2. Get the names of persons who have acted in a movie and also been the sole director of the same movie.
3. Get the titles and years of movies in which James Caan has acted.

8.5 Write an XML Schema specification for the bibliographic XML document of Example 8.3.1. Validate an instance document against the schema.

8.6 Write an XML Schema specification for the geography XML document described in Example 8.3.2. Validate an instance document against the schema.

8.7 Consider the following XML document containing data from the mail-order database:

```
<mo>
  <customers>
    <customer cno="1111">
      <cname>Charles</cname>
      <street>123 Main St.</street>
      <city>Wichita</city>
      <zip>67226</zip>
      <phone>316-636-5555</phone>
    </customer>
    .
    .
    .
  </customers>
  <employees>
    <employee eno="1000">
      <ename>Jones</ename>
      <city>Wichita</city>
      <zip>67226-1555</zip>
      <hdate>1995-12-12</hdate>
    </employee>
    .
    .
    .
  </employees>
  <parts>
    <part pno="10506">
      <pname>Land Before Time I</pname>
      <qoh>200</qoh>
      <price>19.99</price>
      <level>20</level>
    </part>
```

CSc 8711
Spring 2009

Solve problems marked I, II, III, IV

```
    .
    .
    .
  </parts>
  <orders>
    <order ono="1022" takenBy="1001" customer="2222">
      <receivedDate>1995-02-13</receivedDate>
      <shippedDate>1995-02-20</shippedDate>
      <items>
        <item>
          <partNumber>10601</partNumber>
          <quantity>1</quantity>
        </item>
        <item>
          <partNumber>10701</partNumber>
          <quantity>1</quantity>
        </item>
      </items>
    </order>
    .
    .
    .
  </orders>
</mo>
```

The XML document has the `<mo>` element at the root and contains four subelements: `<customers>`, `<employees>`, `<parts>`, and `<orders>` in sequence.

The `<customers>` element contains zero or more `<customer>` subelements, and the `<customer>` element contains one attribute, `cno`, whose value is a 4-digit integer between 1000 and 9999, and five subelements: `<cname>` of type `string`; `<street>` of type `string`; `<city>` of type `string`; `<zip>`, whose value is either a 5-digit number or a 5-digit number followed by a `"-"` followed by a 4-digit number; and `<phone>`, whose value is a 3-digit area code followed by a `"-"` followed by a 3-digit exchange code, followed by a `"-"` and followed by a 4-digit number.

The `<employees>` element contains zero or more `<employee>` subelements, and the `<employee>` element contains one attribute, `eno`, whose value is a 4-digit integer between 1000 and 9999, and four subelements: `<ename>` of type `string`; `<city>` of type `string`; `<zip>`, whose value is either a 5-digit number or a 5-digit number followed by a `"-"` followed by a 4-digit number; and `<hdate>` of type `date`.

The `<parts>` element contains zero or more `<part>` subelements, and the `<part>` element contains one attribute, `pno`, whose value is a 5-digit integer between 10000 and 99999, and four subelements: `<pname>` of type `string`, `<qoh>` of type

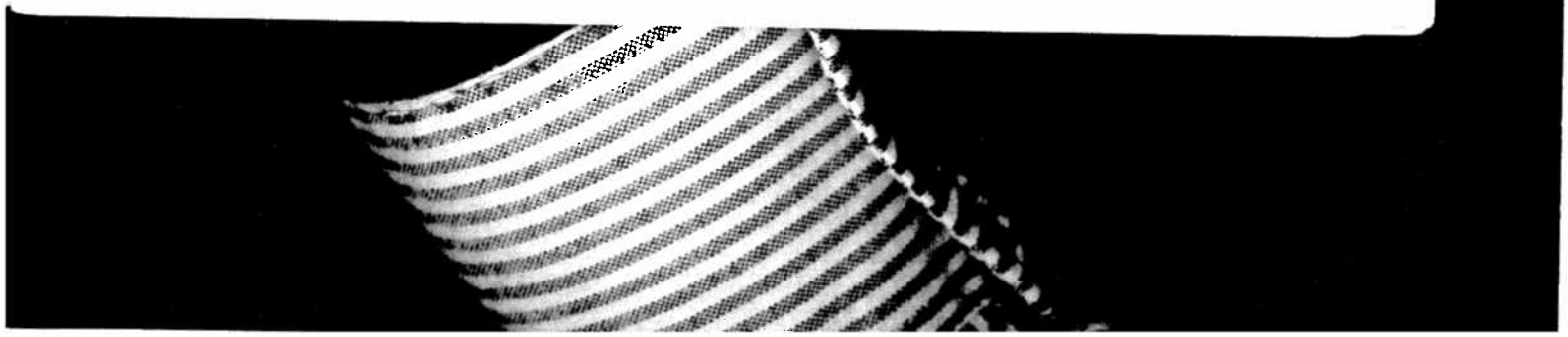
`positive integer`, `<price>` of type `positive decimal` with two fractional digits, and `<level>` of type `positive integer`.

The `<orders>` element contains zero or more `<order>` subelements, and the `<order>` element contains three attributes, `ono`, whose value is a 4-digit integer between 1000 and 9999, `takenBy`, whose value is the same type as an employee number, and `customer`, whose value is the same type as a customer number, and the following subelements: `<receivedDate>` of type `date`, an optional`<shippedDate>` of type `date`, and `<items>`, which is a repeating group (one or more) of `<item>` elements. The `<item>` element contains two subelements: `<partNumber>`, whose value is the same type as a part number, and `<quantity>` of type `positive integer`.

Write an XML Schema specification for the mail order document and validate several instance documents against the schema.

8.8 Consider the following XML document containing data about movies and performers:

```
<mdb>
  <movies>
    <movie id="godfather1">
      <year>1972</year>
      <directors>
        <director idref="francisfordcoppola"/>
      </directors>
      <genres>
        <genre>Crime</genre>
        <genre>Drama</genre>
      </genres>
      <plot>A mafia boss's son...</plot>
      <cast>
        <performer>
          <actor idref="marlonbrando"/>
          <role>Don Vito Corleone</role>
        </performer>
        .
        .
        .
      </cast>
    </movie>
    .
    .
    .
</movies>
<performers>
  <performer id="marlonbrando">
```



8.9 Consider the following XML document containing the transcript for a student:

```
<Transcript>
  <Sid>111111111</Sid>
  <Name>John Doe</Name>
  <Courses>
    <Course>
      <Code>CS308</Code>
      <Sem>F1997</Sem>
      <Grade>B</Grade>
    </Course>
    .
    .
    .
  </Courses>
</Transcript>
```

The `<Sid>` element contains 9-digit social security numbers, the `<Sem>` element contains a letter (`F`, `S`, or `U`) followed by a 4-digit year, and the `<Grade>` element contains one of the grades: `A`, `B`, `C`, `D`, `F`, `IP`, `S`, or `U`. Write an XML Schema specification for the transcript document and validate several instance documents against the schema.

8.10 Consider the following XML document that describes data related to the postings in a typical bulletin board:

```
<postings>
  <posting postId="1" postDate="2003-1-11">
    <postedBy>a@abc.com</postedBy>
    <postSubject>Welcome</postSubject>
    <content>Welcome to the bulletin board</content>
    <followUp postId="2" postDate="2003-1-12">
      <postedBy>b@abc.com</postedBy>
      <postSubject>Posting 2</postSubject>
      <content>This is posting 2</content>
      <followUp postId="7" postDate="2003-1-17">
        <postedBy>c@abc.com</postedBy>
        <postSubject>Posting 7</postSubject>
        <content>This is posting 7</content>
      </followUp>
      <followUp postId="8" postDate="2003-1-18">
        <postedBy>d@abc.com</postedBy>
        <postSubject>Posting 8</postSubject>
```

```
        <content>This is posting 8</content>
      <followUp postId="9" postDate="2003-1-19">
        <postedBy>a@abc.com</postedBy>
        <postSubject>Posting 9</postSubject>
        <content>This is posting 9</content>
      </followUp>
    </followUp>
    </followUp>
      <followUp postId="3" postDate="2003-1-13">
      <postedBy>c@abc.com</postedBy>
      <postSubject>Posting 3</postSubject>
      <content>This is posting 3</content>
    </followUp>
  </posting>
  .
  .
  .
</postings>
```

The root element is the **<postings>** element and contains a repeating group of zero or more **<posting>** subelements. The **<posting>** element has two attributes: **postId**—a unique positive integer—and **postDate** of type **date**. The **<posting>** element also contains the following subelements: **<postedBy>** of type **string**, **<postSubject>** of type **string**, **<content>** of type **string**, and zero or more **<followUp>** elements. The structure of **<followUp>** elements is identical to that of **<posting>**.

Write an XML Schema specification for the postings document and validate several instance documents against the schema.

9.1

the `color` parameter and call the `coloredCell` template. The end result is that the author cells are displayed with one background color and the title cells with another color.

Exercises

9.1 Consider the XML document containing bibliographic information introduced in Example 8.3.1. Write XQuery expressions to answer the following queries:

(a) Get articles that contain the word `"Temporal"` in their titles. Do not distinguish between upper- and lowercase letters.

(b) Get articles authored by `"Raghu Ramakrishnan"`. Do not distinguish between upper- and lowercase letters.

(c) Get the number of articles with more than three authors.

(d) Get articles that are over 40 pages long.

(e) Get a listing of URLs of articles for each author. Output should consist of author names followed by list of URLS, sorted by author names.

(f) Get author names of authors who have not written a single article over 30 pages.

9.2 Consider the XML document containing information about cities and states in the United States introduced in Example 8.3.2. Write XQuery expressions to answer the following queries:

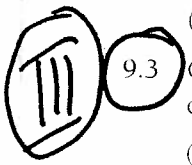
(a) Get the name and city code of the capital city of `"Georgia"`.

(b) Get the name and population of the state in which the city `"Miami"` is located.

(c) Get the number of `"Indiana"` cities in the database.

(d) Get the names of states along with their capital city names.

(e) Get the number of states whose names begin with the letter `"M"`.



9.3 Consider the XML document described in Exercise 8.7 related to the mail-order database. Write XQuery expressions to answer the following queries:

(a) Get the names of parts that cost less than 20.00.

(b) Get the names and cities of employees who have taken orders for parts costing more than 20.00.

(c) Get the names of customers who have ordered parts from employees living in Wichita.

(d) Get the names of employees who have ordered parts only from employees living in Wichita.

(e) Get the names of customers who have ordered all parts costing less than 20.00.

(f) Get the names of employees who have never made a sale to a customer living in their own zip code.

(g) Get order numbers of orders that took longer than two days to ship.

(h) Get the total price of products in order 1022.

(i) Get order number and total price for each order.

(j) Get employee numbers and total sales for each employee.

9.4 Consider the XML document described in Exercise 8.8 related to the movies database. Write XQuery expressions to answer the following queries:

(a) Get the title and years of movies in the `Crime` genre.

(b) Get names of persons who have acted in a movie and have directed it as well.

(c) Get titles and years of movies in which `James Caan` has acted.

(d) Get the names of performers and the number of movies in which they have acted.

(e) Get the names of performers who have acted in at least 10 movies and have directed at least 2 movies.

(f) Get the name(s) of the youngest performer(s).

(g) Get the names of performers who have directed some actor who is older than they are.

9.5 Consider the bibliographic XML document of Example 8.3.1. Write XSLT programs to produce three Web pages for browsing the contents of the document. The first Web page should list the `Journal` names, each of them hyperlinked to the second Web page. The second Web page should list all the volumes for the given journal, each of them hyperlinked to the third Web page. The third Web page should list all the papers in the given volume. The papers should be separated by the different numbers within the volume. All three Web pages should be well formatted and should have appropriate headings. Java servlets may be used to invoke the XSLT programs.

9.6 Consider the geography XML document of Example 8.3.2. Write XSLT programs to display the list of states and their capital cities in an HTML page in tabular format. The state name should be hyperlinked to a detail Web page for that state displaying all information for the state. Java servlets may be used to invoke the XSLT programs.

9.7 Consider the XML document described in Exercise 8.7 related to the mail-order database. Write an XSLT program that takes as input an order number and produces a well-formatted invoice as a Web page for the given order. The invoice should include customer details, employee information, as well as order details for the order. Java servlets may be used to invoke the XSLT program.

to break ties alphabetically by title.

After sorting the bindings, each binding is passed to the return-clause, in the order chosen. By substituting for the variables in the return-clause, we produce from each binding a single `Movie` element. □

12.2.11 Exercises for Section 12.2

Exercise 12.2.1: Using the product data from Figs. 12.4 and 12.5, write the following in XQuery.

a) Find the `Printer` elements with a price less than 100.

b) Find the `Printer` elements with a price less than 100, and produce the sequence of these elements surrounded by a tag `<CheapPrinters>`.

! c) Find the names of the makers of both printers and laptops.

! d) Find the names of the makers that produce at least two PC's with a speed of 3.00 or more.

! e) Find the makers such that every PC they produce has a price no more than 1000.

!! f) Produce a sequence of elements of the form

```
<Laptop><Model>x</Model><Maker>y</Maker></Laptop>
```

where x is the model number and y is the name of the maker of the laptop.

Exercise 12.2.2: Using the battleships data of Fig. 12.6, write the following in XQuery.

a) Find the names of the classes that had at least 10 guns.

b) Find the names of the ships that had at least 10 guns.

c) Find the names of the ships that were sunk.

d) Find the names of the classes with at least 3 ships.

! e) Find the names of the classes such that no ship of that class was in a battle.

!! f) Find the names of the classes that had at least two ships launched in the same year.

!! g) Produce a sequence of items of the form

```
<Battle name = x><Ship name = y />···</Battle>
```

where x is the name of a battle and y the name of a ship in the battle. There may be more than one `Ship` element in the sequence.

! **Exercise 12.2.3:** Solve the problem of Section 12.2.5: write a query that finds

```
<Ships>
    <Class name = "Kongo" type = "bc" country = "Japan"
            numGuns = "8" bore = "14" displacement = "32000">
        <Ship name = "Kongo" launched = "1913" />
        <Ship name = "Hiei" launched = "1914" />
        <Ship name = "Kirishima" launched = "1915">
            <Battle outcome = "sunk">Guadalcanal</Battle>
        </Ship>
        <Ship name = "Haruna" launched = "1915" />
    </Class>
    <Class name = "North Carolina" type = "bb" country = "USA"
            numGuns = "9" bore = "16" displacement = "37000">
        <Ship name = "North Carolina" launched = "1941" />
        <Ship name = "Washington" launched = "1941">
            <Battle outcome = "ok">Guadalcanal</Battle>
        </Ship>
    </Class>
    <Class name = "Tennessee" type = "bb" country = "USA"
            numGuns = "12" bore = "14" displacement = "32000">
        <Ship name = "Tennessee" launched = "1920">
            <Battle outcome = "ok">Surigao Strait</Battle>
        </Ship>
        <Ship name = "California" launched = "1921">
            <Battle outcome = "ok">Surigao Strait</Battle>
    </Class>
    <Class name = "King George V" type = "bb"
            country = "Great Britain"
            numGuns = "10" bore = "14" displacement = "32000">
        <Ship name = "King George V" launched = "1940" />
        <Ship name = "Prince of Wales" launched = "1941">
            <Battle outcome = "damaged">Denmark Strait</Battle>
            <Battle outcome = "sunk">Malaya</Battle>
        </Ship>
        <Ship name = "Duke of York" launched = "1941">
            <Battle outcome = "ok">North Cape</Battle>
        </Ship>
        <Ship name = "Howe" launched = "1942" />
        <Ship name = "Anson" launched = "1942" />
    </Class>
</Ships>
```

Figure 12.6: XML document containing battleship data