XML - Manage many-to-many relationship

1. Eliminate

We could implement the many to many relationships by limiting the extent of information to be categorized in our XML schema. 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. 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.

* Pros: easy to implement, user friendly, understandable
* Cons:
  + Hard to identify many-to-many relationship in this method
  + Many-to-many relationship may be hidden and others cannot tell
  + Not suitable for big data exchange with many attributes

2. Primary Key/Foreign Key Method

Using primary key and foreign key to enforces the many to many relationships.

For example, Concisely, “Pubs” has three tables: “Titles,” “Authors” and “TitleAuthors”. It also has relations in between three of them. And the XML schema could be as follows:

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

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

      <xs:element name="pubs">

            <xs:complexType>

                  <xs:sequence>

                        <xs:element name="Authors"  
 type="TypeAuthors" maxOccurs="unbounded" />

                        <xs:element name="Titles"  
 type="TypeTitles" maxOccurs="unbounded" />

                        <xs:element name="TitleAuthors"  
 type="TypeTitleAuthor" maxOccurs="unbounded" />

                  </xs:sequence>

            </xs:complexType>

.

.

.

      <xs:complexType name="TypeTitles">

            <xs:sequence>

                  <xs:element name="Title\_ID" type="xs:string" />

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

            </xs:sequence>

      </xs:complexType>

      <xs:complexType name="TypeAuthors">

            <xs:sequence>

                  <xs:element name="au\_id" type="xs:string" />

                  <xs:element name="au\_lname" type="xs:string" />

                  <xs:element name="au\_fname" type="xs:string" />

            </xs:sequence>

      </xs:complexType>

      <xs:complexType name="TypeTitleAuthor">

            <xs:sequence>

                  <xs:element name="au\_id" type="xs:string" />

                  <xs:element name="Title\_ID" type="xs:string" />

            </xs:sequence>

      </xs:complexType>

</xs:schema>

But there are also fragments in primary keys and foreign keys need to be added:

<xs:key name="PK\_Authors">

                  <xs:selector xpath=".//Authors" />

                  <xs:field xpath="au\_id" />

            </xs:key>

            <xs:key name="PK\_Titles">

                  <xs:selector xpath=".//Titles" />

                  <xs:field xpath="Title\_ID" />

            </xs:key>

            <xs:key name="PK\_TitleAuthors">

                  <xs:selector xpath=".//TitleAuthors" />

                  <xs:field xpath="au\_id" />

                  <xs:field xpath="Title\_ID" />

            </xs:key>

            <xs:keyref name="FK\_Authors\_TitleAuthors"  
 refer="PK\_Authors">

                  <xs:selector xpath=".//TitleAuthors" />

                  <xs:field xpath="au\_id" />

            </xs:keyref>

            <xs:keyref name="FK\_Titles\_TitleAuthors"  
 refer="PK\_Titles">

                  <xs:selector xpath=".//TitleAuthors" />

                  <xs:field xpath="Title\_ID" />

            </xs:keyref>

* Pros:
  + Easy to understand
  + Could be implemented in pure simple XML schema attributes.
* Cons:
  + Hard to implement
  + Somehow confusing comparing to one-to-one and one-to-many relationships.
  + Not suitable for large data exchange with many attributes.

3. ID/IDREF

We could represent the many-to-many relationship using unique IDs especially when encounter large data transfer. Maybe 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. ID is similar to the primary key, and IDREF is similar to the foreign key if analyzed in data modeling analogy,

* Pros:
  + Easy and efficient to identify elements by their IDs
  + Could handle large data exchange
* Cons:
  + Not user-friendly
  + extra elements and attributes may be needed