Object-Oriented Data Warehousing from Multiple Sources

Wei-Chou Chen
Inst. of Computer and Information Science
National Chiao-Tung University
Hsinchu, 30050, Taiwan, R. O. C.
sirius@cis.nctu.edu.tw

Tzung-Pei Hong and Wen-Yang Lin
Dept. of Information Management
I-Shou University
Kaohsiung, 84008, Taiwan, R.O.C.
{tphong,wylin}@csa500.isu.edu.t

ABSTRACT

A data warehouse is an information provider that collects necessary data from individual source databases to support analytical processing of decision-support functions. In th past, we introduced the concept of object-oriented data warehousing in a single data source and proposed view-maintaining algorithms to support OLAP. In this paper, the proposed data model of the object-oriented data warehousing is extended to be suitable in multiple-source environments. Meta-objects are also used to keep th management and maintenance of object -oriented data warehousing efficient. Moreover, two vie -related algorithms in such a data model are also presented to kee p views desired in a data warehouse.

Keywords: data model, data warehousing, algorithm object-oriented data warehousing, OLAP.

1. Introduction

A data warehouse is an information provider that collects necessary data from individual source databases to support analytical processing of decision-support functions [1, 6]. For a data warehouse to work well, large amounts of source data are needed and appropriate views must be adopted in order to provide up-to-date information to users. In the past decades, t he relational data model gained much attention, and most researches on data warehousing primarily focused on this model. Recently, object -oriented technologies grown rapidly and have been widely adopted in the fields of databases, artificial intelligence, software engineering and geographic information systems. Applications of large object-oriented database systems may also require a data warehouse to improve the efficiency of queries for decision support, especially when the databases are distributed over several places.

In [11], Zhuge and Garica-Molina proposed vie maintenance algorithms in a graph-structured data warehouse, initiating the idea of research in object-oriented data warehousing. In [2, 4], we introduced the concept of the object-oriented data warehousing and proposed three data warehouse models suitable in object -oriented environments with a single data source. That proposed model maintained the original structures in the source database to store the materialized views in the object-oriented data warehouse. That is, the attributes and relationships of the classes and instances necessary, with their original structures, were duplicated from the source databases to the data warehouse [3, 5]. In this paper, we extend this data model to manage views derived from multiple-source environments. Two view-related algorithms including the view-creation and view -deletion, are als proposed to keep the views and their classes and instances desired in the data warehouse.

2. Data Warehousing

The concept of data warehousing was first proposed by Inmon [6] in 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 on several locations, a data warehouse has the responsibility to collect the necessary data and save them in appropriate forms. Figure 1 shows the architecture of a typical data warehousing system.

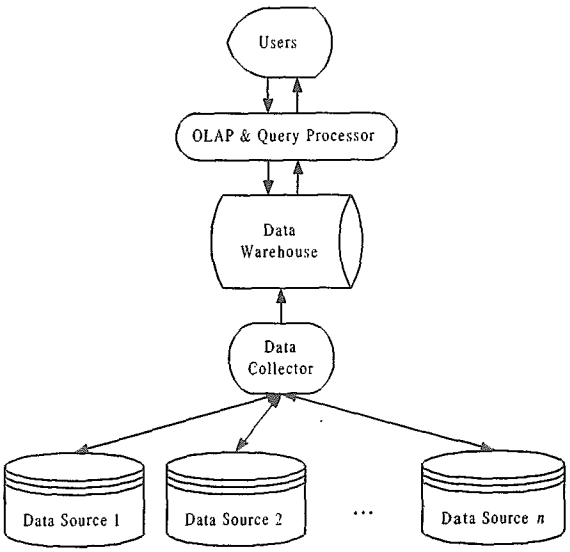

Figure 1. Architecture of a typical data warehousing system

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 t the requirements of end users and the views defined in the data warehouse. The *data warehouse* receives data from the data collector, then filters them and stores them in its own database. The *OLAP and query processor* provides all necessary information for users queries and OLAP requirements.

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 be another class. The classes can be organized into a hierarchical structure, with the function of inheritance among them [7, 8, 9].

Formally, let B be a set of source databases, 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 data warehouse can be defined as follows.

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

Example 1: Figure 2 gives a simple example of four classes, *StudInfo*, *Name*, *Dept* and *Classes*, in the first source database *DS1*. The class *StudInfo* has three attributes *StudID*, *StudInfo*, *StudClass* and one method *Counter*(). The attribute *StudID* is of type character; the attribute *StudName* and *StudClass* are of types *Name* and *Classes*, which are classes. Figure 3 shows the second source database, called *DS2*, which has the same four classes, *StudInfo*, *Name*, *Dept* and *Classes*, as *DS1* has. The class *StudInfo* however has four attributes, *StudID*, *StudInfo*, *StudClass*, *StudPic* and two methods, *Counter*() and *ImageView*(). The attributes *StudID*, *StudName* and *StudClass* have the same types as those in the class *StudInfo* in DS_1 have, and the attribute *StudPic* is of type image.

For the class *StudInfo* in DS_1, *cid* = *StudInfo*, *ca* = {*StudID*, *StudName*, *StudClass*}, *ct* = {char(10), *Name*, *Classes*), and *cm* = {*Counter()*}.

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

Figure 2. An example of classes in *DS1*.

Let C_k be the set of classes defined in the k-th source database. That is $C_k = \{c_{k1}, c_{k2}, \ldots, c_{kn}\}$, where c_{ki} is the i-th class in the k-th source database, $1 \le i \le n$. Let C be the set of C_k's in the object-oriented data warehouse, that is $C = \{C_1, C_2, \ldots, C_j\}$, where j is the number of source databases. An instance is created by referring to a class and inheritin some particular characteristics from the class. Similarly, each instance is associated with a unique *instance*

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

Figure 3. An example of classes in *DS2*.

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 $tv_i \in V$ and tv_i being of type ct_i and $i = 1$ *to* n, and $tm \subseteq cm$.

Example 2: For the example in Figure 2, assume two instances are created by referring to the class *Dept* in *DS1*. 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* respectively with attribute values (Chen, Wei, Chou) and (Hong, Tzung, Pei) 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 b referring to the class *StudInfo* in *DS1*.

For the instance *ST01*, *tid* = *ST01*, *ta* = {*StudID*, *StudName*, *StudClass*}, *tv* = {863201, WCC, A1}, *tm* = {*Counter*()}, and *tc* = *StudInfo*.

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 can be defined as follows.

DEFINITION 3 (Data Warehouse View): A data warehouse view *WV* in an object-oriented data warehouse is a quadruple $\{wvid, wva, wvs, wvf\}$ such that $wvid \in I$, $wva = <wva_1, wva_2, \ldots, wva_n>$ with $wva_i \in A$ and $i = 1$ to n, wvs is a object-oriented query statement (**Select** *S*, **From** *F*, **Where** *W*), where $S = <s_1, s_2, \ldots, s_n>$ with $s_i \in A$, $i = 1$ to n, and $|S|=|wva|$, $F = <b_1 f_1, b_2 f_2, \ldots, b_k f_k>$ with $b_i \in B$, $f_i \in C$, $i = 1$ t k, and W denotes the condition sentences.

A view in the 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 condition sentences. Two kinds of condition sentences may be used here. One is the *independence condition sentence*, in which variables can be retrieved in a single source database. The other is the *dependence condition sentence*, in which variables must be retrieved from more than one source database. Restated, when a condition sentence can be checked in a single source database, it is called an *independence condition sentence.* Otherwise, it is called a *dependence condition sentence.*

Example 3: Figure 4 gives a simple example of a vie definition, *BothClassList*. The condition sentence "DS1.Dept.DeptName = DS2.Dept.DeptName" is a dependence condition sentence.

```
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 4. An example of view definitions

Let V be the set of WV in the object-oriented data warehouse. That is $V = \{WV_1, WV_2, \ldots, WV_j\}$, where WV_i is the i-th view in the data warehouse, $1 \le i \le j$ and j is the number of views defined in the object -oriented data warehouse. Also, meta-objects are used in the data-warehouse to keep the class identifiers and instance identifiers used in view definitions for later management. They are defined as follows.

DEFINITION 4 (Meta-Object): A meta-object *mo* is 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*.

Let O be the set of *mo*'s in the object-oriented data warehouse. That is, $O = \{mo_1, mo_2, \ldots, mo_j\}$, where mo_i is the i-th meta-object, $1 \le i \le j$ and j is the number of view defined in the data warehouse.

DEFINITION 5 (Warehouse): An object-oriented data warehouse W is a quadruple $\{C, V, I, O\}$, where C is a set of classes, V is a set of view definitions, I is a set of instances generated according t C and V, and O is a set of meta-objects generated according to V.

4. View-Related Algorithms for an Object-Oriented Data Warehouse

In this section, two view-related algorithms are proposed t keep the views desired in an object -oriented data warehouse. They are respectively used for view creation and view deletion. Details are described as follows.

4.1 View Creation

A new view WV in the object-oriented data warehouse can be created using the following statement

Create Warehouse View $WV(wva_1, wva_2, \ldots, wva_n)$ **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 $w_1, w_2, \ldots, w_m$

In the above statement, wva_n denotes the n-th attribute in the view WV, $b_{s_n}.a_n$ denotes the n-th attribute from a class in the source database b_{s_n} (if the attributes and the classes of the attributes exists in all the source databases of the view, the parameter b_{s_n} can be omitted), $b_{f_k}.c_k$ denotes the k-th class form the source database b_{f_k}, and w_k denotes the k-th condition. The view-creation algorithm for processing the above statement is proposed as follows.

The view-creation algorithm:

Input: A data warehouse W (C, V, I, O) and a view -creati statement for creating a view WV.

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

Step 1: For every class $b_{f_k}.c_k$ in WV, do the ***class-integration*** procedure, which is used to integrate the class $b_{f_k}.c_k$ in the class set C. (Its procedure is stated below)

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

Step 3: Set *mc* in the meta-object Meta-WV as all the classes $b_{fk}.c_k$'s in WV.

Step 4: Collect all the source datab ases 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 from 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, $1 \le i \le n$, $1 \le t \le k$, $1 \le l \le m$, 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. Rea 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 sub-steps:

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

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

Step 6-4 (c). Form the following query statement Q_{bf}^{m} 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 $m < i$ (i is the number of items in th

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 through the communication network.

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

Step 9: Check whether the instance identifiers exist in I of the warehouse W. Let D denote the set of instance identifiers from the source databases and not currently in I.

Step 10: Request the data collec tor to retrieve the instance contents of D through the communication network.

Step 11: Receive the instances with their contents retrieved from the data collector. Denote them as P.

Step 12: Check whether the instances in P satisfy all the dependence condition sentences 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 referrin instances; dd their instance identifiers (with the source name b_f) into the attribute mi of the meta-object Meta-WV.

Step 15: Add WV to V in warehouse W.

After Step 15, the data warehouse contains all the desired instances, the new view definition WV, and the new meta-object. In Step 1, the Class-Integration Procedure is used to handle the integration of the multiple classes in the multiple source databases. I had bee stated and described in [2].

4.2 View Deletion

A view WV existing in the object -oriented data warehouse can be deleted using the following statement

Delete Warehouse View *WV*.

In the above statement, WV denotes the view to be deleted in the object-oriented data warehouse. The view-deleti algorithm for processing the above statement is proposed as follows.

The view-deletion algorithm:

Input : A data warehouse W (C, V, I, O) and a view-deletion statement for deleting view WV.

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

Step 1. Search the data warehouse W for view WV; If view WV exists in W, do the next step; otherwise, set $W' = W$ and exit the algorithm.

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

Step 3. Read the *m-th* item mc_m (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 O of the data warehouse W. If the class *cid* is used by some other views, do nothing; Otherwise, remove the class *cid* and all the instances inheriting from the class *cid*.

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

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

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

Step 8. Read the *m-th* item mi_m (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 *ti* is used by the other meta-objects in O of the data warehouse W. If the instance *ti* is used by other views, do nothing; Otherwise, remove the instance *ti* from I of the data warehouse W.

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

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

Step 12. Remove WV from V and remove the meta-object Meta-WV from O in the warehouse W.

After Step 12, the data warehouse removes the view definition WV, the meta-object *Meta-WV*, and all the unused classes and instances from the data warehouse. Moreover, the OLAP supporting abilities which involves drill-down/roll-up operations of this model has been described in [2].

5. Conclusion

The research of object -oriented data warehousing is quite new and promising. Many important issues still remain unexplored and deserve further investigation. In this paper, we have extended our previous data model for a single source to multiple-source environments. The meta-objects are presented to make the management and maintenance of views in the object-oriented data warehousing easy and efficient. Moreover, two incremental view-related algorithms, including view-creation and view -deletion, have been proposed to keep the data desired in an object-oriented data warehouse. In the future, we wiil attempt to design other data models to make the object-oriented data warehousing flexible and efficient in different application domains.

References

[1] S. Chaudhuri and U. Dayal, " An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record, vol. 21, no. 1.

[2] W. C. Chen, Object-Oriented Data Warehousing and its Maintenance Technologies, Master Dissertation, I-Shou University, Taiwan, R.O.C., April 1999.

[3] W. C. Chen, T. P. Hong and W. Y. Lin, "View Maintenance of Object-Oriented Data Warehousing Using the Composite Model," accepted and to appear in The Fifth International Conference on Information Systems Analysis and Synthesis, Orlando, U.S.A, 1999.

[4] W. C. Chen, T. P. Hong and W. Y. Lin, "A Compressed Data Model in Object-Oriented Data Warehousing," accepted and to appear in The IEEE 9 International Conference on Systems, Man and Cybernetics, Japan, 1999.

[5] W. C. Chen, W. Y. Lin and T. P. Hong, "View Update in Object-Oriented Data Warehousing usin Uncompressed Data Model," accepted and to appear in The Fourth World Conference on Integrated Design and Process Technology, Kusadasi, Turkey, 1999.

[6] W. H. Inmon and C. Kelley, *Rdb/VMS: Developing The Data Warehouse,* QED Publishing Group, Boston, Massachusetts, 1993.

[7] W. Kim, "Modern Database Systems", ACM Press, New York, New York, 1995.

[8] Y. G. Ra and E. A. Rundensteiner, "A Transparent Schema-Evolution System Based on Object-Oriented View Technology", IEEE Transaction on Knowledge and Data Engineering, Vol. 9, No. 4, 600-624.

[9] E. A. Rundensteiner, "A Methodology for supportin Multiple Views in Object -Oriented Databases", Proceedings of 18th International Conference on Ver Large Databases 1992, Vancouver, Canada, 187-198.

[10] M. H. Scholl, C. Laasch, M. Tresch, "Updateabl Views in Object-Oriented Databases", Second International Conference on Deductive an Object-Oriented Databases, Munich, Germany, 1991, 189-207.

[11] Y. Zhuge and H. Garica-Molina. "Graph structured views and their incremental maintenance," *Proceedings of the International Conference on Data Engineering*, Orlando, FL, 1998.