбом **for** не може довести до Декартовог производа секвенци.

```
XQuery:
xquery for $i in (5,2,8)
let $j := ("a","b")
let $k := $i *2
return <output>{$i,$j,$k}</output>;
```

```
Output:
<output>5 a b 10</output>
<output>2 a b 4</output>
<output>8 a b 16</output>
```

Пример 4.7: Коришћење наредби for и let у једном FLWOR израз

Наредба **return** формира елемент **output** за сваку итерацију наредбе **for**. Вредност тог елемента састоји се од вредности променљивих $\$i$, $\$j$ и $\$k$ где је вредности променљиве $\$i$ вредност тренутног члана секвенце из наредбе **for** док су вредности променљивих $\$j$ и $\$k$ додељене наредбом **let**. Такође, треба напоменути да вредност променљиве $\$k$ није константна већ зависи од тренутне вредности променљиве $\$i$ која се мења у свакој итерацији.

Наредбе where и order by

XQuery упит из примера 4.5 са наредбом **for** послужиће као основа за наредни пример 4.8 како би се детаљније објасниле опционе наредбе **where** и **order by** у FLWOR изразу. У првом случају, пример 4.5 је проширен тако што му је додата наредба **where**. Сада нови израз (пример 4.8) у коначном резултату више не формира елемент **output** за сваку вредност секвенце наведене у наредби **for** већ приликом сваке итерације испитује се да ли је вредност тренутног члана секвенце већа од 2. Уколико је услов испуњен, биће формиран елемент **output** са вредношћу тог члана док се у супротном тај члан одбацује из коначног резултата.

```
(1) XQuery:
xquery for $i in (5,2,8)
where $i > 2
return <output>{$i}</output>;

Output:
<output>5</output>
<output>8</output>

(2) XQuery:
xquery for $i in (5,2,8)
where $i > 2
```

```
order by $i descending
return <output>{$i}</output>;

Output:
<output>8</output>
<output>5</output>
```

Пример 4.8: Коришћење наредби where и order by

Други случај проширује претходни пример са наредбом **order by**. Том наредбом омогућава се да се добијени резултати, пре приказивања крајњем кориснику, уреде на дефинисани начин. У овом примеру, све чланове секвенце чије вредности задовољавају наведени услов потребно је уредити у опадајући поредак и тада вратити крајњем кориснику као коначан резултат. Уколико се користи наредба **order by** а не наведе у ком се поретку враћају резултати, подразумевана опција је да се враћају у растућем редоследу.

4.3.2 Употреба FLWOR израза

Досадашњи примери су послужили како би се синтакса FLWOR израза што једноставније објаснила. Ипак, како би се стекла што боља слика, у наставку ће бити наведени и детаљно објашњени конкретнији FLWOR израз над табелом **запослени** [A.1].

У примеру 4.9 приказана су четири начина како се као резултат може добити списак имена запослених из XML документа.

```
(1) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
return $i/ime;

(2) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni/ime
return $i;
```

```
(3) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")
return $i/zaposleni/ime;

(4) XQuery:
xquery db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni/ime;

Output:
1
------------------------------
<ime>Aleksandar Petrovic</ime>
<ime>Nevena Mijatovic</ime>
<ime>Ivana Stefanovic</ime>
```

Пример 4.9: Добијање имена запослених XQuery упитом

У првом упиту израз који је коришћен у наредби **for** је израз путање који формира секвенцу од елемента **запослени** добијеног из XML документа смештеног у атрибуту **инфо** табеле **запослени**. Наредба **for** пролази кроз све елементе из секвенце и у свакој итерацији, променљивој $\$i$ додељује један члан секвенце односно елемент **запослени** из секвенце. Наредба **return** користи променљиву $\$i$ помоћу које, за тренутни елемент **запослени** у итерацији, издваја елемент **име** и враћа крајњем кориснику као резултат упита.

Други упит се разликује у односу на први по томе што је елемент **име** пребачен из наредбе **return** у наредбу **for**. Односно, одмах се у наредби **for** формира секвенца од елемената **име** која се додељује променљивој $\$i$. Након тога, наредба **return** враћа као резултат вредност променљиве $\$i$.

Трећи упит узима читаве XML документе у наредби **for** и од њих формира секвенцу коју додељује променљивој $\$i$. Та променљива се користи у наредби **return** где се након тога наводи цела путања до траженог елемента у добијеном документу у тренутној итерацији.

Четврти приступ се разликује по томе што не користи FLWOR израз за разлику од прва три упита већ једноставан израз путање који враћа списак свих елемената **име**.

Пример 4.10 проширује пример 4.9 тако што додаје услов да је потребно вратити само име запосленог који има вредност атрибута **ид** 1006. Разлика у односу на претходни пример је та што се наредбом **where** врши филтрирање елемената **запослени**. Наиме, уколико елемент **запослени** испуњава услов да има атрибут **ид** чија је вредност 1006 онда ће за елемент у тој итерацији наредба **return** вратити елемент **име**. У супротном, елемент ће се одбацити и наредба **return** не може вратити такав елемент кориснику у коначном резултату.

```
(1) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/@id = 1006
return $i/ime;

(2) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni/ime
where $i/../@id = 1006
return $i;

(3) XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni[@id = 1006]
return $i/ime;

(4) XQuery:
xquery
db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni[@id = 1006]/ime;

Output:
1
----------------------------
<ime>Nevena Mijatovic</ime>
```

Пример 4.10: Коришћење услова у XQuery упиту

Први и други случај се, као и у претходном примеру, разликују по томе што је елемент **име** пребачен из наредбе **return** у наредбу **for**. Сада оваква измена утиче на услов у наредби **where** пошто се у услову користи променљива из наредбе **for**. У другом случају променљива $\$i$ се везује са елементом **име** и потребно је, у наредби **where**, вратити се један ниво изнад јер је атрибут **ид** дете елемента **запослени**. Овакав начин кретања кроз документ чини други FLWOR израз мало скупљим од првог из угла перформанси.
Трећи FLWOR израз показује да се услов по којем се филтрирају резултати не мора увек наводити у наредби **where** већ се може навести у изразу путање у склопу наредбе **for**.
Четврти приступ и у овом примеру се разликује од осталих по томе што не користи FLWOR израз већ израз путање.

Између наведених начина у претходном примеру нема драстичних разлика из угла перформанси. Што се тиче читљивости упита и одржавања, пожељно је да упит буде што једноставнији тако да је у овом случају четврта опција најпожељнија пошто нема потребе користити FLWOR изразе уколико се жељени резултат може добити једноставним изразом путање.

Пример 4.11 приказује коришћење свих пет наредби у FLWOR изразу. Наиме, потребно је као резултат вратити списак запослених, уређених у опадајућем редоследу, који су из Београда.

```
XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
let $j := $i/ime
where $i/adresa/grad = "Beograd"
order by $i/@id descending
return $j;

Output:
1
------------------------------
<ime>Ivana Stefanovic</ime>
<ime>Aleksandar Petrovic</ime>
```

Пример 4.11: XQuery пример са FLWOR изразом

Израз који је у овом примеру коришћен у наредби **for** је израз путање који формира секвенцу од елемента **запослени** добијеног из XML документа смештеног у атрибуту **инфо** табеле **запослени**. Наредба **for** пролази кроз све елементе из секвенце и у свакој итерацији, променљивој $\$i$ се додељује члан секвенце, односно елемент **запослени** из секвенце.
Следећа наредба у изразу је наредба **let** која променљивој $\$j$ додељује резултат израза путање $\$i/ime$. Односно, пошто променљива $\$i$ садржи елемент **запослени** тренутне итерације, изразом путање $\$i/ime$ формира се секвенца са именом за тренутног запосленог и та секвенца се додељује променљивој $\$j$.
Наредба **where** вреднује наведени услов $\$i/adresa/grad = "Beograd"$ и тај услов ће бити означен као тачан уколико, за тренутног запосленог у итерацији, у елементу **град** стоји да је из Београда. У супротном, услов ће бити означен као нетачан и елемент у тренутној итерацији ће бити одбачен из даљег разматрања у овом XQuery упиту.
За сваки елемент који је задовољио услове из наредбе **where**, наредба **return** формира резултат. Ипак, пре него што наредба **return** врати коначан резултат упита, потребно је уредити у опадајућем редоследу све елементе коју су преостали након филтрирања наредбом **where** пошто је у изразу наведена и опциона наредба **order by**. Коначно, наредбом **return** крајњем кориснику се враћа променљива $\$j$ који садржи сортирана имена запослених.

4.4 Конструктор изрази

Конструктор изрази служе за формирање целих XML докумената или само делова документа у оквиру упита. Формирање XML података у оквиру XQuery језика је једноставно, довољно је користити жељене XML ознаке у оквиру упита и као резултат се могу добити подаци у оквиру XML структуре или цели XML документи. Овакав метод формирања XML података назива се директно формирање XML-а и врши се помоћу директних конструктора.

Ретко када се јавља потреба да се формирају елементи чије су вредности унапред познате. Обично вредности елемената треба израчунати динамички, током извршавања упита. Да би то било остварио, XML елементи морају садржати XQuery променљиве или неки други вид израза помоћу којег се динамички могу добити тражене

вредности. Пример 4.12 показује како се за запосленог чија ја вредност атрибута 1006 може добити нови XML документ које се састоји само од елемената **име** и **град**.

```
XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/@id = 1006
return <info>
        <info_ime>{$i/ime/text()}</info_ime>
        <info_grad>{$i/adresa/grad/text()}</info_grad>
       </info>;

Output:
<info>
    <info_ime>Nevena Mijatovic</info_ime>
    <info_grad>Nis</info_grad>
</info>
```

Пример 4.12: Формирање новог XML документа

Треба приметити да имена елемената у новом документу нису имена која су сачувана у XML документу смештеног у табели **запослени** већ су као резултат добијена нова имена елемената која су дефинисана у оквиру самог упита. Вредности у оквиру новоформираних елемената су добијене динамичким путем односно, приликом формирања упита није наведена ниска карактера већ XPath израз који израчунава вредности за нове елементе.

Уколико је примењен овакав приступ (вредности елемената се добијају динамичким путем) обавезно се XPath израз мора навести у оквиру витичастих заграда. У супротном, XPath израз ће бити посматран као ниска карактера и тако ће бити враћен у коначном резултату. Резултат који се добија у случају да се у примеру 4.12 изоставе витичасте заграде приликом формирања елемента **инфо_град**:

```
<info>
    <info_ime>Nevena Mijatovic</info_ime>
    <info_grad>$i/adresa/grad/text()</info_grad>
</info>
```

У претходном примеру у оквиру XPath израза коришћена је опција /*text*() помоћу које је добијена само вредност траженог елемента без XML ознака. У случају да се у новом документу, који се добија као резултат упита, желе задржати имена елемената из оригиналног XML документа, може се изоставити опција /*text*() из XPath израза. Таквим приступом добија се резултат као у примеру 4.13.

```
XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/@id = 1006
return <info>
        {$i/ime}{$i/adresa/grad}
       </info>;

Output:
1
--------------------------------
<info>
    <ime>Nevena Mijatovic</ime>
    <grad>Nis</grad>
</info>
```

Пример 4.13: Формирање новог XML документа са оригиналним елементима

Формирање XML података помоћу условних израза

Употребом условног израза у оквиру XQuery упита могуће је формирати специфичне XML ознаке у зависности од вредности предиката. Такав је случај уколико је, као у примеру 4.14, потребно формирати елемент **инфо** такав да као атрибут поседује идентификациони број запосленог и да буде родитељ елементу чија је вредност број телефона запосленог са тим идентификационим бројем. Додатан услов је да име тог новог елемента који садржи вредност броја телефона зависи од типа телефона оригиналног елмента **телефон**. На пример, уколико је атрибут **тип**, елемента **телефон** у оригиналном XML документу имао вредност *"приватни"* онда је за резултат потребно добити нови елемент са именом **телефон_приватни**. То се може постићи коришћењем условног, односно *if-then-else*, израза.

```
XQuery:
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni/telefon
return <info>
          {$i/../@id}
          {if ($i/@tip="poslovni")
            then <telefon_poslovni>{$i/text()}</telefon_poslovni>
            else if ($i/@tip="privatni")
                then <telefon_privatni>{$i/text()}</telefon_privatni>
                else if ($i/@tip="kucni")
                    then <telefon_kucni>{$i/text()}</telefon_kucni>
                    else <telefon>{$i/text()}</telefon>
          }
       </info>;

Output:
1
--------------------------------------------------------------------------
<info id="1005"><telefon_poslovni>060-425-443</telefon_poslovni></info>
<info id="1005"><telefon_privatni>064-546-6121</telefon_privatni></info>
<info id="1006"><telefon_poslovni>066-335-2243</telefon_poslovni></info>
<info id="1006"><telefon_privatni>060-116-7713</telefon_privatni></info>
<info id="1007"><telefon_poslovni>060-224-8838</telefon_poslovni></info>
<info id="1007"><telefon_privatni>060-761-5455</telefon_privatni></info>
<info id="1007"><telefon_kucni>011-2344-718</telefon_kucni></info>
```

Пример 4.14: Формирање XML елемената коришћењем if-then-else израза

У претходном примеру обавезно је коришћење угњеждених *if-then-else* израза пошто у XQuery језику није могуће користити *elseif* или *case* изразе. Такође, важно је напоменути да уколико је потребно формирати нови елемент који поседује атрибут, израз за формирање атрибута мора бити наведен одмах након ознака и пре било ког елемента. Тачније, у примеру 4.14, израз $\$i/../@id$ формира атрибут **ид** у склопу елемента **инфо** и из тог разлога, у XQuery упиту тај израз је наведен одмах након почетне ознаке елемента **инфо**. Када би се израз за формирање атрибута навео после израза за формирање елемента добила би се грешка:

```
SQL16015N  An element constructor contains an attribute node named "id"
   that follows an XQuery node that is not an attribute node.
```

Формирање XML докумената

До сада су обрађени случајеви када је било довољно извући само одређене елементе из оригиналног XML документа док при свакодневном коришћењу корисници имају потребу и за формирањем читавих добро дефинисаних XML докумената на основу података из оригиналних XML докумената. Најчешће је то случај приликом потребе за добијањем разних ивештаја. На пример, потребно је направити извештај за сваког запосленог из Београда који ће укључити контакт информације, без кућног броја телефона, с тим да име запосленог мора бити у виду атрибута како се не би понављало за сваки број телефона. Наведени проблем могуће је решити тако што је потребно формирати елементе на три нивоа. За почетак потребан је основни елемент **инфо** који ће садржати све остале елементе. На нивоу испод, као директан потомак основног елемента, може се формирати елемент **контакт_инфо** који ће садржати име запосленог као вредност атрибута док ће потомци тог елемента, односно елементи на трећем нивоу, бити сви елементи чија ће вредност бити број телефона тог запосленог. Пример 4.15 описује наведено решење кроз XQuery упит.

```
XQuery:
xquery
<info>{
    for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
    where $i/adresa/grad = "Beograd"
    return <kontakt_info ime="{$i/ime/text()}">
                {for $p in $i/telefon
                where $p/@tip != "kucni"
                return <telefon>{$p/text()}</telefon>
                }
            </kontakt_info>
      }
</info>;
```

```
Output:
1
-----------------------------------------
<info>
    <kontakt_info ime="Aleksandar Petrovic">
        <telefon>060-425-443</telefon>
        <telefon>064-546-6121</telefon>
    </kontakt_info>
    <kontakt_info ime="Ivana Stefanovic">
        <telefon>060-224-8838</telefon>
        <telefon>060-761-5455</telefon>
    </kontakt_info>
</info>
```

Пример 4.15: Формирање XML документа

На почетку упита потребно је формирати основни елемент **инфо** како би се испунио први услов да коначан резултат буде у форми једног документа а не да се направи посебан документ за сваког запосленог. Након тога употребљен је први FLWOR израз где се са наредбом **for** пролази кроз све запослене из XML документа а са наредбом **where** проверава да ли је тренутни запослени из Београда и уколико није, искључује га из даље обраде. У супротном, када је услов испуњен, наредба **return** формира елемент **контакт_инфо** који садржи атрибут **име**. Добијање вредности за тражени атрибут постиже се једноставним изразом путање $\$i/ime/text()$ где се за тренутног запосленог узима текстуална вредност елемента **име**. Помоћу унутрашњег FLWOR израза омогућава се да сваки елемент **контакт_инфо** има онолико директних потомака колико тренутни запослени има бројева телефона који нису кућни. То се постиже тако што у унутрашњем FLWOR изразу наредба **for** пролази кроз све елементе **телефон** за тренутног запосленог а наредбом **where** се испитује да ли тренутно испитивани телефон није кућни. Уколико је услов нетачан, одностно телефон јесте кућни, он се одбацује док се у супротном, у оквиру наредбе **return** формира елемент **телефон** са вредношћу тренутног телефона.

Треба нагласити да се оба FLWOR израза налазе у оквиру витичастих заграда како би се XQuery преводиоцу показало да наведене изразе у оквиру елемента **инфо**, односно **контакт_инфо**, не треба посматрати као ниску карактера већ да их треба израчунати.

Глава 5

SQL/XML

5.1 Увод

Последњих неколико деценија подаци су чувани у релационим базама података а SQL језик се користио за претраживање и добијање резултата. Ипак, како се проширила употреба XML језика за репрезентацију и складиштење података тако су се и развили нови језици попут XPath или XQuery језика коју су олакшавали кориснику рад са XML подацима. С друге стране, доста људи се запитало зашто претварати све податке у XML и прилагођавати све апликације XQuery језику, зашто једноставно не прилагодити SQL да може да ради и са XML подацима. До те идеје се дошло зато што су релационе базе података и SQL језик прошли тест времена, годинама уназад прилагођавали су се новим потребама корисника и развијали тако да прихвате нове технологије па зашто онда то не учинити и са XML-ом. Пошто то није нимало једноставан задатак, формирана је група људи, названа SQLX, која је преузела на себе унапређење релационих база података и SQL језика. Група SQLX имала је на уму следеће планове [11]:

- извршавање упита над релационим подацима и добијање резултата у XML формату
- складиштење и управљање XML подацима у релационој бази на природан начин
- пресликавање релационих података у XML и обратно

- извршавање упита над XML подацима и постојање могућности да се добијени резултати прикажу и у релационом и у XML формату

Као резултат рада SQLX групе, SQL језику додата је нова функционалност названа SQL/XML која омогућава коришћење XPath или XQuery језика заједно са SQL језиком. По *SQL:2003* и *SQL:2006* стандарду списак SQL/XML функционалности је следећи:

- XML тип података је регуларан SQL тип података као што су то на пример INTEGER или CHAR.
- подаци XML типа могу се уз помоћ функција превести у неки од стандардних, не-XML, типова података. Те функције су XMLSERIALIZE, XMLPARSE и XMLCAST.
- за проверу исправности XML схеме користи се функција XMLVALIDATE док се предикат IS VALIDATED користи за проверу исправности једног дела или целог XML документа.
- формирање нових XML докумената је могуће уз помоћ функција XMLELEMENT, XMLATTRIBUTES и XMLAGG.
- функције које омогућавају да се у оквиру SQL упита користе XPath или XQuery изрази су XMLQUERY, XMLTABLE и XMLEXISTS.

Функције које се најчешће користе су:

- XMLCAST - преводи XML вредности у SQL тип података
- XMLQUERY - користи се у SELECT делу SQL упита за добијање вредности из XML документа
- XMLTABLE - користи се у FROM делу SQL упита за издвајање вредности из XML документа и њихово враћање као да се ради о подацима релационог типа
- XMLEXISTS - користи се у WHERE делу SQL упита уз помоћ које се филтрирају подаци XML типа

5.2 Коришћење функције XMLQUERY

Најједноставнији начин враћања XML података помоћу SQL језика је уколико се као резултат враћа цео XML документ. Наравно, упит се може проширити и WHERE условом (пример 5.1) да би се као резултат добио само одређени XML документ али то и даље није оно што је обично потребно корисницима. Најчешће се корисници сусрећу са захтевима да је као резултат потребно вратити само одређене атрибуте, елементе или делове XML документа. У том случају могу да се користе неке функција које су наведене у уводу овог поглавља.

```
SQL:
SELECT info
FROM zaposleni
WHERE id = 1007;

Output:
INFO
--------------------------------------------------
<zaposleni id="1007">
    <ime>Ivana Stefanovic</ime>
    <adresa drzava="Srbija">
        <grad>Beograd</grad>
        <ulica>Kneza Milosa 11</ulica>
    </adresa>
    <telefon tip="poslovni">060-224-8838</telefon>
    <telefon tip="privatni">060-761-5455</telefon>
    <telefon tip="kucni">011-2344-718</telefon>
</zaposleni>
```

Пример 5.1: Проналажење једног XML документа из табеле

Белине у резултату претходног примера су ручно додате како би резултат био разумљивији за читаоца овог рада док се у пракси резултат добија без белина. Уколико корисник има потребу да добије списак имена свих запослених из XML документа може користити функцију XMLQUERY у SELECT делу SQL упита (пример 5.2). У скаларној функцији XMLQUERY као аргумент се наводи XPath израз у коме се мора нагласити над којим атрибутом табеле XML типа се извршава. У табели **запослени**

постоји само један такав атрибут (атрибут **инфо**) тако да се он наводи као почетак XPath израза. Треба напоменути да референца, на који се атрибут табеле односи XPath израз, мора бити наведена великим словима као и да мора почињати знаком **$**.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime')
FROM zaposleni;

Output:
<ime>Aleksandar Petrovic</ime>
<ime>Nevena Mijatovic</ime>
<ime>Ivana Stefanovic</ime>
```

Пример 5.2: Проналажење једног елемента из сваког XML документа из табеле

Пример 5.2 је само један од начина како се може извршити референца на атрибут табеле XML типа у функцији XMLQUERY. Други начин је да се у склопу XPath израза формира алијас за тражени атрибут табеле и да се тај алијас проследи као референца на почетку XPath израза. Такав начин референцирања наведен је у примеру 5.3.

```
SQL/XML:
SELECT XMLQUERY('$i/zaposleni/ime' PASSING info AS "i")
FROM zaposleni;

Output:
<ime>Aleksandar Petrovic</ime>
<ime>Nevena Mijatovic</ime>
<ime>Ivana Stefanovic</ime>
```

Пример 5.3: Коришћење алијаса приликом референцирања

Претходни примери у резултату имају и XML ознаке заједно са текстуалном вредношћу елемента али то не мора увек бити случај. Уколико постоји потреба да се врати искључиво текстуална вредност траженог елемента то се може урадити додавањем text() опције у XPath израз (пример 5.4).

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()')
FROM zaposleni;
```

```
Output:
Aleksandar Petrovic
Nevena Mijatovic
Ivana Stefanovic
```

Пример 5.4: Добијање текстуалне вредности траженог елемента

Још један начин за уклањање XML ознака из добијених резултата је коришћењем функције XMLCAST. Наиме, све што је потребно урадити је ставити SQL/XML упит као аргумент функције XMLCAST и навести тип података у који се добијени резултат жели превести. Као што се може приметити, пример 5.5 враћа идентичан резултат као и пример 5.4 само што је сада резултат типа VARCHAR(25) а не XML типа.

```
SQL/XML:
SELECT XMLCAST(XMLQUERY('$INFO/zaposleni/ime') AS VARCHAR(25))
FROM zaposleni;

Output:
Aleksandar Petrovic
Nevena Mijatovic
Ivana Stefanovic
```

Пример 5.5: Употреба функције XMLCAST

У свим претходним примерима као резултат се тражио искључиво један атрибут табеле али то није чест случај у свакодневном коришћењу. Обично корисници имају потребу да из XML документа добију податке за већи број елемената а не само за један елемент. То се може постићи коришћењем функције XMLQUERY више пута у SELECT делу упита где ће се свака фунцкија XMLQUERY одвајати зарезом од следеће функције (као што је наведено у примеру 5.6). Коришћење *as ime* иза функције XMLQUERY је стандардна SQL функционалност која служи за преименовање назива атрибута табеле у који се смештају резултати. У супротном, називи атрибута ће имати генеричка имена попут 1, 2, 3.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()') AS ime,
       XMLQUERY('$INFO/zaposleni/adresa/ulica/text()') AS ulica,
       XMLQUERY('$INFO/zaposleni/adresa/grad/text()') AS grad
FROM zaposleni;
```

```
Output:
IME                    ULICA                GRAD
-------------------    ----------------     -------
Aleksandar Petrovic    Ruzveltova 31        Beograd
Nevena Mijatovic       Kneza Milosa 77b     Nis
Ivana Stefanovic       Kneza Milosa 11      Beograd
```

Пример 5.6: Добијење вредности за више од једног елемента из XML документа

Као што је већ раније наведено, једна од великих предности XML-а је што се врло лако могу додати нови елементи. Тако на пример, сваки запослени може имати неколико бројева телефона и по потреби, без проблема, може да му се додели нови број а да се не наруши добро дефинисана структура XML документа. Из тог разлога, поставља се питање како ће се ти елементи вратити у резултату. Пример 5.7 за сваког запосленог враћа његово име и све бројеве телефона. Треба напоменути да се сви бројеви телефона враћају као једна ниска карактера, без икаквих белина између два броја. У примеру испод, белине су убачене због лакшег прегледа добијених резултата.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()'),
       XMLQUERY('$INFO/zaposleni/telefon')
FROM zaposleni;

Output:
 1                     2
-------------------    ----------------------------------------------
Aleksandar Petrovic    <telefon tip="poslovni">060-425-443</telefon>
                       <telefon tip="privatni">064-546-6121</telefon>
Nevena Mijatovic       <telefon tip="poslovni">066-335-2243</telefon>
                       <telefon tip="privatni">060-116-7713</telefon>
Ivana Stefanovic       <telefon tip="poslovni">060-224-8838</telefon>
                       <telefon tip="privatni">060-761-5455</telefon>
                       <telefon tip="kucni">011-2344-718</telefon>
```

Пример 5.7: Добијање вредности понављајућих елемената из XML документа

Пример 5.8 се разликује од претходног примера по томе да је добијени резултат без XML ознака и ту се лепо види да су добијене текстуалне вредности елемента **телефон** само надовезане једна на другу.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()'),
       XMLQUERY('$INFO/zaposleni/telefon/text()')
FROM zaposleni;

Output:
1                     2
-------------------   ------------------------------------
Aleksandar Petrovic   060-425-443064-546-6121
Nevena Mijatovic      066-335-2243060-116-7713
Ivana Stefanovic      060-224-8838060-761-5455011-2344-718
```

Пример 5.8: Добијање вредности понављајућих елемената без XML ознака

5.3 Коришћење функције XMLTABLE

Овакав вид резултата, као у примеру 5.8, обично није од велике користи за корисника па се због тога избегава коришћење функције XMLQUERY када је потребно вратити вредности понављајућих елемената из XML документа. Споменути проблеми се решавају коришћењем функције XMLTABLE. За почетак, у примеру 5.9, биће објашњено како функција XMLTABLE враћа резултате док ће у каснијим примерима бити приказано како се помоћу функције XMLTABLE разрешавају проблеми коју су приказани у примеру 5.8.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ID      INTEGER         PATH '@id',
        ime     VARCHAR(30)     PATH 'ime',
        ulica   VARCHAR(40)     PATH 'adresa/ulica',
```

```
        grad    VARCHAR(20)     PATH 'adresa/grad') AS T;

Output:
ID      IME                     ULICA                   GRAD
----    --------------------    ----------------        -------
1005    Aleksandar Petrovic     Ruzveltova 31           Beograd
1006    Nevena Mijatovic        Kneza Milosa 77b        Nis
1007    Ivana Stefanovic        Kneza Milosa 11         Beograd
```

Пример 5.9: Коришћење функције XMLTABLE

Функција XMLTABLE се састоји из два дела. Први део генерише редове који се враћају као резултат док се у другом делу дефинише који ће се подаци тачно вратити. Прецизније, из примера 5.9 први део функције XMLTABLE је XPath израз **'$INFO/zaposleni'** који се извршава над сваким XML документом наведеним у табели **запослени** и као резултат враћа један или више редова за сваки документ. Број редова који функција XMLTABLE враћа као коначан резултат одређен је бројем редова који враћа наведени XPath израз. Други део функције XMLTABLE је такозвани *columns* израз који врши трансформацију XML података у релациони тип података. Сваки унос у овом изразу састоји се од:

- назива атрибута табеле,
- типа атрибута табеле у који је потребно трансформисати податке,
- путање до елемента у XML документу чије је податке потребно трансформисати.

Треба напоменути да путања у *columns* изразу није апсолутна већ је релативна и то у односу на путању из првог дела функције XMLTABLE тј. XPath израза.

Као што је раније споменуто, коришћење функције XMLTABLE омогућава да се превазиђу проблеми који се јављају при коришћењу функције XMLQUERY као у примеру 5.8. Ипак, као резултат кода наведеног у примеру 5.10 добија се грешка.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
```

```
      COLUMNS
        ime          VARCHAR(30)     PATH 'ime',
        telefon      VARCHAR(20)     PATH 'telefon') AS T;

Output:
An expression of data type "( item(), item()+ )" cannot be used when the
   data type "VARCHAR_20" is expected in the context.
Error QName=err:XPTY0004.. SQLCODE=-16003, SQLSTATE=10507, DRIVER=4.18.60
```

Пример 5.10: Добијање грешке приликом употребе функције XMLTABLE

Разлог за грешку је тај што се као резултат добија више бројева телефона за сваког запосленог који треба да се трансформише у једну VARCHAR вредност, што наравно није могуће. Уколико би било потребно вратити само последње додат број телефона за сваког запосленог или на пример само приватни број телефона, тада би овакав приступ био могућ и не би се добила наведена грешка за резултат. Ипак, уколико постоји потреба да се врате све вредности, за то постоји неколико начина:

- Излистати све вредности у оквиру једног атрибута табеле

Пошто је потребно направити листу свих бројева телефона запослених, у примеру 5.11 узет је тип VARCHAR(200) за елемент **телефон** уместо типа VARCHAR(20) што је био случај у примеру 5.10 када се очекивала једна вредност. Функција string-join захтева два параметра, низ ниске карактера и карактер који ће се користити за раздвајање.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ime          VARCHAR(30)     PATH 'ime',
        telefon      VARCHAR(200)    PATH 'string-join(telefon/text(),",")
   ') AS T;

Output:
```

```
IME                  TELEFON
-------------------  ------------------------------------
Aleksandar Petrovic  060-425-443,064-546-6121
Nevena Mijatovic     066-335-2243,060-116-7713
Ivana Stefanovic     060-224-8838,060-761-5455,011-2344-718
```

Пример 5.11: Коришћење једног атрибута за понављајуће елементе

- Дефинисати засебан атрибут табеле за сваки број телефона

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ime          VARCHAR(30)     PATH 'ime',
        telefon1     VARCHAR(20)     PATH 'telefon[1]',
        telefon2     VARCHAR(20)     PATH 'telefon[2]',
        telefon3     VARCHAR(20)     PATH 'telefon[3]') AS T;

Output:
IME                  TELEFON1        TELEFON2        TELEFON3
-------------------  ------------    ------------    ------------
Aleksandar Petrovic  060-425-443     064-546-6121    NULL
Nevena Mijatovic     066-335-2243    060-116-7713    NULL
Ivana Stefanovic     060-224-8838    060-761-5455    011-2344-718
```

Пример 5.12: Коришћење засебних атрибута за понављајуће елементе

Очигледан недостатак овог приступа је да се унапред мора дефинисати коначан број атрибута табеле који за неког запосленог може бити непотребно велики што ће произвести велики број NULL вредности док је за другог запосленог тај број атрибута мали тако да неће бити могуће добити све његове бројеве телефона у резултату упита.

- Враћање сваког броја телефона у засебном реду

У овом случају, као што је показано у примеру 5.13, резултат се састоји од редова по броју телефона а не по запосленом. Из тог разлога, користи се другачији XPath израз приликом генерисања броја редова (**'$INFO/zaposleni/telefon'**). Мана овог приступа је што ће се име запослених понављати за сваки број телефона.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni/telefon'
      COLUMNS
        ime          VARCHAR(30)     PATH '../ime',
        telefon      VARCHAR(20)     PATH '.',
        tip          VARCHAR(20)     PATH '@tip') AS T;

Output:
IME                    TELEFON          TIP
--------------------   ------------     --------
Aleksandar Petrovic    060-425-443      poslovni
Aleksandar Petrovic    064-546-6121     privatni
Nevena Mijatovic       066-335-2243     poslovni
Nevena Mijatovic       060-116-7713     privatni
Ivana Stefanovic       060-224-8838     poslovni
Ivana Stefanovic       060-761-5455     privatni
Ivana Stefanovic       011-2344-718     kucni
```

Пример 5.13: Коришћење функције XMLTABLE над понављајућим елементима

На крају, из претходних примера се може закључити да функција XMLTABLE даје више могућности у односу на функцију XMLQUERY када се ради о добијању резултата над понављајућим елементима из XML документа. Ипак одговор на питање који је приступ најбоље користити није тако лако дати јер, као што се може видети из примера, сваки од приступа има своје предности и мане тако да остаје на кориснику да одлучи у зависности од конкретне ситуације.

5.3.1 Сортирање добијених резултата

Једна од ствари која раздваја податке XML типа од података који су неког стандардног SQL типа (као што су INTEGER, VARCHAR или DATE) је немогућност поређења два податка тог типа. Такође, сортирање резултата по атрибуту табеле XML типа није могуће баш из разлога што се процес сортирања своди на низ поређења а већ је речено да то није могуће урадити. Уколико је потребно сортирати резултат на основу података који се налазе у XML документу прво је потребно трансформисати жељене податке у неки од стандардних SQL типова података. Након тога, помоћу ORDER BY услова одредити по ком се тачно елементу врши сортирање и у каквом уређењу. Уколико се не наведе ниједна опција, подразумевано уређење је растуће. Пример 5.14 показује како се по имену може сортирати елементе из XML документа у опадајућем редоследу.

```
SQL/XML:
SELECT T.*
FROM zaposlen,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ID      INTEGER         PATH '@id',
        ime     VARCHAR(30)     PATH 'ime',
        grad    VARCHAR(20)     PATH 'adresa/grad') AS T
ORDER BY ime desc;

Output:
ID      IME                     GRAD
----    -------------------     -------
1006    Nevena Mijatovic        Nis
1007    Ivana Stefanovic        Beograd
1005    Aleksandar Petrovic     Beograd
```

Пример 5.14: Сортирање добијених резултата

5.3.2 Недостајуће вредности

Структура XML докумената је таква да може постојати доста опционих елемената тако да нису сви елементи присутни у свим XML документима. Због тога, пожељно је рећи нешто више шта се дешава када се наиђе на неки такав елемент у првом делу функције XMLTABLE, у делу који генерише редове, а шта у другом делу где се одређује који се подаци враћају.

Један од опционих елемената у примеру који је коришћен у овом поглављу је елемент **асистент**. Пример 5.15 показује шта се добија као резултат уколико је потребно вратити име асистента и његов број телефона.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni/asistent'
      COLUMNS
        ime          VARCHAR(30)     PATH 'ime',
        telefon      VARCHAR(20)     PATH 'telefon') AS T;

Output:
IME                    TELEFON
---------------        ------------
Stevan Markovic        063-114-5531
Ana Jeremic            062-334-3155
```

Пример 5.15: Употреба опционих елемената у XPath изразу

XPath израз **'$INFO/zaposleni/asistent'** у функцији XMLTABLE враћа по тачно један ред за сваки XML документ који садржи опциони елемент **асистент** док за XML документ који не садржи тражени елемент не враћа се ниједан ред у резултату.

Пример 5.16 илуструје издвајање имена запосленог и његовог асистента. У том случају, елемент асистент се користи у другом делу функције XMLTABLE (тзв. *columns* израз) и за сваки XML документ у којем не постоји елемент **асистент** враћа се празна секвенца која се у резултату аутоматски конвертује у NULL вредност.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ime_zaposlenog      VARCHAR(30)     PATH 'ime',
        ime_asistenta       VARCHAR(20)     PATH 'asistent/ime') AS T;

Output:
IME_ZAPOSLENOG          IME_ASISTENTA
------------------      --------------
Aleksandar Petrovic     Stevan Markovic
Nevena Mijatovic        Ana Jeremic
Ivana Stefanovic        NULL
```

Пример 5.16: Употреба опционих елемената у columns изразу

Коришћењем **default 'string'** опције пре PATH дела у функцији XMLTABLE, где *string* може бити било која ниска карактера, може се дефинисати подразумевана повратна вредност, уместо NULL, у случају добијања недостајућег елемента у резултату (пример 5.17).

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ime_zaposlenog      VARCHAR(30)     PATH 'ime',
        ime_asistenta       VARCHAR(20)     default 'ne postoji'
                                            PATH 'asistent/ime') AS T;
Output:
IME_ZAPOSLENOG          IME_ASISTENTA
------------------      --------------
Aleksandar Petrovic     Stevan Markovic
Nevena Mijatovic        Ana Jeremic
Ivana Stefanovic        ne postoji
```

Пример 5.17: Дефинисање подразумеване повратне вредности

5.3.3 Обрада грешака

Још једна ствар на коју треба обратити пажњу приликом писања упита је тип података дефинисан у другом делу функције XMLTABLE, а у који се желе трансформисати XML елементи. Наиме, уколико се елемент који има ненумеричку вредност покуша трансформисати у тип INTEGER тај упит ће се завршити са грешком. Слична ствар ће се догодити и уколико се покуша трансформисати ниска карактера дужине n у тип VARCHAR(m), где је $m < n$. Ипак, по најновијем SQL/XML стандарду, уколико дође до наведене ситуације не добија се грешка већ се за резултат добије трансформисана ниска карактера дужине m док ће остатак ниске бити одбачен. Да би се наведене грешке избегле потребно је приликом дефинисања путање у функцији XMLTABLE додати проверу компатибилности типа података и тако отклонити могућност да се као резултат упита добије нека SQL грешка. На примеру 5.18 је показано како се врше неке од провера.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
     XMLTABLE('$INFO/zaposleni'
      COLUMNS
        ime     INTEGER         PATH '(if (ime castable as xs:integer)
                                then ime else -1)',
        ulica   VARCHAR(15)     PATH 'adresa/ulica/
                                (if (string-length(.) <= 15)
                                then . else "Greska")',
        grad    VARCHAR(20)     PATH 'adresa/grad') AS T;

Output:
IME     ULICA                   GRAD
---     --------------          -------
 -1     Ruzveltova 31           Beograd
 -1     Greska                  Nis
 -1     Kneza Milosa 11         Beograd
```

Пример 5.18: Коришћење фунцкија за рад са грешкама

У претходном примеру намерно је направљена грешка да се елемент **име** из XML документа трансформише у тип података INTEGER. Када се не би вршила провера, да ли је могуће трансформисати елемент **име** у INTEGER, наведени пример би се завршио са грешком

```
The value "Aleksandar Petrovic" cannot be constructed as, or cast (using
   an implicit or explicit cast) to the data type "xs:int".
Error QName=err:FORG0001.. SQLCODE=-16061, SQLSTATE=10608, DRIVER=4.18.60
```

Овако, све вредности које се не могу трансформисати као INTEGER имаће вредност -1 а упит ће се успешно извршити. Друга провера је везана за дужину елемента **улица**. Пример 5.18 показује да уколико је дужина вредности елемента **улица** већа од 15 карактера, DB2 ће занемарити остатак и без икаквих упозорења и грешака у резултату вратити само првих 15 карактера из назива улице. Са оваквом провером, у резултату ће се добити ниска карактера која је наведена у *else* грани.

5.4 Коришћење функције XMLEXISTS

Примери који су до сада наведени у овом поглављу нису имали WHERE услов у упиту што значи да су резултати добијени из свих XML докумената из табеле **запослени**. Уколико поред атрибута табеле XML типа, табела садржи и додатне атрибуте, WHERE услов се може написати на традиционалан начин коришћењем само релационих предиката. Ипак, то није довољно. Потребно је имати могућност да се филтрирају резултати на основу вредности у XML документу. За такав вид филтрирања резултата потребно је користити XPath предикате о којима је више било речи у секцији 3.2.2. XPath предикат се користи тако што се наведе као аргумент функције XMLEXISTS у WHERE делу SQL/XML упита. Функција XMLEXISTS израчунава вредност наведеног XPath израза за сваки документ посебно тако да, уколико XPath израз не врати празан резултат, функција XMLEXISTS враћа TRUE и обрада документа се врши у резултујућем скупу. Уколи ипак XPath израз врати празан резултат, функција XMLEXISTS враћа FALSE и обрада документа се не врши у резултујућем скупу. Пример 5.19 ће послужити да понуди детаљније објашњење како се добијају коначни резултати коришћењем функције XMLEXISTS у WHERE услову упита.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni[adresa/grad = "Beograd"]');

Output:
1
-------------------
Aleksandar Petrovic
Ivana Stefanovic
```

Пример 5.19: Коришћење функције XMLEXISTS

За сваки XML документ у табели **запослени** тражи се, помоћу XPath израза, да ли постоји елемент **запослени** који има потомка таквог да елемент **град** има вредност "Београд". Уколико такав документ постоји, функција XMLEXISTS враћа TRUE и тај документ се чува у резултујућем скупу. Након тога функција XMLQUERY, из SELECT дела упита, као резултат враћа текстуалну вредност елемента **име** за сачувани документ из резултујућег скупа. Тако се редом пролази кроз све XML документе из табеле **запослени** и као резултат се враћа име запосленог уколико је из Београда. Уколико запослени није из Београда, резултат XPath израза за таквог запосленог ће бити празна секвенца и само у том случају функција XMLEXISTS ће вратити FALSE. Последица тога је да ће тај XML документ бити одбачен и неће бити сачуван у резултујућем скупу. Из разлога што се XML документ не налази у резултујућем скупу, функција XMLQUERY неће обрађивати тај документ па се и име запосленог, који није из Београда, не може добити у крајњем резултату.

Као аргумент функције XMLEXISTS се може јавити било који облик XPath предиката. Један од типова предиката је и структурни о којем је такође било речи у претходном поглављу. Тако се у примеру 5.20 проверава да ли постоји запослени који нема асистента и уколико постоји такав запослени, у неком XML документу, потребно је вратити његово име и ид број.

```
SQL/XML:
SELECT id, XMLQUERY('$INFO/zaposleni/ime/text()')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni[not(asistent)]');
```

```
Output:
ID      2
----    ---------------
1007    Ivana Stefanovic
```

Пример 5.20: Коришћење структурног предиката са негацијом

Следећа три примера враћају идентичне резултате али је приступ приликом добијања тих резултата различит. Први од њих (пример 5.21) показује како се функција XMLEXISTS може користити заједно са функцијом XMLTABLE. XPath израз у функцији XMLEXISTS проналази тачно један документ који задовољава наведени услов и само тај документ ће бити обрађен функцијом XMLTABLE. Тако да израз *@tip="poslovni"* из *columns* дела функције XMLTABLE не утиче на укупан број враћених докумената већ само одређује шта ће се од већ враћених докумената приказати у коначном резултату.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
    XMLTABLE('$INFO/zaposleni'
        COLUMNS
        ime     VARCHAR(20) PATH 'ime',
        telefon VARCHAR(12) PATH 'telefon[@tip="poslovni"]') AS T
WHERE XMLEXISTS('$INFO/zaposleni[@id = 1005]');

Output:
IME                   TELEFON
------------------    -----------
Aleksandar Petrovic   060-425-443
```

Пример 5.21: Употреба функције XMLTABLE са функцијом XMLEXISTS

Коришћење функције XMLEXISTS није обавезно да би се филтрирали резултати. То се такође може урадити коришћењем XPath израза у функцији XMLTABLE. Пример 5.22 показује како се без коришћења функције XMLEXISTS могу добити идентични резултати као и у примеру 5.21. Наиме, потребно је XPath израз, које је био коришћен у WHERE услову као аргумент функције XMLEXISTS, користити у првом делу функције XMLTABLE. Раније у овом поглављу речено је да се функција XMLTABLE

састоји из два дела где први део генерише редове који се враћају као резултат. Из тог разлога, коришћење XPath израза у том делу филтрира број враћених редова (докумената) на исти начин на који би се то радило у функцији XMLEXISTS.

```
SQL/XML:
SELECT T.*
FROM zaposleni,
    XMLTABLE('$INFO/zaposleni[@id = 1005]'
        COLUMNS
        ime     VARCHAR(20) PATH 'ime',
        telefon VARCHAR(12) PATH 'telefon[@tip="poslovni"]') AS T;

Output:
IME                  TELEFON
-------------------  -----------
Aleksandar Petrovic  060-425-443
```

Пример 5.22: Изостављање функције XMLEXISTS

Што се перформанси тиче, нема велике разлике између два описана приступа тако да се на основу тога не може дати предност једном или другом приступу. Ипак, што се конзистентности упита и његове разумљивости тиче, препоручује се коришћење првог приступа тј. коришћење WHERE услова и функције XMLEXISTS пошто је разумљивије имати критеријум по ком се филтрирају XML документи у WHERE услову него у функцији XMLTABLE.

Последњи приступ такође изостављa коришћење функције XMLEXISTS али има WHERE услов у SQL/XML упиту. У овом случају, врши се филтрирање по релационом атрибуту табеле који је добијен уз помоћ функције XMLTABLE. Наиме, у примеру 5.23 функција XMLTABLE враћа имена запослених у релационом атрибуту **име**, типа VARCHAR(20). Након тога, тај атрибут се користи у WHERE услову упита да би се из резултујућег скупа, као коначан резултат, вратили само они редови који имају вредност "Александар Петровић".

```
SQL/XML:
SELECT T.*
FROM zaposleni,
    XMLTABLE('$INFO/zaposleni'
```

```
        COLUMNS
        ime     VARCHAR(20) PATH 'ime',
        telefon VARCHAR(12) PATH 'telefon[@tip="poslovni"]') AS T
WHERE ime = 'Aleksandar Petrovic';

Output:
IME                    TELEFON
-------------------    -----------
Aleksandar Petrovic    060-425-443
```

Пример 5.23: Филтрирање релационих атрибута табеле генерисаних функцијом XMLTABLE

Уколико постоји потреба за филтрирањем резултата по више критеријума то се може урадити на два начина:

- коришћењем више XMLEXISTS функција са по једним условом
- коришћењем једне XMLEXISTS функције са више услова

У наставку је дат пример коришћења једне XMLEXISTS функције која садржи више услова по којима је потребно филтрирати XML документе. Оваквом приступу се даје предност у односу на приступ са коришћењем више XMLEXISTS функција.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/ime/text()')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni[adresa/@drzava = "Srbija"
                and adresa/grad = "Nis"]');

Output:
1
----------------
Nevena Mijatovic
```

Пример 5.24: Коришћење једне функције XMLEXISTS

5.5 Коришћење FLWOR израза у SQL/XML упиту

Као што је до сада показано у овом поглављу (Поглавље 5) комбинација језика SQL и XPath је веома заступљена у DB2 систему када је потребно вратити податке из базе који су XML типа. Додатна могућност коју поседује DB2 је та да језици SQL и XQuery нису међусобно искључиви већ се такође могу комбиновати. Штавише, коришћење комбинације језика SQL и XQuery представља решење уколико је проблем довољно комплексан да помоћу комбинације језика SQL и XPath није могуће решити проблем. У наставку (пример 5.25) је дат пример комбинације језика SQL и XQuery тачније, коришћење FLWOR израза у SQL/XML упиту [12].

```
XQuery:
SELECT XMLQUERY('for $i in $INFO/zaposleni/ime
                 return $i/text()')
FROM zaposleni
WHERE XMLEXISTS('let $i := $INFO/zaposleni
                 where $i/adresa/grad = "Beograd"
                 return $i');

Output:
1
--------------------
Aleksandar Petrovic
Ivana Stefanovic
```

Пример 5.25: Коришћење FLWOR израза у SQL/XML упиту

Наведени пример даје идентичан излаз као пример 5.19 из претходног поглавља који представља комбинацију језика SQL и XPath и за резултат враћа имена запослених који су из Београда. Као што је већ наведено у секцији 5.4, уколико функција XMLEXIST као аргумент добије празну секвенцу, у овом случају од FLWOR израза, као резултат враћа нетачно и тај ред се елиминише од даљег разматрања. У супротном, уколико се не добије празна секвенца, функција враћа тачно и тренутни документ се чува у резултујућем скупу. У примеру 5.25 FLWOR израз, у WHERE делу упита, додељује промењливој $\$i$ секвенцу која се састоји од од свих запослених из Београда. Уколико у XML документу нема таквих запослених, фунцкија XMLEXIST ће као аргумент добити празну секвенцу. У супротном, променљива $\$i$

ће садржати секвенцу чији су чланови сви подаци запослених који су из Београда што ће бити довољан услов да функција XMLEXIST врати тачно и вредност променљиве $\$i$ смести у резултујући скуп. Након тога FLWOR израз, из SELECT дела упита, наведен као аргумент функције XMLQUERY пролази кроз све чланове добијене секвенце и као коначан резултат кориснику враћа текстуалну вредност елемента **име**.

5.6 Формирање XML докумената

Формирање целих докумената се може постићи и кроз SQL/XML језик. Пример 5.26 показује како се помоћу SQL/XML језика могу добити исти резултати као и у примеру 4.15. SQL/XML упит користи подупит и функцију XMLAGG како би се формирала жељена структура новог XML документа. Први корак је да се у подупиту функцијом XMLEXISTS елиминишу сви запослени који нису из Београда.
Након тога, у унутрашњој SELECT наредби користи се FLWOR израз сличан унутрашњем FLWOR изразу из примера 4.15 уз помоћу којег се формира елемент **телефон** за сваки тип телефона који није кућни. Цео тај израз прослеђен је као аргумент функцији XMLQUERY уз помоћ које се формира елемент **контакт_инфо** који за директне потомке име елемент **телефон** са вредностима добијеним уз помоћ поменутог FLWOR израза. Такође, елемент **контакт_инфо** садржи и име запосленог као вредност атрибута.
Тако добијен резултат не би био пожељан пошто би се сваки елемент **контакт_инфо** добио у засебном реду а не у оквиру једног документа. Из тог разлога користи се функција XMLAGG којој је цела функција XMLQUERY прослеђена као аргумент. Сврха било које агрегатне функције у SQL језику је да од вредности из више редова као резултат врати једну вредност. Исто такво понашање има и фунцкија XMLAGG само за податке који су XML типа.
Функција XMLAGG од добијених елемената **контакт_инфо** формира једну секвенцу. Добијена секвенца је једна вредност у атрибуту табеле XML типа којој је дато име **контакт** како би се могла извршити референца на ту секвенцу. Спољашња SELECT наредба користи направљену секвенцу **контакт** помоћу које се добија садржај за основни елемент **инфо**.

```
SQL/XML:
SELECT XMLQUERY('<info>{$KONTAKT}</info>')
FROM (
    SELECT XMLAGG(
        XMLQUERY('<kontakt_info ime="{$INFO/zaposleni/ime}">
                    {for $p in $INFO/zaposleni/telefon
                     where $p/@tip != "kucni"
                     return <telefon>{$p/text()}</telefon>
                    }
                  </kontakt_info>') ) as kontakt
    FROM ZAPOSLENI
    WHERE XMLEXISTS('$INFO/zaposleni[adresa/grad = "Beograd"]'));
```

Пример 5.26: Формирање XML документа користећи SQL/XML језик

5.7 Грешке при писању SQL/XML упита

Веома честа грешка приликом писања SQL/XML упита је изостављање угластих заграда приликом писања XPath предиката у оквиру функције XMLEXISTS. Овако написан услов, као у примеру 5.27, не означава поређење вредности елемента **град** са ниском карактера "Београд" већ представља Буловски израз $A = B$ који може вратити само TRUE или FALSE. Као што је већ раније објашњено у овом поглављу, уколико функција XMLEXISTS добије празну секвенцу она одбацује тренутни XML документ а уколико добије непразну повратну вредност онда чува тренутни XML документ у резултујућем скупу. Пошто повратна вредност оваквог XPath израза никада не може бити празна секвенца, већ само TRUE или FALSE, то ће за функцију XMLEXISTS значити да сваки XML документ мора сачувати у резултујућем скупу. Такво понашање никада не елиминише ниједан ред из табеле и овако написан израз ће увек вратити све редове из табеле над којом се овакав упит извршава.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/adresa/grad')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni/adresa/grad = "Beograd"');
```

```
Output:
1
--------------------
<grad>Beograd</grad>
<grad>Nis</grad>
<grad>Beograd</grad>
```

Пример 5.27: Изостављање угластих заграда у XPath предикату

Тип грешке који се такође може доста често срести приликом писања SQL/XML упита је када се XPath предикат, који има функцију да филтрира редове који се враћају у резултујућем скупу, користи у SELECT делу упита у оквиру функције XMLQUERY уместо у WHERE делу упита у оквиру функције XMLEXISTS. Тако написан упит увек враћа све редове из табеле што може бити озбиљан проблем уколико табела има велики број редова. На примеру 5.28 је показано како SQL преводилац разрешава претходно описану ситуацију.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni[adresa/grad = "Beograd"]/ime')
FROM zaposleni;

Output:
1
-----------------------------
<ime>Aleksandar Petrovic</ime>

<ime>Ivana Stefanovic</ime>
```

Пример 5.28: Коришћење XPath предиката у SELECT делу упита

Циљ упита је вратити имена свих запослених који су из Београда. Овако написан упит не укључује WHERE услов и никада се не врши филтрирање редова који се враћају у резултату већ функција XMLQUERY прави резултат за сваки XML документ из табеле. Оно што се дешава при приказивању резултата кориснику је да за све запослене који су из Београда враћа њихово име док се за остале се враћа празна секвенца. Из угла перформанси, наведени пример није проблематичан али уколико би се овакав упит извршио над табелом која има више милиона редова (XML докумената) то би представљало озбиљан проблем. Уколико је поента упита на неки

начин филтрирати редове из табеле препорука је да се то ради у WHERE делу упита коришћењем XPath предикат у склопу функције XMLEXISTS. Пример 5.29 показује исправно написан упит који филтрира редове и као резултат не враћа све XML документе већ само оне који задовољавају наведени услов.

```
SQL/XML:
SELECT XMLQUERY('$INFO/zaposleni/telefon[@tip = "privatni"]')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni[@id = 1006]');

Output:
1
--------------------------------------------
<telefon tip="privatni">060-116-7713</telefon>
```

Пример 5.29: Исправан начин филтрирања XML докумената

Функција XMLEXISTS у претходном примеру враћа тачно онај XML документ којем је атрибут **ид** једнак 1006. Само над тим враћеним XML документом извршава се функција XMLQUERY која обезбеђује да крајњи резултат приказан кориснику садржи само приватни број телефона а не списак свих телефона из документа враћеног функцијом XMLEXISTS.

Глава 6

Поређење SQL/XML и XQuery језика

У претходним поглављима дата је детаљна анализа SQL/XML и XQuery језика а идеја овог поглавља је да се наведе кратак осврт на SQL/XML и XQuery упите кроз један заједнички пример како би се на једном месту дала што боља и јаснија слика о употреби и начину функционисања SQL/XML и XQuery језика кроз примере. Пример кроз који ће се дати преглед SQL/XML и XQuery језика је тај да је потребно из XML докумената смештених у бази података као резултат вратити списак запослених чије је место пребивалишта Београд.

SQL/XML

```
SELECT XMLQUERY('$INFO/zaposleni/ime/text()')
FROM zaposleni
WHERE XMLEXISTS('$INFO/zaposleni[adresa/grad = "Beograd"]');

Output:
Aleksandar Petrovic
Ivana Stefanovic
```

Пример 6.1: SQL/XML упит

За сваки XML документ у табели **запослени** тражи се, помоћу XPath израза, да ли постоји елемент **запослени** који има потомка таквог да елемент **град** има вредност "Београд". Уколико такав документ постоји, функција XMLEXISTS враћа тачно и тај документ се чува у резултујућем скупу. Након тога функција XMLQUERY,

из SELECT дела упита, као резултат враћа текстуалну вредност елемента **име** за сачувани документ из резултујућег скупа. Тако се редом пролази кроз све XML документе из табеле **запослени** и као резултат враћа име запосленог уколико је из Београда. Уколико запослени није из Београда, резултат XPath израз за таквог запосленог ће бити празна секвенца и само у том случају функција XMLEXISTS ће вратити нетачно. Као последица тога XML документ ће бити одбачен и неће бити сачуван у резултујућем скупу. Из разлога што се XML документ не налази у резултујућем скупу, функција XMLQUERY неће обрађивати тај документ па се и име запосленог, који није из Београда, не добија у крајњем резултату.

XQuery

```
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
let $j := $i/ime/text()
where $i/adresa/grad = "Beograd"
return $j;
```

Пример 6.2: XQuery упит са FLWOR изразом

Израз који је у овом примеру коришћен у наредби **for** је израз путање који формира секвенцу од елемента **запослени** добијеног из XML документа смештеног у атрибуту **инфо** табеле **запослени**. Наредба **for** пролази кроз све елементе из секвенце и у свакој итерацији променљивој $\$i$ додељује члан секвенце, односно елемент **запослени** из секвенце.

Следећа наредба у изразу је наредба **let** која променљивој $\$j$ додељује резултат израза путање $\$i/ime/text()$. Односно, пошто променљива $\$i$ садржи елемент **запослени** тренутне итерације, изразом путање $\$i/ime/text()$ формира се секвенца са именом за тренутног запосленог и та секвенца се додељује променљивој $\$j$.

Наредба **where** вреднује наведени услов $\$i/adresa/grad = "Beograd"$ и тај услов ће бити означен као тачан уколико, за тренутног запосленог у итерацији, у елементу **град** стоји да је из Београда. У супротном, услов ће бити означен као нетачан и елемент у тренутној итерацији ће бити одбачен из даљег разматрања у овом XQuery упиту.

За сваки елемент који је задовољио услове из наредбе **where**, наредбом **return** крајњем кориснику се враћа променљива $\$j$ који садржи имена запослених чије је место пребивалишта Београд.

Комбинација SQL/XML и XQuery језика

```
SELECT XMLQUERY('for $i in $INFO/zaposleni/ime
                 return $i/text()')
FROM zaposleni
WHERE XMLEXISTS('let $i := $INFO/zaposleni
                 where $i/adresa/grad = "Beograd"
                 return $i');
```

Пример 6.3: Коришћење FLWOR израза у SQL/XML упиту

Као што је већ споменуто, уколико функција XMLEXIST као аргумент добије празну секвенцу, у овом случају од FLWOR израза, као резултат враћа нетачно и тај ред се елиминише из даљег разматрања. У супротном, уколико се не добије празна секвенца, функција враћа тачно и тренутни документ се чува у резултујућем скупу. У примеру 6.3 FLWOR израз, у WHERE делу упита, додељује променљивој $\$i$ секвенцу која се састоји од од свих запослених из Београда. Уколико у XML документу нема таквих запослених, фунцкија XMLEXIST ће као аргумент добити празну секвенцу. У супротном, променљива $\$i$ ће садржати секвенцу чији су чланови сви подаци запослених који су из Београда и као таква, биће довољан услов да функција XMLEXIST врати тачно и вредност променљиве $\$i$ смести у резултујући скуп. Након тога други FLWOR израз, наведен као аргумент функције XMLQUERY у SELECT дела упита, пролази кроз све чланове добијене секвенце и као коначан резултат кориснику враћа текстуалну вредност елемента **име**.

Глава 7

Ажурирање и измена XML докумената

У претходним поглављима овог рада фокус је био на томе како и на које све начине је могуће добити жељене информације из XML докумената смештених у релациону базу података. То представља веома значајан део за целокупну причу о XML-у у релационим базама података али ипак то нису једине операције које могу да се примене на XML документе.
У овом поглављу, фокус ће бити на могућности ажурирања XML докумената из базе као и мењање само структуре документа уколико се то захтева од крајњег корисника. Овај део рада се може поделити на две целине:

- замена постојећих XML докумената новим
- ажурирање постојећих XML докумената

У првом делу ће бити речи како се врши замена целих XML докумената из базе са новим документима који имају ажуриране вредности. У другом делу поглавља фокус ће бити на томе да ће се оригинални XML документи задржавати у бази али ће њихов садржај или структура бити измењена. Неке од могућих операција над XML документом су:

- промена вредности елемента
- преименовање елемената у документу

- брисање елемената из документа
- додавање нових елемената у документ

Све наведене операције над елементима XML документа могу се извршити и над атрибутима.

7.1 Замена XML докумената

Уколико је потребно ажурирати табелу тако да се замени оригиналан XML документ новим XML документом, за потебе те операције, може се користити стандардан SQL UPDATE израз. На кориснику је да формира нови XML документ док ће UPDATE израз тај документ посматрати као обичан аргумент без сазнања шта се конкретно мења у односу на оригинални документ, да ли су све вредности елемената и атрибута потпуно нове или је промењена само једна вредност. Када се мењају XML документи, као и код већине UPDATE израза, од великог значаја је не заборавити и исправно дефинисати WHERE услов у оквиру израза како би се замена докумената извршила само за потребне редове у табели. Лошим дефинисањем или изостављањем WHERE услова велики број постојећих докумената из табеле ће бити замењен новим документом а то у већини случајева није оно што корисник жели.

UPDATE израз у примеру 7.1 замењује постојећи XML документ, сачуван у атрибуту **инфо** табеле **запослени**, са новим XML документом наведеним у SET делу израза. Наведена замена се извршава само за онај XML документ чији је релациони атрибут **ид** табеле **запослени** једнак 1006.

```
UPDATE zaposleni
SET info =
            '<?xml version="1.0" encoding="UTF-8" ?>
                <zaposleni id="1006">
                        <ime>Nevena Mijatovic</ime>
                        <adresa drzava="Srbija">
                                <grad>Nis</grad>
                                <ulica>Kneza Milosa 77b</ulica>
                        </adresa>
```

```
                        <telefon tip="poslovni">066-335-2243</telefon>
                        <telefon tip="privatni">060-116-7713</telefon>
                    <asistent>
                        <ime>Marko Petrovic</ime>
                        <telefon tip="poslovni">060-211-9825</telefon>
                    </asistent>
                </zaposleni>'
WHERE id = 1006;
```

Пример 7.1: Употреба UPDATE израза при замени XML документа

Како би проверили резултате UPDATE израза и да ли се стварно нови XML документ налази у бази, у примеру испод наведени су резултати упита који тражи податке о асистенту запосленог чији је релациони атрибут **ид** табеле **запослени** једнак 1006.

```
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/@id = 1006
return $i/asistent;

Output:
- pre UPDATE izraza
<asistent><
    ime>Ana Jeremic</ime>
    <telefon tip="poslovni">062-334-3155</telefon>
</asistent>

- posle UPDATE izraza
<asistent>
    <ime>Marko Petrovic</ime>
    <telefon tip="poslovni">060-211-9825</telefon>
</asistent>
```

Поред услова по релационом атрибуту из табеле **запослени**, као у примеру 7.1, постоји још неколико начина како се у WHERE делу упита може извршити одабир жељеног документа који је потребно заменити. Неки од тих начина су:

1. услов по вредности елемента из XML документа
2. услов по вредности атрибута из XML документа
3. комбинацијом релационих и XML услова

Сваки од наведених услова могуће је применити у примеру 7.1 уместо услова о релационом атрибуту из табеле **запослени** и добити исти резултат. Списак могућих WHERE услова је наведен у примеру 7.2.

```
(1) WHERE XMLEXISTS('$INFO/zaposleni[ime = "Nevena Mijatovic"]');

(2) WHERE XMLEXISTS('$INFO/zaposleni[@id = 1006]');

(3) WHERE XMLEXISTS('$INFO/zaposleni[ime = "Nevena Mijatovic"]') and id =
    1006;
```

Пример 7.2: Додатни типови WHERE услова

Резултат ће бити исти за све случајеве и исти документ ће бити замењен новим. Ипак треба напоменути да у првом случају, када је написан услов по вредности елемента **име** из XML документа, резултат ће бити исти као у примеру 7.1 само из разлога што у формираној табели **запослени** не постоје две или више особа чије је име Невена Мијатовић. У случају да постоји, нови документ би био замењен за све те случајеве и то је један од разлога зашто треба посебно обратити пажњу при писању WHERE услова како би се избегли овакви потенцијални проблеми. Тако дефинисан услов, као у првом случају, може се поправити додатним условом, као што је показано у трећем случају, где се услову по вредности елемента из XML документа за који се не гарантује да ће бити јединствен додаје услов по релационом атрибуту из табеле. Други WHERE услов из примера 7.2 врши претрагу по вредности атрибута из XML документа и пошто је вредност атрибута **ид** из XML документа јединствена и одговара вредности релационог атрибута **ид** табеле **запослени** резултат ће бити идентичан резултату из примера 7.1.

7.2 Ажурирање XML докумената

Замена целих XML докумената, о којој је било речи у претходној секцији овог поглавља, можда делује као сасвим одговарајуће решење када је потребно направити неке измене у документу али ипак, у великим системима и великим базама података овакав начин рада може бити изузетно скуп и имати огроман утицај на перформансе система. Наиме, таквим приступом и за најједноставнију операцију попут измене само једне вредности неког елемента или атрибута потребно је довући цео XML документ из базе података, изанализирати садржај тог документа, извршити потребне промене у самом документу и тек након тога извршити UPDATE израз како би се у бази података сачувала жељена промена. Како би се овај процес доста убрзао, DB2 систем од верзије 9.5 подржава *XQuery Update Facility*, стандардизовану надоградњу XQuery језика која омогућава корисницима да измене, избришу или додају појединачне елементе или атрибуте у оквиру XML документа.

Списак операција које су дефинисане овом надоградњом XQuery језика и које се могу извршити над XML чворовима, било да су то елементи или атрибути, су:

- промена вредности чвора
- замена чвора другим
- преименовање чвора
- брисање чвора
- додавање нових чворова на одређену локацију
- промена вредности више чворова у једном изразу
- измена више чворова у једном документу

Поменуте операције се могу извршити над XML документом користећи израз трансформисања (енг. transform expression) XQuery језика. Ови изрази могу почињати са кључном речју, која је опциона, **transform** и састоје се од три наредбе:

- copy - формира променљиву којој се додељује улазни XML документ из атрибута XML типа сачуваног у табели релационе базе података

- modify - извршава једну или више измена над променљивом дефинисаној у наредби copy
- return - формира коначан резултат израза трансформисања

Пошто XQuery језик препознаје разлику између малих и великих слова, обавезно је да све кључне речи, као и називи наредби, буду исписане малим словима. Наведени израз трансформисања, помоћу којег се врше жељене измене над XML документима, може се користити у оквиру SQL UPDATE израза али и у оквиру упита. Наиме, уколико се врши измена документа у оквиру упита, XML документ се прочита из табеле базе података и све наведене измене се врше у оквиру самог упита након чега се коначни резултати прослеђују крајњем кориснику. Овакав начин коришћења израза трансформисања оставља оригиналну верзију XML документа непромењену у бази података. Уколико се ипак израз трансформисања користи у оквиру UPDATE израза то значи да ће промена бити трајна и измењени подаци у оквиру XML документа биће сачувани у бази података. У наставку поглавља биће дат кратак осврт на разне типове израза трансформисања који се користе у свакодневном раду приликом ажурирања XML докумената из базе података.

7.2.1 Промена вредности чвора у документу

Најчешћи облик измене XML документа у пракси је ажурирање вредности неког елемента или атрибута. Пример 7.3 показује поступак како се помоћу UPDATE SQL израза може урадити измена XML документа уколико је потребно одређеном запосленом ажурирати место становања. Наиме, помоћу SET наредбе UPDATE израза атрибуту **инфо** табеле **запослени**, у којем се чувају XML документи, прослеђује се нови, ажуриран, XML документ за оног запосленог чији је атрибут **ид** једнак 1006.

```
UPDATE zaposleni
SET info = XMLQUERY('transform
            copy $tmp := $INFO
            modify do replace
                value of $tmp/zaposleni/adresa/ulica
                with "Cara Dusana 43"
            return $tmp ')
WHERE id = 1006;
```

```
Output:
- pre UPDATE izraza
<ulica>Kneza Milosa 77b</ulica>

- posle UPDATE izraza
<ulica>Cara Dusana 43</ulica>
```

Пример 7.3: Промена вредности елемента

Нови XML документ са ажурираним вредностима, који се додељује атрибуту **инфо**, добија се помоћу функције XMLQUERY која садржи израз трансформисања. Наредба *copy* израза трансформисања, променљивој $*tmp* додељује вредност атрибута **инфо** за траженог запосленог односно, смешта XML документ у дефинисану променљиву. Након тога, наредбом *modify* врше се потребне измене над том променљивом тако што се иза ниске *replace value of* наводи путања до елемента којем ће се променити вредности у XML документу а док се нова вредност за жељени елемент наводи после кључне речи *with*. Након наредбе *modify* промељива $*tmp* садржи ажурирану верзију XML документа и та променљива се као резултат израза трансформисања враћа наредбом *return*. Треба напоменути да је у претходном примеру дат најједноставнији облик израза трансформисања а који, уколико има потребе, може бити доста сложенији. Један такав израз дат је у примеру 7.4 где је потребно ажурирати вредности за више различитих врста чворова у једном XML документу. Односно, потребно је ажурирати све вредности о асистенту у XML докумнету за запосленог чији је атрибут **ид** једнак 1006. Елемент **асистент** има два директа потомка, елементе **име** и **телефон** док елемент **телефон** поседује атрибут **тип** и те три вредности потребно је ажурирати кроз један UPDATE израз. Поступак измене више вредности у једном UPDATE изразу разликује се од измене једне вредности само по томе што се у наредби *modify*, у оквиру заграда, наведе списак свих потребних измена у XML документу. Такође, у оквиру наредбе *modify*, потребно је да све наведене измене буду одвојене зарезом.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify
            (do replace
```

```
                value of $tmp/zaposleni/asistent/ime
                with "Jelena Rankovic" ,
            do replace
                value of $tmp/zaposleni/asistent/telefon
                with "064-787-9945" ,
            do replace
                value of $tmp/zaposleni/asistent/telefon/@tip
                with "privatni")
            return $tmp ')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<asistent>
    <ime>Marko Petrovic</ime>
    <telefon tip="poslovni">060-211-9825</telefon>
</asistent>

- posle UPDATE izraza
<asistent>
    <ime>Jelena Rankovic</ime>
    <telefon tip="privatni">064-787-9945</telefon>
</asistent>
```

Пример 7.4: Промена вредности више чворова

7.2.2 Замена чвора у документу

Пример 7.4 је показао како се одједном могу изменити вредности више чворова у XML документу. Да је елемент **асистент** имао више потомака а сваки од потомака један или више атрибута тај израз трансформисања би се веома брзо проширио и не би био погодан за писање и одржавање. Решење за тај потенцијалан проблем је директна замена чворова у XML документу уместо ажурирања свих вредности. На пример, уколико запослени промени место становања, једно од решења за ажурирање његових података је да се напише израз трансформисања који ће променити све

вредности у XML документу везане за адресу становања тог запосленог или се може написати израз трансформисања који ће заменити оригиналан елемент **адреса** новим. У склопу наредбе *modify* наредба *replace value of* мења само вредности наведеног чвора из XML документа док наредба *replace* брише оригиналан чвор и замењује га новим. Још једна од предности овог приступа је што структура новог чвора не мора бити идентична оригиналном чвору тако да се оваквим ажурирањем XML документа, поред измене вредности, могу додати нови чворови у структуру или изменити имена постојећим чворовима. Пример 7.5 показује како се за траженог запосленог, поред ажурирања вредности у XML документу додаје и нови елемент **поштански_број**.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do replace $tmp/zaposleni/adresa
                with <adresa drzava="Srbija">
                        <ulica>Karadjordjeva 144</ulica>
                        <grad>Novi Sad</grad>
                        <postanski_broj>21000</postanski_broj>
                    </adresa>
            return $tmp ')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<adresa drzava="Srbija">
    <grad>Nis</grad>
    <ulica>Cara Dusana 43</ulica>
</adresa>

- posle UPDATE izraza
<adresa drzava="Srbija">
    <ulica>Karadjordjeva 144</ulica>
    <grad>Novi Sad</grad>
    <postanski_broj>21000</postanski_broj>
</adresa>
```

Пример 7.5: Замена елемента

7.2.3 Преименовање чвора у документу

У претходној секцији (7.2.2) споменуто је да приликом замене чворова постоји и могућност промене назива чвора у XML документу. Ипак, у наредби *modify* израза трансформисања постоји посебна наредба чија је основна функција баш промена имена наведеног чвора. У примеру 7.6 приказана је употреба наредбе *rename* која име наведеног чвора у изразу путање замењује са новим именом. Такође, важно је напоменути да ново име чвора мора бити наведено под наводницима.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do rename
                $tmp/zaposleni/adresa/grad as "mesto"
            return $tmp ')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<adresa country="Srbija">
    <ulica>Karadjordjeva 144</ulica>
    <grad>Novi Sad</grad>
    <postanski_broj>21000</postanski_broj>
</adresa>

- posle UPDATE izraza
<adresa country="Srbija">
    <ulica>Karadjordjeva 144</ulica>
    <mesto>Novi Sad</mesto>
    <postanski_broj>21000</postanski_broj>
</adresa>
```

Пример 7.6: Преименовање елемента

7.2.4 Додавање новог чвора у документ

Уколико је потребно додати нови чвор у XML документ потребно је, у изразу трансформисања, навести и позицију у документу на којој се жели формирати нови елемент или атрибут.

Додавање елемента

Помоћу наредбе *insert* у оквиру наредбе *modify* израза трансформисања додаје се нови елемент у XML документ. Постоји пет опција за могућу позиција новог елемента у XML документу и то су:

- *as first into* - новоформирани елемент постаје први директан потомак наведеног елемента
- *as last into* - новоформирани елемент постаје последњи директан потомак наведеног елемента
- *before* - новоформирани елемент постаје брат наведеног елемента а позиција у документу му је одмах испред наведеног елемента
- *after* - новоформирани елемент постаје брат наведеног елемента а позиција у документу му је одмах иза наведеног елемента
- *into* - новоформирани елемент постаје директан потомак наведеног елемента али без прецизно дефинисане позиције (позиција се може разликовати од извршавања до извршавања)

У примеру 7.7 приказана је употреба израза трансформисања када је потребно за одређеног запосленог додати адресу његове електронске поште као елемент у XML документ. У оквиру наредбе *modify*, након кључне речи *insert*, наводи се елемент који се жели додати у XML документ након чега се бира једна од пет опција за могућу позицију новог елемента. Новоформирани елемент **емаил** постаје брат елемента **телефон** а његова позиција у XML документу ће бити позиција пре првог појављивања елемента **телефон**.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do insert
                <email>nevena@ibm.com</email>
                before $tmp/zaposleni/telefon[1]
            return $tmp')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<zaposleni id="1006">
...
    </adresa>
    <telefon tip="poslovni">066-335-2243</telefon>
...
</zaposleni>

- posle UPDATE izraza
<zaposleni id="1006">
...
    </adresa>
    <email>nevena@ibm.com</email>
    <telefon tip="poslovni">066-335-2243</telefon>
...
</zaposleni>
```

Пример 7.7: Додавање новог елемента

Додавање атрибута

По угледу на додавање новог елемента у XML документ, сличан принцип се користи и приликом додавања атрибута с тим што се сада, у изразу трансформисања, поред кључне речи *insert* у оквиру наредбе *modify* наводи и кључна реч *attribute*. Након тога потребно је дефинисати име новог атрибута који се жели додати у XML документ као и вредност тог атрибута у оквиру витичастих заграда. Слично као у

случају када се додаје нови елемент у XML документ и приликом додавања атрибута потребно је навести једну од пет опција за могућу позицију новог атрибута. Разлика у односу на додавање новог елемента је та да опције *as first into*, *as last into* и *into* имају исти ефекат приликом додавања атрибута. Ефекат је исти зато што се у XML моделу података не дефинише поредак међу атрибутима неког елемента. Приликом додавања атрибута опције за позицију имају следећу функцију:

- *as first into, as last into, into* - новоформирани атрибут постаје атрибут наведеног елемента
- *before, after* - новоформирани атрибут постаје атрибут оца наведеног елемента

Пример 7.8 приказује додавање новог атрибута **емаил** за запосленог чија је вредност атрибута **ид** табеле **запослени** једнака 1006. Коришћена опција за позицију новог атрибута је *into* па ће нови атрибут бити додат оном елементу из XML документа који је наведен у наставку наредбе *modify*. У случају примера 7.8, то је у оквиру елемента **име**.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do insert attribute
                email {"nevena@ibm.com"}
                into $tmp/zaposleni/ime
            return $tmp')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<ime>Nevena Mijatovic</ime>

- posle UPDATE izraza
<ime email="nevena@ibm.com">Nevena Mijatovic</ime>
```

Пример 7.8: Додавање новог атрибута

7.2.5 Брисање чвора у документу

Уколико је потребно из XML документа избрисати неки од чворова у оквиру наредбе *modify* може се употребити опција *delete*. Пример 7.9 приказује како се за одређеног запосленог може избрисати елемент **емаил** из XML документа. У изразу трансформисања после кључне речи *delete* потребно је навести израз путање до елемента који се жели избрисати.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do delete
                $tmp/zaposleni/email
            return $tmp')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<zaposleni id="1006">
...
    </adresa>
    <email>nevena@ibm.com</email>
    <telefon tip="poslovni">066-335-2243</telefon>
...
</zaposleni>

- posle UPDATE izraza
<zaposleni id="1006">
...
    </adresa>
    <telefon tip="poslovni">066-335-2243</telefon>
...
</zaposleni>
```

Пример 7.9: Брисање елемента

Треба напоменути да је запослени из наведеног примера поседовао само једну адресу електронске поште али у случају да је било наведено више адреса, овако написан израз трансформисања избрисао би све наведене адресе електронске поште из XML документа. Ипак, уколико је за елемент који се појављује више од једног пута у XML документу потребно избрисати само једно његово појављивање мора се извршити филтрирање тог елемента или по вредности атрибута, уколико елемент поседује атрибут, или по позицији елемента у XML документу.

Такође, може се десити да елемент, који се жели избрисати, има директне потомке, као што је случај са елементом **адреса** из примера A.2, том приликом ће бити избрисан читав фрагмент XML документа у који спада наведени елемент као и сви његови потомци. Пример 7.10 приказује такво понашање и брисање елемента **адреса** са свим његовим директним потомцима.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify do delete
                $tmp/zaposleni/adresa
            return $tmp')
WHERE id = 1006;

Output:
- pre UPDATE izraza
<zaposleni id="1006">
    <ime email="nevena@ibm.com">Nevena Mijatovic</ime>
    <adresa country="Srbija">
        <ulica>Karadjordjeva 144</ulica>
        <mesto>Novi Sad</mesto>
        <postanski_broj>21000</postanski_broj>
    </adresa>
    <telefon tip="poslovni">066-335-2243</telefon>
...

</zaposleni>
```

```
- posle UPDATE izraza
<zaposleni id="1006">
    <ime email="nevena@ibm.com">Nevena Mijatovic</ime>
    <telefon tip="poslovni">066-335-2243</telefon>
...
</zaposleni>
```

Пример 7.10: Брисање фрагмента документа

У случају да је потребно избрисати атрибут из XML документа довољно је у оквиру наредбе *modify* израза трансформисања навести путању до жељеног атрибута.

```
modify do delete $tmp/zaposleni/ime/@email
```

7.2.6 Измена више чворова у једном документу

XQuery Update стандард, који представља надоградњу XQuery језика, дефинише да се све операције у оквиру наредбе *modify* примењују независно једна од друге на оригинални XML документ. Односно, одређена операција не види ефекат осталих операција које су наведене у склопу наредбе *modify* већ се свака операција независно извршава над оригиналном копијом XML документа.

Пример 7.11, који се састоји од додавања и брисања елемента из XML документа, показује наведено понашање. Наиме, у наредби *modify* израза трансформисања прво је наведена операција додавања новог елемента **телефон** у XML документ док се након тога наводи операција брисања елемента **телефон** из документа. У секцији 7.2.5, у оквиру које је било речи о брисању елемената, споменуто је да уколико елемент, који се жели избрисати, има више уноса у XML документу наредбом *delete* биће избрисана сва појављивања тог елемента у документу. Због таквог функционисања, поставља се питање како ће се наредба *delete* понашати у случају као што је приказано у примеру 7.11, да ли ће наредба *delete* избрисати и новоформирани елемент **телефон** наредбом *insert* или да ли редослед наведених операција утиче на коначан резултат. Као што се може видети и из резултата упита примера 7.11, наредба *delete* брише само оне елементе који постоје у копији оригиналног XML документа тако да

операција брисања не утиче на новоформирани елемент **телефон** и он се налази у крајњем резултату. Треба напоменути да редослед извршавања операција у наредби *modify* не утиче на крајње резултате.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify (do insert <telefon tip="kucni">011-311-0692</telefon>
                    after $tmp/zaposleni/adresa,
                    do delete $tmp/zaposleni/telefon )
            return $tmp ')
WHERE id = 1005;

Output:
- pre UPDATE izraza
<telefon tip="poslovni">060-425-443</telefon>
<telefon tip="privatni">064-546-6121</telefon>

- posle UPDATE izraza
<telefon tip="kucni">011-311-0692</telefon>
```

Пример 7.11: Додавање и брисање елемента

Ипак треба скренути пажњу и на случај када се наредбом *delete* покушава избрисати елемент који има директне потомке док се у исто време наредбом *insert* додаје нови потомак тог елемента. Овакве две операције су неусаглашене пошто се у истом изразу трансформисања једном операцијом покушава додати потомак елемента који друга операција жели да избрише. Решење овог конфликта операција је такво да операција брисања има већи приоритет па ће као резултат израза трансформисања бити избрисан жељени елемент са свим својим потомцима. Такође, ни у овом случају редослед операција брисања и додавања елемената не утиче на коначно решење.

Пример 7.12 показује како се може променити структура XML документа тако да жељени елемент постане атрибут неког другог елемента а да један од атрибута постане елемент. Како би се направила оваква промена у структури XML документа потребне су четири операције, две које додају нови елемент односно атрибут и две операције брисања које бришу стари елемент и атрибут. Треба напоменути да се

приликом додавања новог елемента **ид** не наводи директно његова вредност већ се она рачуна на основу вредности атрибута **ид**. Слично се добија вредност и за атрибут **име**, тако што се вредност за атрибут рачуна на основу вредности елемента **име**.

```
UPDATE zaposleni
SET info = XMLQUERY('
            copy $tmp := $INFO
            modify (do insert <id> {$tmp/zaposleni/data(@id)} </id>
                        as first into $tmp/zaposleni,
                    do insert attribute ime {$tmp/zaposleni/ime}
                        into $tmp/zaposleni,
                    do delete $tmp/zaposleni/@id,
                    do delete $tmp/zaposleni/ime)
            return $tmp')
WHERE id = 1005;

Output:
- pre UPDATE izraza
<zaposleni id="1005">
    <ime>Aleksandar Petrovic</ime>

- posle UPDATE izraza
<zaposleni ime="Aleksandar Petrovic">
    <id>1005</id>
```

Пример 7.12: Замена елемента и атрибута

7.2.7 Измена XML документа у упиту

Изрази трансформисања који су коришћени у наведеним примерима из овог поглавља били су наведени у оквиру UPDATE израза и тако написани изрази трансформисања мењају садржај XML докумената у бази података. Идентичан израз трансформисања може се употребити и у функцији XMLQUERY у оквиру SELECT наредбе

SQL упита. Тада се све измене врше у оквиру самог упита након чега се коначни резултати прослеђују крајњем кориснику. Овакав начин коришћења израза трансформисања оставља оригиналну верзију XML документа непромењену у бази података и погодан је за коришћење приликом тестирања због чињенице да је доста сигурнији приступ пошто не може доћи до губитка велике количине података у случају лошег написаног упита.

Коришћење израза трансформирања у оквиру наредбе *return* FLOWR израра можда представља и најинтуитивнији начин употребе. Пример 7.13 показује како се израз трансформирања, који мења име елемента, може написати у склопу наредбе *return* FLOWR израза. Помоћу наредбе *for* и *where* FLOWR израза филтрира се који ће део XML документа и за ког запосленог бити прослеђен изразу трансформирања у наредби *return* FLOWR израза.

```
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/ime = "Ivana Stefanovic"
return copy $tmp := $i/adresa
       modify do rename $tmp/grad as "mesto"
       return $tmp;

Output:
<adresa drzava="Srbija">
    <mesto>Beograd</mesto>
    <ulica>Kneza Milosa 11</ulica>
</adresa>
```

Пример 7.13: Измена XML документа у упиту

Пример 7.13 показује да се као резултат упита добија промењено име елемента док резултат упита из примера 7.14 потврђује да је оригиналан XML документ и даље присутан у бази и како је промена структуре XML документа била само на нивоу упита а не документа.

```
xquery
for $i in db2-fn:xmlcolumn("ZAPOSLENI.INFO")/zaposleni
where $i/ime = "Ivana Stefanovic"
return $i/adresa/grad;
```

```
Output:
<adresa drzava="Srbija">
    <grad>Beograd</grad>
    <ulica>Kneza Milosa 11</ulica>
</adresa>
```

Пример 7.14: Провера структуре XML документа

Глава 8

Закључак

Примарни циљ овог рада је представљање језика за рад са подацима XML типа у РСУБП IBM DB2. XQuery и SQL/XML су два стандардизована језика која се користе за обраду XML података и на којима је био фокус приликом писања рада. Често се поставља питање, који језик је бољи и који језик треба користити. На то питање није лако дати једноставан одговор али би уопштен одговор био да постоји потреба за коришћењем оба језика док који је језик бољи, искључиво зависи од конкретне ситуације у којој се крајњи корисник налази и који су му приоритети у том тренутку.

XQuery је препоручен од стране Конзорцијума за управљање Web-ом и подржале су га велике компаније као што су IBM, Oracle и Microsoft. За доста људи XQuery представља нови језик који треба научити. Карактеристике самог језика су да подржава промењљиве, операторе, изразе, функције, итд. Такође, XQuery језик нуди добре перформансе када су послови везани искључиво за XML језик попут трансформисања XML података или спајање више XML докумената. У таквим ситуацијама са сигурношћу се може рећи да XQuery језику треба дати предност у односу на SQL/XML.

С друге стране, SQL/XML језик није потпуно нов језик већ представља низ проширења и надоградњу SQL језика. Велике компаније попут IBM-а и Oracle-а су дале свој допринос при развијању овог језика пошто је SQL језик синоним за стабилност и дуговечност па одатле и потиче идеја, зашто један такав језик не проширити подршком за рад са XML подацима. Када су апликације доста повезане са релационим подацима треба дати предност коришћењу SQL/XML језика. У таквом окружењу прихватљиво је дати предност SQL/XML језику чак иако ће уложени напор за развијање дела који ради са XML подацима бити већи него да се програмира у XQuery

језику. Ситуација када SQL/XML језику такође треба дати предност у односу на XQuery језик је уколико се јавља честа потреба за комбиновањем типова података и укључивањем релационих вредности из табеле у XML документе, док уколико постоји потреба за формирањем сложенијих XML докумената са угњежденим и понављајућим елементима, лакше је и природније користити XQuery језик.

Додатак А

Коришћени SQL изрази

SQL израз који је коришћен како би се направила табела **запослени** приказан је у примеру А.1. Формирана релациона табела састоји се од два атрибута:

- атрибут **ид**, типа INTEGER, који има функцију основног кључа табеле запослени
- атрибут **инфо**, типа XML, који има функцију да чува XML документе

```
CREATE TABLE zaposleni
    (id     INTEGER NOT NULL,
     info   XML     NOT NULL,
     PRIMARY KEY (id));
```

Пример А.1: Формирање табеле са атрибутом XML типа

Пример А.2 приказује како се попуњава направљена табела из примера А.1. За прва два INSERT израза су узети примери који су коришћени као XPath примери из Поглавља 3 док је трећи INSERT израз накнадно додат како би се добило на сложености и што бољем узорку над којим су у поглављима 4 и 5 објашњени XQuery изрази као и SQL/XML упити и функције [13].

```
INSERT INTO zaposleni(id, info)
VALUES (1005, '<?xml version="1.0" encoding="UTF-8" ?>
                <zaposleni id="1005">
                        <ime>Aleksandar Petrovic</ime>
```

```
                        <adresa drzava="Srbija">
                                <grad>Beograd</grad>
                                <ulica>Ruzveltova 31</ulica>
                        </adresa>
                        <telefon tip="poslovni">060-425-443</telefon>
                        <telefon tip="privatni">064-546-6121</telefon>
                    <asistent>
                        <ime>Stevan Markovic</ime>
                        <telefon tip="poslovni">063-114-5531</telefon>
                    </asistent>
                </zaposleni>');

INSERT INTO zaposleni(id, info)
VALUES (1006, '<?xml version="1.0" encoding="UTF-8" ?>
                <zaposleni id="1006">
                        <ime>Nevena Mijatovic</ime>
                        <adresa drzava="Srbija">
                                <grad>Nis</grad>
                                <ulica>Kneza Milosa 77b</ulica>
                        </adresa>
                        <telefon tip="poslovni">066-335-2243</telefon>
                        <telefon tip="privatni">060-116-7713</telefon>
                    <asistent>
                        <ime>Ana Jeremic</ime>
                        <telefon tip="poslovni">062-334-3155</telefon>
                    </asistent>
                </zaposleni>');

INSERT INTO zaposleni(id, info)
VALUES (1007, '<?xml version="1.0" encoding="UTF-8" ?>
                <zaposleni id="1007">
                        <ime>Ivana Stefanovic</ime>
                        <adresa drzava="Srbija">
                                <grad>Beograd</grad>
                                <ulica>Kneza Milosa 11</ulica>
```

```
                    </adresa>
                    <telefon tip="poslovni">060-224-8838</telefon>
                    <telefon tip="privatni">060-761-5455</telefon>
                    <telefon tip="kucni">011-2344-718</telefon>
            </zaposleni>');
```

Пример А.2: Изрази за попуњавање формиране табеле

Литература

[1] Peter Aiken, David Allen. *XML for Data Management*. Morgan Kaufmann Publishers, San Francisco, CA, U.S., 2004.

[2] IBM Corporation. *SQL Reference Volume 1*. IBM Corporation, Endicott, NY, U.S., 2014.

[3] Matthias Nicola, Fatma Ozcan. *pureXML in DB2 9: Which way to query your XMLdata*, August 2007. URL https://www.ibm.com/developerworks/data/library/techarticle/dm-0606nicola.

[4] Matthias Nicola, Pav Kumar-Chatterjee. *DB2 pureXML Cookbook*. Pearson Education, Inc., Boston, MA, U.S., 2009.

[5] Michael Kay. *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wiley Publishing, Inc., Indianapolis, IN, U.S., 4th edition edition, 2008.

[6] John Simpson. *XPath and XPointer*. O'Reilly Media, Inc., Sebastopol, CA, U.S., 2002.

[7] Andrew Watt. *XPath Essentials*. John Wiley & Sons, Inc., New York, U.S., 2004.

[8] IBM Knowledge Center. *Namespace declaration*, June 2016. URL https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.xml.doc/doc/xqrnsdecl.html.

[9] Priscilla Walmsley. *XQuery*. O'Reilly Media, Inc., Sebastopol, CA, U.S., 2nd edition edition, 2015.

[10] IBM Corporation. *XQuery Reference*. IBM Corporation, Endicott, NY, U.S., 2013.

[11] Stephen Buxton, Jim Melton. *Querying XML*. Morgan Kaufmann Publishers, San Francisco, CA, U.S., 2011.

[12] Cynthia Saracco. *Query DB2 XML data with SQL*, March 2010. URL https://www.ibm.com/developerworks/data/library/techarticle/dm-0603saracco2.

[13] IBM Corporation. *pureXML Guide*. IBM Corporation, Endicott, NY, U.S., 2013.