Name: Rugvedi Jamgaonkar

Class: MSc Computer Science 1

Enroll No.: 2002242

Subject: Advanced Database Concepts

Assignment 3: XML Databases

**1. Write and execute XQuery expressions for the following queries on the company.xml document:**

**1. Write and execute XQuery expressions for the following queries on the company.xml document:**

**a. Retrieve the names and addresses of employees who work for the “Research” department.**

companyDB/employees/employee[@worksFor=5]/fname | //employee[@worksFor=5]/address

Result:

<?xml version="1.0" encoding="UTF-8"?>

<fname>Franklin</fname>

<address>638 Voss, Houston, TX</address>

<fname>John</fname>

<address>731 Fondren, Houston, TX</address>

<fname>Ramesh</fname>

<address>971 Fire Oak, Humble, TX</address>

<fname>Joyce</fname>

<address>5631 Rice, Houston, TX</address>

**b. For every project located in “Stafford”, retrieve the project number, the controlling department number, and the department’s manager’s last name, address, and birth date.**

let $path := doc("C:/XML/company.xml")

let $projPath := $path/companyDB/projects/project[plocation = "Stafford"]

let $deptPath := $path/companyDB/departments/department[@dno = data($projPath/@controllingDepartment)]

let $empPath := $path/companyDB/employees/employee[@ssn = data($deptPath/manager/@mssn)]

for $i in $empPath

return

<result>

<manager>

{

$i/lname |

$i/address |

$i/dob

}

</manager>

<pnumber>

{

data($projPath/@pnumber)

}

</pnumber>

<controllingDepartment>

{

data($projPath/@controllingDepartment)

}

</controllingDepartment>

</result>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<result>

<manager>

<lname>Wallace</lname>

<dob>20-JUN-31</dob>

<address>291 Berry, Bellaire, TX</address>

</manager>

<pnumber>10 30</pnumber>

<controllingDepartment>4 4</controllingDepartment>

</result>

**c. Retrieve the names of all employees who have two or more dependents.**

let $d:=doc("C:/XML/company.xml")

for $e in $d/companyDB/employees/employee

where count($e/dependents/dependent) >= 2

return

<emp\_with\_2\_or\_more\_dependents>

{$e/fname} {$e/lname}

</emp\_with\_2\_or\_more\_dependents>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<emp\_with\_2\_or\_more\_dependents>

<fname>Franklin</fname>

<lname>Wong</lname>

</emp\_with\_2\_or\_more\_dependents>

<emp\_with\_2\_or\_more\_dependents>

<fname>John</fname>

<lname>Smith</lname>

</emp\_with\_2\_or\_more\_dependents>

<emp\_with\_2\_or\_more\_dependents>

<fname>Alex</fname>

<lname>Freed</lname>

</emp\_with\_2\_or\_more\_dependents>

**d. Retrieve the names of managers who have at least one dependent.**

<Managers\_with\_atleast\_1\_dependent>

{

let $d:=doc("C:/XML/company.xml")

for $e in $d/companyDB/employees/employee[@manages]

where count($e/dependents/dependent)>=1

return

<manager>

{$e/fname}{$e/lname}

</manager>

}

</Managers\_with\_atleast\_1\_dependent>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<Managers\_with\_atleast\_1\_dependent>

<manager>

<fname>Franklin</fname>

<lname>Wong</lname>

</manager>

<manager>

<fname>Jennifer</fname>

<lname>Wallace</lname>

</manager>

<manager>

<fname>Alex</fname>

<lname>Freed</lname>

</manager>

</Managers\_with\_atleast\_1\_dependent>

**e. Retrieve the names of employees who work on all projects controlled by department “5”.**

<employees\_working\_for\_dept\_5>

{

let $d:=doc("C:/XML/company.xml")

for $e in $d/companyDB/employees/employee[@worksFor=5]

return

<employee>

{$e/fname}{$e/lname}

</employee>

}

</employees\_working\_for\_dept\_5>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<employees\_working\_for\_dept\_5>

<employee>

<fname>Franklin</fname>

<lname>Wong</lname>

</employee>

<employee>

<fname>John</fname>

<lname>Smith</lname>

</employee>

<employee>

<fname>Ramesh</fname>

<lname>Narayan</lname>

</employee>

<employee>

<fname>Joyce</fname>

<lname>English</lname>

</employee>

</employees\_working\_for\_dept\_5>

**2. Consider the bibliography XML document, bib.xml, provided along with this lab assignment.**

**a. Write a XML Schema specification for the XML document.**

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="editor">

<xs:complexType>

<xs:sequence>

<xs:element ref="last"/>

<xs:element ref="first"/>

<xs:element ref="affiliation"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="last" type="xs:string"/>

<xs:element name="affiliation" type="xs:string"/>

<xs:element name="author">

<xs:complexType>

<xs:sequence>

<xs:element ref="last"/>

<xs:element ref="first"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="price" type="xs:string"/>

<xs:element name="book">

<xs:complexType>

<xs:sequence>

<xs:element ref="title"/>

<xs:element ref="editor" minOccurs="0"/>

<xs:element ref="author" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="publisher"/>

<xs:element ref="price"/>

</xs:sequence>

<xs:attribute name="year" type="xs:string" use="required"/>

</xs:complexType>

</xs:element>

<xs:element name="publisher" type="xs:string"/>

<xs:element name="bib">

<xs:complexType>

<xs:sequence>

<xs:element ref="book" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="title" type="xs:string"/>

<xs:element name="first" type="xs:string"/>

</xs:schema>

DTD Schema:-

<!ELEMENT editor ( last , first , affiliation )>

<!ELEMENT last ( #PCDATA )>

<!ELEMENT affiliation ( #PCDATA )>

<!ELEMENT author ( last , first )>

<!ELEMENT price ( #PCDATA )>

<!ELEMENT book ( title , editor? , author\* , publisher , price )>

<!ATTLIST book

year CDATA #REQUIRED

>

<!ELEMENT publisher ( #PCDATA )>

<!ELEMENT bib ( book+ )>

<!ELEMENT title ( #PCDATA )>

<!ELEMENT first ( #PCDATA )>

**b. Write expressions in XQuery to answer the following queries:**

**i. Find books that have "** **Technology" in their titles.**

let $d:=doc("C:/XML/bib.xml")

for $books in $d/bib/book

where contains($books/title,'Technology')

return $books

Result:

<?xml version="1.0" encoding="UTF-8"?>

<book year="1999">

<title>The Economics of Technology and Content for Digital TV</title>

<editor>

<last>Gerbarg</last>

<first>Darcy</first>

<affiliation>CITI</affiliation>

</editor>

<publisher>Kluwer Academic Publishers</publisher>

<price>129.95</price>

</book>

**ii. Find all books authored by "** **W. Stevens".**

let $d:=doc("C:/XML/bib.xml")

for $books in $d/bib/book

where $books/author/first="W."

return $books

Result:

<?xml version="1.0" encoding="UTF-8"?>

<book year="1994">

<title>TCP/IP Illustrated</title>

<author>

<last>Stevens</last>

<first>W.</first>

</author>

<publisher>Addison-Wesley</publisher>

<price>65.95</price>

</book>

<book year="1992">

<title>Advanced Programming in the Unix environment</title>

<author>

<last>Stevens</last>

<first>W.</first>

</author>

<publisher>Addison-Wesley</publisher>

<price>65.95</price>

</book>

**iii. Find number of books with 3 or more than 3 authors.**

let $d:=doc("C:/XML/bib.xml")

for $books in $d/bib/book

where count($books/author) >=3

return

<Count\_of\_books\_having\_3\_or\_more\_than\_3\_authors>

{count($books)}

</Count\_of\_books\_having\_3\_or\_more\_than\_3\_authors>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<Count\_of\_books\_having\_3\_or\_more\_than\_3\_authors>1</Count\_of\_books\_having\_3\_or\_more\_than\_3\_authors>

**iv. Find all books that have price more than 40 units.**

let $d:=doc("C:/XML/bib.xml")

for $books in $d/bib/book

where $books/price > 40.00

return $books

Result:

<?xml version="1.0" encoding="UTF-8"?>

<book year="1994">

<title>TCP/IP Illustrated</title>

<author>

<last>Stevens</last>

<first>W.</first>

</author>

<publisher>Addison-Wesley</publisher>

<price>65.95</price>

</book>

<book year="1992">

<title>Advanced Programming in the Unix environment</title>

<author>

<last>Stevens</last>

<first>W.</first>

</author>

<publisher>Addison-Wesley</publisher>

<price>65.95</price>

</book>

<book year="1999">

<title>The Economics of Technology and Content for Digital TV</title>

<editor>

<last>Gerbarg</last>

<first>Darcy</first>

<affiliation>CITI</affiliation>

</editor>

<publisher>Kluwer Academic Publishers</publisher>

<price>129.95</price>

</book>

**3. Consider the following MAILORDER relational schema describing the data for a mail order company: parts(pno,pname,qoh,price,olevel) customers(cno,cname,street,zip,phone) employees(eno,ename,zip,hdate)**

**zipcodes(zip,city)**

**orders(ono,cno,eno,received,shipped)**

**odetails(ono,pno,qty)**

**The attribute names are self-explanatory. “qoh” stands for quantity on hand.**

**a. Create a XML representation of the data described there (you should invent your own data**

**instances).**

<?xml version="1.0" encoding="UTF-8"?>

<mailorder>

<parts>

<part>

<pno>01</pno>

<pname>Brake Wire</pname>

<qoh>2</qoh>

<price>15.00</price>

<olevel>small</olevel>

</part>

<part>

<pno>02</pno>

<pname>Clutch plate</pname>

<qoh>3</qoh>

<price>51.00</price>

<olevel>medium</olevel>

</part>

<part>

<pno>03</pno>

<pname>Chain</pname>

<qoh>1</qoh>

<price>30.00</price>

<olevel>large</olevel>

</part>

</parts>

<customers>

<customer>

<cno>101</cno>

<cname>Rakesh</cname>

<street>Ghole street</street>

<zip>411002</zip>

<phone>9745927349</phone>

</customer>

<customer>

<cno>102</cno>

<cname>Shubhankar</cname>

<street>Z street</street>

<zip>411005</zip>

<phone>9448354349</phone>

</customer>

<customer>

<cno>103</cno>

<cname>Ravi</cname>

<street>Gandhi street</street>

<zip>411002</zip>

<phone>9345927349</phone>

</customer>

</customers>

<employees>

<employee>

<eno>1001</eno>

<ename>Atharva</ename>

<zip>411041</zip>

<hdate>08-2-2019</hdate>

</employee>

<employee>

<eno>1002</eno>

<ename>Swaroop</ename>

<zip>411042</zip>

<hdate>12-2-2019</hdate>

</employee>

<employee>

<eno>1003</eno>

<ename>Kunal</ename>

<zip>411040</zip>

<hdate>17-6-2020</hdate>

</employee>

</employees>

<zipcodes>

<zipcode>

<zip>411022</zip>

<city>Pune</city>

</zipcode>

<zipcode>

<zip>411015</zip>

<city>Kolkata</city>

</zipcode>

<zipcode>

<zip>411041</zip>

<city>Wichita</city>

</zipcode>

<zipcode>

<zip>411056</zip>

<city>Goa</city>

</zipcode>

<zipcode>

<zip>411034</zip>

<city>Mumbai</city>

</zipcode>

</zipcodes>

<orders>

<order>

<ono>22801</ono>

<cno>101</cno>

<eno>1001</eno>

<recieved>2</recieved>

<shipped>1</shipped>

</order>

<order>

<ono>22802</ono>

<cno>102</cno>

<eno>1002</eno>

<recieved>2</recieved>

<shipped>2</shipped>

</order>

<order>

<ono>22803</ono>

<cno>101</cno>

<eno>1003</eno>

<recieved>3</recieved>

<shipped>1</shipped>

</order>

</orders>

<odetails>

<odetail>

<ono>22801</ono>

<pno>01</pno>

<qty>2</qty>

</odetail>

<odetail>

<ono>22802</ono>

<pno>02</pno>

<qty>1</qty>

</odetail>

<odetail>

<ono>22803</ono>

<pno>03</pno>

<qty>1</qty>

</odetail>

</odetails>

</mailorder>

**b. Write a XML Schema specification for the XML document constructed in part a.**

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="city" type="xs:string"/>

<xs:element name="part">

<xs:complexType>

<xs:sequence>

<xs:element ref="pno"/>

<xs:element ref="pname"/>

<xs:element ref="qoh"/>

<xs:element ref="price"/>

<xs:element ref="olevel"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="cname" type="xs:string"/>

<xs:element name="odetails">

<xs:complexType>

<xs:sequence>

<xs:element ref="odetail" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="qoh" type="xs:string"/>

<xs:element name="employee">

<xs:complexType>

<xs:sequence>

<xs:element ref="eno"/>

<xs:element ref="ename"/>

<xs:element ref="zip"/>

<xs:element ref="hdate"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="price" type="xs:string"/>

<xs:element name="street" type="xs:string"/>

<xs:element name="customers">

<xs:complexType>

<xs:sequence>

<xs:element ref="customer" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="order">

<xs:complexType>

<xs:sequence>

<xs:element ref="ono"/>

<xs:element ref="cno"/>

<xs:element ref="eno"/>

<xs:element ref="recieved"/>

<xs:element ref="shipped"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="zip" type="xs:string"/>

<xs:element name="cno" type="xs:string"/>

<xs:element name="pname" type="xs:string"/>

<xs:element name="zipcodes">

<xs:complexType>

<xs:sequence>

<xs:element ref="zipcode" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="eno" type="xs:string"/>

<xs:element name="recieved" type="xs:string"/>

<xs:element name="pno" type="xs:string"/>

<xs:element name="zipcode">

<xs:complexType>

<xs:sequence>

<xs:element ref="zip"/>

<xs:element ref="city"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="ono" type="xs:string"/>

<xs:element name="ename" type="xs:string"/>

<xs:element name="shipped" type="xs:string"/>

<xs:element name="phone" type="xs:string"/>

<xs:element name="mailorder">

<xs:complexType>

<xs:sequence>

<xs:element ref="parts"/>

<xs:element ref="customers"/>

<xs:element ref="employees"/>

<xs:element ref="zipcodes"/>

<xs:element ref="orders"/>

<xs:element ref="odetails"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="qty" type="xs:string"/>

<xs:element name="parts">

<xs:complexType>

<xs:sequence>

<xs:element ref="part" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="orders">

<xs:complexType>

<xs:sequence>

<xs:element ref="order" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="employees">

<xs:complexType>

<xs:sequence>

<xs:element ref="employee" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="odetail">

<xs:complexType>

<xs:sequence>

<xs:element ref="ono"/>

<xs:element ref="pno"/>

<xs:element ref="qty"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="olevel" type="xs:string"/>

<xs:element name="hdate" type="xs:string"/>

<xs:element name="customer">

<xs:complexType>

<xs:sequence>

<xs:element ref="cno"/>

<xs:element ref="cname"/>

<xs:element ref="street"/>

<xs:element ref="zip"/>

<xs:element ref="phone"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

DTD Schema:-

<!ELEMENT city ( #PCDATA )>

<!ELEMENT part ( pno , pname , qoh , price , olevel )>

<!ELEMENT cname ( #PCDATA )>

<!ELEMENT odetails ( odetail+ )>

<!ELEMENT qoh ( #PCDATA )>

<!ELEMENT employee ( eno , ename , zip , hdate )>

<!ELEMENT price ( #PCDATA )>

<!ELEMENT street ( #PCDATA )>

<!ELEMENT customers ( customer+ )>

<!ELEMENT order ( ono , cno , eno , recieved , shipped )>

<!ELEMENT zip ( #PCDATA )>

<!ELEMENT cno ( #PCDATA )>

<!ELEMENT pname ( #PCDATA )>

<!ELEMENT zipcodes ( zipcode+ )>

<!ELEMENT eno ( #PCDATA )>

<!ELEMENT recieved ( #PCDATA )>

<!ELEMENT pno ( #PCDATA )>

<!ELEMENT zipcode ( zip , city )>

<!ELEMENT ono ( #PCDATA )>

<!ELEMENT ename ( #PCDATA )>

<!ELEMENT shipped ( #PCDATA )>

<!ELEMENT phone ( #PCDATA )>

<!ELEMENT mailorder ( parts , customers , employees , zipcodes , orders , odetails )>

<!ELEMENT qty ( #PCDATA )>

<!ELEMENT parts ( part+ )>

<!ELEMENT orders ( order+ )>

<!ELEMENT employees ( employee+ )>

<!ELEMENT odetail ( ono , pno , qty )>

<!ELEMENT olevel ( #PCDATA )>

<!ELEMENT hdate ( #PCDATA )>

<!ELEMENT customer ( cno , cname , street , zip , phone )>

**c. Write expressions in XQuery for the following queries:**

**I. Retrieve the names of parts that cost less than 20.00units.**

let $d:=doc("C:/XML/mailOrder.xml")

for $parts in $d/mailorder/parts/part

where $parts/price < 20.00

return

<part\_costing\_less\_than\_20>

{$parts/pname}

</part\_costing\_less\_than\_20>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<part\_costing\_less\_than\_20>

<pname>Brake Wire</pname>

</part\_costing\_less\_than\_20>

**II. Retrieve the names and cities of employees who have taken orders for parts costing more than 50.00 units**

let $d:=doc("C:/XML/mailOrder.xml")

let $part:=$d/mailorder/parts/part[price > 50]

let $odetail:=$d/mailorder/odetails/odetail[pno=data($part/pno)]

let $order:=$d/mailorder/orders/order[ono=data($odetail/ono)]

let $emp:=$d/mailorder/employees/employee[eno=data($order/eno)]

let $zip:=$d/mailorder/zipcodes/zipcode[zip=data($emp/zip)]

for $i in $emp

return

<result>

{$i/ename} {$zip/city}

</result>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<result>

<ename>Swaroop</ename>

<city>Goa</city>

</result>

**III. Retrieve the pairs of customer number values of customers who live in the same zip code.**

**IV. Retrieve the names of customers who have ordered parts only from employees living in the city of Wichita.**

let $d:=doc("C:/XML/mailOrder.xml")

let $city:=$d/mailorder/zipcodes/zipcode[city="Wichita"]

let $emp:=$d/mailorder/employees/employee[zip=data($city/zip)]

let $order:=$d/mailorder/orders/order[eno=data($emp/eno)]

let $cust:=$d/mailorder/customers/customer[cno=data($order/cno)]

for $i in $cust

return

<cust\_names>

{$i/cname}

</cust\_names>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<cust\_names>

<cname>Rakesh</cname>

</cust\_names>

**V. Retrieve the names of customers who have ordered all parts costing less than 20.00uits.**

let $d:=doc("C:/XML/mailOrder.xml")

let $parts:=$d/mailorder/parts/part[price < 20.00]

let $odetails:=$d/mailorder/odetails/odetail[pno=data($parts/pno)]

let $order:=$d/mailorder/orders/order[ono=data($odetails/ono)]

let $cust:=$d/mailorder/customers/customer[cno=data($order/cno)]

for $i in $cust

return $i/cname

Result:

<?xml version="1.0" encoding="UTF-8"?>

<cname>Rakesh</cname>

**VI. Retrieve the names of customers who have not placed a single order.**

let $d:=doc("C:/XML/mailOrder.xml")

let $order:=distinct-values($d/mailorder/orders/order/cno/text())

let $cust:=$d/mailorder/customers/customer

for $i in $cust

return

if (data($i/cno) = data($order)) then()

else

<result>{data($i/cname)}</result> names of customers who have not placed a single order.

Result:

<?xml version="1.0" encoding="UTF-8"?>

<result>Ravi</result>

**VII. Retrieve the names of customers who have placed exactly two orders.**

let $d:=doc("C:/XML/mailOrder.xml")

let $cust:=$d/mailorder/customers/customer

let $order:=$d/mailorder/orders/order

for $i in $cust

return

if (count($order[cno = data($i/cno)]) = 2) then

<result>{data($i/cname)}</result>

else ()

Result:

<?xml version="1.0" encoding="UTF-8"?>

<result>Rakesh</result>

**4. Consider the following GRADEBOOK relational schema describing the data for a grade book of a particular instructor (Note: The attributes A, B, C, and D store grade cutoffs.)**

**catalog(cno,ctitle)**

**students(sid,fname,lname,minit)**

**courses(term,secno,cno,A,B,C,D)**

**enrolls(sid,term,secno)**

**A, B, C, D, and F for each course he or she teaches.**

**a. Create a XML representation of the data described there (you should invent your own data instances).**

<?xml version="1.0" encoding="UTF-8"?>

<gradebook>

<grade>

<catlogs>

<catlog>

<cno>CSc225</cno>

<ctitle>Automata</ctitle>

</catlog>

<catlog>

<cno>CSc226</cno>

<ctitle>Management</ctitle>

</catlog>

<catlog>

<cno>CSc227</cno>

<ctitle>AWS</ctitle>

</catlog>

<catlog>

<cno>CSc228</cno>

<ctitle>Ether Mining</ctitle>

</catlog>

</catlogs>

<students>

<student>

<sid>101</sid>

<fname>Manasvi</fname>

<lname>Nimbalkar</lname>

<minit>60</minit>

</student>

<student>

<sid>102</sid>

<fname>Nachiket</fname>

<lname>Palnitkar</lname>

<minit>30</minit>

</student>

<student>

<sid>103</sid>

<fname>Atharva</fname>

<lname>Mahamuni</lname>

<minit>70</minit>

</student>

<student>

<sid>104</sid>

<fname>Prathamesh</fname>

<lname>Kulkarni</lname>

<minit>89</minit>

</student>

<student>

<sid>105</sid>

<fname>Swaroop</fname>

<lname>Korde</lname>

<minit>70</minit>

</student>

</students>

<courses>

<coures>

<term>1996</term>

<secno>sec1</secno>

<cno>CSc225</cno>

<A>70</A>

<B>60</B>

<C>50</C>

<D>40</D>

</coures>

<coures>

<term>1997</term>

<secno>sec2</secno>

<cno>CSc226</cno>

<A>70</A>

<B>60</B>

<C>50</C>

<D>40</D>

</coures>

<coures>

<term>1998</term>

<secno>sec3</secno>

<cno>CSc227</cno>

<A>70</A>

<B>60</B>

<C>50</C>

<D>40</D>

</coures>

<coures>

<term>1998</term>

<secno>sec4</secno>

<cno>CSc228</cno>

<A>70</A>

<B>60</B>

<C>50</C>

<D>40</D>

</coures>

</courses>

<enrolls>

<enroll>

<sid>101</sid>

<term>1996</term>

<secno>sec1</secno>

</enroll>

<enroll>

<sid>102</sid>

<term>1997</term>

<secno>sec2</secno>

</enroll>

<enroll>

<sid>103</sid>

<term>1998</term>

<secno>sec3</secno>

</enroll>

<enroll>

<sid>102</sid>

<term>1998</term>

<secno>sec3</secno>

</enroll>

</enrolls>

</grade>

</gradebook>

**b. Write a XML Schema specification for the XML document constructed in**

**part a.**

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="catlogs">

<xs:complexType>

<xs:sequence>

<xs:element ref="catlog" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="fname" type="xs:string"/>

<xs:element name="courses">

<xs:complexType>

<xs:sequence>

<xs:element ref="coures" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="A" type="xs:string"/>

<xs:element name="B" type="xs:string"/>

<xs:element name="C" type="xs:string"/>

<xs:element name="D" type="xs:string"/>

<xs:element name="cno" type="xs:string"/>

<xs:element name="student">

<xs:complexType>

<xs:sequence>

<xs:element ref="sid"/>

<xs:element ref="fname"/>

<xs:element ref="lname"/>

<xs:element ref="minit"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="enrolls">

<xs:complexType>

<xs:sequence>

<xs:element ref="enroll" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="students">

<xs:complexType>

<xs:sequence>

<xs:element ref="student" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="minit" type="xs:string"/>

<xs:element name="gradebook">

<xs:complexType>

<xs:sequence>

<xs:element ref="grade"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="sid" type="xs:string"/>

<xs:element name="secno" type="xs:string"/>

<xs:element name="lname" type="xs:string"/>

<xs:element name="catlog">

<xs:complexType>

<xs:sequence>

<xs:element ref="cno"/>

<xs:element ref="ctitle"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="coures">

<xs:complexType>

<xs:sequence>

<xs:element ref="term"/>

<xs:element ref="secno"/>

<xs:element ref="cno"/>

<xs:element ref="A"/>

<xs:element ref="B"/>

<xs:element ref="C"/>

<xs:element ref="D"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="ctitle" type="xs:string"/>

<xs:element name="grade">

<xs:complexType>

<xs:sequence>

<xs:element ref="catlogs"/>

<xs:element ref="students"/>

<xs:element ref="courses"/>

<xs:element ref="enrolls"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="term" type="xs:string"/>

<xs:element name="enroll">

<xs:complexType>

<xs:sequence>

<xs:element ref="sid"/>

<xs:element ref="term"/>

<xs:element ref="secno"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

DTD:

<!ELEMENT catlogs ( catlog+ )>

<!ELEMENT fname ( #PCDATA )>

<!ELEMENT courses ( coures+ )>

<!ELEMENT A ( #PCDATA )>

<!ELEMENT B ( #PCDATA )>

<!ELEMENT C ( #PCDATA )>

<!ELEMENT D ( #PCDATA )>

<!ELEMENT cno ( #PCDATA )>

<!ELEMENT student ( sid , fname , lname , minit )>

<!ELEMENT enrolls ( enroll+ )>

<!ELEMENT students ( student+ )>

<!ELEMENT minit ( #PCDATA )>

<!ELEMENT gradebook ( grade )>

<!ELEMENT sid ( #PCDATA )>

<!ELEMENT secno ( #PCDATA )>

<!ELEMENT lname ( #PCDATA )>

<!ELEMENT catlog ( cno , ctitle )>

<!ELEMENT coures ( term , secno , cno , A , B , C , D )>

<!ELEMENT ctitle ( #PCDATA )>

<!ELEMENT grade ( catlogs , students , courses , enrolls )>

<!ELEMENT term ( #PCDATA )>

<!ELEMENT enroll ( sid , term , secno )>

**c. Write expressions in XQuery for the following queries**

**I. Retrieve the names of students enrolled in the ‘Automata’ class in the term of Fall 1996.**

let $path := doc("/root/mySql/mysql/ORDBMS/xml/assignment/gradebook.xml")

let $cPath := $path/gradebook/grade/catalogs/catalog[cTitle="Automata"]

let $coursePath := $path/gradebook/grade/courses/course[cno=data($cPath/cno)]

let $enrollPath := $path/gradebook/grade/enrolls/enroll[term=data($coursePath/term)]| $path/gradebook/grade/enrolls/enroll[term="1996"]

let $studentPath := $path/gradebook/grade/students/student[sid=data($enrollPath/sid)]

return

<student>{$studentPath/fname}</student>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<student>

<fname>Manasvi</fname>

</student>

**II. Retrieve the SID values of students who have enrolled in CSc226 as well as CSc227.**

let $path := doc("C:/XML/gradebook.xml")

let $couresPath := $path/gradebook/grade/courses/coures[cno="CSc226"] | $path/gradebook/grade/courses/coures[cno="CSc227"]

let $enrollPath := $path/gradebook/grade/enrolls/enroll[term=data($couresPath/term)]

let $studentPath := $path/gradebook/grade/students/student[sid=data($enrollPath/sid)]

return

<student>{$studentPath/sid}</student>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<student>

<sid>102</sid>

<sid>103</sid>

</student>

**III. Retrieve the SID values of students who have enrolled in CSc226 or CSc227.**

let $path := doc("C:/XML/gradebook.xml")

let $couresPath := $path/gradebook/grade/courses/coures[cno="CSc226" or cno="CSc227"]

let $enrollPath := $path/gradebook/grade/enrolls/enroll[term=data($couresPath/term)]

let $studentPath := $path/gradebook/grade/students/student[sid=data($enrollPath/sid)]

return

<student>{$studentPath/sid}</student>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<student>

<sid>102</sid>

<sid>103</sid>

</student>

**IV. Retrieve the names of students who have not enrolled in any class.**

let $path := doc("C:/XML/gradebook.xml")

let $enrollPath := distinct-values($path/gradebook/grade/enrolls/enroll/sid/text())

let $studentPath := $path/gradebook/grade/students/student

for $i in $studentPath

return

if (data($i/sid) = data($enrollPath)) then()

else

<student>{data($i/sid)}</student>

Result:

<?xml version="1.0" encoding="UTF-8"?>

<student>104</student>

<student>105</student>