Delta-Relational Mapping Using The Abstract Behavioral Specification Language

Niken Fitria Apriani

Faculty of Computer Science, Universitas Indonesia
Depok 16424, Indonesia
niken.fitria@ui.ac.id
http://fmse.cs.ui.ac.id/

Abstract. Feature variability of an application in SPL (Software Product Line) can affect the design and the implementation of database schema of the application, if the feature is related to the data storage of the application. However, this crucial thing is not handled by the management technique of variability in SPL, hence this can make database schema inconsistent and incompatible with the application requirements. This study shows delta modeling in ABS is able to adjust database schema based on requirement changes to realize feature variance. To this end, we develop mapping mechanism of delta modeling in ABS into relational database schema. Furthermore we design a case study to evaluate the effectiveness of mapping technique. By integrating the mapping technique with existing ABS Tools, the appropriate database schema for each product of the SPL can be generated.

Keywords: ABS, delta modeling, delta, core, relational database, database schema, constraints

1 Introduction

The development of software engineering cannot be separated from the influence of market and industry demand. The development of market and industry nowadays demand for software which has variability be suited with the choice and the need of the users. This could cause repeated requirement changes for the same kind of software or system.

The problem above actually has been addressed by a methodological approach called SPL (Software Product Line). Software product line (SPL), is a methodology for developing diversed software products and software-intensive systems. According to (1), SPL can be used to develop software products with diversity at lower cost, in shorter time, and with high quality, compared to single product development methodology. ABS (Abstract Behavioral Specification) is a modeling language which supports SPL methodology (2). In SPL software variability is commonly expressed using features, and in ABS these features are organized in feature model to express the dependencies between features (3).

Moreover, ABS also applies delta modeling to handle requirement changes in the process of feature adjustment, which is by implementing changes without changing the default system. Many software application relies on database to manage data, therefore database is an important element of software system. Furthermore database is also influenced by requirement changes resulted by SPL implementation. In SPL, the variability of software, information system for example, will cause variability in database schema and database implementation. Until now, there is no technique in SPL methodology to manage variability in database schema and database implementation, especially in information system (5). This paper is concerned in applying delta modeling in ABS to not only influence and adjust the SPL implementation, but also the changes of database schema.

1.1   Introduction of Case Study

The case study used in this research is a simplification of the donation report based on the real-life activities done by charity organizations. Hence, the requirement analysis is done based on the charity organizations' programs and activities. The donation report system will be focused on the income that is the source of the donation and the expenditure, which consists of projects and recipient.

Based on the study done into some charity organizations, the sources of fund for the charity organizations are donor and profit. The donors could be divided into two types, which are individual and institutional. On the other hand, the recipient of the aid could be divided into three types that are individual (for example the recipient of the scholarship), institution (for example orphanage, school, or hospital), and region (for example the recipient of bridge construction).

Table 1. Type of Projects in Charity Organizations

Field of Activities	Examples	Type of Activities
Health	Hospital construction	Construcion
	Operational cost for free hospital	Regular
	Aid for medical expenses	Once
Education	School construction	Construction
	Scholarship	Regular
	Aid for textbooks expenses	Once
Empowerment	Bridge construction	COnstruction
	Development of quality seeds	Regular
	Aid for capital of livestock	Once
Disaster Relief		Once

Projects of charity organizations could be seen from two main point of views; those are the field of activities and the type of activities. The field of activities consists of health, education, empowerment, and disaster relief. Furthermore, the

activities of each field could be categorized into three types of activities, which are once, regular, and construction. Once-type describes activities that are done only once. Regular-type is for activities that need to be done regularly with fixed amount of money. And the third type of activities is construction-type that is for activities which build something permanently.

The differences of the project types above lie on the kind of data which are need to be stored. The once-type is the common type of the projects, which contains id, name, amount, and date of the project. Furthermore the regular project needs to store more than one date, because the projects are done regularly. Lastly, the construction type needs to store the payment, which contains the amount and the date of the aids payment. The tables below show the data differences among the type of projects, viewed from end-user.

Table 2. Data Stored for Once Type Project

Id	Name	Amount	Date
x	xxxx	xxxxx	xxx

Table 3. Data Stored for Regular Type Project

Id	Name	Amount	Date
x	xxxx	xxxxx	xxx
			xxx
			xxx

Table 4. Data Stored for Construction Type Object

Id	Name	Amount	Payment	
			Amount	Date
x	xxxx	xxxxx	xxx	xxx

Based on the explanation above, a feature model is designed to define the features needed in this case study. The basic functionality of this donation report is the project recording, hence project becomes mandatory along with once-type, and meanwhile the others are optional.

Furthermore, a class diagram of the donation report system is also made to see the relationship between the class and the attributes needed of each class.
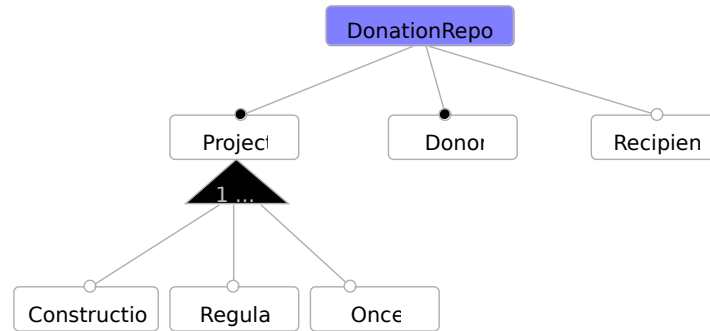
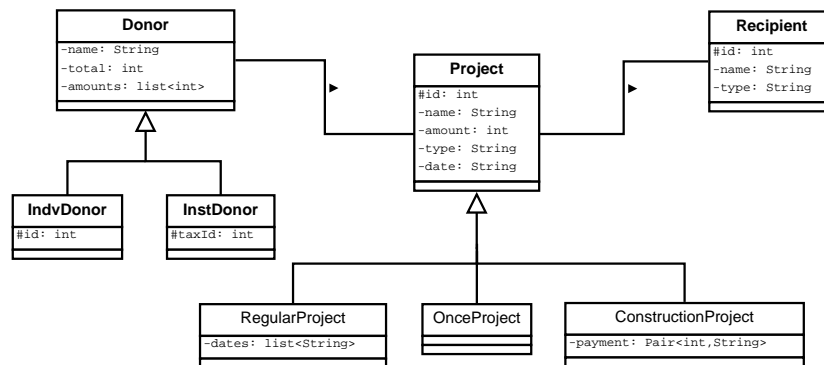**Fig. 1.** Feature Diagram of Donation Report

Fig. 2. Class Diagram of Donation Report

The introduction part of this paper describes the background problem, motivation, and requirement analysis of the case study. The next section of this paper discusses the mapping technique from delta oriented programming into relational database introduced by this research. This section discusses about relational database schema elements, core ABS and delta modules, and introduces mismatch between ABS language and relational database schema. The following section describes the implementation of the case study and implementation of mapping technique on the case study. This section discusses the expected results, case study implementation in ABS, implementation of mapping technique on the case study, and evaluation. The last section provides the conclusion and the future work of this research.

2   Mapping Technique

This section describes the proposal of mapping technique used in the mapping process from ABS code into relational database schema. The mapping technique is defined based on relational database schema and the basic syntax of ABS modules. This section will be divided into several parts as follows:

1. The steps used in defining the mapping technique
2. The elements of relational database schema as the target of the mapping
3. The mapping of core ABS component and delta module into relational database schema elements
4. The mismatch between ABS language and relational database schema and the solution for it

2.1   The Methodology of Mapping Technique

ORM(Object Relational Mapping) does the mapping works between object model in object oriented programming and relational model in relational database. There are three type of perspective used by ORM to do the mapping, those are:

1. Metadata-oriented, which uses metadata as the input to generate the mapping code
2. Application-oriented, which starts from object-oriented application to do the mapping
3. Database-oriented, which does the mapping from the database.

The mapping technique proposed in this research uses the last perscpective used by ORM, which is database-oriented approach. Hence the approach to do the mapping is started from the target of the mapping, which is the relational database schema. The following are the steps used in defining the mapping technique:

1. Analyze the aspect of relational database schema
   In this step we obtain the elements of relational database schema and what is needed to create a relational database schema. Based on those elements, we define the information which needs to be provided by the ABS model so it can be generated into database schema.

2. Analyze the component of ABS modules
   In this step we analyze the component of ABS modules and their syntax. Then we define the components which provide the information needed to create a relational database schema. After that, those components are mapped into the corresponding relational database schema's elements.
3. Analyze the mismatch between ABS language and relational database schema and define the solutions
   This step is done by revisiting step 1 and step 2. In this step we find the elements of relational database schema which are not provided by the ABS language. Furthermore, we also find the components of ABS modules which are not available in relational database concept. So we define the solution to the mismatch, by considering the constraints of each concept.

2.2 Relational Database Schema Elements

The main elements of relational database schema are tables and constraints. Below are the components of a relational database schema.

Table 5. The Representation of Relational Database Schema Components

schema	table	attribute	name
			data type
			[constraints]
		:	
		constraints	Primary Key
			[Foreign Key]
	:		

In a database schema there will be at least one table. In each table, there will be at least one attribute, there must be one primary key, and there could be any number of foreign keys. In each attribute, there must be an attribute name, there must be an attribute data type, and there could be any attribute constraints. Hence, we should extract the ABS code to get the information about:

− table(s)
  • attribute(s)
    * attribute name
    * attribute data type
    * attribute constraint(s)
  • primary key
  • foreign key

2.3   Core ABS and Delta Modules

Based on the analysis on previous section, the information needed to generate a database schema is provided by the core modules of ABS. However ABS implements delta-oriented programming. Hence, we should also consider the modification done by the delta modules which will affect the ABS core.

Core ABS Mapping Core ABS contains at least one module. In each module, there could be any import and export from other modules, any interface declarations, any class declarations, any data type declarations, any function declarations, and main block. For every declared interface, there could be any abstract method declarations. For every class, there could be any class parameters, any interfaces implemented, any fields, and any methods. For every class parameter, there must be a data type and a name. Whereas for every field in a class, there must be a data type and a name, and there could be an initial value.

Table 6. The Representation of Core ABS Components

core	module			
		export		
		import		
		interface	abstract method	
			:	
		:		
		class*	parameter*	data type*
				name*
			:	
			implemented interface	
			:	
			field*	data type*
				name*
				value*
			:	
			method	
			:	
		:		
		data type declaration		
		:		
		function declaration		
		:		
		main block		
	:			

In Table 6, the components that are related with the information needed to generate a database schema are marked with (*). Note that any imported and exported items will not affect the database schema. Furthermore, any methods

in the interface and class, and also functional declaration will not affect the database schema. However, the interface and data type declaration are related with the database schema, because the implemented interface affect the type of the object and data type declared could be the data type of a class parameter or a field. These issues will be discussed in the next section. Table 7 contains the mapping of each component of a module with relational database schema components.

Table 7. Mapping of ABS Module Component into Database Schema Component

Module Component		Database Schema Component
Class	–>	Table
Class Parameter - Data type - Name	–> –> –> –>	Attribute - Attribute type - Attribute name - NOT NULL constraint
Field - Data type - Name -Value	–> –> –> –>	Attribute - Attribute type - Attribute name - Default value for DEFAULT constraint

A class in ABS is considered as a relation or table in relational database schema. Furthermore, the class parameters and the fields of a class will be considered as the attributes or columns in relational database schema, with the data type as the attribute type and the name as the attribute name. Class parameters in ABS must be initialized when creating a new object. Hence, attributes generated from class parameter will have NOT NULL constraints. In ABS, a field declaration in a class declaration must be initialized with a certain value. Hence the value will be used as the default value in the database schema. In other words, objects in ABS will be considered as tuples or rows in the relational database schema.

Delta Modules Table 8 contains the analysis between the possible modifications done by delta modules and the effects on the database schema. Note that every modification related to method and function will not affect the database schema.

2.4   Mismatch between ABS Language and Relational Database Schema

There are several mismatches between ABS language and relational database schema. These mismatches include primary key in relational database, predefined data type in ABS, parametric data type in ABS, object type in ABS, and referential integrity in relational database.

Table 8. Analysis of Delta Module Modification Effect on Database Schema

Possible Modification by Delta		Effect to Database Schema
Adding new class	–>	Add table
Adding new interface	–>	-
Modifying class - Add attribute - Remove attribute - Add/remove method	–> –> –>	Add column Remove column -
Modify interfaces (modify abstract methods)	–>	-
Remove class	–>	Remove table
Modify functional entities - Add and modify data types - Add and modify type synonyms - Add functions	–> –> –>	- - -

Primary Key in Relational Database Primary key is used as the identity of the tuple in the relation. It is important especially in accessing the tuple and when the tuple has relationship with another tuple in another relation. Basically, primary key can be considered as the object identity in object-oriented concepts. However, in ABS, there is no concept of primary key as in relational database. Hence, we will use annotations in ABS to provide the information about primary key in class declarations in ABS.

ABS supports annotations to enrich an ABS model with additional information. Annotations in ABS can appear before any declaration and type usage in ABS programs. In this case, we will use annotation [PK] to describe a class parameter or a field of a class as the primary key of the relation. Note that a relation can have one or more attributes as the primary key. Hence, in a class declaration in ABS the annotation [PK] can be used more than once, either in class parameter or in field declaration. The code below is an example of using annotation in declaring the primary key in class declaration in ABS.

```
1    class Cclass([PK] Int id, String str) implements c{
2
3        [PK]Int val = 0;
4        String s= "";
5        Bool b = True;
6
7    }
```

Predefined Data Type in ABS There are several differences in data types between ABS and SQL. Table 9 contains the mapping of data types in ABS to basic data types in SQL.

Table 9. The Mapping of Predefined Data Types in ABS to Basic Data Types in SQL

ABS Predefined Data Type		SQL Basic Data Type
Bool	–>	Boolean
Unit	–>	NULL
Int	–>	INT
Rat	–>	FLOAT
String	–>	VARCHAR(n)
Fut<T>*		
List<A>**		

Fut<T>is a data type that represents a future, as a result of an asynchronous method call. In this research data type Fut<T>is not mapped into any SQL basic data types. Data type List<A>is one of the parametric data types in ABS. Hence the mapping of it will be discussed in the next section.

Regarding the data type, ABS also has abstract data type declarations. For abstract data type, only the functions that operate on them are known to the client, but not its data type constructors. Hence the initialization or modification of fields which have this type is usually done by calling the functions. This can be realized in ABS by putting such a data type in its own module and by only exporting the data type and its functions, without exporting the constructors. However, abstract data types could also appear in the class declaration, either as the data type of the class parameter or as the data type of the field. This is handled by assigning the name of the abstract data type as the attribute name, whereas the attribute type is assigned as VARCHAR(n).

Another thing to be noted is the maximum length of the VARCHAR type. For this research, the length of the string is capped at 30 characters. Hence every String data type in ABS will be mapped to VARCHAR(30) in SQL database schema.

Besides the predefined data types mentioned above, there are also several predefined data types in ABS Standard Library which can break one of the constraints of the relational database. Those data types are Pair and Triple. Below is the data type declaration of Pair and Triple in ABS Standard Library.

```
1    data Pair<A,B> = Pair(A,B);
2    data Triple<A,B,C> = Triple(A,B,C);
```

A field or class parameter with Pair or Triple type will have more than one value. Hence, these data types will result multivalued attributes, which break the domain constraints. To avoid this, the value of Pair data type and Triple data type will be separated into several attributes in database schema. For Pair data type, there will be two attributes. The first attribute will have [fieldName]0 as the attribute name, and the attribute type of it will be the first type parameter defined inside the angle brackets (<>). Whereas the second attribute will have

[fieldName]1 as the attribute name, and the attribute type of it will be the second type parameter defined inside the angle brackets (<>).

As for the Triple data type, there will be three attributes. The first attribute will have [fieldName]0 as the attribute name, and the attribute type of it will be the first type parameter defined inside the angle brackets (<>). Whereas the second attribute will have [fieldName]1 as the attribute name, and the attribute type of it will be the second type parameter defined inside the angle brackets (<>). And the third attribute will have [fieldName]2 as the attribute name, and the attribute type of it will be the third type parameter defined inside the angle brackets (<>).

Parametric Data Type in ABS In ABS, parametric data types usually used to define general-purpose data types, such as lists, sets, and maps. Those three are already defined in the ABS Standard Library. However, these parametric data types are breaking the domain constraints of the relational database schema, because parametric data types will produce multivalued attribute. As the solution, multivalued attributes must be represented by separate relations. Hence, every class parameter or field in ABS with list, set, or map type, will be generated as a new table in database schema.

For a field or class parameter with list type, the name of the new table will be List[className]. Then the name of the field will be the attribute name in the new table. The attribute type of this new attribute is the type parameter defined inside angle brackets (<>). Besides that this new table will have the primary key attributes of the class as the additional attributes. The name of the additional attribute will be [attributeName]_[className]. The primary key of this new table is the whole attributes of it. This new table will have at least one foreign key, which is every additional attribute will refer to the corresponding attribute of the class.

For a field or class parameter with set type, the name of the new table will be Set[className]. Then the name of the field will be the attribute name in the new table. The attribute type of this new attribute is the type parameter defined inside angle brackets (<>). Besides that this new table will have the primary key attributes of the class as the additional attributes. The name of the additional attribute will be [attributeName]_[className]. The primary key of this new table is the whole attributes of it. This new table will have at least one foreign key, which is every additional attribute will refer to the corresponding attribute of the class.

For a field or class parameter with map type, the name of the new table will be Map[className]. Because one element in a map contains two things, those are key and value, this new table will have two attributes. The first attribute will contain the key, and the attribute name will be [fieldName]0. The attribute type of the first attribute will be the first type parameter defined inside angle brackets (<>). The second attribute will contain the value, and the attribute name will be [fieldName]1. The attribute type of the second attribute will be the second type parameter defined inside angle brackets (<>). Besides that this new

table will have the primary key attributes of the class as the additional attributes. The name of the additional attribute will be [attributeName]_[className]. The primary key of this new table is the whole attributes of it. This new table will have at least one foreign key, which is every additional attribute will refer to the corresponding attribute of the class.

Object type in ABS Only interfaces can be used as types in ABS. Hence the type of the object is based on the interfaces implemented by the class. Since class can implement an arbitrary number of interfaces and an interface can extend arbitrary number of other interfaces, there will be many possible types for the object of a class. Moreover we could not define the object type in the class declaration, because this information will only be available at runtime.

To provide the information of the object type, we will create an additional table for each interface implemented in core ABS. The additional table for an interface will have several attributes. The first attribute is the generated id. The rest of the attributes are all primary key attributes of all classes which implement the interface. Note that the implemented interfaces are not only the interfaces declared in class declarations, but also include the interfaces extended by those interfaces. In other words, the information of interface hierarchy is needed to define all the classes which implement a certain interface. The primary key attributes of each class in the additional table will refer to the primary key attributes of the table of the class as the primary key. And the primary key of the additional table will be the attribute which contains the generated id. Later, when a new object is created, a unique id is generated. Then the unique id along with the primary key value of the object will be stored in the interface type table.

Figure 3 is the example to describe the explanation above. There are three interfaces; those are Customer, Corporate, and Government. The Corporate interface extends Customer interface. Then there are four classes; those are Individual, Family, Company, and Ministry. The Individual class implements Customer interface. The Individual class has id and name as the attributes, with id as the primary key. The Family class implements Customer interface. The Family class has id, name, and member as the attributes, with id and name as the primary key. The Company class implements Corporate interface. The Company class has taxId, name, and branch as the attributes, with taxId as the primary key. The Ministry class implements Corporate interface and Government interface. The Ministry class has regNum and name as the attributes, with regNum as the primary key. This situation is represented by Figure 3.

Based on the interface hierarchy, the possible object type for Individual class and Family class is Customer. The possible object types for Company class are Customer and Corporate, whereas the possible object types for Ministry class are Customer, Corporate, and Government. Hence, the database schema diagram will be as shown in Figure 4.
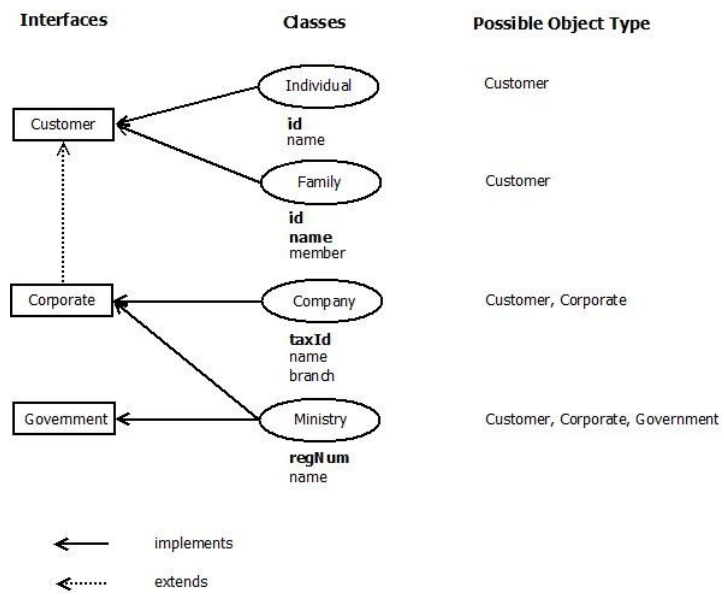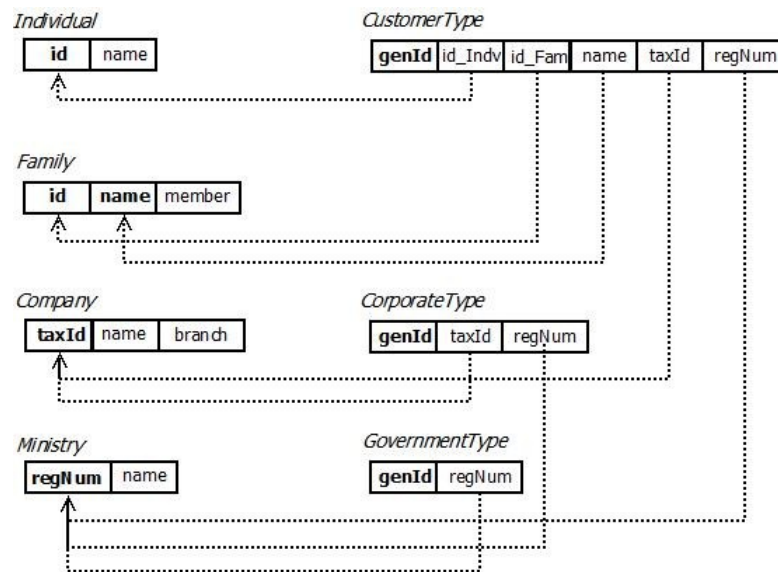
Fig. 3. The Example of Retrieving Object Type

Fig. 4. Database Schema Diagram

Referential Integrity in Relational Database There is another condition in ABS where the referential integrity constraint in relational database could be violated; that is when a class declaration has other objects as the class parameter or field. ABS, which is object-oriented, enables an object to have another object as its field. When a field is generated as the attribute in relational database, it is impossible to store the whole information of an object in an attribute. Hence, the solution is every time there is a class parameter or field declaration has interface type as the data type, the attribute generated will only contain the generated id of the object which is stored in the interface type table. This attribute will be the foreign key which refers to the primary key of the inference type table.

To describe the explanation above, we will use the same situation as the example beforehand. We add a new interface called Account. This interface is implemented by AccountImpl class, which has id, owner, and balance as the attributes, and the attribute id is the primary key. Note that one of the attribute use interface type as the data type; that is customer. This additional situation is described by Figure 5 below.
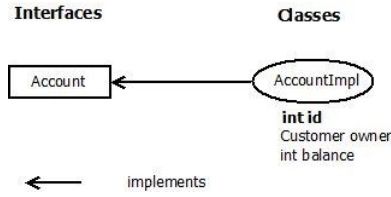

Fig. 5. Account Interface and AccountImpl Class

Figure 7 presents the relational database schema based on the explanation above.

3   Implementation of Mapping Technique towards Case Study

This section presents the implementation of the case study and implementation of the mapping technique of the case study. The requirements and the design of the case study were already presented in the Introduction. The presentation of this section starts with the expected results of the mapping technique applied to the case study. The next subsection discusses the implementation of the case study in ABS. The mapping technique is then implemented on the ABS code of the case study to generate the database schema for each product. The last subsection is the evaluation towards the generated database schema based on the expected results defined before.
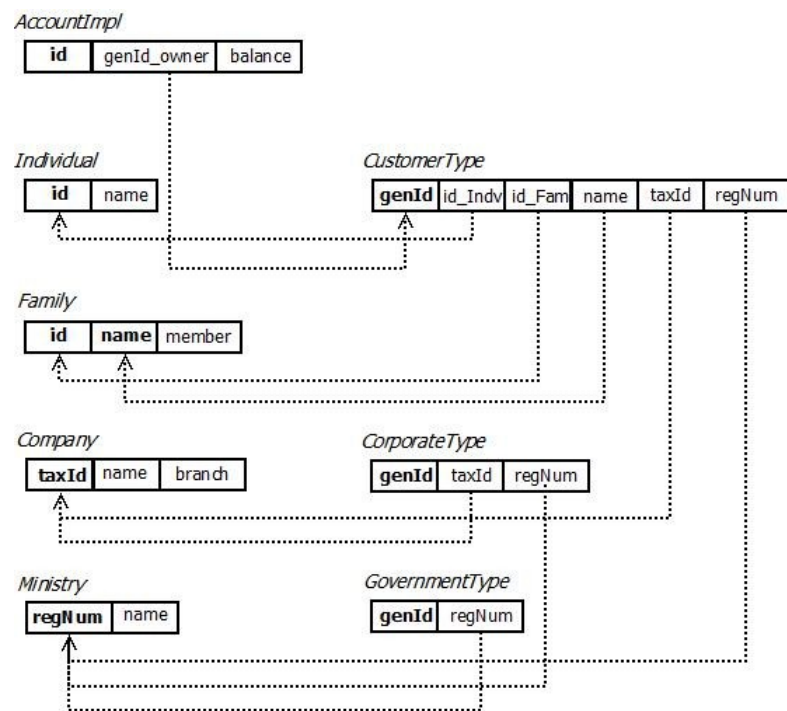
Fig. 6. Database Schema Diagram with AccountImpl Class

3.1   The Expected Results

Below are the possible products which can be produced in this software product line:

1. Once Project: this is the base product, which will store the list of once projects and the donors
2. Regular Project: this product will store the list of regular projects and the donors.
3. Construction Project: this product will store list of construction projects
4. Once Project with Recipient: this product will store list of once projects, donors, and recipients.
5. Regular Project with Recipient: this product will store list of regular projects, donors, and recipients.
6. Construction Project with Recipient: this product will store list of construction projects, donors, and recipients.

Note that, from the class diagram there are two types of donors, those are individual donor and institutional donor, whereas the type for project will be realized using delta modeling in ABS. Figure 8 contains the schema diagrams for each class in the class diagram.

Now we will define the database schema for each possible product mentioned above. The database schema will be represented by the schema diagram. The first product, Once project, only realizes feature Donor and feature Project. Figure 9 is the expected database schema.

Note that, it is impossible for an attribute to refer two different attributes in the same time. This case is already solved by the mapping technique in ABS, and the result will be presented in the next section. The second product is Regular project, which realizes feature Donor and feature Regular project, with the expected schema diagram shown by Figure 10.

The third product is Construction project, which realizes feature Donor and feature Construction project, with the expected schema diagram shown by Figure 11.

The fourth product realizes feature Donor, feature Once project, and feature Recipient. There will be a foreign key from relation Project to relation Recipient to represent the association between these relations. The expected schema diagram is represented by Figure 12.

The fifth product realizes feature Donor, feature Regular project, and feature Recipient. There will be a foreign key from relation Project to relation Recipient to represent the association between these relations. The expected schema diagram is represented by Figure 13.
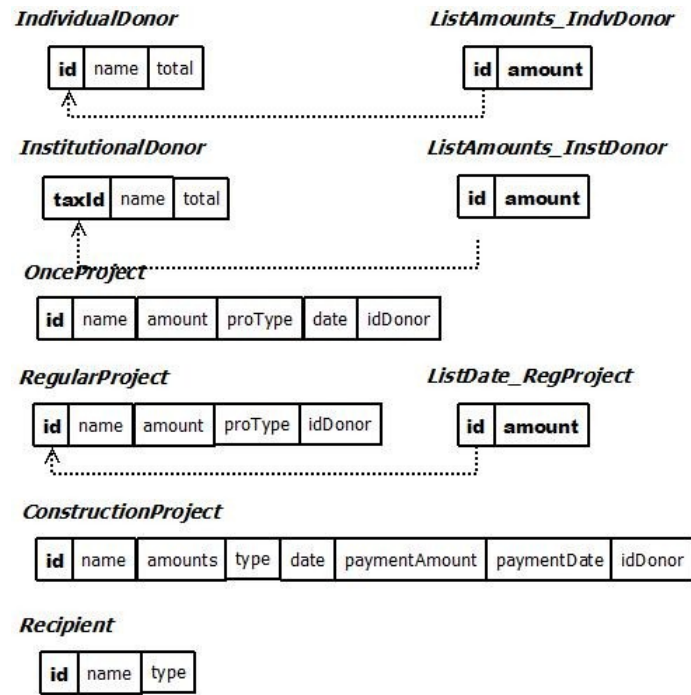
IndividualDonor

id	name	total

ListAmounts_IndvDonor

id	amount

InstitutionalDonor

taxId	name	total

ListAmounts_InstDonor

id	amount

OnceProject

id	name	amount	proType	date	idDonor

RegularProject

id	name	amount	proType	idDonor

ListDate_RegProject

id	amount

ConstructionProject

id	name	amounts	type	date	paymentAmount	paymentDate	idDonor

Recipient

id	name	type

Fig. 7. Expected Schema Diagram for each Class of Case Study

IndividualDonor

id	name	total

ListAmounts_IndvDonor

id	amount

InstitutionalDonor

taxId	name	total

ListAmounts_InstDonor

id	amount

OnceProject

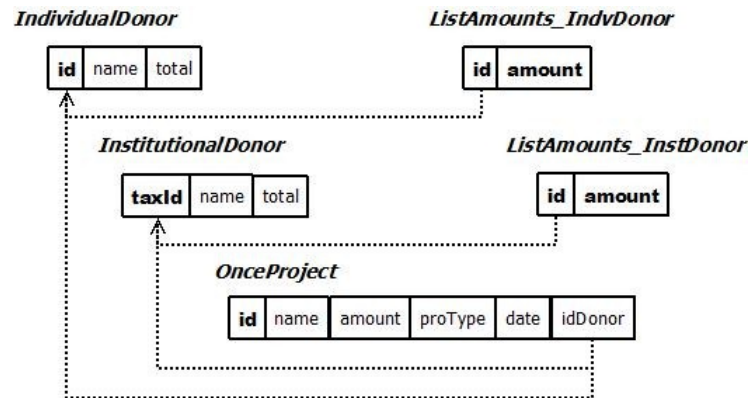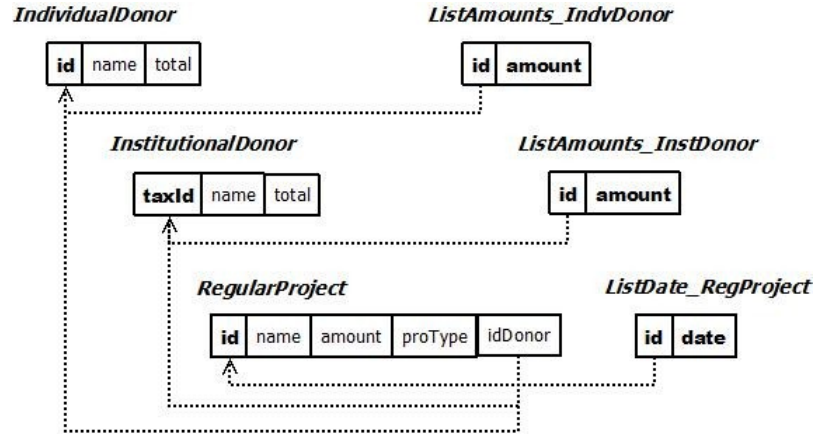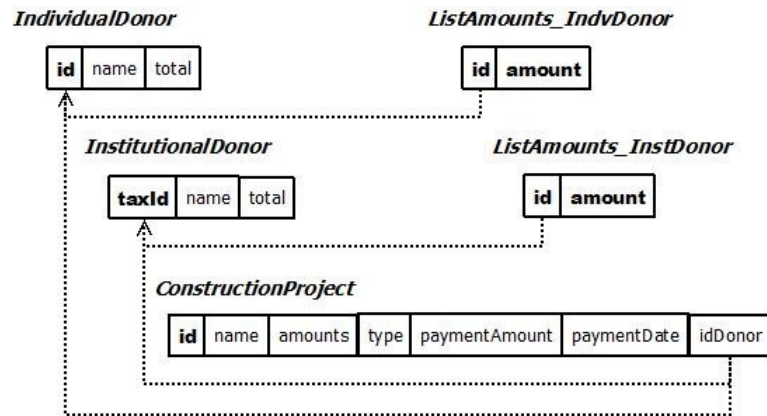| id	name	amount	proType	date	idDonor

Fig. 8. Expected Schema Diagram for the First Product

Fig. 9. Expected Schema Diagram for the Second Product
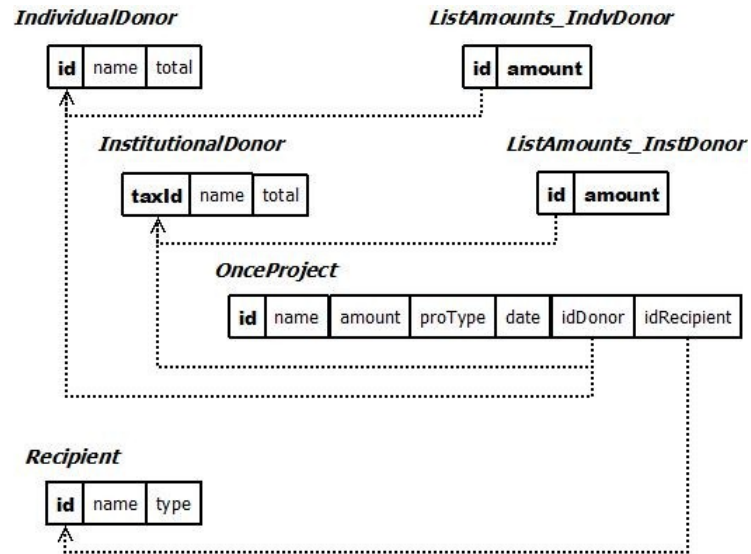

Fig. 10. Expected Schema Diagram for the Third Product

IndividualDonor

id	name	total

ListAmounts_IndvDonor

id	amount

InstitutionalDonor

taxId	name	total

ListAmounts_InstDonor

id	amount

OnceProject

id	name	amount	proType	date	idDonor	idRecipient

Recipient

id	name	type

Fig. 11. Expected Schema Diagram for the Fourth Product

IndividualDonor

id	name	total

ListAmounts_IndvDonor

id	amount

InstitutionalDonor

taxId	name	total

ListAmounts_InstDonor

id	amount

RegularProject

id	name	amount	proType	idDonor	idRecipient

ListDate_RegProject

id	date

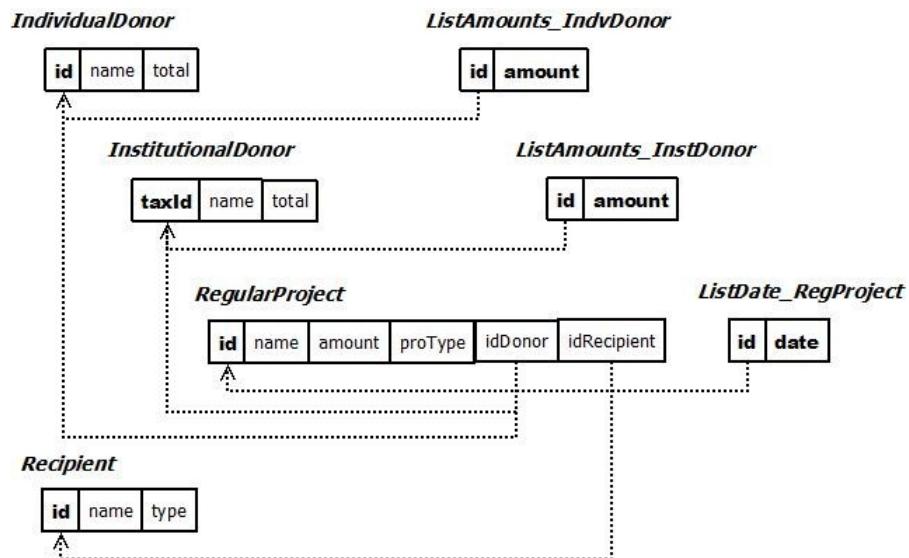**Recipient**

id	name	type

Fig. 12. Expected Schema Diagram for the Fifth Product

The sixth product realizes feature Donor, feature Construction project, and feature Recipient. There will be a foreign key from relation Project to relation Recipient to represent the association between these relations. The expected schema diagram is represented by Figure 14.
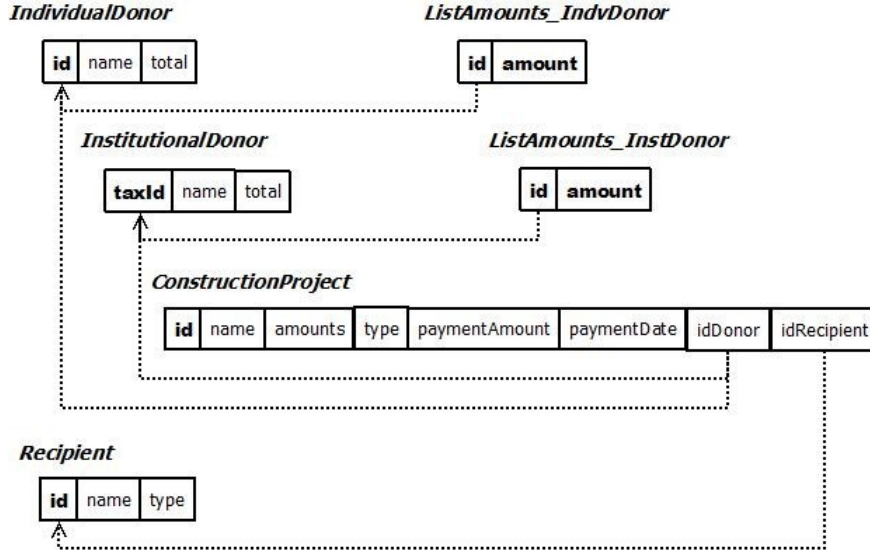

Fig. 13. Expected Schema Diagram for the Sixth Product

3.2   Case Study Implementation in ABS

This section will present the implementation of the case study in ABS. The case study explained in Section 1 is modeled in ABS with the following modules.

Module Project   The first module is module project. This module defines the model of project in ABS. There is an abstract data type declaration which declares the type of the project and constructs its possible value. The abstract data type declaration is followed by function definitions to initialize that abstract data type. This abstract data type is to represent the possible fields of project mentioned in Section 1; those are health, education, empowerment, and disaster relief. And then we define an interface called Project, with several abstract methods. This interface is implemented by the class Project. This class represents the relationship between class donor and class project, by having Donor object as one of its class parameters. Below is shows the ABS code of module Project.

```
1      module Project;
2
3      export *;
4      import Donor from Donor;
5
6      data ProjectType = Health | Education | Empowerment | DisasterRelief;
7
8      def ProjectType createHealthProject() = Health;
9      def ProjectType createEducationProject() = Education;
10     def ProjectType createEmpowermentProject() = Empowerment;
11     def ProjectType createDisasterReliefProject() = DisasterRelief;
12
13     interface Project {
14             Int getId();
15             Int getAmount();
16             Project getProject(Int id);
17             Unit editProject(String n, Int a, String d);
18     }
19
20     class Project([PK]Int id, String name, Int amount, ProjectType proType, Donor donor)
21     implements Project {
22
23             String date = "";
24
25             Int getId() { return id; }
26             Int getAmount() { return amount; }
27             Project getProject(Int id){ return this; }
28             Unit editProject(String n, Int a, String d) {
29                     name = n;
30                     amount = a;
31                     date = d;
32             }
33     }
```

Module Donor This module defines the model of donor in ABS. In this module
we declare an interface called Donor, with several abstract methods to represent
the behavior of Donor model. This interface is implemented by two classes. The
first class is the Individual Donor, and the second class is Institutional Donor.
Below is the ABS code of module Donor.

```abs
1        module Donor;
2        export Donor;
3        interface Donor {
4              Int getId();
5              String getName();
6              Int getTotal();      // to get the total amount of the donation
7              List<Int> getAmounts();   // to get the list of the donation
8              Unit donate(Int amt);   // to add the donation
9        }
10       class IndividualDonor([PK]Int id, String name) implements Donor {
11             Int total = 0;
12             List<Int> amounts = Nil;
13             Int getId() { return id; }
14             String getName() { return name; }
15             Int getTotal() { return total;}
16             List<Int> getAmounts() { return amounts; }
17             Unit donate(Int amt) {
18                   total = total + amt;
19                   amounts = Cons(amt, amounts);
20             }
21       }
22       class InstitutionalDonor([PK]Int taxId, String name) implements Donor {
23             Int total = 0;
24             List<Int> amounts = Nil;
25             Int getId() { return taxId; }
26             String getName() { return name; }
27             Int getTotal() { return total;}
28             List<Int> getAmounts() { return amounts; }
29             Unit donate(Int amt) {
30                   total = total + amt;
31                   amounts = Cons(amt, amounts);
32             }
33       }
```

Delta Modules The delta modules consist of DRegular, DConstuction, and DRecipient. Delta DRegular modifies class Project, removes the field date with String data type, and adds a new date field with parametric data type List<String>. Delta DConstruction modifies class Project, removes the field date with String data type, and add new field called payment which has Pair<Int,String>as the data type. Delta DRecipient adds new abstract data type in module Project, and defines the functions to initialize it. This abstract data type is to represent the type of the recipient, which are individual, institutional, and regional. This delta also adds new interface called Recipient and new class Recipient which implements Recipient interface. To represent the relationship between class Project and class Recipient, delta DRecipient modifies class Project by adding Recipient as new field. Below are the ABS codes of delta modules.

```
1      delta DRegular;
2      uses Project;
3
4      modifies class Project {
5              removes String date;
6              adds List<String> date = Nil;
7
8              modifies Unit editProject(String n, Int a, String d) {
9                      name = n;
10                     amount = a;
11                     date = Cons(d,date);
12              }
13      }
```

```
1      delta DConstruction;
2      uses Project;
3
4      modifies class Project {
5
6              removes String date;
7              adds Pair<Int, String> payment = Pair(0,"");
8
9              modifies Unit editProject(String n, Int a, String d) {
10                     name = n;
11                     Pair<Int,String> pay = Pair(a, d);
12              }
13
14      }
```

```
1       delta DRecipient;
2       uses Project;
3
4       adds data RecipientType = Individual | Institutional | Regional;
5
6       adds def RecipientType createIndvRec() = Individual;
7       adds def RecipientType createInstRec() = Institutional;
8       adds def RecipientType createRegRec() = Regional;
9
10      adds interface Recipient {
11          Int getId();
12          String getName();
13      }
14
15      adds class Recipient([PK]Int id, String name, RecipientType recType)
16      implements Recipient {
17
18          Int getId() { return id; }
19          String getName() { return name; }
20      }
21
22      modifies class Project {
23          adds Recipient recipient;
24      }
```

Feature Model The code below is the feature model in μTVL which represents the feature diagram shown in Figure 1.

```
1       root DonationReport {
2           group allof {
3               opt Recipient /*end of Recipient*/ ,
4               opt Donor /*end of Donor*/ ,
5               Project {
6                   group allof {
7                       opt Construction /*end of Construction*/ ,
8                       opt Regular /*end of Regular*/ ,
9                       opt Once /*end of Once*/
10                  } // end of Project group
11              } /*end of Project*/
12          } //end of DonationReport group
13      }
```

Product Line Configuration The product line defines features and the corresponding delta to realize the features. Delta DRegular will be implemented when feature Regular is chosen. Delta DConstruction will be implemented when feature Construction is chosen. And delta DRecipient will be implemented if feature Recipient is chosen.

```
1        productline Project;
2        features Once, Donor, Construction, Regular, Recipient;
3        delta DConstruction when Construction;
4        delta DRegular when Regular;
5        delta DRecipient when Recipient;
```

Products Declaration Below is the products declaration, as mentioned in the beginning of this section.

```
1        // base product
2        product Once (Once);
3
4        // regular project
5        product Regular (Regular);
6
7        // construction project
8        product Construction (Construction);
9
10       // base product with recipient
11       product OnceRecipient (Once, Recipient);
12
13       // regular project with recipient
14       product RegularRecipient (Regular, Recipient);
15
16       // construction project with recipient
17       product ConstructionRecipient (Construction, Recipient);
```

3.3   Implementation of Mapping Technique in the Case Study

The mapping technique, which is implemented in ABS backend, is implemented in the ABS code of the case study using -dbschema argument in the command line. Below are the results of the mapping technique implementation and the generated database schemas.

generateJava -dbschema -product=Once F:\DonationReport*.abs

This command will generate database schema for the first product, as shown in Figure 23.

```
a = LOAD 'data' USING BinStorage AS (user);
b = GROUP a BY user;
/* Now we are ready to loop */
c = FOREACH b GENERATE COUNT(a) AS cnt;
d = ORDER c BY cnt;
```

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
date                    VARCHAR(30)     DEFAULT '',
proType                 VARCHAR(30)     NOT NULL,
amount                  INT             NOT NULL,
genId_donor             INT             NOT NULL,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                   INT             DEFAULT 0,
taxId                   INT             NOT NULL,
name                    VARCHAR(30)     NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                   INT             DEFAULT 0,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                           INT             NOT NULL,
taxId_InstitutionalDonor        INT,
id_IndividualDonor              INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);


CREATE TABLE ListInstitutionalDonor (
amounts0                INT,
taxId_InstitutionalDonor        INT             NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0                INT,
id_IndividualDonor      INT             NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

Fig. 14. The Generated Database Schema of the First Product

generateJava -dbschema -product=Regular F:\DonationReport*.abs

This command will generate database schema for the second product, as shown in Figure 24.

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
proType                  VARCHAR(30)      NOT NULL,
amount                   INT              NOT NULL,
genId_donor              INT              NOT NULL,
name                     VARCHAR(30)      NOT NULL,
id                       INT              NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                    INT              DEFAULT 0,
taxId                    INT              NOT NULL,
name                     VARCHAR(30)      NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                    INT              DEFAULT 0,
name                     VARCHAR(30)      NOT NULL,
id                       INT              NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                            INT               NOT NULL,
taxId_InstitutionalDonor         INT,
id_IndividualDonor               INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);

CREATE TABLE ListProject (
date0                    VARCHAR(30),
id_Project               INT              NOT NULL,
PRIMARY KEY (date0, id_Project),
FOREIGN KEY (id_Project) REFERENCES Project(id));


CREATE TABLE ListInstitutionalDonor (
amounts0             INT,
taxId_InstitutionalDonor        INT             NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0             INT,
id_IndividualDonor      INT             NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

Fig. 15. The Generated Database Schema of the Second Product

generateJava -dbschema -product=Construction F:\DonationReport*.abs

This command will generate database schema for the third product, as shown in Figure 25.

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
proType                 VARCHAR(30)     NOT NULL,
amount                  INT             NOT NULL,
genId_donor             INT             NOT NULL,
paymentPair1            VARCHAR(30),
paymentPair0            INT,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                   INT             DEFAULT 0,|
taxId                   INT             NOT NULL,
name                    VARCHAR(30)     NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                   INT             DEFAULT 0,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                           INT             NOT NULL,
taxId_InstitutionalDonor        INT,
id_IndividualDonor              INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);


CREATE TABLE ListInstitutionalDonor (
amounts0                INT,
taxId_InstitutionalDonor        INT             NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0                INT,
id_IndividualDonor      INT             NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

Fig. 16. The Generated Database Schema of the Third Product

generateJava -dbschema -product=OnceRecipient F:\DonationReport*.abs

This command will generate database schema for the fourth product, as shown in Figure 26.

generateJava -dbschema -product=RegularRecipient F:\DonationReport*.abs

This command will generate database schema for the fifth product, as shown in Figure 27.

generateJava -dbschema -product=ConstructionRecipient F:\DonationReport*.abs

This command will generate database schema for the sixth product, as shown in Figure 28.

3.4   Evaluation

To ease the evaluation process, we will represent the generated database schema as database schema diagrams. The evaluation process is done using hypothetical evaluation, which compares the expected results with the generated results.

Figure 29 is the diagram schema of the generated database schema for the first product.

Figure 30 is the diagram schema of the generated database schema for the second product.

Figure 31 is the diagram schema of the generated database schema for the third product.

Figure 32 is the diagram schema of the generated database schema for the fourth product.

Figure 33 is the diagram schema of the generated database schema for the fifth product.

Figure 34 is the diagram schema of the generated database schema for the sixth product.

Based on the comparison between the expected diagram schema and the diagram schema of generated database schema for each product, the results of the mapping are appropriate with the expected results. Furthermore, we can say that the mapping results are even better than the expected results, because the

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
date                        VARCHAR(30)     DEFAULT '',
proType                     VARCHAR(30)     NOT NULL,
amount                      INT             NOT NULL,
genId_donor                 INT             NOT NULL,
name                        VARCHAR(30)     NOT NULL,
genId_recipient             INT,
id                          INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Recipient (
recType                     VARCHAR(30)     NOT NULL,
name                        VARCHAR(30)     NOT NULL,
id                          INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                       INT             DEFAULT 0,
taxId                       INT             NOT NULL,
name                        VARCHAR(30)     NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                       INT             DEFAULT 0,
name                        VARCHAR(30)     NOT NULL,
id                          INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                           INT             NOT NULL,
taxId_InstitutionalDonor        INT,
id_IndividualDonor              INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

CREATE TABLE Recipient_TYPE (
genId                           INT             NOT NULL,
id_Recipient                    INT,
PRIMARY KEY (genId),
FOREIGN KEY (id_Recipient) REFERENCES Recipient(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);

ALTER TABLE Project
FOREIGN KEY genId_recipient REFERENCES Recipient_TYPE(genId);


CREATE TABLE ListInstitutionalDonor (
amounts0                    INT,
taxId_InstitutionalDonor        INT             NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0                    INT,
id_IndividualDonor          INT             NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

Fig. 17. The Generated Database Schema of the Fourth Product

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
proType                 VARCHAR(30)     NOT NULL,
amount                  INT             NOT NULL,
genId_donor             INT             NOT NULL,
name                    VARCHAR(30)     NOT NULL,
genId_recipient         INT,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Recipient (
recType                 VARCHAR(30)     NOT NULL,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                   INT             DEFAULT 0,
taxId                   INT             NOT NULL,
name                    VARCHAR(30)     NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                   INT             DEFAULT 0,
name                    VARCHAR(30)     NOT NULL,
id                      INT             NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                           INT             NOT NULL,
taxId_InstitutionalDonor        INT,
id_IndividualDonor              INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

CREATE TABLE Recipient_TYPE (
genId                           INT             NOT NULL,
id_Recipient                    INT,
PRIMARY KEY (genId),
FOREIGN KEY (id_Recipient) REFERENCES Recipient(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);

ALTER TABLE Project
FOREIGN KEY genId_recipient REFERENCES Recipient_TYPE(genId);

CREATE TABLE ListProject (
date0                   VARCHAR(30),
id_Project              INT             NOT NULL,
PRIMARY KEY (date0, id_Project),
FOREIGN KEY (id_Project) REFERENCES Project(id));


CREATE TABLE ListInstitutionalDonor (
amounts0                INT,
taxId_InstitutionalDonor        INT             NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0                INT,
id_IndividualDonor      INT             NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

Fig. 18. The Generated Database Schema of the Fifth Product

```
CREATE SCHEMA NEWSCHEMA;

CREATE TABLE Project (
proType                  VARCHAR(30)      NOT NULL,
amount                   INT              NOT NULL,
genId_donor              INT              NOT NULL,
paymentPair1             VARCHAR(30),
paymentPair0             INT,
name                     VARCHAR(30)      NOT NULL,
genId_recipient          INT,
id                       INT              NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Recipient (
recType                  VARCHAR(30)      NOT NULL,
name                     VARCHAR(30)      NOT NULL,
id                       INT              NOT NULL,
PRIMARY KEY (id));

CREATE TABLE InstitutionalDonor (
total                    INT              DEFAULT 0,
taxId                    INT              NOT NULL,
name                     VARCHAR(30)      NOT NULL,
PRIMARY KEY (taxId));

CREATE TABLE IndividualDonor (
total                    INT              DEFAULT 0,
name                     VARCHAR(30)      NOT NULL,
id                       INT              NOT NULL,
PRIMARY KEY (id));

CREATE TABLE Donor_TYPE (
genId                         INT              NOT NULL,
taxId_InstitutionalDonor      INT,
id_IndividualDonor            INT,
PRIMARY KEY (genId),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));

CREATE TABLE Recipient_TYPE (
genId                         INT              NOT NULL,
id_Recipient                  INT,
PRIMARY KEY (genId),
FOREIGN KEY (id_Recipient) REFERENCES Recipient(id));

ALTER TABLE Project
FOREIGN KEY genId_donor REFERENCES Donor_TYPE(genId);

ALTER TABLE Project
FOREIGN KEY genId_recipient REFERENCES Recipient_TYPE(genId);


CREATE TABLE ListInstitutionalDonor (
amounts0                 INT,
taxId_InstitutionalDonor     INT              NOT NULL,
PRIMARY KEY (amounts0, taxId_InstitutionalDonor),
FOREIGN KEY (taxId_InstitutionalDonor) REFERENCES InstitutionalDonor(taxId));


CREATE TABLE ListIndividualDonor (
amounts0                 INT,
id_IndividualDonor       INT              NOT NULL,
PRIMARY KEY (amounts0, id_IndividualDonor),
FOREIGN KEY (id_IndividualDonor) REFERENCES IndividualDonor(id));
```

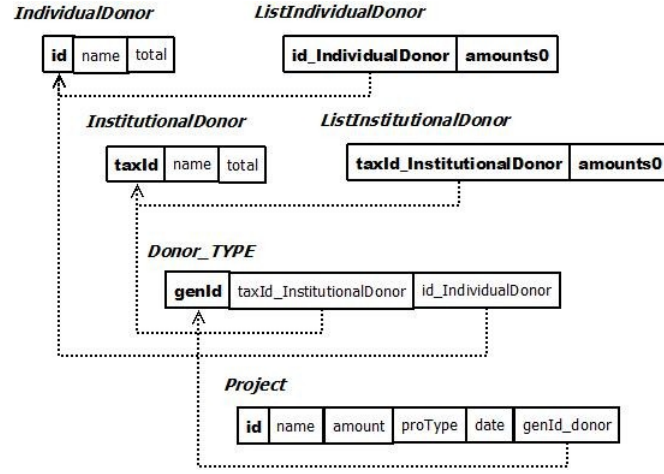**Fig. 19.** The Generated Database Schema of the Sixth Product

Fig. 20. The Diagram Schema of the Generated Database Schema for the First Product
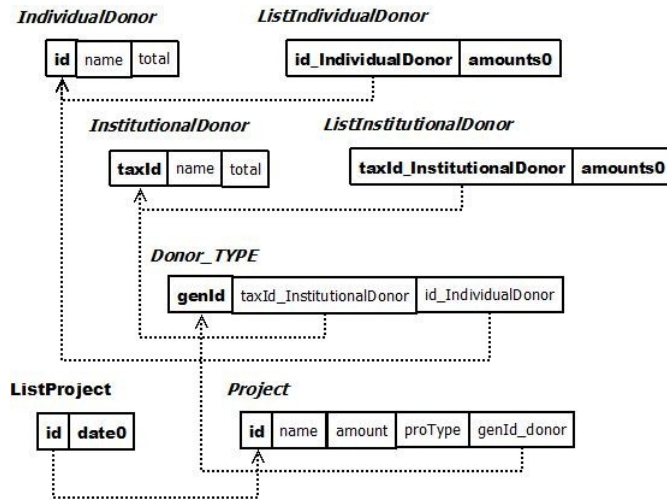

Fig. 21. The Diagram Schema of the Generated Database Schema for the Second Product
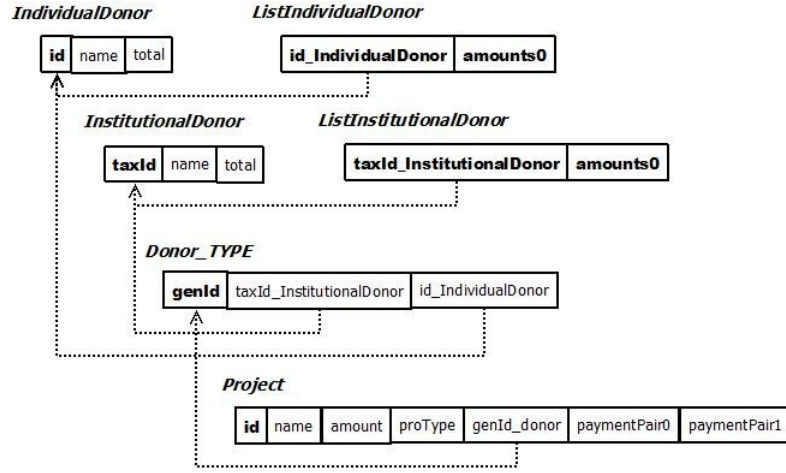
Fig. 22. The Diagram Schema of the Generated Database Schema for the Third Product
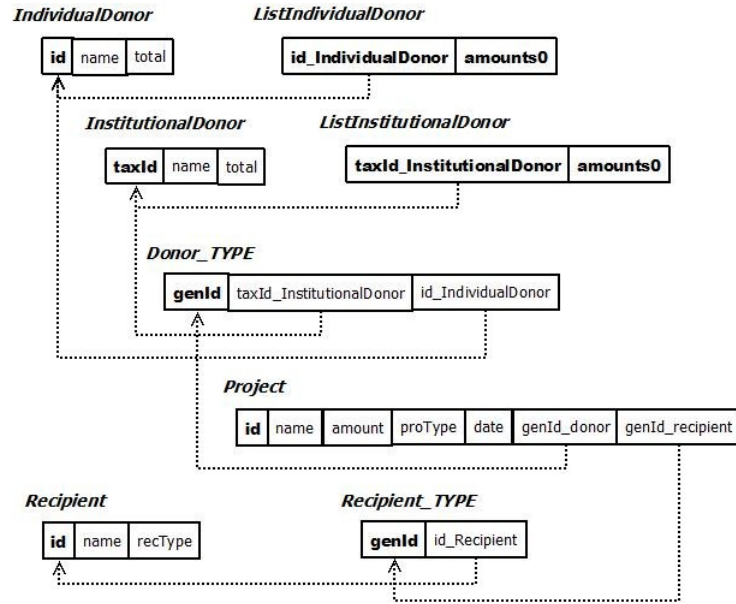

Fig. 23. The Diagram Schema of the Generated Database Schema for the Fourth Product
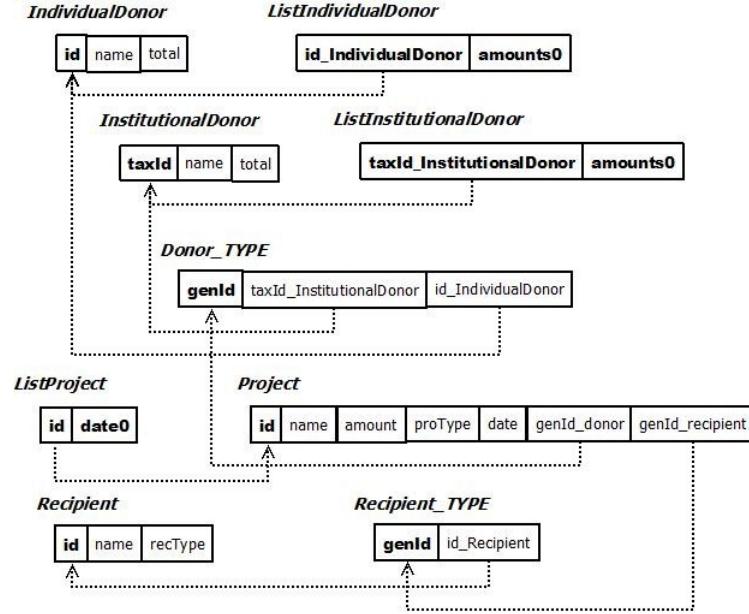
Fig. 24. The Diagram Schema of the Generated Database Schema for the Fifth Product
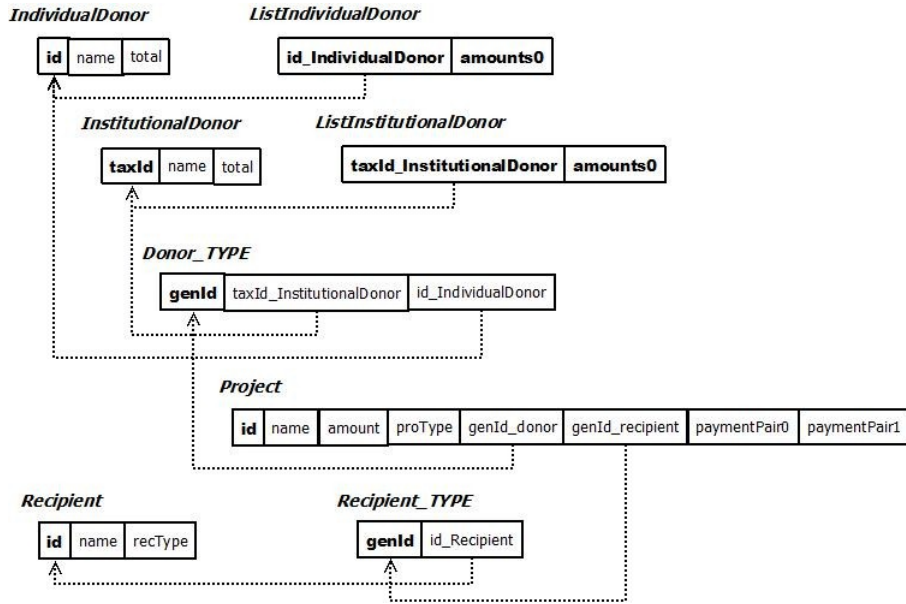

Fig. 25. The Diagram Schema of the Generated Database Schema for the Sixth Product

mapping technique is able to solve the issue of more than one reference in one attribute, by implementing intermediate table.

Moreover, the generated database schema is able to fulfill the relational database constraints. The domain constraint is fulfilled by generating a separate relation for multivalued attributes. The key constraint is fulfilled using annotations in ABS, so that every field or class parameter with [PK] annotations will be declared as primary key of the relation. The constraint on NULL is fulfilled by mapping Nil, Unit, EmptySet, and EmptyMap in ABS as NULL. The entity integrity constraints is fulfilled by declaring NOT NULL constraint for primary key attributes and attributes derived from class parameter. The referential integrity constraint is fulfilled by declaring foreign key, when two classes are related by association.

Besides that, the mapping technique is also able to solve the mismatch between ABS language and relational database concepts, especially the object type issue in ABS. This issue is solved by the mapping technique by generating intermediate table based on the interface type declared in ABS core modules.

By using this mapping technique, we can generate relational database schema directly from ABS model. Hence, it can ease the process of software development, especially software with databases. Moreover, delta-oriented programming implemented by ABS makes product realization in SPLE easier. In other words, this mapping technique can ease the process of product realization in SPLE for software with databases.

The case study used in this research is the simplification of real-life case faced by charity organizations. In reality, there are many variations of programs and activities of charity organizations. For this case study, this mapping technique is able to generate any relational database schemas corresponding to each product defined. Each product of this case study represents the variation of charity organizations' programs.

This mapping technique can be used to map ABS model into relational database schema with following limitations:

1. Class declaration in ABS model does not have Fut<T>as class parameter data type and/or field data type.
2. Class declaration on ABS model does not have any class parameter and/or field which has more than one parametric data type as a data type. For example: List<Pair<Int,String»

Any fields or class parameters that have data type as mentioned above will not be mapped into any attribute (column) in relational database schema generated by this mapping technique.

4   Conclusion and Future Work

This Section concludes the work and discusses about the future work.

4.1   Conclusion

Based on the goal of this research and the evaluation of case study in the previous section, we can conclude:

1. Based on the results of this research, delta modeling in ABS can influence the elements of relational database schema. The elements of relational database schema can be obtained from the core ABS, especially in the class declaration and interface declaration in every module. This base database schema, which is generated from the core ABS, can be affected by the implementation of delta modules. This case happens if the modification by the delta modules related to the class and the fields in a class.

2. The mapping between ABS modeling and relational database schema is possible to be formulated, by considering the elements and the constraints owned by both sides. The elements which have similarities for both sides can be mapped directly. For example, an attribute of a class is mapped into attribute of a relation. However, there are several mismatches between both sides because of the conceptual differences. This can be solved by implementing some technique so that both sides' constraints can be fulfilled. Using ABS annotations to declare primary key and generating intermediate table for object type are the solutions to the mismatches.

3. The mapping technique and product configuration in ABS can be integrated by implementing the mapping technique in ABS backend compiler. The usage of ANTLR and JastAdd by ABS compiler makes the mapping technique integration possible. The mapping technique is implemented in an aspect file, which will later be processed by JastAdd. The aspect file is used to modify the AST classes, which are generated by the JastAdd system from the AST constructed by ANTLR. Moreover, ABS compiler provides '-product=' argument which can be used to generate flattened core ABS by the application of delta modules. By adding '-dbschema' argument to the main program of ABS compiler, we can generate relational database schema from the flattened core ABS by delta modeling implementation, using command line.

4. The simulation of mapping technique and database schema generation is done using donation report case study. The case study is built based on the simplification of real-life cases faced by charity organizations. Moreover, the case study contains the mismatch issues mentioned before, such as multivalued attributes, object type, and referential integrity. Hence, we can say that the result of the case study represents the effectiveness of the mapping technique. The evaluation of the case study implementation in the previous section shows that the result of the database schema generation is better than the expected result defined beforehand. Hence, it is safe to say that the mapping technique is effective to generate simple relational database schema from ABS model.

4.2   Future Work

Basically this research is the first step of a bigger research that connects ABS to databases. Below is possible future work, which can be done to continue the process of connecting ABS to database.

1. The normalization of the generated database schema to fulfill the normal forms in relational database schema.
2. The implementation of the database operations: retrieval, deletion, and update. These operations will correspond to the method invocation in the main block of ABS. Moreover, this implementation relates to the ABS runtime.
3. The usage of DBMS for database implementation in ABS: if DBMS will be used in database implementation by ABS, then the connection between ABS and DBMS must be formulated. An application program, in this case is ABS, accesses the database by sending queries or request for data in the DBMS. A query typically causes some data to be retrieved. A transaction done in application program may cause some data to be updated and written in the database. This connection will make ABS as a running application program which is able to access the database through the DBMS.
4. Maintaining the evolvement of database connected to ABS because of the application requirement changes: requirement changes of application could lead to schema evolution where changes are needed to be applied to the schema. Moreover, this maintenance needs the consideration to preserve the previous data.