

ARCHITECTURE OF A UNIVERSAL DATABASE: A FRAME MODEL APPROACH

JOSEPH FONG

*Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong*

SHI-MING HUANG

*Department of Information Management, Tatung Institute of Technology
Chungshan N. Road, Taipei, Taiwan*

RDB has been dominant in the industry for the last decade. OODB is recognized as a post-relational technology that can improve productivity. Hierarchical Database and Network Database were popular in the '70s, and have been developed into legacy database systems. The DBMS of various data models have proliferated into many companies, and become their important assets. There is a need to integrate these database system into a data warehouse in the company. We investigate a solution to the problem by offering an architecture of a universal database for the connectivity of various DBMSs using different data models. A frame model is chosen to represent the conceptual and logical schema of the universal database, which structures an application domain into classes organized via generalization, aggregation and user-defined relationships, and its data in relational tables. The schemas of the existing database systems are translated into frame model conceptual schemas which are integrated into a global frame model in a knowledge representation that includes classes for object structure descriptions and constraints for supporting user-defined relationships. The universal database is implemented by a relational DBMS as a kernel. In addition to relational tables, the universal database consists of program area to emulate database navigation in nonrelational DBMS, method classes to implement program calls to emulate methods of OODBMS, and constraint classes to preserve semantic and resolve naming conflicts in schema translation and integration. To ensure each database program access the universal database, a database gateway for each source DBMS is developed to translate their DML into SQL, which is chosen to be its kernel database language, for its user-friendliness, standardization and popularity in the industry.

Keywords: universal database, frame model, knowledge-language, database gateway, schema translation, schema integration, data

conversion, data manipulation language emulation, data warehouse

1 Introduction

RDB has been a legacy system since '80s. It has limitations in multimedia systems, intensive engineering application and foreign keys data redundancy. OODB is comparatively abstractive, flexible and productive in database system development and enhancement. There is an increasing trend for organizations to adopt OODB as well as RDB to take the advantage of object-oriented technology. Due to the high risk of changeover, legacy systems of HDB and NDB are still in operation in many companies. To protect companies' investments in database systems, overnight conversion from one database systems to another is too costly and difficult to be adopted. Research surveys on database users show that coexistence of database systems is a likely option. Our objective is to let users run their production database systems without interruption, and use advanced data models for new applications in the same platform. As a result, the old and the new database systems can coexist to form a data warehouse.

This paper offers an architecture of a universal database to meet the requirements of the interoperability among various data models. The methodology translates existing data models into a frame model of the universal database. The frame model consists of the active and dynamic data structure of the legacy data models with the constraints structures to resolve their synonyms and homonyms conflicts. The frame model logical schema in class format is shown in Table 1¹ and stores the method (operations) of each class. The frame model logical schema in relation format is the same as the relational schema to implement the frame model logical schema in class format.

Table 1 The frame model logical schema in class format

Header Class{ Class_Name /* a unique name in all system */
--

Primary_Key /*an attribute name of unique value */ Parents /* a list of class names */ Operation /* program call for operations */ Class_Type /* type of class, e.g. active and static */
Attribute Class { Attribute_Name /* a unique name in this class */ Class_Name /* reference to header class */ Method_Name /* a unique name in this class for data operation */ Attribute_Type /* the data type for the attribute */ Default_Value /* predefined value for the attribute */ Cardinality /* single or multi-valued */ Description /* description of the attribute */}
Method class { Method_Name /* a unique name in this class */ Class_Name /*reference to header class */ Parameters /* a list of arguments for the method */ Method_Type /* the output data type */ Condition /* the rule conditions */ Action /* the rule actions */}
Constraint class { Constraint_Name /* a unique name for each constraint */ Class_Name /* reference to header class */ Method_Name /* constraint method name */ Parameters /* a list of arguments for the method */ Ownership /* the class name of the method owner */ Event /* triggered event */ Sequence /* method action time */ Timing /* the method action timer */ }

The frame model consists of two classes: static classes and active classes. Static classes represent factual data entities and active classes represent rule entities. An active class is event driven, obtaining data from the database when invoked by a certain event. The static class stores data in its own database. The two classes use the same structure. Combining these two types of objects within the inheritance hierarchy structure enables the frame model represent heterogeneous knowledge. The structure of a class includes four main parts: header, attributes, methods, and constraints as follows:

- Header classes - Every header class includes basic frame information to represent the class entity. The header of the class structure includes Class_Name, Parents, Operation, and Class_Type. The Class_Name is a class identifier, a unique name defined by the application developer. Parents represent the generalization/specialization relationship between the current class and its super class. Each class can contain a call to an interactive SQL command file to emulate a method in OODB. Class_Type describes the type of class. Static class is for static data instance and active class is for invoking event-driven rules. The Primary_Key assists the system to define the semantics of an object identifier.
- Attribute classes - These represent the properties of a class. A particular object has a value for each of its attributes. The attribute values of each object become a major part of the data stored in the database. An attribute composed of several more basic attributes is a composite attribute. Attributes that are not divisible are atomic attributes. An attribute value can be derived from the related attributes or objects. For example, for a particular person object, the value of Age can be determined from the current date and the value of the person's date of birth. This type of attribute is a virtual attribute in the frame model, and is the result of a deductive rule or an active rule.
- Method classes - Rules extend the semantics of the data. The specification of rules is an important

part of semantic data modeling, since many of the facts in the real world are derived rather than consisting of pure data². It is increasingly important to integrate rules into data models in new information systems. A crucial characteristic of the frame model is that the paradigm provides mechanisms for structuring and sharing not only data, but also the methods that act on the data. The frame model uses this characteristic to integrate rules and operations into its model. The methods of the frame model represent the behavior, the active rules, and the deductive rules of a particular object. The behavior representation of the frame model is reflected by the different needs of user communities. The method body takes a production rule structure in the frame model.

- Constraints - There are many properties of data that cannot be captured in the form of structures. These properties essentially serve as additional restrictions on the values of the data and how the data may be related. For example, there may be a restriction that if a person is head of a department, the person must belong to the department. Such restrictions cannot be expressed in terms of structures, but must be captured by some additional mechanism. It is a primary consideration of database technology to ensure data correctness and consistency. This requires the system to support integrity constraint functions. These functions are required to allow proper handling of updates of knowledge for interrelated actions and active database rules. There are many semantics present in constraints that can be very useful. Constraints can be used to prevent a possibly expensive database search operation³. The constraint technology used in current database systems requires different levels of integrity constraint.

To resolve the synonyms and homonyms conflicts among data models, their locations are defined in synonyms and homonyms tables, and active classes are provided to define their constraints. To ensure the interoperability among data models, an efficient way is to translate them into a frame model. The translated frame model can be integrated into a global frame model by identifying their related semantics.

The frame model conceptual schema can be represented in a diagram in Figure 1. The emphasis is on representation at a schema level rather than at an instance level. This is useful because a knowledge schema rarely changes; whereas the extension may change frequently. The schema is usually easier to display than the extension of a knowledge base, because it is compact. The frame model conceptual schema integrates the conceptual requirements of individual users into a single “community” view. The frame model logical schema describes the version of the frame model in conceptual schema that can be presented to the DBMS⁴.

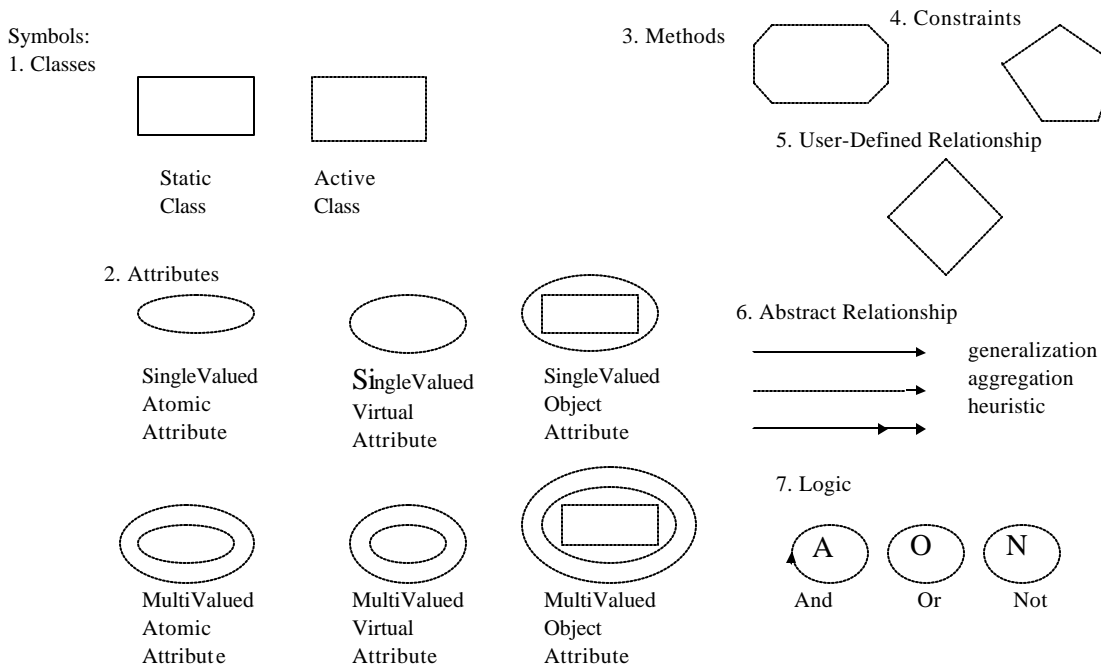


Figure 1 Frame model conceptual schema diagram notations

Figure 2 displays the family knowledge base schema in a frame model conceptual schema diagram. The knowledge base shows that a person is described by Name, Sex, Date_of_Birth, Age, Generation, Address, Father, and Mother. A male is a person and his sex is always male. A person's son is male. He is a child of a person and that person is the parent of the child. This is an inverse relationship. Age and generation are derived by the date of birth of the person. The father and mother of a person are themselves persons. This is a recursive situation. An address is an associated attribute which is described by Street, City, and Postcode. Sex, and Date_of_Birth shown in ovals. Virtual attributes such as Generation and Age are shown in dotted ovals. Object attributes such as Father, Mother, and Address are shown in ovals with an internal rectangular box. Each attribute is attached to its class type or relationship type by a straight line. Methods such as Male are shown in Octagons and each method is attached to its class type or relationship type by a straight line.

Generalization (isa) relationships such as relationship between Person and Male are shown by an arrow line. Disjoint generalization can be shown by Disjoint constraint among Mother class and Father class. A person can either be a natural father or mother, but cannot be both. Aggregation such as relationship between the Address attribute of the Person class and the Address class are shown by a dotted arrow line. Heuristic such as the relationship between Date_of_Birth and Age, or that between Age and Generation, are shown as a double arrow line. Constraints such as Always are shown as a Pentagon and each constraint is attached to its method by a heuristic in a double arrow line. For example, the Always constraint is implemented in the Sex attribute via the Male method. Logical operators such as AND, OR, and NOT are shown in a circle with the first letter of the operators. They are implied by heuristic. For example, the Son class is deduced from the Male and Child entity via an AND operation. Similarly, the Parents class is deduced from the Father and Mother entity via an OR operation.

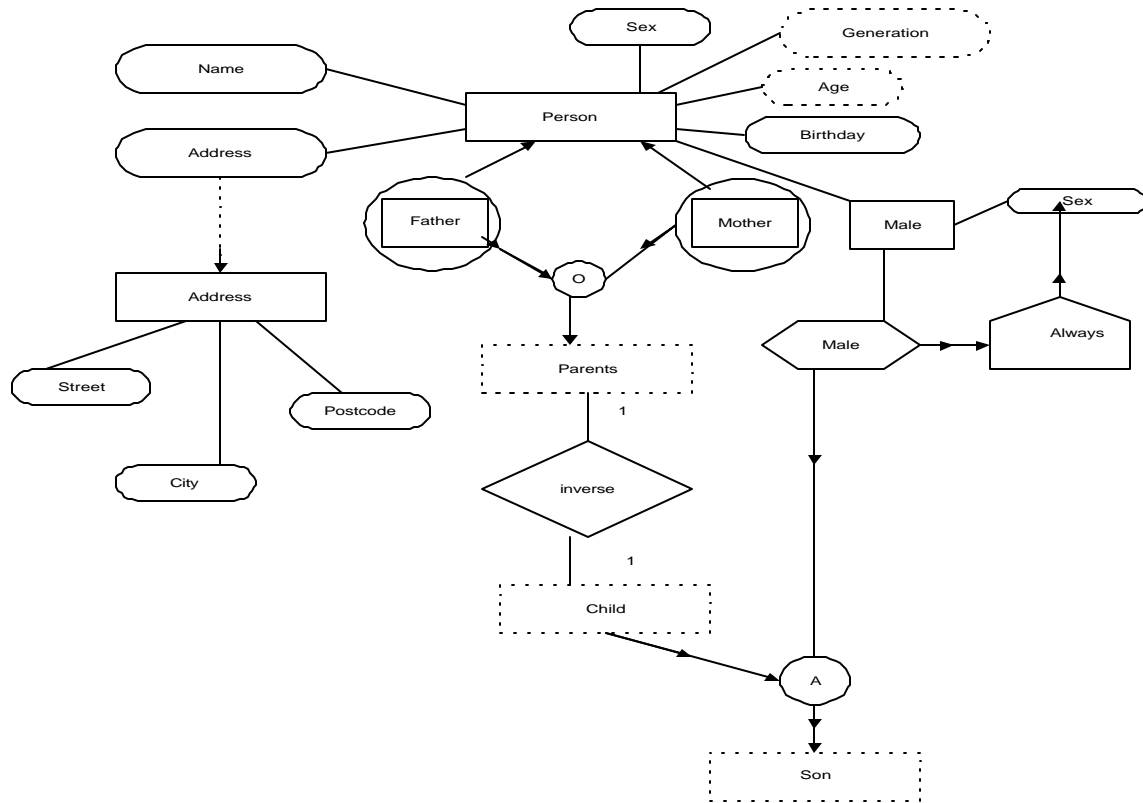


Figure 2 Frame model conceptual schema for the Family Knowledge Base.

- The universal database system consists of following components as shown in Figure 3.
- Schema: the global frame model conceptual schema, the frame model conceptual schema, the frame model logical schema and the frame model internal schema.
 - Data: relational table
 - DBMS: RDBMS
 - DML: SQL

2. Related work

There are many research on universal database and database connectivity. Date⁵ provided an universal data structure for hierarchical, network and relational data models. Hsiao⁶ implemented an universal database that converted other databases' schemas, data and programs into his universal model. UniSQL/M⁷ database gateway applies a client/server architecture approach that allows programs of client access server database of different DBMSs. Microsoft's ODBC software tool allows connectivity among relational DBMSs in the same platform. Hsiao converted existing database programs to his universal database programs before operation. Tardieu & Franckson⁸ proposed an exogeneous approach to abstract ER model into an object management on a hierarchy of meta data where each level being a generalization of the lower one. Tawbi et al⁹ suggested an Event-Condition-Action rules, specified as Ada tasks, giving the system the capability to react automatically to critical situations which can happen in the database. However, UniSQL/M needs 2n database gateways for existing database programs to access other databases where n is the number of different DBMSs. Date and ODBC can integrate nonrelational and relational data models except OODB. Hsiao's method does not include OODB, and is proprietary. Tardieu & Franckson used the meta data approach which is complicated with many levels for implementation. Tawbi et al's idea is based to a computer language Ada, not a general solution.

Their limitations are severe and unacceptable for companies who want to reengineer their existing database programs without interrupting their production systems. The universal database in this paper can help users develop replicated databases for data warehousing without impact to their production. The universal database is an improvement to them because it allows existing database programs access n databases via n database gateways only. The requirement of the universal database is the data quality of the source database which can be ensured by data cleaning in pre-processing¹⁰. Data mining can also be applied to derive hidden relationships between data elements and data integration using statistical techniques¹¹.

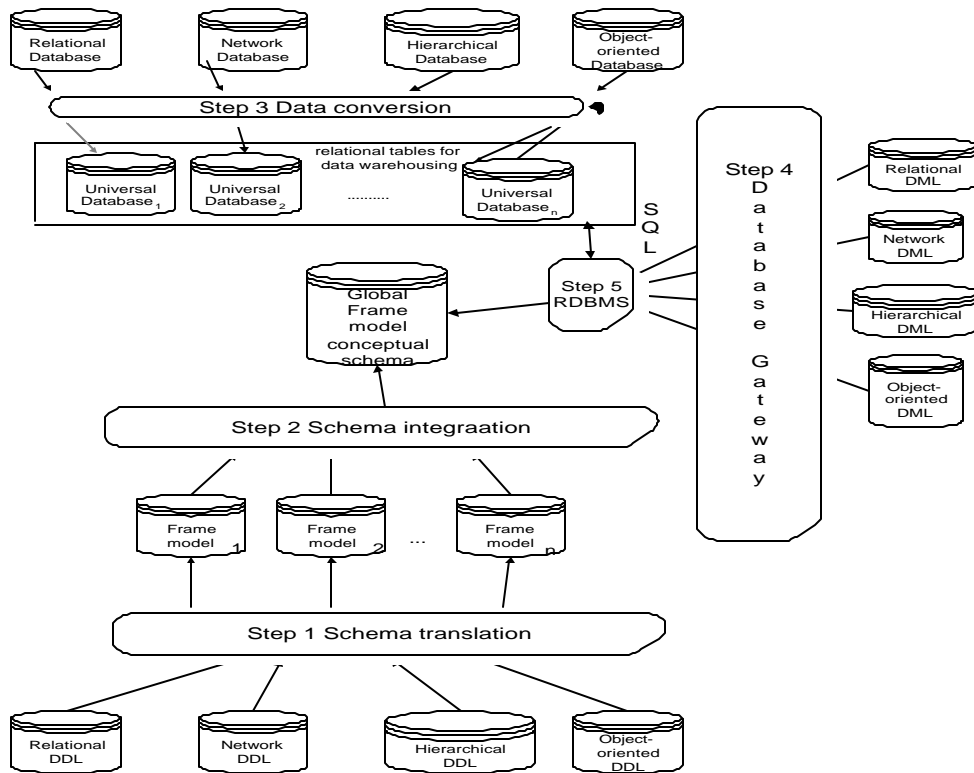


Figure 3 Architecture of the universal database in frame model

3. Schema translation from various data models to frame model

Schema translation is the first step to implement the universal database. After schemas of all existing database systems are translated to the frame model, a global view of the universal database can be provided to the users. For the new development, their conceptual schema must be the global frame model. The followings are the representation of different semantics by various data models and the frame model. There is an one-to-one mapping from HDB/NDB/RDB/OODB to the frame model schema.

Sub-step 1 Preserve entity semantic

A uniquely identifiable significant object in an application is an entity. In RDB, it is a relation. In NDB, it is a record. In HDB, it is a segment. In OODB, it is a class. They can be mapped to a class in frame model logical schema. Since frame model database engine is a RDBMS, the frame model internal schema is the same as RDB. However, if the existing OODB schema is mapped to frame model, the Object Identity is mapped as key in the frame

model internal schema.

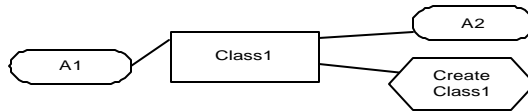
HDB schema
H1 (A1, A2)

NDB schema
Record N1 (A1, A2)

RDB schema
Relation R1 (A1, A2)
/*underlined=primary keys*/

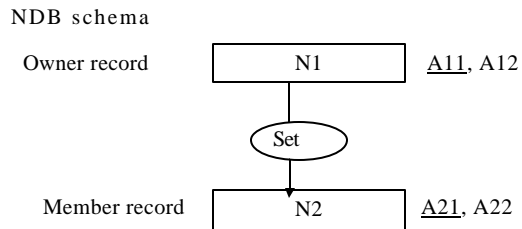
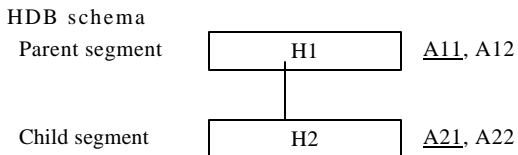
OODB schema
class C1
Attributes A1, A2
Method Create C1
end C1

Frame model conceptual schema



Sub-step 2 Preserve relationship semantic

Entities related to each other via relationship. In RDB, the foreign key must be on the “n” side if the cardinality is 1:n. The foreign key can be on either side if the cardinality is 1:1. A relationship relation must be created if the cardinality is m:n. In NDB, relationship between entities is represented by Set. For binary relationship, it is mapped to a Set between an owner and a member record in 1:n cardinality. It is mapped to two Sets between two owners and one member records if the cardinality is m:n between the two owners. For n-ary relationship, there are n Sets between n owners and one member record. In HDB, relationship is represented by pointer between parent and child segments in 1:n cardinality. For m:n relationship, a relationship segment must be created to represent it. For n-ary relationship, it is mapped to a parent segment and n child segments. In OODB, association between classes is represented by association attribute and its inverse association attribute. The association attribute is an object in a class which points to an object in another class. The data type of the association attribute must be multiple valued if the cardinality is 1:n, and a single value if the cardinality is 1:1. For m:n relationship, an association class can be created to link the two associated classes via two single-valued attributes. In frame model logical schema, for a m:n cardinality, a static class can be created to store the associations between the two associated classes.

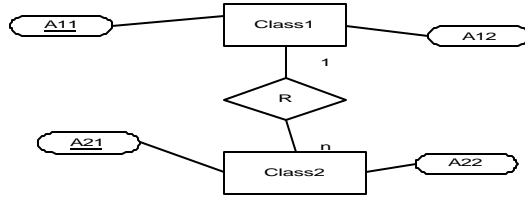


RDB schema
Relation R1 (A11, A12)
Relation R2 (A21, A22, *A11)
/* * * prefixed are foreign keys */

OODB schema

Class C1	Class C2
Attribute	Attribute
A11: integer	A21: integer
A12: string	A22: string
R12: set of C2	R21: C1
end C1	end C2

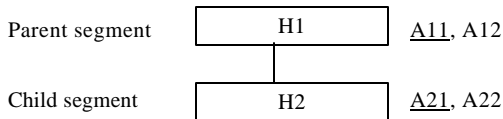
Frame model conceptual schema



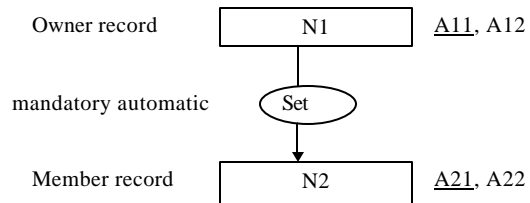
Sub-step 3 Preserve existence dependency

Existence dependency can be represented by weak entity. The existence of the weak entity depends on its strong entity. In RDB, this semantic can be represented in composite key. Whenever a weak relation is created, it must have a composite key pointing to a strong relation. Whenever a strong relation is deleted, its weak relation must be deleted. In NDB, this can be represented in a Set with mandatory automatic membership. Whenever a member record is created, it must be assigned to an owner record automatically. In HDB, such semantic can be represented by its pointer structure between its parent and child segments. In OODB, the existence dependency can be implemented through methods. The method of a subclass checks the existence of its dependable class before creating its own object. The method of the dependable class deletes its dependent object before deleting its own object. In frame model, constraint can be used to implement the existence dependency semantic.

HDB schema



NDB schema



RDB schema

Relation R1 (A11, A12)
 Relation R2 (*A11, A21, A22)

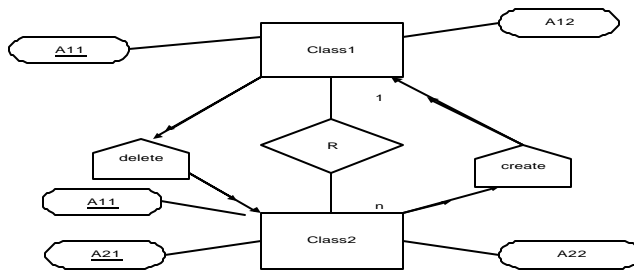
OODB schema

```

Class C1
own C2
Attributes
A11: integer
A12: string
End C1

Class C2
Attributes
A21: string
A22: string
end C2
  
```

Frame model conceptual schema

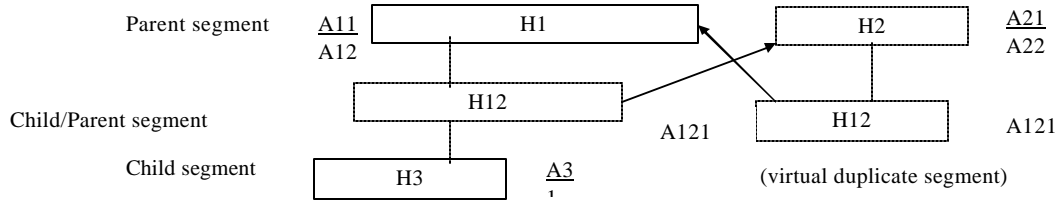


Sub-step 4 Preserve aggregation semantic

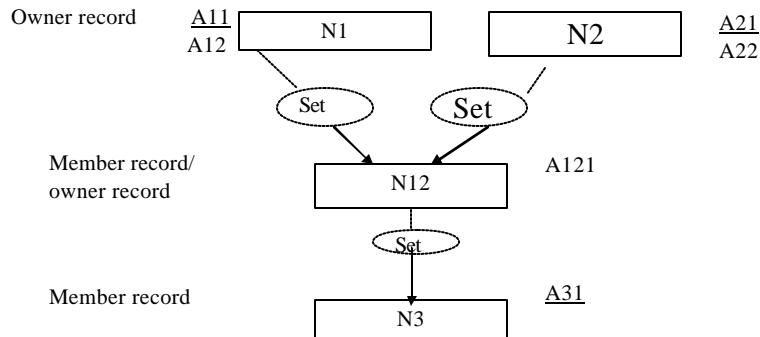
Aggregation semantic can be represented by an aggregation entity. A binary relationship between two entities is treated as an entity when the binary relationship relates to another entity. In RDB, this semantic is represented in compound composite key. Whenever a relationship relation is created, it must have a composite key pointing to the primary keys of two relations. The composite key appears in the third relation as a foreign key. In NDB, this is represented in a relationship record between two owner records. An aggregation

occurs when the member record acts as an owner record by connecting it to another member record. In HDB, such semantic is represented by a logical child segment that links to a logical parent segment. In OODB, the aggregation semantic is implemented by an associated class which links two association classes through association attributes. In frame model, such semantic is represented by aggregation in conceptual schema and implemented by frame model logical schema.

HDB schema



NDB schema



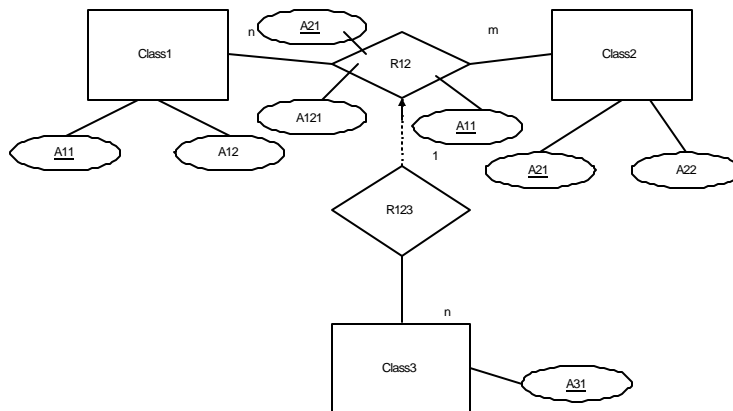
RDB schema

- Relation R1 ($\underline{A11}$, A12)
- Relation R2 ($\underline{A21}$, A22)
- Relation R12 ($\ast\underline{A11}$, $\ast\underline{A21}$, A121)
- Relation R3 ($\ast\underline{A11}$, $\ast\underline{A21}$, $\underline{A31}$)

Object-oriented schema

Class C1	Class C2	Class C12	Class C3
Attributes	Attributes	Attributes	Attributes
A11: integer	A21: integer	A121: string	A31: string
A12: string	A22: string	R123: set of C3	R312: C12
R12: set of C12	R21: set of C12	R1: C1	end C3
end C1	end C2	R2: C2	
		end C12	

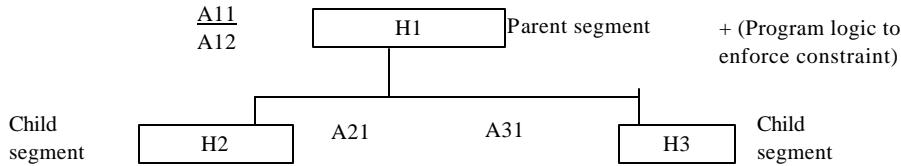
Frame model conceptual schema



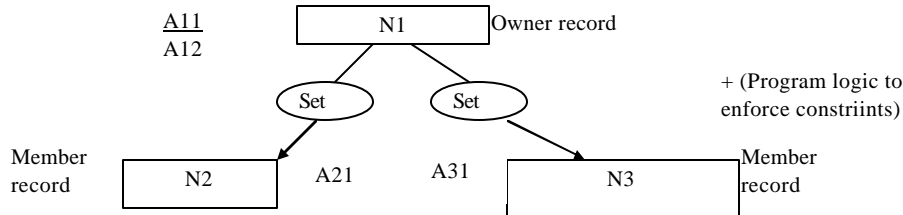
Sub-step 5 Preserve generalization semantic

Generalization semantic is represented by subclass entity and superclass entity. The domain value of the former is a subset of the domain value of the latter. For disjoint generalization, the data occurrences of subclass entities occurs in one subclass entity only. For overlap generalization, there is no such restriction. In RDB, this semantic can be represented by relations having the same primary keys. The one with a primary key that acts as a foreign key is a subclass relation. An optional predicate field in the superclass relation differentiates its subclass data occurrence. In NDB, this is represented in a hierarchy structure with an owner record as a superclass entity, and its member records as subclass entities. The constraints of superclass and subclass with disjoint generalization must be implemented by programs. In HDB, such semantic is represented by a hierarchy structure in the same way as in the NDB. In OODB, the existence dependency is implemented through inheritance. The subclass inherits the attributes and the methods of its superclass. The disjoint generalization is implemented by the methods of the subclasses. When creating a disjoint object in the frame model, constraints must be enforced to ensure the same object not occurring in another subclass. In frame model, logic and generalization are used to enforce semantics constraint.

HDB schema



NDB schema



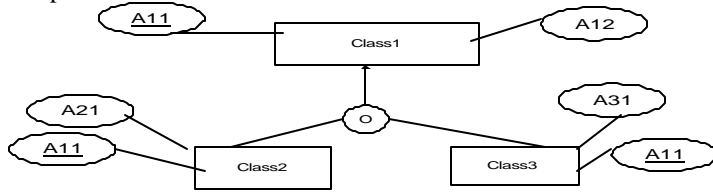
RDB schema

Relation R1 (A11, A12, P) (note: P = predicate field, and is optional)
 Relation R2 (*A11, A21)
 Relation R3 (*A11, A31)

OODB schema

Class C1	Class C2	Class C3
Attributes	inherits C1	inherits C1
A11: integer	Attributes	Attributes
A12: string	A21: string	A31: string
End C1	Method	Method
	when create C2	When create C3
	check not-existence(C3)	check not-existence(C2)
	end C2	end C3

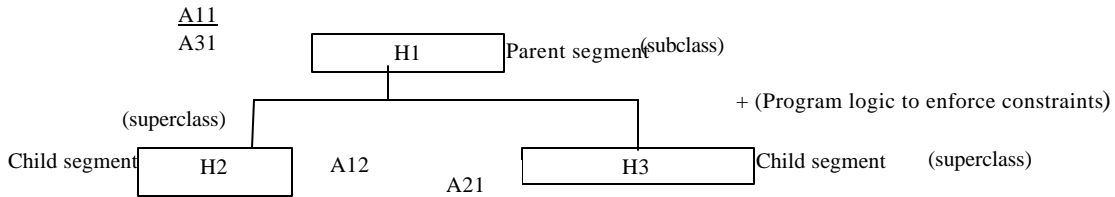
Frame model conceptual schema



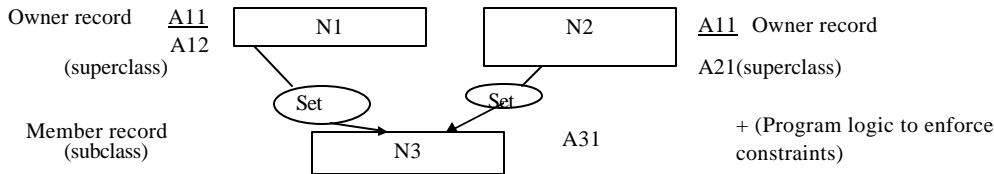
Sub-step 6 Preserve categorization semantic

Categorization semantic is represented by subclass entities and superclass entities. The domain value of the former is a subset of the union domain value of the latter. The data occurrences of subclass entities is subset to its superclasses. In RDB, this semantic is represented in relations having the same primary keys. The one with a primary key that acts as a foreign key is a subclass relation. In NDB, this can be represented by having one member record as a subclass entity, and its owner records as its superclass entities. The constraints of the inheritance of the member record to one of its owner record is enforced by programs. In HDB, the categorization semantic is represented by its hierarchy structure with the parent segment as the subclass entity and child segment as its superclass entities. In OODB, the categorization semantics is implemented through single inheritance. An object of subclass inherits one of its superclass. Such constraint must be implemented by the method of the subclass. In frame model, the symbol of logic and generalization describe the categorization semantics.

HDB schema



NDB schema



RDB schema

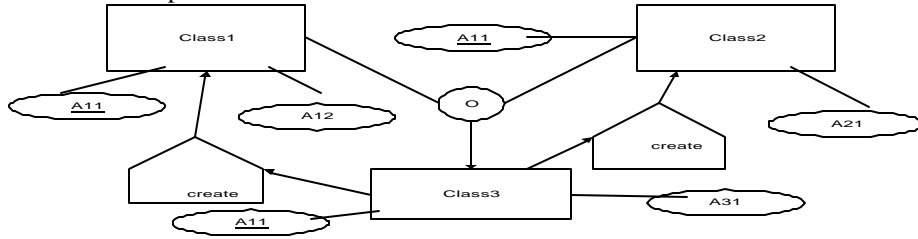
- Relation R1 (A11, A12)
- Relation R2 (A11, A21)
- Relation R3 (*A11, A31)

OODB schema

<p>Class C1</p> <p>Attributes</p> <p>A11: integer</p> <p>A12: string</p> <p>end C1</p>	<p>Class C2</p> <p>Attributes</p> <p>A21: integer</p> <p>A22: string</p> <p>end C2</p>	<p>Class C3</p> <p>Attributes</p> <p>A31: string</p> <p>Method</p> <p>Case C3 is</p> <p>C1: inherits C1</p> <p>C2: inherits C2</p>
--	--	--

end C3

Frame model conceptual schema

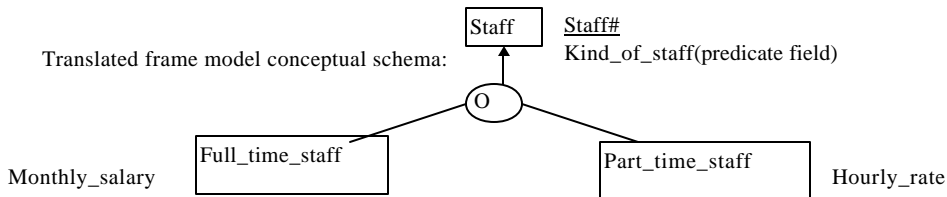


Case study of translate relational schema to frame model

The following is a relational schema that describes a disjoint generalization between full-time staff and part-time staff.

- Relation Staff (Staff#, Kind-of-staff)
- Relation Full-time-staff (Staff#, monthly-salary)
- Relation Part-time-staff (Staff#, hourly-pay)

We can translate the above relational schema into frame model as follows:



Translated frame model logical schema:

Staff_Header_Class relation

Class_Name	Primary_key	Parents	Operation	Class_Type
Staff	Staff#	0	0	Static
Full_time_staff	Staff#	Staff	Call Fulltime_staff	Dynamic
Part_time_staff	Staff#	Staff	Call Parttime_staff	Dynamic

Attribute class

Attribute name	Class name	Method name	Attribute type	Default value	Cardinality	Description
Staff#	Staff		integer		1	Superclass primary key
Kind_of_staff	Staff		string		1	Superclass non-key attribute
Monthly_salary	Full_time_staff		integer		1	Subclass non-key attribute
Hourly_rate	Part_time_staff		integer		1	Subclass non-key attribute

Staff_constraint_class relation

Constraint_name	Method_name	Class_Name	Parameters	Ownership	Event	Sequence	Timing
Fulltime_staff	Full_time_staff	Staff	Staff#	self	Create	before	repeat
Parttime_staff	Part_time_staff	Staff	Staff#	self	Create	before	repeat

Staff_method_class relation

Method_name	Class_name	Parameter	Method_type	Condition	Action
Full_time_staff	Staff	@Staff#	boolean	If (Select * from Part_time_staff where staff# = @staff#) = null	Insert Full_time_staff values (@staff#)
Part_time_staff	Staff	@Staff#	boolean	If (Select * from Full_time_staff where staff# = @staff#) = null	Insert Part_time_staff values(@staff#)

4 Schema integration of frame models into a global frame model conceptual schema

Schema integration provides a global view of multiple schemas. It involves using a bottom up approach to integrate existing database into a global database by pairs as follows:

```

Begin For each existing database do
  Begin If its conceptual schema does not exist
    then reconstruct its frame model conceptual schema by reverse engineering;
  For each pair of existing databases' frame models schema A and schema B do
    begin resolve semantic conflicts between frame model schema A and schema B;/step1/
      Merge classes between frame model schema A and schema B; /*step2*/
      Merge relationships between frame model schema A and schema B; /*step3*/
    end
  end
end
end
    
```

The following three major steps must be followed in its sequence.

Step 1. Resolve conflicts among frame model conceptual schemas.

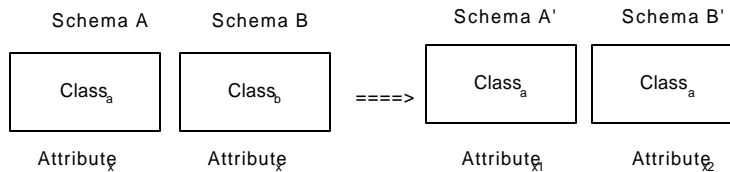
Sub-step 1.1 Resolve conflicts on synonyms and homonyms

Sub-step 1.1 Resolve conflicts on synonyms and homonyms

```

IF A.x and B.x have different data types or sizes
THEN x in A and B may be homonyms, let users clarify x in A and B
ELSE IF x ≠ y, and A.x and B.y have the same data type and size
THEN ((x,y) may be synonyms, let users clarify (x, y));
    
```

e.g.



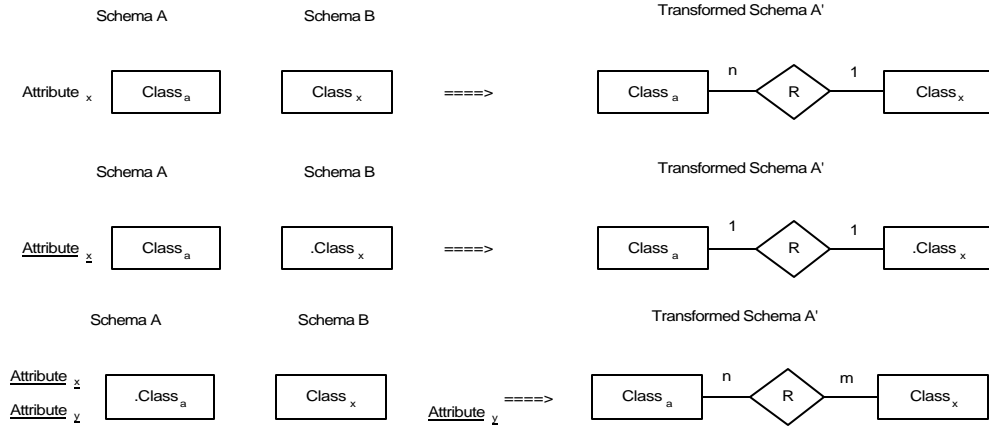
(note: Class_a and Class_b are synonyms, Attribute_x are homonyms)

Sub-step 1.2 Resolve conflicts on data types

```

IF x ∈ (attribute(A) ∩ class(B))
THEN class(A') ← class(B) such that cardinality (A, A') ← n:1
ELSE IF x ∈ (keys(A) ∩ class(B))
THEN class(A') ← class(B) such that cardinality (A, A') ← 1:1
ELSE IF (x ⊂ keys(A)) ∩ (class(B))
THEN class(A') ← class(B) such that cardinality(A, A') ← m:n
    
```

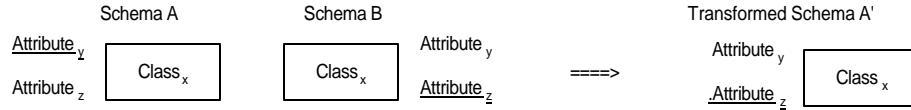
e.g



Sub-step 1.3 Resolve conflicts on key

IF $x \in (\text{key}(A) \cap \text{candidate_keys}(B))$
 THEN let users clarify x in A and B

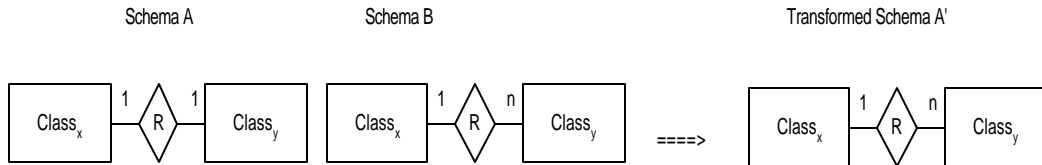
e.g



Sub-step 1.4 Resolve conflicts on cardinality

IF $(\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A_2) = \text{class}(B_2)) \wedge (\text{cardinality}(A_1, A_2) = 1:1) \wedge$
 $(\text{cardinality}(B_1, B_2) = 1:n)$
 THEN $\text{cardinality}(A_1, A_2) \leftarrow 1:n;$
 ELSE IF $(\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A_2) = \text{class}(B_2)) \wedge (\text{cardinality}(A_1, A_2) = 1:1$
 or $1:n) \wedge (\text{cardinality}(B_1, B_2) = m:n)$
 THEN $\text{cardinality}(A_1, A_2) \leftarrow m:n;$

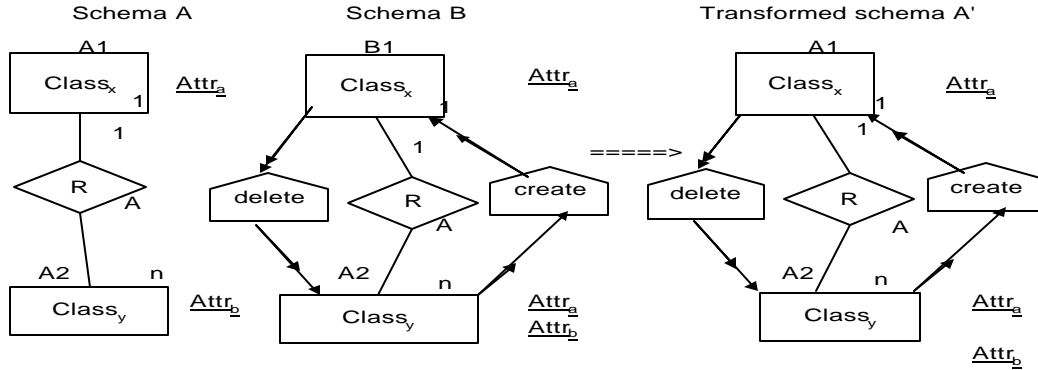
e.g.



Sub-step 1.5 Resolve conflicts on weak class

IF $(\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A) = \text{class}(B)) \wedge ((\text{key}(A_2) = \text{key}(B_2))=0)$
 $\wedge ((\text{key}(B_1)) \cap \text{key}(B_2)) \neq 0)$
 THEN $\text{Key}(A_2) \leftarrow (\text{Key}(A_1) + \text{Key}(A_2))$

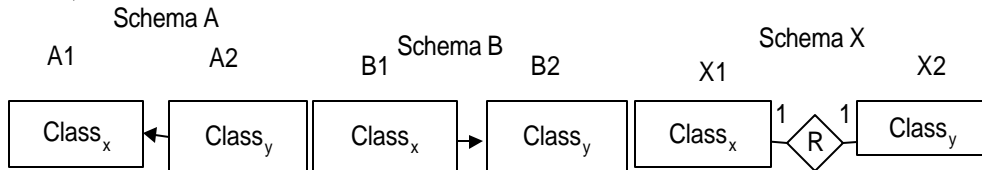
e.g.



Sub-step 1.6 Resolve conflicts on subtype class

IF $((\text{class}(A_2) \subseteq \text{class}(A_1)) \wedge (\text{class}(B_1) \subseteq \text{class}(B_2)) \wedge (\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A_2) = \text{class}(B_2)))$
 THEN begin class $X_1 \leftarrow \text{class } A_1$
 class $X_2 \leftarrow \text{class } A_2$
 cardinality(X_1, X_2) $\leftarrow 1:1$
 end;

e.g..

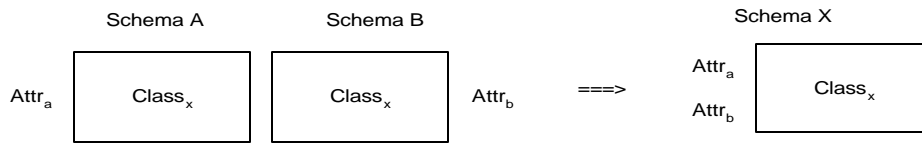


Step 2 Merge classes

Sub-step 2.1 Merge classes by Union

IF $(\text{domain}(A) \cap \text{domain}(B) \neq \emptyset)$
 THEN $\text{domain}(X) \leftarrow (\text{domain}(A) \cup \text{domain}(B))$

e.g..

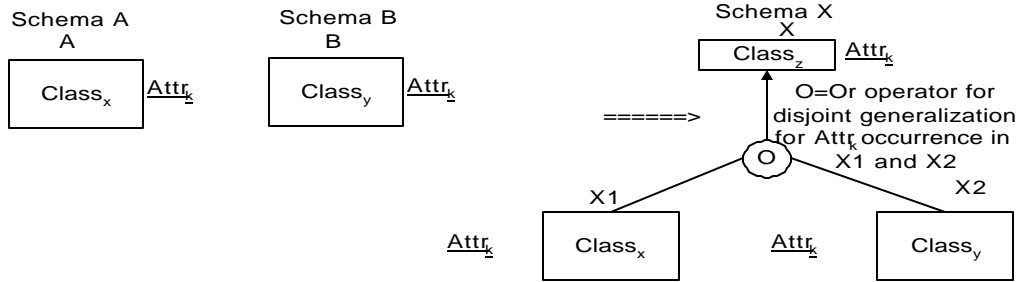


Sub-step 2.2 Merge semantic frame models by Generalization

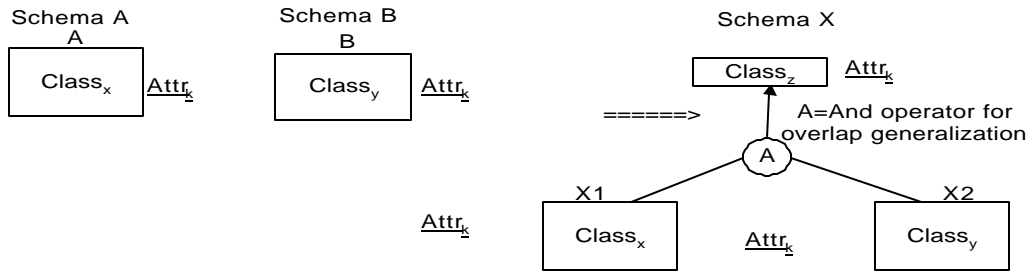
IF $(\text{domain}(A) \cap \text{domain}(B) \neq \emptyset) \wedge ((I(A) \cap I(B)) = \emptyset)$
 THEN begin $\text{Class}(X_1) \leftarrow \text{Class}(A)$
 $\text{Class}(X_2) \leftarrow \text{Class}(B)$
 $\text{domain}(X) \leftarrow \text{domain}(A) \cap \text{domain}(B)$
 $(I(X_1) \cap I(X_2)) = \emptyset$
 end
 ELSE IF $(\text{domain}(A) \cap \text{domain}(B) \neq \emptyset) \wedge ((I(A) \cap I(B)) \neq \emptyset)$
 THEN begin $\text{Class}(X_1) \leftarrow \text{Class}(A)$
 $\text{Class}(X_2) \leftarrow \text{Class}(B)$
 $\text{domain}(X) \leftarrow \text{domain}(A) \cap \text{domain}(B)$
 $(I(X_1) \cap I(X_2)) \neq \emptyset$
 end;

e.g..

Case 1



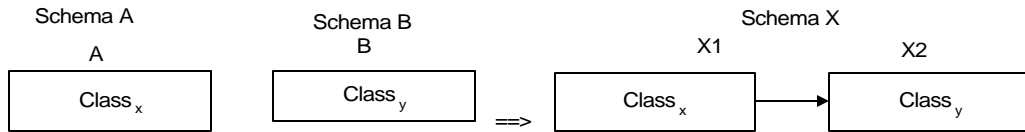
Case 2



Sub-step 2.3 Merge semantic frame models by Subtype Relationship

IF domain(A) ⊂ domain(B)
 THEN begin Class(X₁) ← Class(A)
 Class(X₂) ← Class(B)
 Class(X₁) isa Class(X₂)
 End;

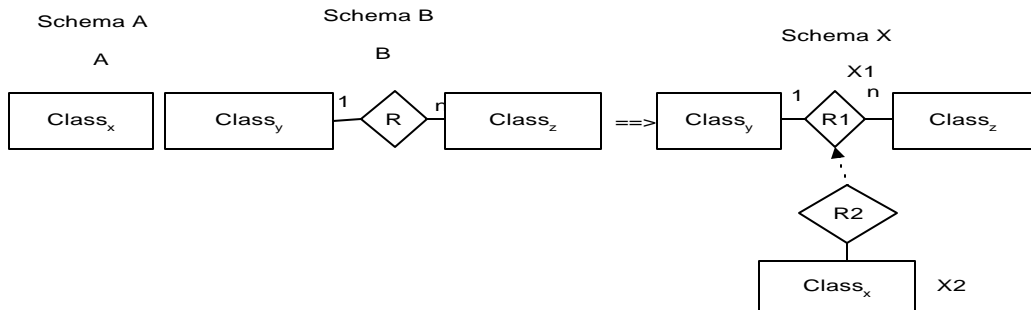
e.g.



Sub-step 2.4 Merge class by Aggregation

IF relationship(B) →→ class(A) /*MVD →→ means multi-value dependency/
 then begin aggregation(X₁) ← (class B₁, relationship B, class B₂)
 class(X₂) ← class(A)
 cardinality (X₁, X₂) ← 1:n
 end;

e.g..



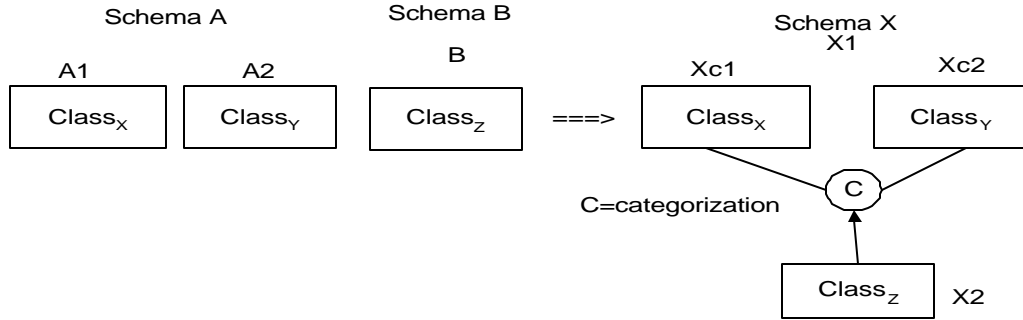
Sub-step 2.5 Merge classes by categorization

IF (I(B) ⊂ I(A₁)) ∨ (I(B) ⊂ I(A₂))
 THEN begin class(X₂) ← class(B)
 Class(X_{c1}) ← class(A₁)
 Class(X_{c2}) ← class(A₂)

Categorization (X_1) \leftarrow (class X_{c1} , class X_{c2})
 ($I(X_2) \text{ isa } I(X_{c1})$) \vee ($I(X_2) \text{ isa } I(X_{c2})$) /* X_2 is subtype to X_{c1} or X_{c2} */

end;

e.g.

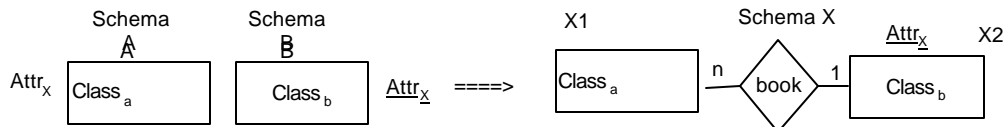


Sub-step 2.6 Merge class by Implied Binary Relationship

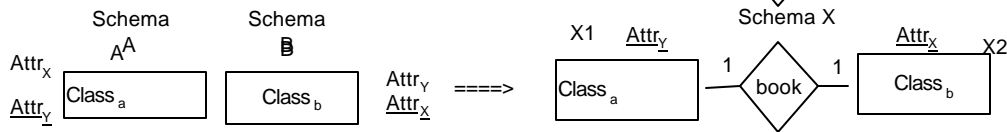
IF $x \in (\text{attribute}(A) \cap \text{key}(B))$
 THEN begin class(X_1) \leftarrow class(A)
 class(X_2) \leftarrow class(B)
 Cardinality (X_1, X_2) \leftarrow n:1
 end
 ELSE IF $((\text{attribute}(A) \cap \text{key}(B)) \neq 0) \wedge ((\text{attribute}(B) \cap \text{key}(A)) \neq 0)$
 THEN begin class(X_1) \leftarrow class(A)
 class(X_2) \leftarrow class(B)
 Cardinality (X_1, X_2) \leftarrow 1:1
 end;

e.g.

Case 1



Case 2



Step 3 Merge relationships

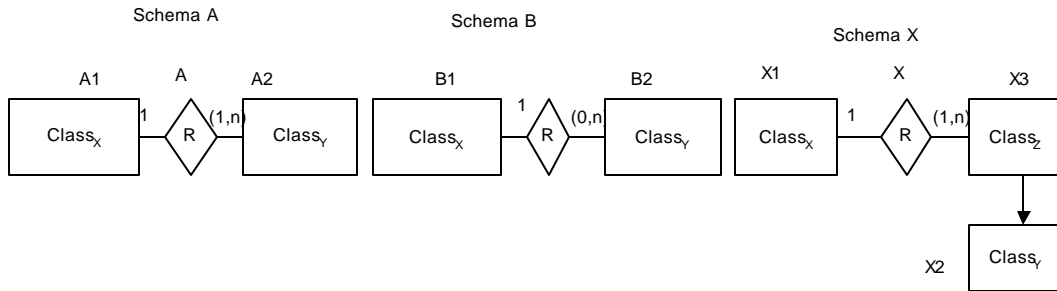
Sub-step 3.1 Merge relationships by subtype relationship

IF $(\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A_2) = \text{class}(B_2)) \wedge (\text{participation}(A_1, A) = \text{total})$ \wedge
 $(\text{participation}(B_1, B) = \text{partial})$
 THEN begin class(X_1) \leftarrow class(A_1)
 Class(X_2) \leftarrow class(A_2)
 Class(X_3) isa class(X_1)
 relationship X \leftarrow class(X_3, X_2)
 participation(X_3, X) \leftarrow total
 end
 ELSE IF $(\text{class}(A_1) = \text{class}(B_1)) \wedge (\text{class}(A_2) = \text{class}(B_2)) \wedge ((\text{relation}(A) \cap \text{relation}(B)) \neq 0)$
 THEN begin class(X_1) \leftarrow class(A_1)
 Class(X_2) \leftarrow class(A_2)
 class(X_3) isa class(X_2)
 class(X_4) isa class(X_2)

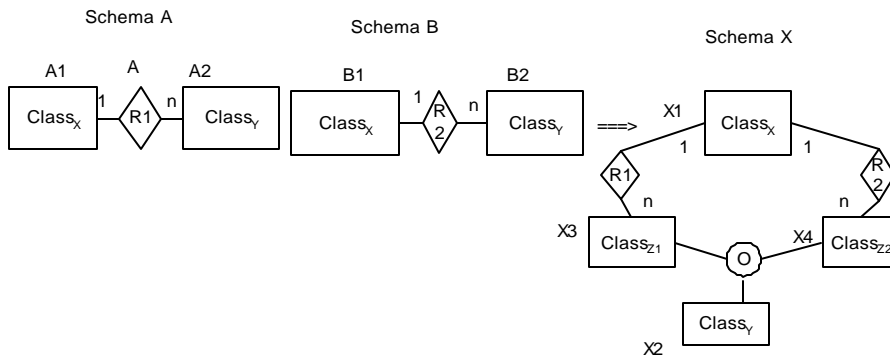
relationship (Xa) ← Relationship (A)
 relationship (Xb) ← Relationship (B)

end;

e.g.
 Case
 1



eg.
 Case
 2



Sub-step 3.2 Absorbing Lower degree Relationship into a Higher degree Relationship

IF ((relationship(A) ⊃ relationship (B) ∧ (degree(A) > degree(B)) ∧ (class(A1)=class(B1)) ∧ (class (A2)=class (B2)))

THEN begin relationship(X) ← relationship(A)

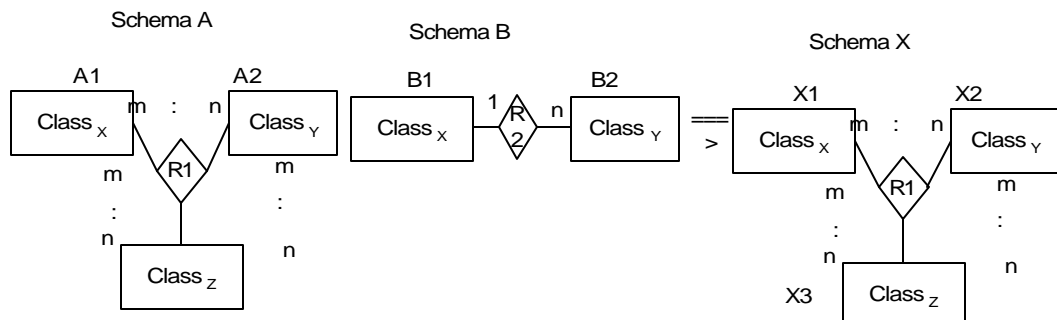
class (X1) ← class (A1)

class (X2) ← class (A2)

class (X3) ← class (A3)

end;

e.g.



Case study of resolving naming conflict between two frame models

To solve the semantic conflict problem between different attributes, active classes are introduced to the frame model. For instance, the value “vacancy” in the Employment

attribute of the Employee class in a frame model indicates that the employee is available for assignment to a new project. In another frame model, the same information is represented by another attribute Availability with values “yes” or “no”. To resolve this conflict problem, a method class can be created as an active class to derive the Availability attribute from the vacancy attribute. The active class can resolve the conflict by the following method in figure 4.

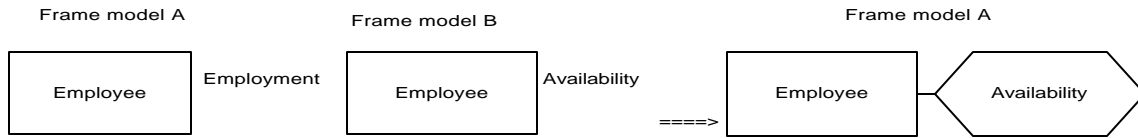


Figure 4 Resolve synonym conflicts by an active class in frame model

The corresponding frame model logical schema in class format is as follows:

```

Class Name:Employee
Attributes:
    Availability: Method(availability)
Methods:
    availability(): text;
    {IF Employment = "vacancy"
    THEN Availability = "yes"
    ELSE Availability = "no" }
end_of_class

```

Its frame model logical schema for constraints implementation is as follows:

Employee_Header_Class

Class_Name	Primary_key	Parents	Operation	Class_Type
Employee	Employee#	0	Call Availability Call Non-availability	Active

Employee method class

Method_name	Class_name	Parameter	Method_type	Condition	Action
Availability	Employee	employment, availability	boolean	If employment = 'vacancy'	availability = 'yes'
Non-Availability	Employee	employment, availability	boolean	If employment ≠ 'vacancy'	availability = 'no'

5 Data Conversion from legacy databases to universal database

To ensure users of different database systems access each other’s databases, they are migrated into a universal database for information retrieval. The global frame model is the conceptual model of the universal database. There are two kinds of data in the files:

- Prime data: the frame model logical schema in relation format for data storage.
- Dynamic rules: the frame model logical schema in relation format for constraints implementation.

Logical data approach in data conversion is applied here. The data of legacy databases are dumped into files. Related data in a record, a segment, or a class are unloaded into a sequential file. The relational tables of RDB are same as the relational tables of the frame

model. Thus, there is no need for data conversion to the relational table. The detailed algorithm for data conversion from object-oriented/network/hierarchical to frame model relational database can be described as follows:

Phase I: Unload records of legacy databases into sequential files

(a) Case of converting object-oriented database to frame model relational database:

According to the translated frame model, the objects and its referenced OID of each class will be unloaded into a sequential file as follows:

1. Map association attribute (referenced OID) to a foreign key by locating its associated class's "primary key" as a result of schema translation.
2. Unload objects of each class into a sequential file with its OID.

Program Unload OODB into sequential files:

```

Begin
  Get all class C1, C2...Cn within an OODB schema;
  For i = 1 to n do
    Begin
      while Ci object found do
        begin
          Get the referred associated attribute's object from Cp where Cp is an associated
            class of Ci;
          Map the referred object to "foreign key" as a result of schema translation from
            OODB to frame model;
          Output non-referred attributes and "foreign key" to a sequential file;
        end;
      end;
    end;
  end;

```

Case of converting network database to frame model relational database:

According to the translated frame model, the data of each record will be unloaded into a sequential file as follows:

1. Create a template file to define the NDB schema and its translated frame model.
2. Unload NDB into sequential files. In the unload process, with the help of template files from step 1, an unload program will read all record occurrences of each record type of the NDB from bottom up and map each record type in a sequential file¹².

Program Unload NDB to sequential files

```

begin          /*n=number of record types, m= number of levels in each path expression
*/
  Get all record type N1, N2....Nn within input NDB schema;
  For i = 1 to n do                                /* for each target record type Ni */
    while Ni record occurrence found do
      begin IF it is first occurrence
        THEN obtain first record Ni within area
        ELSE obtain next record Ni within area;
        For j = m-1 to 1 do                          /* read target record owner records by database
                                                    navigation from level m-1 to level 1, a system-owned records
*/
          Obtain owner records keys Ki(1), Ki(2),...Ki(j) /* obtain the record keys of
                                                    all owners of record Ni along database access path

```

```

                                from bottom up to the system owned record*/
    end-for;
    Case record identifier_type of
'F': begin
                                /* fully internally identified13*/
    IF m = 1
    THEN output Ni record with Ki(m) as record identifier to sequential file i
    ELSE output Ni record with Ki(1), Ki(2)...Ki(m) as foreign key to sequential file i
                                /* Ki(m) = key of owner record key in level m */
    end;
'P': output Ni record with Ki(1), Ki(2),.. Ki(m) as record identifier to sequential file i;
                                /* partially internally identified */
'I': output Ni record with Ki(1), Ki(2),..., Ki(m-1),Sequence# as record identifier to
    sequential file i;                                /* internally unidentified */
    end-case;
end-while;
end-for;
end.

```

Case of converting hierarchical database to frame model relational database:

According to the translated frame model, the data of each segment will be unloaded into a sequential file as follows:

1. Map each segment "Access path" key to a attribute.
2. Perform unload process by reading all segment data top down. Writing data of each segment type along with its "access path" key to a sequential file.

Program Unload HDB to sequential files

```

Begin                                /* H = the number of segment types
                                */
Get all segment type H1, H2...Hh from the hierarchical input schema;
For i = 1 to h do                    /* for each target segment type */
Begin
    Get Hi1, Hi2... Hii segment types; /*get target segment Hii parent segments Hi1...Hi(i-1)
                                */
    Let j = 1 /* start from level 1 of root segment */
    While j > 0 do                    /* processing all target segment occurrences
                                */
Begin case j of
    j=1: begin /* process all root segment occurrences */
        Get next Hi1 segment;
        IF segment found
        THEN Let j = j + 1            /* go down toward target segment */
        ELSE Let j = j - 1          /* go up to get out of the loop */
    End
    i>j>1: begin /* set up parentage position */
        Get next within parent Hij segment;
        IF segment found
        THEN Let j = j + 1            /*go down toward target segment*/
        ELSE Let j = j - 1;          /* go up toward root segment */
    end;
    j=i: begin                        /* process target segment */
        while Hii segment found do
        begin

```

```

Get next within parent Hii segment;                               /*set up parentage */
Case Hii segment identifier type of
  "F": output Hii segment with its parent segment keys Hm to sequential file i;
                                          /* fully internally identified */
          /*Hm=the concatenation of parent segment keys of Hi segment*/
  "P": output Hii segment along with Hi1(key), Hi2(key)...Hi(i-1)(key), Hii(key) to
        sequential file i;                               /* partially internally identified */
  "I": output Hi1 segment along with Hi1(key), Hi2(key).....Hi(i-1)(key),.sequence#
        to sequential file i;                             /*internally unidentified*/
  case-end;
while-end;
Let j = j - 1;                                             /* go up toward root segment */
End;
case-end;
while-end;
for-end;
end.

```

Phase 2: Upload sequential files to the universal database

When the unloaded sequential files are reloaded to the universal database, the prime data are reloaded first. Then, the referenced data are reloaded using modify statements.

Case study of converting network database to relational table of universal database

Before converting data from NDB to RDB, a translated relational schema(frame model internal schema) must be defined before conversion. Figure 5 shows a NDB schema of an university enrollment system and its database with the following steps to demonstrate the different stages in the conversion process:

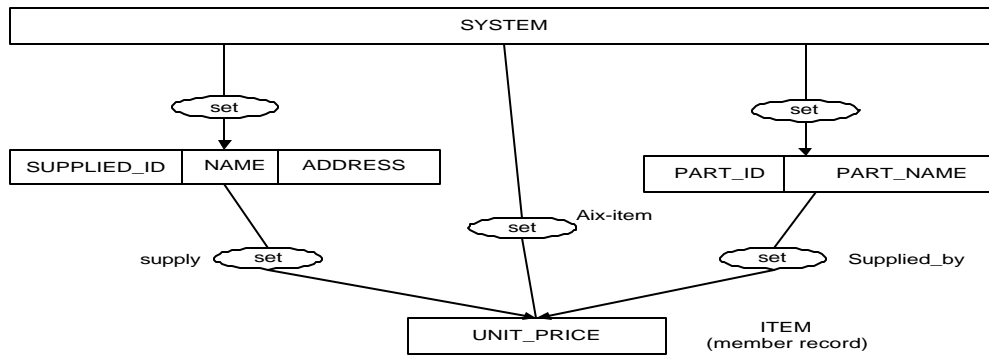


Figure 5 Network database sample schema

Sample data from the NDB could be as follows:

SUPPLIER		
SUPPLIER_ID	SUPPLIER_NAME	ADDRESS
S1	John's Co.	32 Ivy Road
S2	Michael Lee	61 Clark Road
S3	Jack's Store	90 Dicky Road
S4	Michael Lee	61 Clark Road

PART		ITEM
PART_ID	PART_NAME	UNIT_PRICE

P1	Sugar
P2	Orange Juice
P3	Beer
P4	Chocolate

4
5
6

After data conversion, the translated RDB tables of the universal database are shown below¹⁴:

SUPPLIER

SUPPLIER_ID	SUPPLIER_NAME	ADDRESS
S1	John's Co.	32 Ivy Road
S2	Michael Lee	61 Clark Road
S3	Jack's Store	90 Dicky Road
S4	Michael Lee	61 Clark Road

PART

PART_ID	PART_NAME
P1	Sugar
P2	Orange Juice
P3	Beer
P4	Chocolate

ITEM

SUPPLIER	PART_ID	UNIT_PRICE
S1	P1	4
S2	P3	6
S3	P1	5

6 Program conversion from legacy databases to universal database

To translate the legacy database program into the embedded-SQL program, the nature of each DML must be considered. The DML of HDB (e.g. IMS) and NDB (e.g. IDMS) is procedural and navigational while the DML of RDB (e.g. SQL) and OODB (e.g. OSQL and UniSQL) is non-procedural and set oriented. Thus, the translation from IMS or IDMS to SQL is reverse engineering from low to high abstract level while the translation of OSQL to SQL is peer-to-peer with the same abstract level. The processing of IMS and IDMS statements is emulated by SQL statements, and the processing of OSQL by SQL as follows (Since SQL is used as the kernel database language, the source and target database language are the same, there is no need for transaction translation.)

Case of query translation from OSQL to SQL:(by converting is the database access path)

Step 1: Explicit query language syntax transformation

During this step, the OSQL (e.g. UniSQL) is replaced by SQL query language:

```
OSQL: SELECT attributes_list_1
      FROM {class_list}
      WHERE{data_navigation_path} AND/OR {search_condition_1}
      ORDER BY {attributes_list_2}
      GROUP BY {attributes_list_3}
      HAVING {search_condition_2}
```

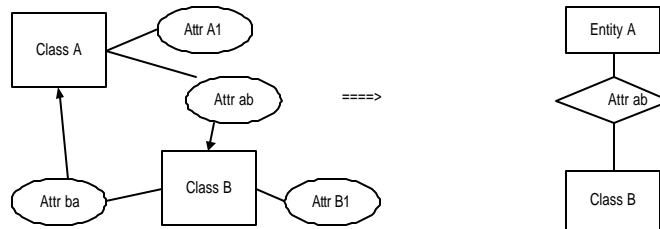
```
SQL:  SELECT {transformation of attributes_list_1}
      FROM {transformation of class_list}
      WHERE {transformation of join_condition} AND/OR
      {transformation of search_condition_1}
      ORDER BY {transformation of attributes_list_2}
      GROUP BY {transformation of attributes_list_3}
      HAVING {transformation of search_condition_2}
```

Step 2: Create OSQL query graph

Classes in OSQL are defined in FROM clause. Searching conditions and navigation path is defined in WHERE clause. To create query graph of OSQL, classes and navigation path of OSQL are formed by tracing the association attribute.

Step 3: Map OSQL query graph to SQL query graph

The mapping of OSQL query graph to SQL query graph is the same as schema mapping from OSQL to SQL.



Step 4: Transform OSQL to SQL query language.

From the query graph, subclass inherit attributes from superclass. The OSQL class navigation access path is transformed to SQL relation join access path. Path expression of OSQL will be modified according to the syntax of SQL query.

For example, map the following OSQL query to SQL query in Figure 6

```
Select motor.cover.description, motor.cover.vehicle_no, motor.policy#, motor.premium
from motor where motor.premium > 500.00 and motor.cover not = Null
```

Step 1 Explicit query language syntax transformation

```
Attributes_list-1: motor.cover.description, motor.cover.vehicle_no, motor.policy#,
motor.premium
class_list: motor, vehicle
navigation_path: motor.cover = vehicle.cover
search_condition: motor.premium > 500.00 and motor.cover not = Null
```

Step 2 and 3 Create OSQL query graph and map it to SQL query graph

The inclusion dependency involved in the OODB query language is $motor.cover \subseteq vehicle.cover$

The relation Item in RDB is a relationship relation between relation Supplier and relation Part. The OODB query graph and SQL query graph can be shown below:

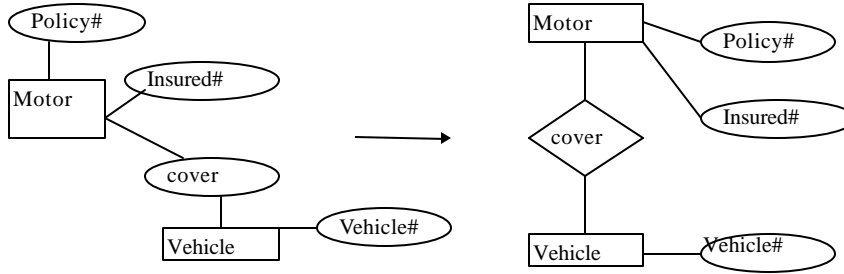


Figure 6 Map OODB query graph to RDB (SQL) query graph

Step 4 The emulated interactive SQL statement is as follows:

Class_name	Primary_key	Parents	Operation	Class_type
Motor	Policy#	0	Call SQL file	active

The translated SQL file:

```
Select vehicle.description, vehicle.vehicle#, motor.policy#, motor.premium from vehicle, cover, motor where vehicle.vehicle# = cover.vehicle# and motor.policy# = cover.policy# and motor.prem > 500.00
```

Each IDMS DML statement will be translated into corresponding SQL statement as follows:

1. Emulate Obtain statement of IDMS into Select statement of SQL -

The output of Obtain statement is to shift the cursor of a record or Set pointer. To emulate such statement in RDB, template of each relation and pointer for each foreign key must be set up to emulate the record template and Set pointer in the NDB. The cursor of the NDB can be mapped to an emulated cursor of SQL. The record search of Obtain statement is then interpreted by the table search of Select statement. But the output of the Select must be one tuple instead of a set of tuples.

2. Emulate Insert statement into Insert statement -

To insert a record in the NDB, cursor of the target record must be set up before insert. Similarly, such set up can be done in the Obtain statement before insert. The output is the creation of an additional record in NDB which maps to an additional tuple in the universal database.

3. Emulate Replace and Erase statement of IDMS into Replace and Delete statement of SQL -

Similar to Insert statement, the translator set up cursor and then perform replace or delete.

Case Study of reengineering network database programs to embedded-SQL program

To illustrate the emulation algorithm in a case study, the following is a NDB program that is to be emulated by an embedded SQL program to access the universal database in Figure 5. The sample NDB program is as follows:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CONVERTED-NETWORK-DATABASE-PROGRAM.
ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
        SOURCE-COMPUTER. DG MV10000.
        OBJECT-COMPUTER. DG MV10000.
DATA DIVISION.
SUBSCHEMA SECTION.
COPY "SUBSUPPLY.COB"
WORKING-STORAGE SECTION.
77 TXT-NO                PIC 9(10).
01 PRICE                 PIC ZZ9.
77 NO-DATA              PIC S9(9) COMP VALUE +100.
77 END-OF-SET           PIC S9(9) COMP VALUE +100.
77 ACCESS-OK            PIC S9(9) COMP VALUE +0.
PROCEDURE DIVISION.
000-MAIN-ROUTINE.
    PERFORM 100-SELECT-ITEM.
    COMMIT.
    FINISH.
    STOP RUN.
100-SELECT-ITEM.
    OBTAIN FIRST ITEM WITHIN AIX-ITEM.
    IF DBMS-STATUS NOT = 00000
        DISPLAY 'NO RECORD IN ITEM TABLE'
    ELSE
        MOVE UNIT-PRICE TO PRICE
        DISPLAY 'SUPPLIER ' AIX-SUPPLIER-ID
            ', PART ' AIX-PART-ID
            ': PRICE ' PRICE
        PERFORM 150-SELECT-NEXT-ITEM UNTIL DBMS-STATUS = 17410.
150-SELECT-NEXT-ITEM.
    OBTAIN NEXT ITEM WITHIN AIX-ITEM.
    IF DBMS-STATUS = 00000
        MOVE UNIT-PRICE TO PRICE
        DISPLAY 'SUPPLIER ' AIX-SUPPLIER-ID
            ', PART ',AIX-PART-ID, ': PRICE ' PRICE.

```

In order to emulate the processing of the NDB program by the universal database system which use a relational DBMS as a kernel system, the same facilities must be created as in the network DBMS. Our approach is to treat each record as a relational table, each SET as a SQL cursor, and then add a program area with record template to store the current tuple of each relational table. The cursor pointing to the primary keys and foreign keys are to navigate the relational table. In this example, the following program workarea is created in Figure7. The NDB program is translated by the following embedded-SQL program with a SQL Select statement to access the translated universal database where the five cursors are defined to simulate the NDB structure for illustration, but only one cursor is actually used as follows:

```

ID DIVISION.
PROGRAM-ID. RELATIONAL-DATABASE-PROGRAM.

```

```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DG MV10000.
OBJECT-COMPUTER. DG MV10000.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL DECLARE C1 CURSOR FOR SELECT * FROM ITEM
    EXEC SQL DECLARE C2 CURSOR FOR SELECT * FROM SUPPLIER
    EXEC SQL DECLARE C3 CURSOR FOR SELECT * FROM PART
    EXEC SQL DECLARE C4 CURSOR FOR SELECT * FROM SUPPLIER
        WHERE SUPPLIER.SUPPLIER_ID = ITEM.SUPPLIER_ID
    EXEC SQL DECLARE C5 CURSOR FOR SELECT * FROM PART
        WHERE PART.PART_ID = ITEM.PART_ID
END-EXEC.
01 SUPPLIER.
    05 SUPPLIER-ID                PIC X(4).
    05 SUPPLIER-NAME              PIC X(20).
    05 ADDRESS    PIC X(20).
01 PART.
    05 PART-ID    PIC X(4).
    05 PART-NAME                PIC X(20).
01 ITEM.
    05 SUPPLIER-ID                PIC X(4).
    05 PART-ID                    PIC X(4).
    05 UNIT-PRICE                 PIC 9(4).
01 PRICE        PIC ZZ9.
77 NO-DATA      PIC S9(9) COMP VALUE +100.
77 END-OF-SET   PIC S9(9) COMP VALUE +100.
77 ACCESS-OK    PIC S9(9) COMP VALUE +0.
PROGRAM DIVISION.
000-MAIN-ROUTINE.
    PERFORM 100-SELECT-ITEM.
    EXEC SQL CLOSE C1 END-EXEC.
    STOP RUN.
100-SELECT-ITEM.
    EXEC SQL OPEN C1 END-EXEC.
    EXEC SQL FETCH C1 INTO :ITEM END-EXEC.
    IF SQLCODE = NO-DATA
        DISPLAY 'NO SELECTED RECORD IN ITEM TABLE'
    ELSE
        MOVE UNIT-PRICE TO PRICE
        DISPLAY 'SUPPLIER ' ITEM.SUPPLIER-ID ', PART ' ITEM.PART-ID, ': PRICE ' PRICE
        PERFORM 150-SELECT-NEXT-ITEM UNTIL SQLCODE = END-OF-SET.
150-SELECT-NEXT-ITEM.
    EXEC SQL FETCH C1 INTO :ITEM END-EXEC.
    IF SQLCODE = ACCESS-OK
        MOVE UNIT-PRICE TO PRICE
        DISPLAY 'SUPPLIER ' ITEM.SUPPLIER-ID ', PART ' ITEM.PART-ID, ': PRICE ' PRICE.

```

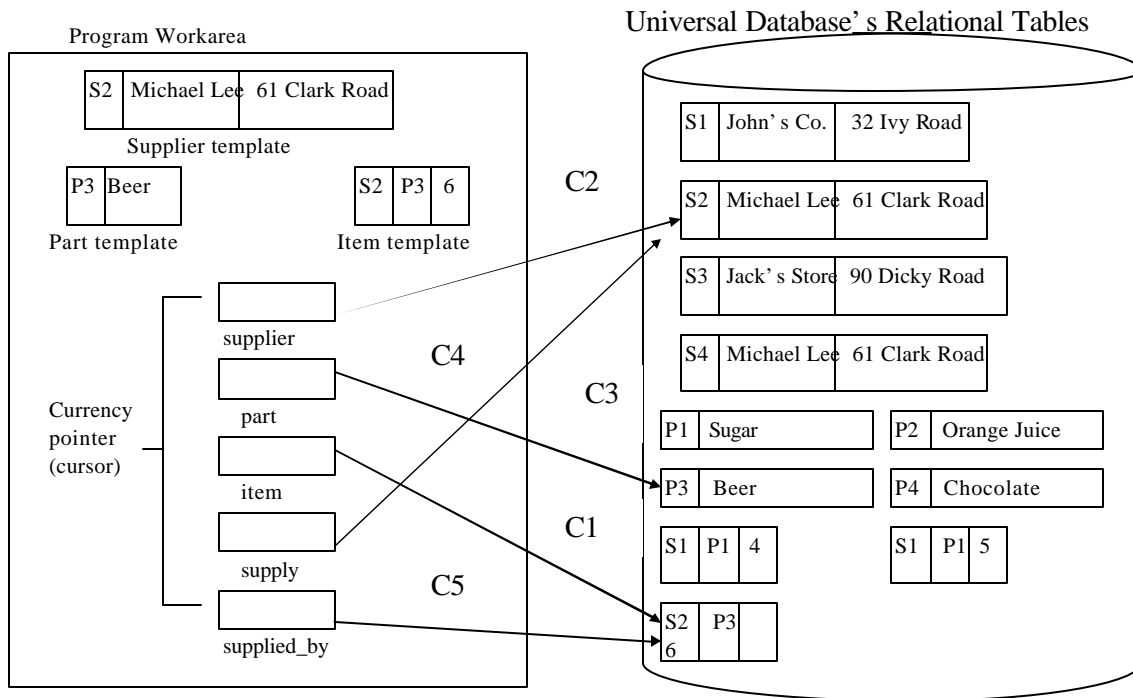


Figure 7 Database navigation emulated by cursors

7 Prototype

A prototype has been implemented for schema translation from RDB schema to frame model. The process is to capture the semantics of entity, weak entity, cardinality, isa, generalization, categorization and aggregation of the relations in relational schema to the header, attribute, method and constraint classes in the frame model. The translation process can be accomplished partially by data querying/data mining its semantics derived from the physical relational database data occurrences with minimum users supervision. For example, the subclass and superclass relations can be derived by confirming all the subclass relation tuples appear in its superclass relation tuples with the same primary key¹⁵. The constraints of the source RDB schema can be preserved by the data operation in the method class and constraint class in the frame model. A sample of the frame report can be shown in Figure 8.

For the application of frame model, users can use SQL to access the frame model database. A frame model Application Program Interface has been developed¹⁶ to invoke the data operation in the frame model for data semantics management. The API process handles the method defined by Constraint class during data modification. The process defined and executed by RDBMS is to provide triggering event fired on the class. When RDBMS process is invoked, it will check from the Constraint class for any associated method. If method is defined in the Constraint class, the method definition will be queries. The process will invoke the stored procedure defined by the queried method definition. If executed result returned by stored procedure violated the rule defined in Constraint class, error message will be returned to the user (refer to Figure 9).

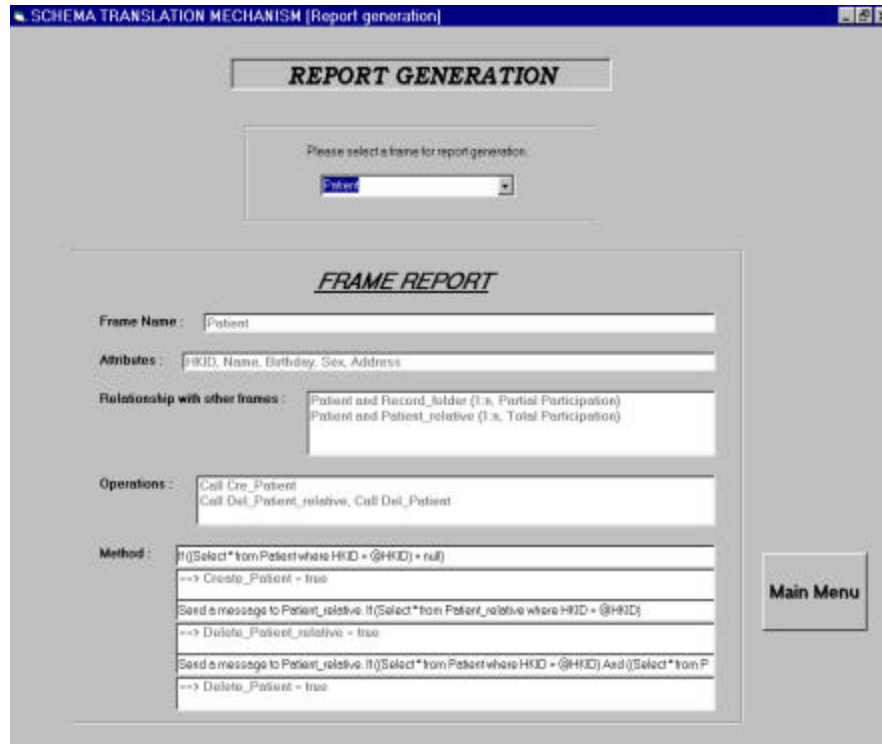


Figure 8 A sample report for frame model

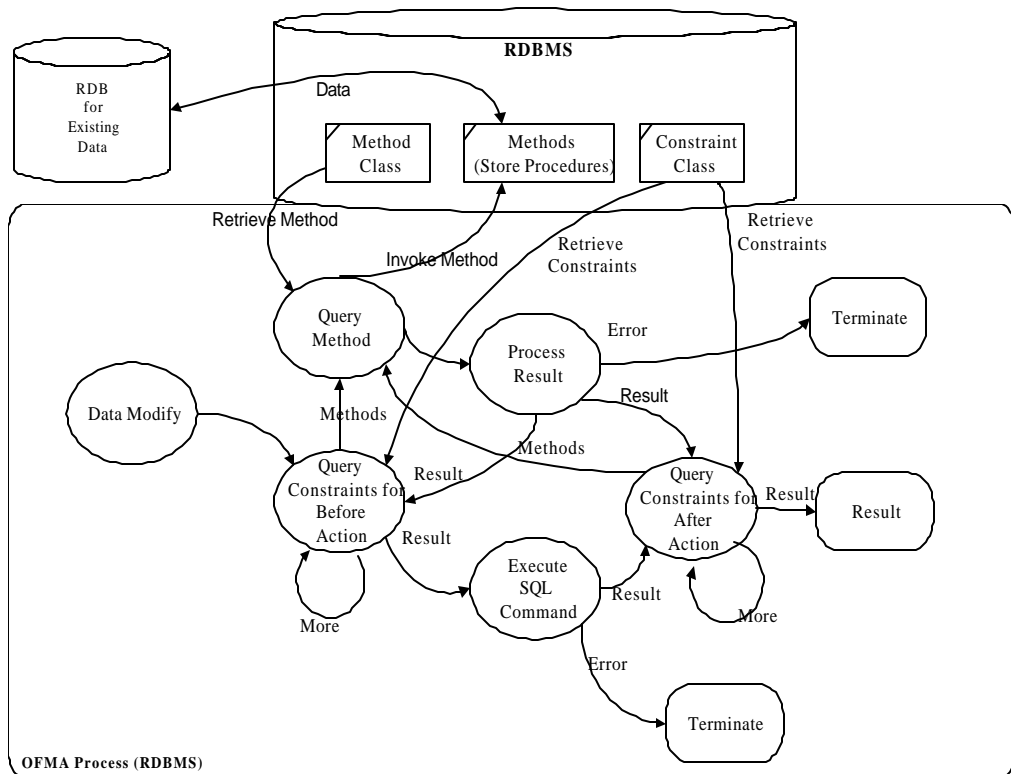


Figure 9 Data flow of frame model API

8 Conclusion

This paper describes a frame model approach to implement a universal database to integrate hierarchical, network, relational and object-oriented data model into an uniform frame model. The frame model in the form of conceptual schema is for data modelling technique. The frame model in the form of logical schema is for Data Definition Language. The frame model consists of classes to preserve constraints after schema translation and integration, and includes relational table to store actual data. The frame model has an operation attribute to store program call to emulate method of OODB. The processing of universal database has five steps: schema translation, schema integration, data conversion, DML emulation and RDBMS processing. Each source database system converts their schema and data into universal databases to form a data warehouse for decision support system. In order for existing database program access the universal database, their DMLs are emulated into SQL statements, which are executed by the kernel relational DBMS. Templates and cursors are set up to emulate the program workarea and pointers in network and hierarchical database. The benefit of the universal database is to let database systems users share their databases among each other without program changes. The impact of this methodology is let database systems integrate each other via universal database in the same way as computer networks connect each other via the Internet.

Compared with other related work, the major contribution of the universal database is its introduction of frame model which can capture data semantics as well as data operation in a meta data as a knowledge-based system and its abilities to resolve constraints conflicts between different data models in the heterogeneous DBMS environment. Furthermore, since schema translation can be done in pre-process, the transaction translation in the database gateway can also be done by DML substitution without much delay at the run time. The performance of the database gateway is thus acceptable for data warehousing.

Reference

1. Fong, Joseph and Huang, Shi-Ming, Information Systems Reengineering, ISBN 981-3081-15-8. *Springer-Verlag*. (1997).
2. Gray, P.M.D., Kulkarni, K.G., and Paton, N.M., Object-Oriented Databases: A Semantic Data Model Approach, *Prentice Hall*, New Jersey, ISBN 0-13-630203-3, (1992).
3. Houstsma, M.A.W. and Apers, P.M.G, Data and Knowledge Model: A Proposal", Advances in Database Programming Languages, *ACM Press*, New York, ISBN 0-201-50257-7, (1990).
4. Howe, D, Data Analysis for Database Design, London: Edward Arnold, p19, (1989).
5. Date, C.,, An Introduction to the unified data language(UDI), *IEEE Computer*, p15-29 (1980).
6. Hsiao, K. David and Kamel, N. Magdi, Heterogeneous Databases: Proliferations, Issues, and Solutions, *IEEE Transactions on Knowledge and Data Engineering*, p45-62, (1989).
7. UniSQL, Inc, UniSQL/X User's Manual, *UniSQL Inc*, (1992).
8. Tardieu, H., and Franckson, M, Contribution of the Entity-Relationship Approach to Object

Management in an Information System Design Workbench, Proceedings of the fifth International conference on Entity Relationship Approach, published by *North-Holland*, pp529-555, (1987).

9. Orr, Ken, Data Quality and Systems Theory, *Communications of the ACM*, Vol 41, No 2, pp67-72, (1998).
10. Lu, H., Fan, W., Goh, C., Madnick, S. and Cheung, D, Discovering and reconciling semantic conflicts: a data mining perspective, *Data Mining and Reverse Engineering, Searching for semantics*, Chapman & Hall, pp409-427, (1997).
11. Fong, Joseph and Bloor, Chris, Data conversion rules from network to relational databases, *Information and Software Technology*, Volume 36, Number 3, P141-154, (1994).
12. Navathe, S. and Awong, A., Abstracting relational and hierarchical data with a semantic data model, *Entity-Relationship Approach*, p305-333, (1988).
13. Fong, Joseph, , Adding a Relational Interfaceto a Nonrelational Database, *IEEE Software*, September 1996, p89-96, (1996).
14. Wong, D and Fong, J, Methodology for schema translation from relational database to frame model using data mining, to appear in the Proceedings of the 9th International Database Conference in July 1999, published by CityUPress.
15. So Chung Wai, OFMA: Object Frame Model Agent, M.Sc dissertation, Department of Computer Science, City University of Hong Kong (1998).
16. Tawbi, C, Jaber, G and Dalmau, M, Rule Specifications in Active Relational DBMS, *Proceedings of the 6th International Conference on Database and Expert Systems Applications (DEXA' 95)*, pp188-196, (1995).