Designing Good Semi-structured Databases

Sin Yeung Lee[1] Mong Li Lee[1] Tok Wang Ling[1] Leonid A. Kalinichenko[2]

[1] School of Computing
National University of Singapore
{jlee, leeml, lingtw}@comp.nus.edu.sg

[2] Institute for Problems of Informatics
Russian Academy of Sciences
leonidk@synth.ipi.ac.ru

Abstract. Semi-structured data has become prevalent with the growth of the Internet and other on-line information repositories. Many organizational databases are presented on the web as semi-structured data. Designing a "good" semi-structured database is increasingly crucial to prevent data redundancy, inconsistency and updating anomalies. In this paper, we define a semi-structured schema graph and identify the various anomalies that may occur in the graph. A normal form for semi-structured schema graph, S3-NF, is proposed. We present two approaches to design S3-NF database, namely, restructuring by decomposition and the ER approach. The first approach consists of a set of rules to decompose a semi-structured schema graph into S3-NF. The second approach uses the ER model to remove anomalies at the semantic level.

1 Introduction

The growth of the Internet and other on-line information repositories has greatly simplified the access to numerous sources of information/data, especially through the World Wide Web. The data is presented in various forms: At one extreme we find data coming from traditional relational/object-oriented databases, with a completely known structure. At the other extreme we have data which is fully unstructured, eg images, sounds and raw text. But most of the data fall somewhere in between (semi-structured) for a variety of reasons: the data may be structured, but the structure is not known to the user; the user may know the structure, but chooses to ignore it for browsing purposes. Examples of semi-structured data include HTML documents where the structure is imposed by tags, and bibliography files where some structure is imposed by fields such as the author and the title to an otherwise unstructured text file. Note that the tags and fields are optional.

The nature of semi-structured data is fundamentally different from data in traditional databases and hence raises many new issues. Some of the common scenarios involve extracting data from the diverse information repositories across the Internet, and integrating the data from heterogeneous sources. These tasks are made more difficult because we have only a partial knowledge of the structure and that the structure is potentially "deeply nested" or even cyclic. Many researchers have proposed semi-structured data models, databases, and languages to model, store and query the World Wide Web data [1, 5, 7, 18, 15]. These works use some graph or tree models [5, 18] which provide a flexible representation of data coming from arbitrary heterogenous sources. Proposed query languages are closely tied to these data models.

Unfortunately, there is no notion of a precise or explicit schema in all these semi-structured databases. All the schematic information is embedded in the graphs which may change dynamically. The lack of a schema poses two problems. First, it is difficult for a user to formulate a meaningful query on the semi-structured data without any knowledge of how the data is organized. Second, it is difficult for the query processor to generate efficient plans for queries on the semi-structured data without any schema to guide it. As a result, [8] introduces DataGuides which dynamically generate and maintain structural summaries of semi-structured databases. The Dataguides are used in the Lore DBMS [15] for the user to carry out structure browsing and query formulation, as well as for the query processor to optimize query execution. More work to discover schemas from semi-structured data can be found in [21, 16, 4] and others.

However, to date, we observe that the concept of a *well-formed* or a *normal form* semi-structured schema has never been considered. This has come to our attention because data redundancy and data inconsistency may occur in a semi-structured database if the schema is not designed properly. In a traditional database, redundancy has inevitably caused updating anomalies [6]. A well-established technique to remove undesirable updating anomalies and data redundancy from relations and nested relations is normalization [13, 20, 14, 17, 19, 12].

In this paper, we will focus on how to design good semi-structured databases. We assume that we can extract the schema from a semi-structured data source or database. We will define the concept of a semi-structured schema graph and investigate the various anomalies that may appear in this model. A normal form for semi-structured schema graph, S3-NF, is proposed. Our normal form not only deals with functional dependencies and multivalued dependencies, but also removes identified anomalies from the semi-structured schema graph. Two approaches to designing S3-NF databases are given, namely, the restructuring approach and the Entity-Relationship (ER) approach. The former consists of a set of rules to decompose a semi- structured schema graph into S3-NF while the latter uses the ER model to remove the anomalies and redundancies at the semantic level. We envisage the growing importance of well designed semi-structured databases due to the increasing popularity of XML for data representation on the Web [3].

The rest of the paper is organized as follows. Section 2 gives the definitions of various concepts in a semi-structured schema graph. The anomalies that may occur in the graph is also presented. In section 3, a normal form for semi-structured schema graph, S3-NF, is defined. We also show how we can restructure a semi-structured schema graph to obtain S3-NF which does not have the undesirable anomalies. Section 4 discusses the ER approach to designing a S3-NF database. Finally, we conclude in Section 5.

2 Semi-Structured Graph: Definition and Anomalies

Data modeling people have long noticed the fact that if the database attributes are fixed in structure, the modeling power of the data model is greatly reduced. With the introduction of XML, semi-structured data model becomes widespread. However, the problem of anomalies, which was well solved for various relational models, including the flat relation model [2, 11] and the nested relation model [12], will appear again. In this section, we define the concept of Semi-Structured Schema Graph and discuss various anomalies that may appear in this model.

2.1 Motivating Example

We use the Object Exchange Model (OEM) [18] adopted in Stanford's Lore DBMS [15] to represent semi-structured data. OEM is self-describing as each object contains its own schema and there is no distinction between schema and data. Each object in OEM has an object identifier (oid) and a value. The value is either atomic, (integer, real, string, gif, html, audio, etc) or complex, that is, a set of object references denoted as a set of (label, oid) pairs. The labels are taken from the atomic type string. We can visualize OEM as a labeled graph in which vertices correspond to objects and edges represent the object-subobject relationship. Each edge has a label describing the precise nature of the relationship. Based on the OEM, Lorel [1] uses the familiar *select-from-where* syntax and path expressions to traverse the semi-structured data. For example, the path expression *Student.Course* specifies the Course subobjects of object Student.

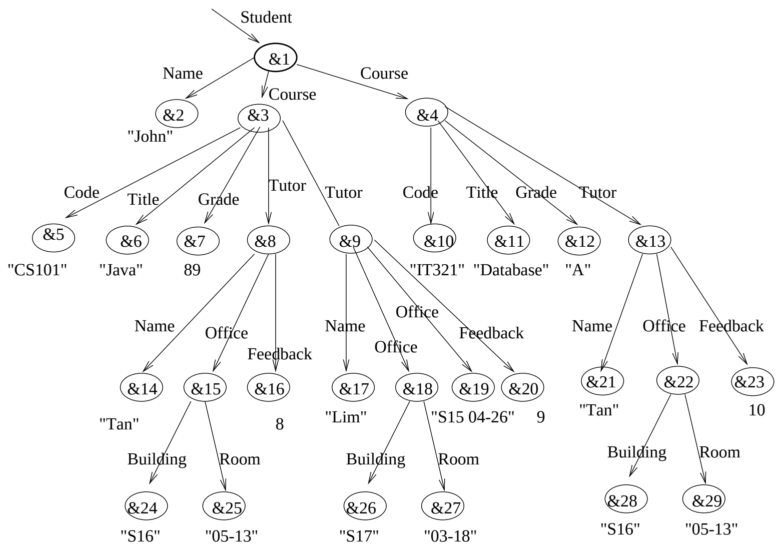

Fig. 1. Example of an OEM graph

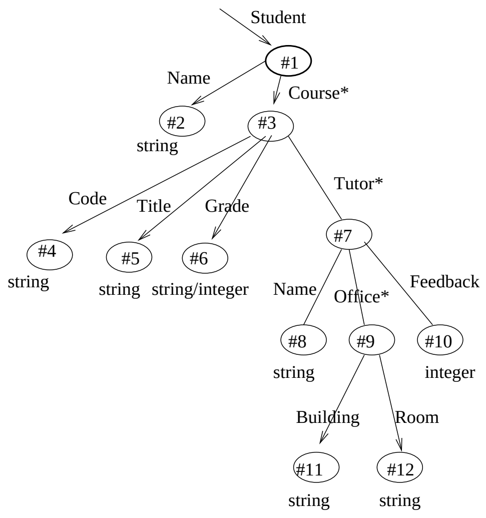

Fig. 2. Schema of Figure 1

Figure 1 shows an example of an OEM graph. A student (with attribute Name) can be enrolled in many courses (with attributes Code and Title). The student's grade for a course is kept in the attribute Grade. A tutor, with attributes Name and Office, can teach more than one course. A student can have more than one tutor for a course. The student's evaluation of a course tutor is kept in the attribute Feedback. There is no fixed or regular structure in the graph

1. a student may take zero, one or more courses
2. a course may have one or more tutors
3. a tutor may have one or more offices
4. a student's course grade may be in marks (0-100) or in grades (A to F)
5. a tutor's office is a complex structure consisting of building and room

The associated schema is shown in Figure 2.

The semi-structured database in Figure 1 is not well-designed because it contains data redundancy. The code and title of a course will be stored as many times as the number of students taking the course. Similarly, information of a tutor such as his/her office and age will also be duplicated since a tutor can teach more than one course and more than one student. Such data redundancies can be removed if we have links, denoted by dashed edges, to Course and Tutor objects as shown in Figure 3. The associated schema is shown in Figure 4.

2.2 Definition of Semi-Structured Schema Graph (S3-Graph)

Definition 1. A *Semi-Structured Schema Graph (S3-Graph)* is a directed graph defined as follows,

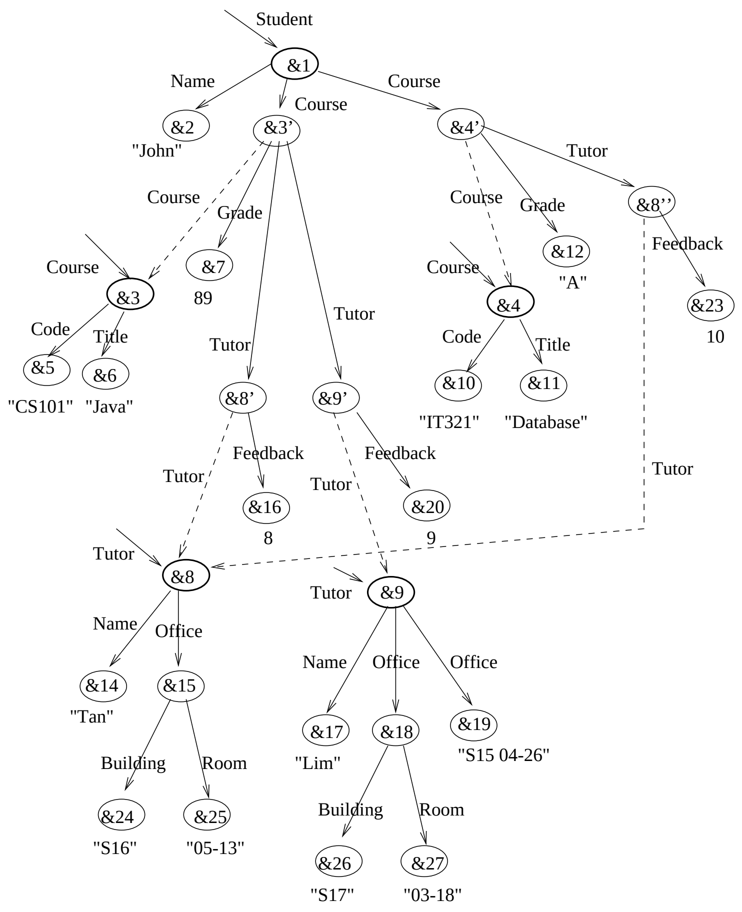

Fig. 3. A Well-Designed Semi-Structured Database

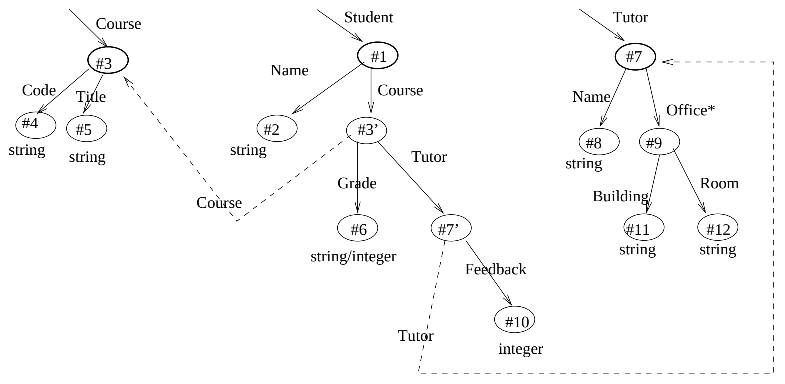

Fig. 4. Schema of the Well-Designed Semi-Structured Database in Figure 3

Each node of the graph can be classified into one of the following types:

1. *entity node* is a node which represents an entity. This entity can be of basic atomic data type such as string, date or complex object type such as student. If the entity node represents a basic atomic data type instead of a complex object type, then the entity node is also known as *leaf entity node*. Intuitively, a leaf node does not have any child (defined later) in the S3-Graph. We further attach the atomic data type as a label to the corresponding leaf entity node in the S3-Graph.
2. *reference node* is a node which references to another entity node.

Each directed edge in the graph is associated with a *tag*. The tag represents the relationship between the source node and the destination node. The tag may be suffixed with a "*". The interpretations of tag and the suffix depend on the type of edge. There are three types of edges:

1. Component Edge
   A node V_1 is connected to another node V_2 via a *component edge* with a tag T if V_2 is a component of V_1. We represent this edge by a solid arrow line. If T is suffixed with a "*", the relationship is interpreted as "The entity type represented by V_1 has many T". Otherwise, the relationship is interpreted as "The entity type represented by V_1 has at most one T".
2. Referencing Edge
   A node V_1 is connected to another node V_2 via a *referencing edge* if V_1 references the entity represented by node V_2. We represent this edge by a dashed arrow line. In this case, the relationship is interpreted as "V_1 references V_2".
3. Root Edge
   A node V_1 is pointed by a *root edge* with a tag T if the entity type represented by V_1 is owned by the database. We represent this edge by a solid arrow line without any source node for the edge. The tag T is not suffixed. In this case, the edge is interpreted as "The database has many T". Furthermore, we shall call V_1 as a *root node* in the S3-Graph.

Finally, some roles R can be associated with a node V if there is a directed *(component or referencing)* edge pointing to V with tag R after removing any suffix "*".

Example 1. In Figure 4, node #1 represents an entity node, which represents the entity STUDENT. This is also one of the root nodes. This node is associated with the role "Student". Node #2 is another entity node of which database instance holds a string representing the NAME of a student. It is associated with the role "Name". It is also a leaf node associated the atomic data type "string". Hence any "NAME" data is of string type. The directed edge between node #1 and node #2 represents "Each STUDENT has at most one NAME".

Node #3' is a reference node. It references the entity node which represents COURSE. In this case, #3' represents the same entity as in #3. The edge connecting #1 and #3' is interpreted as "Each STUDENT has many COURSEs".

Note that in a more complex example, a node can be associated with more than two roles. For example, a PERSON can be both a LAB-MEMBER as well as a LAB-SUPERVISOR.

Definition 2. Given a S3-Graph G, if a node P of role R_P is connected via a component edge to another node C of role R_C, then R_P is referred to as a *strongly connected parent* of R_C, and R_C as a *strongly connected child* of R_P.

Similarly, if there is a path of only component edges which connects a node A of role R_A to a node D of role R_D, then R_A is said to be a *strongly connected ancestor* of R_D. If R_A is a strongly connected ancestor of R_D, then R_D is one of the *strongly connected descendants* of R_A.

Example 2. Refer to Figure 4, Student is a strongly connected parent of Tutor, and Office is a strongly connected descendent of Tutor. However, Office is not a strongly connected descendent of Student because any path that connects a node of Student role to a node of Office role must go through a referencing edge between node #7 and node #7'.

In the above example, we see that strongly connected ancestor is not necessarily transitive. In fact, the strongly connected parent resembles to the physical parents of the hierarchical database models such as IMS. In this case, the logical parents is realized through our referencing edges.

For the rest of the paper, we shall refer strongly connected ancestor simply as ancestor. Likewise, we shall refer strongly connected descendent, strongly connected parent and strongly connected child as descendent, parent and child respectively.

2.3 Anomalies of Semi-Structured Data

A database in a RDBMS can be considered as a special form of semi-structured data. In general, semi-structured data involves anomalies which are similar to those identified for 1NF relations. Before we illustrate the various anomalies, we will first define the database instance – a semi-structured data graph, which is an image of a given schema.

Definition 3. A *semi-structured data graph* D with respect to a S3-Graph G is a graph showing a database instance such that

1. (Node correspondance)
   Each node v in D is associated with one and only one node V of role R in G. We call V in the schema the *definition node* of the data node v. Furthermore, v is playing the role R.
2. (Edge correspondance)
   A component edge in D is represented by a solid arrow line. If v_1 is connected to v_2 via a component edge e with tag T, then their definition nodes in G must be likewise connected via a component edge E with tag T, with or without the suffix "*". We call the edge E in G the *definition edge* of the

edge e in D. Likewise for the referencing edge which is represented by a dashed arrow line.

3. (Root correspondance)

   For each node v, there must be another node w whose definition node is a root node in G, such that either v is w itself or v is connected to w via component edges.

4. (Data type correspondance)

   Each node v whose definition node is a leaf entity node V with data type $Type$ in G must be associated with a data of the matching data type $Type$.

5. (Cardinal correspondance)

   For each compononent edge e with tag T connecting u to v, if the tag associated with e's definition edge is not suffixed with "*", then u cannot connect to another node w with the same tag T.

Furthermore, v_1 in D is an ancestor of v_2 in D if the definition node of v_1 is an ancestor of the definition node of v_2. The definition of descendant, parent and child can be defined similarily.

Finally, we say that the semi-structured data graph D is an *image* of G.

Figure 1 is a semi-structured data graph which is an image of the S3-Graph in Figure 2. This design is not a good design because many anomalies occurs. For example, each tutor has one name, but this information may appear many times in a semi-structured data graph as each tutor may teach many students. For example, the name "Tan" is repeated twice in the semi-structured data graph in Figure 1. Now, we encounter some **anomalies**. If a female tutor needs to change her surname after marriage, we must make sure that all the appearances of this information are consistent. This is the **rewriting anomaly**. Similarly, if the course is not mounted, the information about the tutor may be deleted together with its parent. This is the **deletion anomaly**.

Note that due to the flexibility of semi-structured data, there is no **insertion anomaly**. To insert a tutor and his name, it is possible to insert a new tree that indicates the above information.

In a semi-structured data graph, an object instance can be connected to multiple occurrences of objects of the same role. This introduces other types of anomalies that do not happen in relation in RDBMS that permits only atomic-valued attributes. One of the anomalies can be illustrated as follows.

Refer to Figure 1 again, a tutor can have a set of offices. Since a tutor may appear more than once in the graph, the information "Tan has an office at S16 05-13", which is *independent* of the Student and Course the tutor teaches, is repeated twice in the graph. This introduces anomaly. We refer to such anomaly as **set anomaly**.

3 A Normal Form for Semi-Structured Schema Graph (S3-NF)

In order to remove the anomalies that may exist in a given semi-structured data, we define a normal form for it. In this section, the concept of SS-dependency and this new normal form, called S3-NF, is described.

3.1 SS-Dependency

Definition 4. Given a S3-Graph G, and let $A = \ll A_1, \cdots, A_m \gg$ be a sequence of roles in G. The sequence A is called *hierarchical role sequence* if A_i is an ancestor of A_j whenever $j > i$.

Definition 5. Given a S3-graph G, and a semi-structured data graph D which is an image of G. Let $A = \ll A_1, \cdots, A_m \gg$ be an hierarchical role sequence in G such that A_i is an ancestor of A_j whenever $j > i$. An *instance* of A wrt G in D is a sequence of nodes, $\ll a_1, \cdots, a_m \gg$ in D such that a_i is of role A_i and a_i is the ancestor of a_j whenever $j > i$.

Definition 6. Given a S3-graph G, and a semi-structured data graph D which is an image of G, let $e = \ll a_1, \cdots, a_m \gg$ and $e' = \ll a'_1, \cdots, a'_m \gg$ be two instances in D of an hierarchical role sequence $A = \ll A_1, \cdots, A_m \gg$ in G, e *agrees* e' if and only if for every corresponding node a_i in e and a'_i in e', we have

1. If a_i and a'_i are atomic data, then they have the same value.
2. If a_i and a'_i are objects, then they represent the same object. [1]
3. If a_i and a'_i are references to another objects, then the two object instances referenced by a_i and a'_i are the same object.

Example 3. Refer to Figure 3, the instance $\ll \&8' \gg$ agrees with the instance $\ll \&8'' \gg$. as both of them reference the same object represented in &8. On the other hand, the instance $\ll \&3, \&5 \gg$ does not agree with the instance $\ll \&4, \&10 \gg$ as their course codes are different.

Definition 7. With respect to a S3-Graph G, for a hierarchical role sequence $\ll A_1, \cdots, A_m \gg$, where A_i is an ancestor of A_j whenever $i < j$, and a single entity type B where B is a descendent of A_m, we have $A_1, \cdots, A_m$ SS-determines B, denoted as

$$A_1, \cdots, A_m \Longrightarrow B$$

if in any semi-structured data graph D which is an image of G, whenever any two different instances e_1 and e_2 of A in D agree, it implies that the set of instances in D of role B having e_1 as an ancestor is the same as that of having e_2 as an ancestor.

[1] Deciding if two objects are the same depends on the underlying database model. In general, it can be decided by at least two ways: same key value, and same object-id.

Example 4. Refer to the S3-Graph in Figure 2, we have $Tutor \Longrightarrow Office$. The set of offices that a tutor has solely dependent on the tutor, and is not dependent on the courses he/she teaches, nor the students he/she has. This SS-dependency can be illustrated by one of its images as shown in Figure 1: The two instances "$\ll$ &8 $\gg$" and "$\ll$ &13 $\gg$" agree as they represent the same tutor "Tan". Their office data also agree as they both hold the same value "S16 05-13".

On the other hand, $Tutor \not\Longrightarrow Feedback$. Refer to Figure 1, although the instance "$\ll$ &8 $\gg$" agrees with "$\ll$ &13 $\gg$", their descendent instances of role Feedback, "$\ll$ &16 $\gg$" and "$\ll$ &23 $\gg$", do not agree. For this database, the correct SS-dependency should be $Student, Course, Tutor \Longrightarrow Feedback$.

Theorem 1. *Let G be a S3-Graph, A and B be two hierarchical role sequences such that any of the roles of A is an ancestor of each role in B, We have the following properties for SS-dependency:*

1. *(reflexivity) For any A, $A \Longrightarrow A$.*
2. *(generalization of functional dependency) if $A \longrightarrow B$, then $A \Longrightarrow B$.*
3. *(left augmentation) Let C be a role that is an ancestor of each role in B and $A \Longrightarrow B$, then $AC \Longrightarrow B$ where AC represents the hierarchical role sequence containing all the roles in A and C,*
4. *(right augmentation) Let C be a role that is a descendent of each role in B and $A \Longrightarrow B$, then $A \Longrightarrow BC$. where BC represents the hierarchical role sequence containing all the roles in B and C,*
5. *(transitivity) For any three hierarchical role sequences A, B, C, if $A \Longrightarrow B$ and $B \Longrightarrow C$, then $A \Longrightarrow C$.*

Proof: *The proof of these properties follows directly from the definitions of SS-dependency and functional dependency.*

Definition 8. Let A and B be two hierarchical role sequences, If there exists a hierarchical role sequence C such that

$A \Longrightarrow B$ and
$B \Longrightarrow C$ and
$B \not\longrightarrow A$

then we say that C is *transitively SS-dependent* on A via B.

Example 5. Refer to Figure 2, Code, are transitively SS-dependent on Student via Course since

$Student \Longrightarrow Course$
$Course \Longrightarrow Code$
$Course \not\longrightarrow Student$

Theorem 2. *Given a S3-Graph G, if a role C in G is transitively SS-dependent on another role A via role B, then there exists a semi-structured data graph D which is an image of G such that the rewriting anomaly occurs upon updating the data of role C.*

Proof: *We can build a semi-structured data graph as follows,*

1. *We first construct two instances a_1 and a_2 of role A such that they do not agree.*
2. *we construct the descendents of a_1 and a_2 such that they are of role B and represent an identical object. Since $B \not\longmapsto A$, it does not contradict the assumption that a_1 and a_2 does not agree.*
3. *Since $B \Longrightarrow C$, for the set of descendents of b_1 which plays the role C, it must be the same as the set of descendents of b_2 with the role C.*

We have now constructed a semi-structured data graph. If we update the information of C under b_1, it can cause inconsistency unless updating is also done at the same time to the information of C under b_2.

Hence, there is a semi-structured data graph D, which is an image of the given S3-Graph G, such that the rewriting anomaly occurs when we update C.

3.2 S3-NF and decomposition of a S3-Graph

Definition 9. A S3-Graph G is said to be in S3-NF if there is no transitive SS-dependency in the graph.

In order to restructure a S3-graph to reduce redundancy, we need to remove any transitive SS-dependency in a given S3-Graph. If this can be done, then the schema will be in S3-NF. In this paper, we adopt the decomposition approach to remove transitive dependencies. However, as in the case of relational database, decomposition approach by no means ensures a good solution. Integrity constraint information can be lost during the decomposition. Indeed, as mentioned in [12], it is not always possible to remove every transitive dependency in a nested relation solely by decomposition. As semi-structured data is even more flexible than nested relation, our decomposition method can only transform the schema to reduce redundancy, but may not always remove all transitive dependencies and achieve S3-NF. In future research, we will purpose another synthesis method similar to Bernstein [2] and Ling's [11] method to generate a S3-NF scheme and at the same time, guarantee that no constraint information is lost.

The basic operation of our restructuring is to introduce new reference nodes and decompose the given schema graph. The main goal is to remove transitive SS-dependency in the graph. This can be done by the following step:

Given a S3-Graph G, we can decompose it to to reduce redundancy.

1. For each role B, if there does not exist a role set A such that
   (a) $A \Longrightarrow B$ and
   (b) $B \not\longmapsto A$,
   skip the rest of the following steps and continue to check for another role.
2. Let C_j be the set of the children of B such that
   (a) $B \Longrightarrow C_j$,
   (b) for every descendent of C_j which is of role D, we have $B \Longrightarrow D$.
   If there is no such C_j, then skip the rest of the following steps and continue to check for another entity type.

3. (Graph decomposition) Otherwise, duplicate the node V which has the role B to form a new node V'. It will be the root of a new tree. Move each of the C_j and all the descendents of C_j and their corresponding edges under V'. Now, replace the original node V by a reference node. This reference node shall reference V'. The tag of the referencing edge will be B.

Example 6. Refer to the schema represented in the S3-Graph in Figure 2, we want to restructure the schema to reduce redundancies.

1. We first inspect the role *Student*. Since there is no other role A such that $A \Longrightarrow Student$, no redundancy will be caused by the *Student* role.
2. The next role is *Name*. We have $Student \Longrightarrow Name$, but *Name* has no descendent. Our algorithm skips this role and checks for other roles.
3. For the role *Course*, we have $Student \Longrightarrow Course$. Consider the node #3 in Figure 2, it has four children: #4, #5, #6 and #7. For #4 which represents *Code*, since $Course \Longrightarrow Code$, hence, *Code* is one of the C_j. Similarly, *Title* is another C_j as $Course \Longrightarrow Title$. Note that *Grade* and *Tutor* are not in C_j as $Course \not\Longrightarrow Grade$ and $Course \not\Longrightarrow Tutor$. We now decompose this graph. The subtree $Course, Code$ and *Title* are disconnected from *Student*. The original node #3 is renamed as #3'. It now becomes a reference node which references the entity node *Course* (node #3).
4. Similarly, when we inspect the role *Tutor*, we find out that
   (a) We have $Student, Course \Longrightarrow Tutor$, but $Tutor \not\longrightarrow Student, Course$.
   (b) As $Tutor \Longrightarrow Tutor.Name$, $Tutor.Name$ is one of the C_j which can cause anomaly.
   (c) $Office$ is also in C_j as firstly, $Tutor \Longrightarrow Office$, Furthermore, for the two children of $Office$: *Building* and *Room*, our algorithm needs to verify that the set of *Room* and *Building* that a tutor has can be solely dependent on *Tutor* only, and does not require extra information such as *Course* and *Student*.
   (d) Finally, *Feedback* is not one of the C_j since $Tutor \not\Longrightarrow Feedback$.

   Another decomposition is done to duplicate node #7.
5. No other remaining role requires further decomposition. Our restructuring step stops.

The final restructured S3-Graph is shown in Figure 4. It is also in S3-NF.

4 ER Approach to Semi-Structured Database Design

The task of designing "good" semi-structured database can be made easier if we have more semantics. [9] proposed a normal form for the ER model. Using this ER normal form, [10] can give a good design for nested relations. Using this idea, we can also make use of the ER normal form to design good semi-structured database. This top-down approach [10], which consists of normalizing an ER diagram and converting a normalized ER diagram into the semi-structured database has two advantages:

1. Normalizing an ER diagram effectively removes ambiguities, anomalies and redundancies at a semantic level.
2. Converting a normalized ER diagram into semi-structured database results in a database schema with clean semantics.

In this section, we will discuss breifly how we can use the ER approach to design good semi-structured databases.

The ER approach uses the concepts of entity types and relationship sets to capture real world semantics. An entity type or relationship set has attributes which represents its structural properties. Attributes can be single-valued or multivalued. Figure 5 shows the ER diagram for our Student-Course-Tutor example. We see that students, courses and tutors are modeled as entity types Student, Course and Tutor respectively. Student has an attribute Name while Course has attributes Code and Title. Tutor has a single-valued attribute Name and a composite multivalued attribute Office. Here, we assume that Student.Name, Course.Code and Tutor.Name are the identifiers of the entity types Student, Course and Tutor respectively. The relationship set Enrol captures the association that a student is enrolled in a course and has a single-valued attribute Grade. Since a student taking a course is taught by some tutors, we need to associate the relationship set Enrol with the entity type Tutor. This is accomplished using aggregation which effectively allows us to view Enrol as an entity type for the purpose of participation in other relationship sets. This association is captured in the relationship set SCT. Feedback is a single-valued attribute in SCT as its value is given by the student for each tutor teaching him in some course. It is clear that the ER diagram is also in normal form [9].

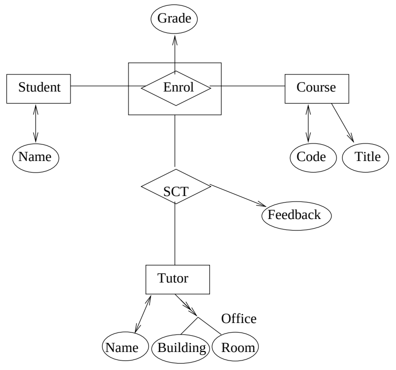

Fig. 5. Entity-Relationship Diagram for Student-Course-Tutor Example

We now outline the translation of the normal form ER diagram into a S3-graph. Details of the translation algorithm will be given in a full paper.

1. Each entity type E becomes an entity node N with role E. Each attribute A of E is a node which is connected to E by a component edge with tag A.

2. For each n-ary relationship set R, we first construct a path to link the participating entity types of R. Let $\prec V_1, V_2, \cdots, V_k \succ$ be the path. Vertex V_1 corresponds to some participating entity type of R which is associated with some entity node N_1. Each vertex V_i, where $2 \leq i \leq n$, corresponds to either a participating entity type of R or a combination of two or more participating entity types of R. We next create reference nodes $N_2, \cdots, N_k$ that is associated with $V_2, \cdots, V_k$ respectively. Then we have a component edge from N_i to N_{i+1}, where $1 \leq i \leq k-1$. Each reference node N_i, where $2 \leq i \leq k$, also has referencing edge(s) to the entity node(s) that is associated with the participating entity type(s) of R corresponding to V_i. Any attribute A of R is a node which is connected to N_k by a component edge with tag A. Note that relationships which are involved in aggregations have to be processed first because they will establish portions of a subsequent path.

The ER diagram in Figure 5 can be translated to the semi-structured schema graph in Figure 4 as follows. The entity types Student, Course and Tutor become entity nodes #1, #3, #7 respectively. The attributes also become nodes and are connected to their owner entity type by component edges. We need to process the relationship Enrol before SCT because Enrol is involved in an aggregation. Suppose we choose to construct the path $\prec Student, Course \succ$ from the participating entity types of Enrol, then the relationship set Enrol becomes a reference node #3', and the entity node #1 has a component edge to #3', which in turn has a referencing edge to the entity type #3. The attribute Grade is a component of the reference node #3'. The relationship set SCT also becomes a reference node #7'. The path corresponding to SCT must be $\prec Student, Course, Tutor \succ$ because Enrol is an aggregate in the relationship set SCT and it has earlier established the $\prec Student, Course \succ$ portion of the path. Node #3' has a component edge to #7' which in turn has a referencing edge to #7. The attribute Feedback is a component of the reference node #7'. We observe that the semi-structured schema graph obtained is not unique but is dependent on the path constructed.

5 Conclusion

In this paper, we have shown the importance of designing good semi-structured databases. We defined a semi-structured schema graph called S3-graph for semi-structured databases. We identified various anomalies, including rewriting anomaly, deletion anomaly and set anomaly, that may arise if a semi-structured database is not designed properly and contains redundancies. We proposed a normal form for semi-structured schema graph, S3-NF. We present two approaches to design good semi-structured databases, namely, the restructuring approach and the ER approach. The former uses the decomposition technique to normalize a semi-structured schema graph which may not guarantee a good solution while the latter uses the normal form ER model to obtain a normal form semi-structured schema graph. Our definition of the semi-structured schema graph attempts to correspond to the XML definition so that we can apply our technique to design good XML databases in future.

References

1. S. Abiteboul, D. Quass, J. Widom, and J. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1), 1997.
2. P.A. Bernstein. Synthesizing third normal form relations form functional dependencies. *ACM Transactions on Database Systems*, 4(1):277–298, 1976.
3. T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (xml) 1.0. W3C Recommendation available at http://www.w3.org/TR/1998, 1998.
4. P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to semistructured data. In *Int. Conference on Database Theory*, 1997.
5. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. ACM SIGMOD*, 1996.
6. E.F. Codd. Further normalization of the database relational model. Database Systems, edited by Randell Rustin, 1972.
7. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3), 1997.
8. R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proc. of the 23rd VLDB*, 1997.
9. T.W. Ling. A normal form for entity-relationship diagrams. In *Proc. of 4th Int. Conference on Entity-Relationship Approach*, pages 24–35, 1985.
10. T.W. Ling. A normal form for sets of not-necessarily normalized relations. In *Proc. of 22nd Hawaii Int. Conference on Systems Science*, pages 578–586, 1989.
11. T.W. Ling, F.W. Tompa, and T. Kameda. An improved third normal form for relational databases. *ACM Transactions on Database Systems*, 2(6):329–346, 1981.
12. T.W. Ling and L.L. Yan. Nf-nr: A practical normal form for nested relations. *Journal of Systems Integration*, 4:309–340, 1994.
13. D. Maier. Theory of relational databases. Pitman, 1983.
14. A. Makinouchi. A consideration on normal form of not-necessarily normalized relation in the relational data model. In *Proc. of 3rd VLDB*, 1977.
15. J. McHugh, S. Abiteboul, R. Goldman, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), 1997.
16. S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Objects: Concise representation of semistructured hierarchical data. In *Proc. of the 13th Int. Conference on Data Engineering*, 1997.
17. Z.M. Ozsoyoglu and L.Y. Yuan. A normal form for nested relations. *ACM Transactions on Database Systems*, 1(12):111–136, 1987.
18. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, pages 251–260, 1995.
19. M.A. Roth and H.F. Korth. The design of 1nf relational databases into nested normal form. In *Proc. of ACM SIGMOD*, 1987.
20. J.D. Ullman. Principles of database systems. Computer Science Press, 1983.
21. K. Wang and H.Q. Liu. Schema discovery from semistructured data. In *Int. Conference on Knowledge Discovery and Data Mining*, 1997.

This article was processed using the LaTeX macro package with LLNCS style