The Novel Model of Object-Oriented Data Warehouses

J. C. Shieh and H. W. Lin
Department of Informatics
Fo Guang University
160, Linwei Rd., Jiaushi, I-Lan, Taiwan
jcshieh@mail.fgu.edu.tw

Abstract

With the variety of analysis tools and heterogeneous data sources, it is getting more important to apply object-oriented techniques to approach data warehouse designs. In this paper, we propose a novel data model for constructing the object-oriented data warehouses from various data sources. The model generates the classes of the objected-oriented data warehouse according to the views of users' requirements and the original structures. Especially, it preserves the original inheritance relationships of objects, which has never been done well before.

Keywords: Object -oriented data warehouse, inheritance hierarchy.

1. Introduction

A data warehouse (DW) is defined as a subject oriented, integrated, nonvolatile, and time variant collection of data in support of management decision-making process [20]. In applications, it is a central repository of significant data collected from an enterprise's various business systems. According to the requirements of users' decision-makings, a data warehouse can aggregate data from heterogeneous data sources with different points of views.

The dimensional fact mode (DFM) [11] was first proposed to develop a complete and consistent methodology for designing data warehouses in relational databases environments. In most relational database management systems, the multidimensional data warehouses are constructed by star schemes [13] with a set of dimension tables and a central fact table. Dimension tables are strongly de-normalized and used to select the facts of interest based on the user queries [10] by data cubes. In the relational DW, we can regard data cubes as the view sets. In order to reduce the overall response time, materialized views [12, 8] are commonly used to pre-compute the information that can be useful to answer frequent queries. Because of storing the extracted data in the data warehouse, it is necessary to consider that the consistency of data and view maintenance [2, 21, 22].

In the past, researches on data warehouse primarily focus on relational data models. By the data multiplications, users' requirements are more complicated to

spark off reduplicating data cubes. It is not only to add the cost of building relational data warehouses but also to increase the difficulty of maintaining the consistency of relational data warehouses. Recently the data warehouse researchers have realized the advantages [5] of applying the object-oriented approach to build data warehouses. Based on object-oriented software engineering [4], an object-oriented approach to transform a start schema of a legacy data warehouse into a dimensional object model [7] was proposed [1]. Some scientific literatures [3, 6, 14] about the design of object-oriented data warehouses only pay efforts on specific issues. However, the complete framework of constructing object-oriented data warehouses is absent. Chen, Hong and Lin [16, 17, 18, 19] had proposed data models to transform the data stored in object-oriented databases to the object-oriented data warehouses. But all the above literatures were without retaining the inheritance relationships from data sources to the data warehouses and pondering on the roles of applied methods. In this paper, we will propose a novel framework to construct object-oriented data warehouses with "inheritance" property and "include" relationship reserves. Section 2 points out the problems of current object-oriented data warehouse designs. Our model is introduced in Section 3. The last section comes with the conclusions.

2. Current Data Warehouses

With the complicated real environments, relational data models cannot completely sustain some advanced internal data structures directly, such as objects composed of other objects, multi-valued attributes, set-valued attributes, and the relationship of generalization and specialization. Consequently, an appropriate extension of object-oriented approaches makes data warehouse with flexibility, extensibility, and reliability. It emphasizes the independence between objects so that the corresponding data warehouse can provide multi-combinations to fit in with different requirements. To cite an instance, when we draw out a line and a circle with the length of seven centimeters and radius of three centimeters respectively from data source to data warehouse, we can form a new cylinder object easily to satisfy the query of portico. As a result, the notion of object-oriented components enables the data warehouses to get information rapidly and elastically.

2.1 Uncompressed Data Model

In [15] [16], the uncompressed model was proposed to store the materialized views of users' requirements in an object-oriented data warehouse that maintaining the original structures. This model replicates all the attributes and relations of necessary classes and instances from the object-oriented data sources based on the structures of views. The views are defined in the data warehouses and the related

objects are reproduced from the original object-oriented data sources.

Let C be the set of classes defined in the source databases. That is $C=\{c_1, c_2, ..., c_n\}$, where c_i is a class, $1 \le i \le n$. Let *ID* be a set of identities, *A* be a set of symbols called attribute names, *T* be a set of data types allowed for *A*, and *M* be a set of processing methods. A class in an object-oriented database can be formally defined as follows:

Definition 2.1.（CLASS）:

A class c is quadruple $\{cid, ca, ct, cm\}$, where $cid \in ID, ca =< ca_1, ..., ca_n >$, with $ca_i \in A$ and $1 \le i \le n$, $ct =< ct_1, ..., ct_n >$ with $ct_j \in T$ and $1 \le j \le n$, and $cm \subseteq M$.

By referring to a class and inheriting characteristics from the classes, an instance is created. Similarly, each instance is associated with a unique instance identifier, a set of attributes, and a set of procedures called method.

Definition 2.2. （Instance）:

An instance $t = \{tid, ta, tv, tm, cid\}$ is created and inherits from a certain class $cid = \{cid, ca, ct, cm\}$ such that $tid \in ID, ta = ca, tv = \{tv_1, tv_2, ..., tv_n\}$, with $tv_i \in U$ U be a set of value, and tv_i being of type ct_i $1 \le i \le n$, and *tm*=*cm*.

The relationships among the classes and instances in the example can be represented by a graph as shown in Figure 1, where a shaded circle represents a class, a circle represents an instance, a solid line links an instance to its attribute and a dashed line stands for an instance generated from a class.

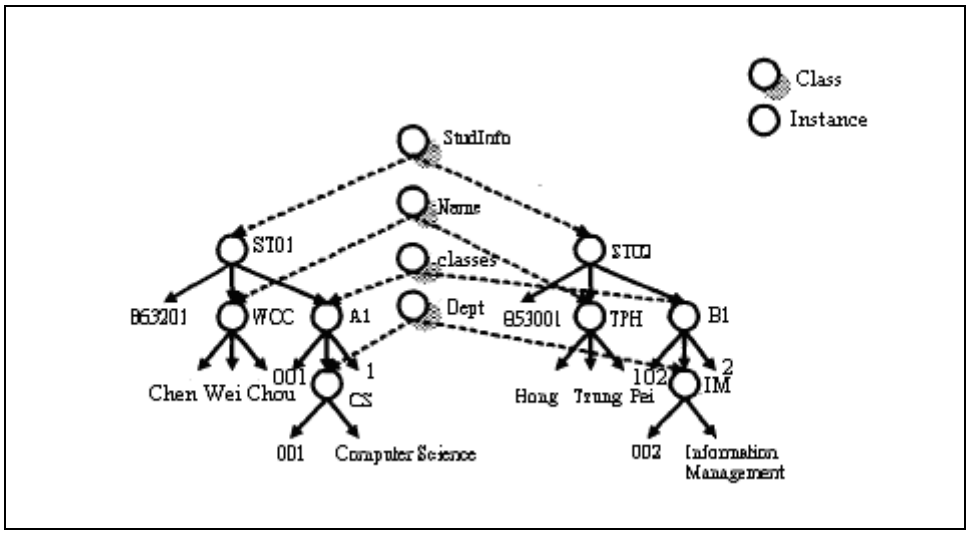

Figure 1. A graphical representation of relationship among classes and instances in the data source

A unique view identifier, a set of attribute names, a set of attribute types and a query sentence characterize a view. The number of attributes is equal to that in the query sentence. Formally, a view can be defined as follows:

Definition 2.3. （**View**）：

A view v in an object-oriented data warehouse is a quadruple $\{vid, va, vv, vs\}$ such that $vid \in ID, va = \{va_1, va_2, ..., va_n\}$ with $va_i \in A$ and $1 \le i \le n$, $vv \subseteq T$, and vs is a query statement (**Select** S, **From** F, **Where** W), where $S =< s_1, s_2, ..., s_n >$ with $s_i \in A$ and $1 \le i \le n$, $F \in C, W$ denotes the query conditions, and $|va| = |S|$.

Figure 2 gives an example of two view definitions, *FreshMan* and *ClassList*.

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

View ClassList ( DeptName char(40), ClassID char(5) )
as {
Select
      DeptOf.DeptName,
      ClassID
From Classes
Where DeptOf.DeptName = "Information
Management";
}
```

Figure 2. An example of views' definitions

After the views are defined, the attributes and the relationships of the necessary classes and instances (according the views) with their original structures are copied from the data sources to the object-oriented data warehouses.

Definition 2.4. （**Uncompressed Data 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 views, and I is the set of instances generated according to C.

Figure 3 shows an example of object-oriented data warehouse with uncompressed data model. According to the views, as shown in Figure 2 and the data source citing an instance in Figure 1, there are only six instances that satisfy the view definition conditions. These six instances are thus copied to the data collector, and saved in the object-oriented data warehouse.

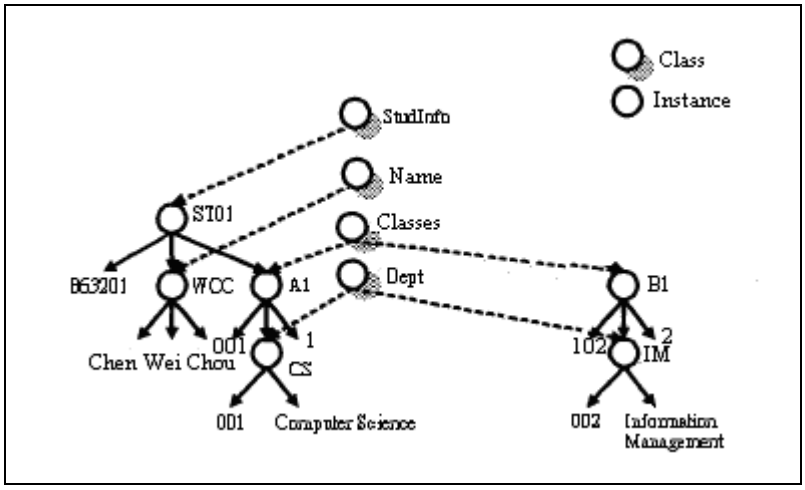

Figure 3. A graphical representation of relationships among classes and instances in the data warehouse

The uncompressed data model can easily provide object-oriented data warehouse clearly when any modification is needed. Since the results of queries are needed to recalculate, its query performance is limited. In other words, the uncompressed model can only maintain the relationships among the instances easily and keep the completeness of data utterly, it has to face up to the fact of poor response time.

2.2 Compressed Data Model

The compressed data model [17, 18] combines all the attributes of different classes defined in a view into a single newly generated class. The instances are produced and stored in the data warehouse according to the new class.

Definition 2.5. **（Compressed Data Warehouse）：**

An object-oriented data warehouse W is a triple { V, VC, I }, where V is the set of view definitions, VC is the set of classes, $VC = \{c_1, c_2, ..., c_n\}$,where c_i is a new class from a certain view, $1 \le i \le n$; I is the set of instances generated according to VC and V .That is, $I = \{i_1, i_2, ..., i_m\}$ where i_i is an instance kept in the data warehouse, $1 \le i \le m$.

As above mentions, the data warehouse creates new classes to represent the satisfied the views. Each new class is named by a string beginning with the character ”C” and concatenated by its view identifier. The attributes of new classes are formed in the definition of the views, *va* and *vv.* As shown in Figure 4, two new classes in the data warehouse, *CFreshMan* and *CClasses* are derived from the two views in Figure 3. The class *CFreshMan* corresponding to the *va* of the *FreshMan* view has three attributes, DeptName, FirstName, and LastName.

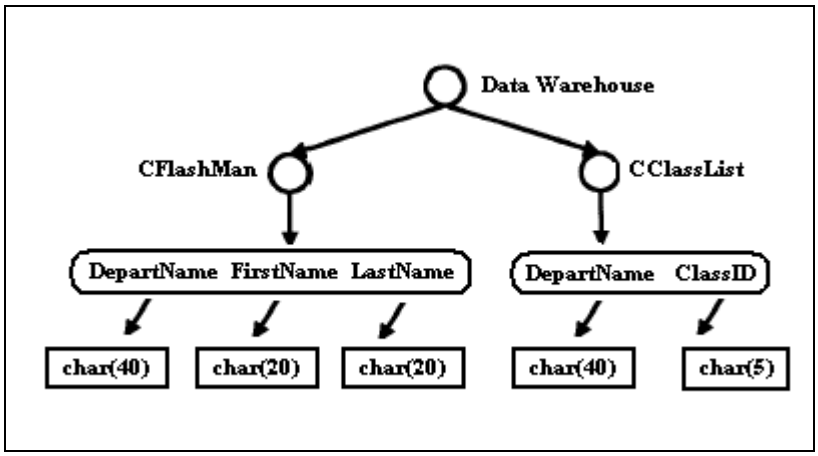

Figure 4. A graphical representation of data warehouse classes

After the new classes are created through a data collector, the data warehouse acquires the needed data to generate their corresponding instances. The class identifier plus "_" and the identifier of the instance name the new instance in the data source. The graphic representation of the data warehouse is shown in Figure 5:

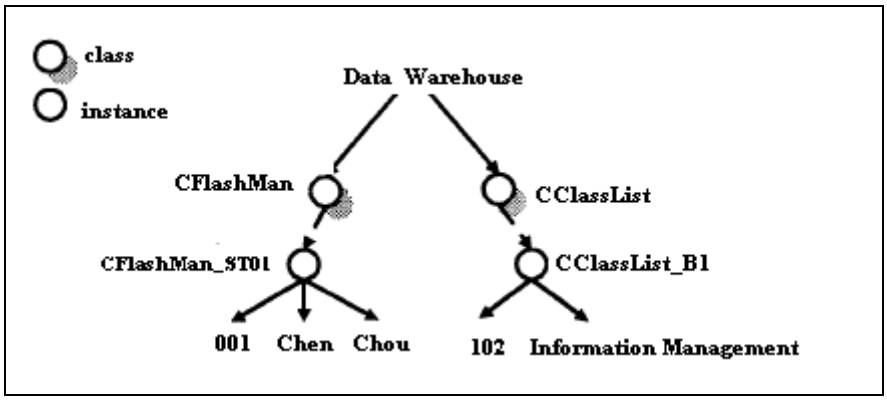

Figure 5. A graphical representation of compressed data warehouse

The performance of its queries will be better than that of the uncompressed model since the classes and instances are stored according to the views' definitions. Since the referential relationships of the classes are hidden in the data warehouse, the compressed model does not record the relationships of original classes and has a higher security than the uncompressed data model. Comparatively, the procedure of maintaining views with the compressed data model is a little more complex than that with the uncompressed data model since the instances are more complicated.

2.3 Composite Data Model

The composite data model [19] has the advantages of both previous data models.

The data warehouse not only stores instances that are according to the new classes but also copies necessary class structures from data sources.

Definition 2.6. （Composite Data Warehouse）:

An object-oriented data warehouse *W* is a quadruple {*V*, C, *VC*, *I*}, where *V* is the set of view definitions, *C* is the set of classes defined in the source database, *VC* is a set of classes, and *I* is the set of instances sent from data sources and generated from the source databases according *VC, V* and *C* .

Assume that the "select" part of view *FreshMan* definition in Figure 2 are changed as below:

Select

StudClass.ClassID

StudName.

According to the composite data model, all the attributes of different classes defined in a view are copied into the specific class and partially keep necessary classes hierarchy relationships, as shown in Figure 6.

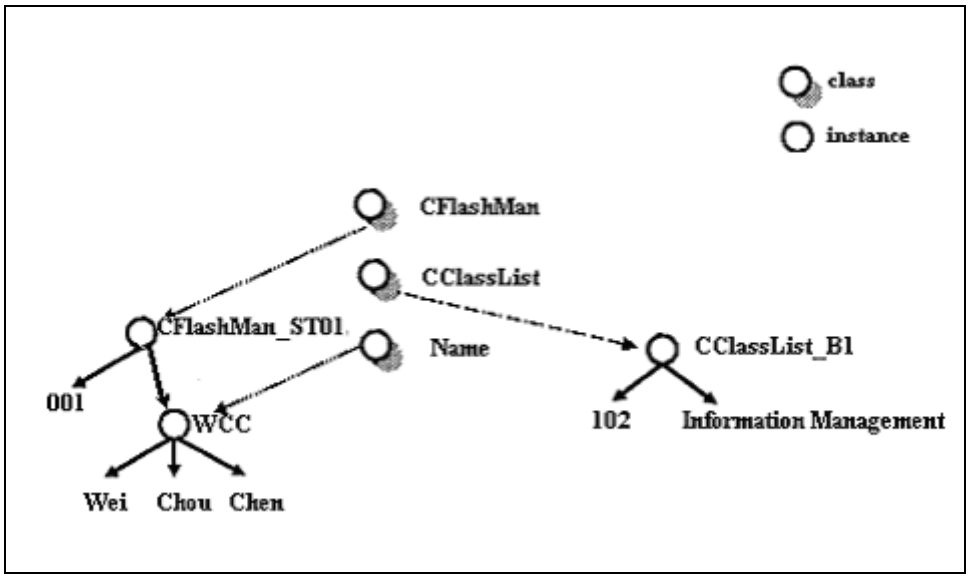

Figure 6. A graphical representation of composite data warehouse

The query performance and security are equal to the compressed model since the classes and instances are stored with respect to the views' definitions. Moreover, the data model keeps the original structures partially; the data completeness and maintenance performance are thus better than that of the compressed data model.

Undeniably, the object-oriented data warehouses proposed by Chen, Hong and Lin [15, 16, 17, 18, 19] really provide object-oriented environments with innovative and serviceable frameworks. Nevertheless, none of the data models can cope with the inherited data well. For real application considerations, it is impossible to ignore such important role which "inheritance" plays. Here is an inheritance example of an

object-oriented database schema as depicted in Figure 7. In the figure, each node represents a class. A node is sub-divided into three levels containing the name of the class, the attributes and the methods respectively. The methods and attributes in the rectangles are inherited from super-classes by means of the bold arcs. Evidently, the previous data models of constructing object-oriented data warehouses are incapable of dealing with the inheritance issues caused by "*Researcher_Student*" class. Thus, the models will result in incomplete object-oriented data warehouses. In the next section, we will propose a novel one to resolve the problems.

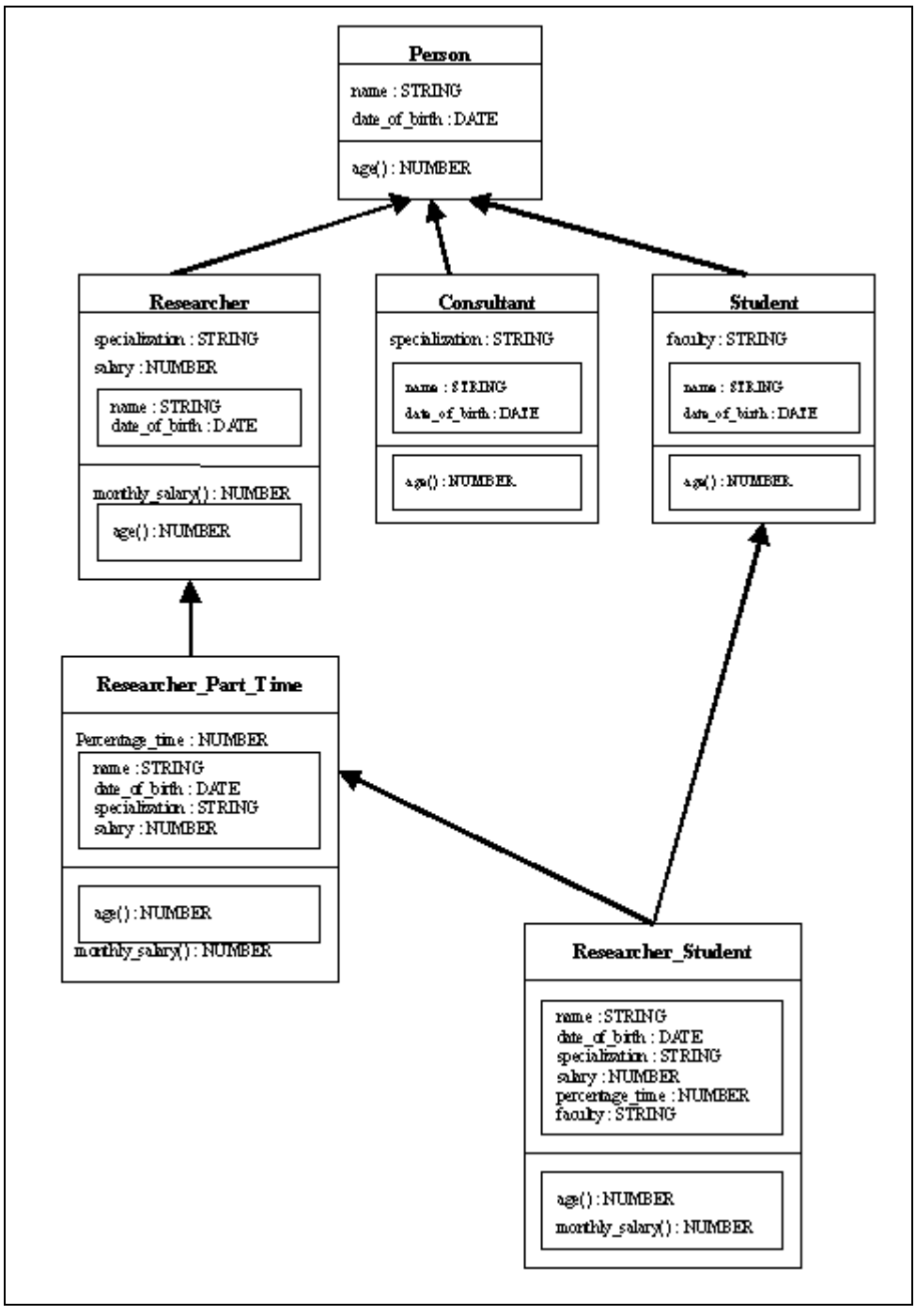

Figure 7. An example of inheritance hierarchy in the data source

3. The Novel Data Model

Let C be the set of classes defined in the source databases. That is $C=\{c_1, c_2, ..., c_n\}$, where c_i is a class, $1 \le i \le n$. Let C*ID* be a set of class identities, *A* be an attribute set of class set, *T* be a set of data types allowed for *A*, and *M* be a set of processing methods, CP is a set of parent IDs'. We formally define a class in an object-oriented database as follows:

Definition 3.1.（Class）:

A class c is quintet$< cid, ca, ct, cm, cp >$, where $cid \in CID$; $ca = ca_i \, \mathrm{Y} \, CAI$ with $ca_i \in A \quad 1 \le i \le n$, and $CAI = \{\mathrm{Y} attributes\_of\_CP\}$, $ct = ct_j \, \mathrm{Y} \, CAT$ with $ct_j \in T$ and $1 \le j \le n$, and $CAT = \{\mathrm{Y} attribute\_type\_of\_CP\}$, Y means "combination"; $cm = cm_i \, \mathrm{Y} \, CMI$, with $cm_i \in M \quad 1 \le i \le n$, and $CMI = \{\mathrm{Y} methods\_of\_CP\}$ $cp = \{cid_1, cid_2, ...cid_n\}$ with $cid_i \in CID$ and $1 \le i \le n$.

Example 3.1.

As shown in Figure 7, there are six classes, which are *Person, Researcher, Consultant, Student, Researcher_Part_Time* and *Researcher_Student*. That is *CID= {Person, Researcher, Consultant,Student, Researcher_Part_Time, Researcher-_Student}, A={{name,date_of_birth}, {specialization,salary}, {specialization}, {faculty}, {percentage_time}, {} }.* As the class "*Person*" in this example, *cid = Person, ca= {name, date_of_birth}, ct= {STRING, DATE}, cm= {age()}* and *cp={}*; in addition, the class *Researcher*, *cid = Researcher, ca_i= {specialization, salary}* and *CAI={name, date_of_birth} , therefore ca={specialization, salary, name, date_of_birth}, ct= {STRING, NUMBER, STRING, DATE}, cm_i= {monthly_salary()}, CMI= {age()}, CM= {monthly_salary(), age()}* and *cp={Person}*.

Similarly, by referring to a class, each instance is created and inherits the characteristics from it.

Definition 3.2. （Instance）:

An instance $t =< tid, cid, ta, tv, tm >$ is created and inherits characteristics from a class $cid =< cid, ca, ct, cm, cp >$ such that $tid \in TID, ta = ca, tv = \{tv_1, tv_2, ..., tv_n\}$, where $tv_i \in U$, *U* be a set of value, and tv_i being of type ct_i, $1 \le i \le n$. That is, tv_i denotes the value of attribute ca_i and if ca_i is a simple attribute, then tv_i consists of a set of elements from *U;* or if $ca_i = ca_{i1}, ..., ca_{in}$ is a composite attribute, then tv_i

consists of a set of the form $[tv_{i1},...,tv_{in}]$, where each component $tv_{ik}, i1 \le ik \le in$ is defined as 1, and *tm=cm*.

Example 3.2.

There are two instances created by referring to the class "*Person*". One is called PO_1 with attribute values *(Mike, 1973/03/11)*, that is, *tid= PO_1, cid=Person, ta= ca= {name, date_of_birth}, tv= {Mike, 1973/03/11}*. The other is called PO_2 with values (*Susun, 1983/05/21*). Thus, there are three instances RO_1, RO_2, RO_3, referring to the class "*Researcher*"; two instances, CO_1, CO_2, referring to the class "*Consultant*"; four instances, SO_1, SO_2, SO_3, SO_4, referring to the class "*Student*"; three instances, $RPTO_1, RPTO_2, RPTO_3$, referring to the class "*Reserarcher_Part_Time*"; two instances, RSO_1, RSO_2, referring to the class "*Researcer_Student*". Similarly, the RO_1, RO_2, RO_3 are with attribute values *(Michael, 1953/01/17, Computer, 40000)*, *(Robert, 1963/02/15, Finance, 37000), (Peter, 1975/12/25, Finance, 23000)* respectively. Continuously, the values of $RPTO_1, RPTO_2, RPTO_3$, are *(Michelangelo, 1973/01/11, Computer, 20000, 70%)*, (*Christina, 1975/01/01, Computer, 25000, 90%)*, *(Floe, 1972/11/21, Computer, 10000, 20%)*. Finally, two instances, RSO_1, RSO_2, are created as below:

<RSO_1, Researcher_Student, {name, date_of_birth, specialization, salary, percentage_time, faculty}, {May, 1980/11/21, Computer, 25000, 100%, MIS}, {age(),month_salary()}>

<RSO_2, Researcher_Student, {name, date_of_birth, specialization, salary, percentage_time, faculty}, {Amy, 1979/05/21, Computer, 10000, 20%, IE}, {age(),month_salary()}>

Definition 3.3. （**View**）：

A view *v* in an object-oriented data warehouse takes on as the form:

View *vid*

{ **Select** *S*

From [only] *F* | all *F*[(except [all] F')]

[**Where** *H*]};

where $vid \in VID$, $S =< s_1, s_2, ..., s_n >$ with $s_i \in ca$ and $1 \le i \le n$, $F \in C, F' \in C$, *H* denotes the query conditions.

If we designate '**all'** in the '**From'** sentence, it signifies that we should search not only F but also its subclass. By contraries, '**only**' means that we should select F purely. On the side, if we desired to select from *all F* but a subclass is in the inherited hierarchy, '**except**' is added. It deserves to mention that the instances that meet the "**Where**" conditions for views must be drawn out *under the local class hierarchy*.

Example 3.3.

View *Count_Salary*

{**Select** *name*, *salary*, *percentage_time*

From all *Person*

Where *salary*>25,000};

The '*salary*' attribute is the local attribute in the '*Researcher*' class. In other words, the instances that we have selected from the data source should be the instances inherited from the local class, "*Researcher*", or from the subclasses, "*Researcher_Part_Time*" and "*Researcher_Student*". In this example, the instances RO_1, RO_2, $RPTO_2$, RSO_1 satisfied the conditions are selected.

After having the view's definition, the data warehouse can be constructed as follows:

Definition 3.4. （Object-Oriented Data Warehouse）:

An object-oriented data warehouse W is a triple < V, I, VIC >. Let V be the set of view definitions; I be the set of instances generated from a certain view, and VIC be the set of new classes referring to the views selected from the attributes and original structures.

The difference between Chen et al.'s model and the proposed model is what the roles the selected instances play. Whether a compressed or a composite data model is proposed, the data warehouse is built un-completely until the new classes' attribute values are copied by the data collector from the data sources to the data repository. That is, the first process is to create all new classes by congregating attributes that come from different classes referenced by selected views. Then, using these new classes, the data collector can filter and copy the relative instances' attribute values from the data sources. Obviously, the new classes or instances created from the data warehouse are more complex and haphazard. However, our proposed data warehouse instead makes use of the "**Where**" conditions of the views to select the qualified instances initially and then employs the selected instances to match the original class structure of each instance. Thus we can construct a new inherited architecture to meet every kind of view. We conclude the following steps to construct it:

Step 1: Find out the instances that satisfy the where conditions.

Step 2: List all attributes that the instances select.

Step 3: According to the classes that the instances belong to, classify these instances and assign each attribute by inherited structures to construct the relationships of classes in the data warehouse.

Step 4: Store the values of the instances in accordance with new hierarchies.

Example 3.4.

In Example 3.3, the instances RO_1, RO_2, $RPTO_2$, RSO_1 satisfy the selected "**Where**" conditions. So there are four instances and twelve attributes (RO_1.*name*, RO_1.*salary*, RO_1.*pertantage_time*, RO_2.*name*, RO_2.*salary*, RO_2.*pertantage_time*, $RPTO_2$.*name*, $RPTO_2$.*salary*, $RPTO_2$.*pertantage_time*, RSO_1.*name*, RSO_1.*salary*, RSO_1.*pertantage_time*) in the data warehouse. Notice that some attributes' values may be 'NULL'. In order to keep the inherited relationships, the data warehouse is built by referring to the hierarchical frameworks in the data sources. If the value of instance attribute is not 'Null', we congregate the attributes in the super-classes. Similarly, the attributes of the subclasses are massed except the attributes of parents'. Therefore, the local attributes in the "*C_Researcher*" class are *nam*e and *salary*; the attribute in the "*C_Researcher_Part_Time*" is *production_bouns* that in the "*C_Researcher_Student*" is 'NULL'. Figure 8 shows the inheritance hierarchy in the data warehouse.

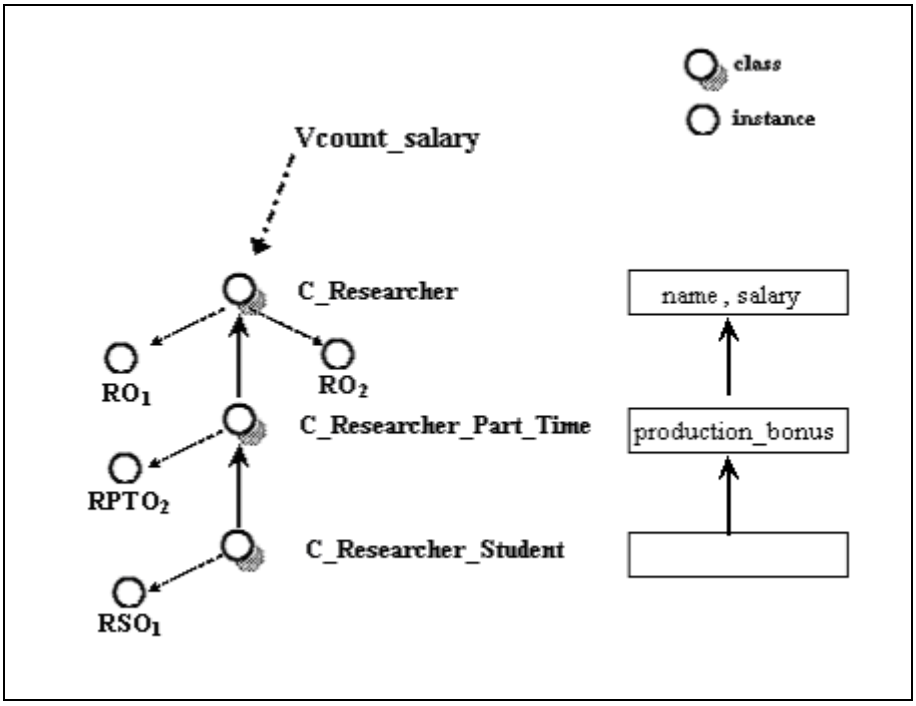

Figure 8. An example of the inheritance hierarchy in the data warehouse

The novel model can manifest the hierarchy in the data warehouse to make complicated data well regulated. Through redefining the relationships of classes and their attributes, the proposed data warehouse deals with not only inheritance hierarchies but also the issues of derived attributes.

4. Conclusion

In this paper, we have proposed a novel data model for constructing an object-oriented data warehouse importantly to preserve the original inherited hierarchies of the data sources. It also makes the data warehouses concise and simple

thus improves the efficiency in data retrieval. We expect that the model can built the object-oriented data warehouses more complete.

Reference

[1] Aluizio Haendchen Filho, Hercules A. Prado, Simao S. Toscani, ”Evolving a Legacy Data Warehouse System to an Object-Oriented Architecture”, The International Conference on Computer Science Society of the Chilean 2000, pp.32-40

[2] D.A., A.EI, A.Abbadi Singh and T.Yurek, “Efficient View Maintenance at Data Warehouses”, ACM SIGMOD AZ USA, 1997.

[3] H.A..Kuno and E.A. Rundensteiner, “Incremental Maintenance of Materialized Object-Oriented Views in MultiView : Strategies and Performance Evalution”, IEEE Transactions on Knowledge and Data Engineering, Vol.10,No.5, Oct.1998.

[4] Ivar Jacobson, “Object-Oriented Software Engineering: a Use Case Driven Approach”, Addison-Wesley, New York, 1992.

[5] Joseph M. Firestone, “A system approach to Dimensional Modeling in Data Marts”, Executive Information Systems, Inc., 1997. **http://dkms.com/DataWarehousing.htm**

[6] Joseph M. Firestone, “Object-Oriented Data Warehousing”, White Paper No. 5. Executive Information Systems, Inc., 1997. **http://dkms.com/OODW2.htm**

[7] Joseph M. Firestone, “Dimensional Object Modeling”, Executive Information Systems, Inc., 1998. **http://dkms.com/DOM.htm**

[8] J. Yang, K. Karlapalem, Q. Li, “A Framework for Designing Materialized View in Data Warehousing Environment”, IEEE, May, 1997.

[9] M. Golfarelli, D. Maio, and S. Rizzi, “Conceptual design of data warehouse from E/R schemes”, Proc. Hawaii International Conference on System Sciences-31, VII, Kona, Hawaii, 1998, pp334-343.

[10] M. Golfarelli, D. Maio, and S. Rizzi, “The Dimensional Fact Model: A Conceptual Model for Data Warehouse”, Invited Paper, International Journal of Cooperative Information Systems, vol. 7, n. 2&3, 1998.

[11] M. Golfarelli, S. Rizzi, “A Methodological Framework for Data Warehouse Design”, Proceedings ACM First International Workshop on Data Warehouse and OLAP（DOLAP）, Washington, 1998.

[12] M. Indulska, “Shared Result Identification for Materialized View Selection”, IEEE, July, 1999.

[13] R. Kimball, “The data warehouse toolkit”, John Wiley & Sons, 1996.

[14] William A. Giovinazzo, “Object-Oriented Data Warehouse Design”, PH PTR ISBN 0-13-085081-0, April 2000.

[15] W. C. Chen, T. P. Hong and W. Y. Lin, "View maintenance in an object-oriented data warehouse", Proceedings of the Fourth International Conference on Computer Science and Informatics, North Carolina, USA, 1998, pp353-356.

[16] W. C. Chen, W. Y. Lin and T. P. Hong, "Object-oriented data warehousing and its incremental view maintenance", Proceedings of the Ninth Workshop on Object-Oriented Technology and Applications, Kaohsiung, Taiwan, 1998, pp139-144.

[17] W. C. Chen, T. P. Hong and W. Y. Lin, "Using the compressed data model in Object-oriented data warehousing", IEEE SMC'99 Conference on proceedings 1999, pp768-772.

[18] W. C. Chen, T. P. Hong and W. Y. Lin, "Three maintenance algorithms for the compressed object-oriented data model", International Journal of Computers and Applications, 2000.

[19] W. C. Chen, T. P. Hong and W. Y. Lin, "A Composite Data Model in Object-Oriented Data Warehousing", The 31st International Conference on Technology of Object-Oriented Languages & Systems, 1999, pp. 400-406.

[20] W.H. Inmon. "Building the Data Warehouse", Second Edition, Wiley Comp, ISBN n 0471-14161-5, USA, 1996.

[21] Y. Zhuge, H. Garcia-Molina and J.L. Wiener, "The Strobe Algorithms for Multi-Source Warehouse Consistency", Proc. Conference on parallel and Distributed Information Systems, Miami Beach, FL, 1996.

[22] Y. Zhuge, H. Garcia-Molina and J.L. Wiener, "Consistency algorithms for multi-source warehouse view maintenance", Journal of Distributed and Parallel Databases, Vol. 6, No.1, 1998, pp7-40.