XML - Managing Data Exchange/The many-to-many relationship

Learning objectives

- Learn different methods to represent a many-to-many relationship using XML
- Create XML schemas using the "Eliminate" and "ID/IDREF" methods to structure content based on a many-to-many relationship
- Create the corresponding XML documents for the "Eliminate" and "ID/IDREF" methods
- Learn to use the key function in an XML stylesheet to format data structured with the "ID/IDREF" method
- Create a basic XML stylesheet that incorporates the key function

Introduction

In the previous chapters, you learned how to use XML to structure and format data based on one-to-one and one-to-many relationships. Because XML provides the means to model data using hierarchical parent-child relationships, the one-to-one and one-to-many relationships are relatively simple to represent in XML. However, this hierarchical parent-child structure is difficult to use to model the many-to-many relationship, a common relationship between entities in many situations.

In this chapter, we will explore the pros and cons of a few methods that are used to model a many-to-many relationship in XML; these methods offer compromises in overcoming the problems that arise when applying this relationship to XML. In particular, we will see examples of how to model the many-to-many relationship using two different methods, "Eliminate" and "ID/IDREF." Additionally, in the XML stylesheet, we will learn how to implement the key function to display the data that was modeled using the "ID/IDREF" method.

Problems: many-to-many relationship

In XML, the parent-child relationship is most commonly used to represent a relationship. This can easily be applied to a one-to-one or one-to-many relationship. A many-to-many relationship is not supported directly by XML; the parent-child relationship will not work as each element may only have a single parent element. There are couple of possible solutions to get around this.

Solutions: many-to-many relationship

Eliminate

Create XML documents that eliminate the need for a many-to-many relationship

By limiting the extent of information that is conveyed, you can get around the need for a many-to-many relationship. Instead of trying to have one XML document encompass all of the information, separate the information where one document describes only one of the entities that participates in the many-to-many relationship. Using our tourGuide relationship for example, one way for us to accomplish this would be creating a separate XML document for each hotel. The relationship with amenity would ultimately then become a one-to-many. This method is more suitable for situations in which the scope of data exchange can be limited to subsets of data. However, using this method for more broadly scoped data exchange, you may repeat data several times, especially if there are many attributes. To avoid this redundancy, use the ID/IDREF method.

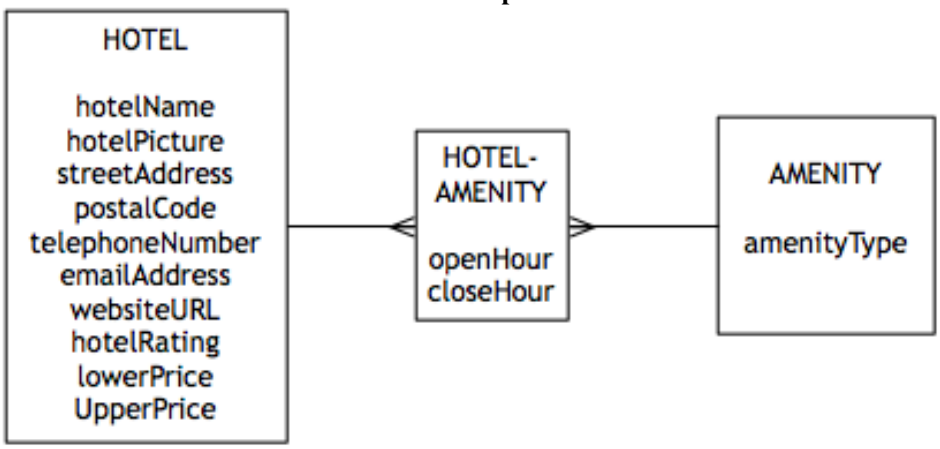
ID/IDREF

Represent the many-to-many relationship using unique identifiers

Although not the most user-friendly way to handle this problem, one way of getting around the many-to-many relationship is by creating keys that would uniquely identify each entity. To do this, an element with ID or IDREF attributes-types must be specified within the XML schema. To use a data modeling analogy, ID is similar to the primary key, and IDREF is similar to the foreign key.

Many-to-many relationship data model

Exhibit 1: Data model for a m:m relationship



The relationship reads, *a hotel can have many amenities, and an amenity can exist at many hotels*.

As you will notice, in order to represent a many-to-many relationship, two entities were added. The middle entity is necessary for the data model to represent an associative entity that stores data about the relationship between hotel and amenity. Using our Tour Guide example, "Amenity" was added to represent a list of possible amenities that a hotel can possess.

The following examples illustrate methods to represent a many-to-many relationship in XML.

Eliminate: sample solution

In this example, the many-to-many relationship has been converted to a one-to-many relationship.

XML schema

Exhibit 2: XML schema for "Eliminate" method <syntaxhighlight lang="XML"> <?xml version="1.0" encoding="UTF-8" ?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

```
<xsd:element name="hotelGuide">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="hotel" type="hotelDetails" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:simpleType name="emailAddressType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\w+\W*\w*@{1}\w+\W*\w+.\w+.*\w*"/>
    </xsd:restriction>
```

```
    </xsd:simpleType>
    <xsd:complexType name="hotelDetails">
        <xsd:sequence>
            <xsd:element name="hotelPicture"/>
            <xsd:element name="hotelName" type="xsd:string"/>
            <xsd:element name="streetAddress" type="xsd:string"/>
            <xsd:element name="postalCode" type="xsd:string" minOccurs="0"/>
            <xsd:element name="telephoneNumber" type="xsd:string"/>
            <xsd:element name="emailAddress" type="emailAddressType" minOccurs="0"/>
            <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0"/>
            <xsd:element name="hotelRating" type="xsd:integer" default="0"/>
            <xsd:element name="lowerPrice" type="xsd:positiveInteger"/>
            <xsd:element name="upperPrice" type="xsd:positiveInteger"/>
            <xsd:element name="amenity" type="amenityValue" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="amenityValue">
        <xsd:sequence>
            <xsd:element name="amenityType" type="xsd:string"/>
            <xsd:element name="amenityOpenHour" type="xsd:time"/>
            <xsd:element name="amenityCloseHour" type="xsd:time"/>
        </xsd:sequence>
    </xsd:complexType>
```

</xsd:schema> </syntaxhighlight>

XML document

Exhibit 3: XML document for "Eliminate" method <syntaxhighlight lang="XML"> <?xml version="1.0" encoding="UTF-8"?> <hotelGuide xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="amenity1.xsd">

```
    <hotel>
        <hotelPicture/>
        <hotelName>Narembeen Hotel</hotelName>
        <streetAddress>Churchill Street</streetAddress>
        <telephoneNumber>+61 (08) 9064 7272</telephoneNumber>
        <emailAddress>narempub@oz.com.au</emailAddress>
        <hotelRating>1</hotelRating>
        <lowerPrice>50</lowerPrice>
        <upperPrice>100</upperPrice>
        <amenity>
            <amenityType>Restaurant</amenityType>
            <amenityOpenHour>06:00:00</amenityOpenHour>
            <amenityCloseHour>22:00:00 </amenityCloseHour>
        </amenity>
        <amenity>
            <amenityType>Pool</amenityType>
            <amenityOpenHour>06:00:00</amenityOpenHour>
```

```
            <amenityCloseHour>18:00:00 </amenityCloseHour>
        </amenity>
        <amenity>
            <amenityType>Complimentary Breakfast</amenityType>
            <amenityOpenHour>07:00:00</amenityOpenHour>
            <amenityCloseHour>10:00:00 </amenityCloseHour>
        </amenity>
    </hotel>
    <hotel>
        <hotelPicture/>
        <hotelName>Narembeen Caravan Park</hotelName>
        <streetAddress>Currall Street</streetAddress>
        <telephoneNumber>+61 (08) 9064 7308</telephoneNumber>
        <emailAddress>naremcaravan@oz.com.au</emailAddress>
        <hotelRating>1</hotelRating>
        <lowerPrice>20</lowerPrice>
        <upperPrice>30</upperPrice>
        <amenity>
            <amenityType>Pool</amenityType>
            <amenityOpenHour>10:00:00</amenityOpenHour>
            <amenityCloseHour>22:00:00 </amenityCloseHour>
        </amenity>
    </hotel>
```

</hotelGuide> </syntaxhighlight>

ID/IDREF: sample solution

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity references "k1," using IDREF. If the IDREF does not match any ID, then the document will not validate.

XML schema

Exhibit 4: XML schema for "ID/IDREF" method <syntaxhighlight lang="XML"> <?xml version="1.0" encoding="UTF-8" ?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

```
    <xsd:element name="hotelGuide">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="hotel" type="hotelDetails" minOccurs="1" maxOccurs="unbounded"/>
                <xsd:element name="amenity" type="amenityList" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
```

```
    </xsd:element>
    <xsd:simpleType name="emailAddressType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\w+\W*\w*@{1}\w+\W*\w+.\w+.*\w*"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="hotelDetails">
        <xsd:sequence>
            <xsd:element name="hotelPicture"/>
            <xsd:element name="hotelName" type="xsd:string"/>
            <xsd:element name="streetAddress" type="xsd:string"/>
            <xsd:element name="postalCode" type="xsd:string" minOccurs="0"/>
            <xsd:element name="telephoneNumber" type="xsd:string"/>
            <xsd:element name="emailAddress" type="emailAddressType" minOccurs="0"/>
            <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0"/>
            <xsd:element name="hotelRating" type="xsd:integer" default="0"/>
            <xsd:element name="lowerPrice" type="xsd:positiveInteger"/>
            <xsd:element name="upperPrice" type="xsd:positiveInteger"/>
            <xsd:element name="amenities" type="amenityDesc" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="amenityDesc">
        <xsd:sequence>
            <xsd:element name="amenityIDREF" type="xsd:IDREF"/>
            <xsd:element name="amenityOpenHour" type="xsd:time"/>
            <xsd:element name="amenityCloseHour" type="xsd:time"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="amenityList">
        <xsd:sequence>
            <xsd:element name="amenityID" type="xsd:ID"/>
            <xsd:element name="amenityType" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

</xsd:schema> </syntaxhighlight>

XML document

Exhibit 5: XML document for "ID/IDREF" method <syntaxhighlight lang="XML"> <?xml version="1.0" encoding="UTF-8"?> <?xml-stylesheet href="amenity2.xsl" type="text/xsl" media="screen"?> <hotelGuide xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="amenity2.xsd">

```
 <hotel>
     <hotelPicture/>
     <hotelName>Narembeen Hotel</hotelName>
     <streetAddress>Churchill Street</streetAddress>
     <telephoneNumber>+61 (08) 9064 7272</telephoneNumber>
     <emailAddress>narempub@oz.com.au</emailAddress>
```

```
        <hotelRating>1</hotelRating>
        <lowerPrice>50</lowerPrice>
        <upperPrice>100</upperPrice>
        <amenities>
            <amenityIDREF>k2</amenityIDREF>
            <amenityOpenHour>06:00:00</amenityOpenHour>
            <amenityCloseHour>22:00:00 </amenityCloseHour>
        </amenities>
        <amenities>
            <amenityIDREF>k1</amenityIDREF>
            <amenityOpenHour>06:00:00</amenityOpenHour>
            <amenityCloseHour>18:00:00 </amenityCloseHour>
        </amenities>
        <amenities>
            <amenityIDREF>k5</amenityIDREF>
            <amenityOpenHour>07:00:00</amenityOpenHour>
            <amenityCloseHour>10:00:00 </amenityCloseHour>
        </amenities>
    </hotel>
    <hotel>
        <hotelPicture/>
        <hotelName>Narembeen Caravan Park</hotelName>
        <streetAddress>Currall Street</streetAddress>
        <telephoneNumber>+61 (08) 9064 7308</telephoneNumber>
        <emailAddress>naremcaravan@oz.com.au</emailAddress>
        <hotelRating>1</hotelRating>
        <lowerPrice>20</lowerPrice>
        <upperPrice>30</upperPrice>
        <amenities>
            <amenityIDREF>k1</amenityIDREF>
            <amenityOpenHour>10:00:00</amenityOpenHour>
            <amenityCloseHour>22:00:00 </amenityCloseHour>
        </amenities>
    </hotel>
    <amenity>
        <amenityID>k1</amenityID>
        <amenityType>Pool</amenityType>
    </amenity>
    <amenity>
        <amenityID>k2</amenityID>
        <amenityType>Restaurant</amenityType>
    </amenity>
    <amenity>
        <amenityID>k3</amenityID>
        <amenityType>Fitness room</amenityType>
    </amenity>
    <amenity>
```

```
        <amenityID>k4</amenityID>
        <amenityType>Complimentary breakfast</amenityType>
    </amenity>
    <amenity>
        <amenityID>k5</amenityID>
        <amenityType>in-room data port</amenityType>
    </amenity>
    <amenity>
        <amenityID>k6</amenityID>
        <amenityType>Water slide</amenityType>
    </amenity>
```

</hotelGuide> </syntaxhighlight>

Key function: XML stylesheet

In order to set up an XML stylesheet using the ID/IDREF method for a many-to-many relationship, the key function should be used. In the stylesheet, the <xsl:key> element specifies the index, which is used to return a node-set from the XML document.

A key consists of the following:

1. the node that has the key
2. the name of the key
3. the value of a key

The following XML stylesheet illustrates how to use the key function to present content that is structured in a many-to-many relationship.

XML stylesheet

Exhibit 6: XML stylesheet for "ID/IDREF" method <syntaxhighlight lang="XML"> <?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```
    <xsl:key name="amList" match="amenity" use="amenityID"/>
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>Hotel Guide</title>
            </head>
            <body>
```

```
                <xsl:apply-templates select="hotelGuide"/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="hotelGuide">
        <xsl:for-each select="hotel">
            <xsl:value-of select="hotelName"/>


            <xsl:for-each select="amenities">
```

```
                <xsl:value-of select="key('amList',amenityIDREF)/amenityType"/>
                <xsl:text>   </xsl:text>
                <xsl:value-of select="amenityOpenHour"/> -
                <xsl:value-of select="amenityCloseHour"/>


            </xsl:for-each>


        </xsl:for-each>


    </xsl:template>
```

</xsl:stylesheet> </syntaxhighlight>

Expedia.de: XML and affiliate marketing

Expedia.de is the German subsidiary of expedia.com, the internet-based travel agency headquartered in Bellevue, Washington, USA. It offers its customers the booking of airline tickets, car rentals, vacation packages and various other attractions and services via its website and by phone. Its websites attract more than 70 million visitors each month. Currently expedia.com employs 4.600 employees serving customers in the United States, Canada, the UK, France, Germany, Italy, and Australia.

For marketing purposes expedia.de set up an affiliate marketing program. Affiliate marketing is a way to reach potential customers without any financial risk for the company intending to advertise (merchant). The merchant gives website owners, which are called affiliates, the opportunity to refer to the merchant page, offering commission-based monetary rewards as incentives. In the case of Expedia.de the affiliate partners receive a commission every time users from their websites book travel on expedia.de. So the affiliates can concentrate on selling and the merchant takes care of handling the transactions.

To ease the business of the affiliate partners – and of course to make the program more attractive – Expedia.de offers its partners a service called xmlAdEd. xmlAdEd is a service providing current product information on using XML. Affiliates using this service are able to request more than 8 million of travel offerings in XML format via HTTP-request. The data is updated several times a day. In the HTTP-request you can set certain parameters such as location, price, airport code, …

The use of XML in this case gives the affiliates several advantages:
- Efficient and flexible processing of the data because of separation of structure, content and style.
- Platform-independent processing of the data.
- Lossless conversion into other file formats.
- Easy integration in their websites.
- Possibility to create an own web shop in individual design

By providing their affiliates product information in XML, expedia.de not only eases the business of their partners, but also ensures that customers receive consistent, up-to-date information on their services.

Summary

When describing a many-to-many relationship in XML, there are a few solutions available for designers to use. In choosing how to represent the many-to-many relationship, the designer not only must consider the most efficient way to represent the information, but also the audience for which the document is intended and how the document will be used.

References

http://www-128.ibm.com/developerworks/xml/library/x-xdm2m.html

http://www.w3.org/TR/xslt#key

Article Sources and Contributors

XML - Managing Data Exchange/The many-to-many relationship *Source*: http://en.wikibooks.org/w/index.php?oldid=2237265 *Contributors*: Adrignola, Christinaserrano, Cserrano, Dramo, Hopugop, Jguk, Lee J Haywood, Liblamb, Rtw, Runnerupnj, Shane, Yomomma, 61 anonymous edits

Image Sources, Licenses and Contributors

Image:XML_Hotel_amenity2.png *Source*: http://en.wikibooks.org/w/index.php?title=File:XML_Hotel_amenity2.png *License*: Public Domain *Contributors*: Christinaserrano

License