三個物件導向資料倉儲之漸進式維護演算法

Three Incremental Maintenance Algorithms on Object-Oriented Data Warehousing

陳威州* 洪宗貝** 林文揚**

Wei-Chou Chen **Tzung-Pei Hong** **Wen-Yang Lin**

(Received July 22, 1999; Revised Mar. 6, 2000; Accepted Mar. 28, 2000)

摘要

資料倉儲的概念主要是由數個獨立的資料庫中，收集有效資訊，先進行初步整合後，放置於資料中心，並提供企業線上決策支援分析作業之使用。近來，有關資料倉儲的相關研究被相當多的學者討論，但多是在關聯性資料模型的環境下進行研究。而本篇論文主要是介紹物件導向資料倉儲的概念及導向，並針對非壓縮資料模型提出三個漸進式維護演算法。最後，對所提出的演算法進行時間複雜度分析以顯示所提出演算法的效率及可行性。

關鍵詞：一致性、資料倉儲、物件導向資料庫、例子、漸進式維護

Abstract

A data warehouse is an information provider that collects necessary data from individual source databases to support the analytical processing of decision-support functions. In the past, research on data warehouses primarily focused on relational data models. In this paper, the concept of object-oriented data warehousing is introduced, and three algorithms, including instance-insertion, instance-deletion and instance-modification, are proposed to maintain the consistency between the data warehouse and the source databases. Time complexity is also analyzed to show their performance.

Keywords: consistency, data warehousing, object-oriented database, instance, incremental maintenance

1. INTRODUCTION

The concept of data warehousing was first proposed by Inmon (Inmon and Kelley, 1993). A data warehouse contains information that is collected from multiple, individual data sources and integrated into a common repository for efficient query and analysis. When the data sources are distributed over several locations, a data warehouse has the responsibil ity to collect the necessary data and save it in appropriate forms. Figure 1 shows the architecture of a typical data warehousing system.

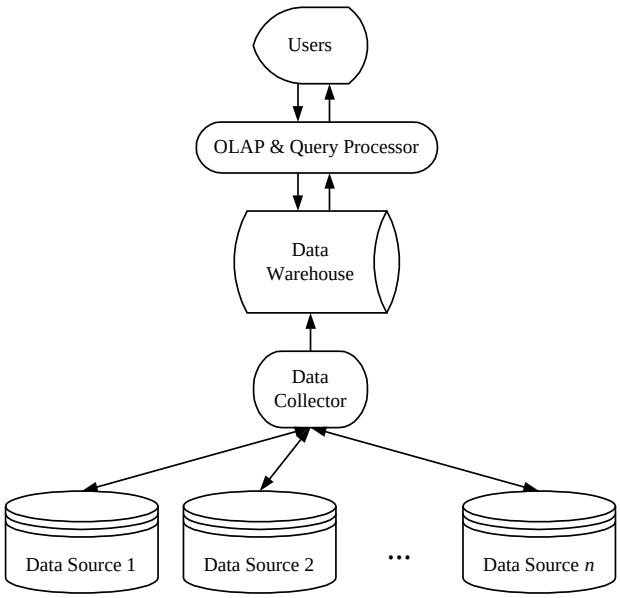

Figure 1 Architecture of a typical data warehousing system

In Figure 1, there are three major components in a data warehousing system: the *data collector*, the *data warehouse*, and the *OLAP and query processor*. The data collector is responsible for collecting necessary information and transaction messages from several individual data sources through communication networks with respect to the requirements of end users and the views defined in the data warehouse. The *data warehouse* receives data from the data collector, then filters it and stores it in its own database. The *OLAP and query processor* provides all necessary information for user queries and OLAP requirements.

Recently, research on the maintenance and performance of a data warehouse has been proposed to help on-line analytical processing (OLAP). Some research topics are maintenance (Huyn, 1996; Huyn, 1997; Mumick et al., 1997; Quass et al., 1996; Quass and Widom, 1997; Yang and Widom, 1998), consistency (Chen et al., 1998; Kawaguchi et al., 1997; Zhuge and Garica-Molina, 1997; Zhuge et al., 1997; Zhuge et al., 1998; Zhuge et al., 1998), query pr ocessing (Chen et al., 1999; Gupta et al., 1995; O'Neil and Quass, 1997) and among others (Gupta, 1997; Gupta et al., 1997; Harinarayan et al., 1996; Labio and Garcia-Molina, 1996; Labio et al., 1997). In the past decades, the relational data model has gained much attention and most research on data warehousing has focused primarily on this model. Recently, object-oriented data models have grown rapidly and have been widely adopted in the fields of database, artificial intelligence, software engineering and geographic information systems. An object-oriented database stores data in the form of classes and instances, and supports encapsulation and inheritance. Each instance in an object-oriented database inherits the characteristics of its preceding object (called its class) from which its structure is defined.

Applications of object-oriented database systems, such as CAD/CAM systems and GIS (Geographic Information Systems), may require a data warehouse to improve the efficiency of queries for decision support, especially when the databases are distributed over several places (陳威州等, 1999). The object-oriented characteristics, such as the inheritance structure and complex referential relationship, make the techniques of data warehousing using the relational data model not completely suitable for the object-oriented environment. Appropriate extension or modification of relational data warehousing is thus needed.

In this paper, the concept of object-oriented data warehousing is introduced and three algorithms, including instance-insertion, instance-deletion and instance-modification, are proposed to maintain the consistency between the source databases and the data warehouse. The properties of object models, including inheritance and encapsulation, are appropriately considered in the proposed maintenance algorithms. Only the necessary inherent relations between classes are kept and used to build instances in the data-warehouse (Chen, 1999). Part of the data and relationships of the classes are thus replicated in the data warehouse to archive this purpose and improve the query and maintenance performance.

The remainder of this paper is organized as follows. The concept of object-oriented data warehousing is introduced in Section 2. Formal definitions of the concepts related to this paper are given in Section 3. Three algorithms for consistency between the object-oriented data warehouse and the source databases are proposed respectively in Section 4 to 6. Examples are also given there

to illustrate the proposed algorithms. The time complexity of these algorithms is analyzed in Section 7. Finally, conclusions are given in Section 8.

2. OBJECT-ORIENTED DATA WAREHOUSING

An object-oriented database is a database that stores large numbers of objects, including classes and instances, with the functions of encapsulation, inheritance and polymorphism. Every instance in an object-oriented database inherits its characteristics from a superior object, called a *class*, which defines the basic structure of objects with common properties. The roles of classes and instances in an object-oriented database are similar to the roles of schema and tuples in a relational database.

An object-oriented data warehousing system includes a data warehouse and underlying data sources (Chen, 1999; Chen et al., 1999). Since the information in a data warehouse is collected from the underlying data sources, appropriate views have to be defined to speed up the OLAP requests and query processing. In an object-oriented database system, the views act as virtual classes to augment the class hierarchy to enhance the modeling and schema restructuring capabilities (Kim, 1995). Many useful view maintenance techniques have been proposed in (Ra and Rundensteiner, 1997; Rundensteiner, 1992; Scholl et al., 1991). In the object-oriented warehouses, the views have to be defined in the data warehouse and the related objects need to be reproduced from the data sources. After the views have been defined in a data warehouse, a procedure is invoked to obtain the necessary information from the source databases and to create appropriate classes and instances according to the definition of views.

Since the objects desired are stored in the data warehouse, the problem of incrementally maintaining these objects to maintain the consistency between data warehouses and data sources will occur. Designing appropriate maintenance algorithms to keep the consistency of the data warehouse and source databases is thus necessary.

3. PROBLEM DEFINITIONS

In an object-oriented database, each class is associated with a unique object *identifier*, a set of *attributes*, and a set of procedures called *methods*. Each attribute has its data type, which may be atomic or another class. The classes can be organized into a hierarchical structure, with the function of inheritance among them.

Formally, let I be a set of identifiers, A be a set of symbols called attribute names, T be a set of data types allowed for A, V be a set of values presenting the meaning of A, and M be a set of processing methods. A *class* in an object-oriented database can be defined as follows.

Definition 1 (Class):

A class c is a quadruple $\{cid, ca, ct, cm\}$, where $cid \in I$, $ca = < ca_1, ca_2, \ldots, ca_n >$ with $ca_i \in A$ and $i=1$ *to* n, $ct = < ct_1, ct_2, \ldots, ct_n >$ with $ct_j \in T$ and $j=1$ *to* n, and $cm \subseteq M$.

Example 1:

Figure 2 gives a simple example of the four classes, *Employee*, *Name*, *Dept*, and *Office*. The class *Employee* has four attributes, *EmployeeID*, *EmployeeName*, *EmployeeDept*, *EmployeeTitle* and one method *Counter()*. The attribute *EmployeeID* is a character type. The attribute *EmployeeName* and *EmployeeClass* are *Name* and *Classes* types, which are classes. The relationship among these classes can be represented by a graph as shown in Figure 3, where a circle represents a class, a rectangle represents an atomic type, and an ellipse represents a set of attributes.

```
1. Class Employee {
     EmployeeID      char(10),
     EmployeeName    Name,
     EmployeeDept    Dept,
     EmployeeTitle   char(10)
     Counter()       int}
2. Class Office {
     State           char(20),
     City            char(20)}
3. Class Name {
     First           char(20),
     Middle          char(20),
     Last            char(20)}
4. Class Dept {
     DeptID          char(3),
     DeptName        char(40)
     DeptOffice      Office
     Counter()       int}
```

Figure 2 An example of classes

For the class *Employee* in this example, *cid* = *Employee*, *ca* = {*EmployeeID*, *EmployeeName*, *EmployeeDept*, *Employee-Title*}, *ct* = {char, *Name*, *Depart*), and *cm* = {*Counter()*}. Let *C* be the set of classes defined in the source databases. That is $C = \{c_1, c_2, \ldots, c_n\}$, where c_i is a class, $1 \le i \le n$.

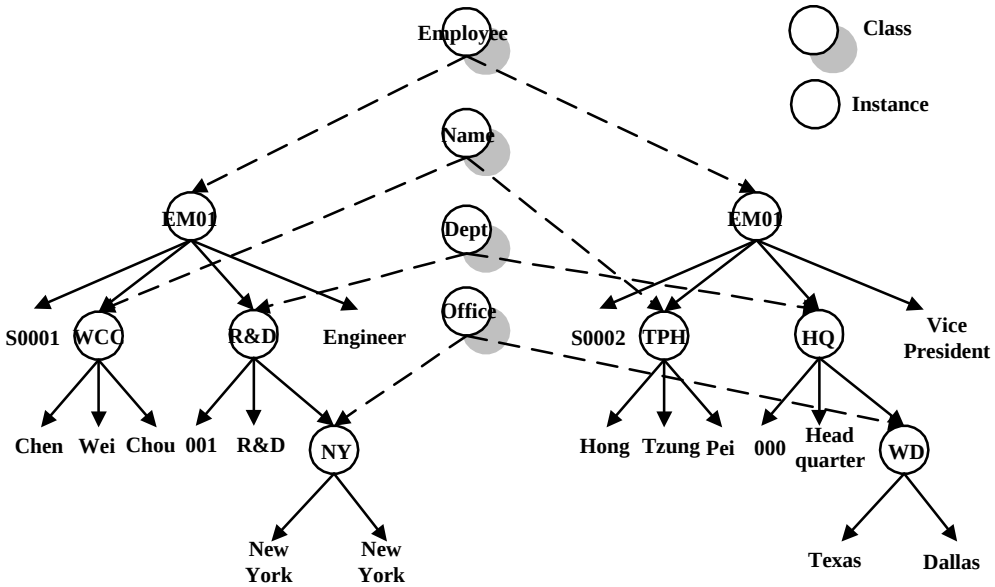

Figure 4 A graphical representation of the relationship among classes and instances

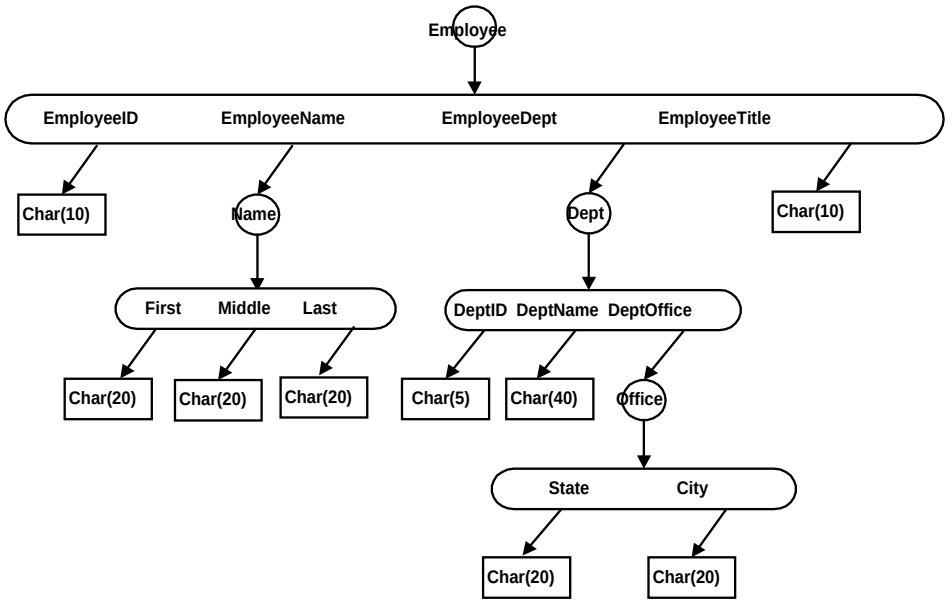

Figure 3 A graphical representation of classes

An instance is created by referring to a class and inheriting some particular characteristics from the class. Similarly, each instance is associated with a unique instance *identifier*, a set of *attributes*, and a set of procedures called *methods*. Each attribute value can be an atomic value or an instance from another class. Formally, an instance in an object -oriented database can be defined as follows.

Definition 2 (Instance):

An instance t = {*tid*, *ta*, *tv*, *tm*, *tc*} is created and inherits from a certain class *tc* = {*cid*, *ca*, *ct*, *cm*} such that $tid \in I$, $ta = ca$, $tv = <tv_1, tv_2, \ldots, tv_n>$ with tv_i being of type ct_i and i = 1 to n, and $tm \subseteq cm$.

Example 2:

```
View R&DEmployee ( FirstName char(20), LastName char(20), City char(20))
as {
Select
   EmployeeName.First,
   EmployeeName.Last,
   EmployeeDept.DeptOffice.City
From Employee
Where EmployeeDept.DeptName = "R&D"};

View TexasDept (DeptID char(3) , DeptName  char(40) , City char(20))
as {
Select
   DeptID,
   DeptName,
   DeptOffice.City
From Dept
Where DeptOffice.State = "Texas"};
```

Figure 5 An example of view definitions

For the classes defined in Figure 2, assume that two instances are created by referring to class *Office*. One is called *NY* with attribute values (New York, New York) and the other is called *TD* with attribute values (Texas, Dallas). Similarly, assume that two instances, *R&D* and *Manager* respectively with attribute values (001, R&D, *NY*) and (000, Headquarter, *TD*), are created by referring to the class *Dept*. Assume that two instances, *WCC* and *TPH* respectively with attribute values (Chen, Wei, Chou) and (Hong, Tzung, Pei), are created by referring to the class *Name*. Also assume that two instances, *EM01* and *EM02* respectively with attribute values (S0001, *WCC*, *R&D*, Engineer) and (S0002, *TPH*, *HQ*, Vice President), are created by referring to the class *Employee*. The relationship among classes and instances in this example can be represented by a graph as shown in Figure 4, where a circle with shadow represents a class, a circle without shadow

represents an instance, a solid line points from an instance to its attribute, and a dashed line points from a class to an instance.

For the instance *ST01* in this example, *tid* = *ST01*, *ta* = {*EmployeeID*, *EmployeeName*, *EmployeeClass*}, *tv* = {863201, WCC, A1}, *tm* = {*Counter()*}, and *tc* = *Employee*.

A view is characterized by a unique *view identifier*, a set of *attributes* and a *query sentence*. The number of attributes is equal to that in the query sentence. Formally, a view in an object-oriented database can be defined as follows.

Definition 3 (View):

A view *WV* in an object-oriented data warehouse is a quadruple { *wvid*, *wva*, *wvv*, *wvs*} such that $wvid \in I$, $wva = <wva_1, wva_2, \ldots, wva_n>$ with $wva_i \in A$ and i = 1 to n, $wvv \subseteq T$, and *wvs* is a query statement (**Select** *S*, **From** *F*, **Where** *W*), where $S = <s_1, s_2, \ldots, s_n>$ with $s_i \in A$ and i = 1 to n, $F \in C$, *W* denotes the query conditions, and $|wva| = |S|$.

Example 3:

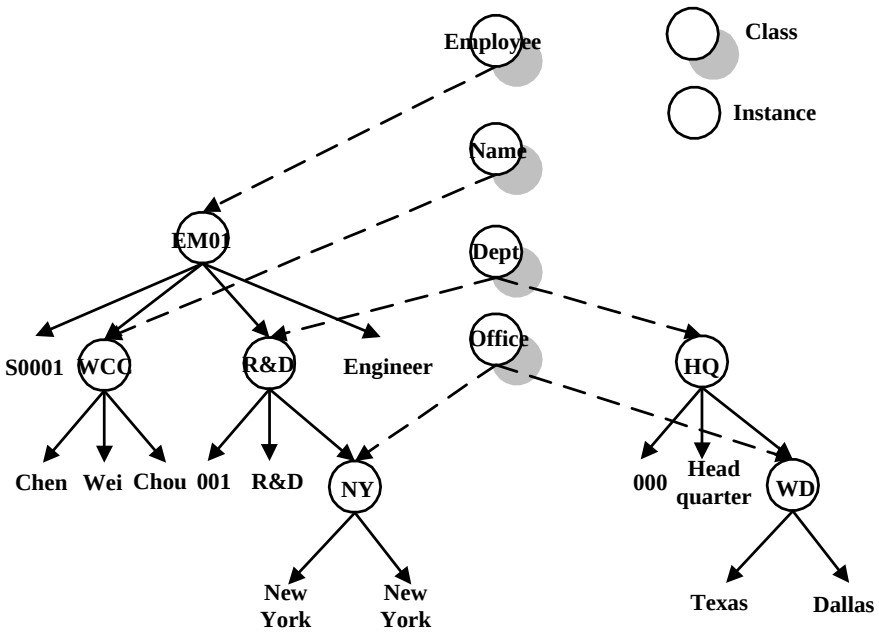

Figure 6 A graphical representation of the relationship among classes and instances in the object-oriented data warehouse

Figure 5 gives a simple example of two view definitions, *R&DEmployee* and *TexasDept*. For the view *R&DEmployee* in this example, *wvid* = *R&DEmployee*, *wva* = (FirstName, LastName, City), *wvv* = (char(20), char(20), char(20)), and *wvs* = **"Select** EmployeeName.First, EmployeeName.Last, EmployeeDept.DeptOffice.City **From** Employee **Where** EmployeeDept.DeptName = "R&D"".

Definition 4 (Warehouse):

An object-oriented data warehouse *W* is a triple {*C*, *V*, *I*}, where *C* is the set of classes, *V* is the set of view definitions, and *I* is the set of instances generated according to *C*.

Example 4:

For example, assume *C* is the same as that given in Figure 3, *V* is the same as that given in Figure 6, and the eight instances stored above exist in the source database. According to *C* and *V*, there are only six instances, including *EM01*, *WCC*, *R&D*, *NY*, *HQ* and *TD*, which satisfy the conditions of view definitions. These six instances are thus sent from the source database to the warehouse and are thus saved in the object-oriented data warehouse. The relationship between classes and instances in the object-oriented data warehouse for this example is represented by a graph as shown in Figure 6.

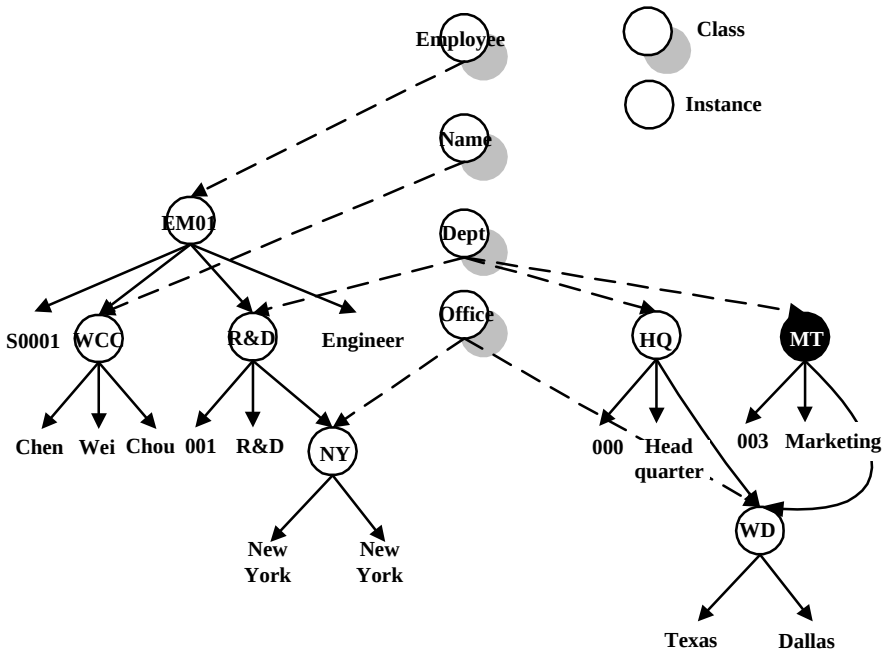

Figure 7 The Graphical Representation for Example 5

An object-oriented data warehouse is said to be consistent with the underlying source databases if it contains the necessary class structure and all instances (with their correct inheritance structure) satisfying the given view definitions even after the source databases have been updated by instance insertion, modification or deletion. Below, three view-maintenance algorithms are proposed to maintain the consistency of an object-oriented data warehouse. They are used respectively for instance insertion, instance deletion and instance modification. The details are described as follows.

4. INSTANCE INSERTION

When a new instance *tid* is inserted into the source database, a transaction message should be sent from the data collector to the data warehouse for view maintenance. The format of a transaction message for instance insertion is proposed as follows:

$$MsgID, insert, tid, cid, \{v_1, v_2, \ldots, v_n\},$$

where *MsgID* denotes the message identifier of this transaction, *insert* denotes type of the message, *tid* denotes the identifier of the instance, *cid* denotes the updated class identifier of this instance, and v_i denotes the *i*-th attribute value in the instance *tid*, $i = 1$ to n. For example, if an instance *MT* of the class *Dept* with attribute values {003, Marketing, TD} is inserted into the source database, the data collector will detect it and send a transaction message (*0001*, *insert*, *MT*, *Dept*, {*003*, *Marketing*, *TD*}) to the warehouse. The view-maintenance algorithm for processing the above instance-insertion transaction message is proposed as follows.

The maintenance algorithm for instance insertion:

Input : A data warehouse W (C, V, I) and an inserted instance *tid* of class *cid*.

Output : A revised data warehouse W' (C, V, I').

Step 1. Receive an instance-insertion transaction message which is formed from the data collector.

Step 2. Check whether the class *cid* is used in the views V in the data warehouse W. If class *cid* is used in V, do the next step; Otherwise, set $W' = W$ and exit the algorithm.

Step 3. Check whether the instance *tid* satisfies the conditions of the views V which refer to the class *cid*; If the instance satisfies the condition of at least one view, add instance *tid* to I of the warehouse W and set $I' = I$, $W' = W$; Otherwise, set $W' = W$ and exit the algorithm.

Example 5:

Continuing Example 4, assume that an instance has been inserted into the source database, and the transaction message was formed as (*0001*, *insert*, *MT*, *Dept*, {*003*, *Marketing*, *TD*}). This message is processed by the instance-insertion algorithm as follows.

Step 1. Receive the transaction message (*0001*, *insert*, *MT*, *Classes*, {*003*, *Marketing*, *TD*}) from the data collector.

Step 2. Since the views *TexasDept* and *R&DEmployee* existing in the warehouse W refer to the class *Dept*, the algorithm thus executes Step 3.

Step 3. Since the instance *MT* satisfies the condition in view *TexasDept*, it is thus added to the instance set I of the data warehouse W with its attribute values {003, Marketing, TD}.

The graphical representation of the warehouse after the instance *MT* has been processed is shown in Figure 7.

5. INSTANCE DELETION

When an existing instance *tid* is deleted from the source database, a transaction message is sent from the data collector to the data warehouse for view maintenance. The format of a transaction message for deleting an instance is proposed as follows:

MsgID, delete, cid, tid.

For example, assume that an instance *NY* of class *Office* is deleted from the source database. The data collector will detect it and send a transaction message (*0002, delete, Office, NY*) to the warehouse. If the instance deleted refers to or is referred to by other instances, each of the following alternatives may be adopted:

1. directly rejecting the deletion operation,
2. modifying the referring attribute values to *null*, or
3. cascading the deletion operation.

The second alternative is chosen here in our algorithm. The view-maintenance algorithm for processing the above instance-deletion statement is proposed as follows.

The maintenance algorithm for instance deletion:

Input : A data warehouse *W* (*C*, *V*, *I*) and a deleted instance *tid* of class *cid*.

Output : A revised data warehouse *W'* (*C*, *V*, *I'*).

Step 1. Receive an instance-deletion transaction message which is formed from the data collector.

Step 2. Search the data warehouse *W* for instances *tid*; If instance *tid* exists in *I*, do the next step; Otherwise, set *W'* = *W* and exit the algorithm.

Step 3. Find the classes in the warehouse W, which have attributes of class type *cid*. Denote these classes as *A*.

Step 4. For every element in *A*, find its instances with an attribute value being *tid*. Change the attribute values fo these instances to *null*.

Step 5. Remove *tid* from *I* in the warehouse *W* and set *I'* = *I*, *W'* = *W*.

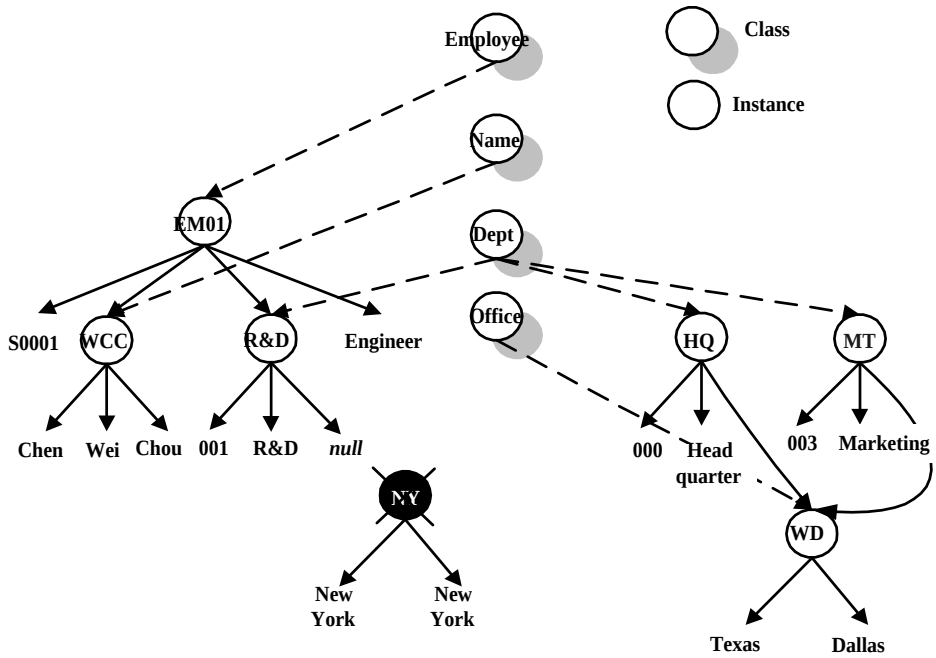

Figure 8 The Graphical Representation for Example 6

After Step 5, the instance *tid* has been deleted from the data warehouse. An example is given below to demonstrate the instance-deletion algorithm.

Example 6:

Continuing Example 5, assume that an instance has been deleted in the source database, and the transaction message is formed as (*0002, delete, Office, CS*). This message is processed by the instance-deletion algorithm as follows.

Step 1. Receive the transaction message (*0002, delete, Office, CS*) from the data collector.
Step 2. Since the instance *CS* has existed in the warehouse *W*, the algorithm executes Step 3.
Step 3. Find the attribute *DeptOffice* of class *Dept*, which is of the type *Office*.
Step 4. Since the value of attribute *DeptOffice* was originally *NY* in the instance *R&D*, the attribute values of *DeptOffice* in the instances is changed to *null*.
Step 5. Remove the instance *NY* from the warehouse *W*.

A graphical representation of the warehouse after the instance *NY* has been deleted is shown in Figure 8. The attribute value of *DeptOffice* in *R&D* has been updated as null.

6. INSTANCE MODIFICATION

When the attribute values of an instance *tid* in the source database are changed, a transaction message is sent from the data collector to the data warehouse for view maintenance. Th e format of a transaction message for modifying an instance is proposed as follows:

MsgID, update, tid, cid,

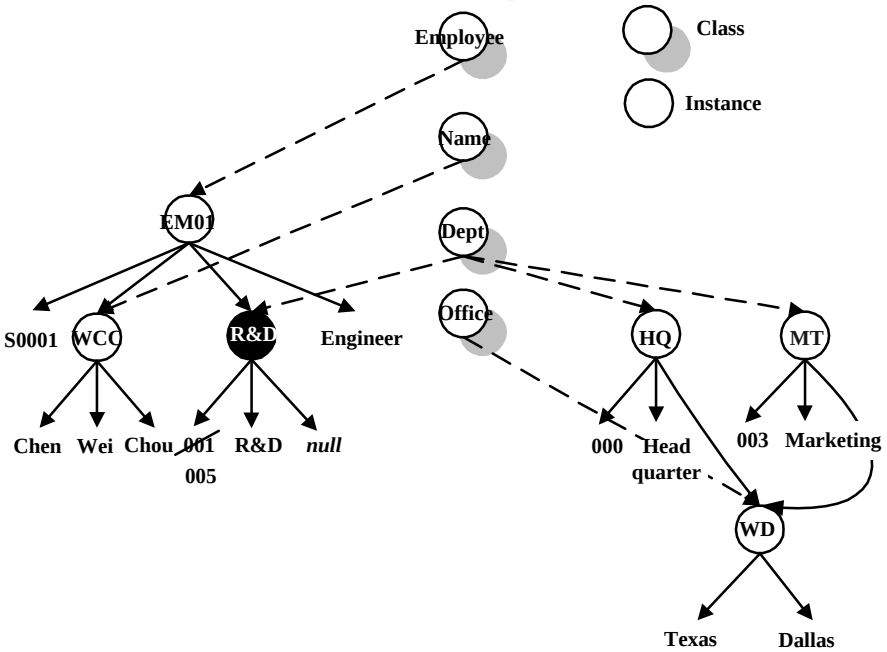

Figure 9 The Graphical Representation for Example 7

$\{(u_1\ v_1),(u_2\ v_2), \ldots, (u_k\ v_k)\}$,

where u_i denotes the i-th attribute name to be updated and v_i denotes the new value of u_i. For example, assume that the value of attribute *DeptID* in instance *R&D* is changed from 001 to 005. The data collector will detect it and send a transaction message (*0003, update, Dept, R&D,* {(*DeptID 005*)}) to the warehouse. The view-maintenance algorithm for processing the above instance-modification transaction message is proposed as follows.

The maintenance algorithm for instance modification:

Input : data warehouse *W* (*C, V, I*) and a modified instance *tid* of class *cid*.

Output : A revised data warehouse *W'* (*C, V, I'*).

Step 1. Receive an instance-modification transaction message which is formed from the data collector.

Step 2. Search the data warehouse *W* for instances *tid*; If instance *tid* exists in *W*, do the next step; Otherwise, set *W'* = *W* and exit the algorithm.

Step 3. For the instance *tid* in the warehouse, change its attribute values according to the transaction message.

Step 4. Check whether the instance *tid* satisfies the conditions of the views *V* which refer to the class *cid*; If the instance satisfies the condition of at least one view, keep instance *tid* in *I* of the warehouse *W*; Otherwise, remove *tid* from *I* in the warehouse *W*.

After Step 4, the attribute values of instance *tid* have been modified in the data warehouse. An example is given below to demonstrate the instance -modification algorithm.

Example 7:

Continuing Example 6, assume that the attribute values of an instance have been modified in the source database, and the transaction message is formed as (*0003, update, Dept, R&D,* {(*DeptID 005*)}). This message is processed by the instance-modification algorithm as follows.

Step 1. Receive the transaction message (*0003, update, Dept, R&D,* {(*DeptID 005*)}) from the data collector.

Step 2. Since the instance *A1* exists in the warehouse *W*, the algorithm executes Step 3.

Step 3. Change the value of attribute *DeptID* of the instance *R&D* from 001 to 005.

Step 4. Since *A1* satisfies the condition of the view *R&DEmployee*, it is kept in *W*.

The graphical representation of the warehouse after the attribute value of instance *A1* has been changed is shown in Figure 9.

7. TIME COMPLEXITY ANALYSIS

The time complexity of the proposed maintenance algorithms is analyzed in this section. Let *m* be the number of classes, *n* be the number of instances, and *k* be the number of view definitions in the data warehouse. Also define *i* is the maximum possible number of attributes in a view and *j* is the maximum possible number of attributes in a class. The time complexity of each step in the instance insertion algorithm is shown in Table 1.

Table 1 The time complexity of instance insertion algorithm

Step No	Time Complexity
Step 1	*O(1)*
Step 2	*O(ki)*
Step 3	*O(kj)*
Total	*O(ki)+O(kj)*

The time complexity of each step in the instance deletion algorithm is shown in Table 2:

Table 2 The time complexity of instance deletion algorithm

Step No	Time Complexity
Step 1	*O(1)*
Step 2	*O(n)*
Step 3	*O(mj)*
Step 4	*O(mnj)*
Step 5	*O(1)*
Total	*O(mnj)*

The time complexity of each step in the instance modification algorithm is shown in Table 3:

Table 3 The time complexity of instance modification algorithm

Step No	Time Complexity
Step 1	*O(1)*
Step 2	*O(n)*
Step 3	*O(j)*
Step 4	*O(kj)*
Total	*O(n)+ O(kj)*

8. CONCLUSION

Maintenance of the data warehouse is very important to the accuracy of the on-line analytical processing. In this paper, we have discussed the concept of the object-oriented data warehouse, and have described the maintenance problems in such a data warehouse. We have also proposed three algorithms to maintain the consistency between the data warehouse and the source databases. Although the proposed algorithms can be used to make object-oriented data warehousing practical, it is only a beginning. Much work still needs to be done in this field. In the future, we will try to apply our research results in fuzzy machine learning (Hong and Lee, 1996; Hong and Chen, 1999; Hong and Chen, 2000; Hong and Tseng, 1997) to data-warehousing and propose other data models for different problem domains (Chen, 1999; Chen et al., 1999).

ACKNOWLEDGEMENT

The authors would like to thank the anonymous referees for their very constructive comments.

REFERENCES

1. 陳威州、洪宗貝與林文揚，「可自動維護之物件導向地理資訊倉儲架構」，中華地理資訊學會年會暨學術研討會，1999。
2. Chen, W. C. *Object-Oriented Data Warehousing and its Maintenance Technologies*, Master Dissertation, I-Shou University, Taiwan, R.O.C., 1999.
3. Chen, W. C., Hong, T. P. and Lin, W. Y. "View Maintenance in an Object-Oriented Data Warehouse," *The Fourth International Conference on Computer Science and Informatics*, 1998, pp. 353-356.
4. Chen, W. C., Hong, T. P. and Lin, W. Y. "View Maintenance of Object-Oriented Data Warehousing Using the Composite Model," *The Fifth International Conference on Information Systems Analysis and Synthesis*, 1999.
5. Chen, W. C., Hong, T. P. and Lin, W. Y. "A Compressed Data Model in Object-Oriented Data Warehousing," *The IEEE 1999 International Conference on Systems, Man and Cybernetics*, 1999.

6. Chen, W. C., Hong, T. P. and Lin, W. Y. "A Composite Data Model in Object-Oriented Data Warehousing," *The 31st International Conference on Technologies of Object-Oriented Language and Systems*, China, 1999.
7. Chen, W. C., Lin, W. Y. and Hong, T. P. "View Update in Object-Oriented Data Warehousing Using Uncompressed Data Model," *The Fourth World Conference on Integrated Design and Process Technology*, 1999.
8. Chen, W. C., Lin, W. Y. and Hong, T. P. "Object-Oriented Data Warehousing from Multiple Sources," *National Computer Symposium* (l). 1, pp. 258-262, 1999.
9. Cui, Y., Widom, J. and Wiener, J. L. "Tracing the Lineage of View Data in a Data Warehousing Environment," *Technical Note*, 1997.
10. Gupta, H. "Selection of Views to Materialize in a Data Warehouse," *Proceedings of the International Conference on Database Theory*, 1997.
11. Gupta, H., Harinarayan, V. and Quass, D. "Aggregate-Query Processing in Data Warehousing Environments,"*Proceedings of the 21st VLDB Conference*, 1995.
12. Gupta, H., Harinarayan, V., Rajaraman, A., and Ullman, J. "Index Selection for OLAP," *Proceedings of the International Conference on Data Engineeri ng*, 1997.
13. Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., and Zhuge, Y. "The Stanford Data Warehousing Project," *IEEE Data Engineering Bulletin*. 1995.
14. Harinarayan, V., Rajaraman, A. and Ullman, J. "Implementing Data Cubes Efficiently," *ACM SIGMOD Conference*, 1996.
15. Hong, T. P. and Lee, C. Y. "Induction of Fuzzy Rules and Membership Functions from Training Examples," *Fuzzy Sets and Systems* (84), 1996, pp. 33-47.
16. Hong, T. P. and Chen, J. B. "Finding Relevant Attributes and Membership Functions," *Fuzzy Sets and Systems* (103), 1999, pp. 389-404.
17. Hong, T. P. and Chen, J. B. "Processing Individual Fuzzy Attributes for Fuzzy Rule Induction," *Fuzzy Sets and Systems* (112:1), 2000, pp. 127-140.
18. Hong, T. P. and Tseng, S. S. "A Generalized Version Space Learning A lgorithm for Noisy and Uncertain Data," *IEEE Transactions on Knowledge and Data Engineering* (9:2), 1997, pp. 336-340.
19. Huyn, N. "Efficient View Self-Maintenance," *Proceedings of the ACM Workshop on Materialized Views*, 1996.
20. Huyn, N. "Efficient Self-Maintenance of Materialized Views," *Technical Note*, 1996.
21. Huyn, N. "Multiple-View Self-Maintenance in Data Warehousing Environments," *Proceedings of the 23rd VLDB Conference*, 1997.
22. Huyn, N. "Exploiting Dependencies to Enhance View Self-Maintainability," *Technical Note*, 1997.
23. Inmon, W. H. and Kelley, C. *Rdb/VMS: Developing The Data Warehouse*, QED Publishing Group, Boston, Massachusetts, 1993.
24. Kawaguchi, A., Lieuwen, D., Mumick, I., Quass, D. and Ross, K. "Concurrency Control Theory for Deferred Materialized Views," *Proceedings of the International Conference on Database Theory*, 1997.
25. Kim W. *Modern Database Systems*, ACM Press, New York, 1995

26. Labio, W. J. and Garcia-Molina, H. "Efficient Snapshot Differential Algorithms for Data Warehousing," *Proceedings of VLDB Conference*, 1996.
27. Labio, W. J., Zhuge, Y., J. Wiener, L., Gupta, H., Garcia-Molina, H. and Widom, J. "The WHIPS Prototype for Data Warehouse Creation and Maintenance," *Proceedings of the ACM SIGMOD Conference*, 1997.
28. Mumick, I., Quass, D. and Mumick, B. "Maintenance of Data Cubes and Summary Tables in a Warehouse," *Proceedings of the ACM SIGMOD Conference*, 1997.
29. O'Neil, P. and Quass, D. "Improved Query Performance with Variant Indexes," *Proceedings of the ACM SIGMOD Conference*, 1997.
30. Quass, D. "Maintenance Expressions for Views with Aggregation," *Proceedings of the ACM Workshop on Materialized Views*, 1996.
31. Quass, D., Gupta, A., Mumick, I. S., and Widom, J. "Making Views Self-Maintainable for Data Warehousing. *Proceedings of the Conference on Parallel and Distributed Information Systems*, 1996.
32. Quass, D. and Widom, J. "On-Line Warehouse View Maintenance for Batch Updates," *Proceedings of the ACM SIGMOD Conference*, 1997.
33. Ra, Y. G. and Rundensteiner, E. A. "A "Transparent Schema-Evolution System Based on Object-Oriented View Technology," *IEEE Transaction on Knowledge and Data Engineering* (9-4), 1997, pp. 600-624.
34. Rundensteiner E. A. "A Methodology for supporting Multiple Views in Object-Oriented Databases," *Proceedings of 18th International Conference on Very Large Data Bases*, 1992, pp. 187-198.
35. Scholl M. H., Laasch C. and Tresch M. "Updateable Views in Object-Oriented Databases," *Second International Conference on Deductive and Object-Oriented Databases*, 1991, pp. 189-207.
36. Widom, J. "Research Problems in Data Warehousing," *Proceedings of the 4th Int'l Conference on Information and Knowledge Management*. 1995.
37. Yang, J. and Widom, J. "Maintaining Temporal Views Over Non-Temporal Information Sources for Data Warehousing," *Proceedings of the 6th International Conference on Extending Database Technology*, 1998.
38. Zhuge, Y. and Garica-Molina, H. "Graph Structured Views and Their Incremental Maintenance," *Proceedings of the International Conference on Data Engineering*, 1998.
39. Zhuge, Y., Garcia-Molina, H., Hammer, J. and Widom, J. "View Maintenance in a Warehousing Environment," *Proceedings of the ACM SIGMOD Conference*, 1995.
40. Zhuge, Y., Garcia-Molina, H. and Wiener, J. L. "Consistency Algorithms for Multi-Source Warehouse View Maintenace. *Journal of Distributed and Parallel Databases* (6:1), 1998, pp. 7-40.
41. Zhuge, Y., Garcia-Molina, H. and Wiener, J. L. "The Strobe Algorithms for Multi-Source Warehouse Consistency," *Proceedings of the Conference on Parallel and Distributed Information Systems*, 1996.
42. Zhuge, Y., Wiener, J. L. and Garcia-Molina, H. "Multiple view consistency for data warehousing," *Proceedings of the International Conference on Data Engineering*, 1997.

About the Authors

Wei-Chou Chen (陳威州**)** received his B.S. degree in information management from Kaohsiung Polytechnical Institute in 1997, and his MS degree in computer science and information engineering from I-Shou University in 1999. He is currently a Ph.D. student in computer and information sciences in National Chiao Tung University

In 1999, he joined the Laboratory of Knowledge Engineering, National Chiao Tung University, where he is working on the technologies of knowledge engineering, object-oriented data warehousing and case-base reasoning. His current interests include data warehousing, object-oriented technologies, expert systems, decision support systems, genetic algorithms, fuzzy set theory, management information systems and internet computing. He is a student member of the ACM and the OOTSIG of the Institute of Information and Computing Machinery.

Tzung-Pei Hong (洪宗貝**)** received his B. S. degree in chemical engineering from National Taiwan University in 1985, and his Ph.D. degree in computer science and information engineering from National Chiao-Tung University in 1992.

From 1987 to 1994, he was with the Laboratory of Knowledge Engineering, National Chiao-Tung University, where he was involved in applying techniques of parallel processing to artificial intelligence. He was an Associate Professor at the Department of Computer Science in Chung-Hua Polytechnic Institute from 1992 to 1994, and at the Department of Information Management in Kaohsiung Polytechnic Institute from 1994 to 1997. He was in charge of the whole computerization planning for National University of Kaohsiung in Preparation from 1997 to 2000. He is currently a Full Professor at the Department of Information Management in I-Shou University and the Director of Library and Information Center in National University of Kaohsiung. He has published more than 150 research papers in international/national journals and conferences. His current research interests include parallel processing, machine learning, neural networks, fuzzy sets, genetic algorithms, expert systems, management information systems and www applications.

Dr. Hong is a member of ACM, IEEE, the Chinese Fuzzy Systems Association, the Taiwanese Association for Artificial Intelligence, and the Institute of Information and Computing Machinery.

Wen-Yang Lin (林文揚**)** received his B. S. and M. S. both in Computer Science and Information Engineering from National Chiao-Tung University in 1988 and 1990, respectively. He then received his Ph.D. in Computer Science and information engineering from National Taiwan University in 1994.

In 1996, he joined the Department of Information Management at I -Shou University and now is an Associate Professor. He is primarily interested in the area of sparse matrix technology and large-scale supercomputing. Currently he is also interested in data warehousing, data mining and evolutionary computations.

Dr. Lin is a member of SIAM, IEEE, the Taiwanese AI Association and the Institute of Information and Computing Machinery.