Listing 1: Unique songs nested in unique singers with genre as an attribute

```
<Songs>
 <Sgr name="Eagles">
  <Song genre="rock" lg="en">Hotel California</Song>
 </Sgr>
 <Sgr name="H␣Belafonte">
  <Song genre="folk" lg="cpe">Day O</Song>
  <Song genre="calypso" lg="en">Jamaica Farewell</Song>
 </Sgr>
 <Sgr name="J␣Prasad">
  <Song genre="folk" lg="pa">Mera Dil Darda</Song>
 </Sgr>
</Songs>
```

Listing 2: Singers nested within songs grouped by genre

```
<Songs>
 <rock>
  <Song lg="en">Hotel California<Sgr name="Eagles"/>
  </Song>
 </rock>
 <folk>
  <Song lg="cpe">Day O<Sgr name="H␣Belafonte"/>
  </Song>
  <Song lg="pa">Mera Dil Darda<Sgr name="J␣Prasad"/>
  </Song>
 </folk>
 <calypso>
  <Song lg="en">Jamaica Farewell<Sgr name="H␣Belafonte"/>
  </Song>
 </calypso>
</Songs>
```

1. Mark the appropriate choices to complete the following XSLT listing to transform Listing 2 into Listing 1.

```
<xsl:variable name="singers" select="_____________"/> <!-- PART (a) -->
<xsl:variable name="songs" select="//Song"/>

<xsl:template match="Songs">
 <Songs>
   <xsl:for-each select="$singers">
     <xsl:variable name="thisSong" select=".."/>
     <xsl:variable name="allSongsByThisSinger"
                   select="_____________"/> <!-- PART (b) -->
     <xsl:copy>
       <xsl:copy-of select="@*|text()"/>
       <xsl:apply-templates select="$allSongsByThisSinger"/>
     </xsl:copy>

   </xsl:for-each>
 </Songs>
</xsl:template>
```

```
<xsl:template match="Song">
   <xsl:copy>
     <xsl:attribute name="genre">
        <xsl:value-of select="_____"/>  <!-- PART (c) -->
     </xsl:attribute>
     <xsl:copy-of select="@*|text()"/>
   </xsl:copy>
</xsl:template>
```

(a) (10 points) The definition of variable singers should be

A. `//Sgr[not(@name=ancestor::*/preceding-sibling::Sgr/@name)]`

B. `//Sgr[not(@name=ancestor::*/preceding-sibling::*/descendant::Sgr/@name)]`

C. `//Sgr[not(@name=preceding-sibling::Sgr/@name)]`

D. `//Sgr[not(@name=parent::*/preceding-sibling::Sgr/@name)]`

Solution:

`//Sgr[not(@name=ancestor::*/preceding-sibling::*/descendant::Sgr/@name)]`

(b) (10 points) The definition of variable allSongsByThisSinger should be

E. `$songs[Sgr/@name=$thisSong/@name]`

F. `$songs[Sgr=$thisSong/Sgr]`

G. `$songs[Sgr/@name=$thisSong//@name]`

H. `$songs/Sgr[./@name=$thisSong/Sgr/@name]`

Solution:

`$songs[Sgr/@name=$thisSong//@name]`

(c) (10 points) The snippet in the attribute element should be

I. `ancestor::*/name()`

J. `name(..)`

K. `name()`

```
L.  ancestor-or-self::*[1]/name()
```

Solution:

name(..)

2. (10 points) Write an XML key for Listing 2 such that each singer name appears at most once in each genre. Indicate on which element this key will be placed, its selector, and field.

Solution:

3. This problem uses the SQL/XML example discussed in class, repeated below.

```
CREATE TABLE singer (
    sgrID VARCHAR2(9) NOT NULL,
    sgrName VARCHAR2(20) NOT NULL,
    sgrInfo SYS.XMLTYPE NULL,
    CONSTRAINT singer_key PRIMARY KEY (sgrID));

CREATE TABLE song (
    songID VARCHAR2(9) NOT NULL,
    songTitle VARCHAR2(20) NOT NULL,
    CONSTRAINT song_key PRIMARY KEY (songID));

CREATE TABLE sings (
    sgrID VARCHAR2(9) NOT NULL,
    songID VARCHAR2(9) NOT NULL,
    CONSTRAINT sings_key PRIMARY KEY (sgrID, songID),
    CONSTRAINT sings_singer FOREIGN KEY (sgrID) REFERENCES singer(sgrID),
    CONSTRAINT sings_song FOREIGN KEY (songID) REFERENCES song(songID));
```

Recognizing that some singers may be associated with multiple genres, we decide to change the implicit schema of the XML sgrInfo column of the singer table. Now we perform inserts such as the following:

```
INSERT INTO singer VALUES ('Sgr-02', 'H Belafonte',
      XMLParse(DOCUMENT
       '<genres><genre>reggae</genre><genre>calypso</genre></genres>'));
```

(a) (20 points) We require an SQL/XML query that finds singers that have at least two genres, and outputs their names along with the titles of each of their songs. The following is an example output from the intended SQL query:

```
SGRNAME              SONGTITLE
-------------------- --------------------
H Belafonte          Day O
H Belafonte          Jamaica Farewell
```

Specify the syntax you are using: Oracle 9i, Oracle 10g, or the SQL:2003 standard.

```
SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE ...
```

Solution: There are several possibilities, including the following in Oracle 10g syntax:

```
SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE z.sgrId = y.sgrId AND y.songID = g.songID AND
  z.sgrInfo.extract('//genres').getStringVal()
    like '%<genre>%</genre>%<genre>%</genre>%';

SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE z.sgrId = y.sgrId AND y.songID = g.songID AND
  z.sgrInfo.extract('//genre[1]/text()').getStringVal() not like
  z.sgrInfo.extract('//genre[2]/text()').getStringVal();

SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE z.sgrId = y.sgrId AND y.songID = g.songID AND
  z.sgrInfo.existsNode('//genre[2]') = 1;

SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE z.sgrId = y.sgrId AND y.songID = g.songID AND
  z.sgrInfo.existsNode('//genres/genre[2]') = 1;

SELECT z.sgrName, g.songTitle
FROM singer z, sings y, song g
WHERE z.sgrId = y.sgrId AND y.songID = g.songID AND
  z.sgrInfo.extract('//genre/text()').count() > 1;
```