BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **16**, No 6
Special issue with selection of extended papers from 6th International Conference on Logistic, Informatics and Service Science LISS'2016

Sofia • 2016 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2016-0074

The Algebraic Operations and Their Implementation Based on a Two-Layer Cloud Data Model

Ying Li[1], *Baotian Dong*[2]

[1]*Guangxi Colleges and Universities Key Laboratory of Scientific Computing & Intelligent Information Processing, Guangxi Teachers Education University, Nanning 530001, China*
[2]*School of Traffic and Transportation, Beijing Jiaotong University, Beijing 100044, China*
Emails: ling.ly@126.com btdong@bjtu.edu.cn

***Abstract*:** *The existing cloud data models cannot meet the management requirements of structured data very well including a great deal of relational data, therefore a two-layer cloud data model is proposed. The composite object is defined to model the nested data in the representation layer, while a 4-tuple is defined to model the non-nested data in the storage layer. Referring the relational algebra, the concept of SNO* (*Simple Nested Object*) *is defined as basic operational unit of the algebraic operations; the formal definitions of the algebraic operations consisting of the set operations and the query operations on the representation layer are proposed. The algorithm of extracting all SNOs from a CAO* (*Component-Attribute-Object*) *set of a composite object is proposed firstly as the foundation, and then as the idea; the pseudo code implementation of algorithms of the algebraic operations on the storage layer are proposed. Logic proof and example proof indicate that the definition and the algorithms of the algebraic operations are correct.*

***Keywords*:** *Cloud database; data model; algebraic operation; key-value; structured data.*

1. Introduction

A cloud database is a kind of cloud computing technology, which has been developed with the conception of "Software-as-a-Service" [1]. The existed cloud data models are key-value model and cloud relational data model. Cloud databases using key-value model are BigTable [2], SimpleDB [3], PNUTS [4], HBase [5], Amazon DynamoDB [6], HugeTable [7], MongoDB [8], CouchDB [9], and so on; while those using cloud relational model are Amazon relational database service

[10], GaianDB [11], SQL Azure [12], and so on. Cloud relational model is still relational model logically but its storage method is changed. Conceptions of row set and table set are used to store data in the cloud in order to process data parallel and improve the efficiency of the join operation. The merits of cloud relational model include: simple data structure, so distributed storage can be implemented easily; relational algebra and query language have been provided, so complex query can be implemented; SQL language is simple and easy to learn, so average user can use it easily. The shortcomings include: implementing the join operation in the cloud is expensive and difficult; a database is hard to expand because the capacity of the database is limited depending on the joined operation; artificial intervention cost will be increased because associated tables should be stored in the same data node. Key-value model is a kind of storage model in which a record is a value without data structure and key is an index of the value. Databases using key-value model are more like a file system. The merits of the key-value model include: data has no structure, so distributed storage and parallel processes can be implemented easily; access data by using a record as a unit and a key as an index, so it is easy to obtain high access performance and good expansibility. The shortcomings include: it cannot be used to represent structured data, so the application domain is limited; query algebra and query language have not been provided, so complex query cannot be implemented; data analysis has to be done by programming, so it is hard to be used by average users without programming technology. It was indicated that a great deal of structured data including relational data needs to be stored in the cloud to be read/write online, analysed and mined in references [13-16]. Thus a kind of cloud databases which have high performance of data access, good expansibility, good usability and powerful query capability are in need to manage the relational data and other structured data. The existing cloud data models are not suitable for such cloud databases; therefore a new type of cloud data model is needed.

The contribution of this paper is shown below.

(1) A two-layer cloud database model is proposed. The composite object is defined to model the nested data in the representation layer; the concept of CAO (Component Attribute Object), schema and a 4-tuple are defined to model the non-nested data on the storage layer.

(2) The formal definitions of the algebraic operations consisting of the set operations and the query operations on the representation layer are proposed. The Simple Nested Object (SNO) is defined as the basic operational unit of the algebraic operations. The definition of attribute-tree is proposed to be applied in the query operations. Example validation is done to verify the correct of the definition of these algebraic operations.

(3) The idea and the pseudo code implementation of algorithms of the algebraic operations on the storage layer are proposed. The CAO set of a SNO is proposed as the basic operational unit of the algebraic operations on the storage layer. The algorithm of extracting all SNOs in their CAO form from the CAO set of a composite object is proposed to be the foundation of all algorithms of the algebraic operations. The definition of parents-child relation and the definition of

CAO-tree are also proposed and applied in extracting SNO. Logic validation and example validation are done to verify the correctness of the algorithms.

2. The two-layer data model

2.1. The representation layer

Definition 2.1. Atomic object. A set which consists of A:V is called an *atomic object*. A:V is called a *component*. A is an attribute and V is a value which is a simple data type such as integer and string. Such A:V is an atomic component and A is an atomic attribute. A is a composite attribute if V is not a simple data. Such A:V is called a *composite component*.

Definition 2.2. Recursive definition of a composite object. An object is a set of A:V, and a) if an object is atomic, it is a composite one; or b) given O= $\{(A_i$: $[O_{i1}, O_{i2},..., O_{im}])|i=1, 2,..., n; n \in \mathrm{N}, m \in \mathrm{N}\}$, if $\exists\, O_{ij}$, $1 \leq j \leq m$, and it is a composite object, then O is a *composite object*. An atomic object and a composite object are called an *object*.

$[O_{i1}, O_{i2},..., O_{im}]$ means an array of objects in the definition above. A pair of curly braces represents a layer of a composite object; and it also represents a nested object. A component is called a key component if it is the one which can uniquely identify a nested object.

Example 1. A composite object.

{“personSetID”: “01”, “authors”: [{ “authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “books”: [{“bookID”: “01”, “bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: “Grafton Books”}]}, {“authorID”: “02”, “firstName”: “Tad”, “lastName”: “Williams”, “books”: [{“bookID”: “03”, “bookname”: “Empire of the Ants”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}, {“bookID”: “04”, “bookname”: “Le Papillon DES Etoiles”, “publishinghouse”: “Librairie generale francaise”}]}], “musicians”: [{“musicianID”: “01”, “firstName”: “Eric”, “lastName”: “Clapton”, “instrument”: “guitar”}, {“musicianID”: “02”, “firstName”: “Sergei”, “lastName”: “Asimov”, “instrument”: “piano”}]}

Components whose form like “authorID”: “**”, “bookID”: “**” and “musicianID”: “**” are the key components in the example above.

2.2. The storage layer

Definition 2.3. Component-Attribute-Object, CAO. Take a key component (for short C) from a nested object x, take an attribute (for short A) from x, and take all the atomic components of one value of A to form an atomic object (for short O). C, A and O forms a 3-tuple which is called a CAO of A.

A has only one CAO if its value is an object; A has more than one CAO if its value is an array of objects. A composite object can have more than one CAO.

Definition 2.4. The schema of a composite object. Given a composite object $\{(A_{i:}\ [O_{i1}, O_{i2},..., O_{im}])|i=1, 2,\ldots, n; n \in N, m \in N\}$. Its schema is a set in which an

element is a map from an attribute to its data type, expressed as Φ=$\{A$->$(A_1, A_2,..., A_r)$ or A->basic data type$|A \in \{A_i\}; A_k \in \{A_i\}; i=1, 2,\ldots, n; k=1, 2,..., r, r \leq n\}$; A is the pre-image of map A->$(A_1, A_2,\ldots, A_r)$, and $(A_1, A_2,..,. A_r)$ is the image of the map.

Definition 2.5. The 4-tuple instance of a composite object. Suppose a composite object O, the storage instance of O is $(O^0, T^0, \Phi, \varphi)$. O^0 is the set of atomic components in the top level; T^0 is the set of composite attributes in the top level; φ is the set of CAO; and Φ is the schema of O.

Example 2. A 4-tuple instance corresponding to the composite object in Example 1.

Answer: $O^0= \varnothing$;

T^0= {“authors”,“musicians”};

Φ ={“authors”-> (“firstName”, “lastName”, “books”), “books”-> (“bookName”, “publishinghouse”), “musicians”->(“firstName”, “lastName”, “instrument”), “firstName”-> String, “lastName”-> String, “bookname”-> String, “publishinghouse”-> String, “instrument”-> String)};

φ = {(“personSetID”: “01”, “authors”, { “authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”}), (“personSetID”: “01”, “authors”, { “authorID”: “02”, “firstName”: “Tad”, “lastName”: “Williams”}), (“personSetID”: “01”, “musicians”, {“musicianID”: “01”, “firstName”: “Eric”, “lastName”: “Clapton”, “instrument”: “guitar”}), (“personSetID”: “01”, “musicians”, { “musicianID”: “02”, “firstName”: “Sergei”, “lastName”: “Asimov”, “instrument”: "piano”}), (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}), (“authorID”: “01”, “books”, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: "Grafton Books”}), (“authorID”: “02”, “books”, { “bookID”: “03”, “bookname”: “Empire of the Ants”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}), (“authorID”: “02”, “books”, {“bookID”: “04”, “bookname”: “Le Papillon DES Etoiles”, “publishinghouse”: “Librairie generale francaise”})}

2.3. The method to transform a CAO to a pair of key-value

The key-value model is used to storage data in the two-layer data model because it supports high expansibility. Each CAO will be converted to a storage unit after a composite object being converted to a CAO set. How to transform a CAO to a pair of key-value can refer to literature [17].

2.4. Data transforming methods between the representation layer and the storage layer

CAO plays a key role in the data converting between representation layer and storage layer. A composite object can concert to more than one CAO, and vice verse. The bidirectional converting algorithms between the representation layer and the storage layer can refer to literature [17].

3. The formal definition of algebraic operations and their implementation in the representation layer

3.1. Relevant definitions

Definition 3.1. Attribute-tree. Given the 4-tuples (O^0, T^0, Φ, φ) of composite object O, create a tree whose root node is an attribute of T^0, and create other nodes according to maps in Φ.The pre-image of a map is a parents node and the images of this map are child nodes. Such a tree is called an attribute-tree of the composite object.

The node set of all attribute-trees is the set of attributes of O; and the edge set of these trees is the map set of O except those maps whose pre-images are atomic attributes.

One or more attribute-trees can be created from a 4-tuple of a composite object, and the number of attribute-trees is equal to the number of elements in T^0. Attribute-trees (a) and (b) in Fig. 1 are attribute-trees of X in Example 1.

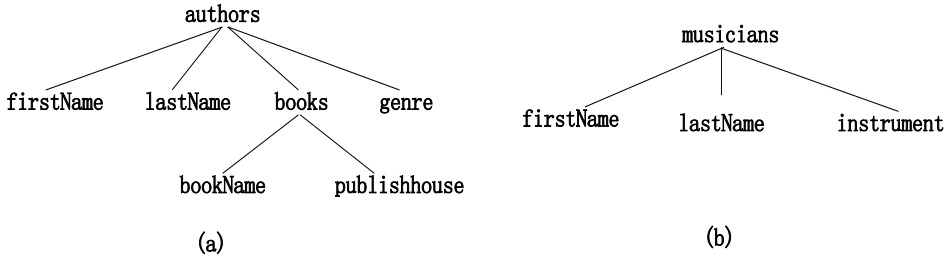

Fig. 1. Two attribute-trees of X in Example 1

Definition 3.2. Simple Nested Object, SNO. Given a composite object O, extract an object from O. If this object meets the conditions: a) it has only one top attribute; b) the value of any attribute is basic data type or another object but not an array; c) its attribute-tree is one of the attribute-trees of O, then this object is called a simple nested object, for short SNO.

The attributes of a SNO can form only one attribute-tree. SNO is the basic operational unit of set operations and query operation. It is just logic operational unit, and the physical one is CAO set of SNO in fact.

3.2. The set operations

The set Operations consist of intersection operation, union operation and difference operation. Given two composite objects X and Y; X^s is the set of SNO of X, and Y^s is that of Y; X and Y cannot participate in the operations directly. They should be converted to SNO sets which take part in the operations. The operational result which is the set of SNO should be converted to a composite object. The formal definitions of intersection operation, union operation and difference operation are below.

(1) *Intersection*: X and Y share at least an attribute-tree, and the result of intersection operation of X^s and Y^s is a set of SNO which shares by X^s and Y^s, expressed as $X^s \cap Y^s$, calculated by formula

(3.1) $$X^s \cap Y^s=\{O^s|O^s \in X^s \wedge O^s \in Y^s\}.$$

(2) *Union*: The union of X^s and Y^s is a set of SNO which belong to X^s or belong to Y^s, expressed as $X^s \cup Y^s$, calculated by formula

(3.2) $$X^s \cup Y^s=\{O^s|O^s \in X^s \vee O^s \in Y^s\}.$$

(3) *Difference*: X and Y share at least an attribute-tree. The difference of X^s and Y^s is a set of SNO which belong to X^s but not belong to Y^s, expressed as $X^s–Y^s$, calculated by formula

(3.3) $$X^s - Y^s=\{O^s|O^s \in X^s \wedge O^s \notin Y^s\}.$$

Use the intersection operation as an example to illustrate how to calculate the set operations on the representation layer.

Example 3. Suppose composite object X={“personSetID”: “01”, “authors”: {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: [{“bookID”: “01”, “bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group"}, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: “Grafton Books ”}]}};

Y={“personSetID”: “01”, “authors”: {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: [{ “bookID”: “01”, “bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}, {“bookID”: “05”, “bookname”: “The Mysterious Island”, “publishinghouse”: “Signet Classics”}]}}, calculate $X \cap Y$.

Answer: Exact the SNO set X^s from X, and t exact the SNO set Y^s from Y firstly. Next calculate $X^s \cap Y^s$. Last convert $X^s \cap Y^s$ to a composite object.

X^s ={{“personSetID”: “01”, “authors”: {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookID”: “01”, “bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}}}, {“personSetID”: “01”, “authors”: {“bookID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookID”: “02”, “bookname”: “End of ternity”, “publishinghouse”: “Grafton Books”}}}};

Y^s ={{“personSetID”: “01”, “authors”: {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookID”: “01”, “bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}}}, {“authors”: {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookID”: “05”, “bookname”: “The Mysterious Island”, “publishinghouse”: “Signet Classics”}}}};

$X^s \cap Y^s$ = {{“personSetID”: “01”, “authors”: {“firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}}}};

$X \cap Y$ ={“personSetID”: “01”, “authors”: {“firstName”: “Isaac”, “lastName”: “Asimov”, “genre”: “science fiction”, “books”: {“bookname”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}}}.

3.3. The query operation

3.3.1. The projection operation

Projection is to filter some components of a composite object and create a new composite object. Given composite object O and its attributes number k, the projection of O on attributes $A_{l_1},\ldots, A_{l_m}$ ($m \le k$; $l_1,\ldots,l_m$ are integers between 1 and k; and the m attributes can form an attribute-tree; $A_{l_1},\ldots, A_{l_m}$ is the projection attribute set) is done by projecting on the SNO set (suppose O^s) of O. Use symbol $P_{l_1,\ldots,l_m}(O^s)$ to represent the projection operation. The formal definition of projection is defined as

$$(3.4)\quad P_{l_1,\ldots,l_m}(O^s) = \{\alpha \mid (\exists\beta)(\beta \in O^s) \wedge (\alpha[C_{j_1}] = \beta[C_{j_1}]) \wedge \ldots \wedge (\alpha[C_{j_m}] = \beta[C_{j_m}]) \wedge (D_\alpha \subset D_\beta) \wedge (R_\alpha \subset R_\beta),\quad \{C_{j_1},\ldots,C_{j_m}\} \subset \{C_{l_1},\ldots,C_{lk}\}, m \le k\},$$

where α and β are both SNOs; $\alpha[C_{j_i}]$ and $\beta[C_{j_i}]$, i =1,…, m, represent atomic component C_{j_i} of α and β respectively. (D_α, R_α) and (D_β, R_β) represent the attribute-tree of α and β respectively. $(D_\alpha \subset D_\beta) \wedge (R_\alpha \subset R_\beta)$ indicates that the attribute-tree of α is a sub-tree of that of β. Equation (3.4) means that the projection on the SNO set is to select some components of a composite object. If the attributes in the projection set cannot form an attribute-tree according to the schema of O, expand the projection set until the attributes in it can form an attribute-tree before doing the projection operation.

Example 4. Given composite object X and its SNO set X^s in Example 3, calculate $P_{\text{authors, lastname, books, bookname}}(O)$.

Answer: First, calculate $P_{\text{authors, lastname, books, bookname}}(X^s)$= {{“authors”: {“lastName”: “Asimov”, “books”: {“bookname”: “Fantastic Voyage”}}}, {“authors”: {“lastName”: “Asimov”, “books”: {“bookname”: “End of ternity”}}}}.

Second, Convert the result to a composite object: {“authors”: [{“lastName”: “Asimov”, “books”: {“bookname”: “Fantastic Voyage”}}, {“authors”: {“lastName”: “Asimov”, “books”: {“bookname”: “End of ternity”}}]}.

3.3.2. The selection operation

Suppose composite object X, and a selection condition F. There are two ingredients in F, one is operand, and another is operator. Operand can be constants or atomic attributes. These attributes belong to an attribute-tree or are shared by several attribute-trees. Such specification will make these attributes appearing in every SNO of a SNO set. Operators include comparing symbol (<, ≤, ≥, >, ≠, =) and logic symbols ($\wedge$, $\vee$). The comparison operator is called CS, and the logical operator is called LS. Conditional expressions can be expressed as “(attribute$_1$ CS value$_1$) LS(attribute$_2$ CS value$_2$)…LS(attribute$_n$ CS value$_n$)”. The result of the selection operation of X under F is also a composite object. The selection operation is

expressed as $\sigma_F(X^s)$. Given the SNO set of X, $X^s = \{O_i^s | i=1, 2,..., n\}$ (O_i^s is a SNO), the selection result is calculated by formula

(3.5) $\sigma_F(X^s) = \{|(O_{x_j}^s \in X^s) \wedge (F(O_{x_j}^s)=\text{true}), 1 \leq x_j \leq n, j=1, 2,..., m; 1 \leq m \leq n\}$.

4. The implementation of algebraic operations on the storage layer

The basic operational unit of the set operation and the query operation are logically SNO on the representation layer, but the implementations of these operations are done on the CAO sets on the storage layer. A composite object consists of more than one SNO in the representation layer and the composite object is decomposed to more than one CAO in the storage layer, so a SNO can consists of one or more CAOs. Therefore how to extract all SNOs from the CAO set of a composite object is the foundation of implementing algebra operations on the storage layer. In addition, the set operation and the query operation need to determine whether two CAO sets are equal and whether two CAOs are equal.

4.1. Relevant algorithms

4.1.1. The algorithm for extracting all SNO sets from a CAO set

Definition 4.1. Parents-child relation of CAO. Given two CAOs: X_1 and X_2, X_1 is the parents of X_2 and X_2 is the child of X_1 if getKeyCon(X_1.O) = X_2.C.

Function getKeyCon(t) is to get the key component of t which is an atomic object. (“personSetID”: “01”, “authors”, {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”}) and (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”}) in Example 2 is a pair CAO which has a parents-child relation, and the former is the parents of the latter, the latter is the child of the former.

Definition 4.2. CAO-tree. Given a CAO set of a composite object, take a CAO whose A is a composite attribute of the top level as the root node of a tree, and then take the children of the CAO to be the child nodes of the root, and add other CAOs as the parents nodes and their children as the children nodes to the tree. Such tree is called a CAO-tree.

According to the definition of CAO-tree, all CAOs in the path from the root node to a leaf node are all the CAOs of a SNO. A set of all SNOs which belong to the same attribute-tree can be acquired by traversing all the nodes on the path from the root node to all the leaf nodes. The CAO-tree can be stored with one-dimensional array. The structure of CAO and that of the CAO-tree are defined below.

```
TYPE CAO=RECORD
    C: String;
    A: String;
    O: ARRAY[1..maxlen] OF String;
END;
```

```
TYPE tnode=RECORD
    data: CAO;
   parents: CAO;
END;
T_CAO =ARRAY[1..p] OF tnode.
```

Construct a CAO-tree first, and then use depth-first-traversal method to traverse the tree to acquire all CAO sets of all SNOs. The CAO-tree is stored in array T_{CAO}. The CAO set of a composite object is stored in array O_{CAO}. The composite attributes set on the top level of a composite object are stored in array LA.

Function createCAOTree (LA, O_{CAO}) is used to construct a CAO tree, and its pseudo code is shown below.

```
FUNC createCAOTree(LA:ARRAY[1.. n] OF String, O_CAO: ARRAY[1.. m] OF
CAO) :ARRAY[1..p] OF tnode;
    T_CAO: ARRAY[1..p] OF tnode;
    i:=1;
    WHILE i<=m DO
     【T_CAO [i].data:=O_CAO [i]; i:=i+1;】
    i:=1;
    WHILE i<=n DO  {these double loops set the parents of a CAO whose
                          attribute is the top composite attribute to be NIL}
     【a:= LA[i]; j:=1;
        WHILE j<=m DO
             IF T_CAO [j].data.A=a THEN
                  【T_CAO [j].parents=NIL; break;】
       ELSE j:=j+1;
     】
    i:=1;
    WHILE i<=m DO {these double loops find a child node of a CAO}
     【i=i+1; j=1;
       WHILE j<=m DO
         IF (getKeyCon(c[i].data.O)= T_CAO [j].data.C)
         THEN T_CAO [j].parents= T_CAO [i].data;
         ELSE j=j+1;
     】
    RETUREN T_CAO;
ENDF.
```

Create the CAO-trees of the composite object in Example 2. These trees are showed in Fig. 2. There are four CAO-trees which can form 6 CAO sets of SNO. These CAO-trees are stored in a two dimensional array, as shown in Table 1.

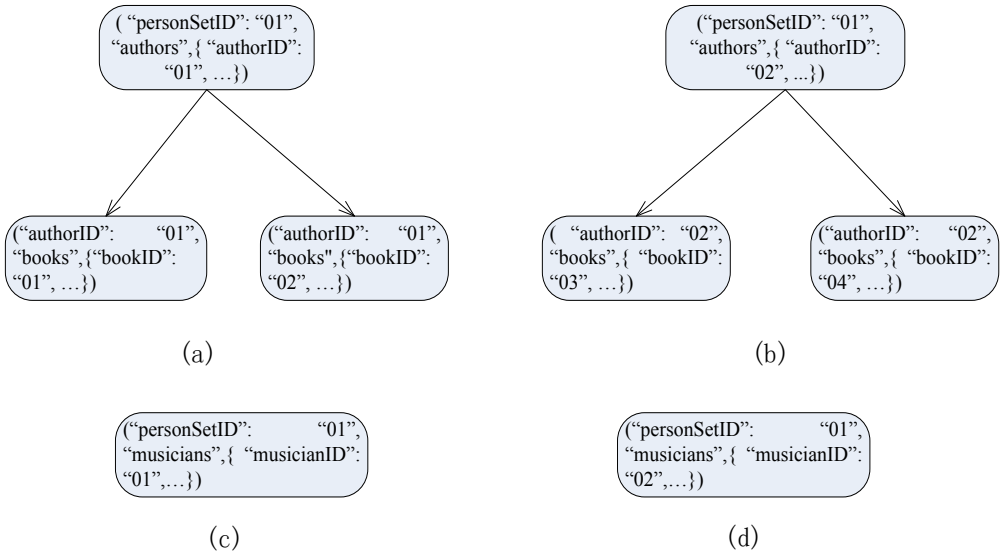

Fig. 2. The CAO-trees of the composite object of Example 2

Table 1. The CAO-trees of the composite object of example 2 are stored in a two dimensional array

(“personSetID”: “01”, “authors”, { “authorID”: “01”, …})	NIL
(“personSetID”: “01”, “authors”,{“authorID”: “02”, …})	NIL
(“personSetID”: “01”, “musicians”, {“musicianID”: “01”,…})	NIL
(“personSetID”: “01”, “musicians”, {“musicianID”: “02”,…})	NIL
(“authorID”: “01”, “books”,{“bookID”: “01”, …})	(“personSetID”: “01”, “authors”, { “authorID”: “01”, …})
(“authorID”: “01”, “books”,{“bookID”: “02”, …})	(“personSetID”: “01”, “authors”, { “authorID”: “01”, …})
(“authorID”: “02”, “books”, {“bookID”: “03”, …})	(“personSetID”: “01”, “authors”, { “authorID”: “02”, …})
(“authorID”: “02”, “books”, {“bookID”: “04”, …})	(“personSetID”: “01”, “authors”, { “authorID”: “02”, …})

Given the two dimensional array of the CAO-tree of a composite object T_{CAO}, stack S_1 storing the root node, stack S_2 storing a CAO set of a SNO in which an element is a CAO, set S_{CAO} storing the CAO sets of SNOs, The data structures of CAO stack and CAO set are defined respectively below.

```
TYPE CAOstack=RECORD
    data: ARRAY[1..n] OF CAO;
   top: integer;
END;
TYPE CAOSet=RECORD
    data: ARRAY[1.. length] OF CAO;
END;
```

Function extractSNOfrCAOSet(T_{CAO}) is used to extract the SNOs in their CAO form from a CAO set of a composite object, and its pseudo code is shown below.

```
FUNC extractSNOfrCAOSet(T_CAO: ARRAY[1..p] OF tnode) :ARRAY[1..p] OF
tnode;
    i:=1;  S_CAO: ARRAY[1..p] OF CAOSet;  S1, S2: CAOstack;
    WHILE i<=p DO
        IF T_CAO[i].parents=NIL THEN
            【push(S1, T_CAO[i].data); delete(T_CAO[i]); p:=p–1; i:=i+1;】
    WHILE S1.top≠0 DO
    【  p:=pop(S1);  push(S2, p);
        WHILE S2.top≠0 DO
      【  p:=S2.data[S2.top]; i:=1;
          WHILE i<=p DO   {This loop finds one child node of p}
           IF CAO equal(T_CAO[i].parents, p)
                  THEN 【push(S2, T_CAO[i].data); break;】
                       {T_CAO[i].data is one child node of p}
             ELSE i:=i+1;
           IF getChild(p)=NIL THEN    {p has no one child node, and p is a
                                       leaf node}
              【 readAllElement(S2, CAO_tmp); { Read all the elements in S2
                                                to CAO_tmp }
                 addElementToSet(S_CAO, CAO_tmp); { Add CAO_tmp to S_CAO }
                 delete(T_CAO, p);    { Delete the i-th element from T_CAO }
                 p:=p–1;
                 pop(S2);
              】
      ELSE
              IF CAOequal(p, S2.data[1])=FALSE THEN {p is not the
                                                     bottom element of S2.}
        【 delete(T_CAO, p);  p:=p–1; pop(S2);】
       】
    】
    RETURN S_CAO;
ENDF.
```

Use the function CAOequal(a, b) in Section 4.1.2 to determine whether two CAOs are equal. Function getChild(p) acquires one child node of p. If the child node is NIL, p is a leaf node. All CAOs on the path from the root node to a leaf node are found when finding a leaf node. These CAOs are all CAOs of a SNO. These CAOs can be put into set CAO_{tmp}, and then put CAO_{tmp} into S_{CAO}. If getChild(p) is not NIL and no a child node of p can be found in T_{CAO}, p cannot be a CAO of any SNO. Therefore, p should be popped from stack S_2, and the node whose "data" field is equal to p should be deleted from T_{CAO} if p is not the bottom element of S_2. All the SNOs belonging to the same attribute-tree have been found if

S_2 is empty. All root nodes of all the CAO-trees are stored in stack S_1. All the SNOs of all the attribute-trees of a composite object have been found if S_1 is empty, therefore the result S_{CAO} can be output. If p is the bottom element of S_2, it need not be deleted from T_{CAO} because it is in S_1 but not in T_{CAO}.

Example 5. Extract all the SNOs on their CAO form from the CAO set of X in Example 2.

Answer:

S_{CAO}={{(“personSetID”: “01”, “authors”, {“authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})}, {(“personSetID”: “01”, “authors”, { “authorID”: “01”, “firstName”: “Isaac”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: “Grafton Books”})}, {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “firstName”: “Tad”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “03”, “bookname”: “Empire of the Ants”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})}, {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “firstName”: “Tad”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “04”, “bookname”: “Le Papillon DES Etoiles”, “publishinghouse”: “Librairie generale francaise”})}, {(“personSetID”: “01”, “musicians”, {“musicianID”: “01”, “firstName”: “Eric”, “lastName”: “Clapton”, “instrument”: “guitar”})}, {(“personSetID”: “01”, “musicians”, {“musicianID”: “02”, “firstName”: “Sergei”, “lastName”: “Asimov”, “instrument”: “piano”})}}.

4.1.2. The algorithm to determine whether two CAOs are equal

Given two CAOs: u_1=(C_1, A_1, O_1) and u_2=(C_2, A_2, O_2), u_1 and u_2 are equal if ($u_1.C_1$= $u_2.C_2$)∧($u_1.A_1$= $u_2.A_2$)∧($u_1.O_1$= $u_2.O_2$). Function CAOequal(u_1, u_2) is used to determine whether two CAOs are equal, and its pseudo code is shown below.

```
FUNC CAOequal (var u1: CAO, u2: CAO ): boolean;
    IF u1.C= u2.C THEN
    【 IF u1.A= u2.A THEN
        【i:=1;
            WHILE i< getLength(u1.O) DO
            【t1:= u1.O[i]; i=i+1; c:= u2.O; j:=1;
                WHILE j< getLength(c) DO
                【 t2:=c[j]; j:=j+1;
                    IF (t1.A≠t2.A) or (t1.V≠t2.V) THEN RETURN
FALSE;
                】
            】
            RETURN TRUE;
        】ELSE RETURN FALSE;
    】ELSE RETURN FALSE;
ENDF.
```

4.1.3. The algorithm to determine whether two CAO sets of SNO are equal

Given two SNOs X_1 and X_2, the CAO set of X_1 is a, and that of X_2 is b. a and b are equal if: (1) the number of element of each set is equal; (2) for any element in a, there is an element in b to be equal to it. Use the function in Section 4.1.2 to determine whether two CAOs are equal. Function SimNetObjEq(a, b) is used to determine whether two CAO sets of SNO are equal, and its pseudo code is shown below.

```
FUNC SimNetObjEq (var a: ARRAY[1.. n] OF CAO, b: ARRAY[1.. m] OF CAO):
boolean;
    IF n=m THEN
        【i:=1;
            WHILE i<=n DO
            【t1:=a[i]; i=i+1; j:=1;
                WHILE j<= m DO
                【t2:=b[j]; j=j+1; IF CAOequal(t1, t2)=FALSE THEN
RETURN FALSE;】
            】
            RETURN TRUE;
        】ELSE RETURN FALSE;
ENDF.
```

4.2. The implementing algorithm of the set operations

The set operations are dual operations. There are two composite objects taking part in the set operations, and the result is also a composite object. The set operation done on the storage layer needs to compare two CAO sets of SNO. Suppose two composite objects O_1 and O_2 taking part in the set operation. Set S_{CAO1} stores the SNOs in CAO form extracted from the CAO set of O_1, and S_{CAO2} is that of O_2. In implementing the algorithms of the set operations, use the function in Section 4.1.3 to determine whether two CAO sets of SNO are equal.

4.2.1. Implementing algorithm of the union operation

When implementing the union operation $S_{CAO1} \cup S_{CAO2}$, read all the elements of S_{CAO2} and put them into set S_{CAO3}, then get an element (suppose t) from S_{CAO1} and compare t with all elements in S_{CAO2}. Add t to set S_{CAO3} if there is no an element in S_{CAO2} to be equal to t. Get other elements in S_{CAO1} and compare them with elements in S_{CAO2} and add suitable element to S_{CAO3} until S_{CAO1} is empty, here S_{CAO3} is the result. Function union(S_{CAO1}, S_{CAO2}) is used to merge two CAO sets of SNO, and its pseudo code is shown below.

```
FUNC union(var SCAO1: ARRAY [1..n] of CAOSet, SCAO2: ARRAY [1..m] of
CAOSet): ARRAY [1..p] of CAOSet;
    i:=1; k:=0; SCAO3:= SCAO2;
   WHILE i<=n do
    【  t1:= SCAO1[i];
```

```
        i:=i+1;
        j:=1;
        flag=true;
        WHILE j<= m do
            【  t2:= SCAO2[j]; j:=j+1;
               IF SimNetObjEq(t1, t2) THEN 【break; flag=false;】
            】
        IF flag THEN   【k:=k+1;        SCAO3[k] := t1;】
    】
    RETURN SCAO3;
ENDF.
```

4.2.2. Implementing algorithm of the intersection operation

When implementing the intersection operation $S_{CAO1} \cap S_{CAO2}$, get an element (suppose t) from S_{CAO1}, compare t with all the elements in S_{CAO2}. Add t to set S_{CAO3} if there is an element in S_{CAO2} to be equal to t. Get element from S_{CAO1} and compare it with elements in S_{CAO2} and add suitable element to S_{CAO3} until there is no element in S_{CAO1}, here S_{CAO3} is the result. Function intersection(S_{CAO1}, S_{CAO2}) is used to get the common elements of two CAO sets of SNO, and its pseudo code is shown below.

```
FUNC intersection (var SCAO1: ARRAY [1..n] of CAOSet, SCAO2: ARRAY [1..m] of
CAOSet): ARRAY [1..p] of CAOSet;
  i:=1; k:=0; SCAO3: ARRAY [1..p] of CAOSet;
  WHILE i<=n do
    【 t1:= SCAO1[i];    i:=i+1; j:=1; flag:=true;
      WHILE j<= m do
      【  t2:= SCAO2[j];          j:=j+1;
         IF SimNetObjEq(t1, t2) THEN 【break; flag=false;】
      】
       IF flag=false THEN 【k:=k+1; SCAO3[k] := t1;】
    】
    RETURN SCAO3;
ENDF.
```

4.2.3. Implementing algorithm of the difference operation

When implementing the difference operation $S_{CAO1} - S_{CAO2}$, get an element (suppose t) from S_{CAO2}, compare t with all element in S_{CAO1}. If there is an element in S_{CAO1} to be equal with t, delete this element from S_{CAO1}. Get element from S_{CAO2} and compare it with elements in S_{CAO1} until there is no element in S_{CAO2}, there S_{CAO1} is the result. Function difference(S_{CAO1}, S_{CAO2}) is used to get the different elements between S_{CAO1} and S_{CAO2}, and its pseudo code is shown below.

```
FUNC difference (var SCAO1: ARRAY [1..n] of CAOSet, SCAO2: ARRAY [1..m] of
CAOSet): ARRAY [1..p] of CAOSet;
```

S_{CAO3}:= S_{CAO1}; i:=1; k:=0;
WHILE i<=n do
【 t_1:= S_{CAO3}[i]; i:=i+1; j:=1; flag=true;
WHILE j<= m do
【 t_2:= S_{CAO2}[j]; j:=j+1;
IF SimNetObjEq(t_1, t_2) THEN 【break; flag=false;】
】
IF flag=false THEN deleteElementfromSet(S_{CAO1}, t_1); {delete element t_1 from set S_{CAO1}}
】
RETURN S_{CAO1};
ENDF.

Example 6. Given composite objects X and Y in Example 3, calculate $X \cap Y$ on the storage layer.

Answer: (1) firstly calculate the CAO set of SNO for X and Y respectively and the results are below.

S_{CAOX}={("personSetID": "01", "authors", {"authorID": "01", "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction"}), ("authorID": "01", "books", {"bookID": "01", "bookname": "Fantastic Voyage", "publishinghouse": "Bantam Doubleday Dell Publishing Group"}), ("authorID": "01", "books", {"bookID": "02", "bookname": "End of Eternity", "publishinghouse": "Grafton Books"})};

S_{CAOY}={("personSetID": "01", "authors", {"authorID": "01", "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction"}), ("authorID": "01", "books", {"bookID": "01", "bookname": "Fantastic Voyage", "publishinghouse": "Bantam Doubleday Dell Publishing Group"}, ("authorID": "01", "books", {"bookID": "05", "bookname": "The Mysterious Island", "publishinghouse": "Signet Classics"})};

(2)Secondly calculate the intersection of S_{CAOX} and S_{CAOY}.

$S_{CAOX} \cap S_{CAOY}$={("personSetID": "01", "authors", {"authorID": "01", "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction"}), ("authorID": "01", "books", {"bookID": "01", "bookname": "Fantastic Voyage", "publishinghouse": "Bantam Doubleday Dell Publishing Group"})}

4.3. The implementing algorithm of query operations

Query operations are unary operation. There is only one composite object (suppose O) taking part in the query operations. Suppose that all the SNOs in CAO form extracted from the CAO set of O is S_{CAO}.

4.3.1. The implementing algorithm of the projection operation

Suppose composite object O and projecting attribute-set E_P. If there is a leaf node in the tree created according to E_P to be a composite attribute, expand E_P according the schema of O until all the leaf nodes are atomic attributes, and suppose the expanded result E_{P1}. When implementing projection operation, for a CAO (suppose x) of a

SNO, if $x.A$ does not belong to E_{P1}, delete this CAO; if $x.A$ belongs to E_{P1}, delete from $x.O$ those components whose attributes does not belong to E_{P1}. The structure of a map in the schema of a composite object is defined below.

```
TYPE map=RECORD
   pre-image: String;
   image: ARRAY[1.. length] OF String;
END.
```

The structure of a node of attribute-tree is defined below. The algorithm for constructing an attribute-tree can refer to that of Section 4.1.1.

```
TYPE tattrnode=RECORD
    data: String;
   parents: String;
END.
```

Function expandAttrTree(E_P, mapSet) is used to expand an attribute-tree, and its pseudo code is shown below.

```
FUNC expandAttrTree (var Tattr: ARRAY [1..n] of tattrnode, mapSet: ARRAY
[1..m] of map): ARRAY [1..p] of String;
    i:=1; flag:=true;
    WHILE i<=n DO  {for each attribute in Tattr, determine whether it is a leaf
node}
        【  x:= Tattr.data[i]; i:=i+1; j:=1;
            WHILE j<=n DO
                【    y:= Tattr.parent[j]; j:=j+1 ;
                   IF x=y THEN 【flag:=false; break;】{x is not a leaf node
                                                 because it is the parents of a node }
                 】
            IF flag THEN {If x is a leaf node, determine whether it is a composite
attribute.}
            【 k:=1;
                    WHILE k<=m DO
                    【 z:= mapSet [k].pre-image; k:=k+1;
                       IF x=z THEN Sattr:=getAttrDescend(x); {x is a
                             composite attribute, get its all descendants}
                     】
           】
    】
    i:=1;
    WHILE i<=n DO 【EP1[i]:= EP[i]; i:=i+1;】{get all attributes from EP to EP1}
    la=length(Sattr); j:=1;
    WHILE j<= la DO 【EP1[i]:= Sattr [j]; i:=i+1; j:=j+1;】{ get all attributes from
                                                          Sattr to EP1}
    RETURN EP1;
ENDF.
```

In the pseudo code above, function getAttrDescend(T_{attr}, x) acquires all the descendant attributes of x. Stack S_1 and S_2 store the temporary data. S_1 stores the sub-attributes of each attribute, while S_2 stores all the descendants of x. Find all sub-attributes of x from T_{attr}, and then push them into S_1. Pop an element of S_1, and push it into S_2, find the sub-attributes of this element, then push the sub-attributes of this element into S_1. Repeat this procedure and all the descendants of x can be found. The data structure of the stacks is defined below.

```
TYPE Attributestack=RECORD
    data: ARRAY[1..n] OF String;
   top: integer;
END.
```

The pseudo code for function getAttrDescend(T_{attr}, x) is shown below.

```
FUNC getAttrDescend (var Tattr: ARRAY [1..n] of tnode, x: String): ARRAY [1..p]
of String;
i:=1; S1, S2: ARRAY[1..maxlen] OF Attributestack;
Sattr: ARRAY[1.. maxlen] OF String;
WHILE i<=n DO {Find the sub-attribute of x and push them into S1.}
【 y:= Tattr [i].parent; i:=i+1； IF x==y THEN push(Tattr [i].data, S1); 】
WHILE S1.top>0 DO  {this loop finds other descendants of x.}
【 j:=1; a=pop(S1); push(a, S2); {transfer the top element of S1 to S2.}
            WHILE j<=n DO  {find the sub-attributes of a}
             【      y:= Tattr [j].parent;
                      IF a=y THEN push(Tattr [j].data, S1); {push the sub-
                                                          attribute of a into S1}
             】
】
j:=1;
WHILE j<= S2.top DO Sattr[j]:= S2.data[j]; {read the attributes from S2 to Sattr.}
RETURN Sattr;
ENDF.
```

Function projection(E_P, S_{CAO}) is used to project E_P on S_{CAO}, and its pseudo code is shown below.

```
FUNC projection (EP1: ARRAY[1..m] OF String, SCAO: ARRAY[1..m] OF
CAOSet) :ARRAY[1..m] OF CAOSet;
i:=1; SCAO: ARRAY[1..p] OF CAOSet; tSNO: ARRAY[1..p] OF CAO; lt:=0;
WHILE i<=m DO
【 bSNO:=SCAO [i]; i:=i+1; lb:=length(bSNO); j:=1;
  WHILE j<=lb DO
    【 x:=b[j]; j:=j+1;
      IF elementInSet(x.A, EP1) THEN {If the A field of x is in EP1, modify
                                                    its O field.}
      【 k:=1; lxo=length(x.O); { get the length of x.O }
        WHILE k<= lxo DO {This loop modify the O field of x.}
        【 c:=x.O[k]; k:=k+1;
```

IF keyComponent(c)=false and elementInSet (attr(c), E_{P1})=false
THEN delete($x.O$, c); { If c is not a key component and its attribute is not in E_{P1}, delete c from $x.O$.}
】
lt:= lt+1; t_{SNO} [lt] :=x; {put x in t_{SNO} after modifying x.}
】
】
】 RETURN t_{SNO};
ENDF.

In the pseudo code above, b_{SNO} and t_{SNO} are the CAO sets of SNO. Function attr(c) is to get the attribute of c.

4.3.2. The implementing algorithm of the selection operation

The conditional expressions defined in the representation layer is "(attribute$_1$ CS value$_1$) LS(attribute$_2$ CS value$_2$)…LS(attribute$_n$ CS value$_n$)". CS is the shorthand of comparing symbol (<, ≤, ≥, >, ≠, =) and LS is that of logic symbols (∧, ∨). When determining whether a SNO can meet the conditional expression, for each attribute which appears in the expression, find it in the "O" field of the CAO in the CAO set of the SNO, and then use the value of the attribute to replace itself in the expression, last calculate the expression after all the attributes are replaced by their values. If the value of the expression is true, keep the SNO, otherwise, discard the SNO. The data structure of the unit of the conditional expression is defined below. The LS in the first expression unit is null.

TYPE ExpressionUnit=RECORD
LS: String;
attribute: String;
CS: String;
Value: String;
END.

Function selection (F, S_{CAO}) is used to do selection on S_{CAO} using condition expression F, and its pseudo code is shown below.

FUNC selection (F: ARRAY[1..n] OF ExpressionUnit, S_{CAO}: ARRAY[1..m] OF CAOSet): boolean;
i:=1;
WHILE i<=m DO {this loop is to replace all the attribute in F with its value in S_{CAO}.}
【 x_{SNO}:= S_{CAO} [i]; {read the ith SNO}
l_{SNO}:=length(x_{SNO}); {get the length of x_{SNO}. The length is also the number of CAO in x_{SNO}.}
j:=1;
WHILE j<= l_{SNO} DO
【 y_{CAO} :=x_{SNO} [j]; {read the j-th CAO of x_{SNO}.}
k:=1; l_O:=length($y_{CAO}.O$); {get the length of $y_{CAO}.O$. }

```
                WHILE k<= l_O DO
              【   c:= y_CAO.O [k]; {read the k-th component of y_CAO.O.}
                   p:=1;
                   WHILE p<=n DO
                   【   exp:=F[p];
                      IF exp.attribute:=getAttr(c) THEN { getAttr (c) is to get the
                                                          attribute of c.}
                     F[p].attribute:=getVal(c) ; { getVal (c) is to get the value of c.}
                              IF exp.LP="∧" THEN exp.LP:="and";
                              ELSE exp.LP:="or";
                              p:=p+1;
                   】
                   k:=k+1;
              】
             j:=j+1;
         】
         i=i+1;
  】
  i:=1;
  result1:=true;  {set the initial value for the previous expression unit}
  WHILE i<=m DO
  【   result2:=true; {set the initial value for the current expression unit}
       CASE
              F[i].CP=">": result2:= result2 and (strToVal(F[i].attribute>
strToVal (F[i].value)));
        F[i].CP="<": result2:= result2 and (strToVal(F[i].attribute< strToVal
    (F[i].value)));
        F[i].CP=">=": result2:= result2 and (strToVal(F[i].attribute>= strToVal
    (F[i].value)));
        F[i].CP="<=": result2:= result2 and (strToVal(F[i].attribute<= strToVal
    (F[i].value)));
        F[i].CP="=": result2:= result2 and (strToVal(F[i].attribute= strToVal
    (F[i].value)));
        F[i].CP="≠": result2:= result2 and (strToVal(F[i].attribute≠strToVal
        (F[i].value)));
              { strToVal () is to convert a string to a number}
    ENDC;
    CASE
      F[i].LP="and":  result1:= result1 and result2;
      F[i].LP="or":  result1:= result1 or result2;
      F[i].LP=NIL:  result1:= result1 and result2;
    ENDC;
  】
  RETURN result1;
ENDF.
```

4.3.3. The implementing method of query operation including the projection operation and the selection operation

The result of doing the projection operation firstly and doing the selection operation secondly is the same as the result of doing on contrary, but it is more effective to do the projection operation firstly. It is time consuming to extract SNOs from a CAO set, so the smaller of the CAO set the more effective of extracting SNO.

Example 7. Given the composite object in Example 3, do projection operation and selection operation. The projecting attribute-set is E_P={“authors”, “lastName”, “books”}, and the condition of selection operation is F=“bookName”: “Fantastic Voyage”.

Answer: (1) do the projection operation. First, expand E_P to be {“authors”, “lastName”, “books”, “bookName”, “publishinghouse”}; second, do the projection operation on S_{CAO} of the composite object. S_{CAO} is the same as that of Example 5. The projection result is shown below.

S_{CAO1}={{(“personSetID”: “01”, “authors”, {“authorID”: “01”, “lastName”:“Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})}, {(“personSetID”: “01”, “authors”, {“authorID”: “01”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: “Grafton Books”})}, {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “03”, “bookname”: “Empire of the Ants”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})}, {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “04”, “bookname”: “Le Papillon DES Etoiles”, “publishinghouse”: “Librairie generale francaise”})}}

(2)Do the selection operation on S_{CAO1}. There are 4 SNOs in S_{CAO1}. Take each SNO from the set and use the conditional expression on each SNO. The procedure is shown below.

(a) {(“personSetID”: “01”, “authors”, {“authorID”: “01”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})};

For this SNO, the value of “bookName” is “Fantastic Voyage”. Use the value to replace the attribute, and the same below.

F: “bookName”=“Fantastic Voyage”-> “Fantastic Voyage”=“Fantastic Voyage” ->TRUE;

(b) {(“personSetID”: “01”, “authors”, {“authorID”: “01”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “02”, “bookname”: “End of Eternity”, “publishinghouse”: "Grafton Books”})};

“bookName”=“Fantastic Voyage”-> “End of Eternity”=“Fantastic Voyage” ->FALSE;

(c) {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “03”, “bookname”: “Empire of the Ants”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})};

“bookName”=“Fantastic Voyage”-> “Empire of the Ants”=“Fantastic Voyage” ->FALSE;

(d) {(“personSetID”: “01”, “authors”, {“authorID”: “02”, “lastName”: “Williams”}), (“authorID”: “02”, “books”, {“bookID”: “04”, “bookname”: “Le Papillon DES Etoiles”, “publishinghouse”: “Librairie generale francaise”})};

“bookName”=“Fantastic Voyage”-> “Le Papillon DES Etoiles”=“Fantastic Voyage” ->FALSE ;

For 4 SNOs in S_{CAO1}, the first SNO meet the condition expression, and the result is shown below.

S_{CAO2}={{(“personSetID”: “01”, “authors”, {“authorID”: “01”, “lastName”: “Asimov”}), (“authorID”: “01”, “books”, {“bookID”: “01”, “bookName”: “Fantastic Voyage”, “publishinghouse”: “Bantam Doubleday Dell Publishing Group”})}}.

5. Conclusion

A cloud-oriented two-layer data model is proposed, in which the composite object is defined to model the nested data in the representation layer and the conceptions of CAO; the schema and a 4-triple are defined to model the non-nested data in the storage layer. The formal definitions of the algebraic operations consisting of the set operation and the query operation are proposed. The conception of SNO is proposed as the basic operational unit for the algebraic operations. Own to this conception, these operations have simple expression and can ensure a correct result. The attribute-tree is defined and is applied in the query operation to offer attributes selection. The ideas of implementing the algebraic operations on the storage are proposed; the implementing pseudo code of these algorithms is also designed. The algorithm of extracting all SNOs on their CAO form from the CAO set of a composite object is proposed as foundation of the algorithms of the algebraic operations. The conception of parent-child relation of CAO and that of CAO-tree are defined and applied in extracting SNO. Logic proof and example proof indicate that the formal definitions and the implementing pseudo code of the algebraic operations are correct. Implementing the two-layer cloud database management system based on these algebraic operations is seen as future work.

***Acknowledgements*:** This paper is supported by Scientific Research Foundation of the Higher Education Institutions of Guangxi, China (Grant No KY2015YB188), the National Natural Science Foundation of China (Grant No 61363074), and Guangxi Natural Science Foundation of China (Grant No 2015GXNSFAA139306).

References

1. Lin, Z.-Y., Y.-X. Lai, C. Lin et al. Research on Cloud Databases. – Journal of Software, Vol. **23**, 2012, No 5, pp. 1148-1166.
2. Chang, F., J. Dean, S. Ghemawat et al. Bigtable: A Distributed Storage System for Structured Data. – ACM Trans. on Computer Systems, Vol. **26**, 2008, No 2, pp. 1-26.

3. Amazon SimpleDB – Simple Database Service.
http://aws.amazon.com/cn/simpledb/
4. Cooper, B. F., R. Ramakrishnan, U. Srivastava et al. Pnuts: Yahoo!'s Hosted Data Serving Platform. – In: Proc. of VLDB Endowment, Vol. **1**, 2008, No 2, pp. 1277-1288.
5. APACHE HBASE.
http://hbase.apache.org/
6. Amazon Dynamo.
http://aws.amazon.com/cn/dynamodb/
7. Qi, J., L. Qian, Z. G. Luo. Distributed Structured Database System Huge Table. – In: M. G. Jaatun, G. S. Zhao, C. M. Rong, Eds. Proc. of 1st Int. Conf. on Cloud Computing (CloudCom'09), Berlin, Springer-Verlag, 2009, pp. 338-346.
8. Chodorow, M., D. B. Mongo. The Definitive Guide Dirolf. Kristina O'Reilly Media, Inc., USA 2010, p. 9.
9. Anderson, J. C., J. Lehnardt, N. Slater. CouchDB: The Definitive Guide. O'Reilly Media, Inc., USA, 2010.2.3.
10. Amazon Relational Database Service.
http://en.wikipedia.org/wiki/Amazon_Relational_Database_Service
11. Gaian, D. B. Easily Accessing All Your Distributed Research Data.
http://cmg.soton.ac.uk/events/event-610/
12. The Architecture of SQL Azure.
http://zh.wikipedia.org/wiki/SQL_Azure
13. Guo, X.-M., X.-G. Huo, Y. He. Cloud Database Design Base on the Requirement of Management Information Ontology. – Mathematics in Practice and Theory, 2014, No 44, pp. 117-122.
14. Liu, Z., Z. Wen, H. Zhang. Cloud Computing and Cloud Data Management Technology. – Journal of Computer Research and Development, 2012, No 49, pp. 26-31.
15. Abadi, D. J. Data Management in the Cloud: Limitations and Opportunities. – In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2009, pp. 1-10.
16. Cloud Database of 2013 – Trends and Predictions.
http://cloudtweaks.com/2013/06/cloud-database-of-2013-trends-and-predictions/
17. Li, Y., Y. Lu. A Two-Layer Cloud Database Model and Its Bidirectional Conversion Algorithms. – In: Proc. of 2016 IEEE 7th International Conference on Software Engineering and Service Science, 2016, pp. 289-294.