Information Storage and Retrieval Schemes for Recycling Products

Takayuki Tsuchida
Graduate School of Engineering, University of Fukui
3-9-1 Bunkyo, Fukui City, 910-8507, Japan
tsuchida@pear.fuis.fukui-u.ac.jp

Teppei Shimada, Tatsuo Tsuji and Ken Higuchi
Graduate School of Engineering, University of Fukui
3-9-1 Bunkyo, Fukui City, 910-8507, Japan
{shimada, tsuji, higuchi}@pear.fuis.fukui-u.ac.jp

Abstract— **A recycling object is in its recycle chain and its status may often change according to recycling stage or reusing status; even its structure could often dynamically change according to its reusing demand. Capturing the structure of recycling objects with a single uniform schema definition is hard and the conventional database management system would be inadequate to deal with this situation. Moreover database management system employed in a recycle management system is better to provide the capability of not only storing and retrieving information but also that of *OnLine Analytical Processing* (OLAP) especially along the *time axis*. OLAP capability serves as an aid of better decision making for recycling manufactures or dealers. This paper considers several requirements needed for recycling management systems, especially on their information storage architecture and retrieval capability for recycling objects. Their realization scheme adapting these requirements will also be discussed. Using concrete examples based on SQL statements, we will demonstrate advantages of our realization scheme. Improving lower level storage organization or internal data structuring strategy for storing recycling objects can provide handling efficiency or retrieval efficiency rather independently of the kind of recycling object. Recycling objects discussed here are assumed to be end products.**

Index Terms—**storage structure, recycling objects, information retrieval, multidimensional data, OLAP**

I. INTRODUCTION

3R(*Reduce*, *Reuse*, and *Recycle*) is a very important principle for contributing toward a sustainable recycle-oriented society. *Reduce* means waste minimization, which is the process and the policy of reducing the amount of waste produced by a person or a society. *Reuse* is using an object more than once. This includes conventional reuse where the object is used again for the same function, and new reuse where it is used for a new function. In contrast, *Recycle* is the breaking down of the used object into raw materials which are used to make new objects.

In recycle management system (RMS) to support these 3R especially in *reuse* step, a powerful and efficient information management system is strongly desired. There have been many research efforts concerning high level organization strategies of RMS or information system supporting RMS[1][2]. On the contrary, little research have been addressed on lower level organization of the information management systems concentrating on the internal data storage architecture suitable for handling recycling objects. [3] employs bar code to administrate information of cell phones for recycling them. [4] proposes a multi-layered quality data model framework including physical data layer to support the product quality control in lifecycle and its evolution course. But , in these papers efficient storage organization strategy for recycling objects are not focused on.

While high level organization strategy such as described in [1][2] should be heavily dependent on the kind of recycling objects to be managed, improving lower level storage organization or internal data structuring strategy for recycling objects can afford handling efficiency and retrieval efficiency rather independently of the kind of recycling object.

This paper considers some requirements needed for recycling management systems, especially on their information storage architecture and retrieval capability of recycling objects. Their realization scheme adapting these requirements will also be discussed with concrete examples. Recycling objects discussed here are assumed to be end products.

We assume the situation where each recycled product is identified by unique identifier called product ID (pid). This product ID is distinguished in each recycle chain and handled in an integrative management system which is designed for retail shops, recycling manufactures, or lease companies.

This paper is based on "Considerations on Information Storage and Retrieval Systems for Recycling Objects", by T. Tsuchida, T. Tsuji and K. Higuchi, which appeared in the Proceedings of 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2007), Montreal, Canada, October 2007, ©2007 IEEE.

© 2008 ACADEMY PUBLISHER

II. Requirements of Information Storage Architecture and Retrieval Capability

In ordinary situation, recycling management system often employs a commercial database management system (DBMS). Especially relational DBMS is used to keep the various properties inherent in the recycling objects. These properties, e.g., *creation time*, *color*, *weight*, are defined in the *columns* of a relational table. The values of these properties are gathered into a single record for each recycling object. Each column value occupies fixed size storage and each record is stored on a consecutive area of disk storage in the input order (Fig.1)

Usually the data type of each column, e.g., *integer, character, date,* in a relational table should be defined statically before using the table. Such simple and fixed storage structure of a relational table seems not to be adequate to capture the dynamicity or diversity of recycling objects. In this section we describe some requirements of information storage architecture suitable for handling recycling objects.

PID	OS	CPU	HDD
1020	MAC	Pentium	250
1021	WindowsXP	Pentium	180
1022	Linux	Athlon	250
1023	WindowsXP	Pentium	80

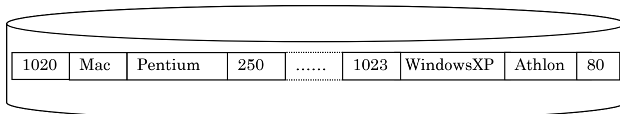


Figure 1. A relational table and its arrangement on secondary storage

A. Handling Dynamicity

A recycling object is in its recycle chain and its status may often change according to its recycling stage or reusing status. A recycling management system can administrate such situation properly and can maintain the information consistency of the recycling objects.

A large number of recycling objects are newly registered and their properties are stored as table records in its database every day; a new record would be inserted on the storage at every addition of a recycling object on RMS. Moreover values of the properties of these table records might be frequently updated according to the status changes of the corresponding objects. These records may continue to exist in the RMS database as *active objects* until the corresponding objects become to be not reused. Therefore many records related to the objects that become inactive would be deleted every day from the RMS database. But even if a recycling object becomes inactive, the information of its recycling history may continue to exist in the archive database for later analysis and mining, which will be discussed later in *E* in this section. Anyway the internal data storage should be organized in order to be adapted to the above stated dynamicity of the recycling objects.

B. Handling Schema Evolution

Properties of a recycling object are not static. When a new kind of object is registered in the RMS DB, its initial set of properties is defined as columns of a relational table associated with the ID that identifies the object uniquely. Such ID is called an *object ID* and referenced by the RMS or users for retrieval purpose. It continues to be valid and is never changed throughout the life cycle of the corresponding object or product. The values of some properties such as *price or retail_shop* may be updated according to the change of the environment of the object. But it should be also noted that not only values of property but also properties themselves may be dynamically changed in the recycling chain.

Now we consider a system for reusing personal computers in a PC lease company. In such a system, indispensable properties of PC, e.g., kind of *CPU* and *operating system*, will be always necessary as *properties*. In contrast, peripheral devices are rather dynamic reflecting the progress of techniques or popularity; e.g. floppy disc drive may be deleted from the set of columns, instead blu-ray disk drive will be added as a new column.

According to these dynamical changes of properties/columns, definition of the corresponding table columns would be dynamically changed. Such situation is generally called *schema evolution.* It includes such as change of column name, change of column type, adding a new column, deleting existing column or adding/deleting index against an existing column. It should be noted that adding new columns and deleting existing columns would cause even the change of table structure. Usually they often result in the total reorganization of the table records in the secondary storage. This operation is very time consuming and sacrifices other real time database operations.

It is an important requirement for RMS DBMS to have an internal storage organizing scheme by which reorganization is not necessary or can be accomplished with small cost against the schema evolution.

C. Handling Semi-structured Data

Commercial products produced according to the same specification have the same initial set of properties. As we stated in *B* in this section, such set of properties is not static, but would be dynamically changed in the recycling chain process. Moreover at the reusing stage some products may often undergo some schema evolution and other set of products may undergo different schema evolution even if these products initially produced according to the same specification. Such difference is depending on their reusing environment.

In consequence, even if commercial products have been produced according to the same specification or industrial standard, the properties of them cannot be captured by a single schema; i.e. a single set of fixed table columns. Instead variety of the schema definitions should be prepared for even the same kind of product, going so far as to say that distinct schema definition becomes necessary for each recycled product. If a conventional relational database is employed in order to

© 2008 ACADEMY PUBLISHER

handle such situation, it may result in "one table for one record" or "a single table with large number of columns that covers all the possible properties". The latter is reflected in Fig.7(a) as an example. This might be unrealistic and lead to very redundant storage organization in realizing these tables.

In order to cope with the above situation, more flexible data schema or metadata definition strategy other than that of strongly typed conventional relational table would be necessitated. Such strategy can define semi-structured data that involves structural difference among data instances. Moreover these semi-structured data can be mapped efficiently to the physical storage and can be handled with good performance.

D. Traceability Along Time Axis

A recycling object or product has its own *recycling history.* From the time of its birth, it often undergoes reproductions or reusing in its recycle chain as the time elapses. RCM should be able to accumulate information of recycling objects along with their *time axis*, and more importantly it should provide the capability to accept the retrieval queries involving *time* in their retrieval conditions that are issued from various viewpoints concerning *time*. Namely, the database management system employed in RCM should have the features of *temporal database* management [10].

In order to respond quickly against the time queries, the internal storage organization should be designed to provide efficient traceability along time axis.

E. Analytical Processing Capability

The scale of databases employed in RCM is usually very large and their data structures are inherently complex. Nevertheless their storage structures should be flexibly organized as stated above. Moreover it is primarily important to guarantee the fast response against the queries from various viewpoints issued from users.

Beyond these basic capabilities, some advanced capability such as online analytical processing (OLAP)[11] is strongly required in DBMS for RCM. It is insufficient only to provide rather passive capability invoked by users' demand of processing in the traditional database management systems. To have analytical processing capabilities from variety of viewpoints greatly contributes to establish better decision making for handling recycling objects. Back to the previous PC lease company example. A great amount of data is accumulated in the DBMS concerning lease history and something like that enough to make a statistical analysis. Under such circumstance, the DBMS is expected to find some knowledge or rules that can support to set up better sales strategy. Construction of a *data cube* can serve to perform statistical analysis efficiently from various view points of data during OLAPing. For an n dimensional relational table R, let C be the set of n columns of R. A value called *measure value* M such as *total sales* or *sold number* is associated with each record in R. For an aggregation function F such as *sum* or *average* operated on M, data cube for R is a set of 2^n -1 *cuboid*s, each of which is a k $(1 \leq k \leq n)$ dimensional dataset aggregated by using F against (aggregated) M for a subset of C.

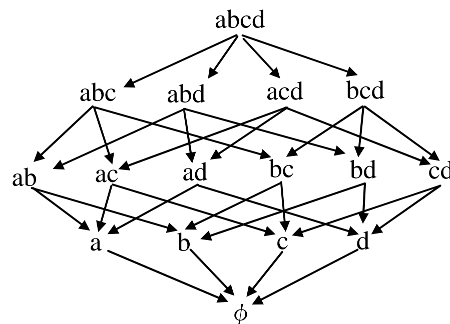


Figure 2. Four dimensional lattice diagram for a data cube

Fig.2 shows a four dimensional data cube, where $C = \{a, b, c, d\}$. For example cuboids *abc, abd, acd* and *bcd* can be aggregated and derived from the base cuboid *abcd,* and cuboids *ab, ac, bc* can aggregated and derived from cuboid *abc*. Thus the set of 2^n-1 *cuboid*s forms a *lattice.* Note that usually construction of a data cube is very time consuming, hence the required cuboids should be precomputed before OLAP operations such as *drill down* or *roll up* in order to analyze OLAP data from various points of view interactively on line. Interested readers can refer to [11][12] for more details on data cubes.

The requirements on the storage organizations for recycling management systems would include efficient handling of data cubes.

Ⅲ. EMPLOYING EXTENDIBLE ARRAYS

In this section, a flexibly extendible data structure and its access method is described. Our storage organization that meets the requirements $A \sim E$ discussed in the previous section will be based on this data structure.

In the conventional implementation of relational tables, each record is placed on secondary storage one by one in the input order as was shown in Fig.1. This arrangement suffers from some shortcomings.

(1) The same column values in different records will have to be stored many times and hence the volume of the database increases rapidly. Such a situation is typical in the columns of categorized value like "blood group", "sex" etc.
(2) In the retrieval process of records of a specified column value, unless some indexes are prepared, it is necessary to load records in the table sequentially in main memory and check the column value. In consequence, the whole table should be fetched into memory. Therefore retrieval time tends to be long.

The implementation using multidimensional arrays can be used to overcome problem (2) above and can perform retrieval not in the sequential manner. Each column of a relational table is mapped to a distinct dimension of the corresponding array. Nevertheless, such an implementation causes further problems:

(3) Conventional schemes for storing arrays do not support dynamic extension of an array and hence

© 2008 ACADEMY PUBLISHER

addition of a new column value is impossible if the size of the dimension overflows.

(4) In ordinary situation, implemented arrays are very sparse.

The concept of *extendible array* we employ will overcome problem (3). It is based upon the index array model presented in [5].

An n dimensional extendible array A has a history counter h and three kinds of auxiliary table for each extendible dimension $i(i=1,...,n)$. See Fig.3. These tables are *history table* H_i, *address table* L_i, and *coefficient table* C_i. The history tables memorize extension history h. If the size of A is $[s_1, s_2,..., s_n]$ and the extended dimension is i, for an extension of A along dimension i, contiguous memory area that forms an $n-1$ dimensional subarray S of size $[s_1, s_2, ..., s_{i-1}, u\ s_{i+1},..., s_{n-1}, s_n]$ is dynamically allocated. Then the current history counter value is incremented by one, and it is memorized on the history table H_i, also the first address of S is held on the address table L_i. Since h increases monotonously, H_i is an ordered set of history values. Note that an extended subarray is one to one corresponding with its history value.

As is well known, an element $<i_1, i_2, ..., i_n>$ in an n dimensional fixed size array of size $[s_1, s_2,..., s_n]$ is allocated on memory using an addressing function like:

$$f(i_1, ..., i_n)= s_2s_3 ...s_ni_1+s_3s_4 ...s_ni_2+ ...+s_ni_{n-1}+i_n \quad (1)$$

We call $<s_2s_3...s_n, s_3s_4...s_n, ..., s_n>$ as a *coefficient vector.* Such a coefficient vector is computed at array extension and is held in a coefficient table.

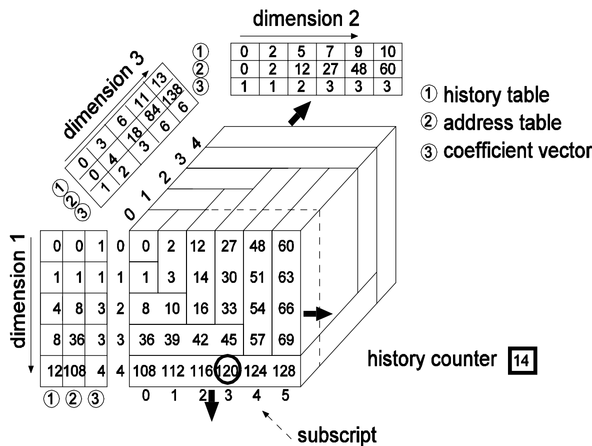


Figure 3. A three dimensional extendible array

Using these three kinds of auxiliary tables, the address of an array element can be computed as follows.

Consider the element <4,3,0> in Fig. 3. Compare the history values $H_1[4]=12$, $H_2[3]=7$ and $H_3[0]=0$. Since $H_1[4]>H_2[3]>H_3[0]$, it can be proved that <4,3,0> is involved in the subarray S corresponding to the history value 4 in the first dimension and the first address of S is found out in $L_1[4]$ =108. From the corresponding coefficient vector $C_1[4]$ =<4>, the offset of <4,3,0> from the first address of S is computed as 4x3+0=12 according to expression (1), the address of the element is determined as 120. Note that we can use such a simple computational scheme to access an extendible array element only at the cost of small auxiliary tables. The superiority of this scheme is shown in [5] compared with other schemes using such as hashing.

Ⅳ PROPOSED STORAGE MODEL

The model that we are going to present is based on the extendible array explained in Section Ⅲ. Each column of a relational table corresponds to a dimension of the extendible array and each column value of a record is uniquely mapped to a subscript of the dimension. A subarray is constructed for each distinct value of a column. Fig. 4 shows a realization of a two column relational table using our new scheme that will be explained.

For a relational table R with n columns, the corresponding logical structure of HOMD (History Offset implementation scheme for Multidimensional Datasets) [6] is the pair (M, A). A is an n dimensional extendible array created for R and M is the set of mappings. Each m_i $(1 \le i \le n)$ in M maps the i-th column values of R to subscripts of the dimension i of A. A will be often called as a *logical extendible array*.

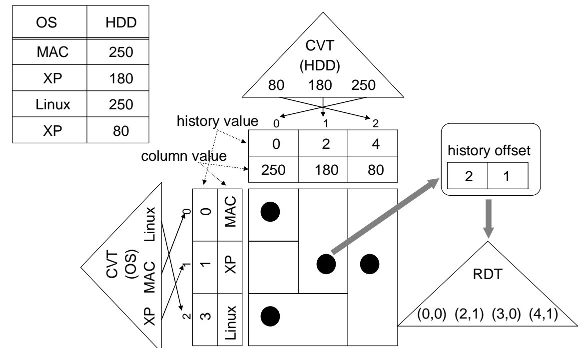


Figure. 4. HOMD physical structure

Each element of an n dimensional extendible array can be specified by its n dimensional coordinate. In our HOMD model, we have directed our attention to that each element can be specified by using the pair of history value and offset value. Note that since each history value h is unique and has a one-to-one correspondence with its corresponding subarray S, S is specified uniquely by its corresponding history value h. Moreover, the offset value of each element in S can be computed as was stated in Section Ⅲ and this is also unique in the subarray. Therefore each element of an n dimensional extendible array can be referenced specifying the pair (*history value, offset value*).

This reference method of an arbitrary element of an extendible array is very important for our HOMD technique. In the coordinate method, if the dimension of the extendible array becomes higher, the length of the coordinate becomes longer proportionally. Since an n-column record can be referenced by its n dimensional

© 2008 ACADEMY PUBLISHER

coordinate in the corresponding multidimensional array, the storage requirement for referencing records become large if the dimension is high. On the contrary, in our history and offset reference, even if the dimension is high, the size of the reference is fixed in short; i.e. only the two kinds of scalar value. It should be noted that this encoded reference also expresses the record itself since the reference can be easily decoded to the set of column values of the record. This greatly saves over all storage requirements for implementing relational tables by HOMD. Moreover, internal handling of table records in DBMS becomes simple and convenient owing to the fixed sized reference.

In the HOMD logical structure (*M*, *A*), each mapping m_i in *M* is implemented using a single B^+ tree called CVT(key subscript ConVersion Tree), and the logical extendible array *A* is implemented using a single B^+ tree called RDT(Real Data Tree) and *n* HOMD tables, each of which is an extension of the three auxiliary tables of an extendible array.

CVT: CVT_k for the *k*-th column of an *n* columns relational table is defined as a structure of B^+ tree with each distinct column value *v* as a key value and its associated data value is subscript *i* of the *k*-th dimension of the logical extendible array *A*. Hence the entry of the sequence set of the B^+ tree is the pair (v,i). The subscript *i* references to the corresponding entry of the HOMD table in the next definition.

HT: HT(HOMD Table) corresponds to the auxiliary tables explained in Section Ⅲ. It includes the history table and the coefficient table. Note that the address table can be void in our HOMD physical implementation.

HT is arranged according to the insertion order. For example, the column value "Linux" is mapped to the subscript 2 as the insertion order, though in the sequence set of CVT, the key "Linux" is in position 0 due to the property of B^+ tree. At insertion of a record, each column value in it is inserted to the corresponding CVT as a key value if it does not still exist, and then the logical extendible array A is extended by one along the dimension, and a new slot in HT is assigned and initialized.

RDT: The set of the pairs (*history value, offset value*) for all of the effective elements in the extendible array are housed as the keys in a B^+ tree called RDT. Here, the effective elements mean the ones that correspond to the records in the relational table.

RDT together with HTs implements the logical extendible array on the physical storage. Note that problem (4) in Section Ⅲ is greatly improved by employing RDT since only effective elements in the logical extendible array corresponding to actual table records are stored in RDT. We assume that a key (*history value, offset value)* occupies fixed size storage and the *history value* is arranged in front of the *offset value*. Hence the keys are arranged in the order of their history values and keys that have the same history value are arranged consecutively in the sequence set of RDT.

HOMD: For an *n* columns relational table, its HOMD implementation is the set of *n* CVTs, *n* HTs and RDT.

We can see that the problems discussed in Section Ⅲ are resolved or alleviated in our HOMD model. In the next section, we will describe how our proposed storage schemes work effectively.

[6] reports the performance evaluation of a prototype system of HOMD implementation of relational tables compared with that of PosgreSQL[13], one of the widely used DBMSs. The prototype system proves to outperform PosgreSQL both in storage cost and retrieval time cost. Interested readers can refer to [6] for details.

V REALIZATION OF THE REQUIREMENTS

In this section, we will explain that HOMD scheme presented in the previous section would be able to contribute to the storage organization and retrieval schemes adapting to requirements *A~E* presented in Section II. In the following examples, note that our HOMD scheme serves as not only data organization scheme, but also as index organization scheme of relational tables in an OLTP database.

A. Handling Dynamicity

Our HOMD realization scheme is based on the dynamicity of *extendible arrays*, in which a new emerging property value of a recycling object can be inserted along the corresponding dimension with extremely low space and time costs. The disadvantage of the sparseness inherent in extendible arrays can be overcome by employing HOMD data structures ; RDT stores only actually existing records encoded very efficiently with (*history, offset*).

In general, dynamicity of storage structure is not compatible with designing fast random accessing method. However, our HOMD implementation model is sufficiently flexible against the dynamical insertions and deletions of large number of records, but it does not sacrifice the fast random accessing capability of multidimensional arrays.

B. Handling Schema Evolution

In the conventional realization of a relational table, schema evolution such as adding new property (column) is a very costly operation. In the conventional realization, records in a table are arranged in the storage sequentially in the input order, and each record (set of column values) occupies consecutive storage area (Fig. 1). When a new column is added, this causes great overhead since reorganization of all records in the storage would be necessary to have a new kind of column value in each record.

In the case of HOMD, the situation is quite different. It is truly remarkable that no reorganization is necessary due to the dimension extendibility of the underlying extendible array structure. If a new column is added to an existing table, a new dimension is provided for the column in the corresponding logical extendible array. The corresponding column values of all the existing records are mapped to subscript 0 of the new dimension.

The CVT and the HOMD table of the new dimension should be created and initialized. But note that no reorganization of the existing records would be caused.

© 2008 ACADEMY PUBLISHER

Fig. 5 shows addition of the third dimension in a two dimensional logical extendible array.

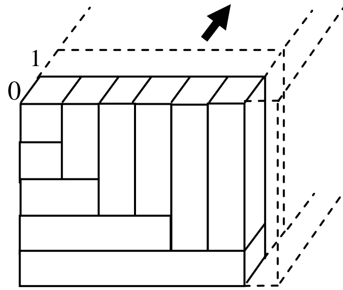


Figure 5. Addition of the third dimension in a two dimensional extendible array

Our HOMD scheme can afford powerful capability in respect of “schema evolution”. In the following example, we consider an index organization scheme using HOMD. It can serve as a part of a storage system in conjunction with the recycling product table implemented in the usual manner described in the beginning of Section II.

Using the PC lease company example stated in Section II, we can show how our scheme satisfies the requirement. Fig. 6(a) shows a recycled product table PT maintained by a PC lease company. Fig. 6 (b) shows an index organization IPT for Fig. 6(a) using a HOMD data structure. Be sure that each PC has a unique product ID, and each element of the index array IPT keeps the set of product ID of the PC’s whose CPU, HDD and MEMORY satisfy the required values. Each record identified by each of these product ID has (a set of) column values specified in the retrieval condition issued by users. As an example, for the product table PT, we consider a retrieval query:

Retrieve the product records whose memory size is greater than 1 gigabytes and HDD storage is greater than 100 gigabytes.

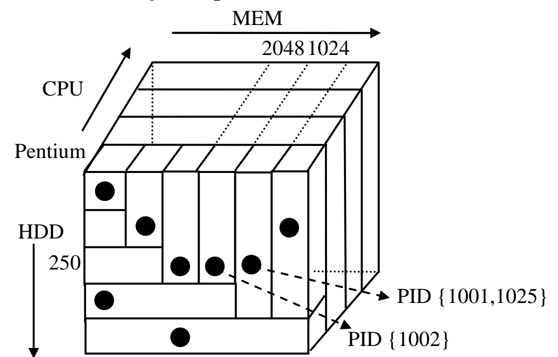
In processing this query, first the index array is searched and the target product ID’s are found out to be {1001, 1002}. Then the records having these ID’s are directly fetched. Such retrieving is accomplished internally in the system by the following SQL like query.

```
select *
from PT
where PT.pid in (
   select IPT.pid
   from IPT
   where IPT.mem > 1000 and
         IPT.hdd > 100
);
```

When another column of the product table other than the ones indexed in IPT has been found out to be frequently accessed in the retrieval system, adding an index for the column to IPT will greatly improve the retrieval performance. In this situation, the adaptability of our HOMD scheme to such kind of schema evolution makes the reorganization of the existing index array unnecessary.

PID	HDD	CPU	MEM.	OS	MFR	PRICE
1001	250	Pentium	1024	MAC	DELL	800
1002	250	Pentium	2048	XP	SONY	650
1003	80	Pentium	512	VISTA	NEC	700
1004	250	Athlon	512	XP	SONY	600

(a) Recycled product table PT



(b) Index array IPT for PT

Figure 6. Recycling product table

C. Handling Semi-structured Data

As we stated in Section II, in order to cope with possibly caused structural diversity among data instances for the commercial products produced according to the same specification, flexible data schema definition strategy would be strongly desired.

Semi-structured data model such as XML(eXtensible Markup Language[7]) seems to be one of the better candidates of such a flexible strategy. XML is now widely used not only for document preparations but also for specifying the logical structure and meaning of data exchanged between applications. It is not a strongly typed language like other conventional database language such as SQL. It allows users to define new *tag*s to indicate the meaning and structure of data instances. Structures can be nested to arbitrary depth.

An XML document can have its own schema, but the schema can be fairly flexible; it can have optional *elements* and *attributes*. This permits XML documents with its schema to involve a set of the same kind of data instances that may slightly differ in their details. See the example shown in Fig. 7(b). It should be also noted that an XML document is not necessary to include its schema; those that include schema are called specifically as *valid XML documents.* To be an XML document, it is sufficient to be only *well formed,* namely *elements* (or tags) are correctly nesting.

Following described semi-structured organization of an XML document seems to well meet the third requirement in Section II. Much research on XML databases has been done concerning efficient physical storage organizations of XML documents. For example, [8] describes an index

© 2008 ACADEMY PUBLISHER

organization scheme for XML documents using relational tables. [9] provides native index organization not depending on relational tables. An efficient mapping scheme of an XML document to our HOMD data structures is an important problem.

PID	DVD	BD	FD	USB	PR	EX_SP
1020	—	1	—	3	—	—
1021	2	—	—	—	EPSON	—
1022	—	—	2	—	—	2
1023	1	—	—	4	CANON	—

(a) Peripheral table

```
<PRODUCTS>
    <PC PID = 1020>
        <BD> 1</BD>
        <USB> 3</USB>
    </PC>
    . . . . . . . .
    <PC PID = 1023>
        <DVD> 1 </DVD>
        <USB> 4</USB>
        <PR> CANON </PR>
    </PC>
    . . . . . . . .
</PRODUCTS>
```

(b) XML representation for (a)

Figure 7. Relational table and XML document

D. Traceability Along Time Axis

It is quite simple to introduce time axis to our HOMD data structure. It can be accomplished by involving time dimension; adding a new CVT and HOMD table corresponding to the time dimension to be added. Even if a time dimension is dynamically added in the recycle chain, the cost is very low due to the adaptability of HOMD against the schema evolution discussed in *B* in this section.

It should be also noted that the values of time dimension are sorted in CVT, so a temporal retrieval query along the time dimension such as:

Retrieve the number of products in every month that have been registrated in these 3 years.

can be efficiently performed.

Back to the example of a PC lease company in Section II. We will further assume that whenever a PC is reproduced by replacing some parts such as HDD or a display, a record keeping the new configuration information of the PC is inserted into the recycled product table PT with the PC's product ID. PT is assumed to have two columns related to handling "time", one of which is *status* and the other is *date*. For the reproduced PC, the value of *status* is 'reproduced_date' and that of *date* is the reproduced date represented in the form of "year/month/day". Besides this reproduced date when the status of a PC is changed by *registration* or *shipping out* and so on, a new record is entered into PT keeping the new status with the product ID of the PC. Namely PT memorizes the recycling history of the PC.

In IPT, the date dimension represented by "year/month/day" is split into the three dimensions, namely *year, month*, and *day* dimension. See Fig. 8. *status* column in the product table PT is added in IPT as the fourth time dimension. These four time dimensions are organized into time index array IPT implemented by HOMD. Note that while the range of *month* and *day* dimensions are fixed, *year* and *status* dimension are extendible.

As in the example in *B* of Section V, we assume each array element in IPT keeps the corresponding product ID's in PT. Such HOMD index provides flexible and efficient handling capability for the temporal retrieval queries along the time dimensions. Here we give some temporal query examples, and show how these queries are processed using the corresponding index array IPT. Consider the above temporary query example. The retrieval for the query can be accomplished internally in the system by the following SQL like query:

```
select IPT.month, count(IPT.pid)
from IPT
where IPT.status = 'registration_date' and
      IPT.year between 2005 and 2007
group by IPT.month;
```

Note that this query can be processed very efficiently due to the capability of HOMD. The specification of *date* dimension value restricts the search range to the corresponding three dimensional *slice* of IPT and the range specification on *year* dimension further restricts the range. Moreover the restricted range in IPT can be directly accessed. See Fig. 8.

The second example is a retrieval query for both PT and IPT:

For each kind of CPU in a product, retrieve the price average of the products that have been shipped out in these 3 years

This query is processed as:

```
select PT.cpu, average(PT.price)
from PT
where PT.pid in
   ( select IPT.pid
     from IPT
     IPT.status = 'shipping_date' and
     IPT.year between 2005 and 2007
   )
group by PT.CPU;
```

The third example is also a retrieval query for both PT and IPT:

© 2008 ACADEMY PUBLISHER

Retrieve the history of the reproduced dates and the price for each product whose registration is in 2007.

This query is processed as:

```
select  PT.pid, PT.day, PT.price
from PT
where PT.status = 'reproduced_date' and
    PT.pid in
    ( select  IPT.pid
     from IPT
      IPT.status = 'registration_date' and
      IPT.year = 2007
    )
group by PT.pid
order by PT.pid;
```

The second and the third queries can be processed very efficiently since the inner sub-query against the time index IPT fetches the required product ID's very fast.

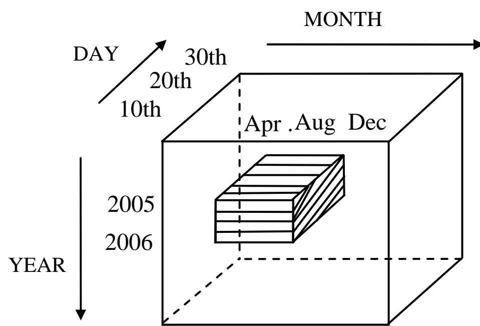


Figure 8. Restricting time range in time index

E. Analytical Processing Capability

On-Line Analytical Processing (OLAP) is a technology that enables to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data. OLAP functionality is characterized by dynamic multi-dimensional analysis of enterprise data supporting end user analytical and navigational activities such as slicing subsets for on-screen viewing, drill-down to deeper levels of trend analysis over sequential time periods. OLAP is implemented in a multi-user client/server mode and offers consistently rapid response to queries, regardless of database size and complexity. OLAP helps the user synthesize enterprise information through comparative, personalized viewing, as well as through analysis of historical and projected data in various scenarios. HOMD storage model would contribute to develop an efficient multidimensional OLAP (MOLAP) system due to its fast random accessing capability and extendibility.

As for *data cubes* stated in *E* of Section II, each *cuboid* in the data cube can be implemented by a multidimensional array each of whose dimensions is the corresponding column of a relational table *R* and the measure value of each record of *R* is stored at the corresponding element of the array. Thus 2^n-1 arrays are necessary to implement the data cube. Instead a single array data model can be used. Fig. 9 shows an example of a two dimensional single array data cube. Each element at (0, *) or (*, 0) denotes the corresponding aggregated measure value and element (0, 0) keeps aggregated total measure value.

Usually in MOLAP (Multidimensional OLAP) such a single data cube array is implemented by using a fixed size multidimensional array. In MOLAP systems, a snapshot of a relational table in a front-end OLTP database is taken and dumped into a multidimensional array periodically like in every week or month. At every dumping, a new empty fixed size array has to be prepared and the relational table will be dumped again from scratch. If the array dumped previously is intended to be used, all the elements in it should be relocated by using the corresponding address function of the new empty array, which also incurs huge cost.

If we employ a HOMD data structure to implement a single array data cube, due to the extendibility of HOMD, incremental construction of a new data cube is possible by adding the records generated only within the current time period to the original data cube. This incremental handling greatly saves the time consuming task of data cube construction.

Sales Data Cube

total sum — MONTH — sum in April

YEAR	ALL	Jan.	Feb.	Mar.	Apr.
ALL	135	10	35	60	30
2005	45		15	30	
2006	40	10			30
2007	50		20	30	

sum in 2007

Figure 9. A single array data cube

VI CONCLUSION

This paper described several requirements needed for recycling management systems, especially on their information storage architecture and retrieval capability of recycling end products. Their realization scheme adapting these requirements has been discussed. The scheme is improving lower level storage organization or internal data structuring strategy, so providing handling efficiency or retrieval efficiency rather independently of the kind of recycling object.

© 2008 ACADEMY PUBLISHER

ACKNOWLEDGMENT

This work was supported in part by a grant-in-Aid for Scientific Research of Japan Soceity for the Promotion of Science in Japan (No.19500079).

REFERENCES

[1] R. Itsuki, N. Kotani, N. Komoda, 'A recycle chain management system concept and an analysis method using RF-ID tags', Proc. of SMC, pp.4509- 4514, 2004.
[2] R. Itsuki, N. Kotani, 'An Information System Concept for Food Recycling', Proc. of ETFA, 10.1109/ETFA.2005.1612682, 2005.
[3]M. Stutz, V. M. Thomas and S. Saar, 'Linking Bar Codes to Recycling Information for Mobile Phones', Proc. of ISEE, pp.313-316, 2004.
[4]X. Tang, H. Yun, 'Data model for quality in product lifecycle', Computers in Industry, Vol.59, 2-3, pp.167-179, 2008.
[5] E. J. Otoo and T. H. Merrett, 'A Storage Scheme for Extendible Arrays', Computing, Vol.31, pp.1-9, 1983.
[6] M. Kuroda, N. Azuma, K. M. A. Hasan, T. Tsuji, K. Higuchi, 'An Implementation Scheme of Relational Tables', Proc. of ICDE Workshops, 1244, 2005.
[7] E. T. Ray, 'Learning XML', Oreilly & Associates Inc, 2nd Edition, 2003.
[8] K. Fujimoto, D. Kha, M. Yoshikawa, T. Amagasa, 'A Mapping Scheme of XML Documents into Relational Databases using Schema-based Path Identifiers', Proc. of WIRI, pp.82-90, 2005.
[9] http://xml.apache.org/xindice/#Releases
[10] M. Koubarakis, et. al. (Eds), 'Spatio-Temporal Databases: The CHOROCHRONOS Approach', LNCS 2520, Springer, 2003.
[11] R. Wrembel, C. Koncilia (Eds), 'Data Warehouses and OLAP : Concepts, Architectures and Solutions', Idea Group Pub., 2006.
[12] V. Harinarayan, A. Rajaraman, and J. D. Ullman, 'Implementing Data Cubes Efficiently', Proc. of the ACM SIGMOD Conference, pp.205-216, 1996.
[13] http://www.postgresql.org/
[14] T. Tsuchida, T. Tsuji and K. Higuchi, 'Considerations on Information Storage and Retrieval Systems for Recycling Objects', Proc. of IEEE SMC 2007, pp.2302-2307, 2007.

Takayuki Tsuchida is currently a candidate of Ph.D. student of Graduate School of Engineering at University of Fukui Japan. He received his B.E. degree from the same university in 2007. His research interests include data warehousing and MOLAP systems.

Teppei Shimada is currently a candidate of Ph.D. student of Graduate School of Engineering at University of Fukui Japan. He received his B.E. and M.E. degree from the same university in 2006 and 2008. His research interests include data warehousing and OLAP systems.

Tatsuo Tsuji received his B.E. degree in Electrical Engineering from Osaka University in 1973, and the M.E. and Ph.D. degree in Information and Computer Science from the same university in 1975 and 1978, respectively. In 1978, he joined the Faculty of Engineering at University of Fukui in Japan. Since 1992, he has been a professor in the Information Science Department of the faculty. His current research interests include database implementation schemes and data warehousing systems. He is the author of the book, "Optimizing Schemes for Structured Programming Language Processors" published by Ellis Horwood (1990).

Ken Higuchi received his B.E., M.E. and Ph.D. degree in communication and system engineering from the University of Electro-Communications, Japan in 1992, 1994 and 1997. He is currently an associate professor of in the Graduate School of Engineering, University of Fukui, Japan. His current research interests include database management system, parallel processing, and XML document search system.

© 2008 ACADEMY PUBLISHER