

# Translating object-oriented database transactions into relational transactions

Joseph Fong<sup>1</sup>

Computer Science Department, City University of Hong Kong, Tat Chee Avenue, Hong Kong  
e-mail: [csjfong@cityu.edu.hk](mailto:csjfong@cityu.edu.hk)

## Abstract

In this paper, we present methods of translating transactions from object-oriented database(OODB) to relational database(RDB). The process involves schema mapping in data definition language(DDL) and transaction translation in data manipulation language(DML). They include scheme definition, data query and transaction operation of insert, update, and deletion. We also discuss the object-oriented features in OODB operations that are not supported by RDB, such as class hierarchy, class composition hierarchy, and set attribute, and provide a general solution to realize those mechanisms by traditional relation operations. The result of the transaction translation can be applied into adding object oriented interface into relational database management system (RDBMS) and to the interoperability between OODB and RDB.

Keywords: OODB, RDB, ORDB, multidatabase, transaction translation, interoperability.

## 1. Introduction

The requirement of interoperability of autonomous databases leads to the multidatabase system which consists of homogenous or heterogeneous databases. In heterogeneous multidatabase system, the translation between operations of databases with different data models is critical<sup>1</sup>. As RDB is dominating database application and object-relational database(ORDB) maybe the next generation database to meet advanced business requirements, it is necessary and practical to search for a solution of transaction translation.

There are methods of transaction translation between OODB and RDB, and each direction reflects, based on the architecture of multidatabase system, a certain system function. W. Meng<sup>2</sup> discussed the transaction translation from RDB to OODB based on a schema translation form OODB to RDB. D. Keim<sup>3</sup> and C. Yu<sup>4</sup> discussed the query translation from OODB to RDB based on a schema translation from RDB to OODB. J. Fong<sup>5,6</sup> provided techniques in translating SQL to OQL and OSQL. Much work<sup>7,8,9</sup> were done on query translation between RDB and OODB. M. Blaha<sup>10</sup> has proposed four ways to handle inheritance by use of relation tables. In order to keep the semantic of inheritance and to simplify the mapping we propose a different way. Our way is, in addition to map each class in a class hierarchy to a table, to extend the attributes of superclass tables to all their subclass tables according to inheritance rules. Our approach on the transaction translation from OODB to RDB after the schema translation from OODB to RDB is performed. Our model on the interoperability of relational and object-oriented databases is described in Figure 1.

The model in Figure 1 is equivalent to build an object-oriented interface on top of a RDBMS. In the model, the Gateway translates object-oriented query language (OSQL)<sup>11</sup> to relational structural query language (SQL).

The following sections are arranged in this way. Section 2 describes relational data model and

---

<sup>1</sup> This research project is supported by a strategic research grant 7001240 of City University of Hong Kong

object-oriented data model. Section 3 discusses transaction translations and gives translation mappings and algorithms for each typical database operation. The operations include scheme definition, data insertion, data query, data update, and data deletion. Section 4 gives a real world case study. Section 5 concludes the paper.

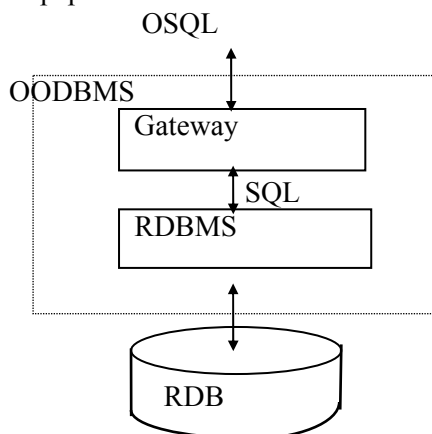


Figure 1. Research Model

## 2. Data Model

The data models described in this section contain the standard features of relational data model<sup>12</sup> and object-oriented data model. We also establish a correspondence between their parallel counterparts with notations.

### 2.1. Relational Data Model

We define a relational database as a 3-tuple:  $RDB = \langle R, T, OP_r \rangle$ , where

$$\begin{aligned}
 R &= \{ R \mid R \text{ is a relation scheme defined in RDB} \} \\
 T &= \{ T_R \mid T_R \text{ is a relation (set of tuples) having } R \text{ as its scheme} \} \\
 OP_r &= \{ op_r \mid op_r \text{ is a relational database operation} \}
 \end{aligned}$$

Relational data model is in the form of two-dimensional tables of relations. Each relation has a scheme definition and tuples. Relation scheme determines the structure of a relation by defining a set of attributes, and each tuple is an instance of the relation with given values for each attribute defined in the scheme. Attribute defined in relation scheme may associate with an atomic data type which restricts on a set of permissible data for the value of the corresponding attribute. A subset of a relation's attributes is called a key which can be referred by the attributes of another relation in a foreign key.

The RDB operations are scheme definition, data insertion, data update, data deletion, and data query. Query can be divided into set operations such as union, intersection, and difference, and relation operations include cartesian product, projection, selection and join.

### 2.2. Object-Oriented Data Model

We define an object-oriented database in a 3-tuple:  $OODB = \langle C, O, OP_o \rangle$ , where

$$\begin{aligned}
 C &= \{ C \mid C \text{ is a class defined in OODB} \} \\
 O &= \{ O_C \mid O_C \text{ is a set of instances which have class } C \text{ as its type} \} \\
 OP_o &= \{ op_o \mid op_o \text{ is an object-oriented database operation} \}
 \end{aligned}$$

Object-oriented data model consists of objects. Each object is uniquely identified by OID(object identifier) defined and assigned by system with a state and behavior associated with it. The state

of an object is determined by the value of its attributes and operations. The behavior of an object is specified by the methods that operate on the object state. Objects with the same attributes and behaviors are grouped in a class as instances of that class.

Class defines attributes and methods for the state and behavior of a group of similar objects. With inheritance, classes are organized in class hierarchies such that subclass inherits attributes and methods from its superclasses, and may reflect generalization or specialization relationship semantics between superclass and subclass. These semantics are presented in database operations.

Attributes in the classes may be divided into atomic attribute, aggregate attribute(or composite attribute), and set attribute. Atomic attribute is defined as a basic type, such as integer, boolean, and string. Aggregate attribute is defined as a stored OID<sup>2</sup> in a class. The aggregate attributes may form a class composition hierarchy in a directed graph. Set attribute is defined to have multiple values of atomic type or aggregate attribute. We regard OID as the value of an implied atomic attribute.

Each OODB operation may contain mechanisms not supported by RDB. For example, range clause in query may denote a class or a group of classes in a class hierarchy, and target clause may contain path expression to navigate through composition hierarchy.

### 2.3. Notations

We define notations to express concepts and operations in RDB and OODB. They represent functions and semantics of typical database operations.

- RDB scheme:

- $A_R$ : an attribute defined in relation scheme R.
- $A_R^{key}$ : key attribute(s) of relation scheme R.
- $A_R^{fkey}$ : foreign key attribute(s) of relation scheme R.
- NAME( $A_R$ ): name of attribute  $A_R$ .
- DOM( $A_R$ ): type of attribute  $A_R$ .
- NAME(R): name of relation R.
- ATT(R): a set of all attributes defined in relation scheme R.

- RDB operation:

- $C_r(R, S_A)$ : create relation R with a set of attributes in set  $S_A$ .
- $I_r(R, S_V)$ : insert a tuple into relation R with a set of values in set  $S_V$ .
- $S_r(R, Q_R, A_R)$ : select the values, satisfying qualification  $Q_R$ , of attribute  $A_R$  of relation R.
- $U_r(R, Q_R, A_R, v_R)$ : update the values, satisfying qualification  $Q_R$ , of attribute  $A_R$  of relation R with new value  $v_R$ .
- $D_r(R, Q_R)$ : delete the tuples of relation R, which satisfy qualification  $Q_R$ .

- OODB class:

- $T^{oid}$ : type of implied attribute having object identifier as its value.
- $T^{atomic}$ : type of atomic attribute.
- $T^{class}$ : type of aggregate attribute,

---

<sup>2</sup> Stored OID is an aggregation attribute in an object of a class which references(pointing) to an OID stored in an object of another class.

- $T^{set}$  : type of set attribute.  
 $A_C$  or  $A_C^T$  : an attribute of class C or a T type attribute of class C.  
NAME( $A_C$ ): name of attribute  $A_C$ .  
DOM( $A_C$ ): class or type of attribute  $A_C$ .  
ELEDOM( $A_C^{set}$ ): class or type of set element.  
NAME(C): name of class C.  
ATT(C): a set of attributes defined in class C.  
SUB(C): a set of subclasses of class C.
- OODB operation:
    - $C_o(C, S_C, S_A)$ : create class C as a subclass of classes in set  $S_C$  with additional attributes in set  $S_A$ .
    - $I_o(C, S_V)$ : create an object of class C with values in set  $S_V$ .
    - $S_o(C, Q_C, A_C)$ : select values, satisfying qualification  $Q_C$ , of attribute  $A_C$  of class C.
    - $U_o(C, Q_C, A_C, v_C)$ : update values, satisfying qualification  $Q_C$ , of attribute  $A_C$  of class C with new value  $v_C$ .
    - $D_o(C, Q_C)$ : delete objects of class C, which satisfies qualification  $Q_C$ .

### 3. Transaction Translation

Transaction translation is to realize the functions and semantics of operations of one set by using another set of operations. For translation, we need to set up mapping regulations between the two set operations. In this section, we map each OODB operation to RDB operations and then based on the mappings, we formulate algorithms to derive the uncertain parameters from the known parameters or parameters determined by system.

#### 3.1. Scheme Creation Translation

Schema is the set of all schemes defined in a database. Scheme is class definition in OODB and is relation in RDB. The scheme creation translation maps a class definition to a relation scheme definition. Each attribute of the class is mapped into an attribute of the corresponding relation with the preservation of the semantics of the class in the mapped relation scheme. However relation scheme doesn't have inheritance and non-atomic data types. As for inheritance, we have  $ATT(C) \supseteq S_A$  in operation  $C_o(C, S_C, S_A)$ . It is necessary to ensure what attributes the class C has by taking  $S_C$  into consideration. As for non-atomic type attributes in OODB, they are mapped to atomic type attributes or relation schemes in RDB.

##### 3.1.1. Translation mapping

We set the scheme mapping to two levels. One is scheme level such that each class is mapped to a relation scheme. The other is attribute level such that different type attributes are treated differently. Implied OID is mapped to a key attribute. Atomic attribute is mapped into an attribute without change. Composite attribute is mapped into a foreign key attribute referencing to the key of the relation corresponding to the attribute domain class. Set attribute is mapped into an atomic attribute plus an extra relation scheme which has two attributes. One is to set a link to the attribute corresponding to the set attribute. The other is to deal with set elements. Suppose we have  $S_C = \{ C^1, \dots, C^m \}$ ,  $S_A = \{ A^1_C, \dots, A^n_C \}$ ,  $m \geq 0$  and  $n \geq 0$  in class creation operation  $C_o(C, S_C, S_A)$ . The scheme mapping  $\rho_S$  is expressed as:

$\rho_s(C_o(C, \{C^1, \dots, C^m\}, \{A^1_C, \dots, A^n_C\})) \rightarrow C_r(R_C, \{A^1_{R_C}, \dots, A^k_{R_C}\})$  and  
 $\{C_r(R_{A^i_C}, \{A^1_{R_{A^i_C}}, A^2_{R_{A^i_C}}\}) | \text{DOM}(A^i_C) = T^{set}\}$

and attribute mapping is:

$\rho_s(A_C^{oid}) \rightarrow A_{R_C}^{key}$

$\rho_s(A_C^{atomic}) \rightarrow A_{R_C}$

$\rho_s(A_C^{set}) \rightarrow A_{R_C} + R_{A_C} (A^1_{R_{A_C}}, A^2_{R_{A_C}})$

$\rho_s(A_C^{class}) \rightarrow A_{R_C}^{fkey}$

### 3.1.2. Translation algorithm

Based on the mapping of  $\rho_s$ , algorithm for scheme translation can be realized by determining relation name with its structure, and attribute name with its data type as follows:

```

scheme_translation(C, C1, ..., Cm, A1C, ..., AnC)
// input: Co(C, {C1, ..., Cm}, {A1C, ..., AnC})
// output: Cr(RC, {A1RC, ..., AkRC}) and
//      {Cr(RAiC, {A1RAiC, A2RAiC}) | DOM(AiC) = Tset}
{
  NAME(RC) = NAME(C);
  att = {ACoid}; // ACoid is an atomic type attribute defined by system //
  att = att ∪ {A1C, ..., AnC};
  for every class Ci ∈ {C1, ..., Cm} do
    att = att ∪ ATT(Ci);
  for every attribute AiC ∈ att do // 1 ≤ i ≤ k = |att| //
    {
      case DOM(AiC) = Toid:
        AiRC = AiC;
      case DOM(AiC) = Tatomic:
        AiRC = AiC;
      case DOM(AiC) = Tclass:
        AiRC = ARDOM(AiC)key;
      case DOM(AiC) = Tset:
        AiRC = Aset; // Aset is an atomic type attribute defined by system //
        NAME(RAiC) = NAME(C) + NAME(AiC);
        A1RAiC = AiRC;
        if (ELEDOM(AiC) = Tclass)
          A2RAiC = ARELEDOM(AiC)key;
        else
          A2RAiC = Aele; // NAME(Aele) = NAME(AiC) & DOM(Aele) = ELEDOM(AiC)
    }
}

```

```

}
}

```

After the scheme translation, both the source scheme and target scheme are kept by system as metadata. The information about the relationship between superclass and its subclasses, between composite class and its component classes, between class and its attributes, between class and its corresponding relations, and between relation and its attributes are available for transaction translations.

### 3.1.3. Example

Below is an example of schema translation. Suppose we have a series of class creation operations for the following class definitions:

```

class S1( A1:integer )
class S2( A2:integer, A3:string, A4:float ) subclass of S1
class S3( A5:S2 ) subclass of S1
class S4( A6:set of (string), A7:integer ) subclass of S3
class S5( A8:S3, A9:S4 ) subclass of S1

```

According to the scheme translation algorithm, the creation operations create the relation schemes as follows:

```

relation scheme S1( S1_OID:atomic type, A1:integer )
relation scheme S2( S2_OID:atomic type, A1:integer, A2:integer, A3:string,
A4:float )
relation scheme S3( S3_OID:atomic type, A1:integer, S2_OID:atomic type )
relation scheme S4( S4_OID:atomic type, A1:integer, S2_OID:atomic type,
A6_SET:atomic type , A7:integer )
relation scheme S4A6 ( A6_SET:atomic type, A6:string )
relation scheme S5( S5_OID:atomic type, A1:integer, S3_OID:atomic type,
S4_OID:atomic type )

```

## 3.2 Creation Translation

Creation translation maps an insertion operation of a class object to the insertion operation of a relation tuple with inserted tuples into appropriate relations mapped from the class. The tuples are reorganized by the given values of the object with OID attribute assigned by system. The set values of an attribute will be mapped into multiple tuples in an additional relation. If the set is empty, there is no tuple inserted to the relation.

### 3.2.1. Translation mapping

Suppose we have  $S_V = \{ v^1_C, \dots, v^n_C \}$  in operation  $I_o(C, S_V)$  which creates an object of class C with values  $v^1_C, \dots, v^n_C$ . Each value  $v^i_C$  corresponds to one attribute  $A^i_C$  defined in the class C,  $1 \leq i \leq n$  and  $n = |\text{ATT}(C)|$ . The value of a composite attribute is an OID. Based on the scheme translation, creation operation mapping is:

$$\rho_C(I_o(C, \{ v^1_C, \dots, v^n_C \})) \rightarrow I_r(R_C, \{ oid_{R_C}, v^1_{R_C}, \dots, v^n_{R_C} \}) \text{ and} \\ \{ I_r(R_{A^i_C}, \{ oid_{R_{A^i_C}}, v^j_{R_{A^i_C}} \}) \mid (\text{DOM}(A^i_C) = T^{set}) \& (1 \leq i \leq n) \& (1 \leq j \leq |v^i_C|) \}$$

### 3.2.2. Translation algorithm

Based on the mapping  $\rho_C$ , we have an algorithm for scheme translation to produce an unique OID.

```

creation_translation(C, v1C, ..., vnC, ρS)
// input: Io(C, {v1C, ..., vnC}) and ρS
// output : Ir(RC, {oidRC, v1RC, ..., vnRC}) and
//      { Ir(RAiC, {oidRAiC, vjRAiC}) | (DOM(AiC) = Tset) & (1 ≤ i ≤ n) & (1 ≤ j ≤ |viC|) }
{
  RC ← ρS(C);
  oidRC = oid_generator();
  for ( i = 1; i ≤ ATT(C); i = i+1 )
  {
    case AiC = Tatomic :
      viRC = viC;
    case AiC = Tclass :
      viRC = viC; // the value of a composite type attribute is an oid of an object //
    case AiC = Tset :
      if ( |viC| = 0 )
        viRC = null;
      else
      {
        viRC = oid_generator();
        RAiC ← ρS(AiC);
        oidRAiC = viRC;
        for ( j = 1; j ≤ |viC|; j = j+1 ) // viC should be a set of values //
          vjRAiC = vjC; // vjC is the jth element of set viC //
      }
    }
  }
}

```

### 3.2.3. Example

Suppose we want to create an object of class S4. The object creation operation is:

I<sub>o</sub>( S4, { 1, s2-oid-i, {‘abc’, ‘efg’, ‘hij’}, 2 } )

After translation, the resultant relation tuple insertion operations are:

I<sub>r</sub>( S4, { s4-oid-j, 1, s2-oid-i, a6-set-k, 2 } )  
 I<sub>r</sub>( S4A6, { a6-oid-k, ‘abc’ } )  
 I<sub>r</sub>( S4A6, { a6-oid-k, ‘efg’ } )  
 I<sub>r</sub>( S4A6, { a6-oid-k, ‘hij’ } )

### 3.3. Qualification Translation

The function of object qualification  $Q_C$  filters certain objects from all objects of class C. In some transactions, we need to map  $Q_C$  to a relational qualification  $Q_R$  which filters the tuples corresponding to the objects qualified by  $Q_C$ . The mapping is denoted as  $\rho_{\text{QUA}}(Q_C) \rightarrow Q_R$ .

Both  $Q_C$  and  $Q_R$  are of boolean expressions, but  $Q_C$  has two extra mechanisms which  $Q_R$  doesn't have. One is path expression operand. The other is set operand with its operator and quantifier. The problem is how to map these two mechanisms to relational correspondents. Since class and set attribute are mapped to relations, the path expression operand used for navigation operation in  $Q_C$  is mapped to join conditions in  $Q_R$  and set operations in  $Q_C$  is mapped to relation operation in  $Q_R$ . We will show a simplified model for  $Q_C$ , and offer algorithms to translate  $Q_C$  to  $Q_R$  by using graph technique. For simplicity we regard  $Q_C$  and  $Q_R$ , being connective form of predicates, as two sets of the predicates.

### 3.3.1 Qualification model

$Q_C$  filters certain objects not only by directly setting predicates on some attributes of class C but also by navigating through the class composition hierarchy starting at class C and setting predicates on some attributes of the direct or indirect component classes of class C. In general,  $Q_C$  is a combination of predicates with logical connectives  $\wedge, \vee, \neg$ . We assume  $Q_C$  has the form:  $Q_C = (q^1_C) \wedge \dots \wedge (q^n_C)$ ,  $n \geq 0$ . That is  $Q_C$  contains zero or more OODB predicates in conjunctive form. Each OODB predicate  $q^i_C$  ( $1 \leq i \leq n$ ) is recursively defined as:

1. Atomic predicate:  $t_1 \theta t_2$

$t_j$  ( $j=1,2$ ) is either a path expression or a constant distinctively. The path expression contains a sequence of attributes derived from an attribute of class C and has the form of  $A_{C_0} \cdot A_{C_1} \cdot A_{C_2} \cdot \dots \cdot A_{C_n}$ ,  $n \geq 0$ ,  $C_0 = C$ , and  $(\text{DOM}(A_{C_i}) = C_{i+1}, 0 \leq i \leq n-1)$  where  $n > 0$ .  $\theta$  is a comparison operator such as  $<, \leq, =, \neq, >, \geq, \supset, \supseteq, \not\supset, \subset, \subseteq, \in, \notin$  depending on the data type of the operands.

2. quantified predicate:  $\Theta(q^i_C)$

$q^i_C$  contains a set value attribute and  $\Theta$  is either the universal quantifier( $\forall$ ) or the existential quantifier( $\exists$ ).

The OODB schema draws a schema graph of two dimensions which presents two natures of OODB. Class hierarchy is a class that may have superclasses and subclasses. Class composition hierarchy is a class that has attributes whose domains are also classes. From class composition hierarchy, we can abstract a sub-graph of qualification graph, denoted as  $G_C$ . From class composition hierarchy, we can abstract the model of  $Q_C$ .

We define object qualification graph  $G_C$  as an annotated directed graph  $\text{OG}(\text{OV}, \text{OE})$ , where  $\text{OV}$  is a set of vertices, and  $\text{OE}$  is a set of directed edges.  $\text{OG}$  is a weak connected graph which has one vertex of source vertex with in-degree being zero and  $m$  ( $m \geq 1$ ) vertices of terminal vertices with out-degree being zero. A non-terminal vertex  $v$  in  $\text{OV}$  associates with a class and a terminal vertex  $v$  in  $\text{OV}$  with a predicate having path expression of  $n=0$  in its operands. A directed edge  $e$  from non-terminal vertex  $v_1$  to non-terminal vertex  $v_2$  in  $\text{OE}$  indicates a composition association such that the class of vertex  $v_2$  is a component class of the class of vertex  $v_1$ , and the annotation attached to the edge  $e$  is an attribute of the class of vertex  $v_1$  and its

domain is the class of vertex  $v_2$ . A directed edge  $e$  from non-terminal vertex  $v_1$  to terminal vertex  $v_2$  in OE indicates that the predicate of vertex  $v_2$  works on attribute of the class of vertex  $v_1$ , and there is no annotation attached to the edge.

### 3.3.2 Organizing OODB qualification graph

For a given  $Q_C$ , we can organize a qualification graph  $G_C$ . The graph has the class  $C$  as its source vertex and each predicate  $q^i_C$  corresponds to a path of the graph. The path starts from the source vertex and ends to one of terminal vertices. Each non-terminal vertex of the graph associates with a class which has the domain of a composite attribute appeared in a path expression and the edge to the vertex is annotated with the attribute. Each terminal vertex of the graph associates with a predicate which contains the last attribute in a path expression. Algorithm of organizing  $G_C$  from  $Q_C$  is:

OQG\_organizing( $Q_C, \rho_S$ )

// input:  $Q_C$  and  $\rho_S$

// output :  $G_C$

```

{
  let C be the source vertex of  $G_C$ ;
  for every  $q^i_C$  in  $Q_C$  do
  {
    set a pointer P to point source vertex;
    for every attribute  $A_{C_i}$  in path expression do
    {
      if  $A_{C_i}$  is not the last attribute in the path expression
      {
        if DOM( $A_{C_i}$ ) is not a vertex adjacent from the vertex pointed by P
        {
          if vertex DOM( $A_{C_i}$ ) does not exist
          {
            create a vertex with class DOM( $A_{C_i}$ );
            let vertex DOM( $A_{C_i}$ ) be adjacent from the vertex pointed by the pointer P;
            let  $A_{C_i}$  be the annotation of the edge to the vertex DOM( $A_{C_i}$ );
          }
          adjust P to point vertex DOM( $A_{C_i}$ );
        }
      }
      else
      {
        create a vertex adjacent from the vertex pointed by P with predicate on  $A_{C_i}$ ;
      }
    }
  }
}

```

For example, graph for  $Q_{SS}: (A1 < 10) \wedge (A8.A5.A2 > 0) \wedge (A8.A5.A3 = 'abc') \wedge (A9.A5.A4 = 5.5) \wedge ('bcd' \in A9.A6)$  is:

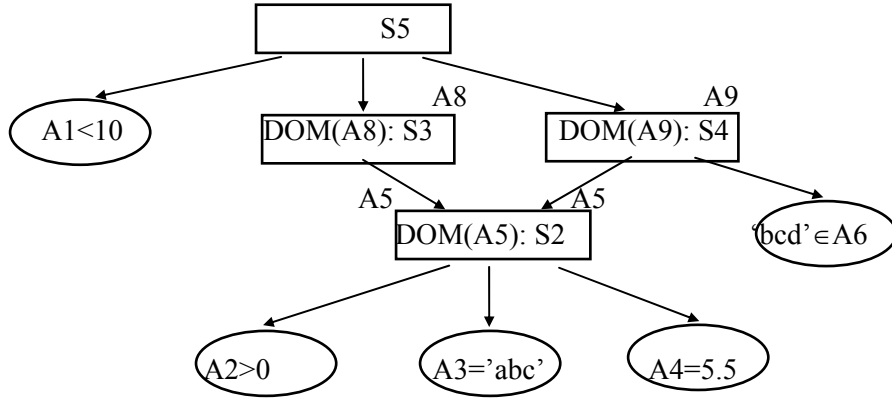


Figure 2. Object Qualification Graph of  $Q_{S5}$

### 3.3.3 Translating OODB qualification graph to RDB qualification graph

Class can be mapped to relation. The semantics of navigation operation used in the query of OODB can be realized by using join operation in the query of RDB. We can map OODB qualification graph to RDB qualification graph of  $G_R$ , which expresses predicates on a set of relations for relational query operation.

We define relation qualification graph  $G_R$  as an annotated directed graph  $RG(RV, RE)$ , where  $RV$  is a set of vertices, and  $RE$  is a set of directed edges. A non-terminal vertex  $v$  in  $RV$  associates with a relation and a terminal vertex  $v$  in  $OV$  associates with a predicate. An edge  $e$  from non-terminal vertex  $v_1$  to non-terminal vertex  $v_2$  in  $RE$  indicates a join association between the two relations, and the annotation attached to edge  $e$  is the join predicate on the two related relations. An edge  $e$  from non-terminal vertex  $v_1$  to terminal vertex  $v_2$  in  $OE$  indicates that the predicate of vertex  $v_2$  works on the attribute of the relation of vertex  $v_1$ , and there is no annotation attached to the edge.

To translate OODB qualification graph to a corresponding RDB qualification graph is to replace class of each non-terminal vertex of  $G_C$  by its corresponding relation. To replace annotated attribute of each edge between two non-terminal vertices of  $G_C$  by join predicate which sets the attribute, corresponding to the annotated attribute of  $G_C$ , of the relation of the tail vertex of the edge equal to the key attribute of the relation of the head vertex of the edge. To replace attribute in each terminal vertex of  $G_C$  by its counterpart attribute in corresponding relation, if the attribute in a terminal vertex is set type, insert a new non-terminal vertex of the relation mapped from the set attribute between the terminal vertex and its adjacent from vertex in  $G_R$ . To annotate the edge to this new inserted vertex with a join predicate that is the first attribute of the relation of the new inserted vertex is equal to the attribute, corresponding to the set attribute, in the relation of its adjacent from vertex. Modify the predicate in the terminal vertex according to the semantics of the set operator. In schema translation<sup>10</sup>, we give the mapping from set related operations such as  $\in$ ,  $\forall$ ,  $\exists$ , and  $\subseteq$  to relation operations. Algorithm of translating  $G_C$  to  $G_R$  is:

OQG\_to\_RQG\_translating( $G_C, \rho_S$ )

// input:  $G_C$  and  $\rho_S$

// output :  $G_R$

{

```

traverse graph  $G_C$  and for every vertex do
{
  if the vertex is not a terminal vertex
  {
    replace the class of the vertex by its corresponding relation;
    for every edge to the vertex do
    {
      replace the annotated attribute of the edge by join predicates (i.e. the
      attribute, corresponding to the annotated attribute, of the relation of the tail
      vertex is set to be equal to the key attribute of the relation of the head vertex);
    }
  }
  else
  {
    if the attribute in the terminal vertex is not a set type
      replace the attribute in the terminal vertex by corresponding relation attribute;
    else
    {
      insert a new vertex between the terminal vertex and its adjacent from vertex;
      set the new inserted vertex as the relation mapped from the set attribute;
      annotate the edge to the new inserted vertex with a join predicate(i.e. the
      first attribute of the relation of the head vertex is equal to the attribute,
      corresponding to the set attribute, in the relation of the tail vertex);
      modify the terminal vertex predicate according to the semantic of set operator;
    }
  }
}

```

As a result, we have:

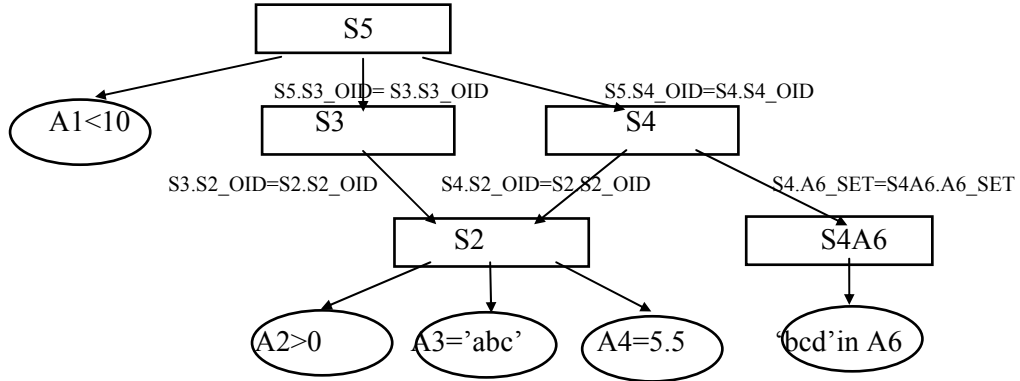


Figure 3.  $G_R$  translated from  $G_C$  of  $Q_{S5}$  in Figure 2

### 3.3.4 Producing RDB qualification

By traversing graph  $G_R$ , we can produce the set of the predicates which constitute the corresponding relation qualification. The translation algorithm is:

RQ\_producing( $G_R$ ,  $\rho_S$ )

// input:  $G_R$  and  $\rho_S$

// output :  $Q_R$

```

{
   $Q_R = \emptyset$ ;
  traverse graph  $G_R$  and for each vertex do
  {
    if the vertex is a terminal vertex
       $Q_R = Q_R \cup \{ \text{the predicate of the vertex} \}$ ;
    else
      {
         $Q_R = Q_R \cup \{ \text{the predicates of the edges to the vertex} \}$ ;
      }
  }
}

```

Continuing to the above example by using breadth-first search to traverse the graph of Figure 3, we get:

$$Q_R = \{ (S5.A1 < 10), (S5.S3\_OID = S3.S3\_OID), (S5.S4\_OID = S4.S4\_OID), \\ (S3.S2\_OID = S2.S2\_OID), (S4.S2\_OID = S2.S2\_OID), \\ (S4.A6\_SET = S4A6.A6\_SET), (S2.A2 > 0), (S2.A3 = 'abc'), \\ (S2.A4 = 5.5), ('bcd' \text{ in } S4A6) \}$$

### 3.3.5 Qualification translation algorithm

As mentioned before, the algorithm of qualification translation is:

qualification\_translation( $Q_C$ ,  $\rho_S$ );

// input:  $Q_C$  and  $\rho_S$

```

// output :  $Q_R$ 
{
  OQG_organizing( $Q_C, \rho_S$ );           // organizing  $G_C$  from  $Q_C$  //
  OQG_to_RQG_translating( $G_C, \rho_S$ ); // translating  $G_C$  to  $G_R$  //
  RQ_producing( $G_R, \rho_S$ );           // producing  $Q_R$  form  $G_R$  //
}

```

### 3.4. Query Translation

Query  $S_o(C, Q_C, A)$  in OODB is divided into three parts. Range part C indicates the class that the selected objects belong. Qualification part  $Q_C$  indicates predicates that the selected objects must satisfy. Target part A indicates the attributes that the result values come from. But compared with RDB counterparts, each part in OODB query has new mechanisms. Range part indicates one class or a set of classes in a class hierarchy. Qualification part may contain new kinds of operands and operators. Target part can be an attribute of the given class in range part or one attribute of a class that is derived by navigating through the class composition hierarchy rooted at the given class.

For query translation, we map object query operation to relation query operations and select the tuples corresponding to the objects qualified by  $Q_C$  from the appropriate relations mapped from indicated class or the class with all of its subclasses. The selected tuples are filtered by relational qualification  $Q_R$ , mapped from  $Q_C$ , and also from path expressions in target part of object query operation.

#### 3.4.1. Translation mapping

The general form of target part A is expressed as  $A_{C_0} \cdot A_{C_1} \cdot A_{C_2} \dots \dots \dots \cdot A_{C_n}$ , where  $n \geq 0$ ,  $C=C_0$  when  $n=0$ , and  $(DOM(A_{C_i})=C_{i+1}, 0 \leq i \leq n-1)$  when  $n > 0$ . The values of  $A_{C_n}$  of the selected objects are the result of query. The mapping  $\rho_Q$  are:

$\rho_Q(S_o(C, Q_C, A_{C_0} \cdot A_{C_1} \cdot A_{C_2} \dots \dots \dots \cdot A_{C_n})) \rightarrow S_r(R, Q_R, A_R)$ , where

- the mapped range part is:

$$R = \begin{cases} R_{C_n} & \text{DOM}(A_{C_n}) \neq T^{set} \\ R_{A_{C_n}} & \text{DOM}(A_{C_n}) = T^{set} \end{cases}$$

- the mapped qualification part is:

$$Q_R = \begin{cases} \rho_{QUA}(Q_C) \cup \left( \bigcup_{i=0}^{n-1} \{(R_{C_i} \cdot A_{R_{C_i}} = R_{C_{i+1}} \cdot A_{R_{C_{i+1}}}^{key})\} \right), & \text{DOM}(A_{C_n}) \neq T^{set} \\ \rho_{QUA}(Q_C) \cup \left( \bigcup_{i=0}^{n-1} \{(R_{C_i} \cdot A_{R_{C_i}} = R_{C_{i+1}} \cdot A_{R_{C_{i+1}}}^{key})\} \right) \cup \{(R_{C_n} \cdot A_{R_{C_n}} = R_{A_{C_n}} \cdot A_{R_{A_{C_n}}}^1)\}, & \text{DOM}(A_{C_n}) = T^{set} \end{cases}$$

•the mapped target part is:

$$A_R = \begin{cases} A_{R_{C_n}}, & \text{DOM}(A_{C_n}) \neq T^{set} \\ A^2_{R_{A_{C_n}}}, & \text{DOM}(A_{C_n}) = T^{set} \end{cases}$$

### 3.4.2. Translation algorithm

Based on the mapping  $\rho_Q$ , algorithm for scheme translation is:

query\_translation1( $C, Q_C, A_{C_0} \cdot A_{C_1} \cdot A_{C_2} \dots \dots \cdot A_{C_n}, \rho_S$ )

//input:  $S_0(C, Q_C, A_{C_0} \cdot A_{C_1} \cdot A_{C_2} \dots \dots \cdot A_{C_n})$  and  $\rho_S$

//output:  $S_i(R, Q_R, A_R)$

```
{
  R ← ρS(Cn);
  QR = qualification_translation(QC, ρS);
  for ( i=0; I<n; i=i+1 )
    QR = QR ∪ {(RCi · ARCi = RCi+1 · ARCi+1key)};
  if (DOM(ACn) ≠ Tset)
    AR = ARCn;
  else
    {
      QR = QR ∪ {(RCn · ARCn = RACn · A1RACn)};
      AR = A2RACn;
    }
}
```

For inference, the object query operation is represented as  $S_0(C^{all}, Q_C, A)$  which indicates that the range part are all classes in the class hierarchy having class C as superclass. In this case we have:

$$S_0(C^{all}, Q_C, A) = \bigcup_{C^i \in \{C\} \cup SUB(C)} S_0(C^i, Q_{C^i}, A), \text{ and}$$

query\_translation2( $C^{all}, Q_C, A, \rho_S$ )

```
{
  for ( every Ci ∈ {C} ∪ SUB(C) ) do
    query_translation1(Ci, QCi, A, ρS);
}
```

### 3.4.3. Example

According to the translation algorithm we translate object query:

$$S_0(S5, Q_{S5}, A9.A7)$$

to relation query:

$$S_r(S4, (S5.A1 < 10) \wedge (S5.S3\_OID = S3.S3\_OID) \wedge (S5.S4\_OID = S4.S4\_OID) \wedge \\ (S3.S2\_OID = S2.S2\_OID) \wedge (S4.S2\_OID = S2.S2\_OID) \wedge \\ (S4.A6\_SET = S4A6.A6\_SET) \wedge (S2.A2 > 0) \wedge (S2.A3 = 'abc') \wedge \\ (S2.A4 = 5.5) \wedge ('bcd' \text{ in } S4A6), A7)$$

### 3.5. Update Translation

For update translation, we map object update operation to relational update operations. The process updates the values of the indicated attribute of the tuples qualified by  $Q_R$  mapped from  $Q_C$  in object update operation from the appropriate relations mapped from indicated class or the class with its subclasses. According to scheme mapping, the elements of a set value are stored in a relation, and different value of a set attribute may have different elements. This means different set values corresponding to different tuples in a relation. Thus, we may need to delete some tuples if the elements of the old value are more than the new one. Otherwise we may need to insert some tuples. Our approach is to delete all tuples of old value and then insert tuples for the new value.

#### 3.5.1. Translation mapping

Based on the scheme translation, the update operation mapping is:

$$\rho_U(U_o(C, Q_C, A_C, v_C)) \rightarrow \begin{cases} U_r(R_C, Q_R, A_{R_C}, v_{R_C}), & \text{DOM}(A_C) \neq T^{set} \\ D_r(R_{A_C}, Q'_R), \\ U_r(R_C, Q_R, A_{R_C}, v_{R_C}) \text{ and} \\ \{I_r(R_{A_C}, \{oid_{R_{A_C}}, v^j_{R_{A_C}}\}) \mid 1 \leq j \leq |v_C|\}, & \text{DOM}(A_C) = T^{set} \end{cases}$$

#### 3.5.2. Translation algorithm

Based on the mapping  $\rho_U$ , algorithm for scheme translation is:

```

update_translation1(C, Q_C, A_C, v_C, ρ_S)
// input: U_o(C, Q_C, A_C, v_C) and ρ_S
// output : U_r(R_C, Q_R, A_{R_C}, v_{R_C}) and
//          D_r(R_{A_C}, Q'_R) and {I_r(R_{A_C}, {oid_{R_{A_C}}, v^j_{R_{A_C}}}) \mid 1 \leq j \leq |v_C|\}
{
  R_C ← ρ_S(C);
  Q_R = qualification_translation(Q_C, ρ_S);
  A_{R_C} ← ρ_S(A_C);
  if (DOM(A_C) ≠ T^{set})
    v_{R_C} = v_C;
  else
    {
      if (|v_C| = 0)
        v_{R_C} = null;
      else

```

```

{
  vRC = oid_generator();
  RAC ← ρS( AC );
  Q'R = QR ∪ { ( RC · ARC = RAC · A1RAC ) };
  for ( j = 1; j ≤ |vC|; j = j+1 )           // vC should be a set of values //
    vjRAC = vjC;                       // vjC is the jth element of set vC //
}
}
}

```

As for inheritance, the object update operation is represented as  $U_o(C^{all}, Q_C, A_C, v_C)$  which indicates the objects to be updated belong to all classes in the class hierarchy having class C as superclass. In this case, we have:

$$U_o(C^{all}, Q_C, A_C, v_C) = \{ U_o(C^i, Q_{C^i}, A_{C^i}, v_{C^i}) \mid (C^i \in \{C\} \cup SUB(C)) \& (0 \leq i \leq |SUB(C)|) \& (C^0 = C) \& (A_{C^i} \in ATT(C^i)) \}, \text{ and}$$

```

update_translation2( Call, QC, AC, vC, ρS )
{
  for ( every Ci ∈ {C} ∪ SUB(C) ) do
    update_translation1( Ci, QC, AC, vC, ρS );
}

```

### 3.5.3 Example

According to the translation algorithm, we translate object update operation:

$$U_o( S4, S4.A1 > 0, A6, \{ 'uvw', 'xyz' \} )$$

to the following relation operations:

$$U_r( S4, S4.A1 > 0, A6\_SET, a6\text{-oid-n} ), \\ D_r( S4A6, S4.A1 > 0 \wedge S4.A6\_SET = S4A6.A6 ) \text{ and} \\ \{ I_r( S4A6, \{ a6\text{-oid-n}, 'uvw' \} ), I_r( S4A6, \{ a6\text{-oid-n}, 'xyz' \} ) \}$$

## 3.6. Deletion Translation

For deletion translation, we map object deletion operation to relational deletion operations by deleting the tuples filtered by qualification  $Q_R$  mapped from  $Q_C$  in object deletion operation from the appropriate relations mapped from indicated class or the class with its subclasses. If the objects to be deleted contain set values, all the tuples corresponding to those set values must be deleted from the related relations mapped from the set attributes. The deleted objects may be components of other composite objects. Thus, we need to set null values to the foreign key attributes in the tuples corresponding to the composite objects.

### 3.6.1. Translation mapping

Based on the scheme translation, the deletion operation mapping is:

$$\rho_D(D_o(C, Q_C)) \rightarrow \{ D_r( R_{A^i_C}, Q^i_R ) \mid (DOM(A^i_C) = T^{set}) \& (1 \leq i \leq |ATT(C)|) \},$$

$$\{U_i(R_{C^j}, Q^j_R, A_{R_{C^j}}, \text{null}) | \text{DOM}(A_{C^j})=C\} \text{ and } D_i(R_C, Q_R)$$

### 3.6.2. Translation algorithm

Based on the mapping  $\rho_D$  algorithm, scheme translation is:

```

deletion_translation1( C, Q_C, \rho_S )
// input: D_o(C, Q_C) and \rho_S
// output: { D_i(R_{A^i_C}, Q^i_R) | (DOM(A^i_C)=T^{set}) & (1 \le i \le |ATT(C)|) },
//      { U_i(R_{C^j}, Q^j_R, A_{R_{C^j}}, \text{null}) | \text{DOM}(A_{C^j})=C } and D_i(R_C, Q_R)
{
  Q_R = qualification_translation(Q_C, \rho_S);
  for ( i=1; i \le |ATT(C)|; i=i+1 )
    if (DOM(A^i_C)=T^{set})
      {
        R_{A^i_C} \leftarrow \rho_S(A^i_C);
        Q^i_R = Q_R \cup \{ (R_C \cdot A_{R_C} = R_{A^i_C} \cdot A^1_{R_{A^i_C}}) \}
      }
  R_C \leftarrow \rho_S(C);
  for ( every C^j \in C )
    for ( every A_{C^j} \in ATT(C^j) )
      if (DOM(A_{C^j})=C)
        {
          R_{C^j} \leftarrow \rho_S(C^j);
          A_{R_{C^j}} = A_{R_C}^{key};
          Q^j_R = Q_R \cup \{ (R_C \cdot A_{R_C}^{key} = R_{C^j} \cdot A_{R_{C^j}}) \}
        }
}

```

As for inheritance, the object delete operation can be represented as  $D_o(C^{all}, Q_C)$  which indicates the objects to be deleted belong to all classes in the class hierarchy having class C as superclass. In this case, we have:

$$D_o(C^{all}, Q_C) = \{D_o(C^k, Q_{C^k}) | (C^k \in \{C\} \cup \text{SUB}(C)) \& (0 \le k \le |\text{SUB}(C)|) \& (C^0 = C)\}, \text{ and}$$

```

deletion_translation2( C^{all}, Q_{C^i}, \rho_S )
{
  for ( every C^k \in \{C\} \cup \text{SUB}(C) ) do
    deletion_translation1( C^k, Q_{C^k}, \rho_S );
}

```

### 3.6.3. Example

According to the translation algorithm, we translate object delete operation:

$$D_o(S4, S4.A1 > 0)$$

to the following relation operations:

$$\{ D_r(S4A6, S4.A1 > 0 \wedge S4.A6\_SET = S4A6.A6\_SET) \},$$
$$\{ U_r(S5, S4.A1 > 0 \wedge S4.S4\_OID = S5.S4\_OID, S4\_OID, null) \} \text{ and}$$
$$D_r(S4, S4.A1 > 0)$$

### 4. Case study

Suppose we have the source object-oriented operations written in UniSQL<sup>13</sup> corresponding to our OODB operations as follows:

1. create class DEPT(dept\_no: integer, dept\_name: string);
2. create class STUDENT(student\_no: integer, name: string, dept: DEPT, course\_taken: set(string));
3. insert into DEPT  
values(12, "Computer Science");
4. insert into STUDENT  
values(2345, "Tom Wang", oid2, {"data structure", "database system"});
5. update STUDENT  
set course\_taken={"Java", "compiler"}  
where dept.dept\_name="Computer Science";
6. delete from STUDENT where name="John Smith";
7. delete from DEPT where dept\_name="Mathematics";

After translation, we get the following resultant relation operations in a SQL-like format corresponding to our relation operation:

1. create table DEPT(dept\_oid: integer, dept\_no: integer, dept\_name: string);
2. create table STUDENT(student\_oid: integer, name: string, student\_no: integer, dept\_oid: integer, course\_taken\_set: integer);  
  
create table STUDENTCOURSE\_TAKEN(course\_taken\_set: integer, course\_taken: string);
3. insert into DEPT values(4, 12, "Computer Science");
4. insert into STUDENT values(5, "Tom Wang", 2345, 4, 6);  
insert into STUDENCOURSE\_TAKEN values( 6, "data structure");  
insert into STUDENCOURSE\_TAKEN values( 6, "database system");

```

5. delete from STUDENTCOURSE_TAKEN
   where STUDENTCOURSE_TAKEN.course_taken_set in
      ( select STUDENTCOURSE_TAKEN.course_taken_set
        from STUDENTCOURSE_TAKEN, STUDENT, DEPT
        where STUDENT.dept_oid=DEPT.dept_oid
          and DEPT.dept_name="Computer Science"
          and
STUDENT.course_taken_set=STUDENTCOURSE_TAKEN.course_taken_set );

update STUDENT
  set course_taken_set=7
  where STUDENT.student_oid in
     ( select STUDENT.student_oid
       from STUDENT, DEPT
       where STUDENT.dept_oid=DEPT.dept_oid
         and DEPT.dept_name="Computer Science" );
insert into STUDENCOURSE_TAKEN values( 7, "Java");
insert into STUDENCOURSE_TAKEN values( 7, "compiler");

6. delete from STUDENTCOURSE_TAKEN
   where STUDENTCOURSE_TAKEN.course_taken_set in
      ( select STUDENTCOURSE_TAKEN.course_taken_set
        from STUDENTCOURSE_TAKEN, STUDENT
        where STUDENT.name="John Smith"
          and STUDENT.course_taken_set=STUDENTCOURSE_TAKEN.course_taken_set );

delete from STUDENT
  where STUDENT.name="John Smith";

7. update STUDENT
  set STUDENT.dept_oid=null;
  where STUDENT.student_oid in
     ( select STUDENT.student_oid
       from STUDENT, DEPT
       where DEPT.dept_name="mathematics"
         and DEPT.dept_oid=STUDENT.dept_oid );
delete from DEPT
  where DEPT.dept_name="Mathematics";

```

## 5. Conclusion

Transaction translation between OODB and RDB plays an important role in the multidatabase interoperability. In this paper, we show methods of mappings and algorithms which translate transactions of OODB to transactions of RDB. The discussion aims at the translating functions in OODB operation which normally are not supported by RDB operations. In transactions translation, the class hierarchy semantics in OODB are mapped to relational union operation. The semantics of navigation through composition hierarchy are mapped to relational join operation, and set operations are mapped to relevant relation operations. To map multiple path expressions originated from one class in a qualification of an object operation, we use graph to model and process the mapping. For the future research of this paper, we will include the PSM<sup>14</sup> routine (cover both user-defined functions and procedures) in the transaction translation processing.

**References:**

1. A. P. Sheth, and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, Vol. 22, No.3, Sept. 1990, P183-236.
2. Xiuzhen Zhang and Joseph Fong, "Translating update operations from relational to object-oriented databases", *Journal of Information and Software Technology*, Volume 42, Number 3, March, 2000.
3. Joseph Fong and Paul Chitson, "Query Translation from SQL to OQL for Database Reengineering", *International Journal of Information Technology*, vol. 3, No. 1, June 1997, pp. 83-101.
4. W. Meng et al., "Construction of a Relational Front-end for Object-Oriented Database Systems", *Proc. of IEEE Data Engineering*, 1993, P476-483.
5. D. Keim, H. P. Kriegel, and A. Miethsam, "Object-Oriented Querying of Existing Relational Database", *Proc. of the 4th Intl. Conference on Database and Expert System Applications(DEXA'93 )*, Sept. 1993.
6. C. Yu et al., "Translation of Object-Oriented Queries to Relational Queries", *IEEE proceedings of the 11th Intl. Conference On Data Engineering*, 1995, P90-97.
7. E. Bertino, M. Negri, G. Pelagatti, and L. Sbatella, "Object\_Oriented Query Languages: The Notion and the Issues", *IEEE Trans. on Knowledge and Data Engineering*, Vol.4, No.3, June 1992, P223-237.
8. V. M. Markowitz, A. Shoshani, "Object Queries over Relational Database: Language, Implementation, and Applications", *Proc. of IEEE Data Engineering*, 1993, P71-80.
9. Q. Xiaolei and R. Louiqa, "Query Interoperation Among Object-Oriented and Relational Databases", *IEEE proceedings of the 11th Intl. Conference On Data Engineering*, 1995, P271-278.
10. M. Blaha, W. Premerlani, and H. Shen, "Converting OO Models into RDBMS Schema". *IEEE Software*, Vo.1, No.1, May 1994, P28-39.
11. J. G. Hughes, "Object-Oriented Databases", *Prentice Hall Intl.*, 1991.
12. D. Maier, "The Theory of Relational Database", *Computer Science Press*, 1983.
13. "UniSQL/X User's Manual", *UniSQL, Inc.*
14. C.J.Date with Hugh Darwen, "A Guide to the SQL Standard", *Addison-Wesley Publishing Company*, 1993.