International Journal of Software Engineering and Its Applications
Vol. 10, No. 5 (2016), pp. 157-166
http://dx.doi.org/10.14257/ijseia.2016.10.5.14

Object Connection Hypergraphs-an Approach for Nested Object Query Optimization

Hung Hoang Bao [(1)], Phuong Ngo Viet [(1)] and Thanh Le Manh [(2)]

[(1)] *Korea - Vietnam Friendship Information Technology College, MIC, Vietnam*
[(2)] *Hue University, MOET, Vietnam*
hunghb@viethanit.edu.vn, phuongnv@viethanit.edu.vn

Abstract

In Object-Oriented Databases (OODB), nested object queries are used regularly. Nested structures are put in conditional expressions of the queries in two forms: nested sub-queries or path expression containing hidden joins – nested predicates in WHERE clauses. For nested queries, when analyzing the estimated cost of the nested algebraic expression, the expression evaluation result gives out an ineffective cost. Therefore, our method proposed in this paper will resolve the problems by leveling nested sub-queries in the nested queries. This method will increase the effectiveness of the query processing cost – We use object connection hypergraphs to present nested queries.

Keywords*: hyper-graphs; optimization method; object queries; algorithm*

1. Introduction

In order to process repeated (nested) predicates, Cho W. [2] introduces a method of cost estimating which is dependent of rates between objects of the beginning class in the path expression and total objects of the class basing on the many-to-many relationship of classes. This rate is one of parameters selected in the physical database designing process.

For nested sub-queries, Cluet S. [8] proposes an optimization method with two steps. Firstly, the queries are transformed syntactically to process common sub-expressions and independent sub-queries effectively. Then, the queries are compiled into nested algebraic expressions using an algebraic transformation. However, when analyzing the estimation of nested loops in algebraic expressions, we recognize that the result expression contains an ineffective cost. Therefore, our method presented later in this article will help solve the problem of processing nested sub-queries using a "leveling" method for nested queries. The method is about the reduction of object connection hyper-graphs, and this makes the estimating method more effective [4].

In this article, inspired by the idea of the query representation and optimization using hypergraphs of Ullman J.D [7] and Han [3], we propose the concept of *object connection hypergraphs* to represent queries written in OQL (Object Query language), especially to process nested queries. Furthermore, we also introduce algorithms to estimate hyperedges and the algorithm of reducing object connection hypergraphs.

2. Object Query Representation using Hypergraphs

We define the formal concept of object connection hypergraphs as follows [4]:

Definition. *An object connection hypergraphs* is a hexa-tuple

$\mathcal{H} = (C_H, V_H, E_H, L_H, s_H, lb_H)$, in which:

ISSN: 1738-9984 IJSEIA

(i) C_H is a finite set of classes involved in the query

(ii) V_H is a finite set of nodes

(iii) L_H is a finite set of labels

(iv) $E_H = E_C \cup E_Q$ – a set of hyperedges finite, in which E_C, E_Q is sets of hyperedges representing classes of objects and elements of the query

(v) $s_H: V_H \rightarrow E_H$ is a mapping initiating hyperedges from the set of nodes

(vi) $lb_H: E_H \rightarrow L_H$ is a function labeling hyperedges, so that $\forall\, e \in E_H\ lb_H(e) \in L_H$

Example 2.1. We consider an object connection hypergraphs represented as below:

select A

from c_1, c_2, c_3

where $c_1.A = c_2.F$ and $(c_1.A + c_1.B > c_3.D)$ and $(c_3.E \leq c_2.G)$

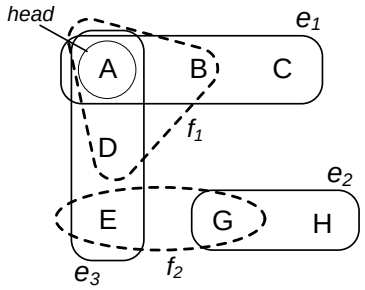

Figure 2.1. Objects Connection Hypergraphsof Example 2.1

in which we have:

$C_H = \{c_1, c_2, c_3\}$, c_1 = (A, B, C), c_2 = (G, H) and c_3 = (D, E, F) are object classes.

V_H = {A, B, C, D, E, F, G, H}: set of nodes, $L_H = \{e_1, e_2, e_3, f_1, f_2, \text{“}head\text{”}\}$: labels, $E_H = E_C \cup E_Q$, with E_C is the set of labelled hyperedges $\{e_1, e_2, e_3\}$representing classes c_1, c_2, c_3. And E_Q has hyperedges that represent results of the queries consequently; the conditional expression of the correspondent query is labelled as $\{f_1, f_2, \text{“}head\text{”}\}$. With “$c_1.A = c_2.F$” condition, we do “merging” of two nodes labelled “A”.

From definition, object connection hypergraphs are hereafter called as hypergraphs, we use hypergraphs notation to represent OQL queries as follows:

- A set of hypergraphs nodes is the set of properties that belongs to classes involved in the query. Each property of class $\boldsymbol{c_i}$ is represented by a node. If two classes $\boldsymbol{c_i}$ and $\boldsymbol{c_j}$ have inherited properties from a hyperclass, or they both inherit all properties of a hyperclass, we still create separate nodes for these properties.

- *Hyperedges)* of the hypergraph are created from conditional expressions and classes $\boldsymbol{c_i}$:

We consider the conditional expressions in a where clause, they are divided into following forms:

+ A = a (2.1)

+ A = B (2.2)

+ A θ B, θ ∈ { < , ≤ , ≠ , > , ≥} (2.3)

+ A θ B, θ ∈ {⊂ , ⊆ , ≠ , ⊃ , ⊇} (2.4)

where, A and B are properties of classes and *a* is a constant.

+ Hyperedges) are sets of finite nodes representing classes, called *object hyperedges*; an object hyperedge is drawn as a close line around nodes of the hyperedge. The hyperedge is labelled with the class name.

+ For each conditonal expression in forms of (2.3) or (2.4) (1.4), we will create a hyperedge containing properties in the expression. These hyperedge are called *conditional hyperedge* and they are represented by close dashed lines.

+ The conditon of form (2.1) will become the label "A = *a*" of the node representing the correspondent property.

+ The conditional expression of form A = B (2.2), with A, B are properties in two classes (they are probaly properties inherited from a certain hyperclass, then we choose a representative property and label it by name of one property.

- If there are two conditions of only one set of properties, we must label hyperedges separately in order to distinguish them.

- Properties in a select clause are bordered in a close line and labelled as "*head*", the so-called *top hyperedges*. The top hyperedges is correspondent to a class – the result of the query.

- *Association hyperedges* contain properties in expressions having {IS, IN, UNION, FORALL, EXIST,...} operators of nested sub-queries. The edges are drawn by a closed dashed line. Association hyperedges are lebelled with the correspondent operator names.

A single query only have a select...from...where (SFW) pattern; Nested queries in OQL have more than one SFW pattern. The nested query represented by a hypergraph is built from hypergraphs of single SFW patterns and connected together with association hyper-edges.

We formally represent the object schema $S = (s_1, \ldots, s_n)$, in which s_i are classes in S and the object query $QE = (s_1, \ldots, s_m, R, p_1, \ldots, p_k)$ in which s_i ($i = 1,\ldots, m$) are classes in the query, R is result class/type of the query; and p_j ($j = 1, \ldots, k$) are conditional expressions in where clause.

Algorithm 2.1: Object Query's Hypergraph Initialization (not containing nested query).

Input: Object schema $S = (s_1, \ldots, s_n)$

Object query $QE = (s_1, \ldots, s_m, R, p_1, \ldots, p_k)$

Output: Hypergraph $\mathcal{H}$

Method:

(1) SC := ∅ //set of object hyperedges of Hypergraph $\mathcal{H}$

(2) $V := (s_1, \ldots, s_m)$

(3) **for** $s_i \in V$ **do**

(4) **if** (s_i is initial hyperclass) **then** //not inherited from other hyperclasses

(5) Initialize object hyperedges $e = s_H(\{s_i\})$ and label $lb_H(e)$

(6) **else if** (s_i is single or multiple inherited class) **then**

(7) Process the problem of name conflicts with inherited properties.

(8) Initialize class s_i' containing class properties and inherited properties

(9) Initialize object hyperedges $e = s_H(\{s_i'\})$ and label $lb_H(e)$

(10) SC := SC $\cup$ e

(11) Initialize top hyperedges $h = s_H(\{R\})$ and $lb_H(h)$ = "*head*"

(12) SC := SC $\cup$ h

(13) SD := $\varnothing$ //set of condition hyperedges of hypergraphs $\mathcal{H}$

(14) **for** $p_i \in (p_1, \ldots, p_k)$ **do**

(15) **if** (p_i is in forms of (2.3) and (2.4) **then**

(16) Initialize conditional hyperedges $f = s_H(\{p_i\})$

(17) SD := SD $\cup$ f

(18) **else** Label the node $lb_H(e)$ = "= a"

(19) $\mathcal{H}$:= SC $\cup$ SD

From the definition we confirm that algorithm 2.1 is correct and its computational complex is $O(n^2)$, with n is the number of classes in the query.

Now we build an algorithm of initializing a hypergraph that represents nested queries. From the steps of forming a single hypergraph, we create the result hypergraph from connections of single hypergraphs with association hyperedges.

Algorithm 2.2: Nested OQL Query's Hypergraph Initialization.

Input: Object schema $S = (s_1, \ldots, s_n)$

Query $QE = (s_1, \ldots, s_m, R, p_1, \ldots, p_k)$, $TT \subseteq$ {is, in, union, diff, forall, exists} is a set of operators in where clause of query QE.

Output: Hypergraph $\mathcal{H}$.

Method:

(1) $\mathcal{H}$:= $\varnothing$

(2) **for** (each sub-query $QE_i \in QE$) **do**

(3) Initialize Hypergraph $\mathcal{H}_i$ with QE_i (Algorithm 2.1)

(4) $\mathcal{H}$:= $\mathcal{H} \cup \mathcal{H}_i$

(5) **for** (each operator $t_i \in TT$) **do**

(6) Initialize association Hyperedges g with label t_i containing the top hyperedges of the hypergraph in the right side of t_i and properties in the left side of t_i

(7) $\mathcal{H}$:= $\mathcal{H} \cup g$

Example 2.2. Find the names of all students living in the same city with lecturers having name as "Hue".

```
define  Student as  p1
select (p1.name)
from p1, p2 Employee
where p1.city = p2.city AND p2.name = "Hue"
```


The query in Example 2.2 contains a value-based join (p1.city = p2.city) and is labelled as "city", we merge two nodes labelled "city".

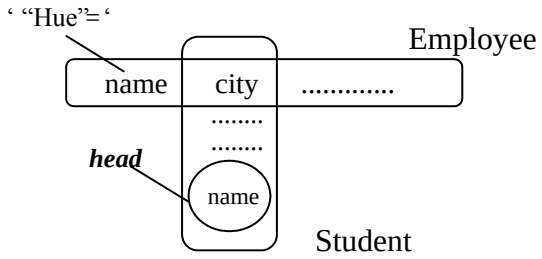

Figure 2.2. Hypergraph Representation of Example 2.2

Example 2.3. We consider a query: finding names of employees in departments granted the budget of more than 250 (unit: milion VND) and having salary level equal or greater than 2.4.

```
select  e.name
from    Employee as e
where  e.salary >= 2.4 AND e.dept IN ( select  s.dept
                                       from    Department as s
                                       where  s.budget > 250)
```

The hypergraph in example 2.3 is initialize as follows: The object hyperedges express classes Employee and Department. For top hyperedges, we have two top hyperedges: *e*.name – top hyperedge query result, *s*.dept – hyperedge of nested SFW patterns. Two conditional hyperedge *e*.salary *>= 2.4*, *s*.budget *> 250* and association hyperedge *e*.dept are labelled **IN**.

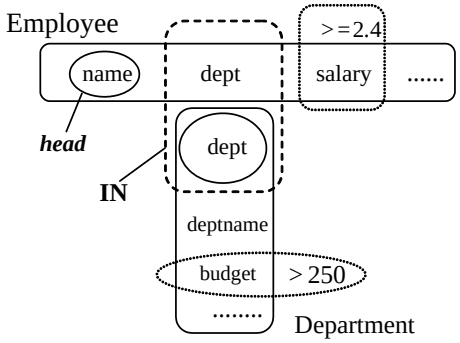

Figure 2.3. Hypergraph Representation of Example 2.3

3. Hypergraph Estimation Method

3.1. Hyperedge Estimation

We formally express the hypergraph of object-oriented query as series of events: $\mathcal{H} = (E_1, E_2, ..., E_n)$, in which events E_i is probably an object hyperedge, a conditional hyperedge or an association hyperedge [4].

The derived class obtained after the effect of an event E_j is noted as $DerivedClass(E_1, ..., E_j)$, in which E_1 must be an object hyperedge (in case the hypergraph) has only one hyperedge, that edge must be an object hyperedge.

The *HyperedgeEstimation* procedure *receives the derived class obtained after the effect of events* E_{j-1} and E_j, the procedure result is the derived class of event E_j in $\mathcal{H}$.

Procedure *HyperedgeEstimation* (*DerivedClass*(E_1, ..., E_{j-1}), E_j)

Input: *DerivedClass* (E_1, ..., E_{j-1}) and the event E_j

Output: *DerivedClass*(E_1, ..., E_j)

Method:

(1) Initialize, *HyperedgeEstimation*(E_1) give out the result: *class DerivedClass* (E_1) = c_1,

in which, c_1 is the correspondent class of the object hyperedge E_1.

(2) **if** (E_j is a condition or a conditional hyperedge) **then**

DerivedClass (E_1, ..., E_j) = σ_F(*DerivedClass* (E_1, ..., E_{j-1}))

in which F is the correspondent conditional expression of E_j

(3) **if** (E_j is a object hyperedge) of class C_j intersecting the hypergraph) **then**

DerivedClass (E_1, ..., E_j) = *DerivedClass* (E_1, ..., E_{j-1})) $\bowtie$ C_j

(4) **if** (E_j is a object hyperedge not intersecting the hypergraph) **then**

DerivedClass (E_1, ..., E_j) = *DerivedClass* (E_1, ..., E_{j-1})) $\times$ C_j

In example 2.1, S = (Employee, Student), we have:

DerivedClass (Employee, Student) = *DerivedClass* (Employee) $\bowtie$ Student.

When all hyperedges of hypergraph $\mathcal{H}$ are estimated by affecting events in turn to obtain the derived classes. The result derived class will be projected over the set of properties in the top hyperedge – This is the answer of the query.

Algorithm 3.1: Estimating hyperedge of a hypergraph.

Input: The hypergraph $\mathcal{H}$ = (E_1, E_2, ..., E_n), R is a top hyperedge.

Output: Result class of the query.

Method:

(1) Expressing the hypergraph $\mathcal{H}$ = (E_1, E_2, ..., E_n), with a series of events E_i

(2) **for** j = 1 **to** n **do**

(3) **Call** *HyperedgeEstimation*(*DerivedClass*(E_1, ..., E_{j-1}), E_j)

(4) **Append** *DerivedClass*(E_1, ..., E_j) into $\mathcal{H}$

(5) $\mathcal{H}$ = π_R(*DerivedClass*(E_1, ..., E_n))

Clause 3.1. Algorithm 3.1 completes after finite steps and gives the correct answer.

Demonstration

Certainly, the number of events in $\mathcal{H}$ is finite, so that algorithm 3.1 will complete after n hypergraph $\mathcal{H}$.

In order to prove that algorithm 3.1 returns a correct answer of given query, we prove inductively as follows:

In the base case: $n = 1$, then $\mathcal{H} = (E_1)$, E_1 is an object hyperedge, we have:

$\mathcal{H} = \pi_R(E_1) = \pi_R(C_1)$ – is the query's answer.

Assuming that k^{th} derived class obtained after the effect of event E_k is the estimation of k hyperedges in the hypergraph (*DerivedClass* (E_1, ..., E_k)). In another hand, the derived class obtained in the k^{th} step is the input of the $(k+1)^{th}$ estimation step, therefore if $k = n$ then after n hyperedge estimation steps, we obtain the derived class:

HyperedgeEstimation (*DerivedClass* (E_1, ..., E_{n-1}), E_n) = *DerivedClass* (E_1, ..., E_n)

Next, we project over top hyperedge and consequently have the query result:

$\mathcal{H} = \pi_R$(*DerivedClass*(E_1, ..., E_n)) – the query result. □

Example 2.4. Considering the query in example 2.2, we have a series of events in hypergraph $\mathcal{H}$ = (Employee, name = "Hue", Student).

Applying algorithm 3.1, we have estimation steps as follow: (1) Initialize the correspondent derived class to object hyperedge Employee, (2) Apply the selection condition "name = "Hue"" on the object hyperedge Employee, (3) Esitmate the object hyperedge Student, that is the traditional value-based join, (4) Project the derived class obtained after connection to top hyperedge (name).

We can recognize from example 2.4 thatthe order of hyperedges in $\mathcal{H}$ will give different series of estimation steps according to the order of sorting executed operators. This is one of the parameters that determines searching space of query execution methods.

In the algorithm 3.1, we have not processed the case of association hyperedge yet, i.e. considering hypergraphs expressing nested object queries. The nested queries have sub-queries execution order from "inside to outside", i.e. the queries in the deepest level will be executed first. Assuming that hypergraphs expressing nested object queries are described formally as a series of events, we build an algorithm to reduce the hypergraph $\mathcal{H} = (E_1, E_2, \ldots E_i, EA_j, E_{i+1}, \ldots, E_k, \ldots)$, where E_i are objects or conditonal hyperedges, EA_j are association hyperedge.

We extend algorithm 3.1 by processing association hyperedges as follows: Firstly, we estimate all hyperedges E_l ($l = i + 1, \ldots, k$) after the association hyperedge EA_j, and without loosing the generality, we assume that E_k is the last hyperedge estimated (before estmating the association hyperedge EA_j). Secondly, we affect the event EA_k over the derived classe obtained after affecting event E_k (EA_j'). Therefore, $\mathcal{H}$ is rewriten as $\mathcal{H} = (E_1, E_2, \ldots E_i, EA_j', E_{i+1}, \ldots, E_k, EA_j, \ldots)$, EA_j is derived from the result class after estimating the association hyperedge.

Algorithm 3.2: Hypergraph Reduction

Input: Hypergraph $\mathcal{H} = (E_1, E_2, \ldots E_i, EA_j, E_{i+1}, \ldots, E_k, \ldots)$, R is a hyperedge.

Output: Result class of the query.

Method:

(1) Express hypergraph $\mathcal{H} = (E_1, E_2, \ldots E_i, EA_j, E_{i+1}, \ldots, E_k, \ldots)$

(2) $i := 1$

(3) **repeat**

(4) **if** (EA_j is a association hyperedge) **then**

(5) **for** $l = i + 1$ **to** $k - 1$ **do**

(6) **Call** *HyperedgeEstimation*(*DerivedClass*(E_j), E_{j+1})

(7) **Append** *DerivedClass* (E_l, ..., E_k) vào $\mathcal{H}$ trước EA_j

(8) **else** **Call** *HyperedgeEstimation*(*DerivedClass*(E_1, ..., E_{i-1}), E_i)

(9) **Append** *DerivedClass*(E_1, ..., E_i) vào $\mathcal{H}$

(10) **inc**(i)

(11) **until** (estimated all hyper-edges in $\mathcal{H}$)

(12) $\mathcal{H} = \pi_R$(*DerivedClass*(E_1, ..., E_k, …))

From Clause 3.1, we can say algorithm 3.2 will complete and give a correct result as an answer. The computational complex of algorithm 3.2 is expressional.

3.2. Query Searching Space

Methods of the query execution in the searching space are considered upon selection possibilities, the estimation of classes and hyperedge of the hypegraph. From then, with algorithms of the estimation and reduction of hypergraph (3.1 and 3.2 algorithms), we will create the searching space of query execution methods as follows (algorithm 3.3): We use a set of lists having with elements containing items of the hypergraph, then we traverse correspondent lists to determine solutions.

Algorithm 3.3. Query searching space.

Input: Hypergraph $\mathcal{H}$

Output: Searching space with all query execution methods

Method:

(1) Sorting classes, object hyperedges, conditions and association hyperedges into a set of lists {L_1}

// Step 1: Estimate condition hyperedges and association hyperedges.

(2) **for** each list L_1 **do**

(3) **for** each hyper-edge E **do**

(4) **if** E is an association hyperedge **then**

(5) Add EA_j into L_1 after the last hyperedge E_k //algorithm 3.2

(6) **else if** E is condition hyperedge **then**

(7) Estimate condition hyperedges//algorithm 3.1

(8) Obtained result is the list {L_1'}

// Step 2: Estimate object hyperedges

(9) **for** each list L_1' **do**

(10) **for** each object hyperedge **do**

(11) Estimate correspondent conditional hyperedge

(12) The result stored in the list {L_2}

// Step 3: Estimate joins

(13) **for** each list L_2 **do**

(14) **for** each hyperedge **do**

(15) Estimate joins over on classes

(16) The result is the list $\{L_3\}$;

The set of $\{L_3\}$ lists is the search space of query execution methods.

We note **KGTK** is the total of methods of object query execution in the search space.

Theorem 3.2. The searching space in algorithm 3.3 has the total of query execution methods as follows:

$$(p + q!+ 2^{q-1}) \leq \textbf{KGTK} \leq (q! + p2^{q-1})$$

in which q is the number of hyperedges of the hypergraph, p is the cost of algebraic operations.

Demonstration

The searching space in algorithm 3.3 is determined by following parameters: (i) The sorting order of hyperedges in the list, (ii) The cost of operation execution, and (iii) The cost of joins in the query. For (i), if q is the number of hyperedges then the number of alternatives of the hyperedge order in the list L_1 is $q!$. Parameters in (ii) depend on the design of the system and the cost of joins on q hyperedges is 2^{q-1} (iii). So that, total of query execution methods are:

$$(p + q! + 2^{q-1}) \leq \textbf{KGTK} \leq q! + p2^{q-1}$$

In the worst case, if the methods in estimation steps are independent and this result list is the input of the next step then **KGTK** = $q! + p2^{q-1}$, otherwise **KGTK** = $q! + p + 2^{q-1}$.□

4. Conclusion

Object-oriented query optimization has attracted many researchers with results based on different approaches such as the optimization method using the query processing cost, the path expression optimization in object-oriented queries, methods of object horizontal and vertical partitioning. In this article, we propose an object-oriented query optimization developed from hypergraph approaches of Ullman J.D [7] and Han [3] to solve the problem of nested object queries by the algorithm of estimation on object hyperedge with a query processing cost which is more effective than the methods of object algebraic expression transformation and the path expression optimization. Moreover, by using object hypergraph notation we can express and optimize complex object queries. The result of this article is also implemented in ObjectStore, an object-oriented database management system. The issue of building a general algorithm to reduce object hypergraph and combining the querying space with the query processing cost model will be the research issue and presented in incoming articles.

References

[1] Doan Van Ban, Le Manh Thanh and Hoang Bao Hung, The Equivalent Expression between the OODB Query Language oql and the Object Algebra, Journal of Computer Science and Cybernetics, Vietnamese Academy of Science and Technology, Vol. 20 (3) **(2004)**, pp. 257–269.
[2] Cho Wan-Sup, Han Wook-Shin, Hong Ki-Hyung and Whang Kyu-Young, Estimating Nested Selectivity in Object-Oriented Databases, *ACM* **(2000)**, pp. 94–101.
[3] Han, Jia Liang, Optimizing Relational Queries in Connection Hypergraphs: Nested Queries, Views, and Binding Propagations, *The VLDB Journal*, 7 **(1998)**, pp.1–11.
[4] Le Manh Thanh, Doan Van Ban and Hoang Bao Hung, The Method for Estimating the Nested Queries in Object - Oriented Databases by Connection Hypergraphs, Special Issue of Posts, Telecommunications and Information Technology Journal, *"Research and Development on Telecommunications and Information Technology"*, ISSN 0886 – 7039, Vol. 14 **(2005)**, pp. 43–49.
[5] Trigoni A., *Semantic Optimization of OQL Queries*, Technical Report, Number 547, University of Cambridge, Computer Laboratory, UCAM-CL-TR-547, ISSN 1476-2986 **(2002)**.
[6] Cattel R.G.G., Barry D.K., *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann Publishers, **(2000)**.

[7] Ullman J.D., *Principles of Database and Knowledge base Systems, Vol I, II,* Computer Science Press, Rockville, **(1989)**.
[8] Cluet, Sophie and Moerkotte, Guido, Nested Queries In Object Bases, *In Fifth International Workshop on Database Programming Languages*, Italy **(1995)**.
[9] Trigoni A. and Bierman G.M., Inferring the Principal Type and the Schema Requirements of an OQL Query, *In 18th British National Conference on Databases* (BNCOD) **(2001)**, pp.185–201.

Authors

Hung Hoang Bao, graduated from Hue University of Education, Vietnam in 1993, majoring in Mathematics. In 2002, Master Degree in IT, Hanoi Polytechnic University. In 2007, PhD thesis at the Institute of IT, Vietnam Academy of Science and Technology. Currently working at the Korea-Vietnam Friendship IT College under the management of the Ministry of Information and Communications. His research interests include object database, GIS, cloud computing.

Phuong Ngo Viet, Bachelor of Science, majoring in Physics, University of Hue in 1996. In 2005, Master Degree in IT at University of Da Nang. Currently working at the Korea-Vietnam Friendship IT College under the management of the Ministry of Information and Communications. His research interests include semantic web; Ontology engineering.

Thanh Le Manh, Graduated from Technology University of Hanoi in 1977, majoring in engineering mathematics works. In 1993 PhD in Hungary, majoring in Mathematics Guarantee for electronic computers. We work at the University of Hue, Vietnam. His research interests include database derived databases and object database.

