Spring 2012

Translating Data Between Xml Schema and 6Nf Conceptual Models

Curtis Maitland Knowles

TRANSLATING DATA BETWEEN XML SCHEMA AND 6NF CONCEPTUAL MODELS

by

CURTIS M. KNOWLES

(Under the Direction of Vladan Jovanovic)

ABSTRACT

Sixth Normal Form (6NF) is a term used in relational database theory by Christopher Date to describe databases which decompose relational variables to irreducible elements. While this form may be unimportant for non-temporal data, it is certainly important for data containing temporal variables of a point-in-time or interval nature. With the advent of Data Warehousing 2.0 (DW 2.0), there is now an increased emphasis on using fully-temporalized databases in the context of data warehousing, in particular with approaches such as the Anchor Model and Data Vault.

In this work, we will explore the concepts of temporal data, 6NF conceptual database models, and their relationship with DW 2.0. Further, we will evaluate the Anchor Model and Data Vault as design methods in which to capture temporal data. Using these methods, we will define a process for translating 6NF-compliant data into a standard eXtensible Markup Language (XML) Schema which can be used to describe and present such data to disparate systems in a structured format suitable for data exchange. Further, we will discuss benefits, advantages, and limitations to using XML representations of 6NF models for the transfer of data into a data warehouse.

INDEX WORDS**:** sixth normal form, 6NF, eXtensible Markup Language, XML, XML schema document, XSD, anchor modeling, Anchor Model, Data Vault, temporal database, semantically temporal data, semantically stable data, semantic data change, valid time, transaction time, Data Warehousing 2.0, DW 2.0, Dimension Fact Model, DFM, eXtensible Stylesheet Language Transformation, XSLT

TRANSLATING DATA BETWEEN XML SCHEMA AND 6NF CONCEPTUAL MODELS

by

CURTIS M. KNOWLES

B.S., North Carolina State University, 1992

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

2012

TRANSLATING DATA BETWEEN XML SCHEMA AND 6NF CONCEPTUAL MODELS

by

CURTIS M. KNOWLES

Major Professor:	Vladan Jovanovic, Ph. D.
Committee:	Lixin Li, Ph. D.
	Wen-Ran Zhang, Ph. D.

Electronic Version Approved:

May 2012

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

With the advent of data warehousing and the emphasis placed on the advantages of using temporal modeling in Data Warehousing 2.0 (DW 2.0) [12], the issue of how to store and model changes to data over time has reemerged as one of practical importance. Temporal data modeling has evolved to the point where there is now a need to describe and present time-sensitive data to disparate data warehouse systems in a structured format suitable for data exchange across the internet and other networked resources. Since eXtensible Markup Language (XML) represents a universal language which has become the preferred means of data exchange over disparate systems [34], an adequate definition of a standard XML Schema document (XSD) template is desired to communicate with DW 2.0-compliant temporal data warehouses.

In this work, we will review a generalization of temporal aspects in database design using Christopher Date's Sixth Normal Form (6NF) and its relationship to DW 2.0 [5]. Furthermore, we will use conceptual and logical techniques such as the Anchor Model (AM) and Data Vault (DV) to visualize 6NF datasets with the intended outcome of presenting a standard template XSD for each based on metamodels derived from a set of formal AM and DV component definitions. We will use Integration Definition for Information Modeling (IDEF1X) as the primary modeling technique. IDEF1X is chosen because it has an ontology closely related to that of the Entity-Relationship (ER) modeling language [1]. The basic ontological categories of IDEF1X are entities, attributes, and relationships. Entities represent concrete objects. Every entity has an associated set of attributes which describe it. In IDEF1X, the set of values an attribute can return is called its domain and the domain is always one of several familiar data types (i.e., string, numeric, boolean, date). There is also an attribute or set of attributes for each entity in IDEF1X called the candidate key which uniquely defines each instance of that entity. Relationships represent associations between two entities. In IDEF1X, one association is a parent entity and the other is a child entity. Relationships may be identifying (the parent candidate key is part of the child candidate key) or non-identifying (the parent candidate key is not part of the child candidate key). Each instance of a child entity is associated with an instance of the parent entity and the number of children a parent is associated with is represented by its cardinality.

IDEF1X supports the following cardinalities identifying how many children a parent may be associated with: (1) zero, one or more, (2) one or more, (3) zero or one, and (4) exactly n.

It should be noted that researchers have previously described an XML Schema definition for the conceptual AM in the paper *"From Anchor to XML"* [33]. We will expand upon this work in several ways. We will provide an interim metamodel to further define the conversion of the conceptual AM to XML Schema. Pointing out differences between the AM and DV models, we will also use the XSD generated by the anchor modeling group as a framework from which to establish a set of formal definitions for basic DV components, generate a metamodel for the DV schema, and produce a standardized 6NF-compliant XSD for the DV. Once this XSD is produced, we will evaluate advantages and limitations in application areas benefiting from the use of the proposed standardized formats.

CHAPTER 2

CURRENT STATE OF THE ART

Temporal Database Design and 6NF Data Modeling

A temporal database is a database containing aspects of time analysis. According to Rönnbäck, the purpose of a temporal database is to "store a body of information under evolution such as to allow historical searches over it" [30]. Specifically, such databases often deal with two concepts of time – valid time and transaction time. Valid time is defined as the time period during which a data fact may be true with respect to the real world (i.e., a time when something actually happened), while transaction time refers to the time period during which a data fact may be stored in the database (i.e., a time when something is measured and/or reported) [10]. In this research, we will include the modeling of both concepts, combining them to represent a bitemporal database which is able to store not only current data but also historical and even future data expected to occur. The concepts of valid time, transaction time, and bitemporal data were all introduced as a part of TSQL2, a language specification developed in 1993 in response to a proposal by Richard Snodgrass for the development of temporal extensions to the Structured Query Language (SQL) [35].

In the field of relational database theory, Sixth Normal Form (6NF) has been used to describe two different concepts of database normalization. Some authors have used the term as a synonym for Domain/Key Normal Form (DKNF) which is a normal form used to describe a database that contains no constraints other than domain constraints (the permissible values of a given attribute) and key constraints (the attributes which uniquely identify a row in a given table) [6]. The other use of the term originates in the work of Christopher Date, and describes temporal databases in which relational operators support treatment of interval data (i.e., sequences of dates or moments in time). We will explain and use Date's definition of 6NF in the rest of this work.

In his book *Temporal Data and the Relational Model*, Date points out several shortcomings when attempting to model fully temporalized data using standard Fifth Normal Form (5NF) principles [5]. To illustrate these shortcomings, first consider a non-temporalized, 5NF-compliant table schema *(see Figure 1)* in which is defined a predicate for a Suppliers relation variable (relvar) S as such: *Supplier S# is under contract, is named SNAME, has status STATUS, and is located in city CITY*:

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London

Figure 1. Non-temporalized Table Schema

To introduce a temporal element into table S, we simply add a temporal attribute SINCE to the table which accounts for valid time *(see Figure 2)*. The resulting table S_SINCE now represents that supplier S# has been under contract since day SINCE:

S_SINCE

S#	SNAME	STATUS	CITY	SINCE
S1	Smith	20	London	D04

Figure 2. Semi-temporalized Table Schema using SINCE

Alternatively, we may want to track an interval during which a relvar is true *(see Figure 3)*. The resulting table S_DURING would represent that supplier S# was under contract during time period DURING. Notice that the ending date for the time interval is null. When a new instance of the S1 supplier is loaded into the database, the ending date of the existing tuple will be updated with the current date, and a new instance of the tuple will be created containing an ending date of null *(see Figure 5)*.

S_DURING

S#	SNAME	STATUS	CITY	DURING
S1	Smith	20	London	[D04:null]

Figure 3. Semi-temporalized Table Schema using DURING

In both of these cases, even though a temporal element has been introduced into the database, the relvars are badly designed for two reasons [4]:

1. One temporal attribute alone is insufficient to model conditions where each of the other individual attributes may vary independently of one another over time, and
2. Data can be lost when single temporal attributes are updated. Historical data thus cannot be tracked.

Due to these shortcomings, Date refers to this type of model as “semi-temporalized” [5].

To overcome the first issue, a SINCE attribute may be added for each descriptive attribute in the table *(see Figure 4)*. The result is a horizontal explosion of table S_SINCE which now can be defined as follows - *Supplier S# was under contract since day S#_SINCE, has been named SNAME since day SNAME_SINCE, has had status STATUS since day STATUS_SINCE, and has been located in CITY since day CITY_SINCE*:

S_SINCE

S#	S#_SINCE	SNAME	SNAME_SINCE		...
S1	D04	Smith	D04		...

...		STATUS	STATUS_SINCE	CITY	CITY_SINCE
...		20	D10	London	D04

Figure 4. Semi-temporalized Table Schema using SINCE

It should be pointed out that S_SINCE is still considered to be "semi-temporalized" since it contains no concept of historical data. For example, if supplier S1's status changes on D12 to 30, the STATUS value of 20 is replaced with 30 and the STATUS_SINCE value of D10 is replaced with D12. The data fact that supplier S1 ever had a status of 20 is now lost.

To overcome the issues of tracking historical data, consider the S_DURING relvar. Given the tuple in *Figure 3*, consider a scenario where supplier S1's status changes to 30 on day 7. The obvious way to account for this change in S_DURING is to update the ending interval date on the existing tuple and insert a new tuple *(see Figure 5)*.

S_DURING

S#	SNAME	STATUS	CITY	DURING
S1	Smith	20	London	[D04:D07]
S1	Smith	30	London	[D07:null]

Figure 5. Semi-temporalized Table Schema using DURING

The problem with this solution is that relvar S_DURING timestamps too much information (supplier is under contract, supplier has a name, supplier has a status, and supplier has a city). To resolve the issue, we must vertically decompose the relvar into a series of four separate relvars, each with its own timestamp attribute *(see Figure 6)*.

S_DURING

S#	DURING
S1	[D04:null]

S_NAME_DURING

S#	SNAME	DURING
S1	Smith	[D04:null]

S_STATUS_DURING

S#	STATUS	DURING
S1	20	[D04:D07]
S1	30	[D07:null]

S_CITY_DURING

S#	CITY	DURING
S1	London	[D04:null]

Figure 6. Fully-temporalized 6NF-compliant Table Schema

Adding transaction time columns CREATED and UPDATED to indicate when a tuple was first created and last updated, respectively, results in a fully-temporalized set of 6NF-compliant relvars capable of tracking historical data *(see Figure 7)*.

S_DURING

S#	DURING	CREATED	UPDATED
S1	[D04:null]	D05	null

S_NAME_DURING

S#	SNAME	DURING	CREATED	UPDATED
S1	Smith	[D04:null]	D05	null

S_STATUS_DURING

S#	STATUS	DURING	CREATED	UPDATED
S1	20	[D04:D07]	D05	D09
S1	30	[D07:null]	D09	null

S_CITY_DURING

S#	CITY	DURING	CREATED	UPDATED
S1	London	[D04:null]	D05	null

Figure 7. Fully-bitemporal 6NF-compliant Table Schema

This vertical decomposition of relvar S_DURING is similar to a classic normalization process and leads us to the introduction by Date of the Sixth Normal Form (6NF) with this formal definition:

> "A relvar (table) *R* is in 6NF if and only if *R* satisfies no nontrivial join dependencies at all, in which case *R* is said to be irreducible." [5]

Before adding historical data, relvar S_DURING (as seen in *Figure 3*) is 5NF-compliant since the only nontrivial join dependencies it satisfies are ones that are implied by its sole candidate

key [S#, DURING]. After adding historical data, however, S_DURING is no longer in 5NF form since the tuple [S#, SNAME, CITY] is redundant over candidate key [S#, DURING]. Therefore, S_DURING can benefit from further decomposition into the set of 6NF projections as shown in *Figure 6*. The result is a set of relvars with irreducible elements, each containing its own interval temporal attribute and each tied to the surrogate key S#.

By using conceptual models, 6NF datasets such as those shown in *Figures 6 and 7* can be easily visualized and integrity constraints can be clearly defined. Furthermore, XML documents and schemas can be created by database analysts and designers to produce a method for exchanging complex 6NF-compliant data among disparate systems in a structured format.

6NF and Anchor Modeling

Anchor Modeling (AM) is a database modeling technique built on the premise that the environment surrounding a data warehouse is in a constant state of change, and furthermore that a large change on the outside of the model should result in only a small change on the inside of the model [32]. When anchor models are translated to physical database designs, content changes can be emulated using standard 6NF principles resulting in relational database tables which are 6NF-compliant [29]. Pieces of data are tied to points in time or intervals of time. Time points are modeled as attributes (i.e., the time a part is shipped), and intervals of time are modeled as a historization of attributes or ties (i.e., times during which a part was in stock). Transaction time elements, such as the time information entered or was updated in the database, are handled by metadata and are not included in the standard anchor modeling constructs.

AM provides a graphical notation for conceptual database modeling similar to Entity-Relationship (ER) modeling, with extensions for temporal data. The model is based on 4 constructs: the anchor, attribute, tie, and knot [32]. A simplified example showing the basic components of the anchor model is presented in *Figure 8*. Anchors model entities and events, attributes model properties of anchors, ties model relationships between anchors, and knots model fixed sets of descriptive entity choices for an attribute that do not change over time (similar to reference lists). Also used in conjunction with ties are identifiers, which are depicted in *Figure 8* as black circles. An identifier is similar to a key in relational databases. It represents a minimal set of roles that uniquely identifies the tie instance. If the circle is filled black, the

connected entity is a part of the identifier (or key) for that tie. If the circle is filled white, the connected entity is not a part of the identifier.

Double lines indicate that an attribute or tie can show a history of changes for the information they model. In *Figure 8*, we see that a history is kept for attribute SUADR (since a supplier may change their address at some point in time), attribute COTRM (since contract terms with a supplier may change over time), and tie SUPA (since the company may change which supplier they order a specific part from at some point in time).

Notice in *Figure 8* that the anchor model can be limited in its adherence to 6NF. Anchors are not temporalized since usually they only contain an identification key for the entity which never changes. Knots are also not temporalized in our example since we are assuming a standard set of knot values that will not change over time or that we are not interested in tracking history for. Attributes and ties can also be selectively temporalized at the discretion of the database designer.

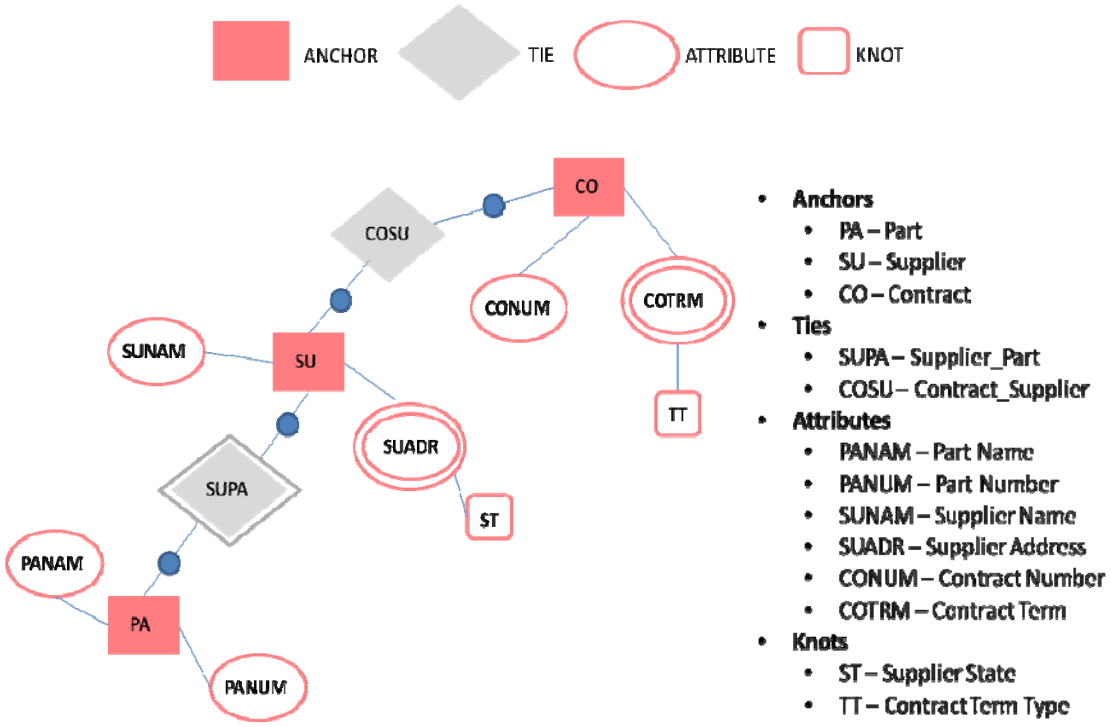


Figure 8. Anchor Modeling – Constructs and a Basic Example

6NF and Data Vault Modeling

Data Vault (DV) modeling is a database modeling technique designed to provide historical storage of data coming in to a data warehouse from multiple systems. Dan Linstedt, the creator of the method, describes a DV database as "a detail-oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business; a hybrid approach encompassing the best of breed between 3NF and star schema; a design that is flexible, scalable, consistent, and adaptable to the needs of the enterprise" [27]. The DV emphasizes a view of historical data for purposes such as auditing, tracing of data, loading speed, and resilience to change [24]. DV modeling attempts to solve the problem of dealing with change in an environment by separating business keys and the association between those keys from the descriptive attributes of the key [18]. The desire to track temporal data for auditing purposes and the subsequent decomposition of the database into separate key, key association, and descriptive attribute tables requires the DV to employ 6NF design principles in its implementation.

In a logical design, the DV model is composed of 4 basic components: the hub, link, satellite, and reference table [19]. A translation of the anchor model of *Figure 8* into its DV counterpart is presented in *Figure 9* with hubs denoted in light blue and links denoted in light beige.

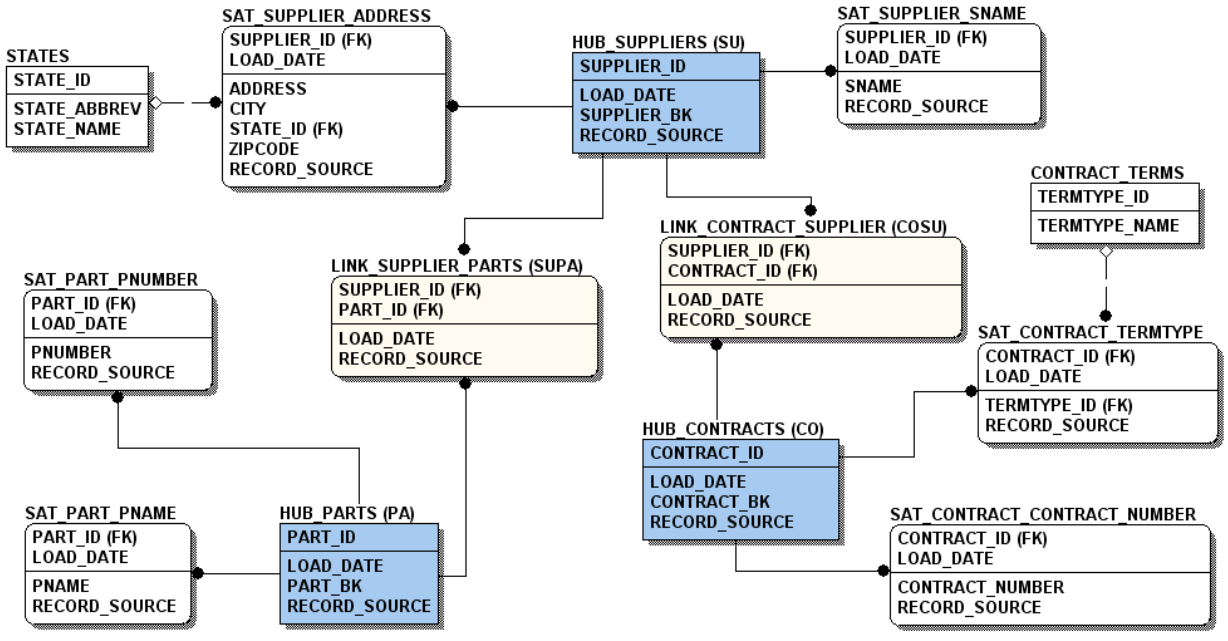


Figure 9. Anchor Data Modeled as a Data Vault

A correlation between the basic components of the DV and AM can be seen where hubs equate to anchors, links equate to ties, satellites equate to attributes, and reference tables equate to knots. Hubs represent a list of unique stable business keys unlikely to change over time (i.e., Customers or Products). Links represent transactions between hubs (i.e., relating Customer to Product through a Purchase transaction). Satellites represent the descriptive attributes of a hub (i.e., a Customer has a Customer Name, Address, Phone, etc.). Reference Tables are fixed lists of values which may be used frequently, and they exist to simply prevent redundancy of data in satellites and links.

Hubs contain a surrogate key, often a sequence identifier serving as the entity's primary key and thus uniquely identifying the entity in the present database. Hubs will also contain a "business key" (denoted in *Figure 9* with a "_BK" suffix). The business key is an identifier of the hub from the source system. Links and satellites contain a "parent key" which is simply a foreign key linked to its parent hub's surrogate key. Each component will also contain an identifier indicating which system a business key originated in ("RECORD_SOURCE" in *Figure 9*).

To accommodate temporal aspects of the data, hubs, satellites, and links contain an audit key or ID ("LOAD_DATE" in *Figure 9*) which represents valid time metadata for the entity. This field will sometimes exist as an Audit ID which points to a separate table containing metadata such as date of initial insertion into the database, load time, or load duration. In satellites, LOAD_DATE is a part of the composite primary key along with the surrogate key indicating that a history of data or reference values is kept for each satellite.

In *Figure 9*, CITY, STATE_ID, and ZIPCODE are assumed to be a part of a complete address (i.e., 1234 Main Street will always be in the same city, state, and zip code) and thus all four fields together are placed in the SAT_SUPPLIER_ADDRESS table as one irreducible entity. If we wish to track zip code changes for a certain locale, a further reduction can be performed to historize the zip code attribute into its own satellite as well. Notice also from this example that it is possible to produce a DV model which is not inherently irreducible in its elements. For instance, if the supplier's name was added to the address information in the SAT_SUPPLIER_ADDRESS table, the model is no longer strictly 6NF-compliant. In this case, the LOAD_DATE would timestamp too much information and the result will be redundant data in the SNAME field. By using 6NF principles of exploding attribute tables into separate satellites

of irreducible elements and assigning each satellite its own temporal component, 6NF-compliance may be achieved.

Implications for Data Warehousing 2.0

Data Warehousing 2.0 (DW 2.0) is a second-generation attempt to define a standard for data warehouse architecture. Some problems of traditional DW architecture include (a) complex updates, (b) difficulty in integrating and reconciling source data, (c) growing delays caused by upfront reconciliation of data, (d) poor performance in loading data, (e) unstable reporting due to integration of data transformations close to volatile sources, (f) a lack of an integrated system of record, (g) lack of traceability from sources to archives, and (h) complex reconstruction of schemas that do not support evolving content and semantics [15]. DW 2.0 seeks to address some of these problems, and the DV is promising as a model for this cause. The DV 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. Because of these factors, the DV model is recognized as the optimal choice for DW 2.0 architecture [15].

A key feature introduced in DW 2.0 is the ability to support changes of data over time. The authors of DW 2.0 address this issue by saying, "The answer is that semantically static and semantically temporal data should be physically separate in all database designs" [12]. 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 an entity changes (i.e., an amount increases from $100 to $200). Semantically temporal data is likely to undergo semantic data change whereas semantically stable data is unlikely to undergo such change. An example of semantically temporal data is a customer profile, where you may want to track new types of information about a customer over time and therefore add new columns of data to the customer entity. For instance, if a company decides to send out a monthly email newsletter, it may add an email address to its customer profiles. An example of semantically stable data is the details of a customer sale. Once a date of sale, amount of sale, item sold, etc. has been determined, this data is unlikely to change over time. If the data is ever updated, the change is most likely the result of correcting an error in the original recording of data and not due to an actual change in the transaction itself.

The designers of DW 2.0 point out that, in an environment where temporal and stable data are grouped together, a simple change in business requirements usually causes the technology infrastructure to "go haywire" [12]. 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 [14]. Date's temporalization approach comes into play here because historical snapshots must be created in such a way as to decompose data into a series of irreducible 6NF-compliant tables. By using this approach, semantically stable data is not affected at all and previously existing semantically temporal data also remains unchanged. 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 [12]. The goal of the AM technique is to "achieve a highly decomposed implementation that can efficiently handle growth of the data warehouse without having to redo any previous work", mimicking the ideas of isolated semantic temporal data put forth in DW 2.0 architecture [32]. The DV technique also follows this same pattern of effective extensibility of infrastructure. Therefore for purposes of DW 2.0 staging, AM can be considered as the preferred conceptual modeling technique while DV can be considered as the preferred logical and physical modeling technique [15].

CHAPTER 3

GENERAL XML SCHEMA DESIGN APPROACH

In this chapter, we describe the approach used to build the XML Schema documents (XSDs) produced in this work. XSDs express shared vocabularies and provide a means for defining the structure, content, and semantics of XML documents. The main components of our XSD are enumerated below.

Schema Namespaces

For all XSDs, a schema default namespace must be declared which will provide a unique identifier for all element and attribute declarations. 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 of the schema will use to access their declared types. We will be producing two XSDs – one for an Anchor Model (AM) schema and one for a Data Vault (DV) schema. For the AM schema, we will use the anchor modeling group's target namespace (http://anchormodeling.com/schema). 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. We will also be making assertions in the form of XPath expressions, so we will set a default namespace for XPath expressions to be the target namespace as well. The schema namespace declarations for our XSDs can be seen in *Appendix B, line 2* and *Appendix E, line 2*.

Design Pattern

The four common patterns for XML Schemas are the Russian Doll, Salami Slice, Venetian Blind, and Garden of Eden [16]. They each differ according to the number of their global elements and types. 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. The advantages of using the Venetian Blind pattern are that all types and element groups

are reusable throughout the schema and there is only one valid XML document since only one global element exists. There are no major disadvantages to using the Venetian Blind pattern [26]. Our single global element will be named "schema" *(see Appendix B, line 65 and Appendix E, line 59)* since we are endeavoring to define AM and DV modeling schemas through our XML, and not actual data. By using only one single root element and all reusable types, our pattern will be suitable for use by both instance developers and schema developers [16].

Elements and Attributes

Elements will represent data produced or consumed by an application and attributes will be used to 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). We will defer to the default minOccurs and maxOccurs values when possible. 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 to be used in case none is specified.

Types

When possible, simple data types defined in the XML Schema namespace will be used rather than creating user-defined data types. However, especially for the AM schema, some user-defined data types will be required. Restricting a simple type reduces possible values of the type while extending a simple type in essence creates a complex type with attribute content. When restricting user-defined types, we will restrict them to standard XML types such as string, numeric, boolean, or data. When extending simple types, reusable complex types will be created describing each of the main AM and DV model components and its properties.

Assertions

Several assertions will be introduced in the XSD to capture conditions an AM or DV model must satisfy. Assertions are XPath expressions which should hold true for any given implementation of a XML Schema definition.

CHAPTER 4

ANCHOR MODEL TO XML

In *"From Anchor Model to XML"* [33], the anchor modeling group generates a standard XML Schema Definition (XSD) and XML representation of a conceptual anchor model (AM). In this chapter, we offer a metamodel of the AM schema in IDEF1X notation and a breakdown of XSD type components which will assist in further defining this transformation.

Using formal definitions of AM components listed in Appendix A, we can visualize an AM schema metamodel using IDEF1X *(see Figure 10)* [32]. The use of sub-types specifies four types of attributes and ties: (a) static, (b) historized, (c) knotted static, and (d) knotted historized [32]. Static attributes and ties do not change over time while historized attributes and ties do. Knotted static attributes and ties are static entities associated with at least one knot while knotted historized attributes and ties are historized entities associated with at least one knot.

Cardinality is used to express relationships within the conceptual AM. A knotted attribute contains one and only one knot while a knot may be in zero or more knotted attributes. An attribute will only be present in one anchor while an anchor will contain one or more attributes. An anchor may be in zero or more ties while a tie will contain two or more anchor roles. A knot may be in zero or more knotted ties while a knotted tie will contain one or more knot roles.

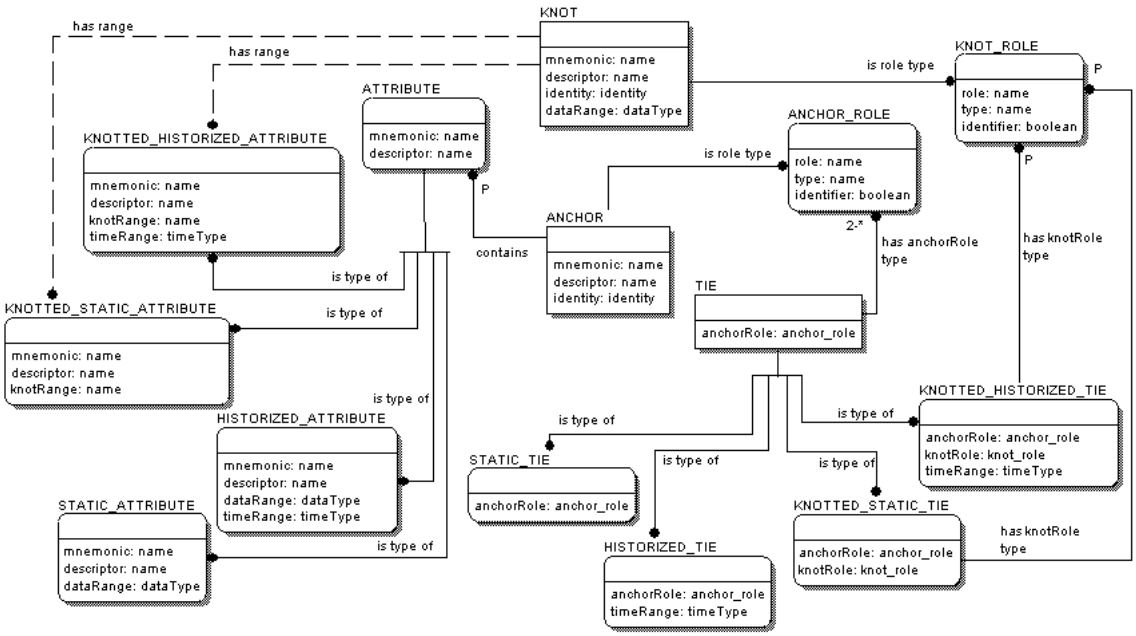


Figure 10. Anchor Schema Metamodel

Simple Types

For XSD translation, three simple data types are created to accommodate the AM schema – "identity", "dataType", and "timeType". All three types are restricted to string values. The identity type captures the data type of the identifying key value for a component, dataType captures the data type of the component itself, and timeType captures the time type (usually "date") for historized components. Additionally, the simple type "name" is introduced and restricted to strings to unify the presentation of names present in most of the elements of the anchor schema [33]. *Figure 11* shows the simple types to be used in constructing the anchor model XSD. The XSD simple type definitions can be seen in *Appendix B, lines 3-14*.

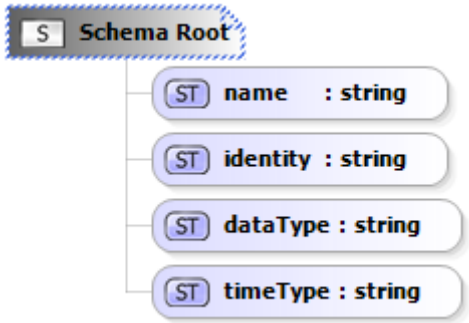


Figure 11. Anchor Model Simple Types

Complex Types

Complex type definitions are also created for the AM schema and are constructed according to each of the main anchor model components – knots, attributes, anchors, and ties.

A. Knots

The knot complex type contains a mnemonic, a descriptor, an identity, and a dataRange *(see Figure 12)* [33]. The mnemonic is a short string used to uniquely identify an element. It also serves as a reference to which attributes and ties can refer. Descriptors are longer strings which serve as name descriptions of the object. The identity attribute is of simple type identity and restricts the value of the knot's identification key. The dataRange attribute is of simple type dataType and restricts the value of the knot's actual data. The XSD knot definition can be seen in *Appendix B, lines 20-25*.

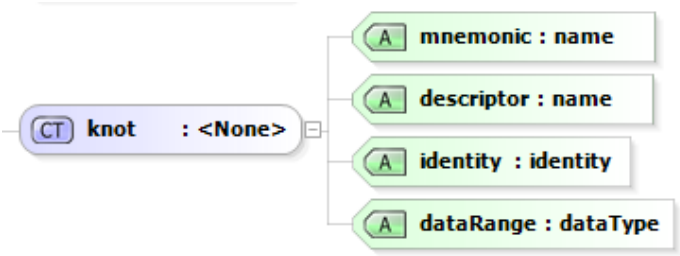


Figure 12. Anchor Model Complex Type – Knot

B. Attributes

It is important not to confuse the AM attribute with the standard XML Schema namespace attribute component. In the final XSD, the AM attribute will always be prefaced by the "xs:complexType:" designation whereas the XML Schema namespace attribute component will always be prefaced by the "xs:" namespace designation. To avoid confusion, we will use the terms "AM attribute" and "namespace attribute" here.

The AM attribute contains a mnemonic, a descriptor, and optional knotRange/dataRange/ timeRange namespace attributes *(see Figure 13)* [33]. All three namespace attributes deal with data type restriction of the AM attribute's data value. The knotRange restricts a value to its corresponding knot, dataRange restricts a value to a certain data type, and timeRange restricts a temporal value to a certain data type (usually "date").

These namespace attributes are present depending upon the type of AM attribute we are dealing with. Static AM attributes only have a dataRange. Historized AM attributes have a dataRange and timeRange. Knotted static AM attributes have a knotRange. Knotted historized AM attributes have knotRange and timeRange. The XSD AM attribute definition can be seen in *Appendix B, lines 26-33*.

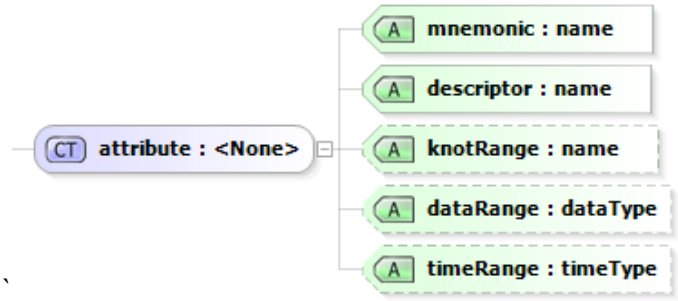


Figure 13. Anchor Model Complex Type – Attribute

C. Anchors

Anchors contain a mnemonic, a descriptor, and an identity *(see Figure 14)*. In addition, they also contain a sequence of one-to-many AM attributes [33]. Note that each anchor must contain at least one AM attribute, and that AM attributes are maintained as elements (as opposed to namespace attributes) in the hierarchical structure of the anchor. The XSD anchor definition can be seen in *Appendix B, lines 34-41*.

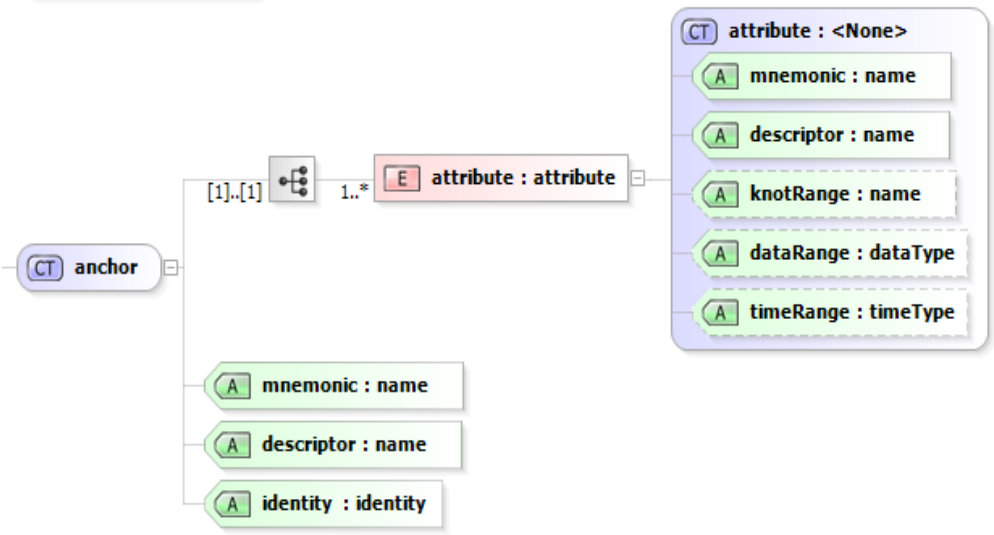


Figure 14. Anchor Model Complex Type - Anchor

D. Ties

Ties form relationships between anchors and knots. According to formal definition, ties are composed of at least two anchor elements and any number of knot elements [32]. Since multiple anchors or knots may appear in the same tie, the concepts of anchor role and knot role are introduced in order to qualify their use inside a tie *(see Figure 10)* [33]. To handle anchor and knot roles, the complex type role is produced *(see Figure 15)* [33]. The XSD role definition can be seen in *Appendix B, lines 15-19*.

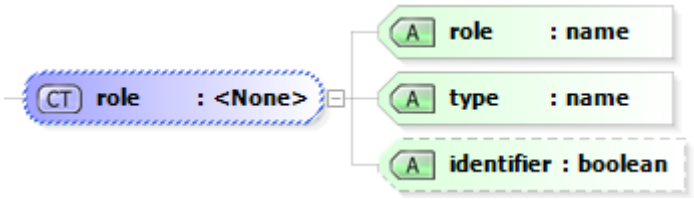


Figure 15. Anchor Model Complex Type – Role

The role complex type consists of a role, a type, and an identifier flag [33]. The type will usually be the same as the mnemonic 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 tie. The identifier flag will be true or false depending upon whether the object is part of the unique identifier of the tie or not.

The tie complex type consists of two or more anchorRole elements, zero or more knotRole elements, and an optional timeRange attribute *(see Figure 16)* [33]. Similar to attributes, ties can be static, historized, knotted static, or knotted historized. Static ties have only anchorRole elements, historized ties have anchorRole elements and a timeRange, knotted static ties have anchorRole and knotRole elements, and knotted historized ties have anchorRoles, knotRoles, and a timeRange. The XSD tie definition can be seen in *Appendix B, lines 42-49*.

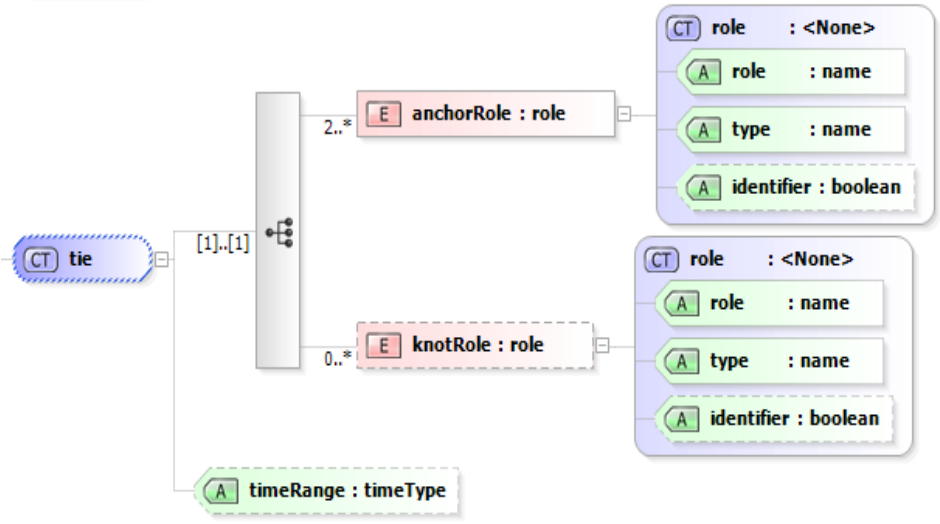


Figure 16. Anchor Model Complex Type - Tie

XSD Transformation

Based on the simple and complex types generated from our metamodel in *Figure 10*, a standard XSD is produced for an AM conceptual schema *(see Appendix B)*. Notice the specification of certain elements and attributes throughout the XSD as "required" or "optional". Also the identifier attribute of the role complex type is given a default value of "false" *(line 18)*. The "minOccurs" is set to 1 for the attribute sequence inside the anchor complex type *(line 36)*, specifying each anchor has at least one attribute according to formal definition. The "minOccurs"

is similarly set to 2 for anchorRoles in a tie and 0 for knotRoles in a tie *(lines 44, 45)*, specifying each tie must be associated with at least two anchors and may or may not have associated knots.

Several assertions have been introduced in the XSD *(lines 32, 48, 56-63)* to capture conditions an anchor schema must satisfy. Assertions are XPath expressions which should hold true for any given implementation of an AM schema. Inside the attribute complex type, an assertion is made that *"count(@knotRange) + count(@dataRange) = 1" (line 32)* which states "each attribute will have either one knotRange or one dataRange, but never both". Inside the tie complex type, an assertion is made that *"anchorRole[@identifier = 'true']" (line 48)* which states "any anchorRole associated with a tie will be at least a partial identifier of the tie". Two types of assertions are made in the schema complex type – (1) *"every $k in anchor/attribute/@knotRange satisfies knot[@mnemonic = $k]" (line 56)* which states "for every attribute found in an anchor that has a knotRange there must exist a knot in the schema whose mnemonic is equal to that knotRange", and (2) *"not(some $t1 in tie, $t2 in $t1/preceding-sibling::tie satisfies deep-equal($t1,$t2))" (line 59)* which states "for no tie an exact copy of that tie may exist".

Using the resulting XSD, we can now generate an XML document for any AM Universe of Discourse. We model one such example for a college registration system in Appendix C and produce the accompanying XML transformation in Appendix D. In this example, business entities are represented by anchors ST_Student, CO_Course, LO_Location, DE_Department, and PR_Professor. Each anchor has one or more attributes (i.e., ST_Student has attributes STNAM_Student_Name, STADR_Student_Address, STGEN_Student_Gender, and STMAJ_ Student_Major). Some attributes are associated with knots (i.e., STADR_Student_Address has knot STA_State indicating only 50 states available to be used in an address and STGEN_Gender has knot GEN_Gender indicating only 2 genders a student can be).

Also in our sample UoD, several ties exist between anchors. One example is DE_by_CO_offered (which can be read as "course CO is offered by department DE"). Several ties also have knots associated with them. For example, tie CO_in_ST_enrolled_TER_during_SEC_having is associated with the knots TER_Term and SEC_Section and can be read as "student ST is enrolled in course CO having section SEC during term TER". All attribute and tie types are modeled in this example, as shown in *Table 1*.

Type	Attributes	Ties
Static	COABR, LOROM, LOBLD, STNAM, PRNAM	DE_by_CO_offered
Historized	DENAM, COTTL	DE_in_PR_works
Knotted Static	STGEN	CO_in_ST_enrolled_TER_during_SEC_having LO_at_CO_held_TER_during_SEC_having PR_by_CO_taught_TER_during_SEC_having
Knotted Historized	STMAJ, STADR	PR_by_ST_advised_DEG_pursuing

Table 1. Attribute and Tie Type Examples

In the next chapter, we will develop a standard XSD format for a Data Vault (DV) logical schema. While the DV and AM are similar in many regards, there are important differences stemming mainly from the fact that an AM is not limited to strict 6NF compliance (and therefore some non-temporal attributes and ties may exist) whereas the DV does have this requirement. We will elaborate on these differences next.

CHAPTER 5

DATA VAULT TO XML

In this chapter, we will define a standard XSD format for the Data Vault (DV) model. This transformation differs to the previous chapter's Anchor Model (AM) to XML translation because certain data conditions exist in the DV which do not exist in the AM. First of all, the DV is required to conform to strict 6NF standards whereas AM is not. AM is seen by its creators as a "general approach with wide applicability" which marks a difference from the DV approach meant to specifically address temporal data loading for data warehouses [32]. Secondly, some conditions are possible in the DV which are not covered by the AM formal definitions. These variations lead to several structural differences in the two models:

1. No static satellites or links are allowed in the DV. The introduction of static attributes and ties into an anchor model implies partial and selective temporalization in AM. However, DV requires the temporalization of all components including every satellite and every link through use of a load date. In fact, for any satellite the load date must be part of its composite primary key.
2. Hubs must be temporalized. In AM, the anchor is not temporalized. However, every insertion into a DV hub must be tracked with a load date.
3. Links can be associated in other links. In AM, this is not possible according to the formal definition of a tie *(see Appendix A, rules 12-15)*. However, we must account for this data condition in the DV. For example, if hub A and hub B are joined by link L1 and hub C and hub D are joined by link L2, another relationship (link L3) can exist between link L1 and link L2. The state of link L3 on day 25 may be defined as "link L1 as of day 10 and link L2 as of day 20". The implication of this design variation is that links must now accommodate a new role for links in addition to roles for hubs and reference tables.
4. Links can also have satellites. This is not possible in the AM, where ties can only be associated with anchors and knots.

The only concept in the DV model not temporalized is the reference table. We assume in this discussion that reference tables are standard lists of data values which seldom change and whose data values can be reconciled downstream from the initial data staging process.

Based on these standards for the DV model, we can generate a set of formal definitions for basic DV components *(see Appendix A)*. Using these definitions, we can visualize a metamodel of a standard DV schema using IDEF1X notation *(see Figure 17)*.

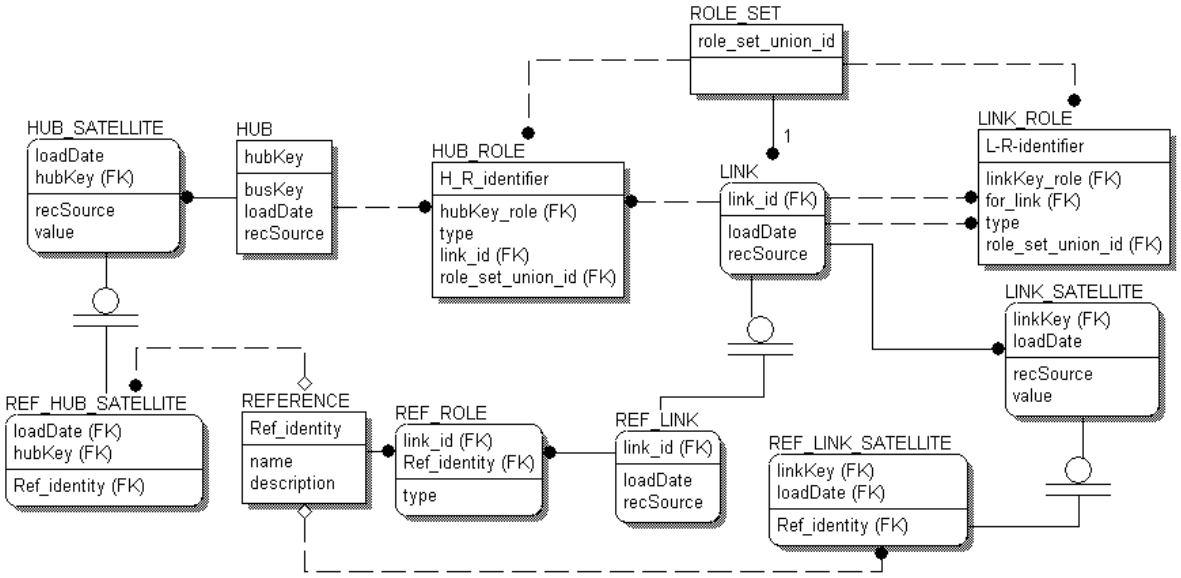


Figure 17. Data Vault Metamodel

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 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 satellites (similar to historized attributes in AM) and the REF_LINK entity represents referenced links (similar to historized ties in AM).

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).

Cardinality is also 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. 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. A link may contain one or more satellites.

Simple Types

For the DV XSD, basic XML Schema simple types are sufficient. The attributes of the basic DV metamodel components will be represented as shown in *Table 2*.

Attribute	**Data Type**
name	xs:string
hubKey	xs:integer
loadDate	xs:date
recSource	xs:string
busKey	xs:string
identity	xs:string
description	xs:string
value	xs:string
refValue	xs:string
role	xs:string
type	xs:string
identifier	xs:boolean

Table 2. XML Schema Types for Standard Data Vault Attributes

Complex Types

Based on the metamodel, complex type definitions can be created for a DV schema and are constructed for each of the main DV components – reference tables, satellites, hubs, and links.

A. Reference Tables

The reference table complex type ("reftable") contains a name, an identity, and a description *(see Figure 18)*. The name is a string used to identify the name of the reference table (i.e., table STATES in *Figure 9*). The identity is a short string used to identify a unique key value within each reference table (i.e., STATES.STATE_ABBREV in *Figure 9*). The description is a longer string used to describe the unique key value within each reference table (i.e., STATES.STATE_NAME in *Figure 9*). The XSD reference table definition can be seen in *Appendix E, lines 8-12*.

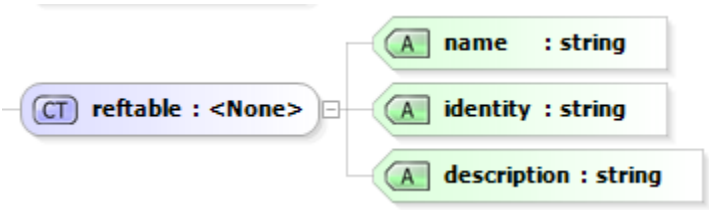


Figure 18. Data Vault Complex Type – Reference Table

B. Satellites

The complex type satellite contains a name, a hubKey, a loadDate, a recSource, and optional value and refValue attributes *(see Figure 19)*. The hubKey is a unique integer key used to identify the hub that the satellite is associated with (i.e., SAT_SUPPLIER_SNAME.SUPPLIER_ID in *Figure 9*). The loadDate is the date of initial load of the satellite into the DV (i.e., LOAD_DATE in *Figure 9*). The recSource is a string indicating the source system from which the data came into the DV (i.e., RECORD_SOURCE in *Figure 9*). The value and refValue attributes are strings representing an actual data value for the satellite (i.e., SAT_SUPPLIER_SNAME. SNAME (non-ref) and SAT_SUPPLIER_ADDRESS.STATE_ID (ref) in *Figure 9*).

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. The XSD satellite definition can be seen in *Appendix E, lines 13-21*.

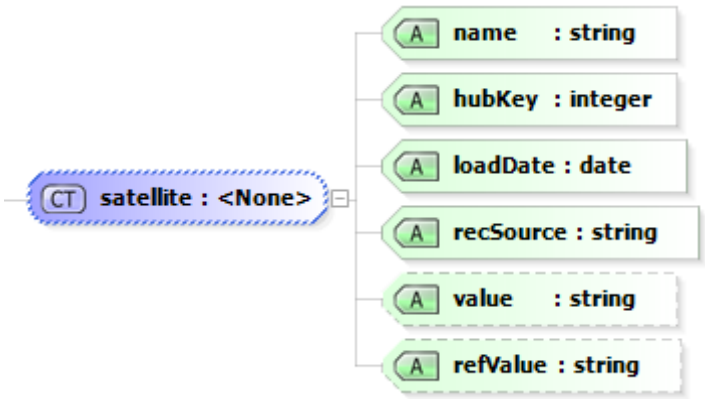


Figure 19. Data Vault Complex Type - Satellite

C. Hubs

Hubs contain a name, hubKey, loadDate, recSource, and a busKey *(see Figure 20)*. They 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., HUB_SUPPLIERS. SUPPLIER_BK in *Figure 9*). The XSD hub definition can be seen in *Appendix E, lines 22-31*.

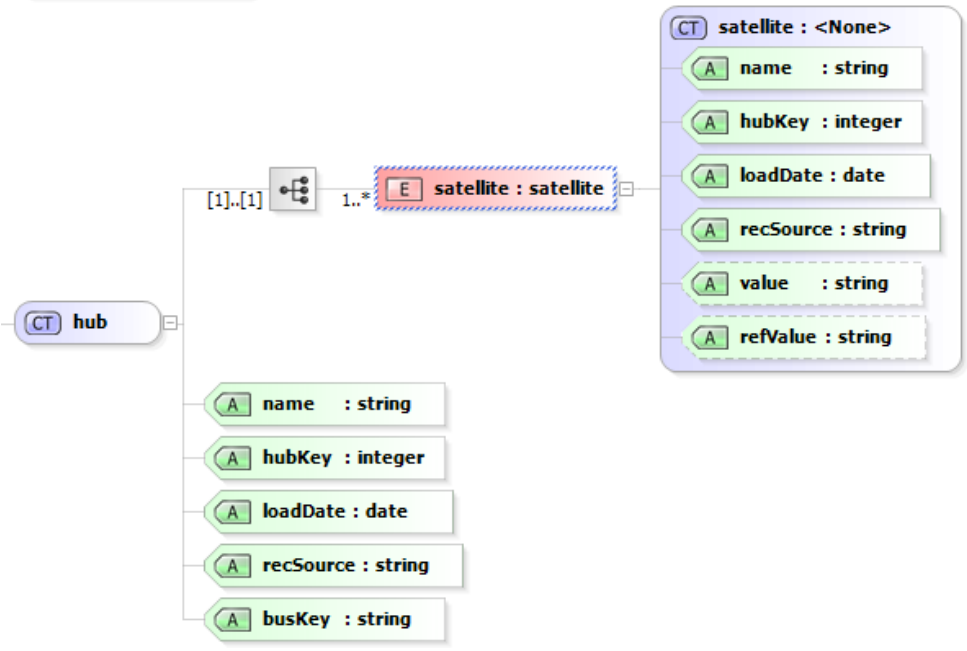


Figure 20. Data Vault Complex Type – Hub

D. 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 21)*. The XSD role definition can be seen in *Appendix E, lines 3-7*.

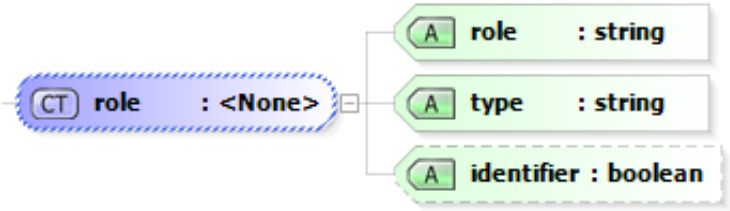


Figure 21. Data Vault Complex Type – Role

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 link complex type consists of two or more hubRole elements, zero or more refRole elements, zero or more linkRole elements, a loadDate, and a recSource *(see Figure 22)*. Similar to satellites, links can be referenced or non-referenced. Referenced links will have one or more linkRole elements. Non-referenced links will not have any linkRole elements. The link also contains a sequence of satellites. The XSD link definition can be seen in *Appendix E, lines 32-42*.

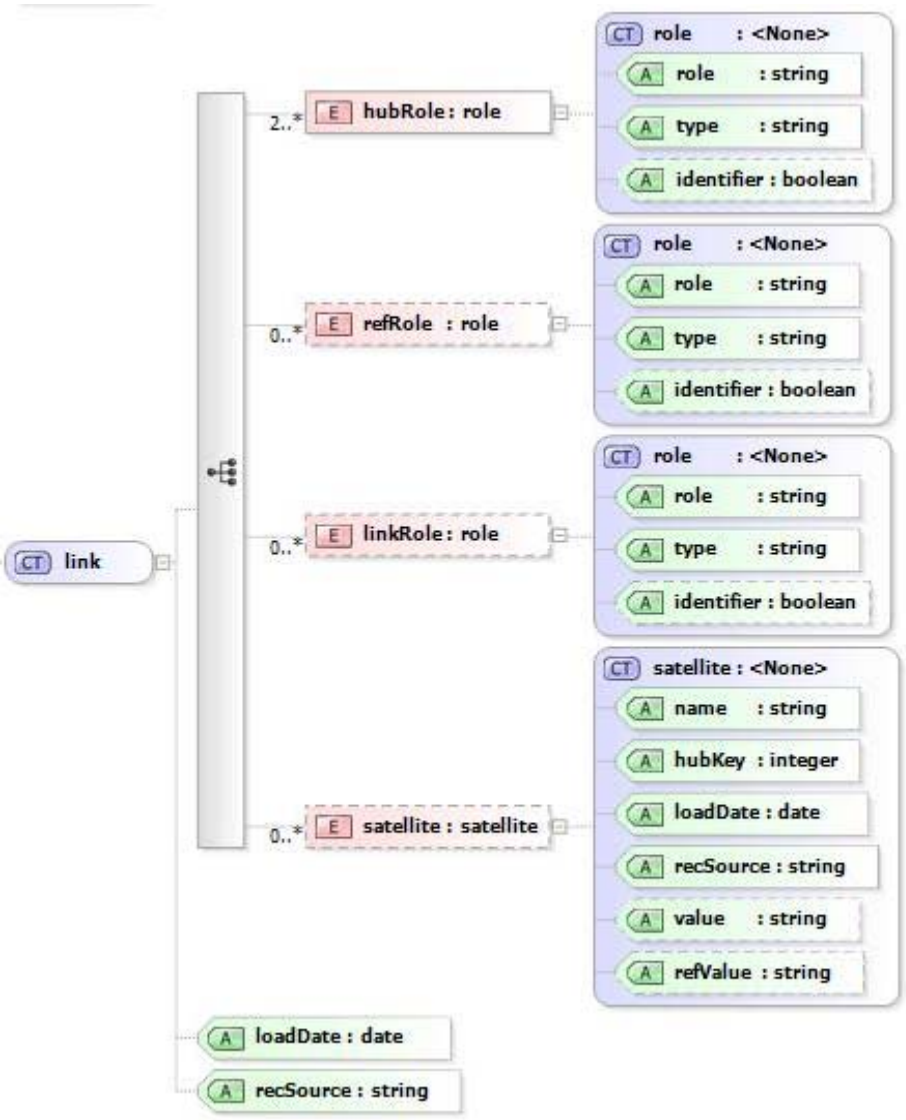


Figure 22. Data Vault Complex Type – Link

XSD Transformation

Based on the simple and complex types generated from our metamodel in *Figure 17*, a standard XSD is produced for a DV schema *(see Appendix E)*. Notice the specification of certain elements and attributes as "required" or "optional". Also the identifier attribute of the role complex type is given a default value of "false" *(line 6)*. The "minOccurs" is set to 1 for the satellite sequence inside the hub complex type *(line 24)*, 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

(lines 34-36), specifying that each link must be associated with at least two hubs and may or may not have associated reference table values and/or associated links.

Several XPath assertions have again been introduced in this XSD *(lines 20, 41, 49-57)* to capture conditions a DV must satisfy. Inside the satellite complex type, an assertion is made that *"count(@refValue) + count(@value) = 1" (line 20)* which states "each satellite will have either one data value or one reference table value, but never both". Inside the link complex type, an assertion is made that *"hubRole[@identifier = 'true']" (line 41)* which states "any hubRole associated with a link will be at least a partial identifier of the link". Two types of assertions are made in the schema complex type – (1) *"every $r in hub/satellite/@refValue satisfies reftable[@identity = $r]" (line 49)* which states "for every satellite found in a hub that has a reference table value there must exist a reference table in the schema that includes an identity equal to that reference table value", and (2) *"not(some $t1 in link, $t2 in $t1/preceding-sibling::link satisfies deep-equal($t1,$t2))" (line 53)* which states "for no link an exact copy of that link may exist".

Using the template XSD for the DV metamodel, we can now craft XML schemas for any DV model logically derived from some source data Universe of Discourse (UoD). As an example, we produce a XML schema of our college registration system UoD (described in Chapter 4) in *Appendix F* and the accompanying logical DV model in *Appendix G*.

CHAPTER 6

BENEFITS, ADVANTAGES, AND LIMITATIONS

In this chapter, we discuss some benefits and advantages of having a standard XML schema representation for 6NF-compliant data models, such as those created in Chapters 4 and 5. We will also present some limitations these representations face.

Data Interchange

By providing an XML representation, we now have an interchange format for Anchor Model (AM) and Data Vault (DV) schemas. While AMs also are seen as a general approach with wide applicability beyond data warehousing, DVs exist for the express purpose of accumulating data from disparate systems into one centralized data warehouse [32]. Therefore, having a standardized XML schema for the DV is especially significant to its purpose. By nature, XML documents consist of nested elements that are distributed over multiple remote or disparate systems to communicate with one another.

While the DV XML schema representation does provide a framework for data interchange, 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

   By nature, DVs do not allow for physical deletion of data [24]. If you wish for the DV 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.

3. Data Errors

   The theme of DV technology is “all the data, all of the time” [3]. A DV load process should never fail due to data errors in the source system. Just because a data format or value is “bad”, this does not mean it should not be loaded into the data warehouse. The

DV records everything as a fact, even the recording of errors. The XML representation 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 certain elements to correct data type mismatches and zero keys can be employed to correct null key values [17]. An error-logging mechanism can also be employed in the loading procedure when one of these instances are encountered.

4. Data Type Variation

   With data coming from disparate systems, one particular satellite in the DV may be represented by two different data types in two different source systems. To handle both data types, treat the XSD as if it were a raw data staging table by using string or varchar as a default data type for all incoming data value elements [36]. Once the XML is deserialized, perform type conversion operations on the data to conform to your DV specifications.

Scalability and Ease of Use

Using XML to represent data for loading into a DV is desired because of its scalability. We are often dealing with massive volumes of data which can scale into the terabyte/pedabyte range and beyond. This data must be loaded in a highly efficient manner with a low arrival latency [17]. XML qualifies as an efficient data transport method to meet this need. Large volumes of data can be expressed as XML text content requiring a relatively low amount of disk space. In addition, with just a few lines of code, an XML file can be read and parsed. The advent of XML serialization object libraries in common language APIs such as Java and C# also facilitate straightforward handling of XML schemas with a set of constructors, methods, and events designed to control how objects are encoded into and decoded from XML.

Automatic Code Generation

The XML representations of the AM and DV schemas can be transformed into SQL code using Extensible Stylesheet Language Transformation (XSLT) technology [11]. As an example, let's consider generating SQL code for a SQL Server database using the HUB_PROFESSOR hub *(line 19-21)* from the XML document in *Appendix F*.

Beginning with an abstract table model from the XML document in Step 1, we build a SQL model in Step 2 followed by the resulting SQL output in Step 3:

Step 1 – Abstract Table Model from XML

```
<hub name="HUB_PROFESSOR" hubKey="integer" loadDate="date"
      recSource="string" busKey="string">
   <satellite name="SAT_PROFESSOR_NAME" hubKey="integer" loadDate="date"
               recSource="string" value="string" />
</hub>
```

Step 2 – SQL Model in XML

```
<create name=”HUB_PROFESSOR” primary-key=”hubKey” >
   <field name=”hubKey” type=”INTEGER NOT NULL” />
   <field name=”loadDate” type=”DATETIME NOT NULL” />
   <field name=”recSource” type=”VARCHAR(20) NOT NULL” />
   <field name=”busKey” tyPe=”VARCHAR(20) NOT NULL” />
</create>
<create name=”SAT_PROFESSOR_NAME” primary-key=”hubKey” >
   <field name=”hubKey” type=”INTEGER NOT NULL” />
   <field name=”loadDate” type=”DATETIME NOT NULL” />
   <field name=”recSource” type=”VARCHAR(20) NOT NULL” />
   <field name=”professorName” type=”VARCHAR(20) NOT NULL” />
</create>
```

Step 3 – SQL Output

```
DROP TABLE IF EXISTS HUB_PROFESSOR;
CREATE HUB_PROFESSOR (
   hubKey INTEGER NOT NULL.
   loadDate DATETIME NOT NULL,
   recSource VARCHAR(20) NOT NULL,
   busKey VARCHAR(20) NOT NULL,
   PRIMARY KEY (hubKey));
DROP TABLE IF EXISTS SAT_PROFESSOR_NAME;
CREATE SAT_PROFESSOR_NAME (
   hubKey INTEGER NOT NULL,
   loadDate DATETIME NOT NULL,
   recSource VARCHAR(20) NOT NULL,
   professorName VARCHAR(20) NOT NULL,
   PRIMARY KEY (hubKey));
```

The XSLT templates used for each step are as follows:

SQL Model Generator

This xml generates a SQL model (Step 2) from the XML abstract model (Step 1). Notice the use of template model “gen:model-type-to=sql” which produces appropriate SQL Server type names for each field, and the use of template model “gen:sat-value-name” which changes attribute name “value” in Step 1 to “professorName” in Step 2. Further statements can be added to the <xsl:choose> construct in the gen:sat-value-name template for each satellite.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:gen="http://www.codegeneration.net/" version="2.0">
   <xsl:template name="gen-sql-model">
      <xsl:param name="model"/>
      <sql>
         <xsl:for-each select="$model/tables/hub">
            <create name="{lower-case(@name)}"
                    primary-key="hubKey">
                  <field name="hubKey"
                    type="{gen:model-type-to-sql(@hubKey)}"/>
                  <field name="loadDate"
                    type="{gen:model-type-to-sql(@loadDate)}"/>
                  <field name="recSource"
                    type="{gen:model-type-to-sql(@recSource)}"/>
                  <field name="busKey"
                    type="{gen:model-type-to-sql(@busKey)}"/>
            </create>
            <xsl:for-each select=”satellite”>
               <create name="{lower-case(@name)}"
                    primary-key="hubKey">
                  <field name="hubKey"
                    type="{gen:model-type-to-sql(@hubKey)}"/>
                  <field name="loadDate"
                    type="{gen:model-type-to-sql(@loadDate)}"/>
                  <field name="recSource"
                    type="{gen:model-type-to-sql(@recSource)}"/>
                  <field name="(gen:sat-value-name(@name)}"
```

```
                    type="{gen:model-type-to-sql(@value)}"/>
            </create>
         </xsl:for-each>
      </sql>
   </xsl:template>
</xsl:stylesheet>

<!--Satellite Value Transform Model-->
<xsl:function name="gen:sat-value-name">
   <xsl:param name="name"/>
      <xsl:choose>
         <xsl:when test="$name eq 'SAT_PROFESSOR_NAME'">
            professorName</xsl:when>
      </xsl:choose>
</xsl:function>

<!--SQL Type Transform Model-->
<xsl:function name="gen:model-type-to-sql">
   <xsl:param name="type"/>
      <xsl:choose>
         <xsl:when test="$type eq 'string'">VARCHAR(20) NOT NULL</xsl:when>
         <xsl:when test="$type eq 'date'">DATETIME NOT NULL</xsl:when>
         <xsl:when test="$type eq 'integer'">INTEGER NOT NULL</xsl:when>
   </xsl:choose>
</xsl:function>
```

SQL Generation Template

This xml generates the SQL output (Step 3) from the SQL model (Step 2). This conversion is straightforward since we have previously performed the field name (“value”) conversion in our SQL model.

```
<xsl:template match="create" mode="sql">
   DROP TABLE IF EXISTS <xsl:value-of select="@name" />;
   CREATE <xsl:value-of select="@name" /> (
   <xsl:apply-templates mode="sql" select="field" />
      PRIMARY KEY ( <xsl:value-of select="@primary-key" /> ));
</xsl:template>
```

```
<!-- Template for SQL field tags -->
<xsl:template match="field" mode="sql">
   <xsl:value-of select="concat(@name,' ',@type,',')" />
   <xsl:text>
   </xsl:text>
</xsl:template
```

These templates demonstrate creation of tables within a DV based on the XSD defined in Chapter 5. Since the design of DV components is static (i.e., every hub has a hubKey, loadDate, recSource, and busKey; every satellite has a hubKey, loadDate, recSource, and value; etc.), this verbage can be referred to directly in the XSLT facilitating ease of code generation. In addition to the CREATE operation, similar templates can be defined for INSERT and UPDATE operations as well.

Extensibility

Data warehouses must enable a business to confront and integrate data seamlessly in a "just-in-time" environment [3]. Design changes should not impact the end user heavily. If properly modulated, data structures can be significantly more stable than a constantly changing organizational structure and can show little impact on the operational system landscape [3].

AMs, DVs, and their XML representations are extensible and can accommodate this seamless integration. They allow for rapid linking of external information without destroying existing content [18]. When a schema is extended, new SQL code can be generated and run to non-destructively reflect those changes in the database. While keeping existing database integrity and structures intact, evolving information requirements are handled simply by adding new elements and attributes to the existing XML Schema. This feature of extensibility provides several benefits to the development process. It simplifies modeling concepts, enables modular and iterative development of the schema, makes the XML less prone to errors, provides a system of upgrading with minimal or no downtime, and allows for effective versioning control such that the XML Schema can be easily rolled back to a specific point in time [30].

Data Flexibility and Quality over Performance and Design

6NF-compliant data models such as the AM and DV are not easy to query, nor are they optimized for query performance in a typical relational database [27]. With the 6NF requirement

of irreducible elements, often more tables and joins are involved in queries. A 4-way join as opposed to a 2-table join is not ideal in terms of performance or design. However, 6NF data design does provide two major benefits: (a) scanning of only relevant data during searches and (b) reduction in data gaps by minimizing null values.

By separating elements to an irreducible state, queries can be performed only on those ties, attributes, links, or satellites necessary to the query results. Other tables can be removed from the operation. The anchor modeling group calls this practice "table elimination" [32]. In table elimination, a table *T* can be removed from a query's execution plan if two conditions are met: (1) no column from *T* is explicitly selected and (2) the number of rows in the returned data set is not affected by the join with *T*. Table elimination improves query performance, and the performance gain increases with the number of tables that can be eliminated because less data is being read and fewer joins are being performed. With the high degree of decomposition in 6NF data, these gains are often substantial [32].

The high rate of data decomposition also leads to sparse (or scattered) data. By separating data to irreducible elements, gaps in data can be reduced. If multiple attributes are being combined into one table, nulls are required to fill in gaps for incomplete data. However, by separating attributes into their own tables, attributes with a null value do not need to be stored at all. This practice reduces database size but can make queries more complex and costly since outer joins become necessary to accommodate all permutations of attribute combinations for a complete data record.

The purpose of a data warehouse is to be historical, integrated, and non-volatile. Taking these factors into consideration along with the fact that data warehouses feed data marts, which are the primary source of decision support data, then the proliferation of tables and query performance of the warehouse itself is a secondary concern to the data flexibility and quality that 6NF provides [27].

CHAPTER 7

CONCLUSION

This paper has presented a methodology for mapping Sixth Normal Form (6NF) temporal database design models such as the Anchor Model (AM) and the Data Vault (DV) to a standard eXtensible Markup Language (XML) Schema format. Due to its flexibility, extensibility, and scalability, XML Schema is the preferred means of describing structured data by which to exchange such information across the internet and other networked resources. By presenting such a methodology, fully-temporalized data conforming to 6NF can be presented to disparate systems in a structured format suitable for data exchange, especially 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.

The application of a 6NF-compliant XML Schema is particularly relevant to Data Warehousing 2.0 (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, and these snapshots must be created in such a way as to decompose data into a series of irreducible 6NF-compliant tables. 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 for the business that can withstand change over time. A 6NF-compliant XML Schema such as those presented in this work provides a mechanism for efficient and straightforward transfer of data into DW 2.0-compliant data warehouses.

While researchers have previously described an XML Schema definition (XSD) for the conceptual anchor model, we have expanded upon their work with several contributions:

1. We have identified and generated a metamodel using IDEF1X format for the AM-to-XSD transformation process, breaking down each simple and complex type used within the predefined schema definition with graphical depictions and detailed descriptions.
2. We have created a standard XSD format for a DV model schema, including the following:

a. Generating a set of formal definitions for DV schema components.

b. Creating an IDEF1X metamodel to describe the DV schema.

c. Identifying differences between the DV and AM schemas and addressing effects on structural format in the underlying metamodels and also in the resulting XSDs.

d. Breaking down the creation of simple and complex types within the DV schema with graphical depictions and detailed descriptions of each.

Finally we have addressed several benefits and advantages of having a standard XSD format described for both the AM and DV, especially pointing out the applicability of a DV XSD to DW 2.0 loading and staging. These factors include data interchange and the DV's ability to overcome common conflicts in data arriving from multiple sources, the scalability and ease of use which XML provides, the extensibility feature facilitated by 6NF-compliant data to make additions to data infrastructure without affecting existing components and while minimizing or eliminating downtime during updates, the ability to automatically generate SQL code from XSDs using Extensible Stylesheet Language Transformation (XSLT) technology, and the trade-off of preferring data flexibility and quality over query performance and table design.

REFERENCES

[1] Bernus, P., Mertins, K. and Schmidt, G. *Handbook on Architectures of Information Systems*, Springer-Verlag, Berlin Heidelberg, 2006, 225 – 234.

[2] Bird, L., Goodchild, A. and Halpin, T. "Object Role Modeling and XML Schema," International Conceptual Modeling Conference, Salt Lake City, Utah: 2000, 309-322.

[3] Damhof, R. and van As, L. "The Next Generation EDW: Letting Go of the Idea of a Single Version of the Truth," *Database Magazine (DB/M)* August 25, 2008, 1-10.

[4] Darwen, H. *Temporal Data and the Relational Model*, The Third Manifesto, University of Warwick, March 2004, available at http://www.dcs.warwick.ac.uk/~hugh/TTM/TemporalData.Warwick.pdf. Retrieved November 20, 2011.

[5] Date, C. J., Darwen, H. and Lorentzos, N. *Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and Relation Theory to the Problem of Temporal Database Management*, Morgan Kaufmann Publishers, Amsterdam, 2002, 53-88.

[6] Fagin, R. "A Normal Form for Relational Databases that is Based on Domains and Keys," *ACM Transactions on Database Systems* (6:3): September 1981, 387-415.

[7] Federal Information Processing Standards Publication 184, "Announcing the Standard for Integration Definition for Information Modeling (IDEF1X)", 1993, available at www.itl.nist.gov/fipspubs/idef1x.doc. Retrieved November 11, 2011.

[8] Golfarelli, M. and Rizzi, S. *Data Warehouse Design: Modern Principles and Methodologies*, McGraw Hill, New York, 2009, 103-108 and 145-148.

[9] Gregersen, H. and Jensen, J. S. "Temporal Entity-Relationship Models - A Survey," *IEEE Transactions on Knowledge and Data Engineering* (11:3), 1999, 464-497.

[10] Gregersen, H. and Jensen, J. S. "The Consensus Glossary of Temporal Database Concepts," *Temporal Databases: Research and Practice* (1399), 1998, 367-405.

[11] Herrington, J. "Code Generation in XSLT 2.0, Part 1: Generate SQL with XSLT 2.0", *IBM developerWorks Technical Library*, February 5, 2005, available at

http://www.ibm.com/developerworks/xml/library/x-xslphp1/index.html. Retrieved March 1, 2012.

[12] Inmon, W. H., Strauss, D. and Neushloss, G. *DW 2.0: The Architecture for the Next Generation of Data Warehousing*, Morgan Kaufmann Publishers, Amsterdam, 2008, 111-122.

[13] Jensen, C. S. and Snodgrass, R.T. "Temporal Data Management", *IEEE Transactions on Knowledge and Data Engineering* (11:1), 1999, 36-44.

[14] Johnston, T. and Weis, R. *Managing Time in Relational Databases: How to Design, Update and Query Temporal Data*, Morgan Kaufmann Publishers, Burlington, MA, 2011, 11-26.

[15] Jovanovic, V. and Bojicic, I. "Conceptual Data Vault Model," SAIS Conference, Atlanta, Georgia: March 23, 2012, 1-6.

[16] Khan, A. and Sum, M. "Introducing Design Patterns in XML Schemas," Oracle Sun Developer Network, 2008, available at http://developers.sun.com/jsenterprise/archive/nb_enterprise_pack/reference/techart/design_patterns.html, Retrieved March 23, 2012.

[17] Linstedt, D. "Data Vault Loading Specification v1.2", *DV Standards*, May 13, 2010, available at http://danlinstedt.com/datavaultcat/standards/data-vault-loading-specification-v1-2/. Retrieved February 1, 2012.

[18] Linstedt, D. "Data Vault Series 1 – Data Vault Overview", *The Data Administration Newsletter*, 2002, available at http://www.tdan.com/view-articles/5054/. Retrieved November 11, 2011.

[19] Linstedt, D. "Data Vault Series 2 – Data Vault Components", *The Data Administration Newsletter*, 2005, available at http://www.tdan.com/view-articles/ 5155/. Retrieved November 11, 2011.

[20] Linstedt, D. "Data Vault Series 3 – End Dates and Basic Joins", *The Data Administration Newsletter*, 2003, available at http://www.tdan.com/view-articles/ 5067/. Retrieved November 11, 2011.

[21] Linstedt, D. "Data Vault Series 4 – Link Tables", *The Data Administration Newsletter*, 2004, available at http://www.tdan.com/view-articles/5172/. Retrieved November 11, 2011.

[22] Linstedt, D. "Data Vault Series 5 – Data Vault Overview", *The Data Administration Newsletter*, 2005, available at http://www.tdan.com/view-articles/5285/. Retrieved November 11, 2011.

[23] Linstedt, D. "Data Vault Versus Dimensional", *danlinstedt.com*, 2010, available at http://danlinstedt.com/datavaultcat/data-vault-versus-dimensional-part-1. Retrieved March 30, 2012.

[24] Linstedt, D. *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*, CreateSpace, 2011, 110-149.

[25] Linstedt, D., Graziano, K. and Hultgren, H. *The New Business Supermodel, The Business of Data Vault modeling, 2nd edition*. Lulu.com, 2008, 75.

[26] MedBiquitous Consortium 2004, XML Schema Design Guidelines Version 1.3, 2004, available at http://www.medbiq.org/std_specs/techguidelines/ xmldesignguidelines.pdf. Retrieved March 20, 2012.

[27] O'Neill, B. "The Information Model & the Data Vault", *TDAN.com*, 2004.

[28] Olive, A. *Conceptual Modeling of Information Systems*, Springer-Verlag Berlin/Heidelberg, 2007, 28-34 and 422-425.

[29] Regardt, O., Rönnbäck, L., Bergholtz, M., Johannesson, P. and Wohed, P. "Analysis of Normal Forms for Anchor Tables", 2010, available at http://www.anchormodeling.com/wp-content/uploads/2010/08/6nf.pdf. Retrieved March 12, 2012.

[30] Rönnbäck, L. "Anchor Modeling – A Technique for Information Under Evolution", *GSE Nordics 2011 Conference*, Stockholm, Sweden, 2011, available at http://www.anchormodeling.com/wp-content/uploads/2011/05/Anchor-Modeling-Ordina.pdf. Retrieved January 13, 2012.

[31] Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P. and Wohed, P. "Anchor Modeling: Naming Convention", 2010, available at http://www.anchormodeling.com/ wp-content/uploads/2010/09/AM-Naming.pdf, Retrieved November 1, 2011.

[32] Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P. and Wohed, P. "Anchor Modeling - Agile Information Modeling in Evolving Data Environments", *Data & Knowledge Engineering* (69:12), December 2010, 1229–1253.

[33] Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P. and Wohed, P. "From Anchor Model to XML", 2010, available at http://www.anchormodeling.com/wp-content/uploads/2010/09/AM-XML.pdf. Retrieved December 12, 2012.

[34] Routlige, N., Bird, L. and Goodchild, A. "UML and XML Schema", *13th Australian Database Conference*, Melbourne, Australia, 2002, 1-10.

[35] Snodgrass, R. *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann Publishers, San Francisco, California, 2000, 12-23.

[36] Talmage, R. "SQL Essentials: SQL Server Data Importing: Do's and Don'ts", *SQL Server Professional*, June 2005, available at http://msdn.microsoft.com/en-us/library/aa175793%28v=sql.80%29.aspx. Retrieved March 3, 2012..

[37] Vrdoljak, B., Banek, M. and Rizzi, S. "Designing Web Warehouses from XML Schemas," *Proceedings 5th International Conference on Data Warehousing and Knowledge Discovery*, Prague, Czech Republic, 2003, 89-98.

[38] W3C (2000): Extensible Markup Language (XML) 1.0, *W3C XML Working Group*, available at http://www.w3.org/TR/REC-xml. Retrieved November 1, 2011.

[39] W3C (2001): XML Schema Parts 0-2: [Primer, Structures, Datatypes], *W3C XML Schema Working Group*, available at http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, and http://www.w3.org/TR/xmlschema-2/. Retrieved November 1, 2011.

[40] Walmsley, P. *Definitive XML Schema*, Prentice Hall, Upper Saddle River, New Jersey, 2002, 3-53 and 220-297.

[41] Wang, F., Zhou, X. and Zaniolo, C. "Using XML to Build Efficient Transaction-Time Temporal Database Systems," *22nd International Conference on Data Engineering*, Washington DC, 2006, 1-4.

[42] XML Tutorial, http://www.w3schools.com/xml/. Retrieved February 3, 2012.

APPENDIX A

FORMAL DEFINITIONS - ANCHOR MODEL AND DATA VAULT COMPONENTS

Anchor Model

Definition 1 (Identities). Let I be an infinite set of symbols, which are 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 (Anchor). An anchor A is a string. An extension of an anchor is a subset of I.

Definition 5 (Knot). A knot K is a string. A knot has a domain, which is I. A knot has a range, which is a data type D. An extension of a knot K with range D is a bijective relation over $I \times D$.

Definition 6 (Static Attribute). A static attribute B_S is a string. A static attribute B_S has an anchor A for domain and a data type D for range. An extension of a static attribute B_S is a relation over $I \times D$.

Definition 7 (Historized Attribute). A historized attribute B_H is a string. A historized attribute B_H has an anchor A for domain, a data type D for range, and a time type T as time range. An extension of a historized attribute B_H is a relation over $I \times D \times T$.

Definition 8 (Knotted Static Attribute). A knotted static attribute B_{KS} is a string. A knotted static attribute B_{KS} has an anchor A for domain and a knot K for range. An extension of a knotted static attribute B_{KS} is a relation over $I \times I$.

Definition 9 (Knotted Historized Attribute). A knotted historized attribute B_{KH} is a string. A knotted historized attribute B_{KH} has an anchor A for domain, a knot K for range, and a time type T for time range. An extension of a knotted historized attribute B_{KH} is a relation over $I \times I \times T$.

Definition 10 (Anchor Role). An anchor role is a string. Every anchor role has a type, which is an anchor.

Definition 11 (Knot Role). A knot role is a string. Every knot role has a type, which is a knot.

Definition 12 (Static Tie). A static tie T_S is a set of at least two anchor roles. An instance t_S of a static tie $T_S = \{R_1,...,R_n\}$ is a set of pairs $[R_i; v_i]$, $i = 1,\ldots,n$, where R_i is an anchor role, $v_i \in I$ and $n \geq 2$. An extension of a static tie T_S is a set of instances of T_S.

Definition 13 (Historized Tie). A historized tie T_H is a set of at least two anchor roles and a time type T. An instance t_H of a historized tie $T_H = \{R_1,\ldots,R_n, T\}$ is a set of pairs $[R_i, v_i]$, $i = 1,\ldots,n$ and a time point p, where R_i is an anchor role, $v_i \in I$, $p \in T$, and $n \geq 2$. An extension of a historized tie T_H is a set of instances of T_H.

Definition 14 (Knotted Static Tie). A knotted static tie T_{KS} is a set of at least two anchor roles and one or more knot roles. An instance t_{KS} of a static tie $T_{KS} = \{R_1,\ldots,R_n, S_1,\ldots,S_m\}$ is a set of pairs $[R_i, v_i]$, $i = 1,\ldots,n$ and $[S_j, w_j]$, $j = 1,\ldots,m$, where R_i is an anchor role, S_j is a knot role, $v_i \in I$, $w_j \in I$, $n \geq 2$, and $m \geq 1$. An extension of a knotted static tie T_{KS} is a set of instances of T_{KS}.

Definition 15 (Knotted Historized Tie). A knotted historized tie T_{KH} is a set of at least two anchor roles, one or more knot roles and a time type T. An instance t_{KH} of a historized tie $T_{KH} = \{R_1,\ldots,R_n, S_1,\ldots,S_m, T\}$ 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 an anchor role, S_j is a knot role, $v_i \in I$, $w_j \in I$, $p \in T$, $n \geq 2$, and $m \geq 1$. An extension of a knotted historized tie T_{KH} is a set of instances of T_{KH}.

Definition 16 (Identifier). Let T be a (static, historized, knotted, or knotted historized) tie. An identifier for T is a subset of T containing at least one anchor role. Furthermore, if T is a historized or knotted historized tie, where T is the time type in T, every identifier for T must contain T.

Data Vault

Definition 1 (Identities). Let U be a finite but arbitrarily large set of symbols, which are 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 S_R is a string. A referenced satellite S_R 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 S_R 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.

Definition 9 (Link Role). A link role R_L is a string. Every link role has a type, which is a link.

Definition 10 (Reference Role). A reference role R_R 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 $L = \{R_1,\ldots,R_n, T\}$ 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 a link L is a set of instances of L_i.

Definition 12 (Referenced Link). A referenced link L_R is a set of at least two hub or link roles, one or more reference roles and a time type T. An instance L_i of a referenced link $L = \{R_1,\ldots,R_n, S_1,\ldots,S_m, T\}$ 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_R is a set of instances of L_i.

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-DV* represent a conceptual Data Vault model. An identifier for *C-DV* is a relation over $U \times H \times L \times S \times R \times T \times D$.

APPENDIX B

XML SCHEMA DEFINITION FOR ANCHOR MODEL SCHEMA

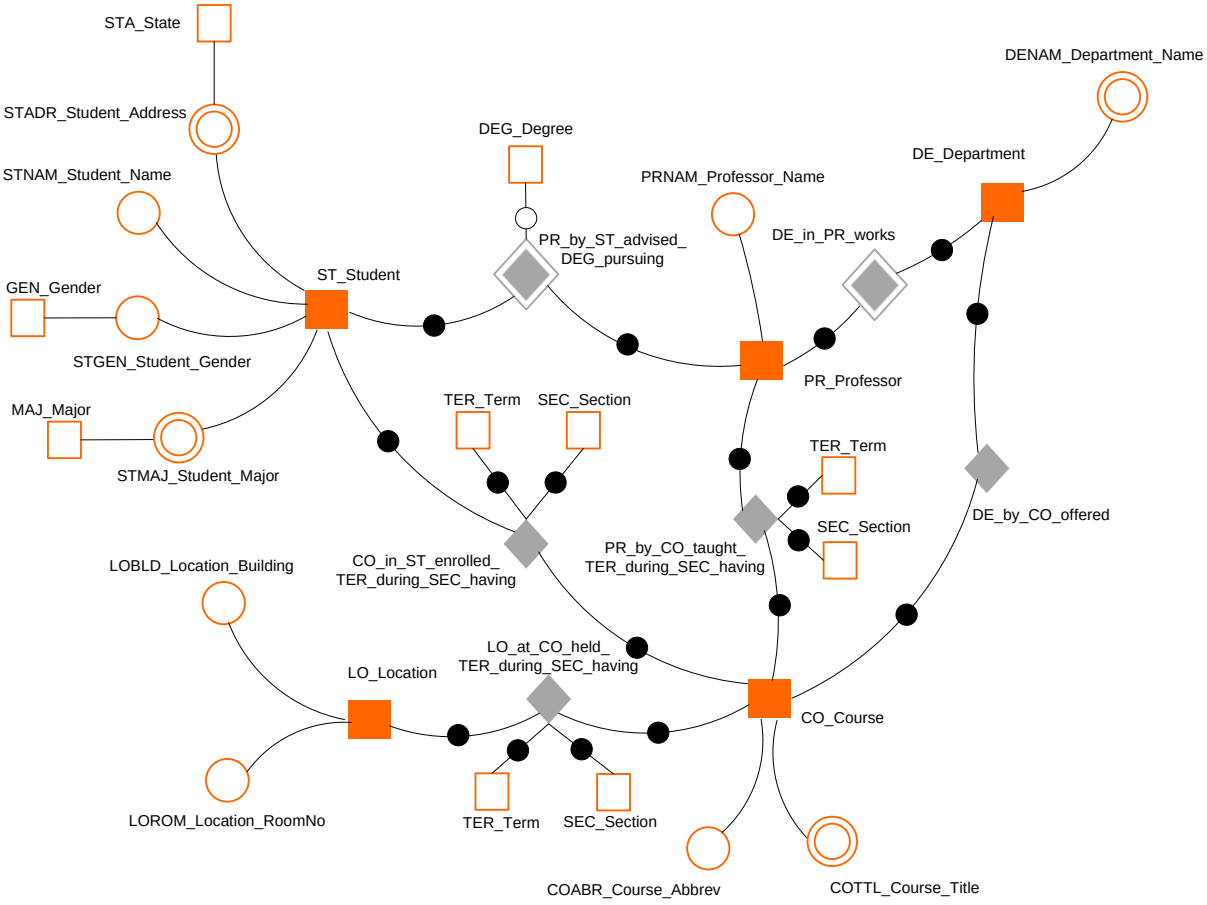
```
1   <?xml version="1.0"?>
2   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://anchormodeling.com/schema”
        targetNamespace="http://anchormodeling.com/schema"
        xpathDefaultNamespace="##targetNamespace"
        elementFormDefault="qualified">
3   <xs:simpleType name="name">
4         <xs:restriction base="xs:string"/>
5   </xs:simpleType>
6   <xs:simpleType name="identity">
7         <xs:restriction base="xs:string"/>
8   </xs:simpleType>
9   <xs:simpleType name="dataType">
10        <xs:restriction base="xs:string"/>
11  </xs:simpleType>
12  <xs:simpleType name="timeType">
13        <xs:restriction base="xs:string"/>
14  </xs:simpleType>
15  <xs:complexType name="role">
16        <xs:attribute name="role" type="name" use="required"/>
17        <xs:attribute name="type" type="name" use="required"/>
18        <xs:attribute name="identifier" type="xs:boolean" use="optional"
              default="false"/>
19  </xs:complexType>
20  <xs:complexType name="knot">
21        <xs:attribute name="mnemonic" type="name" use="required"/>
22        <xs:attribute name="descriptor" type="name" use="required"/>
23        <xs:attribute name="identity" type="identity" use="required"/>
24        <xs:attribute name="dataRange" type="dataType" use="required"/>
25  </xs:complexType>
26  <xs:complexType name="attribute">
27        <xs:attribute name="mnemonic" type="name" use="required"/>
28        <xs:attribute name="descriptor" type="name" use="required"/>
29        <xs:attribute name="knotRange" type="name" use="optional"/>
30        <xs:attribute name="dataRange" type="dataType" use="optional"/>
```

```
31          <xs:attribute name="timeRange" type="timeType" use="optional"/>
32          <xs:assert test="count(@knotRange) + count(@dataRange) = 1"/>
33    </xs:complexType>
34    <xs:complexType name="anchor">
35          <xs:sequence>
36                <xs:element name="attribute" type="attribute" minOccurs="1"
                      maxOccurs="unbounded"/>
37          </xs:sequence>
38          <xs:attribute name="mnemonic" type="name" use="required"/>
39          <xs:attribute name="descriptor" type="name" use="required"/>
40          <xs:attribute name="identity" type="identity" use="required"/>
41    </xs:complexType>
42    <xs:complexType name="tie">
43          <xs:sequence>
44                <xs:element name="anchorRole" type="role" minOccurs="2"
                      maxOccurs="unbounded"/>
45                <xs:element name="knotRole" type="role" minOccurs="0"
                      maxOccurs="unbounded"/>
46          </xs:sequence>
47          <xs:attribute name="timeRange" type="timeType" use="optional"/>
48          <xs:assert test="anchorRole[@identifier = 'true']"/>
49    </xs:complexType>
50    <xs:complexType name="schema">
51          <xs:choice minOccurs="0" maxOccurs="unbounded">
52                <xs:element name="knot" type="knot"/>
53                <xs:element name="anchor" type="anchor"/>
54                <xs:element name="tie" type="tie"/>
55          </xs:choice>
56          <xs:assert test="every $k in anchor/attribute/@knotRange
                satisfies knot[@mnemonic = $k]"/>
57          <xs:assert test="every $a in tie/anchorRole/@type satisfies
                anchor[@mnemonic = $a]"/>
58          <xs:assert test="every $k in tie/knotRole/@type satisfies
                knot[@mnemonic = $k]"/>
59          <xs:assert test="not(some $t1 in tie, $t2 in $t1/preceding-
                sibling::tie satisfies deep-equal($t1,$t2))"/>
60          <xs:assert test="not(some $t1 in tie/*, $t2 in $t1/preceding-
                sibling::node() satisfies deep-equal($t1,$t2))"/>
```

```
61          <xs:assert test="not(some $t1 in anchor, $t2 in $t1/preceding-
                sibling::anchor satisfies $t1/@mnemonic = $t2/@mnemonic)"/>
62          <xs:assert test="not(some $t1 in knot, $t2 in $t1/preceding-
                sibling::knot satisfies $t1/@mnemonic = $t2/@mnemonic)"/>
63          <xs:assert test="not(some $t1 in anchor/attribute, $t2 in
                $t1/preceding-sibling::attribute satisfies $t1/@mnemonic =
                $t2/@mnemonic)"/>
64      </xs:complexType>
65      <xs:element name="schema" type="schema"/>
66      </xs:schema>
```

APPENDIX C

CONCEPTUAL ANCHOR MODEL – COLLEGE REGISTRATION SYSTEM



APPENDIX D

XML OF COLLEGE REGISTRATION SYSTEM ANCHOR SCHEMA

```
1     <?xml version="1.0"?>
2     <schema xmlns="http://anchormodeling.com/schema">
3           <knot mnemonic="STA" descriptor="State" identity="string"
                  dataRange="string" />
4           <knot mnemonic="GEN" descriptor="Gender" identity="string"
                  dataRange="string" />
5           <knot mnemonic="MAJ" descriptor="Major" identity="string"
                  dataRange="string" />
6           <knot mnemonic="DEG" descriptor="Degree" identity="string"
                  dataRange="string" />
7           <knot mnemonic="SEC" descriptor="Section" identity="string"
                  dataRange="string" />
8           <knot mnemonic="TER" descriptor="Term" identity="string"
                  dataRange="string" />
9           <anchor mnemonic="ST" descriptor="Student" identity="integer">
10                <attribute mnemonic="STADR" descriptor="Student_Address"
                             knotRange="STA" timeRange="date" />
11                <attribute mnemonic="STNAM" descriptor="Student_Name"
                             dataRange="string" />
12                <attribute mnemonic="STGEN" descriptor="Student_Gender"
                             knotRange="GEN" />
13                <attribute mnemonic="STMAJ" descriptor="Student_Major"
                             knotRange="MAJ" timeRange="date" />
14          </anchor>
15          <anchor mnemonic="LO" descriptor="Location" identity="integer">
16                <attribute mnemonic="LOBLD" descriptor="Location_Building"
                             dataRange="string" />
17                <attribute mnemonic="LOROM" descriptor="Location_RoomNo"
                             dataRange="string" />
18          </anchor>
19          <anchor mnemonic="PR" descriptor="Professor" identity="integer">
20                <attribute mnemonic="PRNAM" descriptor="Professor_Name"
                             dataRange="string" />
21          </anchor>
22          <anchor mnemonic="CO" descriptor="Course" identity="integer">
```

```
23              <attribute mnemonic="COABR" descriptor="Course_Abbrev"
                           dataRange="string" />
24              <attribute mnemonic="COTTL" descriptor="Course_Title"
                           dataRange="string" timeRange="date" />
25        </anchor>
26        <anchor mnemonic="DE" descriptor="Department" identity="integer">
27              <attribute mnemonic="DENAM" descriptor="Department_Name"
                           dataRange="string" timeRange="date" />
28        </anchor>
29        <tie timeRange="date">
30              <anchorRole type="ST" role="advised" identifier="true" />
31              <anchorRole type="PR" role="by" identifier="true" />
32              <knotRole type="DEG" role="pursuing" identifier="false" />
33        </tie>
34        <tie timeRange="date">
35              <anchorRole type="PR" role="works" identifier="true" />
36              <anchorRole type="DE" role="in" identifier="true" />
37        </tie>
38        <tie>
39              <anchorRole type="CO" role="offered" identifier="true" />
40              <anchorRole type="DE" role="by" identifier="true" />
41        </tie>
42        <tie>
43              <anchorRole type="CO" role="taught" identifier="true" />
44              <anchorRole type="PR" role="by" identifier="true" />
45              <knotRole type="TER" role="during" identifier="true" />
46              <knotRole type="SEC" role="having" identifier="true" />
47        </tie>
48        <tie>
49              <anchorRole type="ST" role="enrolled" identifier="true" />
50              <anchorRole type="CO" role="in" identifier="true" />
51              <knotRole type="TER" role="during" identifier="true" />
52              <knotRole type="SEC" role="having" identifier="true" />
53        </tie>
54        <tie>
55              <anchorRole type="CO" role="held" identifier="true" />
56              <anchorRole type="LO" role="at" identifier="true" />
57              <knotRole type="TER" role="during" identifier="true" />
```

```
58                <knotRole type="SEC" role="having" identifier="true" />
59          </tie>
60    </schema>
```

APPENDIX E

XML SCHEMA DEFINITION OF DATA VAULT SCHEMA

```
1     <?xml version="1.0"?>
2     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://localhost/DataVault/schema"
          targetNamespace="http://localhost/DataVault/schema"
          xpathDefaultNamespace="##targetNamespace"
          elementFormDefault="qualified">
3           <xs:complexType name="role">
4                 <xs:attribute name="type" type="xs:string" use="required"/>
5                 <xs:attribute name="role" type="xs:string" use="required"/>
6                 <xs:attribute name="identifier" type="xs:boolean"
                      use="optional" default="false"/>
7           </xs:complexType>
8           <xs:complexType name="reftable">
9                 <xs:attribute name="name" type="xs:string" use="required"/>
10                <xs:attribute name="identity" type="xs:string"
                      use="required"/>
11                <xs:attribute name="description" type="xs:string"
                      use="required"/>
12          </xs:complexType>
13          <xs:complexType name="satellite">
14                <xs:attribute name="name" type="xs:string" use="required"/>
15                <xs:attribute name="hubKey" type="xs:integer"
                      use="required"/>
16                <xs:attribute name="loadDate" type="xs:date"
                      use="required"/>
17                <xs:attribute name="recSource" type="xs:string"
                      use="required"/>
18                <xs:attribute name="value" type="xs:string"
                      use="optional"/>
19                <xs:attribute name="refValue" type="xs:string"
                      use="optional"/>
20                <xs:assert test="count(@refValue) + count(@value) = 1"/>
21          </xs:complexType>
22          <xs:complexType name="hub">
23                <xs:sequence>
```

```
24                    <xs:element name="satellite" type="satellite"
                          minOccurs="1" maxOccurs="unbounded"/>
25              </xs:sequence>
26              <xs:attribute name="name" type="xs:string" use="required"/>
27              <xs:attribute name="hubKey" type="xs:integer"
                    use="required"/>
28              <xs:attribute name="loadDate" type="xs:date"
                    use="required"/>
29              <xs:attribute name="recSource" type="xs:string"
                    use="required"/>
30              <xs:attribute name="busKey" type="xs:string"
                    use="required"/>
31        </xs:complexType>
32        <xs:complexType name="link">
33              <xs:sequence>
34                    <xs:element name="hubRole" type="role" minOccurs="2"
                          maxOccurs="unbounded"/>
35                    <xs:element name="refRole" type="role" minOccurs="0"
                          maxOccurs="unbounded"/>
36                    <xs:element name="linkRole" type="role" minOccurs="0"
                          maxOccurs="unbounded"/>
37                    <xs:element name="satellite" type="satellite"
                          minOccurs="0" maxOccurs="unbounded"/>
38              </xs:sequence>
39              <xs:attribute name="loadDate" type="xs:date"
                    use="required"/>
40              <xs:attribute name="recSource" type="xs:string"
                    use="required"/>
41              <xs:assert test="hubRole[@identifier = 'true']"/>
42        </xs:complexType>
43        <xs:complexType name="schema">
44              <xs:choice minOccurs="0" maxOccurs="unbounded">
45                    <xs:element name="reftable" type="reftable"/>
46                    <xs:element name="hub" type="hub"/>
47                    <xs:element name="link" type="link"/>
48              </xs:choice>
49              <xs:assert test="every $r in hub/satellite/@refValue
                    satisfies reftable[@identity = $r]"/>
```

```
50              <xs:assert test="every $h in link/hubRole/@type satisfies
                    hub[@hubKey = $h]"/>
51              <xs:assert test="every $r in link/refRole/@type satisfies
                    reftable[@identity = $r]"/>
52              <xs:assert test="every $l in link/linkRole/@type satisfies
                    reftable[@identity = $l]"/>
53              <xs:assert test="not(some $t1 in link, $t2 in
                    $t1/preceding-sibling::link satisfies deep-
                    equal($t1,$t2))"/>
54              <xs:assert test="not(some $t1 in link/*, $t2 in
                    $t1/preceding-sibling::node() satisfies deep-
                    equal($t1,$t2))"/>
55              <xs:assert test="not(some $t1 in hub, $t2 in $t1/preceding-
                    sibling::hub satisfies $t1/@hubKey = $t2/@hubKey)"/>
56              <xs:assert test="not(some $t1 in reftable, $t2 in
                    $t1/preceding-sibling::reftable satisfies $t1/@identity
                    = $t2/@identity)"/>
57              <xs:assert test="not(some $t1 in hub/satellite, $t2 in
                    $t1/preceding-sibling::satellite satisfies $t1/@hubKey
                    = $t2/@hubKey)"/>
58          </xs:complexType>
59      <xs:element name="schema" type="schema"/>
60      </xs:schema>
```

APPENDIX F

XML OF COLLEGE REGISTRATION SYSTEM DATA VAULT SCHEMA

```
1     <?xml version="1.0"?>
2     <schema xmlns="http://localhost/DataVault/schema">
3           <reftable name="STATE" identity="string" description="string" />
4           <reftable name="GENDER" identity="string" description="string" />
5           <reftable name="MAJOR" identity="string" description="string" />
6           <reftable name="DEGREE" identity="string" description="string" />
7           <reftable name="SECTION" identity="string" description="string" />
8           <reftable name="TERM" identity="string" description="string" />
9           <hub name="HUB_STUDENT" hubKey="integer" loadDate="date"
                 recSource="string" busKey="string">
10               <satellite name="SAT_STUDENT_ADDRESS" hubKey="integer"
                           loadDate="date" recSource="string"
                           refValue="STATE" />
11               <satellite name="SAT_STUDENT_NAME" hubKey="integer"
                           loadDate="date" recSource="string"
                           value="string" />
12               <satellite name="SAT_STUDENT_GENDER" hubKey="integer"
                           loadDate="date" recSource="string"
                           refValue="GENDER" />
13               <satellite name="SAT_STUDENT_MAJOR" hubKey="integer"
                           loadDate="date" recSource="string"
                           refValue="MAJOR" />
14          </hub>
15          <hub name="HUB_LOCATION" hubKey="integer" loadDate="date"
                 recSource="string" busKey="string">
16               <satellite name="SAT_LOCATION_BUILDING" hubKey="integer"
                           loadDate="date" recSource="string" value="string" />
17               <satellite name="SAT_LOCATION_ROOMNO" hubKey="integer"
                           loadDate="date" recSource="string" value="string" />
18          </hub>
19          <hub name="HUB_PROFESSOR" hubKey="integer" loadDate="date"
                 recSource="string" busKey="string">
20               <satellite name="SAT_PROFESSOR_NAME" hubKey="integer"
                            loadDate="date" recSource="string"
                            value="string" />
```

```
21          </hub>
22          <hub name="HUB_COURSE" hubKey="integer" loadDate="date"
                 recSource="string" busKey="string">
23               <satellite name="SAT_COURSE_ABBREV" hubKey="integer"
                           loadDate="date" recSource="string"
                           value="string" />
24               <satellite name="SAT_COURSE_TITLE" hubKey="integer"
                           loadDate="date" recSource="string"
                           value="string" />
25          </hub>
26          <hub name="HUB_DEPARTMENT" hubKey="integer" loadDate="date"
                 recSource="string" busKey="string">
27               <satellite name="SAT_DEPARTMENT_NAME" hubKey="integer"
                           loadDate="date" recSource="string"
                           value="string" />
28          </hub>
29          <link loadDate="date" recSource="string">
30               <hubRole type="HUB_STUDENT" role="advised"
                          identifier="true" />
31               <hubRole type="HUB_PROFESSOR" role="by"
                          identifier="true" />
31               <refRole type="DEGREE" role="pursuing"
                          identifier="false" />
32          </link>
33          <link loadDate="date" recSource="string">
34               <hubRole type="HUB_PROFESSOR" role="works"
                          identifier="true" />
35               <hubRole type="HUB_DEPARTMENT" role="in"
                          identifier="true" />
36          </link>
37          <link loadDate="date" recSource="string">
38               <hubRole type="HUB_COURSE" role="offered"
                          identifier="true" />
39               <hubRole type="HUB_DEPARTMENT" role="by"
                          identifier="true" />
40          </link>
41          <link loadDate="date" recSource="string">
42               <hubRole type="HUB_COURSE" role="taught"
```

```
                        identifier="true" />
43              <hubRole type="HUB_PROFESSOR" role="by"
                        identifier="true" />
44              <refRole type="TERM" role="during" identifier="true" />
45              <refRole type="SECTION" role="having" identifier="true" />
46          </link>
47          <link loadDate="date" recSource="string">
48              <hubRole type="HUB_STUDENT" role="enrolled"
                        identifier="true" />
49              <hubRole type="HUB_COURSE" role="in" identifier="true" />
50              <refRole type="TERM" role="during" identifier="true" />
51              <refRole type="SECTION" role="having" identifier="true" />
52          </link>
53          <link loadDate="date" recSource="string">
54              <hubRole type="HUB_COURSE" role="held" identifier="true" />
55              <hubRole type="HUB_LOCATION" role="at" identifier="true" />
56              <refRole type="TERM" role="during" identifier="true" />
57              <refRole type="SECTION" role="having" identifier="true" />
58          </link>
59      </schema>
```

APPENDIX G

LOGICAL DATA VAULT MODEL – COLLEGE REGISTRATION SYSTEM