By Harish Kamath

Table of Contents

Alphabet Soup

In the first two parts of this article, I spent lots of time and space blathering on about the advanced aspects of XML schema design, including such arcane concepts as complex datatypes, derivation by extension and restriction, and type redefinition. You were probably bored out of your wits, but you nodded your head wisely throughout out of politeness, and quietly hoped that that all that jargon was a prelude to something more interesting.

I'm sorry to tell you that it isn't. In fact, this third part is filled with even more technical gobbledygook, including such beauties as "primary key reference" and "selector". None of these terms are likely to make your day any sunnier − but hey, they'll sure teach you a thing or two about designing good schemas. If that sounds like something you'd like to learn more about, keep reading − it's time to take a little detour through the supermarket!

A Day At The Supermarket

What's a supermarket got to with an XML schema, you ask wonderingly? Quite a lot, actually. You see, all supermarkets consist of aisles, with products placed neatly in each aisle for customers and employees alike. In an XML document, this design would be represented as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<supermarket name="MyMart"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<aisle name="fruits" number="1">
<item code="001" quantity="230" price="1.00"/>
<item code="002" quantity="121" price="2.45"/>
<item code="003" quantity="60" price="3.15"/>
</aisle>
<aisle name="vegetables" number="2">
<item code="004" quantity="500" price="1.15"/>
<item code="005" quantity="600" price="0.75"/>
</aisle>
</supermarket>
```

As you can see, I have a list of <aisle> elements, which in turn enclose multiple <item> elements. Each <aisle> is associated with a "name" that represents the category of items in the aisle, and a "number", which is used for easy reference. Each <item> is associated with a "quantity" and a "price".

Writing an XML schema to validate the XML document instance above is child's play, especially considering the amount of practice I've had over the last couple of weeks.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="supermarket">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="aisle"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element
name="item" maxOccurs="unbounded">

<xsd:complexType>

<xsd:simpleContent>

<xsd:extension base="xsd:string">
```

```
<xsd:attribute name="code" type="xsd:positiveInteger"/>

<xsd:attribute name="quantity" type="xsd:positiveInteger"/>

<xsd:attribute name="price" type="xsd:decimal"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute
name="name" type="xsd:string"/>
<xsd:attribute
name="number" type="xsd:positiveInteger"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Now, let's suppose that, one fine day, the store manager decides to add a few items to aisle 1. In the XML universe, he has two options available to him: he could add it to the existing <item> list for the appropriate aisle, or he could add another <aisle> element to the bottom of the document instance, reference it with the same aisle number, and attach the new items there.

```
<?xml version="1.0" encoding="UTF-8"?>
<supermarket name="MyMart"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<aisle name="fruits" number="1">
<item code="001" quantity="230" price="1.00"/>
<item code="002" quantity="121" price="2.45"/>
<item code="003" quantity="60" price="3.15"/>
</aisle>
<aisle name="vegetables" number="2">
<item code="004" quantity="500" price="1.15"/>
<item code="005" quantity="600" price="0.75"/>
</aisle>
<aisle name="fruits" number="1">
<item code="014" quantity="200" price="1.35"/>
<item code="015" quantity="300" price="0.55"/>
</aisle>
</supermarket>
```

With option two, the XML document instance doesn't look as clean as it did initially. Proceeding along this path, it would soon have a number of different entries for the same <aisle> at different locations in the document tree. Obviously, this is a maintenance nightmare.

You can prevent this from happening via the very cool <xsd:unique> element − as in the revised schema below:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="supermarket">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="aisle"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element
name="item" maxOccurs="unbounded">

<xsd:complexType>

<xsd:simpleContent>

<xsd:extension base="xsd:string">

<xsd:attribute name="code" type="xsd:positiveInteger"/>

<xsd:attribute name="quantity" type="xsd:positiveInteger"/>

<xsd:attribute name="price" type="xsd:decimal"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute
name="name" type="xsd:string"/>
<xsd:attribute
name="number" type="xsd:positiveInteger"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
```

```
</xsd:complexType>
<xsd:unique name="NoRepeatAisle">
<xsd:selector xpath="aisle"/>
<xsd:field xpath="@number"/>
</xsd:unique>
</xsd:element>
</xsd:schema>
```

I'm not going to get into a long−winded explanation of the entire schema above. Instead, let me focus on the interesting part:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!−− snip −−>
<xsd:unique name="NoRepeatAisle">
<xsd:selector xpath="aisle"/>
<xsd:field xpath="@number"/>
</xsd:unique>
<!−− snip −−>
</xsd:schema>
```

The <xsd:unique> element is what gets the ball rolling − it is used to impose uniqueness constraints on an XML document instance. You can assign it a name − I've called mine "NoRepeatAisle" − to makes its function clearer.

The <xsd:unique> element encloses <xsd:selector> and <xsd:field> elements, which help to identify the unique components of the document. The "xpath" attribute of the <xsd:selector> element contains an XPath expression that helps to limit the scope within which the uniqueness constraint will be applied. In my case, this is restricted to all the <aisle> elements that are the children of the <supermarket> element only; if there exist any other <aisle> elements in the hierarchy, this constraint is not valid.

The second element component of the uniqueness constraint is the <xsd:field> element, which specifies which attribute values should be unique − in the example above, this is the value of the "number" attribute.

Of Fruits And Vegetables

The schema design on the previous page ensures that a truant store manager cannot damage the integrity of my XML document instance by throwing up new aisles wherever (s)he likes. Now, let's take it one step further and add another integrity check, this one to ensure that the <item>s in each aisle actually exist in the store's inventory system.

Here's the updated document instance − note that, this time around, I've added an extra <items> block that serves as the inventory, matching item codes with a human−readable description of each item.

```
<?xml version="1.0" encoding="UTF−8"?>
<supermarket name="MyMart"
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance">
<aisle name="fruits" number="1">
<item code="001" quantity="230" price="1.00"/>
<item code="002" quantity="121" price="2.45"/>
<item code="003" quantity="60" price="3.15"/>
</aisle>
<aisle name="vegetables" number="2">
<item code="004" quantity="500" price="1.15"/>
<item code="005" quantity="600" price="0.75"/>
</aisle>
<items>
<item code="001">oranges</item>
<item code="002">apples</item>
<item code="003">pineapples</item>
<item code="004">onions</item>
<item code="005">potatoes</item>
</items>
</supermarket>
```

Only items that exist in the inventory "database" should appear within the various <aisle>s of the supermarket.

In order to enforce this rule, I need to update my schema design again.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="supermarket">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="aisle"
maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
```

```
<xsd:element
name="item" maxOccurs="unbounded">

<xsd:complexType>

<xsd:simpleContent>

<xsd:extension base="xsd:string">

<xsd:attribute name="code" type="xsd:positiveInteger"/>

<xsd:attribute name="quantity" type="xsd:positiveInteger"/>

<xsd:attribute name="price" type="xsd:decimal"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute
name="name" type="xsd:string"/>
<xsd:attribute
name="number" type="xsd:positiveInteger"/>
</xsd:complexType>



<xsd:keyref
name="NoIllegalEntries" refer="itemKey">
<xsd:selector
xpath="item"/>
<xsd:field
xpath="@code"/>
</xsd:keyref>



</xsd:element>
<xsd:element name="items">
<xsd:complexType>
<xsd:sequence>
<xsd:element
name="item" maxOccurs="unbounded">

<xsd:complexType>
```

```
<xsd:simpleContent>

<xsd:extension base="xsd:string">

<xsd:attribute name="code" type="xsd:positiveInteger"/>

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>



<xsd:key name="itemKey">
<xsd:selector xpath="items/item"/>
<xsd:field xpath="@code"/>
</xsd:key>



</xsd:element>
</xsd:schema>
```

In case you missed it, here's the important bit:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- snip -->
<xsd:keyref name="NoIllegalEntries" refer="itemKey">
<xsd:selector xpath="item"/>
<xsd:field xpath="@code"/>
</xsd:keyref>

<xsd:key name="itemKey">
<xsd:selector xpath="items/item"/>
<xsd:field xpath="@code"/>
</xsd:key>
<!-- snip -->
</xsd:schema>
```

In order to enforce the "only−add−those−items−to−aisles−that−exist−in−inventory" rule, I need to import two new concepts into my schema, concepts that may already be familiar to you from your work on relational databases: keys and relationships.

In the database world, a primary key uniquely identifies every record in a table − it might be a single field, or a combination of fields, but it serves as a unique fingerprint to identify any record in a table. It also serves as an important component of the relational database model − relations between different tables are created on the basis of primary keys.

Based on this knowledge, it's pretty obvious what the primary key is in the scenario above − it's the "code" attribute of each <item>. Or, to put it in schema lingo,

```
<xsd:key name="itemKey">
<xsd:selector xpath="items/item"/>
<xsd:field xpath="@code"/>
</xsd:key>
```

Key definition takes place via the <xsd:key> element, which identifies the key by a unique name. The <xsd:selector> and <xsd:field> elements are then used, in conjunction with regular XPath expressions, to drill down to the element/attribute combination representing the primary key.

Once the key is defined, the next step is to define a relationship around which it pivots. In the scenario above, it is fairly clear that the key reference has to be maintained between the <item> under the <items> element (the inventory master list) and the <item> under the <aisle> element (the inventory itself).

With this in mind, let's add a condition to the schema definition with the <xsd:keyref> element.

```
<xsd:keyref name="NoIllegalEntries" refer="itemKey">
<xsd:selector xpath="item"/>
<xsd:field xpath="@code"/>
</xsd:keyref>
```

The <xsd:keyref> element is used to indicate a reference to a key defined elsewhere in the schema. I have given the reference an appropriate name − "NoIllegalEntries" − which is displayed to the XML document author by the validator in case a violation of the reference takes place. The "refer" attribute of the <xsd:keyref> element links this reference to the primary key defined previously, via the unique key name "itemKey".

At this point, an integrity check has been added to the schema to ensure that only valid <items> from the inventory appear in the <aisle>s. You can verify this by adding an item to the aisles with a product code not listed in the inventory − your XML validator should barf and throw up lots of ugly errors.

Taking On The Fleet

Let's look at another example to better understand how keys and references work. This time, I'll leave the all−too−human world of supermarkets and travel back to that galaxy far, far away, to see exactly what's sitting in the cargo hold of two of the better−known starships in the Star Wars fleet.

```
<?xml version="1.0" encoding="UTF−8"?>
<fleet xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance">
<ship name="Jabitha">
<droid type="R2D2">2</droid>
<droid type="C3PO">4</droid>
</ship>
<ship name="Millennium Falcon">
<droid type="R2D2">1</droid>
<droid type="C3PO">1</droid>
</ship>
</fleet>
```

Here's the schema against which this document would be validated.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="fleet">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ship" type="shipType"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:complexType name="droidType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="type"
type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="shipType">
<xsd:sequence>
<xsd:element name="droid" type="droidType"
maxOccurs="unbounded"/>
</xsd:sequence>
```

```
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
```

Now, let's suppose I wanted to add a couple more droids to the Falcon. Sure, I could add another <ship> element with the same name...or I could do the smart thing, and add another <droid> element to the existing definition. As discussed in the previous example, the latter option is much cleaner, and also fairly easy to implement via the <xsd:unique> element. Here's the relevant snippet of the updated schema definition:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- snip -->

<xsd:element name="fleet">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ship" type="shipType"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
<xsd:unique name="NoRedefines">
<xsd:selector xpath="ship"/>
<xsd:field xpath="@name"/>
</xsd:unique>
</xsd:element>

<!-- snip -->

</xsd:schema>
```

In order to verify this, you can try creating two <ship> elements with the same "name", and seeing your XML validator throw up all over the screen. It's always fun to watch, and it doesn't hurt anything!

Breaking The Mold

Next, how about introducing a referential integrity constraint similar to the one in the example above? Let's say we have a master list of available droid types, and only those droid types may be requisitioned for the various ships in the fleet. Here's my new XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<fleet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ship name="Jabitha">
<communication>
<droid type="C3PO">4</droid>
</communication>
<repairs>
<droid type="R2D2">6</droid>
<droid type="LE-BO2D9">1</droid>
</repairs>
<administration>
<droid type="CZ-3">1</droid>
</administration>
</ship>

<ship name="Millennium Falcon">
<communication>
<droid type="C3PO">1</droid>
</communication>
<repairs>
<droid type="R2D2">1</droid>
<droid type="LE-BO2D9">4</droid>
</repairs>
<administration>
<droid type="CZ-3">5</droid>
</administration>
</ship>

<droids>
<droid type="R2D2">astromech droid</droid>
<droid type="C3PO">protocol droid</droid>
<droid type="CZ-3">humanoid droid</droid>
<droid type="LE-BO2D9">repair droid</droid>
</droids>

</fleet>
```

There are two significant changes in this version of the XML document. First, I have introduced a listing of droids which will act as a master list for all the droids on the ships of the fleet. To make things easier, I have further classified the droids on each ship into sections like administration, communication and repairs, based

on their advertised functionality.

Now, all I need to do is update the schema to reflect this referential integrity constraint, in a manner similar to that in the previous example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="fleet">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ship" type="shipType"
maxOccurs="unbounded"/>
<xsd:element name="droids"
type="droidsGroupType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:unique name="NoRedefines">
<xsd:selector xpath="ship"/>
<xsd:field xpath="@name"/>
</xsd:unique>

<xsd:key name="droidNameKey">
<xsd:selector xpath="droids/droid"/>
<xsd:field xpath="@type"/>
</xsd:key>
</xsd:element>

<xsd:complexType name="droidType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="type"
type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="shipType">
<xsd:sequence>
<xsd:element name="communication"
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesCommunication"
refer="droidNameKey">
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>
<xsd:element name="repairs"
```

```
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesRepair"
refer="droidNameKey">
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>
<xsd:element name="administration"
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesAdministration"
refer="droidNameKey">
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="droidsGroupType">
<xsd:sequence>
<xsd:element name="droid" type="droidType"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

First, I have to define a key for my master list of droids − the droid code will do very well here.

```
<xsd:key name="droidNameKey">
<xsd:selector xpath="droids/droid"/>
<xsd:field xpath="@type"/>
</xsd:key>
```

I can now reference this key within each functional category on each <ship> via the <xsd:keyref> element:

```
<xsd:complexType name="shipType">

<xsd:sequence>

<xsd:element name="communication"
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesCommunication"
refer="droidNameKey">
```

```
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>

<xsd:element name="repairs"
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesRepair"
refer="droidNameKey">
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>

<xsd:element name="administration"
type="droidsGroupType" maxOccurs="unbounded">
<xsd:keyref name="NoFakesAdministration"
refer="droidNameKey">
<xsd:selector xpath="droid"/>
<xsd:field xpath="@type"/>
</xsd:keyref>
</xsd:element>

</xsd:sequence>

<xsd:attribute name="name" type="xsd:string"/>

</xsd:complexType>
```

As I have three functional groups for droids on each ship, the key reference needs to be repeated thrice, one for each group (you could probably do this in a more efficient manner if you have a large number of groups − I leave that to you as an exercise, preferring this slightly clunkier option for illustrative purposes). The "refer" attribute of each <xsd:keyref> element links this constraint to the droid master list via the unique label "droidNameKey".

And that's it! You now have two constraints in a single schema definition, one ensuring that ship names are unique, and the other ensuring that only valid droids appear in each ship. Try it out and see for yourself!

Two For One

Now, how about letting me twist your mind a little further? In the example on the previous page, I stated that there could only be one entry for each ship in the fleet. Let's now modify that statement a little and permit repetitions, so long as the the combination of ship name and droid code is unique.

The following XML document sample might make this clearer:

```
<?xml version="1.0" encoding="UTF-8"?>
<fleet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<ship name="Jabitha">
<droid type="R2D2">2</droid>
</ship>

<ship name="Jabitha">
<droid type="C3P0">5</droid>
</ship>

<ship name="Millennium Falcon">
<droid type="C3PO">4</droid>
</ship>

<droids>
<droid type="R2D2">astromech droid</droid>
<droid type="C3PO">protocol droid</droid>
<droid type="CZ-3">humanoid droid</droid>
<droid type="LE-BO2D9">repair droid</droid>
</droids>

</fleet>
```

As you can see above, my document includes more than one entry for a particular ship name. However, the combination of ship and droid is unique. In order to ensure that this rule is followed consistently, I need to update my schema definition to use what is known as a "unique composed value". A unique composed value specifies uniqueness of an element by using two (or more) parameters.

Next question: how do you defined a unique composed value?

Take a look:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="fleet">
<xsd:complexType>
```

```
<xsd:sequence>
<xsd:element name="ship" type="shipType"
minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="droids"
type="droidsGroupType"/>
</xsd:sequence>
</xsd:complexType>
<xsd:unique name="NoDuplicates">
<xsd:selector xpath="ship"/>
<xsd:field xpath="@name"/>
<xsd:field xpath="droid/@type"/>
</xsd:unique>
</xsd:element>

<xsd:complexType name="droidType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="type"
type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="shipType">
<xsd:sequence>
<xsd:element name="droid" type="droidType"
minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="droidsGroupType">
<xsd:sequence>
<xsd:element name="droid" type="droidType"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

It's pretty simple − all I've done is add an additional field to the <xsd:unique> element, in order to ensure that only the combined set of ship name and droid type is unique. You don't have to stop there either − a unique composed value may be made up of as many components as you like.

How do I know that this actually works? Create a duplicate entry in the XML document above, validate the file, and see what happens. Your XML validator should start complaining bitterly about uniqueness constraint violations.

And that's about it for the moment. In this article, I introduced you to the concept of uniqueness in the XML Schema world, showing you how to use built−in schema constructs to enforce uniqueness within your XML document instances. I also showed you how you could replicate RDBMS referential integrity constraints within the context of an XML document, using schema equivalents of primary and foreign keys, and creating relationships between the different nodes of an XML document. All these techniques come in handy to reduce errors when you're building schema definitions which contain internal inter−relationships that need to be rigidly enforced across document instances.

In the next (and final) article in this series, I'll be wrapping up this discussion of advanced schema theory with an overview of XML namespaces, and how they fit into the XML Schema picture. Make sure you come back for that one!

Note: All examples in this article have been tested on Linux/i586. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!