

Translating Update Operations from Relational to Object-Oriented Databases

Xiuzhen Zhang, Joseph Fong

Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

email: {xzhang@cs, csjfong@}cityu.edu.hk

Abstract In migrating a legacy relational database system to object-oriented(OO) platform, when database migration completes, application modules are to be migrated, where embedded relational database operations are mapped into their OO correspondents. In this paper we study mapping relational update operations to their OO equivalents, which include UPDATE*, INSERT and DELETE operations. Relational update operation translation from relational to OO faces the touchy problem of transformation from a value-based relationship model to a reference-based model and maintaining the relational integrity constraints. Moreover, with a relational database where inheritance is expressed as attribute value subset relationship, changing of some attribute values may lead to the change of the position of an object in the class inheritance hierarchy, which we call object migration. Considering all these aspects, algorithms are given mapping relational UPDATE, INSERT and DELET operations to their OO correspondents. Our work emphasize in examining the differences in the representation of the source schema's semantics resulting from the translation process, as well as differences in the inherent semantics of the two models.

Key Words: Relational model, OO model, Query translation, Update translation

1 Introduction

Migrating a legacy database system consists of migrating the database and migrating the application programs. With migrating a relational database system to OO platform, we consider a methodology where database is first migrated. In the migration process, a database gateway is necessary providing flexibility for timing of application migration with and target database. Both the database gateway and the application program(with embedded database operations) migration make it necessary to study mapping relational database operations to their OO correspondents.

Database operations include query and update operations. In this paper, we will consider update operation translation from relational to OO model. While query translation from relational to OO

* Note that we use *update* for general update operations and UPDATE for modifying certain data values.

model for database system migration has been discussed[1], there hasn't been work reported on update operation translation. Given an update operation U against a relational view, there exist more than one database update corresponding to U . So relational view update operation translation focuses on choosing the most suitable database update operation corresponding to a view update operation from the candidates.

Considering the schema mapping between relational and OO model, we examine translating relational and update operations into corresponding OO operations. RDB to OODB schema mapping ρ , which is injective and surjective, defines the 1:1 correspondence between database states. So given a relational Q query whose target data is δ , we can form the OO correspondent W of Q whose target data is $\gamma = \rho(\delta)$. Moreover, given an update operation $U: U(S)=T$, where S and T are the relational database states before and after U respectively. Suppose that $\rho(S)=X$, $\rho(T)=Y$, the OO operation(s) corresponding to U is V which changes the OODB from state X to state Y : $V(X)=Y$.

For exposition, we will use SQL[2] expressing relational database operations and OQL and C++ OML defined by ODMG 2.0[3] for OODB operations. A surjective and injective mapping between relational and OO constructs is the precondition for update operation translation from relational to OO model.

In the next section, we'll discuss related work.. Section 3 describes mapping relational to OO schema. In section 4, we propose the object migration problem in update operation translation. Section 5 gives the algorithms mapping relational UPDATE, INSERT and DELETE operations to OO operations. Section 6 is the prototype. Section 7 is the final remark.

2 Related Work and Contribution

Research efforts relating to our work come from two areas: The work on multidatabase provides interoperability between data sources of different data models or provide an interface of certain data model while the underlying database is of another database. This work focuses on translating queries between relational and OO models only[4][5][6][7]. Little is addressed about the update operation translation between different data models. The second kind of related work comes from the examination of translating relational view update operations and has been discussed by several researchers[8][9][10].

The research activities along the two lines, however, have quite different motivations and condition than our problem. With the 1st kind of work, only query translation between existing databases is

considered and uniform and simple mappings between database constructs are assumed. Relational views are defined by queries on the underlying database.

The provision of a relational front-end to different database models has been discussed by several authors [11][12]. In these works the relational schema being queried is created using a set of simple syntactic mapping rules from the target non-relational schema. The simplicity of these rules makes the translation task much easier. In database reengineering however, the target schema is derived from the source relational schema through a process called ‘Schema Translation’[13][14][15].The schema translation process requires user interaction to recapture and re-express the semantics of the Domain of Discourse. Generally the result of the schema translation process will not be describable as the rigorous application of a small set of simple syntactic transformations; and the data manipulation language(DML) translation process must be changed.

The task of database update translation can be described as follows. Given a source update U the task is to find an informational equivalent update U' over the new target database. For U' to be deemed informational equivalent it must be possible to show that (see also Figure 1):

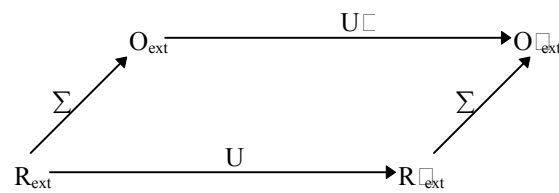


Figure 1 The Update Translation Problem

Given, $\Sigma(R_{ext}) = O_{ext}$

where R_{ext} and O_{ext} are the relational and object oriented database extensions,

and Σ is the information preserving instance level schema transformation.

that, $\Sigma(U(R_{ext})) = U'(O_{ext})$

where $U(R_{ext})$ is the relational database extension after the application of update U ,

U' is the translation of update U , expressed over the object database,

and $U'(O_{ext})$ is the object oriented database extension after the application of update U' .

It has shown by others, that for this statement to be true the schema transformation Σ must be reversible (total, onto, 1:1) [16]. Therefore, before we can specify a method for update translation we must be able to make the assumption that the schema translation process is carried out in such a fashion that its results are reversible. Fortunately, research in schema translation has provided us with the very vehicle to meet such a rigorous requirement, ‘‘Database Transformations’’ [2][13][14][15].

From the available literature on database transformation, several useful reversible transformations have been documented[13][15]. Through application of these transformation operators it is possible to conclude that, (1) the database schema has an equivalent information capacity, and (2) that given an equally expressive database language that an equivalent database update exists. In the following we assume that a user has used such a method in deriving the source schema, therefore an equivalent database update is guaranteed to exist.

The effects of database updates on the position of an object in the class lattice have been studied by various authors [17][18]. Predicates are used to determine class membership (this predicate positioning method is similar to the taxonomic reasoning method presented in [19]). If an object satisfies the predicates of a class it is considered a member of that class. An object in this method can be a member of only one class at a time, any update violating this principle is considered to be unsafe (a relaxation of this restriction and a simple set of migration operators can be found in [18]). Updates that cause an existing object to change its membership represent a potential migration. The update is accepted and the migration occurs if and only if a migration path has been specified between the source and destination classes. We have adapted the work discussed in [17][18][19] to construct methods suitable for the placement and migration of database entities resulting from the execution of a database update expressions.

In this paper we look at the process of update translation between the relational data model and the object-oriented data model for database reengineering. Unlike previous work on the provision of a relational interface to an object-oriented database system, the schema of the object-oriented database is a translation of the relational database schema. Our contribution is that during the process of schema translation the semantics of relational schema are reformulated to better describe the semantics of the domain of discourse. Such changes provide additional complexity to the translation process yet are necessary if we are to provide the freedom for redesign in the reengineering process.

3 Migrating Relational to OO Databases

In this section, we examine migrating a legacy relational database to OO platform. This process consists of two steps: First the relational schema is mapped into an OO schema, then data is migrated to the OO platform according to schema mapping.

3.1 Mapping RDB Schema into OODB Schema

Mapping relational to OO schema consists of two parts: Restructuring the relational structure to form a well-defined OO static structure and deriving the dynamic method definitions for classes. We will

discuss here how to construct the static structure of the OO schema. However, we don't intend to propose new schema mapping method but only summarize the commonly recognized schema mapping techniques as the base for our discussion of database operation translation.

A lot of work has been reported on mapping relational to OO schema. Most work considers only functional dependency(FD) and inclusion dependency(IND) and assumes a relational schema with only key FDs(and so in BCNF) and key-based INDS(foreign key) as the canonical form for mapping into OO schema. However, to further remove the redundancy caused by multi-valued dependency(MVD), we further require that the relational schema is in 4NF. We consider a 4NF relational schema with foreign keys as the canonical form for mapping into OO schema. Given a relational schema, considering the FDs and MVDs, it should first be mapped into 4NF before being mapped into OO schema.

When the meaning is clear from the context, relation schemes are simply called relations. A canonical RDB schema must also satisfy the following conditions: No two relation schemes are the same. That is, we don't support that a relation is partitioned horizontally. Given a foreign key F of $R: R[F] \subseteq S[K]$, then there doesn't exist any other relation S' where $R[F] \subseteq S'[K']$ and $S'[K'] \subseteq S[K]$ and S is called the host relation(scheme) of F . A foreign key can have more than one host relation. Foreign keys are either disjoint or the same (This restriction means that the relational schema can only be ER - designed). Attributes which are neither prime nor involve any foreign key are called local attributes.

Our OO schema conforms to the ODMG 2.0 object model. With our schema mapping defined below, the resultant OODB schema is a typical OODB schema with class inheritance hierarchy(simple class hierarchy). Without considering the system-defined abstract class OBJECT, an OODB schema consists of semantic class hierarchies. A semantic class which is not a subclass of any other semantic classes is called a base class. The static structure of a class is defined by attributes and relationships. Given a class C , an attribute A of C can be an atomic attribute, whose value is of literal type, or complex attribute, whose value is an object identifier of some other class C' and C' is called a component class of C . A relationship is defined between two classes and both take as values object identifiers from the opposite class. An attribute or relationship can also be single-valued or set-valued. The resulting OODB schema is without data redundancy.

Given a canonical RDB schema $\langle R, A \rangle$, and a relation scheme $R \in R, R = K \cup A$, where K and A are subsets of attributes and K is the primary key of R , depending on the structure of K , foreign keys in K are uniquely identified as inheritance, aggregation and relationship and R is uniquely mapped into a base class, a subclass, a binary relationship between two classes or a component of another class:

1) K is a foreign key: $R[K] \subseteq R1[K1]$. This foreign key means inheritance: Suppose $R1$ is mapped into class $C1$, then R is mapped into a subclass C of $C1$:

Class C ISA $C1$ {Definitions_for_Attributes_in_A}

R is called a *subclass relation(SCR)*. C inherits the key K from $C1$.

2) Part of K is a foreign key: $K' \subset K$ and $R[K'] \subseteq R1[K1]$. In ER terms, R corresponds to a weak entity. In OO terms, R represents a component of class $C1$, which is mapped from $R1$. An object of $C1$ (identified by the value of K) aggregates several instances of R . Two cases are distinguished:

- If $A = \emptyset$, $|K - K'| = 1$, $K - K' = \{A'\}$ and R is not the r.h.s of any IND, then R is mapped into a set-valued attribute A' of $C1$ which is literal typed:

Class $C1$ {attribute ...; attribute set <TypeOfA> A }

R is called a *set-valued attribute relation(SAR)*.

- Otherwise R is mapped into class C with K as the key and C is a component class of $C1$:

class $C1$ {attribute ...; attribute set < $C1$ > $A1$ };

class C (keys K) {Attribute_Defs KA };

R is called a *component class relation(CCR)*.

3) $K = K1' \cup K2'$, $K1' \cap K2' = \emptyset$, $R[K1'] \subseteq R1[K1]$, $R[K2'] \subseteq R2[K2]$, $A = \emptyset$ and R doesn't appear on the r.h.s. of any IND. R is a binary relationship between the two entities identified by $K1$ and $K2$ and is called a *Binary Relationship Relation(BRR)*. Suppose that $R1$ and $R2$ are mapped into classes $C1$ and $C2$ respectively. R is mapped into a binary relationship between $C1$ and $C2$:

class $C1$ {... , relationship [Set] < $C2$ > $A1$ inverse $C2::A2$ };

class $C2$ {..., relationship [Set] < $C1$ > $A2$ inverse $C1:A1$ };

Note that in the above notation, depending on the cardinality of the relationship, $A1(A2)$ may be set-valued.

4) Otherwise R is mapped into a class C. In ER terms, relations that fall into this category include relations that represent

- entity where K doesn't contain any foreign key. R is called a *entity class relations(ECR)*. K becomes the key of C and the local attributes of R are mapped into those of C.
- binary relationship with local attributes whose key structure is the same as that in 3) but $A \neq \emptyset$ or n-ary relationship($n > 2$) with or without local attributes. R is called a relationship class relation(RCR). An object in C aggregates objects of the involving classes.

Foreign keys of SCR, CCR, ECR and RCR that are disjoint with primary keys are mapped into relationships between the corresponding classes. Also note that in case 2) the existence dependency of a component object of C on a composite object C1 is not automatically supported by the ODMG data model and the application should enforce it. Together with mapping relational constructs to OO static structure, methods are established on classes to maintain the key of classes. A constructor is created for each class which create objects of the class with all the property values supplied.

Example 1 A sample relational schema is given below and used for our later discussions.

Person(<u>ssn</u> , name, drive)	Person[drive] \subseteq Car[license#]
Per_Hobby(<u>ssn</u> , hobby)	Staff[teach] \subseteq Course[cno]
Staff(<u>ssn</u> , staff_id, title, teach)	Staff[ssn] \subseteq Person[ssn]
Course(<u>cno</u> , title)	Person_Dept[ssn] \subseteq Person[ssn]
Car(<u>license#</u> , model)	Person_Dept[code] \subseteq Department[code]
Department(<u>code</u> , location)	
Person_Dept(<u>ssn</u> , <u>code</u>)	

Primary keys are underlined. Under the schema mapping discussed, *Person*, *Course*, *Car*, and *Department* are ECRs, *Per_Hobby* is a SVR, *Staff* is a SCR. *Person_Dept* is a BRR. The mapped OO schema is as below.

```
class Person (integer ssn, string name, set(string) hobby, Department dept)
class Staff : Person(integer staff_id, string title, set(Course) teach);
class Course(integer cno, string title, Staff taught_by);
class Car( integer license#, string model);
class Department( string code, string location, set(Staff) has )
```

3.2 Data Migration

According to the schema mapping, relational data should be reorganized under the OO schema. Relational schema uses attribute values to represent the association between data. In the OO model, data values are organized into objects and objects are related by direct object identifier references. Non-foreign key attribute values become literal attribute values of objects or elements of collections. Foreign key values establish the relationship between objects.

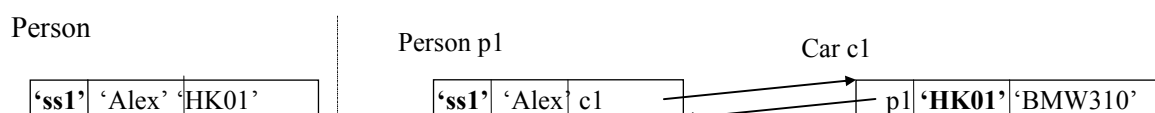
Migrating data from the relational to OO platform consists of unloading relational data, restructuring and uploading data to the OO platform. We will not go into the detail process but only give the data migration rules so that we have an overall picture of data migration[20]. We should first populate those classes which are mapped from ECR, and then subclasses, component classes, classes mapped from RCR and finally establish the relationship between classes.

A problem that deserves special attention is that in reorganizing data, data of a relation $r(R)$ may not necessarily become data of its corresponding class C , but may also be mapped into data of C 's subclasses. This is due to the difference for representing inheritance: In the relational model, both structure and data are inherited. In the OO model, however, only structure is inherited. Where data should be placed in the class hierarchy depends on the lowest position of the key in the IND graph. This also causes the object migration problem in update operation translation, which is detailed in section 4. When database migration completes, a table is formed recording the schema mapping information. For a given relation scheme, its type, mapped OO construct etc are recorded.

4 Object Migration in Class Hierarchy

As stated before, in the relational model, semantic association between data is expressed as constraints between data values. To be more specific, foreign keys are used to express inheritance, aggregation and relationship between entities (An entity consists of a set of attributes and is the basic unit for organizing information). In the OO model, attribute values are organized into objects and the semantic associations between objects, inheritance, relationship or aggregation, are directly expressed as inheritance or references between objects by OIDs. In the relational model, update operations insert new data value, delete existing data value or modify data values. Semantically such treatment may really mean corresponding operations on attribute values, but it may also mean treatment to the association between values.

Example 2 Referring back to the relational schema in example 1 with the sample data shown on the left, according to the schema mapping, the corresponding OO database state is shown on the right



Car

'HK01'	'BMW310'
--------	----------

Course

'cs01'	'Algorithms'
--------	--------------

Staff

--	--

Course cs1

'cs01'	'Algorithms'
--------	--------------

When we insert a new tuple into relation *Staff*:

```
insert into Staff values('ss1', 's1', 'AssProf', 'cs01');
```

The foreign key $Staff[ssn] \subseteq Person[ssn]$ is interpreted as inheritance. Insertion of this new tuple into *Staff* means that both the attributes and their values of the tuple in *Person* are inherited. On the OO side, as structural inheritance is directly supported, but data is directly organized under the class structure, the corresponding database state is:

Person

'ss1'	'Alex'	'HK01'
-------	--------	--------

Car

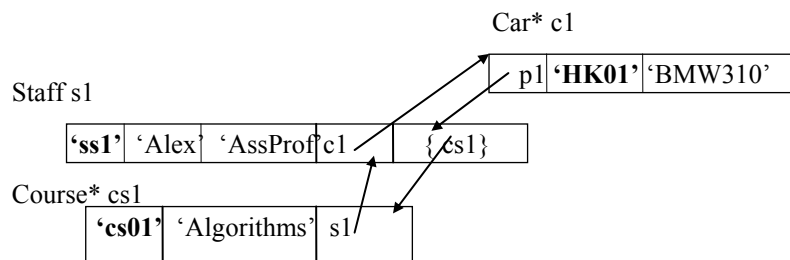
'HK01'	'BMW310'
--------	----------

Course

'cs01'	'Algorithms'
--------	--------------

Staff

'ss1'	'AssProf'	
-------	-----------	--



To bring the OO database from the previous state to the current state, a set of operations is needed, of which the most prominent one that the object *p1* of class *Person* should **migrate to** class *Staff* and becomes object *s1* with the additional data values supplied.

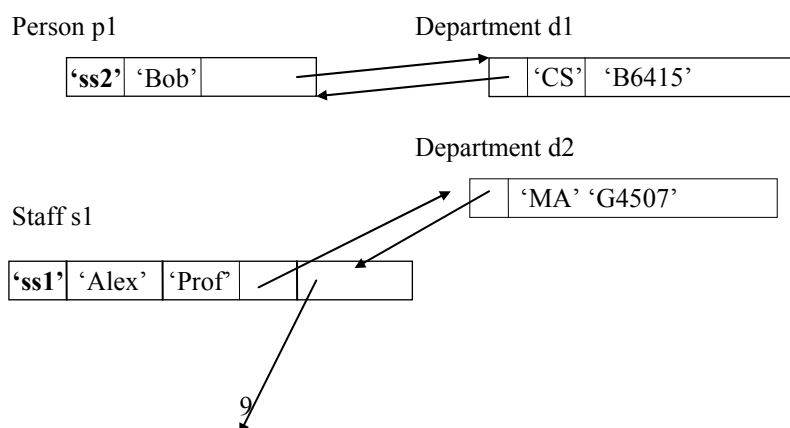
Example 3 Consider a more complicated example. Referring to example 1, given consistent RDB and ODB states shown below

Person

'ss1'	'Alex'
'ss2'	'Bob'

Staff

'ss1'	'Prof'	'cs01'
-------	--------	--------



Department

'CS'	'B6415'
'MA'	'G4507'

Course cs1

'cs01'	'Algorithms'	
--------	--------------	--

Course

'cs01'	'Algorithms'
--------	--------------

Person-Dept

'ss1'	'MA'
'ss2'	'CS'

An UPDATE operation on the relational database

UPDATE Staff SET ssn='ss2' WHERE ssn='ss1'

Person

'ss1'	'Alex'	'MA'
'ss2'	'Bob'	'CS'

Staff

'ss2'	'Prof'	'cs01'
-------	--------	--------

Department

'CS'	'B6415'
'MA'	'G4507'

Course

'cs01'	'Algorithms'
--------	--------------

Person_Dept

'ss1'	'MA'
'ss2'	'CS'

Person p2

'ss1'	'Alex'	
-------	--------	--

Department d1

'CS'	'B6415'
------	---------

Department d2

'MA'	'G4507'
------	---------

Staff s2

'ss2'	'Bob'	'Prof'	
-------	-------	--------	--

Course cs1

'cs01'	'Algorithms'	
--------	--------------	--

The relational database state after the UPDATE operation is shown on the left. According to the mapping defined between the relational and OO schemas, the corresponding ODB state is shown on the right. To bring the previous ODB state to the current state, object *p1* of class *Person* **migrates to** class *Staff* and becomes object *s2*; On the other hand, object *s1* of class *Staff* **migrates to** class *Person* and gain a new identity *p2*. With this example we also want to show that when an object is migrated, although its membership class is changed, its relationship with other classes is retained. In the above example, when *p1* is migrated from class *Person* to class *Staff*, its bidirectional relationship with object *d1* is kept and so is the case with the migration of object *s1*.

The problem of object migration has been examined by several researchers[17][21]. The actual migration operation has been discussed by Li and Dong[18] in some detail. Three goals for migration are defined: identity preservation, behavior transition and information conservation. However, most current available ODBMSs are statically-typed systems and dynamic migration of objects between objects is strictly prohibited. In this respect, we implement the object migration function *migrate()* with available object operations. In a statically-typed ODBMS, the *migrate()* function assign the object a new OID. Depending on the direction of the movement, from superclass to subclass or vice versa, the object keep its information and behavior or not.

According to the ODMG object model, we will make the following assumptions about objects in the object base. Each object has a lifetime identifier(OID) to identify itself. Names are designated by programmers to refer to particular objects. Moreover, the ODL interface provides *constructor* and *destructor* to create and delete objects explicitly, which is implicitly inherited by all user-defined object definitions. We adopt a Java-like syntax for constructor and method *delete()* for destructor. For example, with *Person p= new Person()* we declare an object p of class Person and create an object named p uninstantiated. Whereas with *p.delete()* object p is deleted.

With the basic operations, the function *migrate(x, C, C1)* migrate object x, initially a member of class C, to an instance of class C1. If x is not in class C, return a null object, otherwise if migration succeeds, return the new OID for the object.

Begin

```

if x is not an instance of C  return (C1) null;
C1 p=new C1();
if(C1 is a subclass of C)
    for each attribute ai of C  p.ai=x.ai;
else // C1 is a superclass of C
    for each attribute ai of C1 p.ai=x.ai;
Redirect all references to x to p;
x.delete();
return(p);

```

End

The object operations corresponding to the relational update operation in example 2 are shown below

<pre> Select p1 from Person p1 where ssn='ss2'; Staff s2=migrate(p1, Person, Staff) s2.title=s1.title; s2.teach=s1.teach; </pre>	<pre> //migrate(p1, Person, s2, Staff) Staff s2=new Staff(); s2.ssn=p1.ssn; s2.name=p1.name; </pre>
--	---

p1.dept.people=p1.dept.people- {p1}+ {s2} p1.delete();	s2.dept=p1.dept;
Select s1 from Staff s1 where ssn='ss1'; Person p2=migrate(s1, Staff, Person); s1.dept.people=s1.dept.people- {s1}+p2; s1.delete();	// migrate(s1, Staff, Person) Person p2= new Person(); p2.ssn=s1.ssn; p2.name=s1.name; p2.dept=s2.dept;

Other basic object operations include:

select x from all C x where x.K=k; // Retrieve objects from the class hierarchy rooted at C:

Establish_Relationship_between_Objects_on_foreign_key_K(x, y, Fg)

Break_Relationship_on_foreign_key(x, y, Fg)

5 Update Operation Translation from Relational to OO Databases

The original relational database is considered the view on the migrated OODB. Update operations submitted against the original relational database should be translated into equivalent operations on the OODB.

We will use SQL data manipulation operations as the representative for relational operations. Without considering operations involving cursors, the SQL update¹ operations we will discuss include *INSERT*, *UPDATE(searched)*, *DELETE(searched)*. Translating a relational update statement consists of locating the objects to be manipulated(except INSERT), checking the integrity constraint.

5.1 Integrity Constraint Enforcement

The SQL standard provides methods for defining integrity constraints, including domain constraints, base table and column constraints and general constraints. Most RDBMSs directly support base table constraints which include key, foreign key and check constraint. However, support for arbitrary constraint checking is sadly lacked in many OODBMS products. Some have proposed to enforce constraints by constraint methods defined on classes.

To ensure that the resultant OODB be in a state that can be mapped into a relational database state, the relational update operations and thus the translated OO operations should enforce the integrity

¹ To distinguish update operation and the SQL UPDATE statement, we use upper-case letters for the latter.

constraints defined on the original relational schema. We assume that there are key and foreign key constraints defined on the original relational schema.

Primary key constraint Primary key constraint guarantees the uniqueness of a primary key value in a relation. To enforce primary key constraint we create an index for each relation in the relational schema. This index stores key and associated OID values for all the tuples currently represented in the database, and provides methods to insert, delete and find entries in the index.

Foreign key constraint Foreign key constraint ensure that all foreign keys identify a tuple in the host relation. Using the index created for each relation, we can find the OID for each fully defined foreign key or updated foreign key in a relational tuple. The bidirectionality of associations make the enforcement of foreign keys particularly easy.

5.2 INSERT Operation Translation

Given a relational INSERT operation which supplies a set of attribute values for a new tuple:

INSERT INTO $r(\text{column-commalist})$ VALUES (value-commalist)

using default values and NULL values we can expand the *value-commalist* to contain data for the entire tuple, producing an INSERT operation of the following general form:

INSERT INTO $r(R)$ VALUES (tuple)

To map this relational INSERT operation into corresponding OO operations, *tuple* is first checked against the primary key and foreign key constraints on $r(R)$. Translation proceeds only when *tuple* satisfies the constraint checking. Insertion of a new tuple into $r(R)$ produces different operations on the OO side depending on the type of R , as described in the algorithm below.

Algorithm 1 Translate Relational Single-tuple INSERT to object operations.

Input: $r(R)$ // relation and its scheme, suppose $R=K \cup F \cup A$, where K , F and A are primary keys, foreign keys disjoint with K and local attributes

$\text{ind}(r)$ // primary key index for r

tuple // the data to be inserted

Output: TRUE if a valid sequence of object operations has been formed otherwise FALSE.

Begin

- 1) Check $\text{ind}(r)$ for primary key values of $r(R)$;
- 2) if(tuple violates the primary key constraint) return FALSE;
- 3) for each foreign key in K : $K_i \subseteq K$, $r[K_i] \subseteq r_i[K_i]$, $1 \leq i \leq m$ do
- 4) Suppose $r_i(R_i)$ is mapped into class C_i ;

```

5)   select xi from all Ci xi where xi.Ki=tuple[Ki]; // Check for foreign key constraint
6)   if(xi does not exist) return FALSE;
7)   endfor
8)   for each foreign key in F: Fi⊆F, r[Fi] ⊆ri'[Ki'], 1≤ i ≤ n do
9)     Suppose that ri'(Ri') is mapped into class Ci';
10)    select xi' from all Ci' xi' where xi'.Ki'=tuple[Fi]; // Check for foreign key constraint
11)    if(xi' does not exist) return FALSE;
12)   endfor
13)   if( R is ECR, SCR, RCR or CCR and is mapped into a class C) {
14)     p=new C();
15)     for (each a∈A) do   p.a=tuple[A];
16)     for (each foreign key Fi ⊆F) do
17)       Find the corresponding relationship(fi, fi') on C and Ci' from the mapping table;
18)       Establish_Relationship_on_foreign_key(p, xi', fi, fi');
19)     end for
20)   }
21)   Switch(type of R) {
22)     case ECR(m=0):
23)       for(each k∈K) do {p.k=tuple[k];}
24)     case CCR, SAR(m=1):
25)       if(R is CCR)
26)         Establish_aggregation (x1, p, tuple[K']);
27)       else
28)         Establish_aggregation (x1, tupe[K-K'], tuple[K']);
29)     case RCR, BRR(m≥2):
30)       if(R is a BRR) {
31)         retrieve the relationship pair(k1, k2) from schema mapping;
32)         Establish_relationship (x1, k1, x2, k2);
33)       }
34)     else {
35)       for(i=1, i ≤ m, i++) do
36)         p.ai=xi;
37)       }
38)     case SCR(m=1):
39)       x1 must be an object of class C1 which is the direct superclass of C;
40)       migrate (p, x1, tuple) // x1 obtain the identity p with more attribute values of tuple.
41)   end Switch;
42)   Insert entry into ind(r) for the primary key of tuple and p;
43)   return TRUE;

```

End

Example 4 When inserting a tuple into the relation Person_hobby

```
insert into Person_hobby values('ss3', 'tennis')
```

Person_hobby is a SVR and its host relation is Person. The corresponding OO operations are:

```
select p from all Person p where p.ssn='ss3';
```

```
p.hobby=p.hobby+{'tennis'};
```

Example 5 Given an INSERT operation on relation *Staff*

```
INSERT INTO Staff VALUES('ss2', 'AssProf', 'cs01')
```

This will cause a new object created in *Staff* and the object of class *Person* whose key is 'ss2' migrate to class *Staff*. The OO operations are:

```
SELECT p FROM all Person WHERE p.ssn='ss2';  
SELECT c FROM all Course WHERE c.cno='cs01';  
x=new Staff('ss2', 'AssProf');  
Establish_association(x, c, teach, taught_by);  
migrate(x, p)
```

5.3 DELETE Operation Translation

The SQL standard provides positioned DELETE, which deletes the row pointed to by the current cursor pointer, or a searched UPDATE, which may result in the deletion of several rows. We will focus on the deletion of a single tuple, which takes the form

```
DELETE FROM r(R) WHERE K=a
```

where *K* is the primary key of *R* and *a* is a valid value in *K*'s domain.

As BRRs and SVRs don't appear on the r.h.s. of any INDs, deletions on BRRs and SVRs are unrestricted and deletion of a tuple of such a relation doesn't correspond to the deletion of an object on the OO side but only breaking the relationship between two objects or removal of an element from a collection. Deletion of a tuple of a ECR, CCR or SCR must be first checked against the foreign key constraints defined and it is mapped into deletion of an object of the corresponding class. For SCR, deletion of a tuple may also cause object migration along the class hierarchy. A detailed translation for single tuple DELETE operation is described in algorithm 2.

Algorithm 2 Translate Relational Single-tuple DELETE to object operations.

Input: *r*(*R*) // relation and its scheme, suppose $R=K \cup F \cup A$, where *K*, *F* and *A* are primary keys, foreign keys disjoint with *K* and local attributes

ind(*r*) // primary key index for *r*

a // the primary key value of the tuple to be deleted

Output: TRUE if a valid sequence of object operations has been formed otherwise FALSE.

Begin

- 1) for(each foreign key *F_i* of table *r_i* whose r.h.s. is *R*) do // Check for foreign key constraint
- 2) if (there is any *r_i*[*F_i*]=*a*) return FALSE;
- 3) endfor
- 4) Switch(type of *R*) do

```

5) case ECR, SCR:
6)   Suppose the class corresponding to r is C;
7)   select x from C x where x.K=a; // x must be an object of C but not subclasses of C
8)   if R is a SCR
9)     migrate (x, superclass_of_C);
10)  else
11)    Break relationships with x;
12)    x.delete(); // Once x is deleted, references to x are set to nil.
13) case CCR, SVR:
14)   Suppose the host class is C1 and for CCR suppose the class corresponding to r(R) is C;
15)   select x1 from all C1 x1 where x1.K1=a[K1];
16)   if(R is a CCR) {
17)     select x from C x where x.K=a; // x must be an object of C but not subclasses of C
18)     break relationship with x;
19)     x1.p = x1.p - {x}; // p is the 1:m aggregation attribute, 1:1 aggregation?
20)     x.delete();
21)   }
22)   else
23)     x1.p = x1.p - {A};
24) case BRR, RCR:
25)   for each foreign key in K:  $K_i \subseteq K$ ,  $r[K_i] \subseteq r_i[K_i]$ ,  $1 \leq i \leq m$  do
26)     Suppose  $r_i(R_i)$  is mapped into class  $C_i$ ;
27)     select  $x_i$  from all  $C_i$   $x_i$  where  $x_i.K_i=a[K_i]$ ; // Check for foreign key constraint
28)   endfor
29)   if( R is RCR and is mapped into a class C ) {
30)     Suppose the corresponding attributes are  $p_1, p_2, \dots, p_m$ ;
31)     select x from C x where  $x.p_1=x_1$  and  $x.p_2 = x_2$  and ...  $x.p_m = x_m$ ;
32)     for (each foreign key  $F_i \subseteq F$ ) do
33)       Find the corresponding relationship( $f_i, f_i'$ ) on C and  $C_i'$  from the mapping table;
34)       Break_Relationship_on_foreign_key (x,  $x_i'$ ,  $f_i, f_i'$ );
35)     end for
36)     x.delete();
37)   }
38)   else {
39)     retrieve the relationship pair( $k_1, k_2$ ) from schema mapping;
40)     Break_relationship (x1,  $k_1, x_2, k_2$ );
41)   }
42) end Switch
43) Delete the entry from  $ind(r)$  for the primary key of a;
44) return TRUE;

```

End

Example 6 In the example relational schema, relation *staff* is a SCR and contains a foreign key referencing cno of *Person*. To delete a tuple from staff

Delete from Staff where ssn='ss3'

On the OO side:

select OID into s1 from Staff where ssn='ss3';

```
s1.teach.taught_by=null
migrate(p1, Staff, Person);
```

5.4 UPDATE Operation Translation

We consider an UPDATE operation which sets attribute p of the tuple t identified by the primary key value a the new value val :

```
UPDATE r(R) SET p=val WHERE K=a, where R=K ∪ F ∪ A
```

If $p \in A$, then setting p to a new value won't violate primary key and foreign key constraints and, on the OO side, involves only locating and setting the piece of data to the new value. If $p \in K$ or $p \in F$, setting p a new value must first check against the primary key and foreign key constraints and may change the relationship between objects or even cause object migration. A general translation process is summarized in algorithm 3.

Algorithm 3 Translate Relational Single-tuple UPDATE to object operations.

Input: $r(R)$ // relation and its scheme, suppose $R=K \cup F \cup A$, where K , F and A are primary keys, foreign keys disjoint with K and local attributes

$ind(r)$ // primary key index for r

a // the primary key value of the tuple to be deleted

p // the attribute to be updated

Output: TRUE if a valid sequence of object operations has been formed otherwise FALSE.

Begin

- 1) if(a is not in $ind(r)$) return FALSE;
- 2) if($p \in K$) {
- 3) // Check for primary key constraint
- 4) Suppose the new value formed for K by val is k_val ;
- 5) Search $ind(r)$ for the violation of primary key constraint of k_val ;
- 6) if primary key constraint is violated return FALSE;
- 7) if(R is a ECR, SCR, CCR or RCR) { // r is mapped into some class
- 8) Suppose $r(R)$ is mapped to class C ;
- 9) select x from C x where $x.K=a$;
- 10) if x is not found return FALSE; // the tuple t where $t[K]=a$ is referred to by ISA foreign keys
- 11) if there is any set-valued attribute which takes as its value a non-empty set
- 12) return FALSE; // t is referred to by Component foreign keys
- 13) }
- 14) }
- 15) else if (R is a ECR, SCR, CCR and $p \in F$) {
- 16) Suppose $r(R)$ is mapped to class C ;
- 17) select x from all C x where $x.K=a$;
- 18) Suppose $p \in F_i : r[F_i] \subseteq r'[K_i]$ and the new value formed for F_i by val is f_val ;
- 19) search $ind(r')$ for the entry f_val ;
- 20) if not found

```

21)         return FALSE;
22)         select x1 from all Ci' x1 where x1.Ki=x.Fi; // old value for Fi;
23)         select x2 from all Ci' x2 where x2.Ki=f_val; // new Fi value formed by val
24)         Break relationship between x and x1 on Fi;
25)         Establish relationship between x and x2 on Fi;
26)         return TRUE;
27) }
28) switch ( the type of R) do
29)   case ECR, SCR:
30)     if(p∈A) {
31)       select x from all C x where x.K=a;
32)       x.p=val;
33)     }
34)   else if( p ∈K)
35)     if (R is a ECR)
36)       x.p=val;
37)     else {
38)       Suppose  $r[K] \subseteq r1[K]$  and r1 is mapped into class C1;
39)       select x1 from C1 x1 where x1.K=k_val;
40)       if x1 not found return FALSE;
41)       migrate(x1, C, C1);
42)       migrate(x, C1, C);
43)     }
44)   }
45)   case CCR:
46)   Suppose  $K=K1 \cup B$ , where  $r[K1] \subseteq r1[K1]$ , r1 is mapped to class C1 with aggregation attribute agg;
47)     select x1 from all C1 x1 where x1.K1=a[K1];
48)     if(p∈A or p∈B)
49)       x.p=val;
50)     else if(p∈K1) {
51)       select x2 from all C1 x2 where x2.K1=k_val[K1];
52)       if x2 not found return FALSE;
53)       x.p=val;
54)       x1.agg=x1.agg-{x};
55)       x2.agg=x2.agg+{x};
56)     }
57)   case SVR: // F=φ, A=φ
58)     Suppose  $K=K1 \cup B$ ,  $r[K1] \subseteq r1[K1]$ , r1 is mapped to class C1 with aggregation attribute agg ;
59)     select x1 from all C1 x1 where x1.K1=a[K1];
60)     if(p∈B)
61)       x1.agg = x1.agg - {a[B]} + {val};
62)     else { // p∈ K1
63)       select x2 from all C1 x2 where x2.K1=k_val[K1];
64)       if (x2 can't be found)
65)         return FALSE; // If the foreign key constraint on K1 is violated
66)       x1.agg = x1.agg - {a[B]};
67)       x2.agg= x2.agg + {a[B]};
68)     }
69)   case RCR:

```

```

70)    Suppose  $K=K1 \cup K2 \cup \dots \cup Km$ ,  $r[Ki] \subseteq ri[Ki]$  and  $ri$  is mapped to class  $Ci(i=1, \dots, m)$ ;
71)    Suppose the attribute referring to  $Ci$  is  $ppi$ ,  $i=1, \dots, m$ ;
72)    if( $p \in A$ )
73)         $x.p=val$ ;
74)    else if( $p \in F$ ) {
75)    // However, relationship may between relationships and so  $x.pp1.pp1'.pp1' \dots pp1''''=a[K1]$ 
76)    select  $x$  from all  $C$   $x$  where  $x.pp1.K1=a[K1]$  and  $x.pp2.K2=a[K2] \dots$  and  $x.ppm.Km = a[Km]$ ;
77)        Suppose  $p \in Fi$ :  $r[Fi] \subseteq ri[Ki]$  and  $ri$  is mapped to class  $Ci$ ;
78)        select  $x1$  from all  $Ci$   $x1$  where  $x1.Ki=x.Fi$ ;
79)        select  $x2$  from all  $Ci$   $x2$  where  $x2.Ki=f\_val$ ;
80)        break_relationship( $x, x1$ );
81)        establish_relationship( $x, x2$ );
82)    }
83)    else { //  $p \in K$ 
84)        Suppose  $p \in Ki$ ;
85)        select  $xi$  from all  $Ci$   $xi$  where  $xi.Ki=k\_val[Ki]$ ;
86)         $x.ppi=xi$ ;
87)    }
88)    case BRR: //  $F=\phi, A=\phi$ 
89)        Suppose  $K=K1 \cup K2$ ,  $r[Ki] \subseteq ri[Ki]$ , and  $ri$  is mapped to class  $Ci$ ,  $i=1, 2$ 
90)        Suppose, without losing generality,  $p \in K1$ ;
91)        select  $x1$  from all  $C1$   $x1$  where  $x1.K1=a[K1]$ ;
92)        select  $x2$  from all  $C2$   $x2$  where  $x2.K2=a[K2]$ ;
93)        select  $x1'$  from all  $C1$   $x1'$  where  $x1'.K1=k\_val[K1]$ ;
94)        break_relationship( $x1, x2$ );
95)        establish_relationship( $x1', x2$ );
96)    end Switch
97)    Delete the entry from  $ind(r)$  for the primary key of  $a$ ;
98)    return TRUE;

```

End

Example 7 Update the key value of a tuple t of CCR means that cut the old aggregation between the composite object and the component object o (corresponding to t) and establish the new aggregation relationship and modify the key value of o if necessary.

Update Person_Hobby set $ssn='ss2'$ where $ssn='ss1'$ and hobby='football'

The OO correspondent is:

```

select OID into :p1 where  $ssn='ss1'$ ;
select OID into :p2 from all Person where  $ssn='ss2'$ ;
p1.hobby =p1.hobby-{football};/* Cut the old association */
p2.hobby=p2.hobby+{tennis}

```

6 Prototype of mapping SQL to OSQL

To translate transaction from RDB to OODB, we can apply symbolic transformation technique which contains syntax translation and semantic translation for SQL. For syntax translation, SQL statement

will be modified. For semantic translation, navigational syntax will be modified. For example, the join operation in RDB can be replaced by class navigation(association) in OODB. Transaction of source language are built in our model. We navigate the transaction graph (TG) of SQL and then map it to the TG of OSQL(object-oriented SQL)[22] with reference to the intermediate result of schema translation. Semantic rules (transformation definition) for transaction transformation from source language to target language will be applied. Then, transaction of target language will be produced. The output transaction OSQL should be syntactic and semantic equivalent to the source SQL.

The OSQL, the object-oriented extension to SQL, allows data retrieval using path expressions, and data manipulation using methods. In transaction transformation, a syntax-directed parser converts the input OSQL into multi-way trees. The transformation process is then performed, based on the subtree matching and replacement technique. The process of SQL to OSQL transformation is in Figure 2.

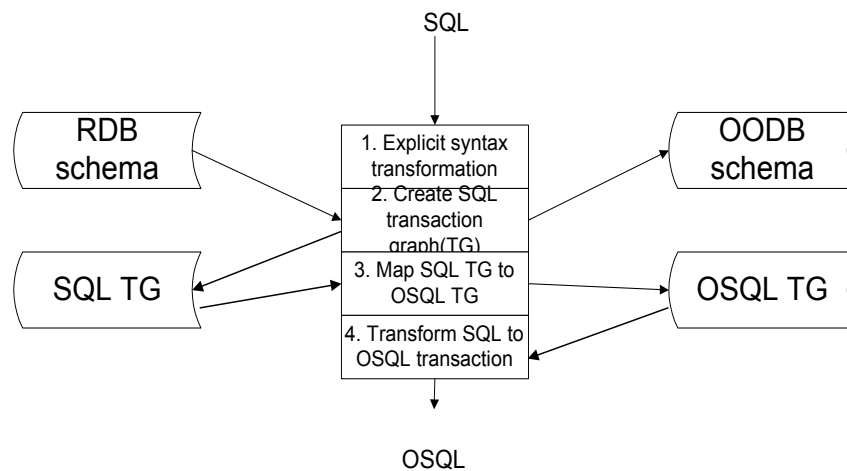


Figure 2 Process for SQL transformation to OSQL

After schema mapping from RDB to OODB, we can transform RDB transaction(SQL), to OODB transaction(OSQL). The nature of the query transactions of the SQL and OSQL are different. SQL is to navigate from relation to relation through foreign key. OSQL is to navigate from class to class through OID. The concept of database navigation can be used to verify the equivalence of an SQL and its translated OSQL statement. To ensure the database state is preserved after transaction translation, the target data records accessed by the OSQL statements must be the same as the ones obtained by the SQL statements. The concept of a navigation statement can show the access paths of SQL and OSQL statements.

For example, the SQL statement for the transaction “Get the names of the courses taken by student who are major in Computer Science“ is shown in Figure 3.

SQL transactions:

Select Course-name from Course, Course-register, Student where Course.Course-no=Course-register.Course-no and Course-register.Student#=Student.Student# and Major="Computer Science"

Insert Course-register (Student_no, Course_no) values (124, 'CS101')

Its relational schema is:

Relation Student (Student_no, Major)

Relation Course (Course_no, Course_name)

Relation Course-register (Student_no, Course_no)

Its navigation statement is shown in Figure 3 as follows:

Course.Course-no.[Course-register.Student#[Student.Major.ComputerScience.[Student#].
[Course-no].Course-name²

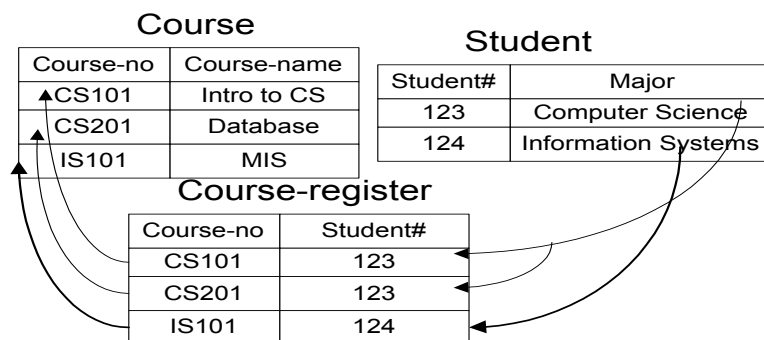


Figure 3 Navigation access paths of SQL

To verify the equivalence between the SQL and its translated OSQL statement, we can show that their target data are the same by expressing them in the same domain of navigation statement. The navigation statements present their database access paths. After deriving the nested OSQL transaction and the nested SQL transaction along with their navigation statements, we can show them to be equivalent. For example, for the SQL transaction in Figure 4, its equivalent OSQL transaction with TG in Figure 4 can be shown below:

The translated OODB schema is:

Class Student

Attribute

Major: string

Student_no: integer

Course-taken: set of Course

End

Class Course

Attribute

Course_no: integer

Course_name: string

Course-registered: set of student

End

² Refer to Appendix A Navigation Statement

Translated OSQL transactions:

Select Course-taken.Course-name from Student where Major = “Computer Science”

Select * from Student where Student_no = 124 into :OID_{student}

Update Course set Course-registered = Course-registered + :OID_{student}

Select * from Course where Course_no = ‘CS101’ into :OID_{course}

Update Student set Course-taken = Coruse-taken + OID_{course}

Its navigation statement is:

Student.Major.ComputerScience.Course-taken.Course.Course-name

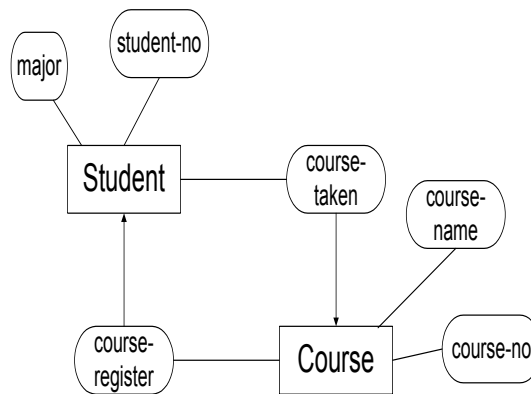


Figure 4 Navigation access path of the translated OSQL

7 Concluding Remarks

In this paper we have studied mapping relational update operations to their OO equivalents, which include UPDATE*, INSERT and DELETE operations. Relational update operation translation from relational to OO faces the touchy problem of transformation from a value-based relationship model to a reference-based model and maintaining the relational integrity constraints. Moreover, with a relational database where inheritance is expressed as attribute value subset relationship, changing of some attribute values may lead to the change of the position of an object in the class inheritance hierarchy, which we call object migration. Considering all these aspects, algorithms are given mapping relational UPDATE, INSERT and DELET operations to their OO correspondents. The continuation of this research project will be to convert RDB programs to OODB methods.

Appendix A Navigation Statement

* Note that we use *update* for general update operations and UPDATE for modifying certain data values.

A navigation statement is a function which maps one set of addresses (departure point) into another set of addresses (arrival point). We can use these departure and arrival points to model the navigation path of an object-oriented model as follows[23][24]:

Let A be a set of addresses, N be a set of data names and V be a set of atomic values. Suppose that A , N and V are disjoint sets. We consider the content of a database as a “database content” that is a collection of addresses, data names and data values such that $CON \subseteq A \times N \times (A \cup V)$. A database content contains the triple $\langle a_1, n, a_2 \rangle$ or $\langle a, n, v \rangle$. The former means that at address a_1 , there is datum with name n and a pointer leading to data at address a_2 . The latter means that at address a , there is a datum with name n and atomic value v . For example, for the following OODB schema:

Class COURSE (Course-no: integer, Course-name: string, Course-register: set (STUDENT))

Class STUDENT (Student-no: integer, Major: string, Course-taken: set (COURSE))

Its OSQL query graph and navigation statement can be shown in Figure 5.

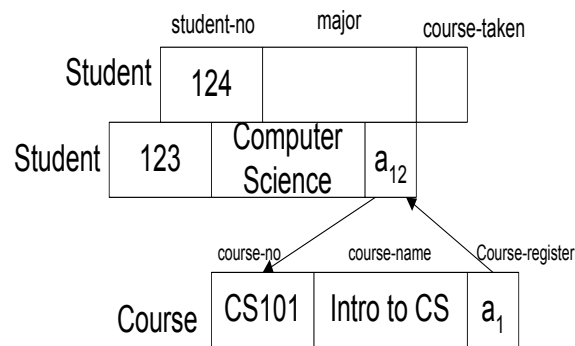


Figure 5 Navigation access path of the OODB

The following data are representations of the database content for the database CON_1 :

$\langle a_1, \text{student}, a_{10} \rangle, \langle a_1, \text{student}, a_{11} \rangle, \langle a_1, \text{student}, a_{12} \rangle, \langle a_1, \text{student}, a_{13} \rangle,$

$\langle a_{10}, \text{student-no}, 123 \rangle, \langle a_{11}, \text{major}, \text{computer-science} \rangle, \langle a_{12}, \text{course-taken}, a_2 \rangle,$

$\langle a_{13}, \text{student-no}, 124 \rangle, \langle a_2, \text{course}, a_{21} \rangle, \langle a_{21}, \text{course-no}, \text{CS101} \rangle$

Let S be the set of navigational statements. We denote by $\|S\|$ the semantics of the statements. The function $\|S\|$ maps a set of addresses into another set of addresses. The set of arguments of the navigational statements will be the departure points, and the result will be the arrival points. $\|S\|$ depends on particular database content CON , and is regarded as a mapping of $\|S\|: \rho(A) \times CON \rightarrow \rho(A)$, where CON is a set of all possible database contents, i.e., from one address of the database content, we can navigate to another address within the database content through pointers.

Let us consider a navigational statement $s \in S$, where $S = S_1, S_2, \dots, S_k$. If the statement S is applied to the set of addresses $A_0 \subseteq A$, then we can define a trajectory of navigation in a sequence of address sets: $\langle A_0, A_1, \dots, A_k \rangle$ where $A_1 = \|S_1\| (A_0)$, $A_2 = \|S_1.S_2\| (A_0)$, $A_3 = \|S_1.S_2, \dots, S_k\| (A_0)$. For example, the following table contains navigation statement representations for some transactions in Figure 5.

Table 1 Navigation Statement of Transactions

Transaction Function	Navigation Statement
Get all object occurrence of student	Student
Get Student-no of all student	Student.Student-no
Get Course-no of student with student-no = 123	Student.Student-no.123.Course-taken.Course.Course-no

For sets of navigation statements, if S is a navigation statement containing expressions which are equivalent to sets of expression, we can group them together in one expression. For example, the navigation statements for “get course numbers of the courses taken by student with student# = 123 or 124” is

OSQL statement	Navigation statement
Select Course-taken.Course-no from Student	Student.Student#.(123, 124).Course-taken.

where Student# = 123 or Student# = 124 Course.Course-no

For nested navigation statements, an inner statement S_1 can be considered a set of values placed at addresses which constitute the arrival points of navigation according to S_1 . After evaluating the inner navigation statement, the outer navigation can be evaluated. For example, the navigation statement of “Get the student number of students who have the same major as the student with student number 123” is

Student.Major.[Student.123].Student#

References

- [1] J Fong, P Chitson, Query Translation from SQL to OQL for Database Reengineering, International Journal of Information Technology, 3(1), 1997, pp. 83-101.
- [2] C J Date, H Darwen, A Guide to the SQL Standard: A user’s guide to the standard language SQL(4rd ed.), Addison-Wesley Publishing, 1997.
- [3] R G G Cattell etc. (eds), *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, 1997.
- [4] V M Markowitz, A Shoshani, Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach, ACM TODS, 17(3), 1992, pp. 423-464.

- [5] W Meng, C Yu, W Kim, G Wang, T Pham, and S Dao, Construction of a Relational Front-end for Object-Oriented Database Systems, ICDE'93, pp. 476-483.
- [6] X Qian, L Raschid, Query Interoperation Among Object-Oriented and Relational Databases, ICDE'95, pp. 271-278.
- [7] C Yu, Y Zhang, W Meng, W Kim, G Wang, T Pham, S Dao, Translation of Object-Oriented Queries to Relational Queries, ICDE'95, pp. 90-97.
- [8] F Bancilhon, N Spyrtos, Update Semantics of Relational Views, ACM Trans. on Database Systems, 6(4), Dec. 1981.
- [9] U Dayal, P A Bernstein, On the Correct Translation of Update Operations on Relational Views, ACM Trans. on Database Systems, 7(3), Sept. 1982.
- [10] A M Keller, The Role of Semantics in Translating View Updates, IEEE Computer, January 1986, pp. 63 - 73.
- [11] W., Meng, C. Yu, and W. Kim, A Theory of Translation from Relational Queries to Hierarchical Queries, IEEE Transaction on Knowledge and Data Engineering, vol 7, no. 2, April 1995, pp. 228-245.
- [12] A. Rosenthal, and D. Reiner, Querying Relational Views of Networks,, pp. 107-124.
- [13] J-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon, Transformation-Based Database Reverse Engineering, ERA'93, pp. 364-375.
- [14] S. Dumpala and S. Arora, Schema translation using the entity-relationship approach, Entity-Relationship Approach to Information Modeling and Analysis, 1983, p337-356.
- [15] A. Rosenthal, A., and D. Reiner, Tools and Transformations -Rigorous and Otherwise-for Practical Database Design, ACM TODS, vol. 19, no. 2, June 1994, pp. 167-211.
- [16] R. Hull, Relative Information Capacity of Simple Relational Schemata, SIAM Journal of Computing, Vol 15, No. 3, August (1986), p856-886.
- [17] M E El-Sharkawi, Y Kambayashi, Object Migration Mechanisms to Support Updates in Object-Oriented Databases, N Rishe, S Navathe, D Tal(eds), *Databases: Theory, Design and Applications*, Computer Society Press, 1991, pp. 73-93.
- [18] Q Li, G Dong, A framework for object migration in object-oriented databases, Data & Knowledge Engineering, Vol. 13, 1994, pp. 221-242.
- [19] S. Bergmaschi, and C. Sartori, On Taxonomic Reasoning in Conceptual Design, ACM TODS, vol 17, no., 3, Sept. 1992, pp. 385-422.
- [20] J. Fong, Converting Relational to Object-Oriented Database, SIGMOD RECORD, Volume 26, Number 1, March 1997, pp. 53-58.

- [21] T W Ling, P K Teo, Object Migration in ISA Hierarchies, DASFAA'95, 1995, pp. 292-299.
- [22] J.G. Hughes, Object-Oriented Database, Prentice-Hall International, 1991, pp. 153-154.
- [23] K. Subieta, Navigational Facilities for Relational Databases, Information Systems, Volume 8, 1993, pp 29-36.
- [24] K. Subieta, High-Level Navigational Facilities for Network and Relational Databases, Proceedings of 9th International Conference on Very Large Databases, October, 1993.