Schemaless Semistructured Data Revisited
—Reinventing Peter Buneman's Deterministic Semistructured Data Model—

Keishi Tajima

Kyoto University, Yoshida-Honmachi, Sakyo, Kyoto 603-8501 Japan
tajima@i.kyoto-u.ac.jp

Abstract. This paper reviews the design of data models for semistructured data, particularly focusing on their schemaless nature. Uniform treatment of schema information and data, in other words, uniform treatment of metadata and data, is important in the design of such data models. This paper discusses what data and metadata are, and argues that attribute names, which are usually regarded as metadata, and key values, which are usually regarded as data, play similar roles when we organize large data sets. The paper revises one of the standard semistructured data models in accordance with that argument, and eventually reinvents the deterministic semistructured data model proposed by Peter Buneman and his colleagues. The contribution of this paper is an additional rationale of the design of that data model, a rationale based on the similarity between attribute names and key values.

Keywords: semistructured, schemaless, self-describing, metadata, attribute name, key value, edge label, graph, table, multidimensional table

1 Introduction

In the 1990s, data with nested irregular structure but without predefined schema became prevalent, and the management of such *semistructured* data became an important research topic in the database community [1, 4]. Data models and query languages for semistructured data were first discussed [14, 8, 15, 6], and they were soon followed by research on all other aspects of semistructured data management. After that, the focus of the research shifted to the management of XML data, which to some extent represents the convergence of semistructured data management and document management [2]. The XML data format, however, was not originally designed for semistructured data, and is not necessarily appropriate for such data. The design of data models for semistructured data, therefore, remains a technically interesting problem.

This paper reviews the data models for semistructured data that were proposed and studied in the 1990s. It particularly focuses on the *schemaless* nature of semistructured data. The term "schemaless" is related to a couple of aspects of semistructured data, but what is of interest here is the uniform treatment of schema information and data; in other words, the uniform treatment of *metadata* and data. This paper therefore first discusses what data and metadata are, and argues that *attribute names*, which are usually regarded as metadata, and *key values*, which are usually regarded as data, play similar roles when we organize large data sets. Revising one of the standard semistructured data models proposed in the 1990s in accordance with the argument, we see that the result is the *deterministic semistructured data model* proposed by Peter Buneman and his colleagues in [9]. In other words, this paper reinvents that data model.

Much of the discussion in this paper has already been shown in the literature, such as [1, 4, 2] and, of course, [9]. This paper also includes many things that were not explicitly written in [9] but must have already been discussed by Peter Buneman and his colleagues. It may also even include something the author first heard from Peter Buneman or his colleagues but then forgot. The contribution of this paper is that it clarifies the similarity between attribute names and key values, thereby providing an additional rationale for the design of the deterministic semistructured data model in [9].

The remainder of this paper is organized as follows. The next section reviews key issues in the design of data models for schemaless semistructured data, and briefly explains one of the semistructured data models proposed in the 1990s. Section 3 discusses what data and metadata are, and shows that attribute names and key values play similar roles in indexing data items in large data sets. Section 4 extends the model previously explained in Section 2 in accordance with the discussion in Section 3, and "reinvents" the deterministic semistructured data model. Section 5 briefly discusses the relation between that data model and table-based data models, and Section 6 concludes the paper.

2 Semistructured Data Models

Two major data models for semistructured data were proposed in the 1990s, one by the Stanford University Database Group (which is now the Stanford University InfoLab) [3] and the other by Peter Buneman and his colleagues at the University of Pennsylvania [6].

One of the most important properties of these semistructured data models is that they are schemaless. That is, in these data models, data are not accompanied by a separate predefined schema that describes the structure of the data. In ordinary data models for database systems, e.g., in the relational data model, a schema mainly plays the following roles:

1. The most important role of a schema is to index and annotate each data item in the database. For example, attribute definitions in a relational schema are used by the system for parsing the tuples, and attribute names let users know the meaning of the attributes. Such data that describes the structure or meaning of another data is sometimes called *metadata*.
2. A schema also represents structural constraints on data. For example, attribute specifications in a relational schema work also as constraints on the structure and contents of tuples.

3. A schema as a whole also works as a data catalogue (i.e., a concise summary of the stored data) for users to browse or query the database.

In semistructured data models, however, the existence of a schema is not assumed [1, 4, 2] because

– data may have irregular structure, making it hard to define a compact schema,
– data may have a schema that changes frequently, and
– the information on the structure of data may not be available in advance when we start to store partial data.

When we do not assume a schema, we need some substitutes to play the roles explained above. For the roles of constraints and data catalogs (i.e., Items 2 and 3 above), graph schemas [5] and DataGuide [11] have been proposed. Graph schemas represent constraints on the structure of graph data in a looser way than ordinary rigid schemas (e.g., relational schemas in the relational data model). DataGuide is a summary of a database that is created a posteriori from the stored data. It can be used as a data catalog for users, and can also be used as data constraints in query optimization.

For indexing and annotation (i.e., Item 1 above), most existing semistructured data models take the same approach: they embed such metadata within the data itself. That is why semistructured data are sometimes called *self-describing*. This paper takes the same approach. When this approach is taken, how to embed metadata in data becomes the key issue in the design of data models.

The two data models proposed by the Stanford group and the UPenn group also embed metadata in data. Both data models are essentially labeled graphs. Graphs are used to represent nested irregular data structure, and both basic data values and metadata are represented by labels on nodes or edges. As a result of this, the distinction of data and metadata becomes unclear. In a semistructured data model, such a uniform representation of data and metadata has the following advantages [1, 4, 2]:

– Because the structure of semistructured data can be irregular and/or dynamic, we often want to query schema information as well as data. If both data and metadata are represented in a uniform way, we can use the same querying functions for both of them.
– In some applications, updates to schema information are also as frequent as updates to data. In such applications, if both data and metadata are represented in a uniform way, we can use the same updating functions for both of them.

In the data model proposed in [6], data and metadata are represented in an extremely uniform way: metadata are represented by edge labels, and atomic data are also represented by labels of terminating edges (i.e., edges to nodes without further outgoing edges).

Figure 1 shows an example graph in this data model, which represents a part of a movie database [6]. The root node at the top of the figure is the entry point of

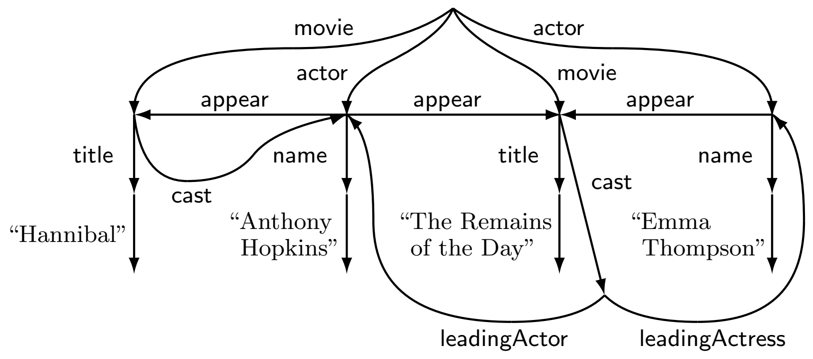

Fig. 1. An example data in the data model proposed in [6]

the database, and it has references to all entries of movies and actors/actresses. Movie entries have two attributes title and cast, and actor/actress entries have two attributes name and appear. Those attributes are represented by the edges outgoing from the node representing each entry. Because this is semistructured data, the structure inside attributes with the same attribute name can be heterogeneous. For example, the cast attribute of the movie "Hannibal" directly refers to an actor/actress entry, while the cast attribute of the movie "The Remains of the Day" leads to branching edges labeled leadingActress and leadingActor.

While attribute names, such as cast and title, are represented by edge labels, atomic values, such as the movie titles "The Remain of the Day" and "Hannibal," are also represented by edge labels. Because both attribute names and atomic values are represented by edge labels in this data model, there is no distinction between them in their query language.

Formally, the type τ of data in this data model is defined as follows [4]:

$$\begin{aligned} \tau &= set(l \times \tau) \\ \text{where}\ \ l &= int \mid string \mid \ldots \mid symbol\ . \end{aligned}$$

That is, a node in this data is a set of pairs of a label and another node. Each pair represents the label and the destination of each edge outgoing from the node. Labels can be either atomic values or symbols. Symbols are usually used as attribute names or other metadata. In this way, atomic data and metadata are modeled in a uniform way in this data model.

This model has already achieved the uniform treatment of data and metadata to some extent, but if we further pursue the interchangeability and intermingling of data and metadata, two questions arise:

– Although data and metadata are modeled in a uniform way in the data model, their usage in the example data is completely distinct. Symbols are used as metadata on internal edges, and the other atomic values are used as data on terminating edges. Are there any cases where symbols and other atomic values should be used in a more intermingled way? One example explained in [4] is the encoding of arrays in this data model, where they use integers as labels on internal edges. Is there any more?

– In this data model, any atomic values can be used in place of metadata (i.e., as edge labels). Do we need to extend this "any atomic values" to "any data"?

To answer these questions, the next section examines what data and metadata are.

3 Data vs. Metadata

To examine what data and metadata are, this paper reviews the most classic and the most popular way to organize large amount of data: tables. In this paper, the term "tables" does not mean tables in relational databases, but it means tables used in print media or Web pages.

3.1 Simple Tables vs. Multidimensional Tables

We use a variety of types of tables, but the two most popular ones for organizing large data are *simple tables* representing some entities and *multidimensional tables.* Table 1 shows an example of a simple table representing some entities. It shows nutrition facts for menu items at some hamburger shop. On the other hand, Table 2 shows an example of a multidimensional table, which represents a mileage chart of a trail near Philadelphia.

Both Table 1 and Table 2 organize cells into two-dimensional structure in order to concisely represent two contexts of each cell by its horizontal and vertical positions, but their structure is slightly different. In the multidimensional table in Table 2, rows and columns are symmetric, but in the simple table in Table 1, rows and columns have asymmetric structure. In Table 1, cells in the same column store the same type of values. For example, the cells in the column **Item** store strings, those in the column **Cal** store integer values whose unit is "Cal.", and the last three columns store boolean values. On the other hand, cells in the same row store values related to the same menu item, but not necessarily of the same type.

In computer science, particularly in the relational data model, simple tables are usually interpreted as a special kind of n-ary relations that have column names (also called attribute names). If we interpret a table in that way, a table is a set of rows corresponding to entities, a row is a tuple consisting of n components representing n attributes of the entity, and the components are indexed by column names describing the meaning of the attributes. In that interpretation, column names are metadata, and values in other cells are data. In Table 1, the last three columns have hierarchical column names, but they can be expanded to simple column names, such as "Allergen.Milk".

On the other hand, in OLAP (OnLine Analytical Processing), multidimensional tables are interpreted as multidimensional arrays (or matrices when they

Table 1. An example of a simple table

Nutrition Facts

Item	**Cal**	**Sugar**	**Fat**	···	**Allergen**		
					Milk	**Wheat**	**Egg**
Hamburger	250	5.5	9	···	-	√	√
Cheeseburger	300	6.5	12	···	√	√	√
Potato(S)	230	0.0	11	···	-	√	-
Potato(M)	380	0.0	19	···			
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gigaburger	540	8.8	29	···	-	√	√

Table 2. An example of a multidimensional table

Schuylkill River Trail Mileage Chart

	Philadelphia	**Manayunk**	**Conshohocken**	···	**Tamaqua**
Philadelphia	—	7	13	···	114.5
Manayunk	7	—	6	···	107.5
Conshohocken	13	6	—	···	101.5
⋮	⋮	⋮	⋮	⋮	⋮
Tamaqua	114.5	107.5	101.5	···	—

are two-dimensional). Arrays are usually indexed by integer values, but multidimensional tables in OLAP (also called DataCube [12]) are a special kind of arrays that are indexed by arbitrary values. For example, in Table 2, both rows and columns are indexed by place names. In this interpretation, the place names in the first row and the first column are regarded as column names and row names, which are metadata, while values in the other cells are data.

These examples show that simple tables and multidimensional tables have different structure for metadata. The distinction between them is, however, not always clear. In Table 2, rows and columns are completely symmetric, and it is quite unreasonable to interpret it as a simple table, but this is rather an extreme case. Table 3, which shows car sales data in each month and in each state in U.S., is a typical multidimensional table in OLAP (except that tables in OLAP usually have more dimensions), but this table can also be regarded as a simple table only if we add an attribute name “Month” to the first column, or if we add an attribute name “State” to the first row and transpose the table.

Similarly, Table 1 is usually interpreted as a simple table, but it can also be interpreted as a multidimensional table. In the previous interpretation, this table only has column names and does not have row names. The first column of this table, however, obviously plays a different role from those played by the other columns. The values in the first column are unique to each row, and they specify the meaning of each row, while no other column can play such a role. Therefore,

Table 3. An example of a multidimensional table in OLAP

Car Sales by Month and State

	NY	**NJ**	**PA**	⋯	**CA**
Jan 2012	233	149	183	⋯	258
Feb 2012	358	187	170	⋯	286
Mar 2012	285	174	191	⋯	225
⋮	⋮	⋮	⋮	⋮	⋮
Dec 2012	169	89	115	⋯	188

only if we regard the names of menu items in the first column, e.g., "Hamburger", as the row names, we can interpret this table as a multidimensional table.

In the relational data model, columns like the first column of this table are called "keys". Therefore, more generally speaking, we can interpret a simple table also as a multidimensional table only if we interpret its key values as its row names. Sometimes, we need more than one column to define keys. In such a case, we can produce composite row names from values of those columns, just as we did for hierarchical column names in Table 1.

Notice that we cannot clearly distinguish simple tables and multidimensional tables simply by whether all cells store the same type of values of which we can compute aggregation. Aggregation is required only for OLAP, and is not necessarily required for multidimensional tables in general. For example, it is unlikely that we want to compute any aggregation for Table 2, but this table is usually regarded as a multidimensional table. If aggregation is not required, the definition of "the same type of values" becomes ambiguous, because any value can be regarded as an instance of the type Object or the type Value.

As shown above, many tables can be interpreted either as a simple table or as a multidimensional table, and whether a given cell is data or metadata depends on how we interpret the table. If we interpret Table 1 as a simple table, the values in the first row are metadata, and the others are data. If we interpret it as a multidimensional table, the values in the first row and the first column are metadata, and the others are data. Similarly, if we interpret Table 3 as a multidimensional table, the values in the first row and the first column are metadata, while if we interpret it as a simple table, only the values in the first row or only the values in the first column are metadata.

In addition, when we have some data set, there are more than one way to organize it into a table. For example, the data in Table 3 can also be organized into a table shown in Table 4, as we actually do in ROLAP (Relational OLAP). In this representation, if we interpret this table as a simple table, "**Date**", "**State**", and "**Sales**" in the first row are metadata, and the other values are data.

The discussion above shows that we cannot uniquely determine which part of a given data set should be regarded as metadata. It depends on how we organize data. The examples above, however, demonstrate that two types of data are most likely to play a role of metadata: attribute names and key values. In addition,

Table 4. Relational encoding of a multidimensional table in ROLAP

Car Sales by Month and State

Date	**State**	**Sales**
Jan 2010	**NY**	233
Jan 2010	**NJ**	149
Jan 2010	**PA**	183
⋮	⋮	⋮
Jan 2010	**CA**	258
Feb 2010	**NY**	358
Feb 2010	**NJ**	187
Feb 2010	**PA**	170
⋮	⋮	⋮

there are advantages of organizing data as multidimensional tables, in other words, advantages of interpreting key values as metadata, as explained in the next subsection.

3.2 Advantages of Multidimensional Tables

We first compare the expressive power of simple tables and multidimensional tables. Both can represent information on some entities as shown in Table 1, which can be interpreted either as a simple table or as a multidimensional table.

Multidimensional data can also be represented either by a multidimensional table or by relational encoding as shown in Table 3 and Table 4. One difference between these two representations of multidimensional data is their space efficiency. For dense data, relational encoding is space-inefficient because the number of the repetition of the same values grows exponentially when the data has many dimensions. On the other hand, for sparse data, relational encoding can be more space-efficient if we omit rows corresponding to cases where we do not have data.

Exactly the same discussion also holds for information on many-to-many relationships. It can be represented either by multidimensional tables or by relational encoding, and their space efficiency depends on the data.

As shown above, the two types of tables have similar expressive power. Next, we compare their intuitiveness and easiness to understand. One advantage of relations is that they can always represent data in a flat two-dimensional structure. On the other hand, one advantage of multidimensional tables is that they are more intuitive when the data really have multidimensional nature. For example, Table 3 is easier to read for human readers than Table 4 is. More importantly, the interpretation of tables as multidimensional arrays is more intuitive than the interpretation as a relation even when tables represent information on some entities. For example, suppose we read out Table 1. If we interpret this table as a relation, we must read it out as:

> *"there is a row where the item is Hamburger, the calorie is 250,*
> *the sugar is 5.5, . . . , and the egg is true,*
> ⋮
> *and,*
> *there is a row where the item is Gigaburger, the calorie is 540,*
> *the sugar is 8.8, . . . , and the egg is true.*

On the other hand, if we interpret this table as a matrix with column names and row names, we must read this table out as:

> *"the calorie of Hamburger is 250, the sugar of Hamburger is 5.5,*
> *. . . , and the egg of Hamburger is true,*
> ⋮
> *and,*
> *the calorie of Gigaburger is 540, the sugar of Gigaburger is 8.8,*
> *. . . , and the egg of Gigaburger is true.*

For ordinary users who are not familiar with (and not biased toward) the relational data model, the latter description must be more natural.

Another advantage of multidimensional tables is their symmetricity. When we interpret Table 1 as a matrix indexed by column names (attribute names) and row names (key values), rows and columns have symmetric structure, and we do not need to distinguish attribute names and key values in the query languages. It means multidimensional tables treat data and metadata more interchangeably than relations. Such symmetricity of rows and columns is also useful when we interactively manipulate tables through some graphical user interface [16].

We should review why the relational data model adopted relations to represent data. In database systems, the set of attributes to be stored is usually static, while the set of entities in a database is usually changed frequently. Therefore, when we consider data models for ordinary database systems, it is reasonable to interpret tables as relations with static set of columns and dynamic set of rows. When we consider semistructured data, however, we do not assume that the set of attributes for entities is static, as explained before. Another asymmetricity of columns and rows in the relational data model is that cells in the same column store the same type of values, while cells in the same row may not. In semistructured data, however, we do not assume such regularity, either. Yet another, actually the most important, reason of the proposal of the relational data model is data independence [10]. This issue will be discussed later in the next section.

The discussion above is summarized as follows: where semistructured data is concerned, interpreting key values as metadata achieves more uniform treatment of data and metadata, and also achieves more intuitive representation of data. This conclusion leads to the design of the data model in the next section.

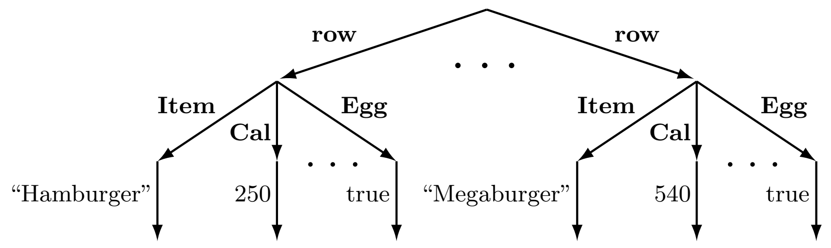

Fig. 2. Ordinary graph representation of a simple table

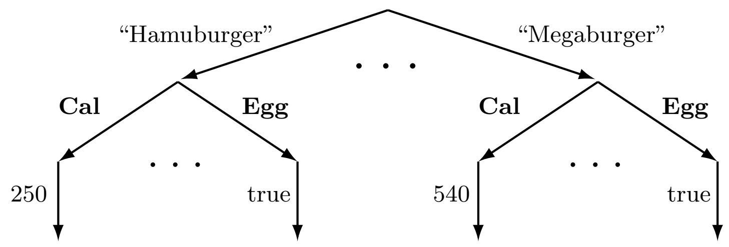

Fig. 3. Graph representation of a simple table using key values as labels

4 Deterministic Semistructured Data Model

This section gets back to the following two questions explained before:

- In edge-labeled graph models for semistructured data, are there cases where it is useful to use symbols and other atomic values in more intermingled way?
- Do we need to extend the edge-labeled graph models so that we allow any values including non-atomic values to be used as edge labels?

4.1 Symbols vs. Atomic Values

The answer to the first question is obvious from the discussion in the previous section. We should use key values as edge labels as well as attribute names. For example, the data in Table 1 is usually represented in a edge-labeled graph model as shown in Fig. 2. If we use key values as edge labels, however, this data can also be represented as shown in Fig. 3. Because we can use attribute names and key values interchangeably, this data can also be represented as shown in Fig. 4. As demonstrated in these examples, both key values and attribute names are useful to index data items in a data set, and therefore, we should use both of them for indexing, i.e., as edge labels, in semistructured data.

These examples raise another question: if we use key values and attribute names as edge labels, do we need to allow multiple edges with the same label outgoing from the same node? In semistructured data models, it is preferable

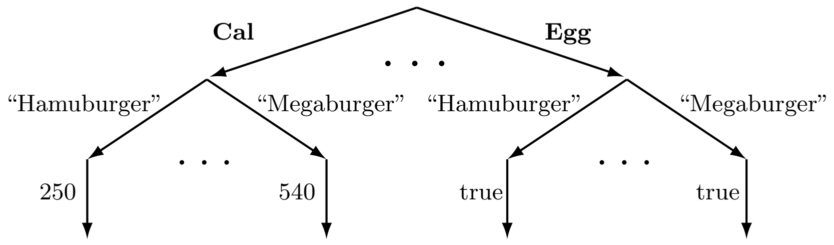

Fig. 4. Another graph representation of a simple table using key values as labels

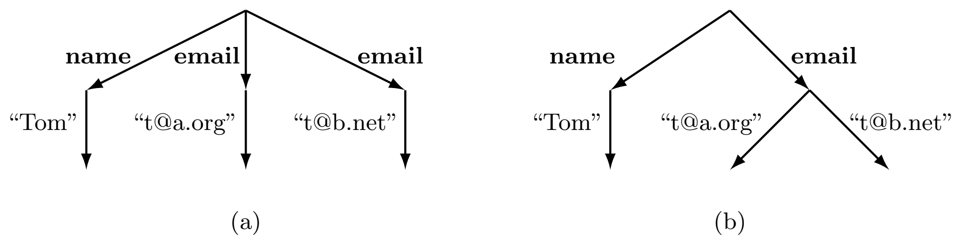

Fig. 5. Merging multiple edges representing a set value

not to distinguish set values and non-set values [1]. In that perspective, if we have a data shown in Fig. 5 (a), it is better represented by the graph shown in Fig. 5 (b). The advantage of the latter representation is that the value of some attribute can be extracted simply by extracting the subtree beneath the edge representing the attribute, no matter whether it is set-valued (e.g., **email** attribute in this example) or single-valued (e.g., **name** attribute in this example). In the existing data models that use the representation in Fig. 5 (a), similar uniformity is achieved by their carefully designed query languages [3, 6].

Such merging of multiple edges, however, is not always possible. For example, suppose we have data shown in Fig. 6 (a). If we merge the car edges in this data and transform it into the graph shown in Fig. 6 (b), the correspondence between IDs and colors of the cars will be lost. However, if we use key values as labels, we can represent this data by the graph shown in Fig. 6 (c).

As demonstrated in these examples, if we use key values as edge labels, we do not need to allow multiple edges with the same label outgoing from the same node, as long as we have keys everywhere [9].

4.2 Atomic Values vs. Composite Values

Next, the second question is discussed: do we need to use not only atomic values but also composite values as edge labels? One possible answer is: *"Yes, because we often have composite keys"*. However, do we really need to use composite key values as edge labels?

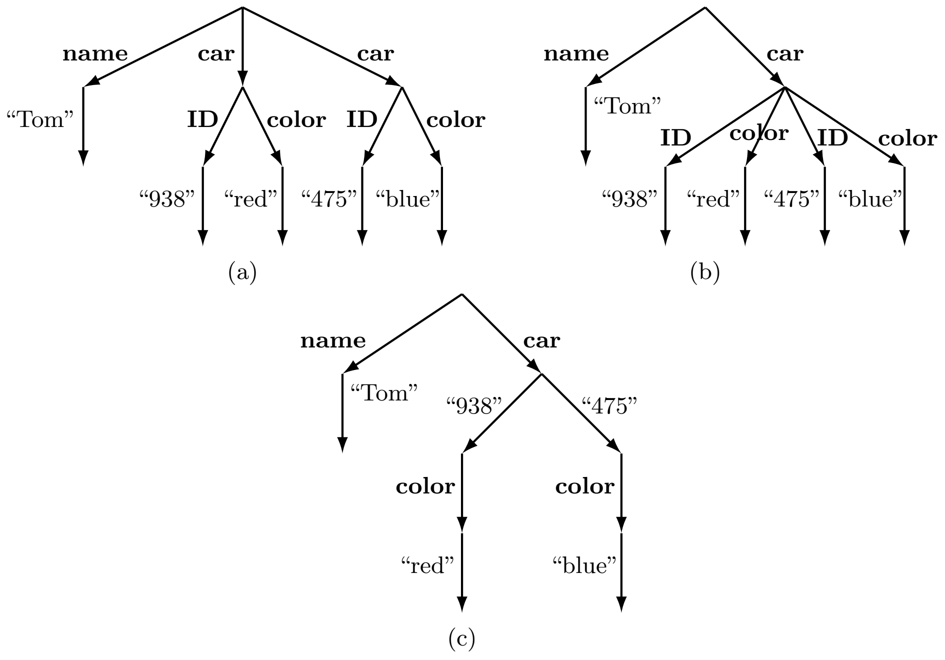

Fig. 6. Inappropriate (b) and appropriate (c) merging of edges with the same label (a)

For example, Table 5 shows information on a many-to-many relationship between students and courses, and it has a composite key consisting of **StudentID** and **CourseID**. Information in this table can be represented by a graph with hierarchical indexing structure, as shown in Fig. 7. Of course, we may also use the opposite order of **StudentID** and **CourseID**.

Such an interpretation of composite keys as hierarchical indices, however, is not always appropriate. For example, suppose there is a table that has a composite key consisting of **GivenName** and **Surname**. In this case, it does not make much sense to organize rows of this table hierarchically by first grouping them based on their **GivenName** and then grouping them based on their **Surname** (or in the opposite order), as shown in Fig. 8, because rows sharing the same **GivenName** (or the same **Surname**) are not necessarily related to each other, and a group of such rows has no useful meaning.

In addition, even in the former example of **StudentID** and **CourseID**, choosing and enforcing one specific order among these attributes causes the problem of *data independence*. That is, it enforces a specific access path on accessing programs, and if the order among the attributes is changed for some reason, we need to rewrite the programs.

The main motivation of the adoption of flat relations in the proposal of the relational data model was to achieve data independence [10]. Since one purpose of semistructured data model is to deal with data with irregular nested structure,

Table 5. Relation representing many-to-many relationship

Course Enrollment of Students

Student ID	Course ID	Grades
20091853	**I295**	80
20091853	**E108**	65
20091875	**I117**	75
20091875	**E108**	50
20101725	**I295**	70
⋮	⋮	⋮

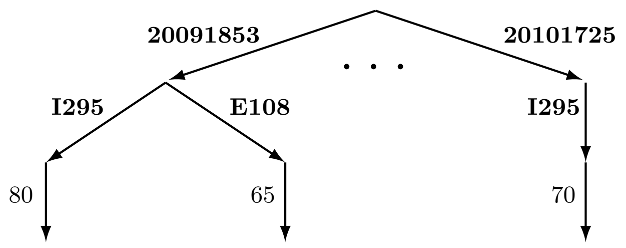

Fig. 7. Hierarchical representation of composite keys

it is difficult to fully eliminate hierarchical structure from the data model, but when hierarchical structure has no useful meaning, we should avoid it as much as possible.

There is yet another reason why we should use composite key values as edge labels: key values can be set values. When key values for some entities are set values, and each key value may have different number of elements, if we decompose these key values into hierarchical indexing structure with only atomic values on edges, each entity would have a different depth in the hierarchy, which introduces unnecessary irregularity. In addition, we need to define some canonical order among elements of a key value.

Table 6 is an example of a table whose key values are set values with different number of elements [13]. This table lists polygons, e.g., those in a CAD system, and here we assume that a polygon can be uniquely identified by specifying its set of vertices. If two polygons have the same set of vertices, they are regarded as the same polygon.

Because of these three reasons, the answer to the second question is: *"Yes, we should allow composite values as edge labels"*. Then, what kind of composite values should we allow? In the semistructured data model explained in Section 2, edge-labeled graph structure is the only data structure, and both records and sets are represented by this data structure. In addition, a key of some data may even include edge labels representing attribute names or class names, as shown in [7]. Therefore, we should allow any edge-labeled graph to be embedded as an

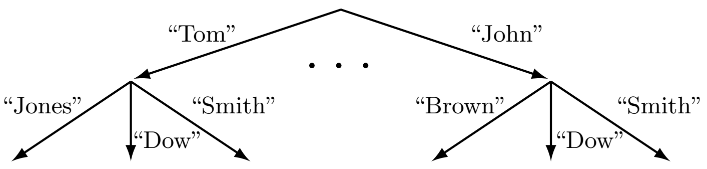

Fig. 8. Meaningless hierarchical representation of composite keys

Table 6. A table whose key values are set values

Polygons in CAD

vertices		**color**	**owner**
x	**y**		
3	5		
5	4	red	ken
4	2		
0	4		
1	6		
3	5	blue	joe
2	1		
⋮	⋮	⋮	⋮

edge label in another graph. For example, the data in Fig. 8 can be represented as shown in Fig. 9. In the graph shown in Fig. 9, edges from the root node have labels representing composite values consisting of a given name and a surname.

4.3 Definition of the New Model

In this section, the model in Section 2 has been extended in the following two ways:

- Any edge-labeled graph can be used as an edge label in another graph.
- No node can have multiple outgoing edges with the same edge label.

Accordingly, the type τ of data in the data model is redefined as follows:

$$\begin{aligned} \tau &= \mathit{set}(\tau \times \tau) \mid l \\ \text{where}\ \ l &= \mathit{int} \mid \mathit{string} \mid \ \ldots \ \mid \mathit{symbol}\ . \end{aligned}$$

That is, we now allow any value of type τ to be a label, and because we do not distinguish values and labels, any value of the type l is now regarded as a value of the type τ. Notice that l (i.e., atomic values) can also be used as the destination of an edge in this new definition.

If we emphasize the second extension, i.e., the uniqueness of labels of outgoing edges from a node, we can also define τ in the following way:

$$\begin{aligned} \tau &= \tau \rightharpoonup_{\mathit{fin}} \tau \mid l \\ \text{where}\ \ l &= \mathit{int} \mid \mathit{string} \mid \ \ldots \ \mid \mathit{symbol} \end{aligned}$$

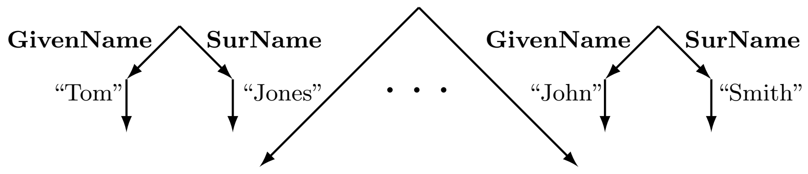

Fig. 9. Representation using composite keys as edge labels

where $\tau \rightharpoonup_{fin} \tau$ denotes a finite partial function from τ to τ.

Now, this is exactly the core part of the deterministic semistructured data model proposed by Peter Buneman and his colleagues in [9]. In this way, based on the discussion on the meanings of the term "schemaless" in the context of semistructured data model, and the discussion on what data and metadata are, this section revised the data model explained in Section 2, and reinvented the deterministic semistructured data model proposed in [9].

5 Graphs vs. Tables

The previous section reinvented the deterministic semistructured data model by starting from the graph-based data model explained in Section 2. Section 3, however, discussed data and metadata in the context of table representations of data. Then a question that may arise is: why do we use graphs rather than tables? To answer this question, this section briefly examines whether nested multidimensional tables are appropriate for representing semistructured data.

One advantage of the table-based representation of information over the graph-based representation is that it can naturally represent data indexed by combinations of two or more values. On the other hand, its disadvantage against the graph-based representation is that it is space-inefficient when the data is sparse. Because semistructured data has irregular structure, and its table representation can be sparse, we consider its relational encoding, which is more space-efficient for sparse data. In relational encoding of nested multidimensional tables, there is at most one row for each combination of indexing values. Therefore, the type τ' of such a data can be defined as a finite partial function below:

$$\begin{gathered} \tau' = (\tau' \times \cdots \times \tau') \rightharpoonup_{fin} \tau' \mid l \\ \text{where} \;\; l = int \mid string \mid \ldots \mid symbol \;\; . \end{gathered}$$

In the definition above, the components of the tuple of indexing values are τ' because we should allow composite values and set values as indexing values, as shown in the previous examples. The codomain of the function is also τ' because the contents of a cell may be a nested table.

$(\tau' \times \cdots \times \tau')$ in this definition is a product of nested relations, but we can encode them in τ'. Therefore the type τ' above is a subset of the type τ defined in the previous section. That is, the deterministic semistructured data model can be used for concisely representing sparse nested multidimensional tables.

Then the next question is why we do not flatten nested tables into flat tables, as we do in the relational data model. The answer is simple. We can normalize nested tables into flat tables only if we can have key values that are not set-valued [10]. In semistructured data models, however, we want to allow keys that take set values, as explained before.

In general, an important difference between graph-based data models and table-based data models is how to represent references to other data. Most graph-based models use some kind of IDs, and most table-based models use foreign keys. In the model proposed in [9], two edges outgoing from the same node never have the same label, and therefore, any node can be identified by the sequence of edge labels on the path from the root to the node. Such a sequence of edge labels is used to represent a reference to a node in their model. That is, their model is value-based model just like the relational data model. Advantages and disadvantages of ID-based data models and value-based data models have been discussed extensively, and it is beyond the scope of this paper.

6 Conclusion

This paper reviewed what is schemaless semistructured data, and showed that one of the most important issues in the design of a data model for schemaless semistructured data, is the uniform treatment of data and metadata. Then this paper discussed what data and metadata are in the context of various table representations of data, and concluded that data which corresponds to attribute names or key values are most useful as metadata. In accordance with that discussion, this paper extended the standard edge-labeled graph model, which resulted in a reinvention of the deterministic semistructured data model proposed by Peter Buneman and his colleagues in [9]. Finally, this paper also showed that we can also reinvent the model by starting from nested multidimensional tables.

Acknowledgement

I would like to express the most sincere appreciation to Peter Buneman, who kindly allowed me to visit the database research group of the University of Pennsylvania from August 2000 to August 2001. This visit gave me a valuable opportunity to work with him and learn much from discussions with him. He also helped me and my wife enjoy life in Philadelphia during our stay there. Those thirteen months are a memorable period in both my research career and my personal life. He has also had a great influence on me through his research and his publications. I always enjoy reading his fascinating papers.

I also would like to thank Wenfei Fan, Leonid Libkin, Wang-Chiew Tan, Val Tannen, and Limsoon Wong, editors of the Festschrift in Honor of Peter Buneman, for providing me with the opportunity to contribute to it.

References

1. Abiteboul, S.: Querying semi-structured data. In: Proc. of ICDT. LNCS, vol. 1186, pp. 1–18. Springer-Verlag (Jan 1997)
2. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann (1999)
3. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.L.: The Lorel query language for semistructured data. International Journal of Digital Libraries 1(1), 68–88 (Apr 1997)
4. Buneman, P.: Semistructured data. In: Proc. of ACM PODS. pp. 117–121 (May 1997)
5. Buneman, P., Davidson, S., Fernández, M., Suciu, D.: Adding structure to unstructured data. In: Proc. of ICDT. LNCS, vol. 1186, pp. 336–350. Springer-Verlag (Jan 1997)
6. Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization techniques for unstructured data. In: Proc. of ACM SIGMOD. pp. 505–516 (Jun 1996)
7. Buneman, P., Davidson, S.B., Fan, W., Hara, C.S., Tan, W.C.: Keys for XML. In: Proc. of International WWW Conference. pp. 201–210 (Jan 2001)
8. Buneman, P., Davidson, S.B., Suciu, D.: Programming constructs for unstructured data. In: Proc. of International Workshop on DBPL. pp. 1–12 (Sep 1995)
9. Buneman, P., Deutsch, A., Tan, W.C.: A deterministic model for semistructured data. In: Proc. of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats (in conjunction with ICDT). pp. 14–19 (Jan 1999)
10. Codd, E.F.: A relational model of data for large shared data banks. CACM 13(6), 377–387 (1970)
11. Goldman, R., Widom, J.: DataGuides: Enabling query formulation and optimization in semistructured databases. In: Proc. of VLDB. pp. 436–445 (Aug 1997)
12. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. Data Mining and Knowledge Discovery. 1(1), 29–53 (1997)
13. Makinouchi, A.: A consideration on normal form of not-necessarily-normalized relation in the relational data model. In: Proc. of VLDB. pp. 447–453 (1977)
14. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object exchange across heterogeneous information sources. In: Proc. of IEEE ICDE. pp. 251–260 (Mar 1995)
15. Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Widom, J.: Querying semistructured heterogeneous information. In: Proc. of International Conference on Deductive and Object-Oriented Database Systems (DOOD). pp. 319–344 (Dec 1995)
16. Tajima, K., Ohnishi, K.: Browsing large HTML tables on small screens. In: Proc. of ACM Symposyum on User Interface Software and Technology (UIST). pp. 259–268 (Oct 2008)