ABSTRACT

With the conceptualization of next generation architecture for Data Warehousing, that is the DW 2.0, there is now an increased emphasis on using fully-temporalized databases, in particular with approaches such as the Data Vault. In this paper we present a template XML Schema for Data Vault model concepts (at a metamodel level) and a process for creating XML Schemas for data warehouse designs represented as Data Vault models. These templates can be used to describe and present data from disparate systems in a structured format suitable for exchange and loading into a data warehouse.

KEYWORDS: Data Warehouse, DW 2.0, XML Schema, Data Vault, temporal database, semantically temporal data, semantically stable data, semantic data change.

INTRODUCTION

The standard Data Warehouse (DW) design practice is using Dimensional Modeling for logical representation of the reporting side of the DW. The significance of this effort is, on the staging side, in addressing exchange of data from, to, and among the next generation of data warehouses using universal XML format. The scope of the DW staging side, in respect to some real system and its transactional database, within the DW system is shown in Figure 1. The function of data staging is to preserve source data and output transformation to prepare data to be used for analytical processing and reporting. An XML Schema format is needed to accommodate the related data exchanges in a distributed environment.

The specification for Data Warehousing 2.0 (DW 2.0) is a result of efforts led by Inmon [3]. A key feature in DW 2.0 is the ability to support changes of data over time. The authors of DW 2.0 address the question of proper temporal data storage and handling by saying, “The answer is that semantically static and semantically temporal data should be physically separate in all database designs” [3]. Semantically temporal data is likely to undergo semantic data change whereas semantically stable data is unlikely to undergo such change. Semantic data change occurs when the definition of a data entity changes (i.e., new columns are added to an existing entity). This is different from content data change, where the value of a data entity changes (i.e., an amount increases from $100 to $200).

The designers of DW 2.0 point out that, in an environment where temporal and stable data are grouped together, many changes in business requirements often cause the technology infrastructure to change [3]. However, separating temporal and stable data can mitigate such an effect. At the point where business change occurs, a new snapshot of the semantics can simply be created and delimited by a time factor using to and from dates. A history of these changes can then be reconstructed by extracting the deltas between successive snapshots [5]. Capturing these snapshots generates a historical record of data by which an analyst or end-user can easily locate and retrieve information.
while also providing a technological infrastructure that can withstand change over time [3].

The Data Vault (DV) is a modeling technique specifically designed to address temporal data loading for data warehouses [12]. DV holds promise for DW 2.0 because it supports evolving transformations of data, allows structural changes by addition only, provides for loading into a staging area without delays, and defers reporting-related data transformations that are beyond the scope of the staging process. Due to these factors, the DV model is recognized as a recommended choice for DW design and implementation within the DW 2.0 architecture [15]. There are currently no available schemas for linking DV models to fully-temporalized databases. To address this problem, this paper seeks to propose a template XML Schema for DV model concepts and a process for creating XML Schemas for data warehouse designs as DV models.

We start with a simple example of data source semantics which will later help to visualize DV concepts. In Figure 2, consider source data from independent databases sharing Suppliers where System 1 tracks Parts and the Suppliers who

Figure 2. Source Data Model

Figure 3. An Example Data Vault Model
provide them and System 2 tracks Contracts that the company currently has with Suppliers. Contracts are not tracked in System 1 and Parts are not tracked in System 2. We want to bring the relationships between Parts, Contracts, and Suppliers together into one DV for analytics and reporting purposes.

The resulting DV data model (as seen in Figure 3) provides historical storage of data coming in to a data warehouse from multiple systems and attempts to solve the problem of semantic data change by separating business keys and the association between those keys from descriptive attributes of the key [9]. In a logical design, the DV model is composed of 4 basic components: the hub, link, satellite, and reference table [9]. Starting with the set of source data systems from Figure 2, an integrated DV model (representing HUBs and LINKs in light blue and beige colors respectively), is shown in Figure 3.

Hubs represent a list of unique stable business keys unlikely to change over time (i.e., Suppliers or Parts). Links represent transactions between hubs (i.e., relating Supplier to Part through a Delivery transaction). Satellites represent the descriptive attributes of a hub (i.e., a Supplier has a Name, Address, etc.). Reference tables are fixed lists of values which may be used frequently, and they are used to simplify the model (avoiding unessential links and satellites, such as States and Contract Terms).

THE DATA VAULT METAMODEL

For purposes of data exchange involving data warehouses either as the target or the source of the exchanges, an abstract metamodel of the DV must be produced and mapped to a general XML Schema for the DV, giving database analysts and designers a standardized template for exchanging complex fully-temporalized data among disparate systems in a structured XML format.

First, a formal definition for each DV model component is established as shown in Definitions 1-14 below. These definitions are modeled after a formalization of the Anchor Model [15] [16], which is a comparative technique also used to model temporal data. From these definitions, we can form a DV metamodel (see Figure 4) which outlines a structure of elements comprising the basic DV components that can be included in an XML Schema definition document (XSD).

Definition 1 (Identities). Let $U$ be a finite but arbitrarily large set of symbols, used as identities.

Definition 2 (Data type). Let $D$ be a data type. The domain of $D$ is a set of data values.

Definition 3 (Time type). Let $T$ be a time type. The domain of $T$ is a set of time values.

Definition 4 (Hub). A hub $H$ is a string. An extension of a hub is a subset of $U$.

Definition 5 (Reference Table). A reference table $R$ is a string. A reference table has a domain, which is $U$. A reference table has a range, which is a data type $D$. An extension of a reference table $R$ with range $D$ is a bijective relation over $U \times D$.

Definition 6 (Satellite). A satellite $S$ is a string. A satellite $S$ has a hub $H$ or a link $L$ for a domain, a data type $D$ for range, and a time type $T$ as time range. An extension of a satellite $S$ is a relation over $U \times D \times T$.

Definition 7 (Referenced Satellite). A referenced satellite $SR$ is a string. A referenced satellite $SR$ has a hub $H$ or a link $L$ for a domain, a referenced table value $R$ for range, and a time type $T$ for time range. An extension of a referenced satellite $SR$ is a relation over $U \times U \times T$.

Definition 8 (Hub Role). A hub role $R_h$ is a string. Every hub role has a type, which is a hub.

Figure 4. Data Vault Metamodel using Idef1X notation
Definition 9 (Link Role). A link role $R_i$ is a string. Every link role has a type, which is a link.

Definition 10 (Reference Role). A reference role $R_j$ is a string. Every reference role has a type, which is a referenced table value.

Definition 11 (Link). A link $L$ is a set of at least two hub (or link) roles and a time type $T$. An instance $L_i$ of a link $L = \{R_1, \ldots, R_n, S_j, \ldots S_m, T\}$ is a set of pairs $[R_i, v_i], i = 1, \ldots, n$, and a time point $p$, where $R_i$ is one hub or link role, $v_i \in U$, $p \in T$, and $n \geq 2$. An extension of $L$ is a set of instances of $L_i$.

Definition 12 (Referenced Link). A referenced link $L_a$ is a set of at least two hub or link roles, one or more reference roles and a time type $T$. An instance $L_{ai}$ of a referenced link $L_a = \{R_1, \ldots, R_n, S_j, \ldots S_m, T\}$ is a set of pairs $[R_i, v_i], i = 1, \ldots, n$, $[S_j, w_j], j = 1, \ldots, m$, and a time point $p$, where $R_i$ is one hub or link role, $S_j$ is one reference role, $v_i \in U$, $w_j \in U$, $p \in T$, $n \geq 2$, and $m \geq 1$. An extension of a referenced link $L_a$ is a set of instances of $L_{ai}$.

Definition 13 (Identifier). Let $L$ be a link. An identifier for $L$ is a subset of $L$ containing at least one hub or link role, and every identifier for $L$ must contain $L$.

Definition 14 (C-DV). Let $C$ be a conceptual Data Vault model. An identifier for $C-DV$ is a relation over $U \times H \times L \times S \times R \times T \times D$.

Notice in Figure 4 that the HUB, HUB_SATELLITE, LINK_SATELLITE, LINK, and REFERENCE entities represent the 4 basic DV components. The HUB_SATELLITE represents satellites associated to hubs and LINK_SATELLITE represents satellites associated to links. Notice the existence of the hubKey and loadDate in the HUB, HUB_SATELLITE, and LINK_SATELLITE. The hubKey is a surrogate key uniquely identifying which a hub a satellite belongs to. The loadDate is not a part of the surrogate key in a hub, but is used in the satellite as a composite key with the hubKey providing a historical time component. Each LINK also has a loadDate. However, it is not a part of the link’s key. The REF_HUB_SATELLITE and REF_LINK_SATELLITE entities represent referenced value satellites and the REF_LINK entity represents referenced value links.

The HUB_ROLE entity identifies which hub is associated via link with other hubs. The LINK_ROLE entity identifies which, if any, links are associated with other links. The REF_ROLE entity identifies which, if any, reference table values are associated with a link. At least two hubs are required in every link and all such hubs combine to form a composite key for the link. This union of involved hub keys is represented semantically as a helper entity ROLE_SET via a role_set_union_id. The semantic use of link_id in these entities is intended to represent a union of all hubKey_role and linkKey_role identifiers (i.e., all unique HUB and LINK roles associated with the link).

As can also be seen from Figure 4, cardinality is used to express each relationship in the DV model. A referenced satellite must contain one and only one reference table value while a reference table value may be in zero or more referenced satellites. A satellite will only be present in one hub while a hub will contain one or more satellites. A hub may be in zero or more links while a link will contain two or more hub roles (and possibly other link roles, i.e. indirect originating hub roles). A reference table value may be in zero or more referenced links while a referenced link will contain one or more reference roles. A link may be associated with zero or more other links through link roles.

**GENERIC XML SCHEMA FOR DATA VAULT METAMODEL**

A XSD provides a means for defining the structure, content, and semantics of XML documents [20]. Having identified the basic components of the DV metamodel, we can now build a standard XSD format which will express the DV schema (see Appendix A).

**Schema Namespaces**

For all XSDs, a default namespace must be declared which provides a unique identifier for all element and attribute declarations [21]. We will use the World Wide Web Consortium’s (W3C) standard XML Schema namespace (http://www.w3.org/2001/XMLSchema) as our default. A target namespace must also be declared which will serve as the namespace assigned to the schema we are creating. This will be the namespace which XML instances will use to access their declared types. For the DV schema, we will create a namespace of our own (http://localhost/DataVault/schema). We will qualify the namespace of all XML elements regardless of whether they are defined in the global or local namespace of the XSD. The schema namespace declarations for our XSD can be seen in Appendix A, line 2.

**XML Schema Design Pattern selection**

The four common patterns [7] used for outlining design of XML Schemas are the Russian Doll, Salami Slice, Venetian Blind, and Garden of Eden. They each differ according to the number of their global elements and types. The Russian Doll schema contains a single global element with all element declarations nested within the single global declaration. However element attributes are not combined into complex types and are not reusable. The result is a set of repeating attribute declarations within the schema. When developing a schema template, we want to make efficient use of reusable components if possible. Both Salami Slice and Garden of Eden schemas contain multiple global elements and types, resulting in many roots within the schema. For a schema template of a metamodel representation, it is ideal to have all elements contained within a single root system. Therefore, we will employ the Venetian Blind pattern which contains only one global element with all other elements being local. In Venetian Blind, element declarations are nested within a single global declaration by means of simple types, complex types and named element groups. All types and element groups are reusable throughout the schema, making it suitable for use by both instance and schema developers [7]. Our single global element will be named “schema” (see Appendix A, line 48) since we are endeavoring to define a DV modeling schema through our XML, and not actual data.

**Elements and Attributes**

Elements represent data produced or consumed by an application and attributes represent metadata describing an element. Only the root “schema” element will be defined in the global namespace. All other elements will be defined locally and will use types defined in the global namespace. Elements will indicate a type (usually a reference to a pre-defined complex type) and a minimum/maximum number of occurrences when included in a sequence (through the use of minOccurs and maxOccurs).
For instance, the “minOccurs” is set to 1 for the satellite sequence inside the hub complex type (see Appendix A, line 23), specifying that each hub has at least one satellite. The “minOccurs” is similarly set to 2 for hubRoles in a link and 0 for refRoles and linkRoles in a link (see Appendix A, lines 33-35), specifying that each link must be associated with at least two hubs and may or may not have associated reference table values and/or links.

All attributes will be defined locally within the context of an element and will indicate type (usually a reference to a simple type), usage (required or optional), and a default flag. For instance, the default value for a role identifier is “false” (see Appendix A, line 6).

Simple Types

XML Schema simple types indicate elements containing unstructured text. Restricting a simple data type reduces possible values of the type while extending it in essence creates a complex type with attribute content. We will use simple data types defined in the default XML Schema namespace to describe attributes of DV schema components. As seen in Table 1, the various attributes of the DV metamodel in Figure 4 will be assigned to the simple data types of string, integer, date, or boolean.

Complex Types

Reusable complex types will be created describing each of the main DV model components and their properties.

Reference Tables

The reference table complex type ("reftable") contains a name, identity, and description (see Figure 5). The name is a string used to identify the name of the reference table (i.e., table STATES in Figure 3). The identity is a string used to identify a unique key value within each reference table (i.e., STATES.STATE_NAME in Figure 3). The description is a string used to describe the unique key value within each reference table (i.e., STATES.STATE_NAME in Figure 3). The XSD reference table definition can be seen in Appendix A, lines 8-12.

Satellites

The complex type satellite contains a name, hubKey, loadDate, recSource, and optional value and refValue attributes (see Figure 6). The hubKey is a unique integer key used to identify the hub a satellite is associated with (i.e., SAT_SUPPLIER_SNAME.SUPPLIER_ID in Figure 3). The loadDate is the satellite’s date of initial load into the DV (i.e., SAT_SUPPLIER_SNAME.LOAD_DATE in Figure 3). The recSource is a string indicating the source system from which the data came into the DV (i.e., SAT_SUPPLIER_SNAME.RECORD_SOURCE in Figure 3). The value and refValue attributes are strings representing actual data values for the satellite (i.e., SAT_SUPPLIER_SNAME.SNAME (non-ref) and SAT_SUPPLIER_ADDRESS.STATE_ID (ref) in Figure 3).

The value and refValue attributes are present depending upon the type of satellite we are dealing with. Non-referenced satellites have a value and reference satellites have a refValue.

Hubs

Hubs contain a name, hubKey, loadDate, recSource, and a busKey (see Figure 7). The hubKey is a unique integer key used to identify the hub (i.e., HUB_SUPPLIERS.SUPPLIER_ID in Figure 3). The loadDate is the hub’s date of initial load into the DV (i.e., HUB_SUPPLIERS.LOAD_DATE in Figure 3). The recSource is a string indicating the source system from which the data came into the DV (i.e., HUB_SUPPLIERS.RECORD_SOURCE in Figure 3).

Hubs also contain a sequence of satellites. Note that each hub must contain at least one satellite, and that satellites are maintained as elements in the hierarchical structure of the hub. The busKey is a string representing a business key used in conjunction with the recSource to identify a hub record in its originating system (i.e.,

---

**Table 1.** XML Simple Types for Data Vault Schema Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>xs:string</td>
</tr>
<tr>
<td>hubKey</td>
<td>xs:integer</td>
</tr>
<tr>
<td>loadDate</td>
<td>xs:date</td>
</tr>
<tr>
<td>recSource</td>
<td>xs:string</td>
</tr>
<tr>
<td>busKey</td>
<td>xs:string</td>
</tr>
<tr>
<td>identity</td>
<td>xs:string</td>
</tr>
<tr>
<td>description</td>
<td>xs:string</td>
</tr>
<tr>
<td>value</td>
<td>xs:string</td>
</tr>
<tr>
<td>refValue</td>
<td>xs:string</td>
</tr>
<tr>
<td>role</td>
<td>xs:string</td>
</tr>
<tr>
<td>type</td>
<td>xs:string</td>
</tr>
<tr>
<td>identifier</td>
<td>xs:boolean</td>
</tr>
</tbody>
</table>
HUB_SUPPLIERS.SUPPLIER_BK in Figure 3). The XSD hub
definition can be seen in Appendix A, lines 21-30.

Links

Links form relationships between hubs, reference tables and
other links. Links are composed of at least two hub elements and
any number of reference table and/or link elements. Since more
than one hub, reference table, or link may appear in the same
link, the hubRole, refRole, and linkRole are introduced in order
to qualify their use inside the link. To accommodate hubRoles,
refRoles, and linkRoles, the complex type role is produced (see
Figure 8). The role complex type consists of a role, type, and
identifier flag. The type will usually be the same as the name
of the component being modeled. The role will consist of short
descriptive text defining the relationship between the component
and other components associated with the link. The identifier flag
will be true or false depending upon whether the object is part of
the unique identifier of the link or not. The XSD role definition
can be seen in Appendix A, lines 3-7.

The link complex type consists of two or more hubRole
elements, zero or more refRole elements, zero or more linkRole
elements, zero or more satellites, a loadDate, and a recSource (see
Figure 9). The hubRole represents a role between a hub and a link,
the refRole represents a role between a reference table and a link,
the linkRole represents a role between two links. The loadDate is
the link’s date of initial load into the DV (i.e., LINK_SUPPLIER-
PARTS.LOAD_DATE in Figure 3). The recSource is a string
indicating the source system from which the data came into the
DV (i.e., LINK_SUPPLIER_PARTS.RECORD_SOURCE in
Figure 3).

Similar to satellites, links can be referenced or non-referenced.
Referenced links will have one or more linkRole elements. Non-
referenced links will have zero linkRole elements. The link also
contains a sequence of satellites. The XSD link definition can be
seen in Appendix A, lines 31-40.

XML SCHEMAS FOR
EXAMPLE DATA VAULT MODELS

Using the template XML Schema for the DV metamodel,
we can now craft XML schemas for any DV model (that is a
data warehouse example design) logically derived from some
source data Universe of Discourse (UoD). A two step process is
necessary as actual data content (i.e., XML documents containing
data record sets from/fpr a particular DV data warehouse) has
to satisfy specific schemas (referring to specific hubs, links and
satellites).

As an example, we present a college registration system UoD.
The UoD being modeled tracks course enrollment for students.
Business entities are the courses offered by the college, locations
on campus where those courses are taught, departments offering the
courses, professors teaching the courses, and students enrolled in
the courses. Each business entity has one or more attributes which
describe it. Courses have an abbreviation which never changes
and a name which may change over time. If a course abbreviation
ever changes, it is deemed to be a new course offering. Locations
consist of a building name/number and a room number which
never changes. Departments have a name which can possibly
change over time. Professors have a name which we assume will
never change. Students also have a name and gender which never
changes, and an address and major which can change. Several
business entities are related to each other. Professors work in
certain departments. Courses are offered by certain departments.
Students are advised by certain professors as they work towards certain degrees. Furthermore, courses are taught by certain professors, are held in certain locations, and are attended by certain students. The same course is taught many terms and many sections of the same course may be taught in a single term.

Figure 10 shows the DV model for our college registration system and Appendix B presents the accompanying XML Schema definition for this model. Business entities are represented by hubs HUB_STUDENT, HUB_COURSE, HUB_LOCATION, HUB_DEPARTMENT, and HUB_PROFESSOR. Each hub has one of more satellites (i.e., HUB_STUDENT has satellites SAT_STUDENT_NAME, SAT_STUDENT_ADDRESS, SAT_STUDENT_MAJOR, and SAT_STUDENT_GENDER). Some satellites are associated with reference tables (i.e., SAT_STUDENT_ADDRESS.STATE_CODE references table STATE which indicates there are only 50 states available to be used in an address and SAT_STUDENT_GENDER references table GENDER which indicates there are only 2 genders a student can be). Several links exist between hubs. One example is LINK_DEPARTMENT_COURSE (“course is offered by department”). Several links use reference tables. For example, link LINK_COURSE_STUDENT (“student is enrolled in course”) contains references to tables SECTION and TERM indicating the section and term of the course the student is enrolled in.
CONCLUSION

This paper has presented a methodology for mapping DV schema to a standard XML Schema format. Due to its flexibility, extensibility, and scalability, XML Schema is the preferred means of describing and exchanging structured data across the internet and other networked resources. By defining a DV to XML methodology, fully-temporalized data can be presented to disparate systems in a structured format suitable for data exchange. This capability is especially significant in the case of the DV where the sole purpose of the model is to facilitate imports of data from various remote systems into one centralized data warehouse and frequently to export data from a DV to distributed users and/or data marts.

It should be noted that, while the DV XML schema provides a framework for data exchange, there are some limitations that exist. However, each of these can easily be overcome.

1. **Identifying Hub Keys**

   In the case of the DV, hub keys will not always be known by the source system providing the data, and therefore will not always be included in the XML. However, by using a combination of the agreed-upon business key and other identifying data from the source system, hub keys can be generated for inserts or retrieved for updates.

2. **Data Deletions and Errors**

   By nature, DVs do not allow for physical deletion of data [11]. If you want to record a deletion from the source system, a satellite needs to be created in the XML which will represent an active/delete flag bit for the associated hub. A DV load process should also never fail due to data errors in the source system. The DV records everything as a fact, even errors. XML is uniquely fitted to handle this requirement as it is completely flexible with regard to text content. In order to get incorrect data into the DV, default values can be applied for elements to correct data type mismatches and zero keys can be employed to correct null key values [8]. Logging can also be used for error tracking and correction.

3. **Data Type Variation**

   One satellite in the DV may be represented by two different data types in two different source systems. To handle both, the XSD can be treated as a raw data staging table by using string or varchar as default data types for all incoming elements. Once the XML is deserialized, type conversions can be performed to conform to your DV specifications.

Having a standard XML Schema format for a fully-temporalized DV is particularly relevant to DW 2.0. The authors of DW 2.0 have placed emphasis on support of temporal databases and have addressed the separation of semantically static and semantically temporal data as a key component of their architecture. To this end, they propose maintaining historical data snapshots at points where business change occurs with minimal or no change to existing infrastructure. Capturing such snapshots generates a historical record of data by which an analyst or end-user can easily locate and retrieve information while also providing a technological infrastructure for the business that can withstand change over time. The approach of creating a standard XSD format for the DV model is an extensible solution allowing businesses to attain this end.

REFERENCES


APPENDIX A. DATA VAULT METAMODEL XML SCHEMA

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://localhost/DataVault/schema"
  targetNamespace="http://localhost/DataVault/schema"
  elementFormDefault="qualified">
  <xs:complexType name="role">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="role" type="xs:string" use="required"/>
    <xs:attribute name="identifier" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
  <xs:complexType name="reftable">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="identity" type="xs:string" use="required"/>
    <xs:attribute name="description" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="satellite">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="hubKey" type="xs:integer" use="required"/>
    <xs:attribute name="loadDate" type="xs:date" use="required"/>
    <xs:attribute name="recSource" type="xs:string" use="required"/>
    <xs:attribute name="value" type="xs:string" use="optional"/>
    <xs:attribute name="refValue" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:complexType name="hub">
    <xs:sequence>
      <xs:element name="satellite" type="satellite" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="hubKey" type="xs:integer" use="required"/>
    <xs:attribute name="loadDate" type="xs:date" use="required"/>
    <xs:attribute name="recSource" type="xs:string" use="required"/>
    <xs:attribute name="busKey" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="link">
    <xs:sequence>
      <xs:element name="hubRole" type="role" minOccurs="2" maxOccurs="unbounded"/>
      <xs:element name="refRole" type="role" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="linkRole" type="role" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="satellite" type="satellite" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="loadDate" type="xs:date" use="required"/>
    <xs:attribute name="recSource" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="schema">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="reftable" type="reftable"/>
      <xs:element name="hub" type="hub"/>
      <xs:element name="link" type="link"/>
    </xs:choice>
  </xs:complexType>
  <xs:element name="schema" type="schema"/>
</xs:schema>
```
APPENDIX B. COLLEGE REGISTRATION DATA VAULT XML SCHEMA

```xml
<?xml version="1.0"?>
<schema xmlns="http://localhost/DataVault/schema">
  <reftable name="STATE" identity="string" description="string" />
  <reftable name="GENDER" identity="string" description="string" />
  <reftable name="MAJOR" identity="string" description="string" />
  <reftable name="DEGREE" identity="string" description="string" />
  <reftable name="SECTION" identity="string" description="string" />
  <hub name="HUB_STUDENT" hubKey="integer" loadDate="date" recSource="string" busKey="string"
refValue="STATE" />
    <satellite name="SAT_STUDENT_ADDRESS" hubKey="integer" loadDate="date" recSource="string"
value="string" />
    <satellite name="SAT_STUDENT_NAME" hubKey="integer" loadDate="date" recSource="string"
value="string" />
    <satellite name="SAT_STUDENT_GENDER" hubKey="integer" loadDate="date" recSource="string"
refValue="GENDER" />
    <satellite name="SAT_STUDENT_MAJOR" hubKey="integer" loadDate="date" recSource="string"
refValue="MAJOR" />
  </hub>
  <hub name="HUB_LOCATION" hubKey="integer" loadDate="date" recSource="string" busKey="string"
value="string" />
    <satellite name="SAT_LOCATION_BUILDING" hubKey="integer" loadDate="date" recSource="string"
value="string" />
    <satellite name="SAT_LOCATION_ROOMNO" hubKey="integer" loadDate="date" recSource="string"
value="string" />
  </hub>
  <hub name="HUB_PROFESSOR" hubKey="integer" loadDate="date" recSource="string" busKey="string"
value="string" />
    <satellite name="SAT_PROFESSOR_NAME" hubKey="integer" loadDate="date" recSource="string"
value="string" />
  </hub>
  <hub name="HUB_COURSE" hubKey="integer" loadDate="date" recSource="string" busKey="string"
value="string" />
    <satellite name="SAT_COURSE_ABBREV" hubKey="integer" loadDate="date" recSource="string"
value="string" />
    <satellite name="SAT_COURSE_TITLE" hubKey="integer" loadDate="date" recSource="string"
value="string" />
  </hub>
  <hub name="HUB_DEPARTMENT" hubKey="integer" loadDate="date" recSource="string" busKey="string"
value="string" />
    <satellite name="SAT_DEPARTMENT_NAME" hubKey="integer" loadDate="date" recSource="string"
value="string" />
  </hub>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_STUDENT" role="advised" identifier="true" />
    <hubRole type="HUB_PROFESSOR" role="by" identifier="true" />
    <refRole type="DEGREE" role="pursuing" identifier="false" />
  </link>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_PROFESSOR" role="works" identifier="true" />
    <hubRole type="HUB_DEPARTMENT" role="in" identifier="true" />
  </link>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_COURSE" role="offered" identifier="true" />
    <hubRole type="HUB_DEPARTMENT" role="by" identifier="true" />
    <refRole type="TERM" role="during" identifier="true" />
    <refRole key="SECTION" role="having" identifier="true" />
  </link>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_STUDENT" role="enrolled" identifier="true" />
    <hubRole type="HUB_COURSE" role="in" identifier="true" />
    <refRole type="TERM" role="during" identifier="true" />
    <refRole key="SECTION" role="having" identifier="true" />
  </link>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_COURSE" role="taught" identifier="true" />
    <hubRole type="HUB_PROFESSOR" role="by" identifier="true" />
    <refRole type="TERM" role="during" identifier="true" />
    <refRole key="SECTION" role="having" identifier="true" />
  </link>
  <link loadDate="date" recSource="string">
    <hubRole type="HUB_COURSE" role="held" identifier="true" />
    <hubRole type="HUB_LOCATION" role="at" identifier="true" />
    <refRole type="TERM" role="during" identifier="true" />
    <refRole key="SECTION" role="having" identifier="true" />
  </link>
</schema>
```