



ELSEVIER

Transforming RDB schema into well-structured OODB schema

Xiuzhen Zhang^{a,*}, Yanchun Zhang^b, Joseph Fong^a, Xiaohua Jia^a

^aDepartment of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, People's Republic of China

^bDepartment of Mathematics and Computing, University of Southern Queensland, Australia

Received 30 October 1997; received in revised form 17 November 1998; accepted 24 November 1998

Abstract

When transforming relational database (RDB) schema into object-oriented database(OODB) schema, much effort was put on examining key and inclusion dependency (ID) constraints to identify class and establish inheritance and association between classes. However, in order to further remove the original data redundancy and update anomaly, multi-valued dependency (MVD) should also be examined. In this paper, we discuss class structures and define well-structured classes. Based on MVDs, a theorem is given transforming a relation schema into a well-structured class. To transform RDB schema into OODB schema, a composition process simplifying the input RDB schema and an algorithm transforming the simplified RDB schema into well-structured OODB classes are developed. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Integrity constraints; Multi-valued dependency; Object-oriented model; Relational model; Schema transformation

1. Introduction

With the development of object-oriented technology, relational database (RDBs) are seeking ways to transit to object-oriented database (OODBs). This makes transforming RDB schema into OODB schema an important task and considerable effort has been seen on this topic [1,8]. All studies done so far focus on examining keys and key-based inclusion dependency (IDs) [6] to identify classes and derive inheritance and association between classes. Such transformation scheme usually results in classes having the same flat structure as the original relation schemas. In our view, however, this kind of class structure fails to make use of the OO modeling power to the largest extent and has not eliminated the data redundancy and update anomaly problem existing in the original RDB schema. Example 1 below explains the notion of class structure.

Example 1. Given a relation schema *Service*, key is underlined, and multi-valued dependency (MVDs) and IDs on the schema are explicitly listed (Later discussions follow the same style).

Service(flight, working-day, plane-type):
{ flight →→ working-day, flight →→ plane-type}

An example relation *service* over the schema is shown in Table 1. Owing to the MVDs, data redundancy and update anomaly is obvious in *service*: The same value for *flight* is repeated four times and so when it is updated, four tuples are affected.

How are we going to transform *Service* into a class? The most direct approach is as follows:

```
class Service-1 (  
    flight: int,  
    working-day: string,  
    plane-type: int  
);
```

With this class definition, corresponding to each tuple of relation *service* we have an object in class *Service-1* (Table 2). In Table 2 each row represents an object and object identifiers (OIDs) are explicitly listed. However, data redundancy of the original relation is retained. When we update the value of *plane-type* for object *O₁* from 747 to 727, to maintain the MVDs holding on *Service*, we must change the value of *plane-type* for object *O₂* accordingly. In other words, with such schema transformation, we must use the method mechanism of the OO model to enforce the MVDs of the relational model.

* Corresponding author. Department of Computer Science, University of Melbourne, Parkville VIC 3052, Australia.

E-mail address: xzhang@cs.mu.oz.au (X. Zhang)

Table 1
Relation *service*

Flight	Working-day	Plane-type
106	Monday	747
106	Thursday	747
106	Monday	1011
106	Thursday	1011
204	Wednesday	707
204	Wednesday	727

A more appropriate class structure for this transformation is as follows:

```
class Service-2 (
    flight:      int,
    working-day: set(string),
    plane-type:  set(int)
);
```

As shown in Table 3, under this class definition, there are only two objects in the resulting class. Objects are no longer in 1 : 1 correspondence with tuples of the original relation, instead, an object is the composition of several tuples. No information is lost, but the data redundancy of the original relation is completely removed. When we change the value of either *plane-type* or *working-day* of one object, only that object itself is affected. The MVDs on *Service* are built into the class definition and enforced by that attributes *working-day* and *plane-type* are defined to be set-valued.

Comparing the two classes *Service-1* and *Service-2* defined over the same attribute set, we say they are of different structures. Moreover, we say *Service-2* is well-structured but *Service-1* is not.

Organization of the paper is as follows: in Section 2 we define well-structured class and propose a theorem on constructing well-structured class from relation schema. In later discussions, when confusion does not arise, relation schemas will simply be called relations. In Section 3, based on MVDs, a composition process is proposed to first simplify the relational schema before the real transformation. Based on keys, key-based IDs, and MVDs, an algorithm is proposed mapping RDB schema into equivalent well-structured OODB schema. Section 4 is the conclusion.

Table 2
Class *Service-1*

Oid	Flight	Working-day	Plane-type
O ₁	106	Monday	747
O ₂	106	Thursday	747
O ₃	106	Monday	1011
O ₄	106	Thursday	1011
O ₅	204	Wednesday	707
O ₆	204	Wednesday	727

Table 3
Class *Service-2*

Oid	Flight	Working-day	Plane-type
O ₁	106	{Monday, Thursday}	{747, 1011}
O ₂	204	{Wednesday}	{707, 727}

2. Constructing well-structured class

Let us first examine the semantics of an attribute taking a set of values from its domain. Given a class *C*, assume that there are two objects in *C* (Table 4).

```
class C (
    A1: set(char),
    A2: set(int),
    A3: set(char)
);
```

As each object independently takes a set of values from its domain, when we “flatten” *C* so that each attribute takes a single value from its domain, we get eight combinations of attribute values, as shown in Table 5. Ignoring the explicitly listed OIDs, we can consider the eight rows of Table 5 as eight objects of the following class *C'*:

```
class C' (
    A1: char,
    A2: int,
    A3: char
);
```

However, to ensure that *C'* expresses set-valued attributes correctly, we must also use the method mechanism. In addition, *C'* has obvious data redundancy while *C* does not. This notion is expressed in the following definition and proposition.

Definition 1. A class *C* with attributes A_1, \dots, A_n , is *well-structured* if and only if (iff) A_i ($1 \leq i \leq n$) is defined as set-valued when there exist objects of *C* taking 2 or more values for A_i from its domain, independently of any other attribute A_j ($1 \leq j \leq n$ and $j \neq i$).

An OODB schema *C* consisting of a set of classes, where $C = \{C_1, C_2, \dots, C_n\}$, is well-structured if each class $C_i \in C$, $1 \leq i \leq n$, is well-structured.

Table 4
Class *C*

Oid	A ₁	A ₂	A ₃
O ₁	{a}	{1, 4}	{x, y}
O ₂	{a, c}	{1}	{s, t}

Table 5
Flattened C

Oid	A_1	$A_2 >$	A_3
O_1	a	1	x
O_1	a	1	y
O_1	a	4	x
O_1	a	4	y
O_2	a	1	s
O_2	a	1	t
O_2	c	1	s
O_2	c	1	t

Proposition 1. A class C is without data redundancy iff C is well structured.

Schema transformation is a ubiquitous concept in database design [2]. In transforming a source schema S into a target schema T , where S and T may be of the same or different data models, we should ensure the equivalence between S and T , denoted as $T \Leftrightarrow S$. Equivalence in our context is attribute value equivalence between the source and the target, which follows the generic equivalence definition of [3].

A RDB schema is a pair $\langle R, S \rangle$, where R is a set of relations, and S are the integrity constraints defined on R . An OODB schema consists of a set of class definitions. At the present stage of the OO technology, integrity constraints are more or less maintained by methods. In transforming a RDB schema into an OODB schema, we should try to express the constraints on the original relations as much as possible with the OO modeling power itself but not resort to methods.

If we now think of Table 5 as a relation over schema $R(oid: int, A_1:char, A_2:int, A_3:char)$, where we assume OID is of int type. We find that the following MVDs hold on R

$$oid \twoheadrightarrow A_1, oid \twoheadrightarrow A_2, oid \twoheadrightarrow A_3$$

This means that in a class C , if an attribute A is set-valued, $oid \twoheadrightarrow A$. Conversely, in schema transformation from RDB to OODB, when an MVD M holds on the original RDB schema, each object should take a different value for the attributes on the left-hand side (l.h.s.) of M and multiple values for the attributes on the right-hand side (r.h.s.) of M . So in the resulting definition, the l.h.s. attributes should be single-valued whereas those on the r.h.s. should be set-valued. Classes thus formed are not only equivalent to the input relations but are also well-structured. In example 1, assuming that the MVDs are enforced by methods, both class *Service-1* and *Service-2* are equivalent to relation *Service*. However, *Service-2* is well-structured but *Service-1* is not.

Let us first introduce the notations that will be used in later discussions. Given a relation schema R ,

- let $R(X:D_x, Y:D_y, Z:D_z)$ denote that X, Y and Z are subsets of R taking domains D_x, D_y and D_z respectively, and $R = X \cup Y \cup Z$. Not considering the domains of X, Y or Z, R is

simply written as $R(XYZ)$. Similar notation is also adopted for classes.

- $r(R)$, or simply r , is the relation over R . For $X \subseteq R$, $r[X]$ denotes $\pi_X(r)$.

Theorem 1. Given a relation $R(X:D_x, Y:D_y, Z:D_z): \{X \twoheadrightarrow Y, X \twoheadrightarrow Z\}$, a composite class C defined in the following is well-structured and $C \Leftrightarrow R$, where y and z are attributes referring to a set of objects of C_y and C_z , component classes of C .

class C (

$X: D_x$
 $y: set(C_y)$
 $z: set(C_z)$

);

class C_y (

$Y: D_y$

);

class C_z (

$Z: D_z$

);

Note that when $Y(Z)$ consists of a single attribute, $C_y(C_z)$ is then an atomic data type. As a result, $y(z)$ becomes set-valued atomic attribute of C and we do not need to define component class $C_y(C_z)$ by ourselves, which is the case with class *Service-2* mapped from relation *Service* of Example 1.

Proof. To map R into class C , let us first decide the structure of C . Obviously each object of C has a different X value. As $X \twoheadrightarrow Y(X \twoheadrightarrow Z)$, a set of $Y(Z)$ values is uniquely determined by the OID of an object. Thus when transforming R into a class C , X should be single-valued, both Y and Z should be set-valued. As set-value is only for single attribute, component classes [4] C_y and C_z are defined on Y and Z and we define attributes referring to a set of objects of C_y and C_z . Class C and its component classes C_y and C_z resulting from the aforementioned construction process are well-structured.

Let r be a relation on R and $r[X] = X, r[Y] = Y, r[Z] = Z$, map this relation r to a class C as follows:

- For each $y \in Y$, create an object O_y in C_y with $O_y Y = y$.
- For each $z \in Z$, create an object O_z in C_z with $O_z Z = z$.
- Under $X \twoheadrightarrow Y$ and thus $X \twoheadrightarrow Z$, assume that for each $x \in X$ the set of Y and Z values determined by x is denoted as Y_x and Z_x :

$$Y_x = \pi_Y \sigma_{"X=x"}(r), \quad Z_x = \pi_Z \sigma_{"X=x"}(r)$$

Table 6
Relation r_2

A_1	A_2	A_3	A_4
1	a	b	x
1	a	b	y
1	a	c	x
1	a	c	y
2	b	e	s
2	b	e	t

For each $x \in X$, create an object O in C with

$$O \cdot X = x,$$

$$O \cdot y = \{O_y \in C_y \mid O_y \cdot Y = y, y \in Y_x\},$$

$$O \cdot z = \{O_z \in C_z \mid O_z \cdot Z = z, z \in Z_x\}$$

For a tuple $t \in r$ with $t[X] = x, t[Y] = y, t[Z] = z$, its correspondence in C is an object $O \in C$, where $O \cdot X = x, O \cdot y \cdot Y = y, O \cdot z \cdot Z = z$. Conversely in a similar way each object of class C, C_y and C_z can find a counterpart in relation r . So $C \Leftrightarrow R$.

We can see from the transformation process that no data redundancy exists in the resulting classes. In class C no two objects have the same X values, y refers to a set of objects in C_y , and z refers to a set of objects in C_z . With C_y and C_z , no data redundancy exists for the values of Y or Z : each Y or Z value appears in only one object.

Example 2. Given a relation r_2 (Table 6) on schema R_2 :

$$R_2(A_1 : int, A_2 : char, A_3 : char, A_4 : char):$$

$$\{A_1 \twoheadrightarrow A_2 A_3, A_1 \twoheadrightarrow A_4\}$$

R_2 is transformed into class C_2 and component class C_2' , as shown below. Objects of C_2 and C_2' are shown in Tables 7 and 8.

```
class C2(
    A1: int,
    a2: set(C2'),
    a4: set(char)
);
class C2'(
    A2: char,
    A3: char
);
```

Table 7
Class C_2

Oid	A_1	a_2	A_4
O_{11}	1	$\{O_{21}, O_{22}\}$	$\{x, y\}$
O_{12}	2	$\{O_{23}\}$	$\{s, t\}$

Table 8
Class C_2'

Oid	A_2	A_3
O_{21}	a	b
O_{22}	a	c
O_{23}	b	e

3. Transforming relations into well-structured classes

The analysis in Section 2 results in a complete transformation process from RDB schema to a well-structured OODB schema. Before starting the transformation, a preprocess *composition* is needed. The rationale is given in the following.

3.1. Composition

When nontrivial MVDs hold on a relation R , to reduce data redundancy, R is often decomposed into several relations in 4NF where only trivial MVDs hold [5]. In view of schema transformation, however, this decomposition will only blur the semantic interpretation of relations and increase the number of classes in the target OODB schema. So, before proceeding with the transformation, we should recombine the relations resulting from this decomposition into one relation.

Theorem 2. For a relation r_1 over schema $R_1(XY): X \twoheadrightarrow Y$, and a relation r_2 over schema $R_2(XZ): X \twoheadrightarrow Z$. If $r_2[X] = r_1[X]$, we can construct a relation $r = r_1 \bowtie r_2$ over schema $R(XYZ): \{X \twoheadrightarrow Y, X \twoheadrightarrow Z\}$ and $R \Leftrightarrow \{R_1, R_2\}$.

This theorem shows a composition process, a reverse process of lossless decomposition [5]. So its validity is obvious. In addition to preserve attribute values, R also preserves all the MVDs and IDs on $\{R_1, R_2\}$. Before starting with schema transformation, in the input RDB schema, we repeatedly replace any pair of such schemas $\{R_1, R_2\}$ with the schema R resulting from the composition process described in Theorem 2. This composition and replacement process is denoted as

$$R: \{M, I\} \Leftarrow compose(R_1: \{M_1, I_1\}, R_2: \{M_2, I_2\})$$

With this expression we state explicitly that the MVDs(M_1 and M_2) and IDs(I_1 and I_2) on R_1 and R_2 are maintained by $R(M$ and $I)$ resulting from composition.

Example 3. Given two relations *service-day* (Table 9) and *service-plane* (Table 10) on schemas *Service-day* and *Service-plane*:

```
Service-day(flight, working-day):
    {flight \twoheadrightarrow working-day}
Service-plane(flight, plane-type):
    {flight \twoheadrightarrow plane-type}
```

To map these two relation schemas into OODB schema, we

Table 9
Relation *service-day*

Flight	Working-day
106	Monday
106	Thursday
204	Wednesday

first compose them into a single relation schema *Service* (Example 1),

$\text{Service}(\text{flight}, \text{working-day}, \text{plane-type})$:
 $\{\text{flight} \twoheadrightarrow \text{working-day}, \text{flight} \twoheadrightarrow \text{plane-type}\}$

and then transform *Service* into class *Service-2* as discussed before.

3.2. Transforming RDB schema into well-structured OODB schema

The input RDB schema is assumed to be well defined [6]: only key functional dependencies and key-based IDs exist in the RDB schema and the ID graph is a directed acyclic graph. Some basic assumptions about keys and IDs are:

- Each relation is uniquely identified by its key. Given a relation R , K_R denotes the key of R .
- IDs describe direct subset relationship between two sets of values: if two IDs have the same l.h.s., then at least one of them have an ID in the opposite direction and thus expresses its equivalence with its r.h.s. For example, given two IDs I_1 and I_2 , $I_1: r_1[X] \subseteq r_2[X]$, $I_2: r_1[X] \subseteq r_2'[X]$, then one of them, suppose it is I_2 , have a reverse ID $I_2': r_2'[X] \subseteq r_1[X]$. In this case, I_2 and I_2' together are simply denoted as $r_1[X] = r_2'[X]$.

Under these assumptions, when the composition process described in Section 3.1 completes, each ID has a clear semantic interpretation and so each relation is uniquely classified. Given a relation schema R ,

- R is a base relation if $\exists R_1$ s.t. $r[K_R] \subseteq r_1[K_{R1}]$.
- if there exists only one R_1 s.t. $K_{R1} = K_R$ and $r[KR1] \subseteq r_1[K_{R1}]$, each tuple in $r(R)$ has a correspondent in $r_1(R_1)$ but the converse does not hold. We say R is a *specializing* relation that specializes R_1 . R_1 is denoted as $rhs(R)$.
- if there exists only one R_1 s.t. $K_{R1} \subset K_R$ and $r[KR1] \subseteq r_1[K_{R1}]$, R is a *dependent* relation whose identification depends on R_1 . R_1 is denoted as $rhs(R)$.

Table 10
Relation *service-plane*

Flight	Plane-type
106	747
106	1011
204	707
204	727

- R is a *composite relation* if R is neither base nor dependent, i.e.,
- $\exists R_1, \dots, R_m$ s.t. $K_{R1} \cup \dots \cup K_{Rm} \subseteq K_R$ ($m \geq 2$) and $r[[K_{Ri}]] \subseteq r_i[K_{Ri}]$ for $1 \leq i \leq m$.
- $\{R_1, \dots, R_m\}$ is denoted as $rhs(R)$.

Given a relation $R(A_1:D_1, \dots, A_n:D_n)$, without losing generality, assume that keys and IDs are defined on the first several attributes. Key and ID constraints help to classify relations, identify classes and establish inheritance and association between classes. Different transformation rules are formed based on the type of R :

- (a) R is a base relation, it becomes a class:

$C(A_1:D_1, \dots, A_n:D_n)$ which is the base of an inheritance hierarchy.

- (b) R is a specializing relation that specializes $R_1: R(K_{R1}, A_s:D_s, \dots, A_n:D_n)$. This specialization is the *isa* relationship [4] in OO model. Assume that R_1 was transformed into class C_1 , R then becomes a subclass of C_1 :

$C(A_s:D_s, \dots, A_n:D_n)$ as subclass of C_1

- (c) R is a dependent relation (semantically a weak entity) depending on $R_1: R(K_{R1}, A_s:D_s, \dots, A_n:D_n)$. Assume that R_1 becomes class C_1 , R is then transformed into

$C(A_1: C_1, A_s:D_s, \dots, A_n:D_n)$

- (d) R is a composite relation: $R(K_{R1}, \dots, K_{Rm}, A_i:D_i, \dots, A_n:D_n)$. R describes some relationship between R_1, \dots, R_m . Assume that R_1, \dots, R_m becomes C_1, \dots, C_m in the target OODB, R is then transformed into a class describing an association between C_1, \dots, C_m :

$C(B_1:C_1, \dots, B_m:C_m, A_i:D_i, \dots, A_n:D_n)$

The above transformation rules only define attributes and its domains for a class. We should also define methods enforcing the relational key, ID and other integrity constraints. For how to define these methods, please refer to [7]. To further remove the data redundancy and update anomaly of the original relational schema, we must determine the structure of each class. To this end, the MVDs of the original relations must also be examined. The composition process in Section 3.1 and the mapping rules in Section 3.2 results in a detail transformation algorithm.

Algorithm T. Transform a well-defined RDB schema into a well-structured OODB schema.

Input: A well-defined RDB schema $\langle R, S \rangle$. $R = \{R_1, R_2, \dots, R_n\}$ and $S = F \cup I \cup M$, where F, I, M are sets of FDs, IDs and MVDs.

Output: A well-structured OODB schema $C \Leftrightarrow R$.

Begin

// Step 1: Composition and replacement

```

while( $\exists \{R_1 \in R: (M_1=X \rightarrow Y_1, I_1), R_2$ 
   $\in R: (M_2=X \rightarrow Y, I_2)\}$  s.t.  $r_1[X] = r_2[X]$ ) do
   $R: \{M, I\} \Leftarrow \text{compose}(R_1: \{M_1, I_1\}, R_2: \{M_2, I_2\});$ 
end while
// Step 2: Classify the input relations.
Classify R according to F and I, resulting in base relation set B, specializing relation set Z, dependent relation set D and composite relation set P.
// Step 3: Construct base classes.
for (each  $R \in B$ ) do
  transform  $R: \{M \in M\}$  into class C according to rule a) and Theorem 1;
end for
// Step 4: Repeatedly transform relations in Z, D, P into classes.
repeat {
  // Step 4.1: Dealing with specializing relation.
  for(each  $R \in Z$  where  $rhs(R)$  was transformed into class  $C_1$ ) do
    transform  $R: \{M \in M\}$  into class C as a subclass of  $C_1$  according to rule b) and Theorem 1;
    let  $Z = Z - \{R\}$ ;
  end for
  // Step 4.2: Dealing with dependent relation.
  for(each  $R \in D$  where  $rhs(R)$  was transformed into class  $C_1$ ) do
    transform  $R: \{M \in M\}$  into class C according to rule (c) and Theorem 1;
    let  $D = D - \{R\}$ ;
  end for
  // Step 4.3: Dealing with composite relation.
  for(each  $R \in P$  where  $rhs(R) = \{R_1, \dots, R_m\}$  and  $R_1, \dots, R_m$  become  $C_1, \dots, C_m$ ) do
    transform  $R: \{M \in M\}$  into class C referencing  $C_1, \dots, C_m$  according to rule (d) and Theorem 1;
    let  $P = P - \{R\}$ ;
  end for
} until ( $Z = D = P = \emptyset$ )
End

```

As the ID graph is acyclic, algorithm T definitely ends. Each relation in R is processed only once and all the resulting classes are constructed according to theorem 1, so the resulting OODB schema C is well-structured and $C \Leftrightarrow R$.

Theorem 3. Given a RDB schema $\langle R, S \rangle$, the OO schema C resulting from Algorithm T is well structured and $C \Leftrightarrow R$ w.r.t. S.

The aforementioned transformation process can be further explained with a representative example below.

Example 4. Look at the following RDB schema. Keys are underlined. $I = \{I_i | i = 1, \dots, 9\}$ and $M = \{M_1, M_2, M_3\}$ denote the IDs and MVDs. For simplicity, I_9 denotes 3 pairs of IDs of opposite directions.

```

Course(code : varchar(6), title: varchar(20))
Person(ssn : int, name: varchar(10), sex: char)
Staff(ssn : int, staff-id: int, dept: char(2))
Staff-dependent(stf : int, ssn : int, name: varchar(10),
  relationship: varchar(10))
Student(ssn : int, sno: int, prog: varchar(10))
Lecturer (course : varchar(6), stf : int):  $\{M_1 = \text{course} \rightarrow \text{stf}\}$ 
Course-stud(course : varchar(6), stud : int):  $\{M_2 = \text{course} \rightarrow \text{stud}\}$ 
Sched(course : varchar(6), hour : varchar(7), room :
  varchar(5)):  $\{M_3 = \text{course} \rightarrow \text{hour, room}\}$ 
 $I_1: \text{Staff}[\text{ssn}] \subseteq \text{Person}[\text{ssn}]$ 
 $I_2: \text{Staff-dependent}[\text{stf}] \subseteq \text{Staff}[\text{ssn}]$ 
 $I_3: \text{Student}[\text{ssn}] \subseteq \text{Person}[\text{ssn}]$ 
 $I_4: \text{Lecturer}[\text{course}] \subseteq \text{Course}[\text{code}]$ 
 $I_5: \text{Lecturer}[\text{stf}] \subseteq \text{Staff}[\text{ssn}]$ 
 $I_6: \text{Course-stud}[\text{course}] \subseteq \text{Course}[\text{code}]$ 
 $I_7: \text{Course\_stud}[\text{student}] \subseteq \text{Student}[\text{ssn}]$ 
 $I_8: \text{Sched}[\text{course}] \subseteq \text{Course}[\text{code}]$ 
 $I_9: \text{Lecturer}[\text{course}] = \text{Course-stud}[\text{course}] = \text{Sched}[\text{course}]$ 

```

Step 1. Composition and replacement.

With I_9 and M_1, M_2, M_3 , the composition process can be applied to Lecturer, Course-stud and Sched:

```

Schedule:  $\{M_1, M_2, M_3\} \Leftarrow \text{compose}(\text{Lecturer: } \{M_1\}, \text{Course-stud: } \{M_2\}, \text{Sched: } \{M_3\})$ 

```

We have $\text{Schedule}(\text{course, hour, room, stf, stud})$ and $I_5' \Leftarrow I_5, I_7' \Leftarrow I_7, I \Leftarrow \{I_4, I_6, I_8\}$:

```

 $I_5': \text{Schedule}[\text{stf}] \subseteq \text{Staff}[\text{ssn}]$ 
 $I_7': \text{Schedule}[\text{stud}] \subseteq \text{Student}[\text{ssn}]$ 
 $I: \text{Schedule}[\text{course}] \subseteq \text{Course}[\text{code}]$ 

```

Now we have five relations: Course, Person, Staff, Staff-dependent, Student, and Schedule. $I = \{I_1, I_2, I_3, I_5', I_7', I\}$. $M = \{M_1, M_2, M_3\}$, where M_1, M_2, M_3 are defined on Schedule.

Step 2. Classification. According to relation keys and I, Person and Course are base relations. Staff and Student specialize Person. Staff_Dependent is a dependent relation of Staff. Schedule is a composite relation.

Step 3. Base class construction. According to rule (a), *Course* and *Person* are transformed into base classes:

```
class Ccourse (
    code: string,
    title: string
);
class Cperson (
    ssn: int,
    name: string,
    sex: char
);
```

Step 4.1. Constructing subclasses. According to rule (b), *Staff* and *Student* become subclasses of *Cperson*.

```
class Cstaff as subclass of Cperson (
    staff-id: int,
    dept: string
);
class Cstudent as subclass of Cperson (
    sno: int,
    prog: String
)
```

Step 4.2. Create class for dependent relation. According to rule (c), *Staff_dependent* is mapped into class *Cstaff-dependent*:

```
class Cstaff-dependent(
    stf: Cstaff,
    ssn: int,
    relationship: string
);
```

Step 4.3. Establishing association between classes. According to rule (d) and Theorem 1, *Schedule* becomes class *Cschedule*:

```
class Cschedule (
    course: Ccourse,
    hour-room: set(Chour-room),
    staff: set(Cstaff),
    student: set(Cstudent)
);
class Chour-room(
    hour: string;
    room: string;
);
```

The final OODB schema consists of the seven classes as defined earlier.

4. Conclusion

In transforming RDB schema into OODB schema, while key and key-based IDs are critical for identifying classes and establishing inheritance and association between classes, MVDs are important for defining well-structured classes. A well-structured class is not only equivalent to the original relation schema but also remove the original data redundancy. However, not much work was seen on this topic.

In this article, we propose the notion of class structure and define well-structured class. Based on MVDs, a theorem is given on transforming a relation schema into an equivalent well-structured class. In transforming RDB schema into OODB schema, a composition process is proposed to reduce the input RDB schema. All the analysis above helps to produce an algorithm that completes the transformation from RDB to OODB schema. Theorems are given to prove that the transformation is information-preserving and the resulting classes are well-structured.

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments.

References

- [1] J. Fong, Converting relational to object-oriented databases, SIGMOD Record 26 (1) (1997) 53–58.
- [2] J.-L. Hainaut, C. Tonneau, M. Joris, M. Chandelon, Transformation-based Database Reverse Engineering, in: R. A. Elmasri, V. Kouramajian, B. Thalheim (Eds.), Entity Relationship Approach – ER'93, LNCS 823, Springer, Berlin, 1993, pp. 364–375.
- [3] R. Hull, Relative information capacity of simple relational database schemata, SIAM J. Comp. 15 (3) (1986) 856–886.
- [4] W. Kim, Introduction to Object-Oriented Databases, MIT Press, Cambridge, MA, 1990.
- [5] D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, MD, 1983.
- [6] K.-J. Raiha, H. Mannila, The Design of Relational Databases, Addison-Wesley, Reading, MA, 1992.
- [7] B. Narasimhan, S.B. Navathe, S. Jayaraman, On mapping ER and relational models into OO schemas, in: R.A. Elmasri, V. Kouramajian, B. Thalheim (Eds.), Entity Relationship Approach – ER'93, LNCS 823, Springer, Berlin, 1993, pp. 402.
- [8] L.-L. Yan, T.-W. Ling, Translating relational schema with constraints into OODB schema, in: D.K. Hsiao, E.J. Neuhold, R. Sacks-Davis (Eds.), Interoperable Database Systems, Elsevier, Amsterdam, 1993, pp. 69–85.