Storing and Maintaining Semistructured Data Efficiently in an Object-Relational Database

Yuanying Mo
National University of Singapore
moyuanyi@comp.nus.edu.sg

Tok Wang Ling
National University of Singapore
lingtw@comp.nus.edu.sg

Abstract

We propose to use object-relational database management systems to store and manage semi-structured data. ORA-SS (Object-Relationship-Attribute model for Semi-Structured data) [9] is used as the data model. It not only reflects the nested structure of semi-structured data, but also distinguishes between object classes and relationship types, and between attributes of object classes and attributes of relationship types. ORA-SS can specify the degree of n-ary relationship types and indicate if an attribute is an attribute of a relationship type or an attribute of an object class. Existing semi-structured data models cannot specify such information. We use these information to translate XML Schemas/DTD to ORA-SS schemas, then to object-relational databases correctly and without avoidable redundancy. The existing techniques have a lot of redundancy in storage and introduce node IDs of the tree instance which are not needed in our approach.

1. Introduction

Semi-structured data is becoming ubiquitous. The emergence of XML, which is a data format for semi-structured data, will increase the availability of semi-structured data.

Modeling semi-structured data as a graph has been the preferred approach so far. In the current data models for semi-structured data it is not possible to model the kind of information that is traditionally needed when organizing data storage, for example the degree of an n-ary relationship type and the attribute of the relationship type. In this paper, we use a richer data model for semi-structured data, ORA-SS (Object-Relationship-Attribute model for Semi-Structured data) [9]. The richer semantics of ORA-SS enables us to capture more of the real world semantics, and use them in storage organization.

ORA-SS not only reflects the nested structure of semi-structured data, but it also distinguishes between object classes and relationship types, and between attributes of object classes and attributes of relationship types. Knowing the degree of an n-ary relationship type from ORA-SS leads to more efficient storage and access the data. Such information is lacking in other existing semi-structured data models, and we use these information to design an efficient storage system for semi-structured data without avoidable redundancy. We can translate the XML documents correctly. If the update to XML documents is valid, it can be translated correctly into update to the stored database.

Currently semi-structured data is usually stored in flat files. But it is difficult to query or update. In the relational approach, such as the edge approach [11, 12, 20], the attribute approach [11, 20], universal table [11, 12, 21], normalized universal approach [11, 21] and STORED [7, 8], handling multi-valued attributes is expensive. In the storage manager approach, such as Shore [4, 20] and B-tree [4, 20], it is still inconvenient when doing the search or update. Relational DBMS, for its maturity and scalability, is a viable and promising approach for storing and querying semi-structured data. But it is not efficient in handling multi-valued attributes. We can see the details in section 4.

In this paper, we describe a storage system for XML documents in the object-relational database.

ORA-SS reflects the nested structure of semi-structured data, and multi-valued attributes are treated as repeating groups in nested relations, there is less join to retrieve the multi-valued attributes, so store ORA-SS in nested relations is better.

The rest of the paper is organized as follows. Section 2 describes the ORA-SS data model and the reason why we choose ORA-SS as our data model. In section 3, we specify the translation from ORA-SS schema diagrams to object-relational databases. Section 4 compares the storage system using ORA-SS data model with other systems using other data models that have been proposed for semi-structured data. Section 5 discusses the conclusions.

2. Data Model -- ORA-SS

In our study, we found that the efficient storage and access depend not only on the transformation methodology, but also on the expressiveness of the chosen semi-structured data model. We adopt ORA-SS because it is a semantically richer data model that has been proposed for modeling semi-structured data compared to OEM [15] or XOM [24].

There are three main concepts in the ORA-SS data model (Object-Relationship-Attribute model for Semi-Structured data) [9], they are object class, relationship type and attribute (of object class or relationship type). The ORA-SS data model distinguishes object classes, relationship types and attributes. The main advantages of ORA-SS over existing data models is its ability to express the degree of an n-ary relationship type, and distinguishing between attributes of relationship types and attributes of object classes. These semantics are essential and very important for implementing an efficient storage management system.

2.1 Object classes

An object class is similar to a set of entities in the real world, an entity type in an ER diagram, a class in an object-oriented diagram or an element in semi-structured data model.

An object class is represented as a labeled rectangle. The attributes are represented as labeled circles joined to their object class by an edge. Keys are filled circles.

2.2 Relationship types

Two object classes are connected via a relationship type. Each relationship type has a degree and participation constraints. A relationship type of degree 2 (i.e. a binary relationship type) relates two object classes. One object class is the parent and the other the child. A relationship type of degree 3 (i.e. a ternary relationship type) is a relationship type between three object classes. In a ternary relationship type, there is a binary relationship type between two object classes, and a relationship type between this binary relationship type and the other object classes.

In an ORA-SS schema diagram, relationship types are denoted by directional labeled edges. The direction of the edge is from the parent object class to the child object class. The label can be described by 4 concepts, *name, n, p, c*, where *name* denotes the name of the relationship type and is optional; *n* is an integer indicating the degree of the relationship type (*n*=2 indicates binary, *n*=3 indicates ternary, etc.), it is optional, the default value is 2; *p* is a participation constraint of the form *min*:*max* of the parent object class in the relationship type, we also use XML notations ?, *, + to represent 0:1, 0:n, 1:n, it is optional and the default value is *; and *c* is the participating constraint of the child object class, it is optional and the default value is +. We can see it clearly in Example 2.2.

An object class cannot be identified by the value of its own attributes, but has to be identified by its relationship with other object classes. Such a relationship type is called identifier-dependency relationship type, it is represented by a relationship type diamond labeled with symbol "IDD".

2.3 Attributes

Attributes represent properties. And attribute can be a property of an object class or a property of a relationship type.

Attributes are denoted by labeled circles, the label consists of *name*, [F|D: *value*]. The *name* is compulsory, and the rest of the label is optional. The letter F precedes a fixed value, while D precedes a default value. The identifiers are indicated by filled circles, while other candidate keys are a double circle with the inner circle filled. An attribute's cardinality is shown inside the attribute circle, using ?, *, + to represent 0:1, 0:n, 1:n, where the default is 1:1. An attribute can be single-valued or multi-valued. A multi-valued attribute is represented using an * or + inside the attribute circle.

The special attribute name ANY denotes an attribute of unknown or heterogeneous structure.

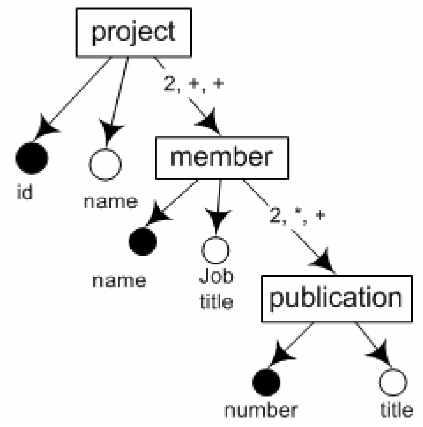
Attributes of an object class can be distinguished from attributes of a relationship type. The former has no label on its incoming edge while the latter has the name of the relationship type to which it belongs on its incoming edge.

2.4 References

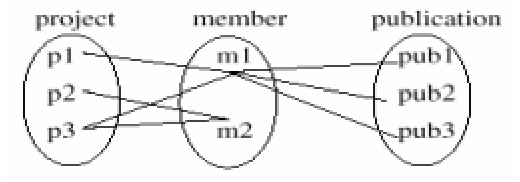
A reference depicts an object class referencing another object class, and we say a reference object class references a referenced object class. References are denoted by the dashed arrows from a referencing object class to a referenced object class. The reference and referenced object classes can have different labels and different attributes and relationship types.

Now let us see the advantages of ORA-SS model compared with the existing data models.

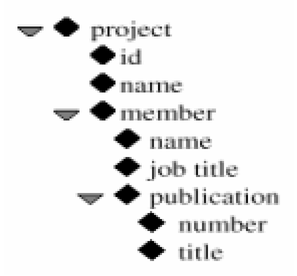
Example 2.1 Figure 2-1(a) shows a binary relationship type between *project* and *member* and a binary relationship type between *member* and *publication*. Figure 2-1(b) shows an instance of this schema with a relationship type between *projects* and *members*, and another between *members* and *publications*, but no relationship type between *projects* and *publications*. From this diagram, we can deduce that member *m1* has publications *pub1*, *pub2* and *pub3*, but we do not know which projects the publications are associated with. A Dataguide [6, 13] for this diagram is shown in Figure 2-1(c). Notice that there is no relationship type between project and publication in the ORA-SS schema diagram in Figure 2-1(a) and if the data is nested as is suggested in Figure 2-1(a), then all the publications for each member will be repeated for every project the member works on.



(a) ORA-SS Schema Diagram



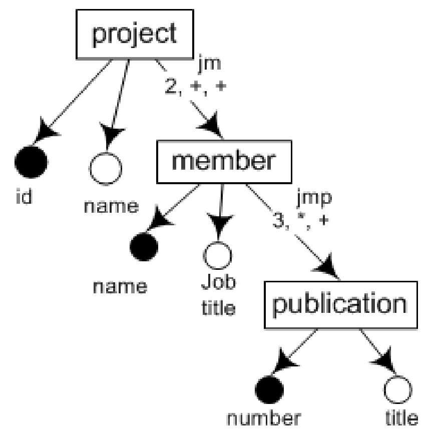
(b) Instance Relationship



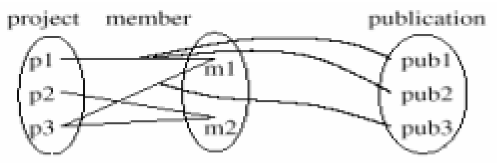
(c) Dataguide

Figure 2-1. Representing binary relationship types

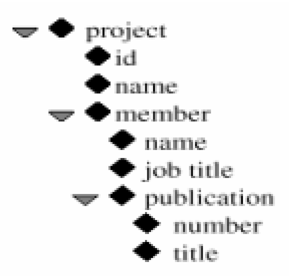
In contrast, Figure 2-2(a) shows a ternary relationship type between *project*, *member* and *publication*. There is a binary relationship type (named *jm*) between *project* and *member*, and a relationship type (named *jmp*) between *jm* and *publication*. Figure 2-2(b) shows an instance of this schema. It shows a relationship type between *project* and *member*, and another relationship type between the *project* and *member* relationship type and *publications*. From this diagram, we can deduce that publications *pub1* and *pub2* are associated with member *m1* and project *p1*. A Dataguide for this diagram is shown in Figure 2-2(c). The constraints on the relationship types in the ORA-SS diagrams in Figure 2-1(a) and Figure 2-2(a) are quite different. The schema in Figure 2-2(a) models the relationship type between papers written by a particular member while working in a particular project, and if the data is nested as is suggested in Figure 2-2(a) then only the publications written by a member while working on a project will be nested within that member and project.



(a) ORA-SS Schema Diagram



(b) Instance Relationship



(c) Dataguide

Figure 2-2. Representing ternary relationship types

The distinction between binary and ternary relationship types cannot be made in other semi-structured data models. Note that a Dataguide for the schema in Figure 2-2(c) is the same as that in Figure 2-1(c) although the constraints on the relationship types in the ORA-SS diagrams are quite different.

Example 2.2 Let us see an example of the ORA-SS schema diagram in Figure 2-3.

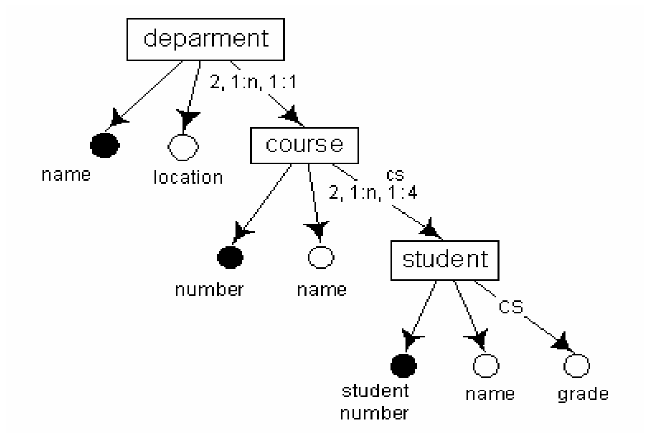


Figure 2-3. ORA-SS schema diagram of a Dept-Course-Student database

Figure 2-3 shows the schema, distinguishing object classes, relationship types and attributes, highlighting the degree of n-ary relationship types, the participation of object classes in relationship types and whether an attribute is a relationship attribute or an object attribute. The label on the edge between *department* and *course* (2, 1:n, 1:1) indicates the relationship type between *department* and *course*. There are only 3 columns, no name is assigned to this relationship type. The number 2 means that this is a binary relationship type. Each *department* must minimum offer one *course (*1:n). And each *course* belongs to one and only one *department* (1:1). The identifier of *course* is *number* (indicated by the filled circle) and *course* has another attribute *name*, that is not necessarily unique. The default cardinality of *name* is 1:1 so every *course* must have a *name*. Every *student* has a *student number* and a *name*, where the *student number* is unique. The binary many-to-many relationship type between *course* and *student* has the name *cs*. For each *course*, there can be minimum 1 *student* to many *students* (1:n). And for each *student*, he can take minimum 1 *course* and maximum 4 *courses* (1:4). The label *cs* on the edge between object class *student* and attribute *grade* indicates the attribute *grade* belongs to relationship type *cs* rather than the object class *student*.

cs is a many-to-many relationship type, and the attribute *grade* belongs to relationship type *cs*. This semantic information is helpful when repositories are being designed. Because *cs* is a many-to-many relationship type, we know that if we nest *student* within *course*, the attributes of *student* will be repeated for every *course* they take. Also, because *cs* is a many-to-many relationship type, the relationship type attribute *grade* cannot be stored in either *student* or *course* and must be stored in something that represents the relationship type between the object classes *student* and *course*.

We note that traditional semantic data models such as the Entity-Relationship model [5] cannot support XML naturally and fully. ER model is flat, it can not reflect the tree structure, while it is important in XML data. XML consists of nested element structures and the relationships of elements are modeled directly by hierarchies and reference. In contract, the Entity-Relationship model is flat and normalized.

Existing semi-structured data models, like OEM, are not possible to represent the participation constraints of object classes in relationship types, whether an attribute is an attribute of an object class or an attribute of a relationship type, and the degree of n-ary relationship types for the hierarchical semistructured data. It is best illustrated in Example 2.2.

The inadequacy of the Dataguide is its inability to express the degree of n-ary relationships for the hierarchical semistructured data, which introduces ambiguous data representations, as we have illustrated in Example 2.1.

These semantic informations are essential and very important for storage structure. The existing approaches have problems in storing semi-structured data since they cannot express this kind of semantic information.

ORA-SS has characteristics which are very similar to XML: self-describing, deeply nested or even cyclic, and irregular. It is possible to specify the participation constraints of object classes in relationship types and distinguish between attributes of relationship types and attributes of object classes. Such information is lacking in existing semi-structured data models. At the same time, the inherently hierarchical structure of ORA-SS schema diagram, the ability to model disjunction and ordering on object classes and attributes, and the ability to express the cardinality and heterogeneity of attributes makes ORA-SS the ideal model for mapping other formats to and from XML.

When a semi-structured data instance is given, the ORA-SS schema diagram can be generated using data-mining techniques, combined with the users' input. With user input, we can provide an ORA-SS schema diagram that is closer to the user expectation, and preserves the inherent semantics and implicit structures. An algorithm has developed to extract ORASS schema from XML documents, by scanning and processing the XML file, and asking necessary questions to users.

3. Translation from ORA-SS to object-relational database

In order to design an efficient and consistent organization of data in a data store, it is essential to have an algorithm that maps the logical data model to the data store. For example, the mapping algorithms from the ER data model to the relational data model. In this section, we outline an algorithm that maps ORA-SS schema diagrams to the object-relational model. Such an algorithm demonstrates how semi-structured data can efficiently and consistently be stored in a nested relational database management system like Oracle 8i or its newer version Oracle 9i.

Main rules from ORA-SS schema diagrams to object-relational databases are as follows.

1. Each object class together with its attributes forms a nested relation while multi-valued attributes as repeating groups of this relation (Object relation).
2. Each relationship type together with its attributes forms a nested relation while multi-valued attributes as repeating groups of this relation (Relationship relation).

(I) Object Class Translation Algorithm

For each object class, create a (possibly nested) relation.

O1 The identifier and candidate key of this object class is the primary key and candidate key of the generated relation.

O2 Each single-valued attribute of this object class is a single-valued attribute of the generated relation.
Composite attributes of ORA-SS diagrams are represented directly. They are replaced by their components in the generated relation. Store the value for composite attribute by listing its component attributes within parentheses.

O3 Each multi-valued attribute of this object class forms a repeating group in this relation.
Store the multi-valued attribute in the next level-repeating group.
Each multi-valued composite attribute of this object class forms a repeating group in this relation. Its components are in the repeating group.

O4 Each reference is a foreign key in this relation.

O5 Each disjunctive attribute is represented by two attributes in the generated relation. One is the flag of one bit to show which attribute the instance is related to. The other is used to store the value.

O6 For the ID dependency relationship type, the rule for the ID dependent object class is the same as the rule for the regular object class. The ID dependent object class together with its attributes forms a nested relation within its parent object class.

(II) Relationship Type Translation Algorithm

For each relationship type, create a (possibly nested) relation.

R1 The identifiers of all the object classes participating in this relationship type form the single-valued attributes of the nested relation.
The key of the relationship type can be determined by the participation constraint of the relationship type.

R2 Each single-valued attribute of this relationship type is a single-valued attribute of the generated relation.

R3 Each multi-valued attributes of this relationship type forms a repeating group in this relation.
This mapping rule is the same as in object classes. So as the other attributes of this relationship type.

R4 A disjunctive relationship type is treated as two relationship types.

R5 There is no need to translate ID dependency relationship type.

(III) Translation for Ordering

Either for the three kinds of ordering, we define another attribute named *ordinal* within the ordered object class (ie, the ordered attribute).

(IV) Translation for ANY

For the unknown structured attribute or an attribute may have a different structure for different instances, which is denoted as ANY, we define a separate table as (Identifier, ANY, ANY-value). Identifier is the identifier of the object class or the relationship type which this ANY belongs to. ANY is the different structure name (the TAG) for the different instances. ANY-value is its value.

This table will not be too long. Compared with creating the relation of the object class or the relationship type with the ANY attribute together with all the other attributes, it is more efficient and economical.

If an ORA-SS schema is in normal form [22], then the undesirable update anomalies in semistructured databases are removed and any redundancy due to many-to-many relationships and n-ary relationships are controlled. Followed these rules, the Normal Form ORA-SS schema will result in the normal form nested relations.

Example 3.1 Given an ORA-SS schema diagram as shown in Figure 3-1, the relations generated by the object class translation rule and the relationship type translation rule are shown in Figure 3-2.

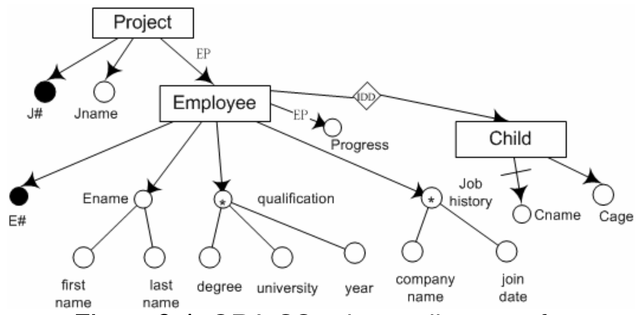
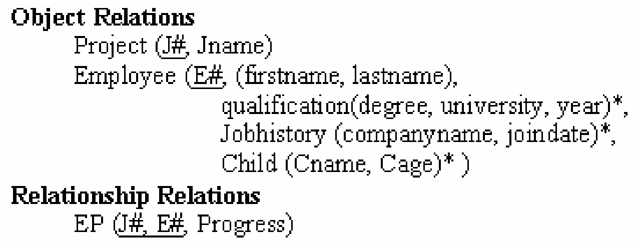


Figure 3-1. ORA-SS schema diagram of a Project-Employee database



Object Relations

Project (J#, Jname)

Employee (E#, (firstname, lastname), qualification(degree, university, year)*, Jobhistory (companyname, joindate)*, Child (Cname, Cage)*)

Relationship Relations

EP (J#, E#, Progress)

Figure 3-2. Nested relational storage for the Project-Employee database

For the object class *Project*, an object relation *Project* is created. *J#* is the primary, it is indicated underline in Figure 3-2. The relation only has a single-valued attribute *Jname*. For the object class *Employee*, an object relation *Employee* is created. *E#* is the primary key(O1). The attribute *Ename* is a composite attribute with component attributes *firstname*, and *lastname* in this relation(O2). The attribute *qualification* of *Employee* is multi-valued composite attribute, it is stored as an embedded nested relation in *Employee*, with its component *degree*, *university* and *year* in this embedded nested relation *qualification*(O3). For the object class *Child*, it is an ID dependency object class, its attributes *Cname* and *Cage* are embedded in *Employee*(O6). For the relationship relation *EP* created from relationship type *EP*, it is a binary relationship type between the object classes *Employee* and *Project*. The primary key is {*J#, E#.*}, it is determined by the participation constraints *, +, it is a many-to-many relationship type(R1). The attribute *Progress* is a single-valued attribute(R2).

We can see that in our object-relational database, accessing multi-valued attributes is easy, there is no need to do join. But for the relational database, 3 relations are needed for employee, one is the basic information of employee like employee name, one is for the information of qualification, the other is for the job history. When we want to get all the information about the employee, we should do the join operation, join these 3 relations together.

Example 3.2 In a Project-Supplier-Part database, each supplier supplies parts with fixed prices. And the database contains the information about the quantity of every part that its supplier supplies to any project. The ORA-SS schema diagram is as shown in Figure 3-3.

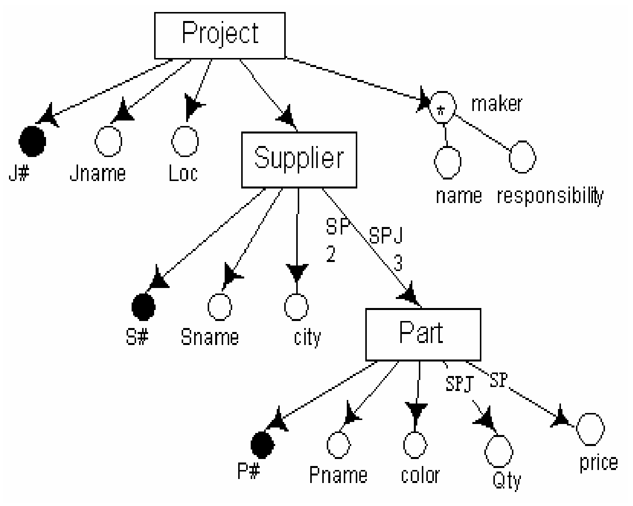
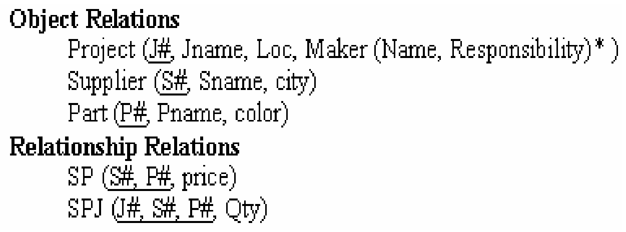


Figure 3-3. ORA-SS schema diagram for the Project-Supplier-Part database

Using the object class translation algorithm and the relationship type translation algorithm, we get the relations in Figure 3-4. The primary keys are underlined.



Object Relations

Project (J#, Jname, Loc, Maker (Name, Responsibility)*)

Supplier (S#, Sname, city)

Part (P#, Pname, color)

Relationship Relations

SP (S#, P#, price)

SPJ (J#, S#, P#, Qty)

Figure 3-4. Nested relational storage for the Project-Supplier-Part database

In the ORA-SS schema diagram, we can have 2 labels on an edge. The relationship type *SP* is a binary relationship type between object classes *Supplier* and *Part*, while *SPJ* is a ternary one among object classes *Project*, *Supplier* and *Part*. There is no participation constraints of the participating object class for these two relationship types, it means *, +, the relationship types are both many-to-many. The attribute *price* is related to the binary relationship type *SP*, while *Qty* is related to the ternary relationship type *SPJ*. Also we have the functional dependency {*S#, P#*} $\rightarrow$ *price* and {*S#, P#, J#*} $\rightarrow$ *Qty*.

Note that we have the relation SPJ (J#, S#, P#, Qty), not SPJ (J#, S#, P#, price, Qty), because the attribute *price* is only related to the relationship type *SP*, it is only dependent on *S#* and *P#*, we can see it from the ORS-SS schema diagram. It is not correct to translate *price* and *Qty* to SPJ (J#, S#, P#, price, Qty) in designing the database. In contrast to existing models, ORA-SS enables the mapping algorithm to correctly associate the attribute *price* with part and supplier in the *SP* relation, and the attribute *Qty* with project, part and supplier in the *SPJ* relation. Other existing approaches may get this result SPJ (J#, S#, P#, price, Qty), since they cannot know *price* is determined only by S# and P#, has nothing to do with J#. Without knowing this information, they will store wrongly.

4. Comparison with Related Work

The ORA-SS data model extends semi-structured data models that have been proposed in the literature. These models commonly represent semi-structured data as directed labeled graphs [1, 3, 19]. Of the models proposed, OEM (Object Exchange Model) [2, 15] is a representative example. In OEM, entities are represented by objects, each object has an object identifier and each object is either atomic or complex, i.e. the value of the object is atomic or a set of references respectively. OEM is a simple and flexible model, for representing semi-structured data. However, unlike ORA-SS data model, OEM is not possible to express some semantic information such as the degree of n-ary relationship types that is needed to design an efficient non-redundant data repository. Such information is lacking in other existing semi-structured data models also.

To apply techniques like normalization, effectively, it is necessary to know the cardinality of object classes in relationship types, the degree of n-ary relationship types and whether attributes are attributes of object classes or attributes of relationship types. In OEM, the relationship types between object classes are not specified. Like in the object-oriented data model, the inter-object references may introduce maintenance problems as a result of redundancy [14]. In contrast, in the ORA-SS data model, the relationship types between object classes and the properties of these relationship types are stated explicitly.

Approach 1.(Text File) Currently, semi-structured data is usually stored in text, such as store XML documents as ASCII files in the operating file system. But it has several major drawbacks. First, XML files in ASCII format need to be parsed every time when they are accessed for either browsing or querying. Second, the entire parsed file, which is always much larger than the original XML document, must be memory-resident during query processing. Third, it is hard to build and maintain indices on documents stored this way. Another drawback is that update operations are difficult to implement.

Let us see an example how the existing approaches store the data in Figure 4-1.

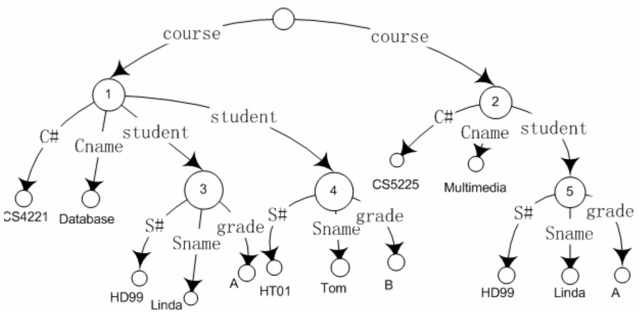
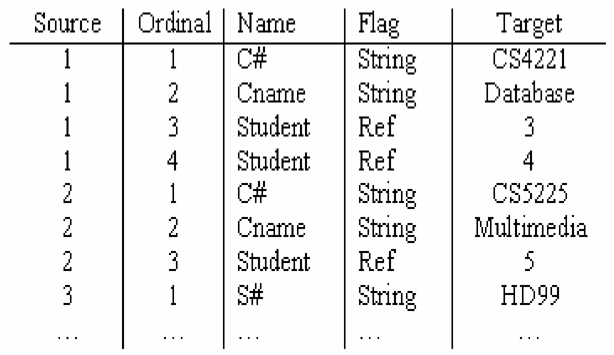


Figure 4-1. A Simple OEM database

Approach 2.(The edge table approach) It [11, 12, 20] stores all edges of the graph in a single table using relational database, as shown in Figure 4-2. It is often large, too expensive, and with much redundancy. This approach needs to have ID values for nodes which are not part of XML document. The *ordinal* number is used to record the order of any children nodes.



Source	Ordinal	Name	Flag	Target
1	1	C#	String	CS4221
1	2	Cname	String	Database
1	3	Student	Ref	3
1	4	Student	Ref	4
2	1	C#	String	CS5225
2	2	Cname	String	Multimedia
2	3	Student	Ref	5
3	1	S#	String	HD99
...	...	...	...	...

Figure 4-2. Edge Table for example in Figure 4-1

Approach 3.(The attribute approach) It [11, 20] groups all edges with the same name into one table, this approach corresponds to a horizontal partitioning of the edge table [11, 20], as shown in Figure 4-3. It is very complex to do the search and update.

AC#

Source	Ordinal	Target
1	1	CS4221
2	1	CS5225

Agrade

Source	Ordinal	Target
3	3	A
4	3	B
5	3	A

Figure 4-3. Attribute Tables for example in Figure 4-1

Approach 4.(The universal table approach) It [11, 12, 21] generates a single universal table to store all the edges. The universal table corresponds to the result of an outer join of all attribute tables [11, 20], as shown in Figure 4-4. The universal table has many fields which are set to null, and it also has a great deal of redundancy. And another problem is that we cannot know how many attributes and some attributes' name in some cases.

Source	OrdC#	TargC#	…	Ordstudent	Targstudent	OrdS#	TargS#	…
1	1	CS4221	…	3	3	null	null	…
1	1	CS4221	…	4	4	null	null	…
2	1	CS5225	…	3	5	null	null	…
3	null	null	…	null	null	1	HD99	…
…	…	…	…	…	…	…	…	…

Figure 4-4. Universal Table for example in Figure 4-1

Approach 5.(The normalized universal approach) It [11, 21] is a variant of the universal table approach [11, 21], as shown in Figure 4-5. The difference is that multi-valued attributes are stored in separate overflow tables in the normalized universal approach.

Source	OrdC#	TargC#	…	OrdS#	TargS#	…
1	1	CS4221	…	null	null	…
2	1	CS5225	…	null	null	…
3	null	null	…	1	HD99	…
…	…	…	…	…	…	…

Overflowstudent

Source	Ord	Target
1	3	3
1	4	4
2	3	5

Figure 4-5. UnivNorm and Overflow Tables for example in Figure 4-1

The above approaches (2, 3, 4, 5) all need to have ID values for nodes which are not part of XML document

Approach 6.(STORED) It [7, 8] uses OEM model and relational database management system to store and manage semi-structured data, as shown in Figure 4-6. There are many fields set to null, and it also has much redundancy.

C#	Cname	S#	Sname	grade
CS4221	Database	HD99	Linda	A
CS4221	Database	HT01	Tom	B
CS5221	Multimedia	HD99	Linda	A

Figure 4-6. STORED storage for example in Figure 4-1

Relational DBMS, for its maturity and scalability, is a viable and promising approach for storing and querying semi-structured data [3, 7, 8, 12, 18, 19]. But using relational database is not efficient for handling multi-valued attributes as retrieving of multi-valued attributes involves join operation which is expensive.

Approach 7.(Shore) In the storage manger object approach, Shore [4, 20] is used as the underlying storage system. The solution is to store each XML element of the XML file as a separate object, as shown in Figure 4-7. If the updated object increases in size when updating a object, it is complex. There are several drawbacks of this approach if the file needs to be frequently updated.

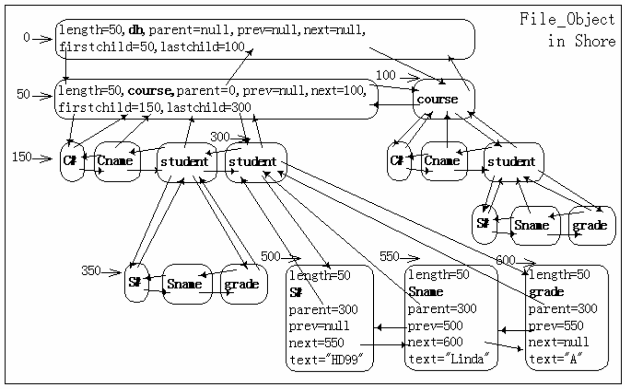

File_Object in Shore
0
length=50, db, parent=null, prev=null, next=null, firstchild=50, lastchild=100
50
length=50, course, parent=0, prev=null, next=100, firstchild=150, lastchild=300
100
course
150
C#
Cname
student
300
student
C#
Cname
student
S#
Sname
grade
350
S#
Sname
grade
500
length=50 S# parent=300 prev=null next=550 text="HD99"
550
length=50 Sname parent=300 prev=500 next=600 text="Linda"
600
length=50 grade parent=300 prev=550 next=null text="A"

Figure 4-7. Shore storage for example in Figure 4-1

Approach 8.(B-tree approach) It [4, 20], as shown in Figure 4-8, eliminates the drawback in Shore. Every object has a key. But when doing search, we do not know the key-value of the object. It is still inconvenient.

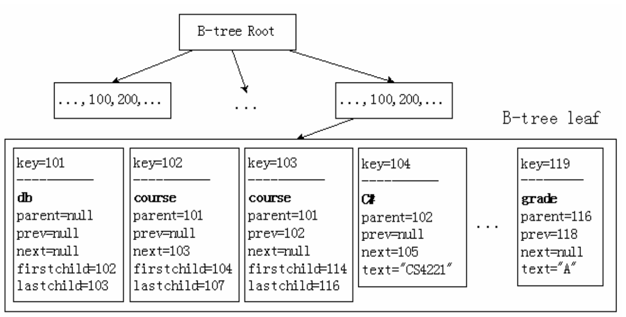
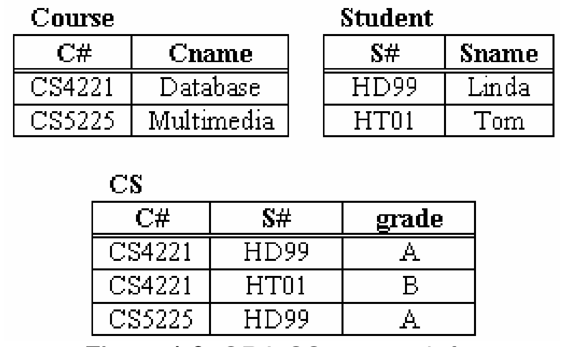


Figure 4-8. B-tree approach for example in Figure 4-1

Our approach. In contrast, our approach is to use ORA-SS as our data model and use object-relational database as the database management system. We can store and access the semi-structured data correctly, more efficient and without avoidable redundancy. There is no node ID needed in our approach.


Course

C#	Cname
CS4221	Database
CS5225	Multimedia

Student

S#	Sname
HD99	Linda
HT01	Tom

CS

C#	S#	grade
CS4221	HD99	A
CS4221	HT01	B
CS5225	HD99	A

Figure 4-9. ORA-SS approach for example in Figure 4-1

5. Conclusion

In this paper, we propose to use ORA-SS as our data model and use object-relational database management systems to store and manage semi-structured data. ORA-SS can specify the degree of n-ary relationship types and indicate if an attribute is an attribute of a relationship type or an attribute of an object class. Existing semi-structured data models cannot specify such information. We use these information to translate the XML documents to ORA-SS and then to object-relational databases correctly and without avoidable redundancy. We have presented the algorithms of storing ORA-SS in the object-relational databases, and we have presented some examples. Although the examples in this paper are quite simple, they are representative of more complicated situations that people represent using semi-structured data.

Compared with other existing approaches, such as the edge table approach, the attribute approach, the universal table approach, the normalized universal approach, STORED, Shore, B-tree approach, our approach is efficient and no removable redundancy.

In conclusion, our methodology is able to produce an efficient and non-redundant storage organization for semi-structured data.

6. Reference

[1] Serge Abiteboul. Querying semi-structured data. In Proceedings of the 5th IDCT'97, volume 1186 of Lecture Notes in Computer Science, pages 1-18. Springer, 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, "The Lorel Query Language for Semistructured Data," In Proceedings of ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, 1997.

[3] Peter Buneman. Semistructured Data. In Proceedings of the 6th ACM Symposium on Principles of Database Systems, pages 117-121. ACM Press, 1997.

[4] M. Carey, D. DeWitt, J. Naughton, M. Solomon, et. al, Shoring Up Persistent Applications, Proc. of the 1994 ACM SIGMOD Conference

[5] P.P Chen. The Entity-Relationship Approach to Logical Database Design. Q.E.D. Information Sciences, Inc., 1977

[6] Byron Choi. Lore: Lorel, DataGuide and Semistructure Databases. Available at http://www.cis.upenn.edu/~kkchoi/dbpaper2.html

[7] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data in Relations, ICDT'99

[8] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing semistructured data with STORED. In SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA, pages 431-442, 1999

[9] Gillian Dobbie, Xiaoying Wu and Tok Wang Ling and Mong Li Lee. ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. Available at http://techrep.comp.nus.edu.sg/techreports/2000/TR21-00.asp

[10] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In L. M. Haas and A. Tiwary, editors, SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, pages 414-425. ACM Press, 1998.

[11] D. Florescu, D. Kossman, A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Rapport de Recherche No. 3680 INRIA, Rocquencourt, France, May 1999

[12] Daniela Florescu, Donald Kossmann: Storing and Querying XML Data using an RDMBS. IEEE Data Engineering Bulletin 22(3): 27-34(1999)

[13] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In Proceedings of 23rd VLDB'97, pages 436-445. Morgan Kaufmann, 1997.

[14] Tok Wang Ling and Pit Koon Teo. A normal form object-oriented entity relationship diagram. In Proceedings 13th ER'94, volumn 881 of Lecture Notes in CS, pages 241-258. Springer, 1994.

[15] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore: A Database Management System for Semistructured Data," SIGMOD Record (ACM Special Interest Group on Management of Data), 26:3, pp. 54-66, 1997.

[16] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In Philip S. Yu and Arbee L. P. Chen, editors, Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, pp. 251--260. IEEE Computer Society (1995).

[17] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructure heterogeneous information. In International Conference on Deductive and Object Oriented Databases, pages 319-344, 1995.

[18] D. Suciu. An overview of semistructured data. Database Theory Column (ed V. Vianu), Sigact News, 29(4):28--38, 1998

[19] D. Suciu. Semistructured data and XML. In Proceedings of the 5th International Conference on Foundations of Data Organization and Algorithms (FODO), Kobe, Japan, November 1998.

[20] Feng Tian, David J. DeWitt, Jianjun Chen, Chun Zhang, The Design and Performance Evaluation of Alternative XML Storage Strategies, 1999

[21] Jeffrey D. Ullman. Principles of Database and Knowledge-base Systems, Volumes I, II. Computer Science Press, Rockville MD, 1989.

[22] Xiaoying Wu, Tok Wang Ling, Mong Li Lee, Gillian Dobbie. Designing Semistructured Databases Using ORA-SS Model, in Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE), PP:171-180, IEEE Computer Society, Kyoto, Japan, December 2001.

[23] D. Zhang, Y.S. Dong. A Data Model and Algebra for The Web. 10th Workshop on Database and Expert Systems Applications, pages 711-714, 1999.