Hong, et al., Int J Comput Softw Eng 2017, 2: 123
https://doi.org/10.15344/2456-4451/2017/123

International Journal of
Computer & Software Engineering

Review Article Open Access

Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources

Tzung-Pei Hong[1,2*], Wei-Chou Chen[3], Wen-Yang Lin[1] and Shyue-Liang Wang[4]

[1]*Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan*
[2]*Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan*
[3]*Coretech Knowledge Inc., Taiwan*
[4]*Department of Information Management, National University of Kaohsiung, Taiwan*

Abstract

Data warehousing has recently been applied in different applications because it can provide relevant integrated data views according to the requirement of users. Along with the popularity of object-oriented concepts, a data warehouse can be implemented effectively and efficiently with objects for representing relationships among complex data. In this paper, we consider the implementation issues of view creation and deletion in an object-oriented data warehouse with multiple underlying data sources. The same class names may exist in different data sources, but the contents in these classes may be different. Meta-objects and a class-integration procedure are thus designed to help the execution of view creation and deletion. The implementation algorithms for view creation and deletion under multiple data sources are also presented.

Publication History:

Received: September 25, 2017
Accepted: December 14, 2017
Published: December 16, 2017

Keywords:

Class integration, Meta-object, Multiple data sources, Object-oriented data warehouse, View creation.

Introduction

Data warehousing is very popular in these years because it can provide a relevant integrated data source to a variety of users [1,4]. The data in the integrated data source come from one or several underlying databases and are well organized for user queries. A data collector is responsible for collecting the necessary information and then passes it to the data warehouse. In a data warehouse, view creation is very important to meeting the requirements of group users. Two kinds of views are commonly used. One is called materialized view, which copies data from underlying databases to a data warehouse according to given view definitions; the other one is called virtual view, which generates data from other materialized views.

Object-oriented representation could easily depict complex relations among objects. It can also show interwoven composition of attributes within objects. Besides, it has the advantage of inheritance, encapsulation, and polymorphism. Thus, the object-oriented concept has been embedded into different techniques, including databases and data warehouses. For example, Chen et al. introduced a data warehouse model suitable in object-oriented environments with a single data source [2,3]. That proposed model maintained the original structures in the source database to store the materialized views in the object-oriented data warehouse. Zhuge and Garica-Molina designed algorithms for view maintenance in a data warehouse of graph structure [12].

Many useful view maintenance techniques for object-oriented databases were proposed as well [7-9]. Trujillo et al. implemented the object-oriented modeling of data warehouses by the Unified Modelling Language [11]. Suri and Sharma embedded the object–oriented technology to some existing applications [10]. Pahwa and Chhabre introduced an approach to transfer a relational schema from into an object-oriented data warehouse [6].

In this paper, we extend the uncompressed data model to manage views derived from multiple-source environments. Meta-objects are presented and a class-integration procedure is designed to help the execution of view creation and deletion in an object-oriented data warehousing with multiple data sources.

Preliminary

Class is a basic group concept in object-oriented representation. A class may include some attributes and methods. The value in an attribute may be atomic or come from another class which has been defined. An instance can be declared based on a class and will own the attributes and methods in the class. If a class is a descendent of another class, then the former can inherit the attributes and methods from the latter.

After classes and instances are generated, views can then be defined to act as virtual classes for increasing the modeling and schema restructuring capability [5]. A view is usually defined by a query sentence. Objects satisfying the condition of a view are sent from the underlying databases to the data warehouse. The number of attributes in the view is equal to that given in the query sentence. A view in a data warehouse can be defined to retrieve the objects from more than one data source, with the relationships between these data sources being determined by the conditions in the query sentence. Two kinds of condition sentences may be used here. One is the independent condition sentence, in which variables can be retrieved in a single source database. The other is the dependent condition sentence, in which variables must be retrieved from more than one source database. Restated, when a condition sentence can be checked from a single source database, it is an independent condition sentence; otherwise, it is a dependent condition sentence. An object-oriented data warehouse can then be specified by the given classes, instances and view definitions.

Meta-Object and Class Integration

A meta-object is used in this paper to keep the class identifiers and

***Corresponding Author:** Prof. Tzung-Pei Hong, Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan; E-mail: tphong@nuk.edu.tw

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

instance identifiers used for a view definition of a data warehouse for increasing the processing efficiency of view management. It can be defined by a triple {*Meta-mv, mc, mi*}, where *mv* is the identifier of a view, *mc* is the set of classes used in *mv*, and *mi* is the set of instances kept in the data warehouse for *mv*. With the meta-object, an object-oriented data warehouse *W* can thus be formally defined as a quadruple {*C, V, I, M*}, where *C* is a set of classes, *V* is a set of view definitions, *I* is a set of instances generated according to *C* and *V*, and *M* is a set of meta-objects generated according to *V*.

Since the paper handles a data warehouse formed from multiple databases, the classes and instances may come from different databases. Given a class name from an underlying database in a data warehouse, the class name may also exist in some other underlying databases or may have been used in other views. A procedure called the Class-Integration Procedure is then designed to handle the integration of the classes with the same names in the multiple data sources. It uses the meta-objects to speed up the integration process. The procedure is stated as follows.

The Class-Integration Procedure

Input: A data warehouse *W(C, V, I, M)* and a class *b.c*, where *b* is a source databaseand *c* is a class in the database *b*.

Output: A revised data warehouse *W'(C', V, I', M)* which integrates the class *b.c* withthe other classes in *W*.

Step 1. Search the meta-objects *M* for the class *b.c*. If b.c can be found in *M* (meaning it is used by some views in *V*), set *W'* = *W* and exit the procedure; otherwise, do Step 2.

Step 2. Send the request to the data collector to retrieve the class definition of *b.c*.

Step 3. Receive the class definition of *b.c* from the data collector. If the class name *c* (without b) exists in *C* of the data warehouse *W* (meaning the classes of the same name *c* in other source databases are used by some views in *V*), do the next step, otherwise, create the class *c* in *C* of the data warehouse *W* and exit the procedure.

Step 4. Check whether the set of attributes of the class *b.c.* is a subset of the class *c* in *C*, If it is, exit the procedure, otherwise, do the next step.

Step 5. Modify the class c in C to include the attributes in b.c. That is,
New attributes of c in C = (old attributes of c in C) ∪(attributes of b.c)
New methods of c in C = (old methods of c in C) ∪ (methods of b.c).

Step 6. Modify all instances in I inheriting from the class c according to the new

After Step 6, the structure of the class *c* in the underlying database *b* has been integrated with the class *c* in *C* of the data warehouse *W*. The class *c* can thus be correctly used in the warehouse *W*.

The Algorithm of View Creation under Multiple Data Sources

Based on the discussion above, a view-creation algorithm under multiple data sources is designed here. The select-from-where syntax for a new warehouse view (*WV*) is stated as follows:

Create Warehouse View *WV(wva1, wva2, …, wvan)* as
Select $b_{s_1}.a_1, b_{s_2}.a_2, \ldots, b_{s_n}.a_n$
From $b_{f_1}.c_1, b_{f_2}.c_2, \ldots, b_{f_k}.c_k$
Where *w1, w2, …, wm*

Here, wva_i represents the *i*-th attribute in the view *W*, $b_{s_1}.a_1$ represents the *i*-th attribute which comes from a class in the source database b_{s_i} (if the attributes and the classes of the attributes exist in all the source databases of the view, the parameter b_{s_i} can be omitted), $b_{f_i}.c_i$ denotes the *i*-th class from the source database b_{f_i} and w_i denotes the i-th condition. The implementation algorithm for processing the above statement is proposed as follows.

The view-creation algorithm

Input: A data warehouse *W(C, V, I, M)* with a view-creation statement for creating a new view *WV*.

Output: A revised data warehouse *W'(C', V', I', M')* after *WV* is created.

Step 1: For every class $b_{f_i}.c_i$ in *WV*, do the class-integration procedure, which is used to integrate the class $b_{f_i}.c_i$ in the class set *C*.

Step 2: Create a new meta-object with its name as Meta-*WV* in *M* of the warehouse *W.*

Step 3: Set mc in the meta-object of Meta-WV as all the classes ($b_{f_i}.c_i$) in WV.

Step 4: Collect all the source databases existing in *WV*, denote them as *A*.

Step 5: For every source database b_f in *A*, collect all the attributes, classes and independent conditions for b_f to form the following query statement Q_{bf}:

Select $b_f.a_{f1}, b_f.a_{f2}, \ldots, b_f.a_{fi}$
From $b_f.c_{f1}, b_f.c_{f2}, \ldots, b_f.c_{ft}$
Where $w_1, w_2, \ldots, w_l$,

In the above statement, 1≤ i ≤n, 1 ≤ t ≤ k, 1 ≤ l ≤ m, and the select part, from part and where part are respectively the subsets of the corresponding parts in WV.

Step 6: For every query statement Q_{bf}, do the following:

Step 6(1): Initially set the counter *m* = 1, where the counter *m* is used to count the looping number.

Step 6(2): Read a_m from the select part of the query statement.

Step 6(3): Find all the attribute names in a_m, whose types are classes; denote them as *B*.

Step 6(4): For every element in *B*, do the following substeps:

Step 6(4a): Find its class *cid* and do the *class-integration* procedure to integrate the class *cid* with the classed *C* in *W*.

Step 6(4b): Add it into the attribute *mc* of the meta-object Meta-*WV.*

Step 6(4c): Form the following query statement Q^m_{bf} to retrieve the instances desired:

Select tid
From b_f.cid
Where $w_1, w_2, \ldots, w_l$
where each w_j (*j* = 1 to *l*) contains the attribute name of class *cid.*

Step 6(5): Set m = m + 1.

Step 6(6): If m is less than the number of items in the select part of the query statement (Q_{bf}), go to Step 6(2).

Step 7: Send all of the query statements formed in Steps 5 and 6 to the data collector.

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

Step 8: Get from the data collector the instance identifiers (*tid's*), which satisfy thequery statements.

Step 9: Find the set of instance identifiers which are not currently in *I* of the warehouse *W*. Denote it as *D*.

Step 10: Request the data collector to get the contents of the instances in *D*.

Step 11: Get the instances from the data collector. Denote them as *P*.

Step 12: Check whether the instances in *P* satisfy all the dependent conditions in *WV*. Denote the instances desired as *G*.

Step 13: Add the instances set *G* into *I* of the warehouse *W*.

Step 14: Find all the instances in *I* which satisfy the query statement in *WV* and find all their referring instances, and add their instance identifiers (with the source name b_j) into the attribute *mi* of the meta-object Meta-*WV*.

Step 15: Add *WV* to *V* in *W*.

After Step 15, the data warehouse will contain all the desired instances, the new view definition *WV*, and the new meta-object for *WV*.

An Example for View Creation

An example is given below to demonstrate the above view-creation algorithm. Assume a data warehouse is formed from two underlying object-oriented data sources shown in Figures 1 and 2, respectively.

The two databases have the same class names, but lightly different attributes and methods in the two classes *StudInfo* and *Name*. Assume two instances are created by referring to the class *Dept* in DS_1. One is called *CS* with attribute values (001, Computer Science) and the other is called *IM* with attribute values (002, Information Management). Similarly, assume two instances *A1* and *B1* respectively with attribute values (001, CS, 1) and (102, IM, 2) are created by referring to the class *Classes*, two instances *WCC* and *TPH* with attribute values (Chen, Wei, Chou) and (Hong, Tzung, Pei) respectively are created by referring to the class *Name*, and two instances *ST01* and *ST02* respectively with attribute values (863201, WCC, A1) and (853001, TPH, B1) are created by referring to the class *StudInfo* in *DS1*. For *DS2*, assume three instances are created by referring to the class *Dept* in *DS2*. One is called *IE* with attribute values (101, Industrial Engineering), another is called *BM* with attribute values (102, Business Management) and

```
Class StudInfo {                         Class Dept {
      StudID        char(10),                  DeptID        char(3),
      StudName      Name,                      DeptName      char(40)
      StudClassClasses,                  }
      Counter()     int
}                                        Class Classes {
                                               ClassID       char(5),
Class Name {                                   DeptOf        Dept,
      First         char(20),                  Grade         int,
      Middle        char(20),                  Counter()     int
      Last          char(20)             }
}
```

Figure 1. The classes in the first data source (DS_1).

```
Class StudInfo {                         Class Name {
      StudID        char(10),                  First         char(20),
      StudName      Name,                      Middle        char(20),
      StudClass     Classes,                   Last          char(20),
      StudPic       Image,                     Nickname           char(20),
      Counter()     int,                 }
      ImageView()   Image
}                                        Class Classes {
                                               ClassID       char(5),
Class Dept {                                   DeptOf        Dept,
      DeptID        char(3),                   Grade         int,
      DeptName      char(40)                   Counter()     int
}                                        }
```

Figure 2. The classes in the second data source (DS_2).

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

the other is called *IM* with attribute values (002, Information Management). Similarly, assume three instances *B1*, *C1* and *D1* with attribute values (102, IM, 2), (101, IE, 1) and (103, BM, 3) respectively are created by referring to the class *Classes,* three instances *WCY, WTT* and *TMW* with attribute values (Wang, Chen, Yang, Simon) , (Wang, Tzi, Ting, Julia) and (Tsai, Ming, Wen, Joe) respectively are created by referring to the class *Name*, three instances *ST03*, *ST04* and *ST05* with attribute values (873201, WCY, C1, *image*), (873204, WTT, B1, *image*) and (853202, TMW, D1, *image*) respectively are created by referring to the class *StudInfo*. Assume two view definitions, *FreshMan* and *BothClassList* are given in Figure 3.

Two meta-objects, Meta-FreshMan and Meta-BothClassList, are created in the data warehouse. For the meta-object Meta-FreshMan, Meta-*mv* = Meta-FreshMan, *mc* = (DS1.StudInfo, DS1.Name, DS1.Classes, DS2.StudInfo, DS2.Name, DS2.Classes), and *mi* = (DS1.ST01, DS1.WCC, DS1.A1, DS2.ST04, DS1.WCY, DS1.C1). The above meta-objects are represented by a graph as shown in Figure 4.

Only eight instances, including *ST01, ST04, WCC, WTT, A1, B1, C1* and *IM* satisfy the conditions of the view definitions. These eight instances are thus sent from the source database to the warehouse and are thus reformatted and saved in the object-oriented data warehouse. Also assume the following new view in Figure 5 is to be defined in the data warehouse:

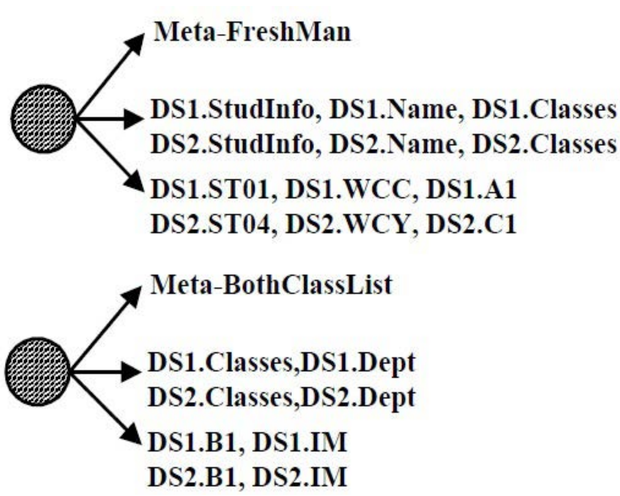

Figure 4. A graphical representation of the meta-objects in the example.

```
View FreshMan (DeptName char(40), FirstName char(20), LastName char(20) ) as
{
Select
    StudClass.ClassID,
    StudName.First,
    StudName.Last,
From DS1.StudInfo, DS2.StudInfo
Where DS1.StudClass.Grade = 1, DS2.StudClass.Grade = 1;
}

View BothClassList (DeptName char(40), ClassID char(5) ) as {
Select
    DeptOf.DeptName,
    ClassID
From DS1.Classes, DS2.Classes,
Where DS1.Dept.DeptName = DS2.Dept.DeptName;
}
```

Figure 3. Two view definitions in the example

```
View SecondStud ( StudID char(10), ClassID char(5), DeptName char(40))
as {
Select
  StudClass.DeptOf.DeptName
  StudClass.ClassID,
  StudID,
From DS1.StudInfo, DS2.StudInfo
Where DS1.StudClass.Grade = 2, DS1.StudClass.Grade = DS1.StudClass.Grade
}
```

Figure 5. A new view definition

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

The view-creation algorithm processes the new view definition as follows.

Step 1. Since the classes *DS1.StudInfo* and *DS2.Studinfo* in the new view definition have been selected by the previous views in *W*, the algorithm executes Step 2.

Step 2. Create a new meta-object Meta-SecondStud in W.

Step 3. The attribute mc in the meta-object *Meta-SecondStud* is set as DS1.StudInfo and DS2.StudInfo.

Step 4. Since the two source databases DS1 and DS2 are used in the new view, A is *{DS1, DS2}*.

Step 5. Form the following query statements:

Query Q_{DS1} for DS1
Select StudClass.DeptOf.DeptName, StudClass.ClassID, StudID
From StudInfo
Where StudClass.Grade = 2

Query Q_{DS2} for DS2
Select StudClass.DeptOf.DeptName, StudClass.ClassID, StudID
From StudInfo

Note that in this step, the condition DS1.Dept.DeptName= DS2. Dept.DeptName is not put in the above queries since it is not an independent condition.

Step 6. For the query statement QDS1, do the following substeps:

Step 6(1). Set the counter m = 1.

Step 6(2). Read a_1 = *StudClass.DeptOf.DeptName*.

Step 6(3). Since in a1, the attribute names *StudClass.DeptOf* and *StudClass* are of type class, B is thus *{StudClass.DeptOf, StudClass}*.

Step 6(4a). Since the class of *DeptOf* is *Dept* and the class of *StudClass* is *Classes*, both of them are thus checked by the *Class-Integration* procedure. Since both of them have been used in the data warehouse *W*, they are not processed.

Step 6(4b). Add the items DS1.Dept and DS1.Class into the attribute mc in the meta-object *Meta-SecondStud*.

Step 6(4c). Form the following two query statements as follows:

Query Q^{11}_{DS1}:
Select tid
From *DS1.Classes*
Where Grade = 2;

Query Q^{12}_{DS1}:
Select tid
From *DS1.Dept*

Step 6(5). Set *m = m + 1; m* is thus 2.

Step 6(6). Repeat Steps 6-2 to 6-4 to find the other query statements as follows:

Query Q^{21}_{DS1}:
Select tid
From *DS1.Classes*
Where Grade = 2;
Query Q^{11}_{DS2}:
Select tid
From *DS2.Classes.*
Query Q^{12}_{DS2}:
Select tid
From *DS2.Dept*
Query Q^{21}_{DS2}:
Select tid
From *DS2.Classes*

Also, the items DS2.Dept and DS2.Class are added into the attribute mc in the meta-object *Meta-SecondStud.*

Step 7. Send the query statements Q_{DS1}, Q_{DS2}, Q^{11}_{DS1},Q^{12}_{DS1}, and Q^{11}_{DS2} (The query statements Q^{21}_{DS1} and Q^{21}_{DS2} are the same as Q^{11}_{DS1} and Q^{11}_{DS2} and are thus removed) to the data collector.

Step 8. In the source database *DS1*, the instance *ST02* satisfies Q_{DS1}, instance B1 satisfies Q^{11}_{DS1} and instances *CS* and *IM* satisfy Q^{12}_{DS1} In the source database *DS2*, the instances *ST03*, *ST04* and *ST05* satisfy Q_{DS2}, instances *B1, C1* and *D1* satisfy Q^{11}_{DS2} instances CS, IE and BM satisfy Q^{12}_{DS2} The eleven instance identifiers {ST02, ST03, ST04, ST05, B1, C1, D1, CS, IM, IE, BM} (Since the instances B1 in DS1 and B1 in DS2 inherit from the same class *Classes*, only one instance identifier is sent back from the data collector), are thus sent back to the warehouse *W.*

Step 9. Check the eleven instance identifiers in *W. D* is thus {ST02, ST03, ST05, B1, C1, D1, IE, BM}.

Step 10. Request the data collector to get the instances in *D*.

Step 11. The instances *ST02, ST03, ST05, B1, C1, D1, IE* and *BM* with their contents are thus sent back to the warehouse *W*. Denote them as *P*.

Step 12. Since only the instances *ST02* and *ST03* satisfy the dependent condition sentence DS1.Dept.DeptName = DS2.Dept.DeptName in *WV,* G is thus *{ST02, ST03}*.

Step 13. Add G to I of W.

Step 14. Since the instances *ST02, B1* and *IM* in *DS1* and the instances *ST03, B1* and *IM* in *DS2* are retrieved by view SecondStud, the items DS1.ST02, DS1.B1, DS1.IM, DS2.ST03, DS2.B1 and DS2.IM are thus added into the attribute *mi* of meta-object *Meta-SecondStud.*

Step 15. The view definition *SecondStud* is added to the warehouse *W.*

After the view-creation procedure is executed, all of the instances used in the view *SecondStud* are stored in the data warehouse. Also, the meta-object *Meta-SecondStud* is automatically created for later management and maintenance of the view *SecondStud*. The graphical representation of the warehouse after the view *SecondStud* has been inserted is shown in Figure 6, where a circle represents a class, a rectangle represents an atomic type, and an ellipse represents a set of attributes.

The Algorithm of View Deletion under Multiple Data Sources

When a view is no longer needed, it may be deleted from the object-oriented data warehouse. The statement for deleting a warehouse view is stated as follows:

Delete Warehouse View *WV.*

The implementation algorithm for processing it is stated below.

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

The view-deletion algorithm:

Input: A data warehouse *W(C, V, I, M)* with a view-deletion statement for deletingview *WV*.

Output: A revised data warehouse *W'(C', V', I' , M')* after *WV* is deleted.

Step 1. Check whether the view *WV* has been in *W*. If it has, execution the next step; otherwise, set *W'* = *W* and stop the execution.

Step 2. Initially set the counter $j = 1$, where the counter j is used to count the looping number.

Step 3. Read the *j-th* item mc_j (representing b_f.*cid*, the class cid of the source database b_f) from the *mc* part in *Meta-WV.*

Step 4. Check whether the class *cid* is used by the other meta-objects in *M*. If it is, then do nothing; otherwise, remove the class *cid* and all the instances inheriting from the class *cid.*

Step 5. Set $j = j + 1$.

Step 6. If $j < |mc|$, go to Step 3; otherwise, do the next step.

Step 7. Set the counter $j = 1$, where the counter j is used to count the looping number.

Step 8. Read the *j-th* item mi_j (representing b_f.*tid*, the instance $_{tid}$ of the source database b_f) from the *mi* part in *Meta-WV*.

Step 9. Check whether the instance *tid* is used by the other meta-objects in *M*. If it is, then do nothing; otherwise, remove the instance *tid* from *I.*

Step 10. Set $j = j + 1$.

Step 11. If j < |mi|, go to Step 8; otherwise, do the next step.

Step 12. Remove *WV* from *V* and remove the meta-object Meta-*WV* from M.

After Step 12, the view definition *WV,* the meta-object *Meta-WV,* and all the unused classes and instances can be removed from the data warehouse. An example is given below to demonstrate the view-deletion algorithm.

An Example for View Deletion

Continuing the above example, assume the following statement is given to delete the view *FreshMan* in the data warehouse:

Delete Warehouse View FreshMan.

The view-deletion algorithm processes the statement as follows.

Step 1. Since the view *FreshMan* has existed in the data warehouse W, the algorithm executes Step 2.

Step 2. Set the counter $j = 1$.

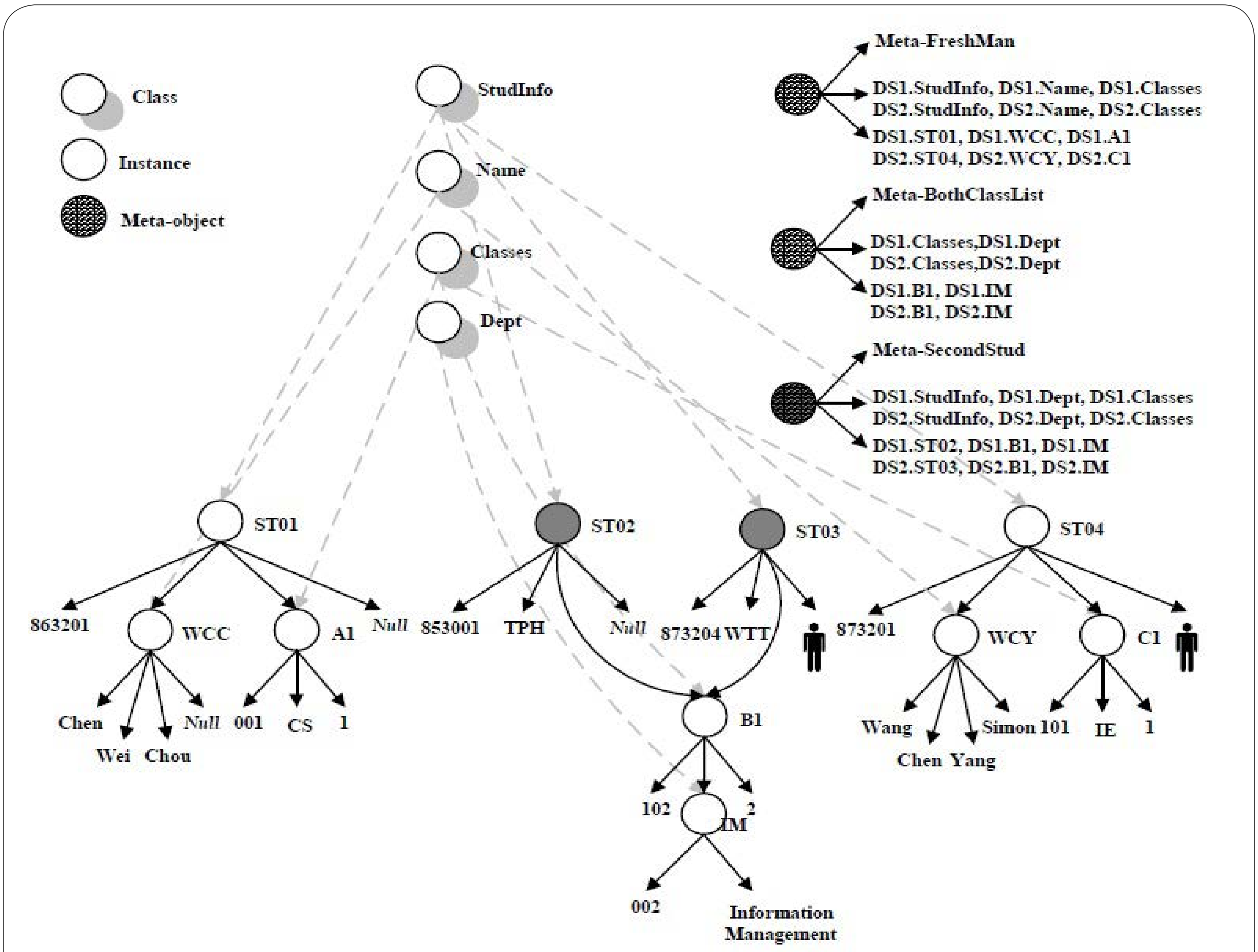

Figure 6. The graphical representation of the warehouse after the view has been inserted.

Citation: Hong TP, Chen WC, Lin WY, Wang SL (2017) Implementation of View Creation and Deletion with Class Integration under Multiple Data Sources. Int J Comput Softw Eng 2: 123. doi: https://doi.org/10.15344/2456-4451/2017/123

Step 3. Read the first class *DS1.StudInfo* from the *mc* part of the meta-object *Meta-FreshMan*.

Step 4. Since the class *StudInfo* is used by the view *SecondStud*, all of the instances of the class *StudInfo* are kept.

Step 5. Set $j = j + 1$. j is thus 2.

Step 6. Repeat Steps 3 to 5 for the remaining five classes *DS1.Name*, *DS1.Classes*, *DS2.StudInfo, DS2.Name* and *DS2.Classes*. Since the class *Name* is not referred to by views in the data warehouse *W* except by view *FreshMan*, the class *Name* and its instances *WCC* and *WCY* are thus removed from *C* and *I* of the warehouse *W*.

Step 7. Set the counter $j = 1$ again.

Step 8. Read the first instance *DS1.ST01* from the mi part of the meta-object *Meta-FreshMan.*

Step 9. Since the instance *ST01* is not referred to by other views, the instances *ST01* is thus removed from *I*.

Step 10. Set $j = j + 1$. j is thus 2

Step 11. Repeat Steps 8 to 10 for the remaining five instances *DS1.WCC, DS1.A1, DS2.ST04, DS2.WCY* and *DS2.C1*. Since the instances *ST04, A1* and *C1* are not referred to by views in the warehouse W except by view *FreshMan* (the instances *WCC* and *WCY* were removed in Step 6), they are thus removed from I.

Step 12. The view definition *FreshMan* and the meta-object *Meta-FreshMan* are removed from *V* and *M*.

After Step 12, the view FreshMan and its meta-object are removed from the data warehouse. Moreover, the instances *ST01, ST04, A1, C1* and all the instances inheriting from the class Name are deleted. A graphical representation of the warehouse after the view *FreshMan* has been deleted is shown in Figure 7.

Conclusion

In this paper, we have extended our previous data model from a single source to multiple-source environments. The meta-objects have been presented to make the creation and deletion of views in an object-oriented data warehousing easy and efficient. A class-integration procedure has also been designed to handle the integration of the classes with the same names in the multiple data sources. Moreover, two implementation algorithms for view creation and view deletion have been proposed. In the future, we will attempt to use ontology to flexibly find out similar classes, instead of the classes with the same names, in class integration from multiple data sources.

Conflict of Interest

No authors have a conflict of interest or any financial tie to disclose.

References

1. Chaudhuri S, Dayal U (1997) An overview of data warehousing and OLAP technology. ACM SIGMOD Record 21: 65-74.

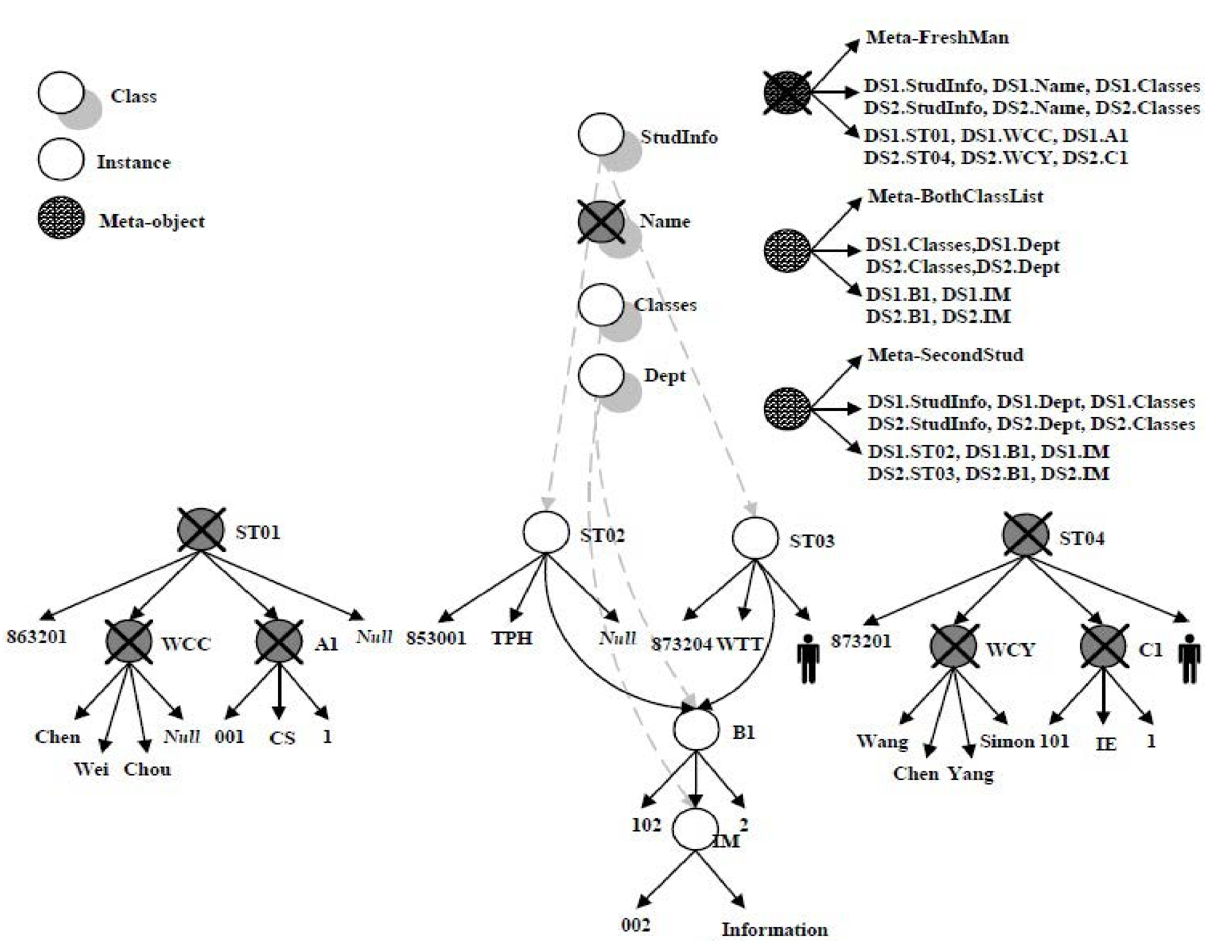

Figure 7. The graphical representation of the warehouse after the view has been deleted.

2. Chen WC, Hong TP, Lin WY (1998) View maintenance in an object-oriented data warehouse. The Fourth International Conference on Computer Science and Informatics, USA, pp. 353-356
3. Chen WC, Lin WY, Hong TP (1998) Object-oriented data warehousing and its incremental view maintenance. The Ninth Workshop on Object-Oriented Technology and Applications, Taiwan, pp. 139-144
4. Inmon WH, Kelley C (1993) Rdb/VMS: Developing the Data Warehouse. QED Publishing Group, Boston, Massachusetts
5. Kim W (1995) Modern Database Systems, ACM Press, USA
6. Pahwa P, Chhabra R (2014) An object oriented data warehouse design. International Journal of Soft Computing and Engineering 4: 5-7
7. Ra YG, Rundensteiner EA (1997) A transparent schema-evolution system based on object-oriented view technology. IEEE Transaction on Knowledge and Data Engineering 9: 600-624
8. Rundensteiner EA (1992) A methodology for supporting multiple views in object-oriented databases. The 18th International Conference on Very Large Data Bases, Vancouver, Canada, pp. 187-198
9. Scholl MH, Laasch C, Tresch M (1991) Updatable views in object-oriented databases. The Second International Conference on Deductive and Object-Oriented Databases, Munich, Germany, pp. 189-207
10. Suri P, Sharma M (2011) The succession of data warehouse using object oriented approach. International Journal of Engineering Science and Technology 3: 1153-1158
11. Trujillo J, Palomar M, Gomez J, Song IY (2001) Designing data warehouses with OO conceptual models. IEEE Computer 34: 66-75
12. Zhuge Y, Garcia-Molina H (1998) Graph structured views and their incremental maintenance. The 14th International Conference on Data Engineering, Orlando, USA, pp. 116-125