Concurrent data materialization for Object-Relational database with semantic metadata

Joseph Fong, Ringo Pang , Anthony Fong*, Francis Pang,, Kenny Poon
Computer Science Department *Electronic Engineering Department
City University of Hong Kong,Tat Chee Avenue, Hong Kong, email: csjfong@cityu.edu.hk

Abstract
For a company with many databases in different data models, it is necessary to consolidate them into one data model interchangeable and present data in one data model concurrently to users. The benefit is to let user stick to his/her own data model to access database in another data model. This paper presents a semantic metadata to preserve database constraints for data materialization to support user's view of database on ad hoc base. The semantic metadata can store the captured semantics of a relational or an object-oriented database into classes and stored procedures triggered by events. The stored constraints and data can be materialized into a target database upon user request. The user is allowed to perform the data materialization many times alternatively. The process can provide a relational as well as an object oriented view to the users simultaneously. This concurrent data materialization function can be applied into data warehouse to consolidate heterogeneous database into a fact table in a data model of user's choice. Furthermore, a user can obtain either a relational view or an object-oriented view of the same dataset of an object-relational database interchangeably.
Keywords: Data materialization, semantic metadata, schema translation, relational database, object-oriented database

1 Introduction
For a company with an object-relational database, it is not clear how a user can obtain a relational view while another user obtains an object oriented view at the same time. Hence, a metadata containing the materialization of different data model must be maintained. Data types in different data model need different access mechanism.

To tackle the problems, a semantic metadata aims to capture the semantics of the source database. The semantic metadata contains different data constraints that the target database needs to implement after the data materialization. The procedure for materializing data into any data model is as follows:
Step 1: Capture data semantics of source database into semantic metadata.
1. Extract semantics from the Source database by examining their data occurrences.
2. Store data semantics into the semantic metadata.
3. Capture source database into sequential files.
4. Store source database into the semantic metadata.

Step 2: Transfer the stored data semantics and data into a target database
1. Stored data semantics are transferred into a target database schema.
2. Stored source database are transformed into a target database.

We need to perform schema translation before data materialization[1]. Data materialization is to transform data into different data models upon user request. As a result, a user can reuse a relational database as an object oriented database or vice versa with the same dataset in the semantic metadata. The process can provide a relational view and an object oriented view of the same database to the users concurrently.

2 Related Works
Schema translation translates schema from one data model into another. [2] proposed to translate ER diagram directly into their self-defined OO model which allowed specification of OO schema in three planes. [3] gave a survey of transformation methodologies and approaches for the various directions which were relevant to conceptual and logical database design. [4] proposed to translate a Relational schema into Class definition by use of five phases: Initialis_ation, Decomposition, Classification, Generalis_ation and Identification. [5] proposed a two-phase methodology to translate relational schema to OO schema and performed data migration using schema transformation. [6] proposed heuristic algorithms to approximate types semi-automatically. [7] investigated a RDB schema translation process to recapture lost semantics via an EER model using data search technique. [8] proposed three parallel discovery models based on horizontal, vertical, and matrix database table

slicing techniques. [9] proposed algorithms for discovering functional dependencies from data. [10] presented a schema mapping procedures applied on existing relational database without changing their schema. [11] improved the reverse engineering of relational database through a combination of data dictionary, data schema and data instance analysis.

[12] described the VAT system for data conversion. [13] presented a translation system, based on schema-matching, aimed at simplifying the intricate task of data conversion. [14] presented a technique for querying and transforming scientific data. [15] described a methodology for schema translation and data conversion from Relational database to Object oriented database. [16][17] implemented a data conversion system with mathematical verification. [18] presented an approach to load large volumes of data, and highlighted factors of reengineering used for repeatable processes in large industrial settings. [19] described a framework for data integration based on OO types hierarchies and late binding. [20] depicted an approach to database interoperation that exploits the semantic information provided by integrity defined on the component databases. [21] rendered a framework for data integration, based on a special class of mediators called Squirrel integration mediators. These mediator supported views that integrate data from multiple data sources, and supported the materialized and virtual approaches, and hybrids of them.

[22] stated that the study of semantics and the reality of computing have a strong relationship. [23] described a methodology to unravel the semantics of conceptual schema. [24] described an algorithm that reduced the database activity considerably. [25] introduced the ER model to capture the semantic and the structure of the data in the conceptual schema. [26] introduced Entity-Category-Relationship Model with various variant called the Extended/Enhanced ER Model (EER). [27] introduced Object Modeling Technique (OMT) to cover the operations and rules applied to the data in addition to semantic data modelling techniques. [28] presented a set of abstraction to build an integrated enterprise model, and suggested framework provided an integration of semantic, pragmatic and non-traditional communication dependencies. [29] gave a definition of the semantics of active database using a structured approach. [30] used the Semantic metadata to capture the semantic of a universal database. [31] described a methodology with verification for schema integration for Object Relational Database System. [32] defined a set of primitive transformations for an EER model and by defining some of the common schema transformations as sequences of these primitive transformations. [33] showed how schema transformation can migrate or warp data, queries and updates between semantically equivalent schemas. Other important contributions in the field include the ones by the group of Paolo Atzeni [34, 35, 36] and the ones by the group of Phil Berrstein [37, 38, 39, 40].

This paper suggests a unique metadata solution for data materialization by allowing concurrent relational and object-oriented views of the same data set upon users' demands.

3 Semantic metadata

Semantic metadata is a metadata model. It is a higher-order synthesis with semantic frame concepts, semantic data modeling concepts, object-oriented concepts and production rules concepts. All abstract data semantics in the conceptual schema can be recognized and mapped into the logical representations. It can represent the object behaviour and the relational functionality. A class in the semantic metadata contains four main parts: headers, attributes, methods and constraints. A general description of semantic metadata is given in table 1.

Table 1 Semantic metadata (frame model)

HEADER CLASS	
Class Name	/* a unique name in all system */
Key	/* an attribute name or by default a class *
Parents	/* a list of class names */
Methods	/* a list of methods */
Constraints	/* constraint methods for the class */
Description	/* class description */
ATTRIBUTE CLASS	
Class Name	/* a unique name in all system */
Attributes Name	/* a unique name in this class */
Attribute Type	/* the data type for the attribute */
Default Value	/* predefined value for the attribute */
Cardinality	/* is the attribute single or multi-valued */

Constraints /* constraint methods for the attribute */
Description /* a description of the attribute */

METHOD CLASS

Class Name /* a unique name in all system */
Method Name /* a unique name in the class *
Parameters /* a list of arguments for the method */
Method Type /* the final result data type */
Condition /* the conditional of the method */
Action /* action to be taken for the method */
Description /* the description of the method */

CONSTRAINT CLASS

Class Name /* a unique name in all system */
Method Name /* a unique name in the class *
Parameters /* a list of arguments for the method */
Ownership /* class name of the owner of the method */
Event /* triggered event */
Sequence /* method action time */
Timing /* the method action timer */
Description /* the description of the method */

The header class stores the proper class information and defines the class structure and relationships. Attribute class depicts the properties of a class. It can be filled by values, pointer to other objects, or procedures defined in the method part. It adds procedures for deductive rules functionality to the ordinary EER attribute type [41] and creates a dynamic structure to the Frame model. The Method class depicts rules and behaviour of a class. It denotes the definition of a procedure defined in the attribute part, a deductive rule or an active rule. The Constraint class captures the restrictions on data. It necessitates the knowledge completeness and consistency in the database, and enforces the integrity in the database. Factual data entities are stored in the database through the static class. Rules and Restrictions act on the data invoked by certain event through the active class. Combining these two types of classes within one data model provides the mechanism of structuring and sharing not only data, but also rules and procedures that act on the data to the database.

4 Capture data semantics with semantic metadata for schema translation

The semantic metadata offers the architecture of a metadata for the connectivity and schema translation of heterogeneous databases. Data semantics of different data models can be captured into the semantic metadata, and translated into another data model. A semantic metadata stores an application domain into classes linked via generalization, aggregation and user-defined relationships. Its data are stored in relational tables. The schemas of the source database systems are captured into the semantic metadata, and mapped into a target database in another data model as shown in Figure 1.

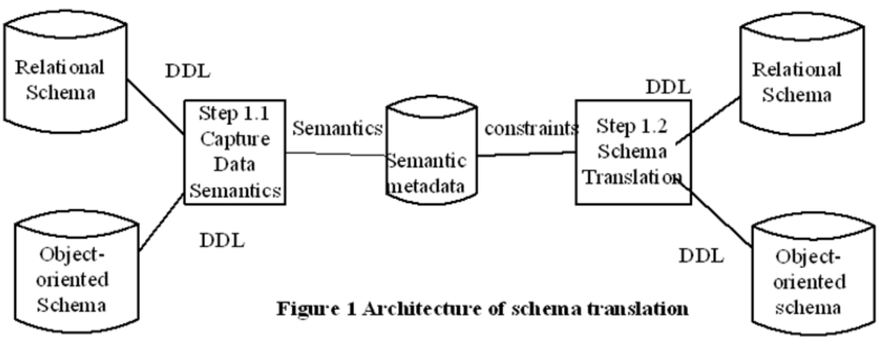


Figure 1 Architecture of schema translation

Data operation can be used to examine data occurrence of source database which can be interpreted as data semantics. Once captured, the following data semantics can be stored in the metadata trigger procedure:

Step 1.1Capture the is_a relationship of a legacy database into the semantic metadata

An is_a relationship is a superclass and subclass relationship such that the domain of subclass is a subset of its superclass. The following algorithm examines data occurrence of is_a relationship: Furthermore, the subclass can inherit attributes from its superclass along with its own attributes in OODB.

Relational View	Object-Oriented View

Let tList = a Table List of the Child Table in the Foreign Key List in which all Primary Key attribute is also Foreign Key attribute;
Begin Search from tList
Let x = Total Record Count of Child Table
Let y = Total Record Count of Parent Table
Check there has 1:1 relationship in between the parent and child tables;
Check there has 1:n relationship in between the parent and child tables;
If (x <= y) AND (Not(is1:1)) AND (Not(is1:n))
Then begin
Parent Table is Super Table,
Child Table is Sub Table,
Parent and child tables are in IS_A relationship
End If
End Search
	Given two classes and their object ID C_x, OID(C_x), C_y, OID(C_y) in a OO schema S, given that they are not participated in a binary relationship, we can locate their IS_A relationships as:
	Begin
	Select Count(OID(C_x)), OID(C_x) from C_x;
	Select Count(OID(C_y)), OID(C_y) from C_y;
	All_OID = OID(C_x) Union OID (C_y);
	Select Count(*)=Allcount from All_OID;
	IF Count(OID(C_x)) ≥ Allcount and OID(C_y) has more attributes than OID(C_x)
	THEN begin
	IS_A-relationship (C_y, C_x) := True;
	C_y := subclass;
	C_x := superclass;
	End;
	ELSE IF Count(OID(C_y)) ≥ Allcount and OID(C_x) has more attributes than OID(C_y)
	THEN begin
	IS_A-relationship (C_x, C_y) := True;
	C_x := subclass;
	C_y := superclass;
	End;
	End;

The following metadata stores the captured is_a relationship:

Header Class

Class_Name	Primary_key	Parents	Operation	Class_type
R_x	PK(R_x)	0		Static
R_y	PK(R_y)	R_x		Static

Step 1.2 Capture participation semantic of a legacy database into semantic metadata

The existence of a "weak" entity depends on its "strong" entity. Consequently, their relationship is total participation. Otherwise, their relationship is partial participation. The following algorithm examines data occurrence of participation:

Relational View	Object-Oriented View
For each 1:n relationship If the Foreign Key can be nullable Then They are in Partial Participation Else They are in Total participation End If Next	Given a class: CC, associates to another class AC with its REF(AC), an association attribute of stored OID pointer, we can locate its participation as: Let participation (CC, AC) = total; Select count (*)=C from CC where REF(AC)=null; IF C > 0 THEN participation (CC, AC) = partial;

The following metadata stores the captured participation:

Header Class

Class_Name	Primary_key	Parents	Operation	Class_Type
R	PK(R)	0	Call Delete_R	Active
R_j	PK(R_j)	0	Call Create_R_j	Active

Method class

Method_ name	Class_ name	Para meter	Seq_ no	Method_ type	Condition	Action	Next_ Seq_ no
Delete_R	R	@ PK(R)			If (Select * from R_j where PK(R_j) = @PK(R) ≠ null	Delete * from R_j where PK(R_j) = @PK(R)	
Create_R_j	R_j	@ PK(R_j)			If (Select * from R where PK(R) = @ PK(R_j)) ≠ null	Insert R_j (@PK(R_j))	

Step 1.3 Capture generalization of a legacy database schema into semantic metadata

A generalization can be represented by more than one subclasses having a common superclass. The following algorithm examines data occurrence of disjoint generalization such that subclass instances are mutually exclusive stored in each subclass.

Relational View	Object-Oriented View

Let tList = a Super Table List from IS_A relationship List such that the Super Table has two or more Sub Table;
For Each Super Table in tList
Let subTableList = the corresponding Sub Table;
Calculate the duplicate Record Count from each two Sub Table
If there are duplicate Record between Sub Tables
Then
Super Table and sub tables are in Overlap relationship
Else
Super Table and sub tables are in Disjoint relationship
End If
Next
	Given a superclass and its OIDs: C, OID(C), referring to its subclass and their OID: C_{j1}, OID(C_{j1}), …C_{jn}, OID(C_{jn}), their generalization can be located as:
	If IS_A-relationship (C_{j1}, C) = True and … and IS_A-relationship (C_{jn}, C) = True
	Then Generalization (C, C_{j1}, …C_{jn}) := Disjoint;
	For h := 1 to n do
	For k := 1 to n do
	IF h < k
	THEN begin
	Select Count(OID(C_{ih})) from C_{ih};
	Select Count(OID(C_{ik})) from C_{ik};
	Join_OID = OID(C_{ih}) Union OID(C_{ik})
	Join_Count = select count (Join_OID) from Join_OID
	IF Join_Count < Count(OID(C_{ih})) + Count(OID(C_{ik}))
	THEN Begin
	Generalization (C, C_{j1}, …, C_{jn}) := Overlap;
	Exit;
	End;
	End;
	Next; /* for loop k */
	Next; /* for loop h */

The following metadata stores the captured disjoint generalization:

Header Class

Class_Name	Primary_key	Parents	Operation	Class_Type
R	PK(R)	0		Static
R_{j1}	PK(R_{j1})	R	Call Create_R_{j1}	Active
R_{j2}	PK(R_{j2})	R	Call Create_R_{j2}	Active

Method class

Method_ Name	Class_ name	Param eter	Seq_ no	Method_ Type	Condition	Action	Next_ Seq_no
Create_R_{j1}	R_{j1}	@ PK(R_{j1})		Boolean	If (Select * from R_{j2} where PK(R_{j1}) = @ PK(R_{j1})) = null	Create_R_{j1} = true	
Create_R_{j2}	R_{j2}	@ PK(R_{j2})		Boolean	If (Select * from R_{j1} where PK(R_{j2})= @ PK(R_{j2})) = null	Create_R_{j2} = true	

Step 1.4 Capture cardinality of schema in a legacy database into the semantic metadata

The cardinality specifies data volume relationship in the database. The following algorithm examines data occurrence of cardinality of 1:1, 1:n and n:m.

Relational View	Object Oriented View

Discover 1:1
Let x = the Total Record Count of Child Table
Let y = the Total Record Count of Parent Table
Let z = the Distinct Record Count of Child Table
If (x = y) AND (y = z) Then they are 1:1
 	
Discover 1:n for cases
Let x = the Total Record Count of Child Table
Let y = the Total Record Count of Parent Table
Let z = the Distinct Record Count of Child Table
Check if there is null value in Child Table = n
Case 1: (z < y)
Case 2: ((z = y + 1) AND (n))
Case 3: (z = y) AND (z > y)
Case 4: (z = y) AND (z < y) AND (n)
	Given two classes and their reference attributes C_1, Ref(C_1), C_2, Ref(C_2) in an OO schema S, we can locate its cardinality as:
	
	For i = 1 to 2
	Select C_i, Ref(C_i), from S;
	max(i) = 1
	While not end of instance(Ref(C_i)) do
	Begin If instance(Ref(C_i)) = NULL
	then
	Minimum = True;
	Else If Ref(C_i) is a set reference
	then max(i) = n;
	End;
	End;
	If Minimum

Discover m:n
Get a Table List of the Child Table in 1:n relationship which has two or more 1:n relationship = tList
If (Primary Key of Child Table in tList is
compound Primary Key) and (All Primary Keys
are also the Foreign Key in such relationship)
then It is m:n relationship
End If
End If
	then
	Card(i) = (0, max(i));
	Else
	Card(i) = max(i);
	End;
	End /* for loop */
	Cardinality (C_1, C_2) = card (1) : card (2)

The following metadata stores the captured 1:n cardinality between R and R_j:

Attribute Class

Class_name	Attribute_Name	Method_Name	Attribute_type	Default_value	Cardinality	Description
R			R_j		n	Associated class attribute
R_j			R		1	Associated class attribute

Step 1.5 Capture aggregation of a legacy database schema into the semantic metadata.

Aggregation is an abstraction concept for building composite objects from their component objects. The following algorithm examines data occurrence of aggregation object which must consist of all of its component objects: (note: Reference attribute refer to the component class)

Relational View	Object Oriented View

Let rList = a list of relationship in binary relationship
For Each rList
If there has another relationship which is related to the relationship in rList
Then the entities and relationship in rList are in aggregation
End If
Next
	Given an aggregate class with its reference attribute pointers, AC, Ref_1(AC),…,Ref_n(AC) referring to its component classes with its OID, CC_1,…,CC_n, OID(CC_1),…, OID(CC_n) from schema S, the aggregation can be located as:
	For i = 1 to n
	For k = 1 to n
	IF Ref_i(AC) = OID(CC_k) THEN
	Select Ref_i(AC) from AC;
	While not at the end of instance(Ref_i(AC))do
	Begin Select Count(Ref_i(AC)) = $C_k$from AC
	where Instance(Ref_i(AC)) = NULL;
	End;
	Break;
	End;
	Next;
	Next;
	For k = 1 to n
	IF C_k>0 THEN Aggregation (AR, CC_k) = True;
	ELSE Aggregation (AR, CC_k) = False;
	End;
	Next;

The following metadata stores the captured aggregation:

Header Class

Class_Name	Primary_key	Parents	Operation	Class_Type
CR_1	PK(CR_1)	0		static
CR_2	PK(CR_2)	0		static
AR	PK(CR_1), PK(CR_2)	0	Call Create_AR	active

Method class

Method_Name	Class_name	Parameter	Seq_no	Method_type	Condition	Action	Next_Seq_no
Create_AR	AR	@PK(CR_1), @PK(CR_2)			If ((Select * from CR_1 where PK(CR_1) = @ PK(CR_1)) ≠ null) and If ((Select * from CR_2 where PK(CR_2) = @ PK(CR_2) ≠ null)	Insert AR (@PK(CR_1), @PK(CR_2))	

Translating the semantic metadata to a legacy database schema

Step 1.2 Translating the semantic metadata to a legacy database schema

If the Target database is an OODB, the object oriented data model holds the method and constraint class from the semantic metadata. If the Target database is a RDB, the method class from the semantic metadata is implemented using stored procedures; and the constraint class is implemented using triggers as shown below:

- Step 1.2.1 Map semantic metadata to relational schema
  For i = 1 to M do
  begin
  Get objects $(FMO)_i$ in Semantic metadata (FM);
  Create table T_i using $(FMO)_i$;
  end;
- Step 1.2.2 Map semantic metadata to object-oriented schema
  For i = 1 to M do
  begin
  Get objects $(FMO)_i$ in Frame Model (FM);
  Create class C_i using $(FMO)_i$;
  end;

5 Concurrent Data Materialization

One set of data produced from the source RDB or OODB can be materialized into another target RDB or OODB depending on user needs. The result is an interchangeable data set between RDB and OODB as shown in figure 2:

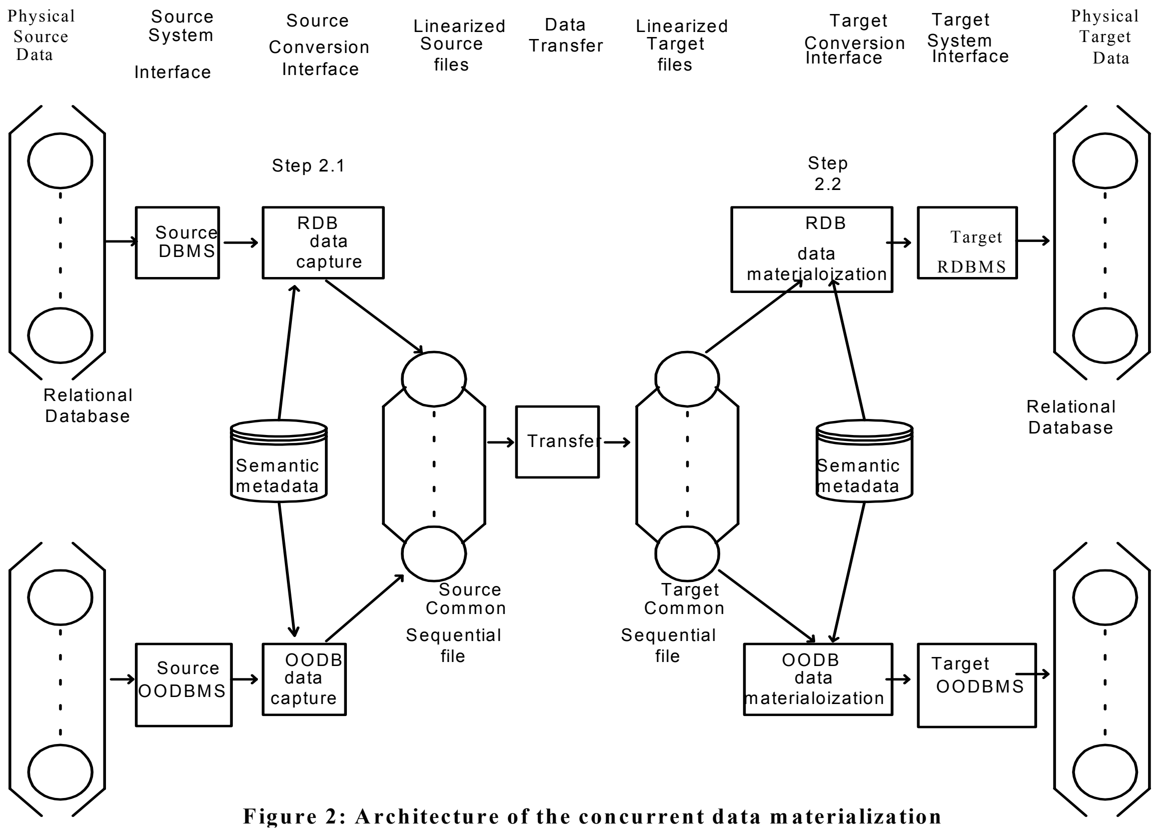


Figure 2: Architecture of the concurrent data materialization

Step 2.1: Capture Data from the Source Database into the ASCII Files

The captured data will be output to sequential files. If the Source database is a RDB, the ASCII file formed contains the key value of this tuple and the attributes with base type. The Object_Key attribute contains the key value and the Attribute1_Name, … AttributeN_name contain the attributes of base type. For each class, one ASCII file will be formed. If the Source database is an OODB, the ASCII file formed contains the OID (called ObjKey_Value here) of this object, the attributes with base type, the association attributes with the OID of objects within associated class (called AssAtr1Key_Value…

here), and the set value type attributes that contain attributes of the base type (called SetAtr1_Value…)as shown in table 2.

Table 2 File Structure for an ACSII file stores captured data from the Source database

ASCII File Name: Class_Name	
Object_Key	ObjKey_Value
Attribute1_Name	Attribute1_Value
Attribute2_Name	Attribute2_Value
…	
AttributeN_Name	AttributeN_Value
AssocAtr1_Name	AssocAtr1Key_Value
…	
SetAtr1_Name	SetAtr1_Value1
	SetAtr1_Value2
…	…

Step 2.1.1: Capture Data from the Relational Database to the ASCII Files
The data will be downloaded into the ASCII files directly from the RDB.
Step 2.1.2: Data materialization from the OODB to the ASCII Files
This step captures the OID and data of the base types attributes of every object within the chosen class to the sequential files. For each object in the class, the values of the association attributes will be captured to the sequential files. For each object in the class, the values of the set values attributes are captured into sequential files as follows:

Given: input: classes C_1 to C_P within the target OODB
objects O_{i1} to O_{iQ} within the class C_i
classes $(AC)_{ik}$ with association attributes A_{ik} (from A_{i1} to A_{iR}) within Class Ci
set values attributes S_{i1} to S_{iT} in class C_i
output: OID and data of the base types attributes to sequential file $(SF)_I$

```
For i = 1 to P do
begin
   for j = 1 to Q do
   begin
      Get OID and data of all attributes from objects O_ij;
      Output OID and data of the base types attributes to sequential file (SF)_I;
For k = 1 to R do
begin
         Case association of
            One-to-one association with A_ik
               Get OID from (ACO)_ijk and output to the sequential file (SF)_i;
            One-to-many association with A_ik
               Get all OIDs form (AC)_ik and output to (SF)_i;
       end;
     For m = 1 to T do
         If S_im within C_i is of base types
         then Output value of object O_ijm that own S_im attrihute to the sequential file (SF)_i;
         else Get all OIDs from (SC)_im and output to (SF)_i;
  end;
end;
```

Step 3: Data materialization from the ASCII Files into the Target Database
The process will read object data from the ASCII file according to the semantic in the semantic metadata and materializes data into the Target database.

Step 3.1: Data materialization from the ASCII files to the RDB
After reading data in the sequential files, data is materialized into the target relational database. The set values attributes' values are materialized into other tables in the target relational database as follows:

Given input: single value $(SD)_{ij}$ in Sequential file S_i
set value $((SV)_{ijk})_m$ in Sequential file S_i
output: tables T_1 to T_M within the target Relational database
For i = 1 to M do

```
begin
   For j = 1 to N do
     Get data (SD)ij containing all attributes' values of the record Rij from the sequential file Si;
       Insert record Rij by using (SD)ij;
If  Set values attributes exist for Rij
  then  begin For k = 1 to P do
              begin  For m = 1 to Q do
               begin   Get set value ((SV)ijk)m;
                 Insert (OKV)ij value and ((SV)ijk)m value into record (((RS)ij)k)m in table ((TS)ik)m;
             end;
          end;
       end;
   end;
```

Step 3.2: Data Materialization from the ASCII files to the OODB

After reading data in the sequential files, the data with non-associated key attributes is materialized to the target OODB. Non-associated key attributes with OIDs within each class are generated. The OID of every object in each subclass is updated as the OID of the same object in the superclass. The value of association attributes will be updated as the OID of their associated object in other class. The set values of objects are updated to hold a set of associated OIDs (i.e. stored OID) as follows:

```
Given: single value (SD)ij in Sequential file Si
          set value ((SV)ijk)m in Sequential file Si
          output: Classes C1 to CM within the target Object-oriented database
For i = 1 to M do
begin
  For j = 1 to N do
  Get data (SD)ij with all non-associated key attributes' values of object Oij from sequential file Si;
  Insert object Oij by using (SD)ij;
end;
end;
For i = 1 to M do
begin
 For j = 1 to N do
Get data (SD)ij with all non-associated key attributes' values of object Oij from sequential file   Si;
 Get object key (OK)ij with value (OKV)ij from the sequential file Si;
 Get object Oij in Class Ci;
 Get superclass and subclass relationship from the Frame model (FM) ;
 If Class Ci has superclass (CS)
 then begin
           Get OID (POID)ij of superclass object (SO)ij in (CS) by using the values (OKV)ij;
           Update the OID in object Oij with (POID) ij;
         end;
   end;
end;
For i = 1 to M do
begin
 For j = 1 to N do
Get data (SD)ij with all non-associated key attributes' values of object Oij from sequential file   Si;
       Get object Oij in Class Ci;
       If Oij has association attributes contained in Si
       then begin
              For k = 1 to P do
               Case association of
               One-to-one association with object Oij :
               begin
                  Get association attribute (AA) ijk with value (AAV)ijk from the sequential file Si;
                  Get OID (AOID) ijk of the associated object (AO) ijk  using the values (AAV) ijk;
                  Update blank association attribute (AA)ijk in object Oij with (AOID) ijk;
               end;
```

```
        One-to-many association with object O_ij :
        begin
         For m = 1 to Q
          Get association attribute(AA)_ijkm with value (AAV)_ijkm from sequential file S_i;
          Get OID (AOID)_ijkm of associated object (AO)_ijkm using values (AAV)_ijkm;
          Update association attribute (AA)_ijk in object O_ij with added (AOID)_ijkm;
         end;
        end;
        end;
       end;
      end;
     end;
end;
For i = 1 to M do
begin
 For j = 1 to N do
 Get data (SD)_ij with all non-associated key attributes' values of object O_ij from sequential file S_i;
 Get object O_ij in Class C_i;
  If O_ij has some set values attributes contained in S_i
  then begin
        For k = 1 to P do
          For m = 1 to Q do
            Get set value ((SV)_ijk)_m from sequential file S_i;
            If  set value ((SV)_ijk)_m is of base types
            then    Update set values attribute (SA)_ijk in object O_ij with added ((SV)_ijk)_m;
            else
               begin
                  Get OID (SOID)_ijkm of  set values object (SO)_ijkm in other class using
                  values ((SV)ijk)_m;
                  Update set values attribute (SA)_ijk in object O_ij with added (SOID)_ijkm;
               end;
          end;
        end;
      end;
   end;
end;
```

In summary, a database can be materialized without any loss of information if p maps a state of a relational database into an object-oriented database and p' maps a state of an object-oriented database into a relational database, then it can be shown that p(p'(R)) = R where R is the relational database before materialization.

6 Verification of data materialization rules

Information Capacity equivalence [42] is a formal approach taken by us to provide sets of equivalence preserving transformations of the schema and data materialization without information loss. A classification of generic integration and translation tasks based on their operation goals is defined by Rosenthal and Reiner. There are different levels of operational goals (G1, G2, G3, G4) in using database systems. They range from queries to update. We prove that the state of the database is the same for both the source schema (Object oriented database schema) and the target schema (Relational database schema). It is valuable to analyse whether the original schema and the transformed schema after the mapping function are in a dominance or equivalence domain. The mapping function to extend the information capacity is required to provide different levels of goals. Mapping functions can be classified into five categories that have different information capacity.

1. Functional

Let A and B be sets. The mapping is Functional if $f : A \to B, \forall a \in A, \exists b \in B \bullet f(a) = b$

2. Injective

If the inverse of the binary mapping relation is also Functional, the function is injective. $f^{-1} : B \to A, \forall b \in B, \exists a \in A \bullet f^{-1}(b) = a$

3. Total

If the Functional binary mapping relation is defined on every element of A, then the function is total. It is an information capacity preserving mapping between the instance of two schemas S1 and S2. (S1 denotes the original schema and S2 denotes the transformed schema.)

$f : I(S1) \rightarrow I(S2)$ where $I(Sn)$ denotes the set of all (data) instances of schema Sn

and $S1$ dominates $S2$ (i.e. $S2 \leq S1$), that is $\forall I(S1) \in S1, \exists I(S2) \in S2$

$f^{-1} : I(S2) \rightarrow I(S1)$

4. Surjective

If the inverse of the function is total, it is an information capacity preserving mapping between the (data) instance of two schema $S1$ and $S2$. This Total and injective function is called a Surjective function.

$f^{-1} : I(S2) \rightarrow I(S1)$ and $S2$ dominates $S1$ (i.e. $S1 \leq S2$), that is $\forall I(S2) \in S1, \exists I(S1) \in S1$

5. Bijection

If the mapping function meets all the above properties, it is an information equivalence preserving mapping.

$f : I(S1) \rightarrow I(S2)$ where $I(Sn)$ denotes the set of all (data) instances of schema Sn

and $S1$ and $S2$ are equivalent via f.

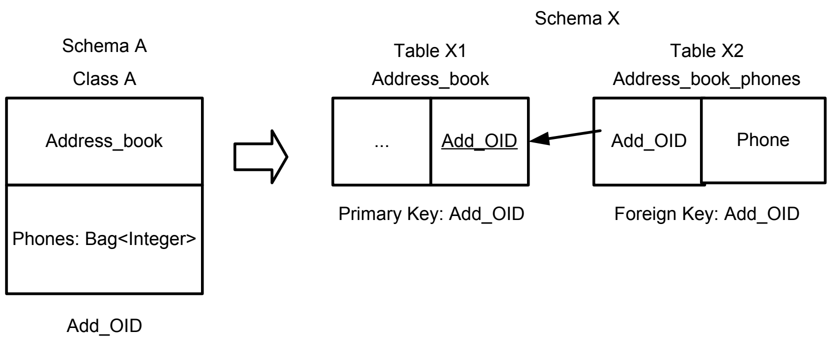
That is $\forall I(S1) \in S1, \exists I(S2) \in S2 \wedge \forall I(S2) \in S2, \exists I(S1) \in S1 \bullet S1 \equiv S2$

The four levels of operational Goals (G1, G2, G3 and G4) for systems involving two schemas and their relative information capacity are:

- G1 targets to make a query via S1 the data stored under S2, where S1 is a view of S2. In other words, S1 provides a logical view on the physical database of S2. Therefore, a Total function is required at instance level for achieving this minimum operational goal. As the query q on I(S1) is mapped to the unique query q on I(S2), the function does not have to be information preserving to achieve G1. The information capacities of S1 and S2 may be incommensurate.
- G2 can achieve G1. Moreover it can also be used to view through S1 the entire database stored under S2. A Total Injective function is required to achieve G2 because the function needs no loss of information where information preserving mapping is required to achieve G2. S1 must dominate S2.
- G3 can achieve G2. Moreover it can also be used to update the data stored under S2 using the view S1. A Total Injective and Surjective function is required. An update u that changes instance of S1, i1 to a new instance i1', that is u(i1) = i1'. Instance i1' should determine a unique instance of S2. As a result, f must be Surjective. As the function f must be an information preserving mapping to achieve G3, S2 must dominate S1.
- G4 targets to query through S1 that data stored under S2 and also through S2 the data stored under S1. A bi-direction and information preserving mapping in both direction is needed. S1 and S2 must be equivalent in both directions to allow updates be done through both S1 and S2. Function f is a Bijection function to achieve G4, S1 and S2 must be equivalent.

In our approach, a successful data materialization should require information capacity of the original schemas to be equivalent or dominated by the transformed schema. In order to achieve the information capacity needs, proof will be shown that each proposed data conversion process can fulfil up to the third level of its operational goal (G3) to ensure information completeness. The following three major steps must be preformed.

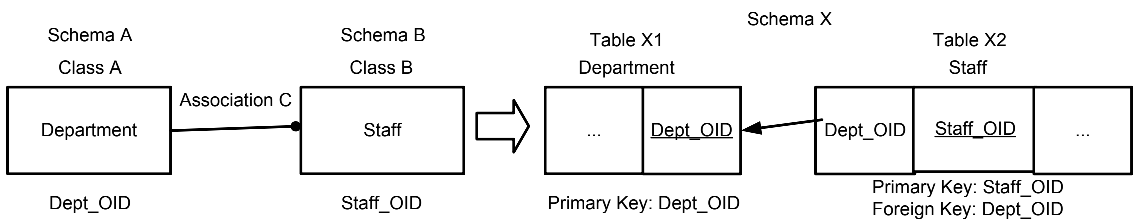
Step 1. Converting One-to-many association (Set Attributes)
Example:



Rule	Let C, D, E be the class A (where class A contains set attributes), table X1 and table X2 (where X2 is a generated table containing set attribute values) respectively. Let $SetAtt$ be a function to select the attribute type from the Class C attribute that is of set attribute type IF class A contains set attributes THEN begin $dom(D) \leftarrow dom(C) - SetAtt(C)$: $E \leftarrow \Pr iKey(C) \cup SetAtt(C)$: Cardinality $(D, E) \leftarrow 1:n$ $\forall e \in E, \exists d \in D \bullet ForKey(e) = \Pr iKey(d)$ end
Pre-cond.	$SetAtt(C) \neq \Phi$
Post-Cond	$dom(D) \leftarrow dom(C) - SetAtt(C)$: $E \leftarrow \Pr iKey(C) \cup SetAtt(C)$: Cardinality $(D, E) \leftarrow 1:n$ $\forall e \in E, \exists d \in D \bullet ForKey(e) = \Pr iKey(d)$
Proof: G1	Let $f:(C) \to (D.E)$ be the step to map class A to table X1 and X2 **Proof of f is functional** (D, E) provides a view to C. From the mapping rule, tuples in D and E can be found such that D has all the attribute type as C except the set attribute that formed by the function $SetAtt$. E contains the primary key C and the values of the set attribute type. $\forall c \in C \wedge SetAtt(C) \neq \Phi, \exists d \in D, e \in E \wedge dom(d) = dom(c) - SetAtt(c) \wedge$ $e = \Pr iKey(c) \cup SetAtt(c) \wedge RD(d, e) \bullet ForKey(e) = \Pr iKey(d)$ Hence f is functional **2. Proof of f is total** Since f is defined for every elements of (C, D), f is total.
G2 G3 G4	**3. Proof of f is Injective** (C, D) are mapped into (E, F) and preserved in (C, D). There is a bi-directional one to one mapping between elements of (C, D) and (E, F). $\forall$ (e,f) $\wedge$ (E,F) $\in$ RD(e, f) $\wedge$ e = *ForKey* (*f*) =*PriKey* (*e*), $\exists$ c $\in$ C, d$\in$C $\wedge$ RC(c,d) $\bullet$ $f^{1}(e, f) = (c, d)$ For every element in (E, F), the corresponding element in (C, D) can be found. Hence f is injective **4. Proof of f is Surjective** Since f^{-1} is defined for every elements of (E, F), f^{-1} is total. Hence f is surjective **5. Proof of f is a Bijection** From Proof 1, 2, 3 and 4, f is a bijection. (E, F) is proved to be equivalent to (C, D)

Step 2. Converting One-to-many association

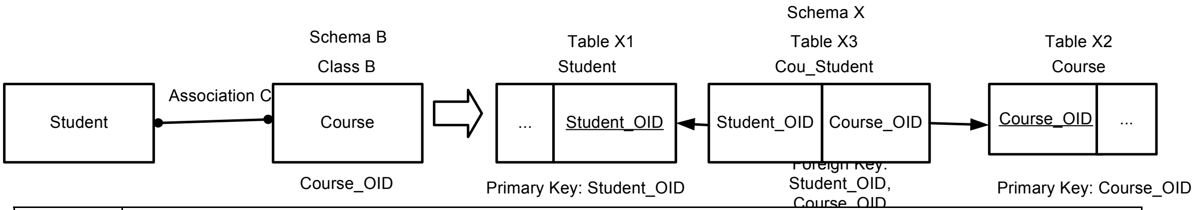
Example:



Rule	Let C, D, E, F be the class A, class B (where class A is on the "one" side and class B is on the "many" side), table X1 and table X2 (where class X1 is on the "one" side and class X2 is on the "many" side) respectively IF class A has a one to many relationship with class B THEN begin $E \leftarrow C : F \leftarrow D$: Cardinality$(E,F) \leftarrow 1:n$ $\forall f \in F, \exists e \in E \bullet ForKey(f) = \Pr iKey(e)$ end
Pre-cond.	Association $(C,D) \leftarrow 1:n$: $\exists c \in C, d \in D \bullet RC(c,d)$ and RC is a one to many association.
Post-Cond	$E \leftarrow C : F \leftarrow D$: Cardinality$(E,F) \leftarrow 1:n$ $\exists e \in E, f \in F \bullet RD(e,f)$ and RD is a one to many relation $\forall f \in F, \exists e \in E \bullet ForKey(f) = \Pr iKey(e)$
Proof: G1 G2	Let $f:(C,D) \rightarrow (E.F)$ be the step to merge class A and B to table X1 and X2 **Proof of f is functional** (E,F) provides a view to (C,D). From the mapping rule, tuples in E and F can be found that tuples in E has a one to many association with tuples in F, and tuples in E has the same attributes as class C, and tuples in F has the same attributes as class D. $\forall (c,d) \in (C,D) \wedge RC(c,d), \exists e \in E, f \in F \wedge RD(e,f) \wedge ForKey(f) = \Pr iKey(e)$ $\bullet f(c,d) = (e,f)$ Hence f is functional **2. Proof of f is total** Since f is defined for every elements of (C,D), f is total. **3. Proof of f is Injective** (C,D) are mapped into (E,F) and are preserved in (C,D). There is a bi-directional one to one mapping between elements of (C,D) and (E,F). $\forall (e,f) \in (E,F) \wedge RD(e,f) \wedge e = ForKey(f) = \Pr iKey(e),$ $\exists c \in C, d \in C \wedge RC(c,d)$ $\bullet f^{-1}(e,f) = (c,d)$ For every element in (E,F), the corresponding element in (C,D) can be found. Hence f is injective
G3 G4	**4. Proof of f is Surjective** Since f^{-1} is defined for every elements of (E,F), f^{-1} is total. Hence f is surjective **5. Proof of f is a Bijection** From Proof 1, 2, 3 and 4, f is a bijection. (E,F) is proved to be equivalent to (C,D) **5. Proof of f is a Bijection** From Proof 1, 2, 3 and 4, f is a bijection. (E,F) is proved to be equivalent to (C,D)

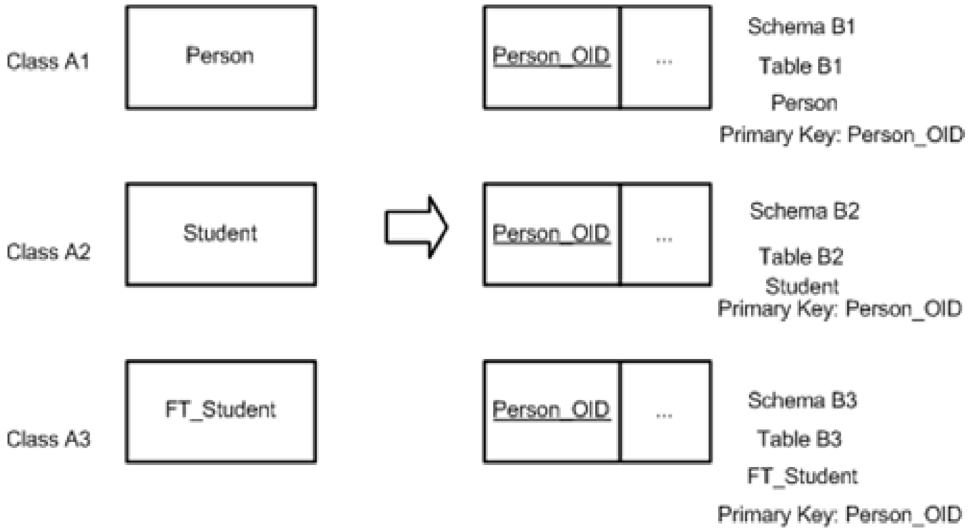
Step 3 Converting many-to-many association

Example:



Rule	Let C, D, E, F, G be the class A, class B, table X1 and table X2 and X3 (containing primary key of X1 and X2) respectively IF class A has a many to many association with class B THEN begin $E \leftarrow C \; : \; F \leftarrow D \; : \; G \leftarrow \Pr iKey(E) \cup \Pr iKey(F)$ Cardinality $(E, G) \leftarrow 1:n \; : \; \forall g \in G, \exists e \in E \bullet ForKey(g) = \Pr iKey(e)$ Cardinality $(F, G) \leftarrow 1:m \; : \; \forall g \in G, \exists f \in F \bullet ForKey(g) = \Pr iKey(f)$ end
Pre-cond.	Cardinality $(C, D) \leftarrow n:m \; : \exists c \in C, d \in D \bullet RC(c, d)$ and RC is a many to many relation
Post-Cond	$E \leftarrow C \; : \; F \leftarrow D$: Cardinality $(E, F) \leftarrow 1:n$ $\exists e \in E, g \in G \bullet RD(e, g)$ and RD is a one to many relationship $\forall g \in G, \exists e \in E \bullet ForKey(g) = \Pr iKey(e)$ $\exists f \in E, g \in G \bullet RE(f, g)$ and RE is a one to many relationship $\forall g \in G, \exists f \in F \bullet ForKey(g) = \Pr iKey(f)$
Proof: G1 G2 G3 G4	Let $f : (C, D) \rightarrow (E.F, G)$ be the step to map class A and B to table X1, X2 and X3 **Proof of f is functional** (E, F, G) provides a view to (C, D). From the mapping rule, tuples in E, F, G can be found that tuples in E has a one to many association with tuples in G, tuples in F has a one to many association with tuples in G, tuples in E has the same attributes as class C, and tuples in F has the same attributes as class D. $\forall (c, d) \in (C, D) \wedge RC(c, d), \exists (e, f, g) \in (E, F, G) \wedge RD(e, g) \wedge RE(f, g) \wedge$ $ForKey(g) = \Pr iKey(f) \wedge ForKey(g) = \Pr iKey(f) \bullet f(c, d) = (e, f, g)$ Hence f is functional **2. Proof of f is total** Since f is defined for every elements of (C, D), f is total. **3. Proof of f is Injective** (C, D) are mapped into (E, F, G) and preserved in (C, D). There is a bi-directional one to one mapping between elements of (C, D) and (E, F, G). $\forall (e, f, g) \in (E, F, G) \wedge RD(e, g) \wedge RE(f, g) \wedge ForKey(g) = \Pr iKey(f) \wedge$ $ForKey(g) = \Pr iKey(f), \exists c \in C, d \in C \wedge RC(c, d) \bullet f^{-1}(e, f, g) = (c, d)$ For every element in (E, F, G), the corresponding element in (C, D) can be found. Hence f is injective **4. Proof of f is Surjective** Since f^{-1} is defined for every elements of (E, F, G), f^{-1} is total, Hence f is surjective **5. Proof of f is a Bijection** From Proof 1, 2, 3 and 4, f is a bijection. (E, F, G) is proved to be equivalent to (C, D)

Step 4. Converting Inheritance
Example:

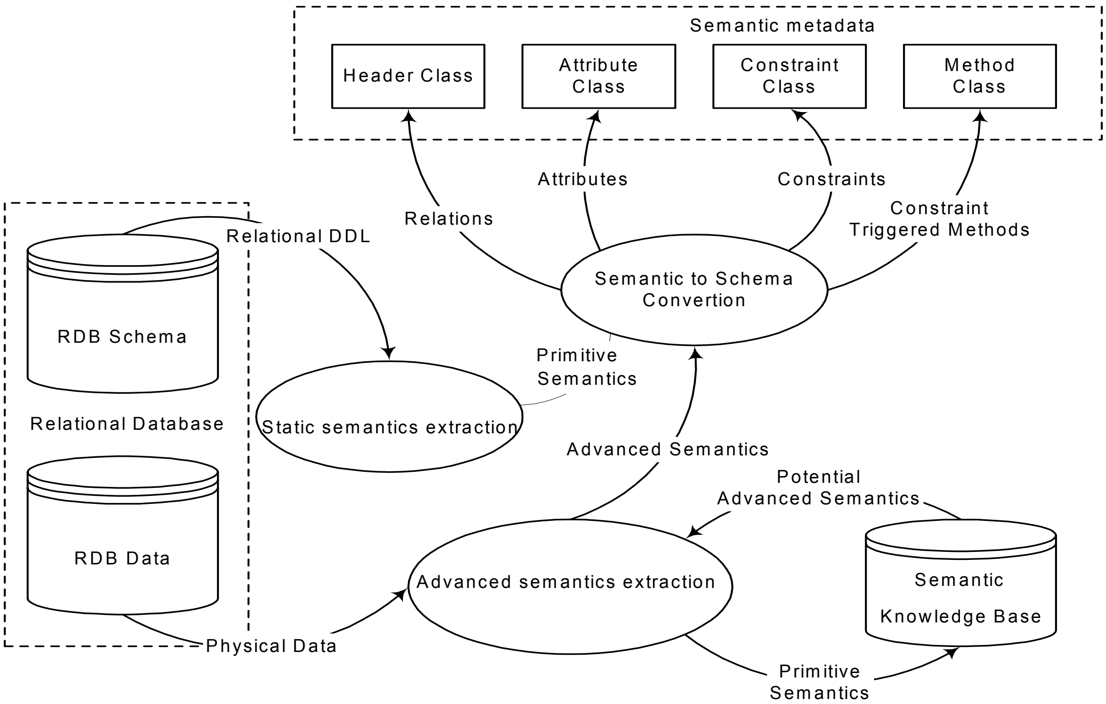


Rule	Let $A_1, A_2, A_3 \cdots A_n$ be classes (A_n "is-a" $A_{n-1} \ldots A_2$ "is-a" A_1) and tables $B_1, B_2, B_3 \cdots B_n$ are mapped from them and OID of A_1 be A_1_oid. IF $A_1, A_2, A_3 \cdots A_n$ classes (A_n "is-a" $A_{n-1} \ldots A_2$ "is-a" A_1) THEN begin $B_1 \leftarrow A_1 \cdots B_n \leftarrow A_n$: $\Pr iKey(B_1) = \cdots \Pr iKey(B_n) = A_1\_oid$ end
Pre-cond.	$A_1, A_2, A_3 \cdots A_n$ classes (A_n "is-a" $A_{n-1} \ldots A_2$ "is-a" A_1)
Post-Cond	$B_1 \leftarrow A_1 \cdots B_n \leftarrow A_n$: $\Pr iKey(B_1) = \cdots \Pr iKey(B_n) = A_1\_oid$
Proof: **G1** **G2** **G3**	Let $f_i : A_i \rightarrow B_i$ be a set of steps to map classes $A_1, A_2, A_3 \cdots A_n$ to tables $B_1, B_2, B_3 \cdots B_n$. **Proof of f_i is functional** B_i provides a view to A_i. From the mapping rule, table B_i can be found such that it has the primary key A_1_oid and the same attributes as class A_i. $\forall a_i \in A_i, \exists b_i \in B_i \wedge \Pr iKey(b_i) = A_1\_oid \bullet f_i(a_i) = b_i$ Hence f_i is functional **2. Proof of f_i is total** Since f_i is defined for every elements of A_i f is total. **3. Proof of f_i is Injective** A_i is mapped into B_i and preserved in A_i. There is a bi-directional one to one mapping between elements of A_i and B_i. $\forall b_i \in B_i \wedge \Pr iKey(b_i) = A_i\ \_oid, \exists a_i \in A_i \bullet f_i^{-1}(b_i) = a_i$ For every element in B_i, the corresponding element in A_i can be found. Hence f is injective **4. Proof of f_i is Surjective**

G4	Since f_i^{-1} is defined for every elements of B_i, f_i^{-1} is total. Hence f_i is surjective **5. Proof of f_i is a Bijection** From Proof 1, 2, 3 and 4, f_i is a bijection. **6. Proof of $(A_1, A_2, A_3 \cdots A_n)$ is equivalent $B_1, B_2, B_3 \cdots B_n$** From Proof 5, all f_i are bijections, hence $(A_1, A_2, A_3 \cdots A_n)$ is equivalent $B_1, B_2, B_3 \cdots B_n$

7 Prototypes

The flow chart for the schema translation from the RDB to the semantic metadata is shown in figure 3. It examines data occurrence to capture advanced semantics in the RDB. A semantic knowledge base is required as an intermediate storage for all primitive semantics of the is_a and cardinality relationship. When the data examination process is completed with the semantics were extracted, the captured semantics are translated into the semantic metadata



.Figure 3 Flow chart for Schema translation from a RDB to the Semantic metadata

The data flow diagram of the schema translation from the semantic metadata to the OODB is shown in figure 4. The Header and Attribute classes in the frame model contain class entities and attribute data types. They form the basic class structure in the OODB. Advanced semantics are translated into the methods in the OODB schema through the Event Extraction Process and Method Extraction Process.

Case study: Materializing RDB to OODB:

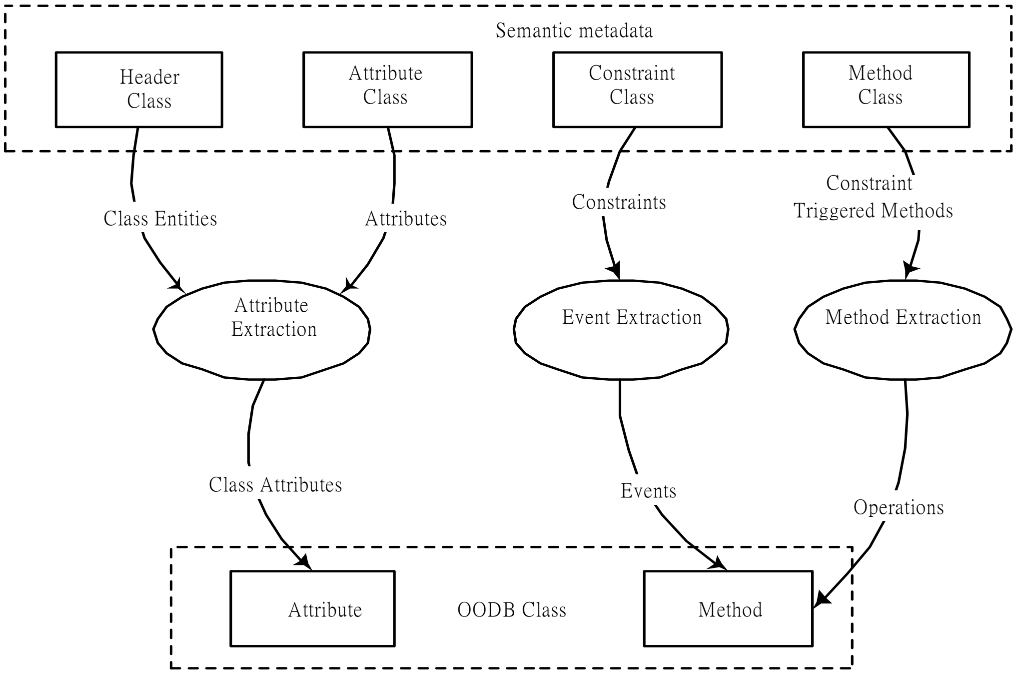


Figure 4 Data flow diagram of mapping semantic metadata to OODB schema

The whole database resides in the Microsoft SQL Server 7.0. Notice that underline means primary key and * means foreign key. The relational schema is listed below :

Relation Patient	(HKID, Patient_name)
Relation Borrower	(*Borrower_no, Borrower_name)
Relation Borrow	(*Borrower_no, *Folder_no)
Relation Medical_rec	(Medical_rec_no, Create_date, Sub_type, *Folder_no)
Relation Ward_rec	(*Medical_rec_no, Ward_no, Admission_date, Discharge_date)
Relation Outpatient_rec	(*Medical_rec_no, OPD_no, Specialty)
Relation AE_record	(*Medical_rec_no, AE_no)
Relation Department	(Borrower_no, Department_name)
Relation Doctor	(Borrower_no, Doctor_name)
Relation Other-hospital	(Borrowe_no, Hospital_name)
Relation Record_folder	(Folder_no, location, *HKID)
Relation Loan_history	(*Borrower_no, *Folder_no, Loan_date)
Relation Insurance_cover	(Insurance_nm, *HKID)

The data contained in the database is as follows:

Table Doctor

Borrower_no	Doctor_name
1	Bloor
2	Smith
3	Kim
4	Chitson
5.	Navathe

Table Other_hospital

Borrow_no	Hospital_name
21	Mac Neal
22	Riveredge
23	Stone Town
24	North Community
25	Golden Park

Table Department

Borrower_no	Department_name
11	X-Ray
12	Infant
I3	Chest
I4	Skin
I5	Therapy

Table Borrower

Borrower_no	Borrower_name
1	Bloor
2	Smith
3	Kim
11	X-Ray
12	Infant
I4	Skin
21	Mac Neal
22	Riveredge
25	Golden Park

Table Record_folder

Folder_no	Location	HKID
F_21	Hong Kong	E3766849
F_22	Kowloon	E8018229
F_23	New Territories	E6077888

Table Insurance_cover

Insurance_no	HKID
I_1	E3766849
I_2	E8018229

Table Patient

HKID	Patient_name
E3766849	Smith
E8018229	Bloor
E6077888	Kim

Table AE_Record

Medical_rec_no	AE_no
M_352001	AE_1
M_362001	AE_2

Table Loan_history

Borrower_no	Folder_no	Loan_date
1	F_21	Jan-10-2000
1	F_21	Jan-10-2001
2	F_22	Sep-29-1999
11	F_21	Jun-12-1999
12	F_22	Jan-7-2000
14	F_23	Jan-11-2000
21	F_21	Feb-1-2001
22	F_21	Mar-03-2001

Table Medical_rec

Medical_rec_no	Create_date	Sub_type	Folder_no
M_311999	Jan-1-1999	W	F_21
M_322000	Feb-2-2000	O	F_21
M_331998	Nov-10-1998	W	F_22
M_341999	Dec-20-1999	O	F_22
M_352001	Jan-15-2001	O	F_21
M_362001	Feb-01-2001	O	F_21
M_382001	Feb-22-2001	O	F_23

Table Ward_rec

Medical_rec_no	Ward_no	Admission_date	Discharge_date
M_311999	W_41	Jan-1-1999	Mar-20-1999
M_322000	W_43	Nov-12-1998	Dec-14-1998

Table Outpatient_rec

Medical_rec_no	OPD_no	Specialty
M_331998	O_51	Heart
M_341999	O_52	Ophthalmic
M_382001	O_53	Therapy

Table Borrow

Borrower_no	Folder_no
1	F_21
1	F_22
2	F_22
3	F_23
11	F_21
12	F_22
14	F_23
21	F_21
22	F_21
25	F_23

Step 1. Schema Translation from the Relational Database to the semantic metadata

After capturing data semantics, we get an overview of the EER model as shown in Figure 5 where C = categorization and G = disjoint generalization.

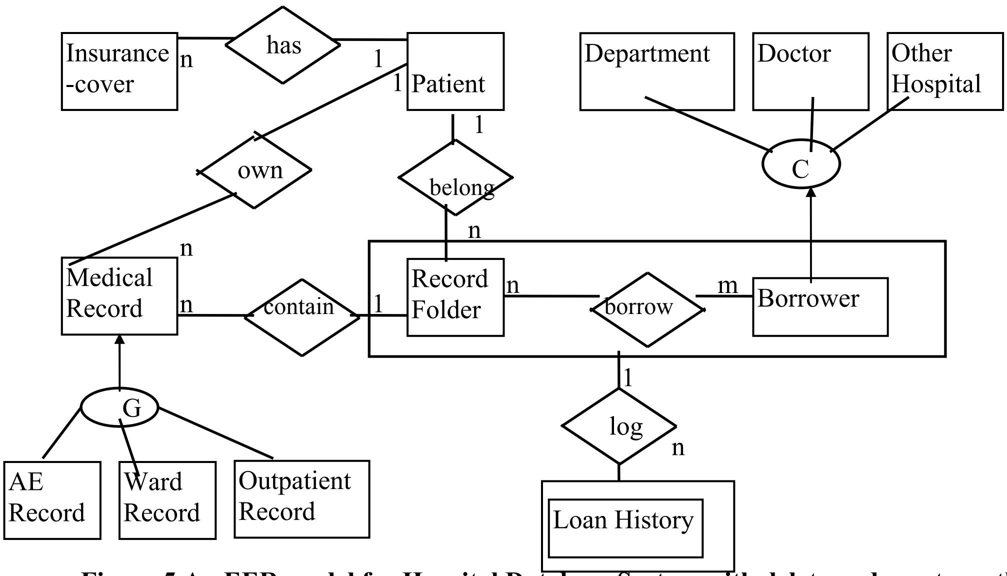


Figure 5 An EER model for Hospital Database System with delete and create method

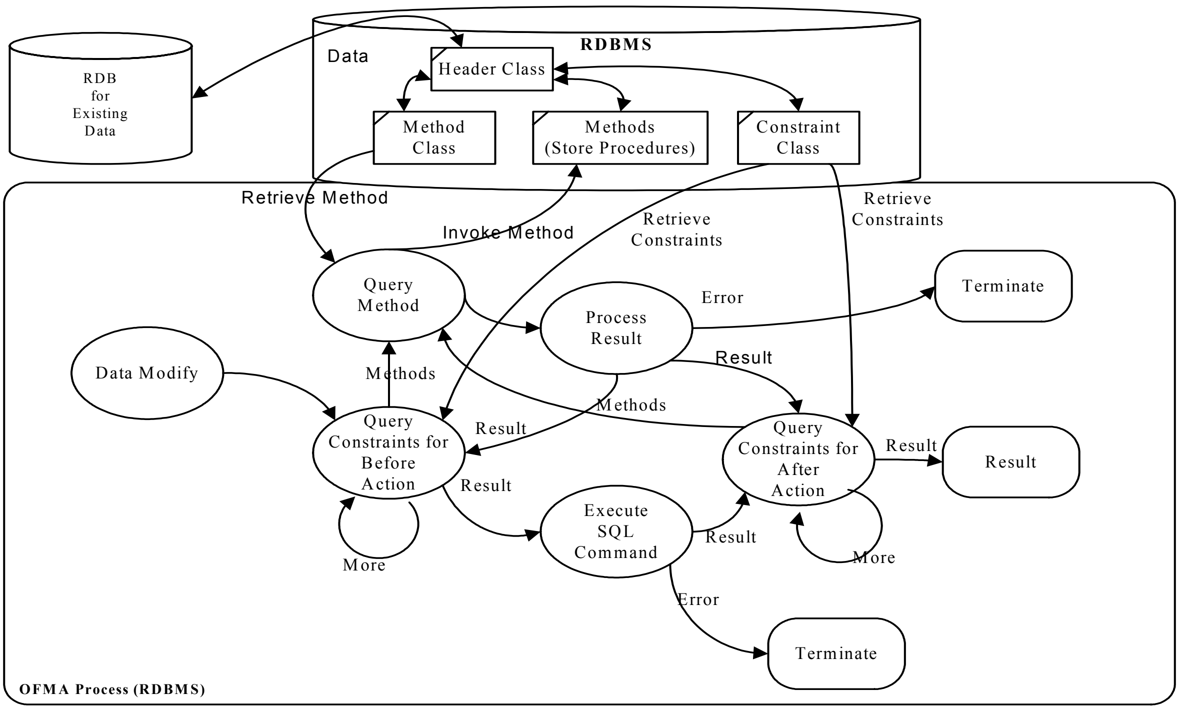


Figure 6 Data flow diagram of semantic metadata

The translated OODB schema is shown as follows:

```
Class ApplicationObject
 super: Composite
 description: "Hospital Record tracking application"
{};

Class Doctor
 super: ApplicationObject
 description: "Holds details of a doctor"
{
  class:
  Doctor    find(Integer in_no );
 instance:
  Integer    Borrower_no    unique:;
  String    Doctor_name;
};

Class Other_hospital
 super: ApplicationObject
 description: "Holds details of other hospital borrower"
{
```

```
 class:
   Other_hospital     find(Integer in_no );
 instance:
   Integer      Borrower_no      unique:;
   String       Hospital_name;
};

Class Department
 super: ApplicationObject
 description: "Holds details of a department"
{
 class:
   Department     find(Integer in_no );
 instance:
   Integer      Borrower_no      unique:;
   String       Department_name;
};

Class Borrower
 super: Doctor, Department, Other_hospital
 description: "Holds details of a borrower"
{
 instance:
   String       Borrower_name;
   Bag<Borrow>              Borrow_rec;
};

Class Borrow
 super: ApplictionObject
 description: "Holds details of a borrow transaction"
{
 class:
   Borrow      find(Borrower in_b, Record_folder in_p);
 instance:
   Borrower                 Borrow_rec;
   Record_folder            Rec_borrow;
   Bag<Loan_history>        Borrow_history;
 };

Class Loan_history
 super: ApplictionObject
 description: "Holds details of a past borrow records"
{
 class:
   Loan_history              find(Borrower in_b, Loan_history in_p);
 instance:
   Date                 Return_date;
   Borrow                   Borrow_history;
};

Class Record_folder
 super: ApplicationObject
 description: "Holds details of a record folder"
{
 class:
   Record_folder      find(Integer in_no );
 instance:
   Integer      Folder_no      unique:;
   String       Location,
   Patient                       Patient_record;
   Bag<Borrow>                  Rec_borrow;
   Bag<Medical_rec>             Med_folder;
   Bag<Insurance_mn>            Coverage;
};

Class Insurance_cover
 super: ApplicationObject
 description: "Holds details of a patient insurance record"
{
 class:
   Insurance-cover      find(Integer in_no );
 instance:
   Integer      Insurance_no      unique:;
   Patient      Coverage
};

Class Medical_rec
super: ApplicationObject;
 description: "Holds details of a Medical record"
{
 class:
   Medical_rec find(Integer in_no );
 instance:
   Integer      Medical_no      unique:;
   Date         Create_date;
   String       Sub_type;
   Patient                          Contain_med;
   Record_folder                    Med_folder;
};

Class Patient
Description: "Holds details of patient records"
{
 class:
   Patient find(integer in_no);
 Instance:
   String                              HKID;
   String
          Patient_name;
   Bag<record_folder>
          Patient_record;
   Bag<Insurance_cover>             Coverage;
    Bag<medical_rec>
          Contain_Med;
};

Class Ward_rec
 super: Medical_rec;
 description: "Holds details of a ward record"
{
 class:
   Ward_rec  find(Integer in_no );
 instance:
   Integer     Ward_no;
   Date        Admission_date;
```

```
  Date        Discharge_date;
};

Class Outpatient_rec
 super: Medical_rec;
 description: "Holds details of a outpatient record"
{
  class:
    Outpatient_rec  find(Integer in_no );
  instance:
    Integer     OPD_no;
```

```
    String      Specialty;
};

Class AE_record
 super: Medical_rec;
 description: "Holds details of a AE record"
{
  class:
    AE_rec  find(Integer in_no );
  instance:
    Integer     AE_no;
};
```

Step 2. Data capture from the RDB into the ASCII files

ASCII files are formed with one ASCII file for each relation. "*^*" is a separator within a particular ASCII file to separate the different records. An example of the ASCII file "Doctor" is shown below.

ASCII File Name: Doctor
O: R:Borrower_no:1 R:Doctor_name:Doctor 1 *^* O: R:Borrower_no:2 R:Doctor_name:Doctor 2

Step 3. Data materialization from the ASCII files to the OODB

After the data capture process, the Target database and created classes as shown in figure 7 where Borrower is a subclass of Doctor, Department and Other_hospital. The same object appears both in its superclass and in its subclass (i.e. Borrower object with the Oid #3:1 is the same object as Doctor object with the Oid #1:1) as shown below:

Class Doctor (C1)

Oid	Borrower_no	Doctor_name
#1:1	1	Bloor
#1:2	2	Smith
#1:3	3	Kim
#1:4	4	Chitson
#1:5	5	Navathe

Class Department (C2)

Oid	Borrower_no	Department_name
#2:1	11	X-Ray
#2:2	12	Infant
#2:3	13	Chest
#2:4	14	Skin
#2:5	15	Therapy

Class Other_hospital(C10)

Oid	Borrow_no	Hospital_name
#10:1	21	Mac Neal
#10:2	22	Riveredge
#10:3	23	Stone Town
#10:4	24	North Community
#10:5	25	Golden Park

Class Borrow (C4)

Oid	Borrow_no	Folder_no
#4:1	#1:1	#5:1
#4:2	#1:2	#5:2
#4:3	#2:1	#5:1
#4:4	#2:2	#5:2
#4:5	#1:1	#5:2
#4:6	#10:1	#5:1
#4:7	#10:2	#5:1
#4:8	#1:3	#5:3
#4:9	#2:4	#5:3
#4:10	#10:5	#5:3

Class Insurance_cover (C12)

Oid	Insurance_no	Coverage
#12:1	I_1	#13:1
#12:2	I_2	#13:2

Class AE (C11)

Oid	Medical_rec_no	AE_no
#6:5	M_352001	AE_1
#6:6	M_362001	AE_2

Class Borrower (C3)

Oid	Borrower_no	Borrower_name	Borrow_no
#1:1	1	Bloor	{#4:1,#4:5}
#1:2	2	Smith	{#4:2}
#1:3	3	Kim	{#4:8}
#2:1	11	X-Ray	{#4:3}
#2:2	12	Infant	{#4:4}

#2:4	14	Skin	{#4:9}
#10:1	21	Smith	{#4:6}
#10:2	22	Bloor	{#4:7}
#10:5	25	Golden Park	{#4:10}

Class Loan_history (C9)

Oid	Loan_date	Borrow-history
#9:1	Jan-10-2000	#4:1
#9:2	Jan-10-2001	#4:1
#9:3	Sep-29-1999	#4:2
#9:4	Jun-12-1999	#4:3
#9:5	Jan-7-2000	#4:4
#9:6	Jan-11-2000	#4:5
#9:7	Feb-1-2001	#4:6
#9:8	Mar-03-2001	#4:7

Class Record-folder(C5)

Oid	Folder_no	Location	Patient_record	Rec-borrow	Med_folder
#5:1	F_21	Hong Kong	#13:1	{#4:1,#4:3,#4:6,#4:7}	{#6:1,#6:2,#6:5,#6:6}
#5:2	F_22	Kowloon	#13:2	{#4:2,#4:4,#4:5}	{#6:3,#6:4}
#5:3	F_23	New Territories	#13:3	{#4:8,#4:9,#4:10}	{#6:7}

Class Patient (C13)

Oid	HKID	Patient_name	Coverage	Contain_med	Patient_record
#13:1	E3766849	Smith	#12:1	{#6:1,#6:2,#6:5,#6:6}	#5:1
#13:2	E8018229	Bloor	#12:2	{#6:3,#6:4}	#5:2
#13:3	E6077888	Kim	#12:1	{#6:7}	#5:3

Class Medical_rec (C6)

Oid	Medical_rec_no	Create_date	Sub_type	Med_folder	Contain-med
#6:1	M_311999	Jan-1-1999	W	#5:1	#13:1
#6:2	M_322000	Feb-2-2000	O	#5:1	#13:1
#6:3	M_331998	Nov-10-1998	W	#5:2	#13:2
#6:4	M_341999	Dec-20-1999	O	#5:2	#13:2
#6:5	M_352001	Jan-15-2001	O	#5:1	#13:1
#6:6	M_362001	Feb-01-2001	O	#5:1	#13:1
#6:7	M_382001	Feb-22-2001	O	#5:3	#13:3

Class Ward_rec (C7)

Oid	Medical_rec_no	Ward_no	Admission_date	Discharge_date
#6:1	M_311999	W_41	Jan-1-1999	Mar-20-1999
#6:2	M_322000	W_43	Nov-12-1998	Dec-14-1998

Class Outpatient_rec (C8)

Oid	Medical_rec_no	OPD_no	Specialty
#6:3	M_331998	O_51	Heart
#6:4	M_341999	O_52	Ophthalmic
#6:7	M_382001	O_53	Therapy

Inverse materializing OODB back to original RDB for validation

A Hospital Database Systems (HDS) is also implemented in the JASMINE ODBMS. The details of the object oriented schema is the same as the materialized OODB schema as derived in the case study from RDB to OODB.

Step 1. Schema translation from OODB to semantic metadata

After doing the seven steps in the semantic discovery, we can recover semantics from OODB and map them into corresponding RDB schema as follows:
The translated relational schema is:

Relation Patient	(Patient_Oid, HKID, Patient_name)
Relation Borrower	(*Borrower_Oid, Borrower_no, Borrower_name)
Relation Borrow	(Borrow_Oid, *Borrower_Oid, *Record_folder_Oid)
Relation Medical_rec	(Medical_rec_Oid, Medical_rec_no, Create_date, Sub_type,

	*Patient_Oid, *Record_folder_Oid)
Relation Ward_rec	(*Medical_rec_Oid, Medical_rec_no, Ward_no, Admission_date, Discharge_date)
Relation Outpatient_rec	(*Medical_rec_Oid, Medical_rec_no, OPD_no, Specialty)
Relation AE_record	(*Medical_rec_Oid, AE_no)
Relation Department	(Borrower_Oid, Borrower_no, Department_name)
Relation Doctor	(Borrower_Oid, Borrower_no, Doctor_name)
Relation Record_folder	(Record_folder_Oid, Folder_no, location, *Patient_Oid)
Relation Other-hospital	(Borrower_Oid, Borrowe_no, Hospital_name)
Relation Loan_history	(Loan_history_Oid, Return_date, *Borrow_Oid)
Relation Insurance_cover	(Insurance_Oid, Insurance_nm, *Patient_Oid)

Step 2. Data capture from the OODB to the ASCII files

Source data in the Object oriented database are the same as the converted OODB in case study 1. ASCII files are formed, one for each class. An example of the ASCII file "Doctor" is shown below.

ASCII File Name: Doctor
O:OID:#1:1 R:Borrower_no:1 R:Doctor_name:Doctor 1 *^* O:OID:#1:2 R:Borrower_no:2 R:Doctor_name:Doctor 2

Step 3. Data materialization from the ASCII files to the RDB

After data materialization process, the Target database include data as follows:

Table Doctor (D1)

Borrower_Oid	Borrower_no	Doctor_name
#1:1	1	Bloor
#1:2	2	Smith
#1:3	3	Kim
#1:4	4	Chitson
#1:5	5	Navathe

Table Other_hospital (D10)

Borrower_Oid	Borrow_no	Hospital_name
#10:1	21	Mac Neal
#10:2	22	Riveredge
#10:3	23	Stone Town
#10:4	24	North Community
#10:5	25	Golden Park

Table Doctor (D1)

Borrower_Oid	Borrower_no	Doctor_name
#1:1	1	Bloor
#1:2	2	Smith
#1:3	3	Kim
#1:4	4	Chitson
#1:5	5	Navathe

Table Borrower (D3)

Borrower_Oid	Borrower_no	Borrower_name
#1:1	1	Bloor
#1:2	2	Smith
#1:3	3	Kim
#2:1	11	X-Ray
#2:2	12	Infant
#2:4	14	Skin
#10:1	21	Mac Neal
#10:2	22	Riveredge
#10:5	25	Golden Park

Table Patient (D13)

Patient_Oid	HKID	Patient_name
#13:1	E3766849	Smith
#13:2	E8018229	Bloor
#13:3	E6077888	Kim

Table Insurance_cover (D12)

Oid	Insurance_no	Coverage
#12:1	I_1	#13:1
#12:2	I_2	#13:2

Table AE_Record (D11)

Medical_rec_Oid	Medical_rec_no	AE_no
#6:5	M_352001	AE_1
#6:6	M_362001	AE_2

Table Borrow (D4)

Borrow_Oid	Borrower_Oid	Record_folder_Oid
#4:1	#1:1	#5:1
#4:2	#1:2	#5:2
#4:3	#2:1	#5:1
#4:4	#2:2	#5:2
#4:5	#1:1	#5:2
#4:6	#10:1	#5:1

#4:7	#10:2	#5:1
#4:8	#1:3	#5:3
#4:9	#2:4	#5:3
#4:10	#10:5	#5:3

Table Loan_history (D9)

Loan_Oid	Loan_date	Borrow-history
#9:1	Jan-10-2000	#4:1
#9:2	Jan-10-2001	#4:1
#9:3	Sep-29-1999	#4:2
#9:4	Jun-12-1999	#4:3
#9:5	Jan-7-2000	#4:4
#9:6	Jan-11-2000	#4:5
#9:7	Feb-1-2001	#4:6
#9:8	Mar-03-2001	#4:7

Table Record_folder (D5)

Record_folder_Oid	Folder_no	Location	Patient_Oid
#5:1	F_21	Hong Kong	#13:1
#5:2	F_22	Kowloon	#13:2
#5:3	F_23	New Territories	#13:3

Table Medical_rec (D6)

Medical_rec_Oid	Medical_rec_no	Create_date	Sub_type	Record_folder_Oid	Patient_Oid
#6:1	M_311999	Jan-1-1999	W	#5:1	#13:1
#6:2	M_321999	Feb-2-2000	O	#5:1	#13:1
#6:3	M_331998	Nov-10-1998	W	#5:2	#13:2
#6:4	M_341999	Dec-20-1999	O	#5:2	#13:2
#6:5	M_352001	Jan-15-2001	O	#5:1	#13:1
#6:6	M_362001	Feb-01-2001	O	#5:1	#13:1
#6:7	M_382001	Feb-22-2001	O	#5:3	#13:3

Table Ward_rec (D7)

Medical_rec_Oid	Medical_rec_no	Ward_no	Admission_date	Discharge_date
#6:1	M_311999	W_41	Jan-1-1999	Mar-20-1999
#6:2	M_321999	W_43	Nov-12-1998	Dec-14-1998

Table Outpatient_rec (D8)

Medical_rec_Oid	Medical_rec_no	OPD_no	Specialty
#6:3	M_331998	O_51	Heart
#6:4	M_341999	O_52	Ophthalmic
#6:7	M_382001	O_53	Skin

The figure 7 shows the data materialization is feasible.

8 Conclusion

This paper offers a methodology translating the schema among different data models using the semantic metadata. The metadata captures the semantics from legacy database. It consists of classes to preserve constraints after the schema translation, and sequential files to store actual data. It invokes the program call to emulate the method of the OODB, and translates legacy database schema into semantic metadata and to another target legacy database schema upon users request. We present a unique semantic metadata to capture data semantics as well as data operation in a metadata which acts as a knowledge-base system by resolving constraints conflicts between different data models in a heterogeneous database environment. After capturing data and their semantics in a metadata, a united data materialization for the RDB and the OODB using the semantic metadata is shown to be feasible. With captured semantics and data stored in the semantic metadata, the data can be materialized into the relational database or the object oriented database upon users request. The resulting target database maintains the same semantics in the source database. Using the semantic metadata and data contained in the ASCII file, the data can be materialized to the Relational database or the Object oriented database concurrently.

A validation on data materialization by use of information capacity has been done to show that the semantics of a relational database can be recovered and preserved after materialization. Also the original relational database can be recovered from the object-oriented database without any loss of information. This shows the logical equivalence between the relational database and the materialized object-oriented database. Such validation is needed to convince the user of the correctness of the materialization process and thus suggest that it is feasible to use a single data set for a concurrent relational view and object-oriented view of an object-relational database. The future research of this paper is universal data warehousing.

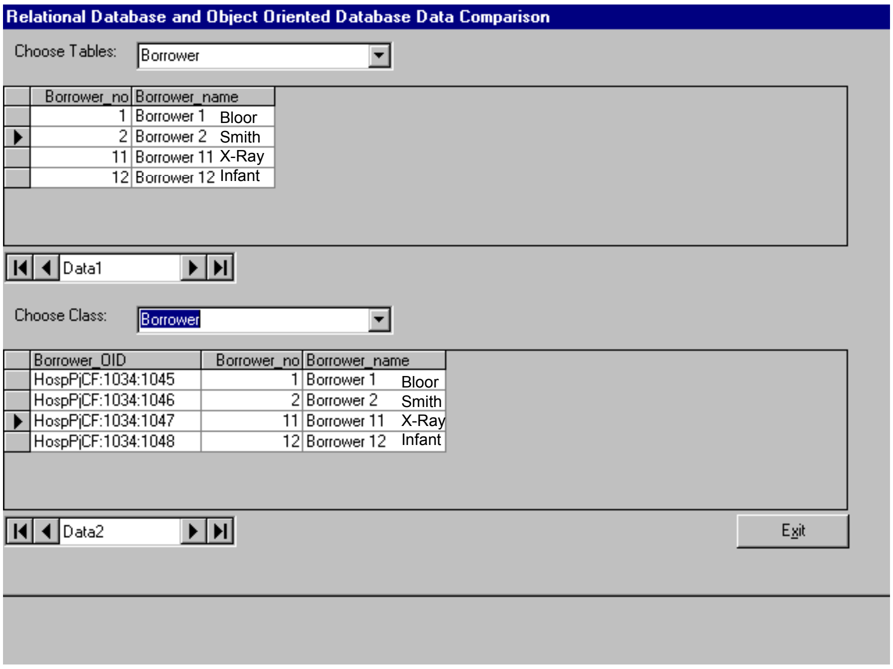


Figure 7 Comparison between RDB and OODB for the same dataset

9 References

1. Wong, D. (2001), "A Knowledge-based system for Relational Database and Object-Oriented Database Schema Translation using Frame Meta-data Model". MPhil Thesis at City University of Hong Kong, 2001.
2. Ayache, M. and Flory, A. (1995), "Transforming Conceptual ER schema into OO Databases", Proceeding of the 2nd International Workshop on Advances in Databases and Information, Moscow,1995, p. 143-157.
3. Fahrner, Christian. and Vossen, Gottfried. (1995), "A survey of database design transformations based on the Entity-Relationship model", Data & Knowledge Engineering, Vol. 15, 1995, p. 213-250
4. Siu, Brian and Cheung, T. Y. (1996), "Towards a Method for Schema Translation from Relational to Object-Oriented Databases", Proceedings of 7th International Hong Kong Computer Society Database Workshop May 1996, Springer, p. 307-323.
5. Behm, A., Geppert, A. and Dittrich, K. R. (1997), "Deductive Entity Relationship Modelling", IEEE Transaction on Knowledge and Data Engineering. Vol. 5, No. 3, 1997, p. 439-450.
6. Miura, T., and Shioya, I. (1997), "Type Approximation for schema Discovery", Proceedings of the 5th International Conference on Re-Technologies for Information Systems (RE-TIS'97), Australia, 1997, p. 117-130.
7. Li, S. H., Huang, S. M. and Chen, H. H. (1997), "Discovering Missing Semantics from Existing Relational Databases", Proceeding of 8th International HKCS Database Workshop, Hong Kong, 1997, p. 275-285
8. Lim, Wie Ming. and Harrison, John. (1997), "Parallel Approaches for Discovering Functional Dependencies from Data for Information System Design Recovery", Proceeding of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN 97'), Taipei, Taiwan, ROC, 18-20 Dec, 1997.
9. Li, S. H., Huang, S. M. and Chen, H. H. (1997), "Discovering Missing Semantics from Existing Relational Databases", Proceeding of 8th International HKCS Database Workshop, Hong Kong, 1997, p. 275-285
10. Ramanathan, Shekar. and Hodges, Julia. (1997), "Extraction of Object-Oriented Structures from Existing Relational Database", SIGMOD Record, Vol. 26, No. 1, March 1997, p. 59-64
11. Soutou, Christian. (1998) "Inference of Aggregate Relationships through Database Reverse

Engineering", Proceeding of the 17th International Conference on Conceptual Modeling, November 1998, Singapore, p. 121-149.

12. Cluet, Sophie., Delobel, Claude., Siemon, Jerome. and Smaga, Katarzyna. (1998), "Your mediators need Data Conversion!", Proceeding of the 1996 ACM SIGMOD international conference on Management of data, June 1-4, 1998, p. 177-188
13. Milo, Tova. and Zohar, Sagit. (1998), "Using Schema Matching to Simplify Heterogeneous Data Translation", Proceeding of the 24th VLDB Conference, New York, USA, 1998, p. 122-133.
14. Wong, L. (1995), "A Data Transformation System for Biological Data Sources", Proceeding of the 21st VLDB Conference, Zurich, Switzerland, 1995, p. 158-168.
15. Fong, Joseph (1997), "Converting Relational to Object-Oriented Databases", SIGMOD Record, Vol. 26, No. 1, March 1997, p. 53-58.
16. Pang, Ringo. and Tsang, Philip. (1999), "Heterogeneous and Internet Database", Proceeding of the 9th International Database Conference Industrial Volume, published by City University Press.
17. Pang, Ringo. (1999), "Data Conversion from Object-Oriented to Relational Database", Proceeding of the 9th International Database Conference, Academic Volume, published by City University Press, p. 335-348
18. Illback, James H. (1999), "Software Reuse: Data Conversion Experiences and Issues", Proceeding of the fifth symposium on Software reusability, Los Angeles, CA, USA, May 21-23, 1999, p. 10-16
19. Josifovski, Vanja., and Risch, Tore. (1999), "Integrating Heterogeneous Overlapping Database Through Object-Orient Transformations", Proceeding of the 25th VLDB Conference, Edinburgh, Scotland, 1999, p. 435-446
20. Vermeer Mark M. W. and Apers, Peter M. G. (1996), "The Role of Integrity Constraints in Database Interoperation", Proceeding of the 22nd VLDB Conference, Mumbai(Bombay), India, 1996, p. 425-435
21. Hull, Richard. and Zhou, Gang. (1996), "A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches", Proceeding of the 1996 ACM SIGMOD international conference on Management of data, June 4-6, 1996, p. 481-492
22. Honda, Kohei. (1999), "Semantics Study and Reality of Computing", ACM Computing Surveys, Vol. 31, Issue 3es, 1999
23. Papazoglou, M. P. (1995), "Unraveling the Semantics of Conceptual Schema", Communication of the ACM, September 1995, Vol. 38, No. 9
24. Toivonen, Hannu. (1996), "Sampling Large Databases for Association Rullles", Proceeding of the 22nd VLDB Conference, Mumbai(Bombay), India, 1996, p. 134-145
25. Chen, P. (1976), "The entity-relationship model - toward a unified view of data", ACM transaction on database systems, Vol. 1, No. 1, p. 9-36
26. Elmasri, R., Weeldreyer, J. and Hevner, A. (1985), "The category concept: An extension to the entity-relationship model", International Journal on Data and Knowledge Engineering, Vol. 1, No. 1, p. 75-116.
27. Rumbaugh, J. (1991), "Object-Oriented Modeling and Design", Prentice Hall, 1991.
28. Gustas, Remigijus. (1998), "Integrated Approach for Modeling of Semantic and Pragmatic Dependencies of Information Systems", Proceeding of the 17th International Conference on Conceptual Modeling, November 1998, Singapore, p. 121-134
29. Fraternali, Piero. and Tanca, Letizia. (1995), "A Structured Approach for the definition of the Semantics of Active Databases", ACM Transactions on Database Systems, Vol. 29, No. 4, December 1995, p. 414-471.
30. Fong, J. and Huang, S. M., "Architecture of a universal database: a frame model approach", International Journal of Cooperative Information Systems, Vol. 8, No. 1, 1999, p. 47-82
31 Fong, Joseph., Pang, Francis., Fong, Anthony. and Wong, Daniel. (1999), "Schema Integration for Object-Relational Databases with Data Verification", Proceedings of the International Computer Symposium Workshop on Software Engineering and Database Systems, p. 185-192.
32. Poulovassilis, A. and Mc. Brien, P., (1998) "A general formal framework for schema transformation", Data and Knowledge Engineering 28, pp. 47-71.
33. McBrien and Poulovassilis, A., (1999), "Automatic Migration and Wrapping of Database Applications – A Schema", Lecture Notes in Computer Science (LNCS) 1728 on Conceptual Modeling – ER'99 by Springer Verlag, pp.96-113.
34. Riccardo Torlone, Paolo Atzeni: A Unified Framework for Data Translation over the Web. WISE (1) 2001: 350-358.
35. Paolo Atzeni, Riccardo Torlone: MDM: a Multiple-Data-Model Tool for the Management of

Heterogeneous Database Schemes. SIGMOD Conference 1997: 528-531.
36. Paolo Atzeni, Riccardo Torlone: Management of Multiple Models in an Extensible Database Design Tool. EDBT 1996: 79-95.
37. Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, Alon Y. Halevy: Representing and Reasoning about Mappings between Domain Models. AAAI/IAAI 2002: 80-86.
38. Philip A. Bernstein: Generic Model Management: A Database Infrastructure for Schema Manipulation. CoopIS 2001: 1-6.
39. Suad Alagic, Philip A. Bernstein: A Model Theory for Generic Schema Management. DBPL 2001: 228-246.
40. Philip A. Bernstein, Thomas Bergstraesser: Meta-Data Support for Data Transformations Using Microsoft Repository. IEEE Data Engineering Bulletin 22(1): 9-14 (1999).
41. Batini, C., Ceri, S. and Navathe, S B. (1992), "Conceptual Database Design: An Entity Relationship Approach", Benjamin/Cummings Publishing Company, 1992.
42. Miller, R., J., Ioannidis, Y.E and Ramakrishnan R. (1993) The Use of Information Capacity in Schema Integration and Translation, Proceedings of the 19th VLDB Conference Dubin, Ireland 1993.