Intelligent Query Processing Using a Meta-Database KaDB

Soon-Young Huh*, Kae-Hyun Moon**

Graduate School of Management
Korea Advanced Institute of Science and Technology
207-32 Cheongryangri-dong, Dongdaemun-gu, Seoul, Korea
syhuh@green.kaist.ac.kr, khmoon@green.kaist.ac.kr
Tel: 82-2-958-3626
Fax: 82-2-958-3604

Abstract

Query language has been widely used as a convenient tool to obtain information from a database. However, users demand more intelligent query processing systems that can understand the intent of an imprecise query and provide additional useful information as well as exact answers. This paper introduces a meta-database and presents a query processing mechanism that supports a variety of intelligent queries in a consistent and integrated way. The meta-database extracts data abstraction knowledge from an underlying database on the basis of a multilevel knowledge representation framework KAH. In cooperation with the underlying database, the meta-database supports four types of intelligent queries that provide approximately or conceptually equal answers as well as exact ones.

Key words: Intellignet query processing, Data Abstraction, Approximate answers.

1. Introduction

Conventional database systems do not admit inaccurate or vague queries and provide null information when there exist no exact answers. Thus, database users are required to precisely understand both the database schema and the problem domain knowledge. Several approaches [7, 9, 10, 16, 18, 20, 26] have been proposed to remedy such shortcomings and to enhance the effectiveness of information retrieval. They support fault-tolerant and intelligent query processing that can analyze the intentions of a vague query and provide neighborhood information relevant to the query as well as exact answers. Such intelligent query processing requires a knowledge representation framework capturing the knowledge on semantic relationships between raw data values and useful abstract concepts.

Although a variety of knowledge representation frameworks have been studied based on the approaches including semantic distance [14, 15, 17], fuzzy set [1, 21, 27], rule [4, 10, 12], and conceptual classification [2, 3, 7, 9, 23, 26], those studies have disadvantages because they have mostly focused on the query answering process but insufficiently addressed the issue of the semantic knowledge maintenance. To remedy such a problem, we have proposed the Knowledge Abstraction Hierarchy (KAH) [13] that extends the conceptual classification approaches by capturing not only data values but also domain-related abstract information. Also, we have constructed a meta-database, named Knowledge Abstraction Database (KaDB), incorporating the semantics involved in the KAH and discussed the advantages of the KaDB in respect to knowledge maintenance.

Furthermore, the semantics of the KAH is richer than that of other approaches in facilitating more effective query processing as well as more dynamic knowledge maintenance. For the base work on intelligent query processing, we have developed basic query relaxation methods using the KaDB such as value generalization and specialization. In this paper, we will define KaDB operations for intelligent query processing and formally develop query processing mechanisms supporting various types of *intelligent queries* in a consistent and integrated manner. Due to the rich semantics of the KAH, the KaDB increases the diversity of intelligent queries accepted as well as accommodates dynamic knowledge maintenance.

This paper is organized as follows: Section 2 introduces the KAH and KaDB to review our previous work for intelligent query processing. In section 3, we define KaDB operations fundamental to the query processing. Section 4 classifies intelligent queries and explains the query processing mechanisms for each type of queries. Section 5 exemplifies the intelligent query processing. Section 6 provides the comparison with other approaches and the conclusion of the paper.

---

* 한국과학기술원 테크노경영대학원 조교수
** 한국과학기술원 테크노경영대학원 박사과정

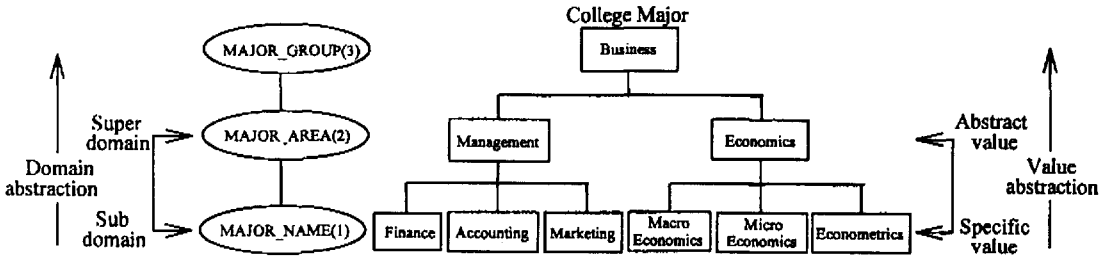

Figure 1. Example of KAH Instances.

2. Knowledge Abstraction Database

2.1 Knowledge Representation Framework KAH

The KAH was developed as a knowledge representation framework that facilitates multilevel representation of data and meta-data for an underlying database by using data abstraction. To illustrate the KAH, Figure 1 shows KAH instances derived from a same underlying database.

The KAH is composed of two types of abstraction hierarchies: *value abstraction hierarchy* and *domain abstraction hierarchy*. First, in the value abstraction hierarchy, a *specific value* is generalized into an *abstract value* and the abstract value can be generalized further into a more abstract value. Conversely, a specific value is considered as a specialized value of the abstract value. Thus, a value abstraction hierarchy is constructed on the basis of generalization/specialization relationships between abstract values and specific values in various *abstraction levels*, which is obtained by using value abstraction. The value abstraction relationship can be interpreted as IS-A relationship. For instance, Finance is a (major name of) Management while Management is a (major area of) Business. As such, higher levels provide a more generalized data representation than lower levels.

While the cardinal relationship between an abstract value and its specific values is assumed to be one-to-many, a specific value can also have multiple abstract values that are located in different abstraction levels along a path from the specific value to its most abstract value at the highest abstraction level. In such capacity, an abstract value is called *n-level abstract value* of the specific value according to the abstraction level difference *n*.

Second, the domain abstraction hierarchy consists of *domains* that encompass all individual values in the value abstraction hierarchy and there exist INSTANCE-OF relationships between the domains and values. Much as generalization/specialization relationships exist between the data values in two different abstraction levels of the value abstraction hierarchy, a *super-domain/sub-domain* relationship exists between two different domains, which is obtained by *domain abstraction*. For instance, MAJOR_AREA is the super-domain of MAJOR_NAME. All the abstract values of instance values in a sub-domain correspond to the instance values of the super-domain. Thus, the super-domain MAJOR_AREA is more generalized than sub-domain MAJOR_NAME, since MAJOR_AREA contains more generalized values than MAJOR_NAME. The cardinal relationship between two adjacent domains is assumed to be one-to-one and a super-domain is called *n*-level super-domain of the sub-domain according to the abstraction level difference *n*.

The abstraction relationships between values and domains can be formally represented as shown in Table 1. In the table, D_i denotes a domain at the abstraction level i and $v_i^{j_i}$ is a specific value of the domain D_i. The relationships (1) and (2) respectively represent 1-level value and domain abstraction relationships. On the basis of the 1-level abstraction relationships, (3) and (4) represent general n-level abstraction relationships.

Multiple KAH instances, as shown in Figure 1, can exist for a single underlying database when multiple perspectives are required for the database. Though a same value can be located in multiple hierarchies and thus, in multiple domains, a domain should be unique and be located only in one hierarchy.

Table 1. Formal Representation of Value and Domain Abstraction Relationships.

Relationship	Formal notation
(1) 1-level domain abstraction	$D_i \Rightarrow D_{i+1}$ where D_i is a sub domain and D_{i+1} is its super-domain.
(2) 1-level value abstraction	$v_i^{j_i} \in^* v_{i+1}^{j_{i+1}}$ where $v_i^{j_i}$ is a specific value and $v_{i+1}^{j_{i+1}}$ is its abstract value.
(3) n-level value abstraction	$v_i^{j_i} \overset{n}{\in^*} v_{i+n}^{j_{i+n}}$ iff $\exists v_{i+1}^{j_{i+1}}, \ldots, \exists v_{i+n-1}^{j_{i+n-1}}$ s.t $v_i^{j_i} \in^* v_{i+1}^{j_{i+1}} \in^* v_{i+2}^{j_{i+2}} \in^* \cdots v_{i+n-1}^{j_{i+n-1}} \in^* v_{i+n}^{j_{i+n}}$
(4) n-level domain abstraction	$D_i \overset{n}{\Rightarrow} D_{i+n}$ s.t $v_i^{j_i} \overset{n}{\in^*} v_{i+n}^{j_{i+n}}, v_i^{j_i} \in D_i, v_{i+1}^{j_{i+1}} \in D_{i+1}, \ldots, v_{i+n}^{j_{i+n}} \in D_{i+n}$ for all j_i

In this sense, a value can exist in multiple hierarchies, its abstract values and specific values are not uniquely determined by the value itself. Domain information is additionally needed to identify the abstract and specific values for a given specific value. We call this property the *domain-dependency* of abstract values and specific values.

2.2 KaDB Accommodating the KAH

To use the semantics of the KAH for intelligent query processing, it was incorporated into a meta-database KaDB [13]. A sample KaDB accommodating the KAH instances in Figure 1 is provided in Figure 2, which comprises three relations: DOMAIN_ABSTRACTION, VALUE_ABSTRACTION, ATTRIBUTE_MAPPING.

DOMAIN_ABSTRACTION

Domain	Super_Domain	Hierarchy	Abstraction _Level
MAJOR_NAME	MAJOR_AREA	College Major	1
MAJOR_AREA	MAJOR_GROUP	College Major	2
MAJOR_GROUP		College Major	3

VALUE_ABSTRACTION

Value	Domain	Abstract_Value
Finance	MAJOR_NAME	Management
Accounting	MAJOR_NAME	Management
Marketing	MAJOR_NAME	Management
Macro Economics	MAJOR_NAME	Economics
Micro Economics	MAJOR_NAME	Economics
Econometrics	MAJOR_NAME	Economics
Management	MAJOR_AREA	Business
Economics	MAJOR_AREA	Business
Business	MAJOR_GROUP	

ATTRIBUTE_MAPPING

Relation	Attribute	Domain
EMPLOYEE_MAJOR	MAJOR	MAJOR_NAME
CAREER_PATH	Task	COURSE_NAME
CAREER_PATH	Prerequisite_Task	COURSE_NAME
TASK_MAJOR	Task	COURSE_NAME
TASK_MAJOR	Required_Major_Area	MAJOR_AREA
TASK_HISTORY	Task_Performed	COURSE_NAME
....	....	....

Figure 2. The KaDB Accommodating the Example KAH Instances.

The DOMAIN_ABSTRACTION maintains the semantics of the domain abstraction hierarchy. Since a domain is unique in the KAH instances and has one-to-one mapping correspondence with its super-domain, 'Domain' attribute becomes the key attribute. The VALUE_ABSTRACTION maintains the semantics of the value abstraction hierarchy. Since a same value name can exist in multiple hierarchies and is differentiated by its domain, both the 'Value' and 'Domain' attributes become the composite key of the relation, which is related with domain-dependency. In terms of abstraction relationships, each tuple of the relations represents only the 1-level abstraction relationship. On the basis of such a 1-level abstraction relationship, an abstract value in any arbitrary level can be transitively retrieved. The ATTRIBUTE_MAPPING maintains the domain information of the underlying database and helps to analysis of the query intent in query processing. Since one attribute name in the underlying database can be used in multiple relation, the 'Relation' and 'Attribute' become the composite key in the ATTRIBUTE_MAPPING relation.

3. KaDB Operations for Intelligent Query Processing

In this section, fundamental KaDB operations to facilitate intelligent query processing are discussed. Query transformation to relax the search condition constitutes a core part of the intelligent query processing, which requires generalization and specialization operations for values and domain by using the KaDB. In the KAH, n-level the value generalization (specialization) operation is equal to moving up (down) the KAH by n-level from a given value and returns an n-level abstract value (a set of n-level specific values). Such an abstract value conversely corresponds to multiple specific values at lower levels in the hierarchy. The operations can be developed on the basis of the functional dependencies among the KAH constructs, i.e., values, abstract values, domains, and super-domains. In the previous section, we defined the cardinal relationships among KAH constructs and explained the domain-dependency in the presence of multiple KAH instances. From the features, we come up with a set of functional dependencies summarized in Table 2.

In the table, functional dependencies are expressed by the notation → and →→ where "A→B" implies that A determines only one B, while "A→→B" implies that A determines multiple Bs.

Table 2. Basic Functional Dependencies.

	Functional dependency	Relation representing the functional dependency
(1) Domain/Super-Domain	$D_i \rightarrow D_{i+1}$ for D_i, D_{i+1} where $D_i \Rightarrow D_{i+1}$	DOMAIN_ABSTRACTION
(2) Domain/Sub-Domain	$D_i \rightarrow D_{i-1}$ for D_{i-1}, D_i where $D_{i-1} \Rightarrow D_i$	DOMAIN_ABSTRACTION
(3) Value/Abstract Value	$(v_i^{j_i}, D_i) \rightarrow v_{i+1}^{j_{i+1}}$ for $v_i^{j_i}, v_{i+1}^{j_{i+1}}$ where $v_i^{j_i} \in^* v_{i+1}^{j_{i+1}}$	VALUE_ABSTRACTION
(4) Value/Specific Value	$(v_i^{j_i}, D_i) \rightarrow\rightarrow v_{i-1}^{j_{i-1}}$ for $v_i^{j_i}$, $v_{i-1}^{j_{i-1}}$ where $v_{i-1}^{j_{i-1}} \in^* v_i^{j_i}$	VALUE_ABSTRACTION
(5) Relation/Attribute	(Relation, Attribute) → Domain	ATTRIBUTE_MAPPING

Table 3. Basic KaDB Operations (1-level Generalization and Specialization).

Operations	Operators	Dependency
1-level domain generalization	*Get_Super_Domain*(D_i) = D_{i+1} where $D_i \Rightarrow D_{i+1}$	(1)
1-level domain specialization	*Get_Sub_Domain*(D_i) = D_{i-1} where $D_{i-1} \Rightarrow D_i$	(2)
1-level value generalization	*Get_Abstract_Value*(v_i, D_i) = v_{i+1} where $v_i \in^* v_{i+1}$	(3)
1-level value specialization	*Get_Specific_Values*(v_i, D_i) = {v_{i-1}, ...} where $v_{i-1} \in^* v_i$	(2), (4)

The dependencies (1) and (2) mean that, if a domain name is known, its super-domain and sub-domain can be identified. On the other hand, the dependencies (3) and (4) mean that, to identify abstract value or specific values of a value, we must know its domain first, which is related with the domain-dependency. Finally, the dependeny (5) helps us to identify the domain of an attribute in the relation specified in a query as a result of the query analysis. On the basis of these functional dependencies, we can define 1-level generalization and specialization operations for values and domains as shown in Table 3.The 1-level domain generalization and specialization operations respectively provide 1-level super-domain and sub-domain for a given domain. Similarly, the 1-level value generalization and specialization operations provide 1-level abstract value and the set of 1-level specific values for a given value. Internally, these operations use DOMAIN_ABSTRACTION and VALUE_ABSTRACTION relations, and we express the operations with the SQL statements as follows:

(1) *Get_Super_Domain*(D_i)=D_{i+1}

```
SELECT    Abstract_Domain
FROM DOMAIN_ABSTRACTION
WHERE     Domain = Di
```

(2) *Get_Sub_Domain*(D_i)=D_{i-1}

```
SELECT    Domain
FROM DOMAIN_ABSTRACTION
WHERE     Super_Domain = Di
```

(3) *Get_Abstract_Value*($v_i^{j_i}$, D_i) = $V_i^{j_i}$

```
SELECT    Abstract_Value
FROM VALUE_ABSTRACTION
WHERE     Value = v_i^{j_i} and Domain=d
```

(4) *Get_Specific_Values*($v_i^{j_i}$, D_i) = { $v_{i-1}^{j_{i-1}}$, ...}

```
SELECT    V.Value
FROM      VALUE_ABSTRACTION V,
          DOMAIN_ABSTRACTION D
WHERE     D.Super_Domain= Di And
          D.Domain=V.Domain And
          V.Abstract_Value = v_i^{j_i}
```

(5) *Get_Attribute_Domain*(*R*,*A*)=*D*

```
SELECT    Domain
FROM      ATTRIBUTE
WHERE     Relation=R AND Attribute=A
```

To control the extent of the query relaxation flexibly, we must increase or decrease the abstraction level of the abstract value by using n-level generalization and specialization operations. The operations are implemented on the basis of the 1-level generalization and specialization operations as shown in Table 4.

Table 4. n-Level Value and Domain Generalization and Specialization Operations.

Operations	Operators and algorithms	Dependency
n-level domain generalization	From given D_i, execute *Get_Super_Domain*( ) n times by replacing the domain with its super-domain, until D_{i+n} is obtained.	(1)
n-level domain specialization	From given D_i, execute *Get_Sub_Domain*( ) n times by replacing the domain with its sub-domain, until D_{i-n} is obtained.	(2)
n-level value generalization	From given v_i and D_i, execute *Get_Abstract_Value*( ) and *Get_Super_Domain*( ) n times by replacing the value and domain with its abstract value and super-domain, until v_{i+n} is obtained.	(1,), (3)
n-level value specialization	From given v_i and D_i, get D_{i-1} and the set of specific values of v_i by executing *Get_Sub_Domain*( ) and *Get_Specific_Values*( ), and repeat executing these two operations for individual specific values n times by replacing the domain and value with its sub-domain and specific value, until the set of the n-level specific values are all obtained.	(2), (4)

4. Intelligent Query Processing Mechanism Using the KaDB

4.1 Classification of Intelligent Queries

The KaDB supports two methods of relaxing the search conditions – *approximately equal search* and *conceptually equal search* – by representing the conditions in the form of "WHERE A =? B", where =? is a relaxation operator and A is an attribute and B is either an attribute (i.e., join condition) or a specific value (i.e., selection condition). In such capacity, A is referred as a *target attribute*, while B is referred as a target attribute or a *target value*.

The approximately equal search is used to relax a search condition in which the terms A and B exist in the same abstraction level, and interpreted to check if the value of A attribute is equal to B or approximately equal. We regard different specific values such as Finance, Accounting, and Marketing in Figure 3 as "approximately equal" because they are located in the same abstraction level and have the same 1-level abstract value, i.e., Management.

In conceptually equal search, the terms A and B in the search condition exist in different abstraction levels, and the condition is interpreted to check if the values of A

and B are conceptually equal. In other words, the abstract value of the value in the lower abstraction level is compared with the value in the higher level.

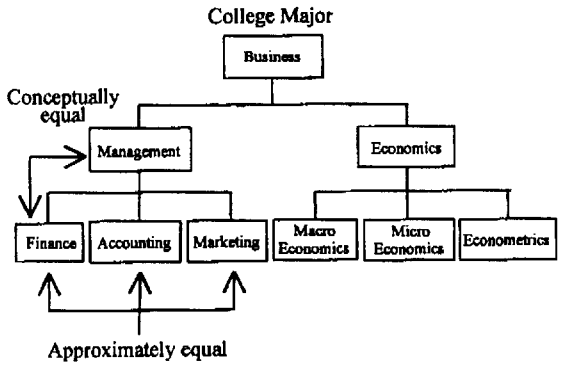

Figure 3. Two Kinds of Equality for Relaxing Search Conditions.

The two query relaxation methods can be extended in the join operation where the target attribute is joined with a range of neighboring values of another join attribute. Accordingly, depending on the kinds of operations and relaxation methods, four types of intelligent queries can be implemented: approximate selection, approximate join, conceptual selection, and conceptual join as seen in Table 5.

Since the detail procedures of the intelligent query processing are differentiated according to the types of queries, we must identify the type of a given query by analyzing the query condition. Specifically, the two operations, *Analyze_Selection* and *Analyze_Join*, are used to conduct the search condition analysis in a given query. They identify the domains of A and B and classifies the condition as one of the four types. These operations are defined as follows:

Notation
R = a source relation specified in a query
A = a target attribute
$v_{i+d}^{j_{i+d}}$ = a target value at the (i + d) abstraction level
D_i = the domain of the target attribute
D_{i+d} = the domain of the target value

Output: (D_i, D_{i+d}, d)
Analyze_Selection(R, A, $v_{i+d}^{j_{i+d}}$)
{
Get_Attribute_Domain(R, A)=D_i
If($v_{i+d}^{j_{i+d}} \in D_i$) Then Return D_i
Else Get D_{i+b} i.e., the domain of the value $v_{i+d}^{j_{i+d}}$
End If
Compute the difference (d) of the abstraction levels between D_i and, D_{i+d}
} // i=1, 2, ... , d = 0, 1, 2, ...

Output: (D_i, D_{i+d}, d)
Analyze_Join(R_k, A_k, R_l, A_l)
{
Get_Attribute_Domain(R_k, A_k)=D_i (i.e., the domain of A)
Get_Attribute_Domain(R_l, A_l)=D_{i+d} (i.e., the domain of B)
Compute the difference (d) of the abstraction levels between D_i and, D_{i+d}
} // i=1, 2, ... , d = 0, 1, 2, ...

4.2 ABSTRACTION relation

We note that the query relaxation methods are all concerned with the target attribute values and their abstract values. In this sense, building abstraction relationships between the target attribute values and their abstract values in a given abstraction level will facilitate the relaxation of a search condition. To this end, **ABSTRACTION** is additionally provided with the following definition.

ABSTRACTION{abstraction(Value, Abstract_Value)}

While ABSTRACTION concerns the target attribute for relaxing the search condition, dealing with ABTRACTION specifically uses the domain of the target attribute – we call the domain *base domain* – since the domain name can identify the values which happen to exist in a KAH instance and are capable of abstraction. Thus, the ABSTRACTION can be interpreted as a relation containing abstract value of every value which is capable of abstraction among the target attribute values.

Figure 4 shows the two instances of ABSTRACTION for the MAJOR_NAME domain which are derived from the KAH in Figure 1. The 1-level ABSTRACTION captures the 1-level abstraction relationship between MAJOR_NAME and MAJOR_AREA while the 2-level ABSTRACTION between MAJOR_NAME and MAJOR_GROUP.

Constructing n-level ABSTRACTION consists of two steps: creation of the 1-level ABSTRACTION and update of the 1-level ABSTRACTION to produce the n-level ABSTRACTION. The first step inserts tuples representing the 1-level abstraction relationship into an empty ABSTRACTION. This is accomplished by *Insert_Abstraction* operation that selects the tuples having the specified base domain (i.e., the domain of the target attribute).

Insert_Abstraction(D_i) = (D_i, { ($v_i^{j_i}$, $V_i^{j_i}$), ...})

```
INSERT INTO   ABSTRACTION (Value,
              Abstract_Value)
SELECT        Value, Abstract_Value
FROM          VALUE_ABSTRACTION
WHERE         Domain = Di
```

Table 5. Classification of Intelligent Queries.

Operations / Relaxation methods	Selection	Join
Approximately Equal Search	Approximate Selection Search	Approximate Join Search
Conceptually Equal Search	Conceptual Selection Search	Conceptual Join Search

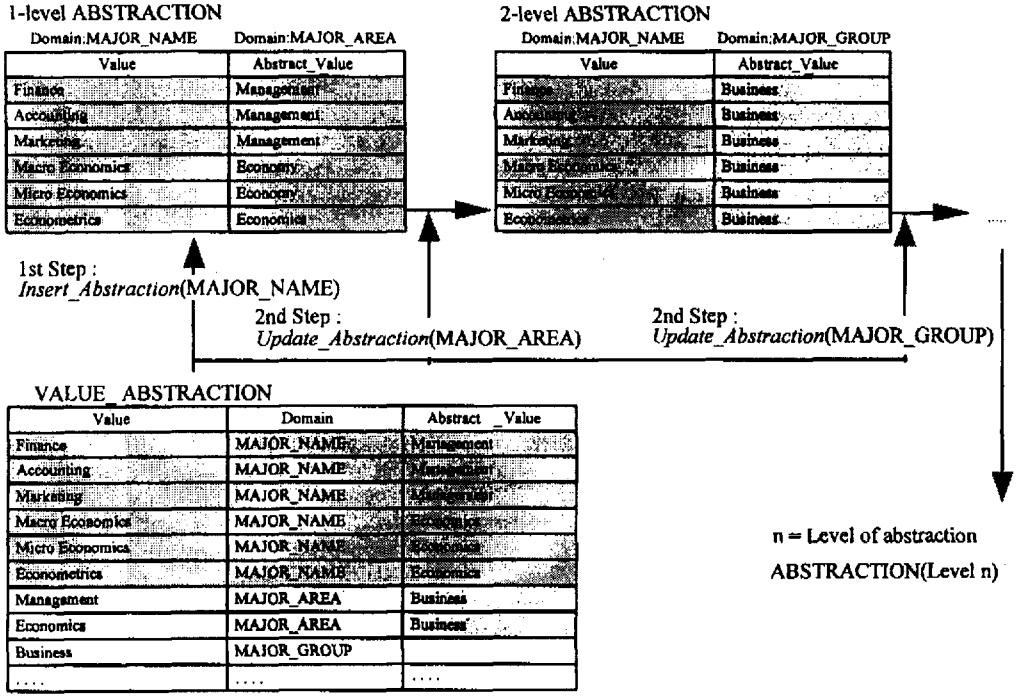

Figure 4. The Procedure of Constructing ABSTRACTION Relation.

The second step updates those tuples in the 1-level ABSTRACTION by executing the n-level generalization operation recursively until the n-level abstraction relationship is produced. For this, *Update_Abstraction* operation is provided. Note that the *i*-level ABSTRACTION in general represents the *i*-level abstract values for every specific value in the base domain.

$$Update\_Abstraction(D_{i+1}) = (\{V_i^{j_1}, V_i^{j_2}, \ldots\})$$

```
UPDATE      ABSTRACTION
SET         Abstract_Value = V.Abstract_Value
FROM        VALUE_ABSTRACTION V
WHERE       V.Domain = D(i+1) AND
            ABSTRACTION.Abstract_Value =
            V.Value
```

On the basis of the two operations, a generic operation, *Construct_Abstraction* is defined for constructing an arbitrary-level ABSTRACTION regardless of the abstraction level difference. Thus, when the abstraction level difference determining the relaxation extent is greater than 1, it produces an ABSTRACTION intance by executing *Update_Abstraction* repeatedly.

```
Construct_Abstraction(Di, d) {
  //Input: the output of Analyze operation, i.e., (Di, d)
  If(d==0){                    //Approximate Equal
  Insert_Abstraction(Di)       //Construct 1-level
                               //ABSTRACTION relation.
  }
  else{                        //Conceptually Equal
  Insert_Abstraction(Di)       //Construct 1-level
                               //ABSTRACTION relation.
  Get_Super_Domain(Di)         //Get super-domain to expand
                               //to 2-level abstraction
  for(k=1; k<d; k++){          //Repeat the Update procedure
        Update_Abstraction(Di+k)
        Get_Super_Domain(Di+k)
  }
  }
}
```

4.3 Relaxing the Search Condition with ABSTRACTION

Joining a target attribute with ABSTRACTION enables us to replace the specific values in the target attribute with their abstract values and thus relaxes a search condition.

Based on the join operation, the ABSTRACTION makes it possible to support the four types of intelligent queries in a consistent and integrated manner through the following steps:

1. Analyze the query condition using the *Analyze_Selection* or *Analyze_Join* operation.
2. Construct an ABSTRACTION instance using *Construct_Abstraction* operation and reduce the scope of the abstraction relationship according to whether the query condition is selection or join condition.
3. Perform query relaxation on the target attribute by joining it with ABSTRACTION.

In the step 1, analyze operations return domain and abstraction level of the target attribute and values in the query. Using the domain and abstraction level, Step 2 constructs an ABSTRACTION instance representing the

abstraction information for the target attribute. Step 3 relaxes the query condition and returns neighborhood answers by simply joining the target attribute in the queried relation with ABSTRACTION. In step 3, depending on the query condition, we need to vary the range of abstraction relationship in ABSTRACTION that initially covers all the abstraction relationship producible from the KAH. Specifically, when the query condition is a join condition, every combination of the join attribute values, their abstract values in more precise terms, are to be compared and thus the every abstraction relationship in ABSTRACTION is used for obtaining the abstract values of join attribute values.

However, when the query condition is a selection condition, the subset of ABSTRACTION should be used for join since only the abstraction relationship containing the abstract value of the target value are useful in joining ABSTRACTION with the target attribute. To reduce the range of abstraction relationship in ABSTRACTION to the neighborhood values of the target value, we use *Reduce_Abstraction* operation defined as follows:

Notation

v_i^{ji} = *the target value in a given query*
d = *the abstraction level difference, which determines the ypes of a query condition*

```
Reduce_Abstraction(v_i^ji , d)
{
if(d==0){        //Approximate Selection
  Get_Abstract_Value(v_i^ji )= V_i^ji //Get abstract value of the
                                      // target value.
  Select tuples from ABSTRACTION where Abstract_Value
  = V_i^ji
  //Select the neighborhood values which share the same abstract
  //value with the target value.
}
else{            //Conceptual Selection
  Select tuples from ABSTRACTION where bstract_Value= v_i^ji
  //Select a set of neighborhood values which are contained by
  //the target abstract value itself.
}
}
```

Even in a selection condition, reducing Abstraction varies according as the search condition is an approximate selection or a conceptual selection, which is determined by the abstraction level difference between the target attribute and target value. The query condition in Figure 8, 'employee_major.major =? Finance', is a typical example of the approximate selection. In such an approximate selection condition, *Reduce_Abstraction* selects the neighborhood values that share the same abstract value (e.g., Management) with the target value, Finance. A conceptual selection condition can be rewritten in the query condition in Figure 8 by replacing Finance with its abstract value, Management, as 'employee_major.major =? Management'. In case of a conceptual selection, *Reduce_Abstraction* selects the neighborhood values directly using the target abstract value itself, Management.

5. Illustration of Intelligent Query Processing

In this section, we exemplify and explain how the four types of intelligent queries are answered on the basis of a simplified personnel database shown below.

EMPLOYEE {(id, emp_name, dept, title)}
TASK_HISTORY{(id, beginning_date, ending_date, task_performed)}
EMPLOYEE_MAJOR{(id, major, entrance_data, graduation_date)}
CAREER_PATH{(task, prerequisite_task)}
TASK_MAJOR{(task, required_major_area)}

EMPLOYEE describes the current job position of an employee, while TASK_HISTORY summarizes the task history by specifying the beginning date and ending date of individual tasks performed by the employee. EMPLOYEE_MAJOR contains the college education records and CAREER_PATH shows the career development path defining prerequisite relationships among tasks. Finally, TASK_MAJOR defines the relationships between individual tasks and the required college major area. A meta-database in Figure 2 is built on these relations.

5.1 Approximately Selection Search

This search provides approximate neighborhood information besides the exact answers that can be obtained by the conventional query processing. In the personnel database example, suppose that a personnel manager wants to find out employees majoring in finance or related majors as shown in the top of Figure 6. In the query, the relaxation operator, =?, is used to specify the relaxation requirement of the search condition. The selection condition, m.major =? "Finance", is to find the employee_major records whose major attribute value is either "Finance" or an approximate neighborhood value. As such, the approximate selection search is to find a set of approximate values nearby a specified target value in the selection condition. In terms of KAH, the approximate selection refers to the selection operation in which the target attribute (i.e., major) and value (i.e., Finance) exist in same abstraction level. The detailed process of the approximate selection search is illustrated in the figure.

Step 1 analyzes the search condition by using *Analyze_Selection* operations. It identifies the domains of 'major' attribute and 'Finance' and classifies the search condition as an approximate selection since the difference between their abstraction level is 0. In step 2, as the search condition is selection condition, ABSTRACTION is reduced by *Reduce_Abstraction* operation after constructing ABSTRACTION instance by using *Construct_Abstraction*. To reduce ABSTRACTION, we select tuples whose attribute value of 'Abstract_Value' are the abstract value of 'Finance', i.e., 'Management'. In step 3, joining target attribute with ABSTRACTION returns three tuples as approximate neighborhood answers whose major attribute value is Finance, Accounting or Marketing. The detailed intelligent query processing is summarized in the following steps:

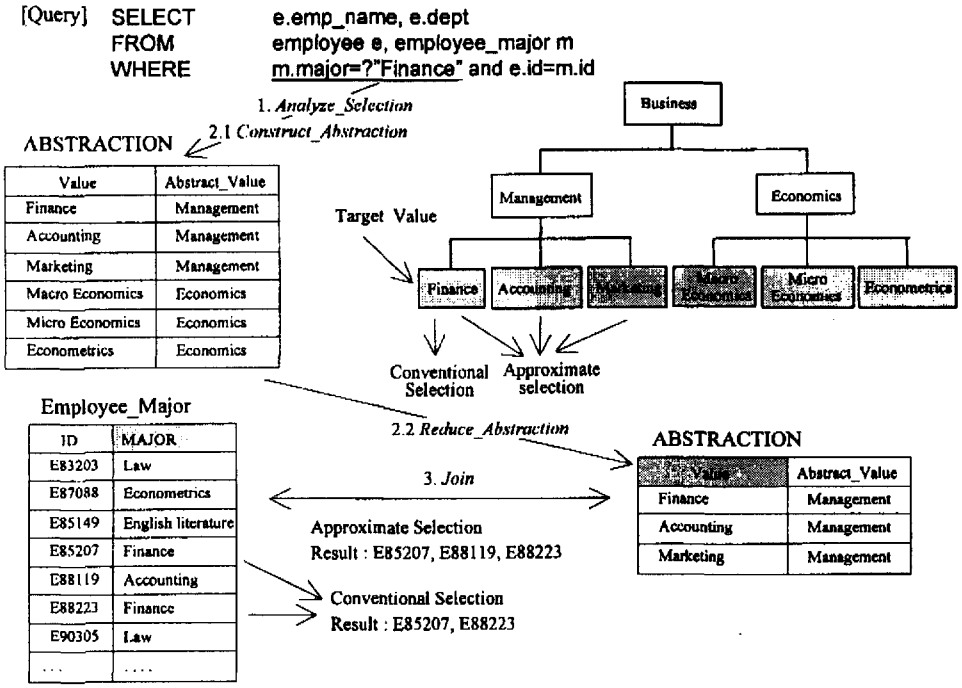

Figure 6. The Process of Approximate Selection.

Search Condition: c.major =? "Finance" and c.id = c.id
Step 1. Analyze a query using the *Analyze_Selection*( ).
Analyze_Selection(employee_major, major, Finance){
  Get_Attribute_Domain(employee_major, major)=MAJOR_NAME
  Return MAJOR_NAME
  Compute the difference of the abstraction levels between MAJOR_NAME and MAJOR_NAME // d=0
}=(MAJOR_NAME, MAJOR_ NAME, 0)

Step 2. Construct and reduce ABSTRACTION relation.
Construct_Abstraction(MAJOR_NAME, 0) = {
  Insert_Abstraction(MAJOR_NAME)
}
Reduce_Abstraction(Finance, 0){
  Get_Abstract_Value(Finance)=Management
  Select tuples from ABSTRACTION where Value = 'Management' or Abstract_Value = 'Management'
}

Step 3. Join employee_major with reduced ABSTRACTION.

5.2 Conceptually Join Search

Conceptually equal search is divided into conceptual selection search and conceptual join search as seen in Table 5. In conventional query processing, these queries are rejected because the domains of the target attributes and target value are not identical.

As an extension of the conceptual selection search, the two attributes in the join condition can have different domains and thus be in different abstraction levels. In the personnel database example, the task_major relation prescribes the required major area for each task. In such capacity, a user may want to find people whose college major belongs to the major area required for performing a certain task, e.g., Cost Accounting tasks. A conceptual join search is composed to answer such a need and can be written as shown in the top of Figure 9. In conventional query processing, the join operation 't.required_major_area =? c.major' returns no answer because the domains of their join attributes are not identical. In terms of KAH, the conceptual join refers to the join operation in which the two target attributes are associated with different domains and thus exist in different abstraction levels. The detailed process of the conceptual join search is illustrated in the figure.

The whole procedure is similar with that of approximate join search except that the level difference is 1. In step 3, employee_major is joined with ABSTRACTION to get the abstract value of the join attribute values, and finally is joined with the other queried relation on the basis of the abstract values. The detailed intelligent query processing is summarized in the following steps:

Search Condition: t.required_major_area =? c.major
Step 1. Analyze a query using the *Analyze_Join*( ).
Analyze_Join(task_major, required_major_area, employee_major, major){
  Get_Attribute_Domain(task_major, required_major_area)= MAJOR_AREA
  Get_Attribute_Domain(employee_major, major)= MAJOR_NAME
  Compute the difference of the abstraction levels between MAJOR_AREA and MAJOR_NAME // *d*=1
}=(MAJOR_AREA, MAJOR_NAME, 1)

Step 2. Construct and reduce ABSTRACTION relation.
Construct_Abstraction(MAJOR_NAME, 1) ={
  Insert_Abstraction(MAJOR_NAME)
}
Step 3. Join employee_major and ABSTRACTION.

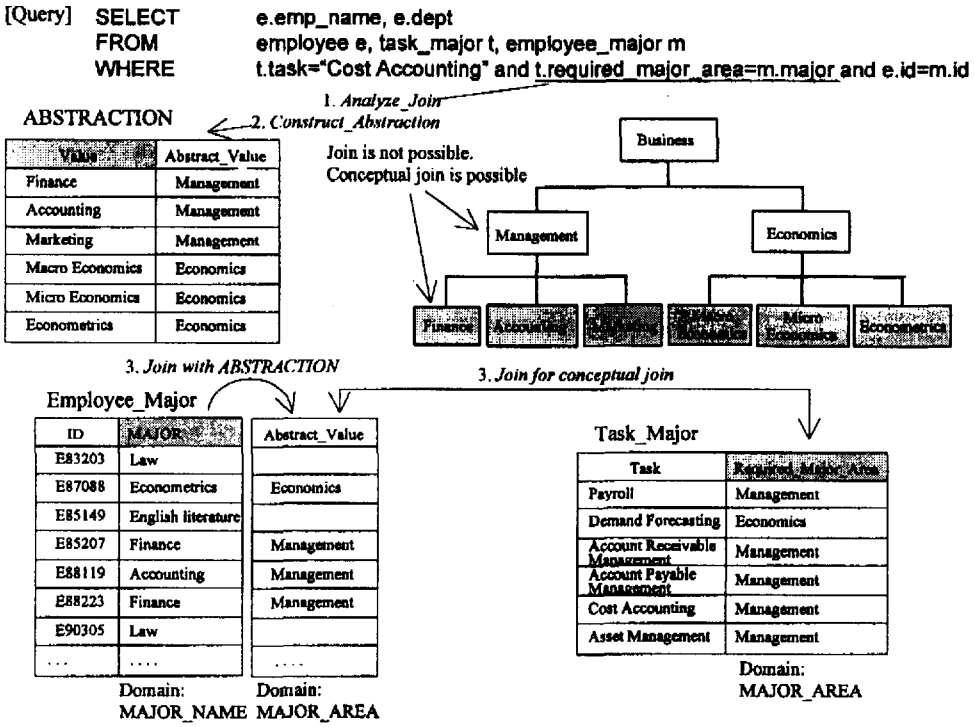

Figure 9. The Process of Conceptual Join.

6. Comparison and Conclusions

The proposed KaDB and intelligent query processing mechanism have an advantage over other approaches, including the conceptual clustering approach [2, 3, 7, 9, 23, 26], the object-oriented database approach, and the rule-based approach [4, 10, 12]. First, existing approaches do not support the relaxation of join condition (i.e., approximately join search and conceptually join search) which could facilitate multiple relations to be approximately or conceptually joined at once, and thus require several intermediate selection queries to perform the same join search. This is because the abstract values of every attribute value related with join operation can not be obtained straightforwardly in existing approaches. However, because of the additional domain information involved in the KAH used as a knowledge representation framework, KaDB can support the relaxation of join condition. The domains information of target attributes makes it possible to obtain the abstraction relationships of the every join attribute values straightforwardly. By virtue of these features, the KaDB can support more diverse types of query processing than existing approaches.

Second, the proposed query processing mechanism is more intuitive enough for users to carry out interactive control than other approaches. The KaDB can guide more interactive and flexible query transformation processes. Through the generalization and specialization processes, the value abstraction hierarchy is used as base query relaxation knowledge, and the domain abstraction hierarchy supplements more dynamic query transformation by using information about domain and abstraction level. Specifically, information about the abstraction level enables the comparison of abstraction levels in multiple domains, supports the query relaxation process both incrementally and directly from a certain preferred level, and provides users with flexible relaxation control.

In this paper, we introduced a meta-database KaDB to support intelligent query processing which relaxes the search condition and provides approximate neighborhood information when exact answers are unavailable. The KaDB was constructed on the basis of a knowledge representation framework, namely KAH, that extracts abstract values and their domains from an underlying corporate database into several hierarchies using value abstraction and domain abstraction. We also presented a query processing mechanism handling a variety of intelligent queries in a consistent and integrated way. Thus, the proposed KaDB and query processing improve the effectiveness of information retrieval and decision making.

The relaxation of the search condition plays a key role in intelligent query processing. To this end, several KaDB operations are defined – the elementary operations such as value generalization and specialization and the other supplementary operations providing necessary information for the query processing procedure. Also, the notions of "approximately equal" and "conceptually equal" are presented to define the equality between data values. Based on the two equality, we classified intelligent queries into four types – approximately selection search, approximate join search, conceptual selection search, conceptual join search. Based on the KaDB operations and the types of intelligent queries, whole query processing

mechanism was developed.

To support all types of queries in a consistent and integrated way by using the KaDB operations, the proposed query answering mechanism uses the ABSTRACTION relation which represents the abstraction information for the given target attribute values. We can relax the search condition and obtain approximate neighborhood answers by joining target attribute in the queried relation with ABSTRACTION. In the whole query processing, the information about domain and abstraction level of the target attribute and values specified in the search condition plays a key role for relaxing the search condition.

Currently, we are working on testing a prototype with a personnel database system to demonstrate the effectiveness and practicality of the KaDB in ordinary database application systems. Looking into the future, we plan to extend the KAH to retrieve and manage data in the federated database systems where component databases have the autonomy of data and store similar data. Such an extended KAH will facilitate the query relaxation processes across the multiple component databases and draw a wider range of approximate answers which would not be obtainable from existing federated database approaches. Fuzzy set theory is also being studied, since both approximate and conceptual conditions have some commonality with fuzzy conditions. Fuzzy systems are believed to enrich the semantics of the KAH and increase the intelligent characteristics of the KaDB.

Reference

1. Buckles, B. P. and F. E. Petry, "A Fuzzy Representation of Data for Relational Databases", *Fuzzy Sets Systems*, Vol. 7, No. 3(1982), 213-226.
2. Cai, Y., N. Cercone and J. Han, "Attribute-Oriented Induction in Relational Databases", *Knowledge Discovery in Databases*, AAAI Press/ The MIT Press, 1993.
3. Chen, Q., W. Chu and R. Lee, "Providing Cooperative Answers via Knowledge-Based Type Abstraction and Refinement," *Proceeding of the 5th International Symposium on Methodologies for Intelligent Systems*, Knoxville, TE, 1990.
4. Cholvy, L. and R. Demolombe, "Queryng a Rule Base", *Proceeding of 1st International. Conference of Expert Database System*, 1986, 365-371.
5. Chu, W., H. Yang, K. Chiang, M. Minock, G. Chow and C. Larson, "CoBase: A Scalable and Extensible Cooperative Information System," *International Journal of Intelligence Information Systems*. Vol. 6, 1996.
6. Chu, W., H. Yang and G. Chow, "A Cooperative Database System (CoBase) for Query Relaxation", *Proceeding of the Third International Conference on Artificial Intelligence Planning Systems*, Edinburgh, 1996.
7. Chu, W. and Q. Chen, "A Structured Approach for Cooperative Query Answering", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 5(1994), 738-749.
8. Chu, W., R. Lee and Q. Chen, "Using Type Inference and Induced Rules to Provide Intensional Answers", in *Proceeding of the 7th International Conference on Data Engineering*, Kobe, Japan, 1991, 396-403.
9. Chu, W., Q. Chen and R. Lee, "Cooperative Query Answering via Type Abstraction Hierarchy" *S.M. Deen, editor, Cooperative Knowledge Base System*, North-Holland, Elservier Science Publishing Co., Inc., 1991, 271-292.
10. Cuppens, F. and R. Demolombe, "Cooperative Answering: A Methodologies to Provide Intelligent Access to Databases" *Proceeding of 2nd International Conference on Expert Database Systems*, 1989, 621-643.
11. Godfrey, P., J. Minker and L. Novik, "An Architecture for a Cooperative Database System" *Proceeding of the 1994 International Conference on Applications of Databases*, 1994.
12. Hemerly, A., M. Casanova and A. Furtado, "Exploiting User Models to Avoid Misconstruals" *Nonstandard Queries and Nonstandard Answers*, Oxford Science Publications, 1994, 73-98.
13. Huh, S.Y. and K.H. Moon, "Knowledge Abstraction Hierarchy for Cooperative Query Answering" Working Paper, Submitted for Publication, Graduate School of Management, KAIST, 1998.
14. Ichikawa, T. "ARES: A Relational Database with The Capability of Performing Flexible Interpretation of Queries" *IEEE Transactions on Software Engineering*, SE-12, 5, 1986.
15. Jain, A. K. and R. C. Dubes, *Algorithms for Cluster Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1988.
16. Minock, M. J. and W. Chu, "Explanation for Cooperative Information Systems", *Proceeding of Ninth International Symposium on Methodologies for Intelligent Systems*. 1996.
17. Motro, A., "VAGUE: A User Interface to Relational Databases that Permits Vague Queries", *ACM Transactions on Office Information Systems*, Vol. 6, No. 3(1988), 187-214.
18. Motro, A., "FLEX: A Tolerent and Cooperative User Interface to Databases", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 2(1990), 231-246.
19. Motro, A., "Intensional Answers to Database Queries", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 3(1994), 444-454.
20. Prade, H. and C. Testemale, "Generalizing Database Relational Algebra for the Treatment of Incomplete or Uncertain Information and Vague Queries", *Information Science*, Vol. 34, No. 2(1984), 115-143.
21. Ram, S., "Intelligent Database Design Using the Unifying Semantic Model", *Information and Management*, Vol.29, No. 4(1995), 191-206.
22. Scheer, A., *Enterprise-Wide Data Modeling: Information Systems in Industry*, Springer-Verlag, 1989.
23. Shum, C. D. and Muntz, "An Information-Theoretic Study on Aggregate Responses" *Proceeding of 14th International Conference on Very Large Databases*, Los Altos, CA: Morgan Kaughmann, 1988, 479-490.
24. Turban E., *Decision Support and Expert Systems: Management Support Systems*, Macmillan, 1988.
25. Ullman, J. D., *Database and Knowledge-Base Systems, Vol 1*, Computer Science Press, 1987.

26. Vrbsky, S. V. and W. S. Liu, "APPROXIMATE-A Query Processor that Produces Monotonically Improving Approximate Answers", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 6(1993).
27. Zemankova, M. and Kandel, A., "Implementing Imprecision in Information Systems", Information Science, Vol. 37, No. 1(1985), 107-141.