Hybrid Relations for Database Schema Evolution

Junichi Takahashi
IBM Research, Tokyo Research Laboratory
5-19, Sanbancho, Chiyoda-ku, Tokyo 102, JAPAN

Abstract

This paper describes hybrid relations in relational databases that allow existing relation to be altered by the addition of new attributes without reorganization of the database schema. The values of new attributes with respect to an existing relation are stored separately from the relation as a set of triples of tuple identifier, attribute name, and value. At query time, a hybrid relation, which has only the attributes requested in a query, is derived virtually by combining the relation and this set of triples. A relation can be reorganized by upgrading its attribute values from these triples. The hybrid relation is defined as an algebraic expression, and equivalent expressions of a query on the hybrid relations are shown for efficient query processing.

1 Introduction

Schema evolution is one of the most important aspects of the use of a database, because once the database has been constructed, users' requirements tend to change. The reason for this is that different people have their own perspectives on a particular object in a database, and even the same person may have different perspectives at different times. Not all the properties of each object are necessarily described in the database schema definition phase. At the same time, a single property of an object can be described with various attribute names.

A typical example is an image database where alphanumeric attribute data associated with each image are managed in a relational database [7, 11]. The contents of image data are intuitively expressed by a limited number of attributes, and not can be described completely by a given set of attributes; thus the perspectives of all users are never satisfied. Annotations to images should be easily accommodated within the existing relations [12]. Schema evolution has also been discussed in the context of object-oriented databases [1], and it has been pointed out that the most frequent change of schema is the addition of new properties to an existing class [6].

In conventional relational database systems such as SQL/DS [5], new attributes can be added by reorganizing existing relations. If the schema changes frequently, the following problems will occur in this reorganization:

1. *Suspension of database use due to reorganization:* Redesigning the schema and reorganizing the database every time users' requirements change places a heavy burden on the database administrator(DBA). Moreover, the users cannot start accessing the new relations until the reorganization is completed.
2. *Inefficient storage:* When users' requests to include new attributes into one relation are accepted, the relation tends to become sparse. In other words, almost all records have null values for some attributes in the relation.

Relational databases have been extended so that text can be stored in long fields [8, 14]. Schek and Pistor enabled typical information retrieval queries to be processed in relational databases by providing special predicates for textual attributes such as matching the sequence of keywords in text [13]. These approaches are flexible for storing arbitrary information as text, but unsuitable for schema evolution because there is no system support for schema refinement.

Motro proposes loosely structured databases that consist of a heap of binary facts. Although two special browsing facilities, navigation and probing, are provided, an interface for updating binary facts remains to be investigated [9]. Borgida and Williamson discuss the problems of making exceptional facts conform to a database schema when they are actually correct representations of the state of the real world but are not valid within the schema constraints [3]. Such facts are stored in a separate logical file of records, each of which holds a single fact. But the focus of their work is on schema refinement tools for DBAs, and details of data manipulation for exceptional facts are not mentioned.

Our basic idea for allowing flexible schema evolution in relational databases is to store the values of new attributes separately from the existing relation without reorganizing the database schema, and altering the relation virtually by adding the new attributes at query time. Our overall approach is described in section 2. Section 3 shows a relational interface for manipulating the virtual relation, called a hybrid relation. In section 4, we give algebraic definition of the hybrid relation, and show equivalence expressions of a query on the hybrid relations for efficient query processing. Section 5 discusses some implementation considerations.

2 Hybrid Relations

Figure 1 shows our overall approach. The ***base relation*** is a relation that represents an entity or a relationship in the real world with all its attributes. The base relation has a key attribute, *ID*, as tuple identifier. When the base relation has a composite key, the *ID* is made into a single attribute by concatenating a set of composite key values or by assigning system-unique values(surrogates) to them. The

0730-3157/90/0000/0465$01.00 © 1990 IEEE

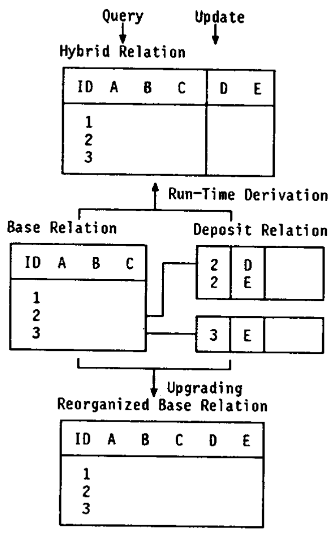

Figure 1. Overall approach to flexible schema evolution

values of new attributes not in the base relation are stored in a separate file called a ***deposit relation***. A relational interface is provided for retrieval and update of the deposit relation. A query result is shown as a ***hybrid relation***, which is derived virtually by combining the base relation and the deposit relation. ***Upgrading*** is an operation to reorganize a base relation by merging the base relation with the deposit relation.

2.1 Deposit Relations

Associated with each base relation is a ***deposit relation***, which is a collection of triples (*ID*,*ATTR*,*VAL*). Logically, a deposit relation takes the form of a relation with three special attributes. The first attribute, *ID*, identifies a tuple in the base relation. The second and third attributes are the name and value, respectively, of the attribute. A pair of *ID* and *ATTR* identifies tuples in the deposit relation. A tuple in a deposit relation represents one attribute of an entity or a relationship in the base relation.

2.2 Derivation of Hybrid Relations

At query time, the tuples in the base relation are viewed together with the corresponding set of tuples in the deposit relation as one virtual relation, named a ***hybrid relation***. Retrieval and update are performed by means of this hybrid relation. Derivation of a hybrid relation is a process for getting virtual tuples, each of which consists of all the listed properties of an object in a database. This process in shown in Figure 2. First, a deposit relation is ***transformed*** so that it can be joined to the base relation. Then, the hybrid relation is derived by joining the base relation and the transformed deposit relation according to the values of *ID* The deposit relation and the transformed deposit relation have the same meanings in the real world, but differ in their representation. Note that an outer join [4] should be performed so as not exclude those tuples in the base

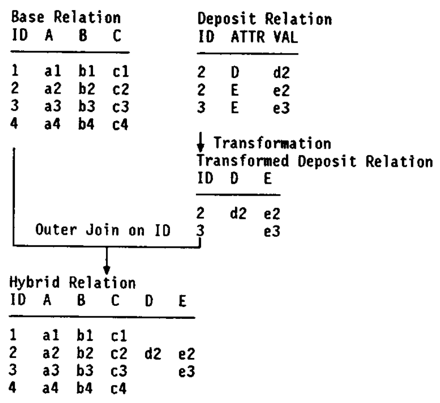

Figure 2. Derivation of the hybrid relation

relation that have no parts counter in the deposit relation.

The attributes of a hybrid relation are determined dynamically according to the values of *ATTR* in the deposit relation. The hybrid relations differ from conventional views, in all of which attributes are defined in the design phase. Note that adding new attribute names in the deposit relation does not change an existing database schema.

2.3 Upgrading

When particular attribute names in the deposit relation are referred to frequently by many users, the DBA would be expected to reorganize the base relation in order to reduce the cost of derivation of the hybrid relation. ***Upgrading*** of the deposit relation is an operation whose purpose is to reorganize the base relation by merging it with the deposit relation. This process is shown in Figure 3. First, the base relation is altered by adding a set of attribute names in the deposit relation (duplicate names are eliminated). Next, each tuple in the deposit relation is copied into the values of the corresponding attribute in the new base relation. Finally, all copied tuples are erased from the deposit relation. Note that the deposit relation becomes empty after upgrading.

3 Manipulating Hybrid Relations

3.1 Querying Hybrid Relations

Because a hybrid relation is regarded externally as a (virtual) relation in a relational database, relational operators can also be defined on it. When an SQL interface is provided on the hybrid relations, users can refer to hybrid relations in the same way as conventional relational views. The syntax of a query is:

```
SELECT <Target_List>
FROM   <Relation_Name_List>
WHERE  <Search_Conditions>
```

<Relation_Name_List> may contains the names of base relations and hybrid relations, and a join among base relations and hybrid relations can be specified. <Target_List> and <Search_Conditions> may

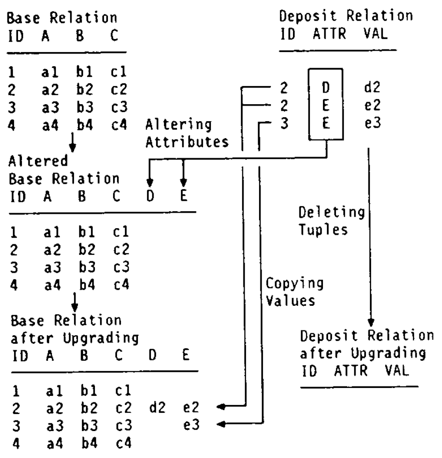

Figure 3. Upgrading the deposit relation

contain the names of attributes in the deposit relation as well as in the base relation.

3.2 Updating Hybrid Relations

Insert: A tuple to be inserted into a hybrid relation has a new *ID* with respect to the base relation. This insertion is mapped to a insertion into the base relation and to multiple insertions into the deposit relation. If the attribute name is not defined in the base relation, then the pair of the attribute name and the value is stored in the deposit relation with the *ID* that has been inserted into the base relation. For example, insertion of a tuple $< ID = 5, A = a_5, E = e_5 >$ into the hybrid relation in Figure 2 causes insertion of a tuple $< ID = 5, A = a_5 >$ into the base relation and insertion of a tuple $< ID = 5, ATTR = E, VALUE = e_5 >$ into the deposit relation.

Delete: A tuple to be deleted from a hybrid relation has a *ID* that already exists in the base relation. This deletion is mapped to a deletion from the base relation and multiple deletions from the deposit relation. Note that not all attributes referred to in the search condition are defined in the base relation. For example, in Figure 2, deletion from the hybrid relation of all tuples that satisfy $E = e_3$ causes deletion of tuples from the base relation whose *ID* s match those of *ID*s in the deposit relation where $ATTR = E$ and $VAL = e_3$. At the same time, this deletion causes deletion from the deposit relation of all tuples that have the same *ID*s as those to be deleted from the base relation.

Update: When all the attributes of a hybrid relation to be updated are defined in the base relation, update of the hybrid relation is directly mapped to that of the base relation. When a user is willing to specify that the value of an attribute that is not in the base relation should be updated, the semantics of this update vary according to the pair of attribute name and new value. This case is subdivided into the following three instances:

- *Insertion into deposit relation:* If the deposit relation contains no tuple whose *ID* and *ATTR* match those of the updated tuple, then the pair of the attribute name and the value is stored in the deposit relation with the *ID*. For example, for the tuple in the hybrid relation where $ID = 3$, in Figure 2, setting the value of *D* to d_3 causes the insertion of the tuple $< ID = 3, ATTR = D, VAL = d_3 >$ into the deposit relation.
- *Deletion from deposit relation:* If the update value is null and there is a tuple in the deposit relation whose *ID* and *ATTR* match those of the updated tuple in the hybrid relation, then the tuple is deleted from the deposit relation. For example, for the tuple in the hybrid relation where $ID = 2$, in Figure 2, setting the value of *E* to null causes the deletion of the tuple $< ID = 2, ATTR = E, VAL = e_2 >$ from the deposit relation.
- *Update of deposit relation:* In other cases, when the attribute of the updated tuple is not in the base relation, an update is interpreted as an update of *VAL* attribute of the deposit relation.

4 Algebraic Operations for Hybrid Relations

In this section, we first define the hybrid relation using algebraic operators by introducing new operators, transformation and outerjoin with transform, with the conventional algebraic operators; union($\bigcup$), difference($-$), selection(σ), projection(π), Cartesian product($\times$), and natural join($\bowtie$). We then describe some properties of these new operators, which give equivalent expressions of the query on the hybrid relation for alternative query execution.

4.1 Definition of Transformed Deposit Relations

Let $atr(R)$ denotes a set of attributes of relation *R*, and $dom(A)$ denotes a domain of attribute *A*.

__*Definition 1*__ *(Base relation and deposit relation)*: Let *B* be a relation on schema S_B over attributes $\{ID, C_1, \ldots, C_l\}$, and *D* be a relation on schema S_D over attributes $\{ID, ATTR, VAL\}$, where *ID* is a key in *B* and a foreign key in *D*, $dom(ATTR)$ is a finite set of attribute names, $S_B, S_D \subseteq dom(ATTR)$, $atr(B) \cap atr(D) = \{ID\}$, and $\pi_{ID} D \subseteq \pi_{ID} B$. *B* is called a ***base relation***, and *D* is called a ***deposit relation*** of *B*.

The constraint: $\pi_{ID}D \subseteq \pi_{ID}B$, in definition 1, means that each value of attribute *ID* in a deposit relation must also appears as a tuple identifier in a base relation. Let us consider horizontal partitions $d_{i,j}$ of deposit relation *D* for $s_i \in adom(ID,D)$, $i \in I = [1..n]$, and $A_j \in adom(ATTR,D)$, $j \in J = [1..m]$:

$$d_{i,j} = \sigma_{ID = s_i \wedge ATTR = A_j} D$$

where a set $adom(X,R)$ denotes *active domain* of attribute *X* of relation *R*; a set of all values that actually occurs in *X* of *R* at a given database state. If there is no such a tuple in *D* that satisfies the condition $ID = s_i \wedge ATTR = A_j$, then $d_{i,j} = \phi$. Otherwise, each $d_{i,j}$ is a singlton relation. *D* is given as union of $d_{i,j}$ for all *i* and *j*:

$$D = \bigcup_{i \in I} \bigcup_{j \in J} d_{i,j} = \bigcup_{j \in J} \bigcup_{i \in I} d_{i,j}$$

For each $d_{i,j}$, consider a relation $D_{i,j}$:

$$D_{i,j} = \begin{cases} \{<s_i, \perp>\} & \text{if } d_{i,j} = \phi \\ \delta_{VAL \leftarrow A_j} \pi_{ID,VAL}\, d_{i,j} & \text{otherwise} \end{cases}$$

The symbol $\perp$ denotes null value, and the operator $\delta_{A \leftarrow B}$, defined as below, renames attribute name A of relation R to B.

$$\delta_{A \leftarrow B} R = \{s \mid t \in R\,; \quad s[atr(R) - A] = t[atr(R) - A] \wedge s(B) = t(A)\}$$

Each $D_{i,j}$ is a binary relation with $atr(D_{i,j}) = \{ID, A_j\}$, and has one tuple whose identifier is s_i.

Transformation of a relation is an unary operator that can be applied to the relation which has attributes ID, $ATTR$, and VAL. This operator maps one or more tuples in the relation all of which have a same value of ID, to a single tuple. All distinct values of attribute $ATTR$ in the relation appear as attribute names in the resultant relation.

Definition 2 *(Transformation)*: Relation T is a transformed relation of relation R with $\{ID, ATTR, VAL\} \subseteq atr(R)$, denoted as $T = \tau R$, if following conditions hold:

a: $atr(T) = atr(R) - ATTR - VAL \cup \pi_{ATTR} R$
   $atr(R) \cap \pi_{ATTR} R = \phi$

b: $\pi_{ID, A_j} \sigma_{ID = s_i} T =$

$$\begin{cases} \{\perp\} & \text{if } \sigma_{ID = s_i \wedge ATTR = A_j} R = \phi \\ \delta_{VAL \leftarrow A_j} (\pi_{ID,VAL} (\sigma_{ID = s_i \wedge ATTR = A_j} R)) & \text{otherwise} \end{cases}$$

$s_i \in adom(ID,D)$, $i \in I = [1..n]$,
$A_j \in adom(ATTR,D)$, $j \in J = [1..m]$

c: $(\forall r \in R)(\exists t \in T)$;
   $r[ID] = t[ID] \rightarrow r[atr(R) - ATTR - VAL] = t$
   $(\forall r_1, \forall r_2 \in R)$;
   $r_1[ID] = r_2[ID] \rightarrow r_1[atr(R) - ATTR - VAL] = r_2[atr(R) - ATTR - VAL]$

The last condition in the above definition guarantees that the transformation operator can also be applied to the relation which has some attributes in addition to ID, $ATTR$, and VAL, if the relation satisfies a functional dependency: $ID \rightarrow atr(R) - ATTR - VAL$.

Example: Relation $T(ID, A_1, A_2, C) = \{<1, a_1, c_1>, <1, a_2, c_1>\}$ is a transformed relation of $R(ID, ATTR, VAL, C) = \{<1, A_1, a_1, c_1>, <1, A_2, a_2, c_1>\}$.

Definition 3 *(Transformed deposit relation)*: Let D be a deposit relation. Relation $T = \tau D$ is called ***transformed deposit relation*** of D.

Let $D_{i,\bullet}$ be a J-ary singlton relation for each identifier s_i, whose tuple consists of values of all attributes A_j, and let $D_{\bullet,j}$ be a binary relation for each A_j, which consists of tuples for all s_i.

$$D_{i,\bullet} = \underset{j \in J}{\bowtie} D_{i,j}\,, \quad D_{\bullet,j} = \bigcup_{i \in I} D_{i,j}$$

Note that $atr(D_{i,\bullet}) = \{ID, A_1, \ldots, A_J\}$, and $atr(D_{\bullet,j}) = \{ID, A_j\}$. Since

$$\pi_{ID} D_{j_1,\bullet} = \pi_{ID} D_{j_2,\bullet} = \{s_i\}$$

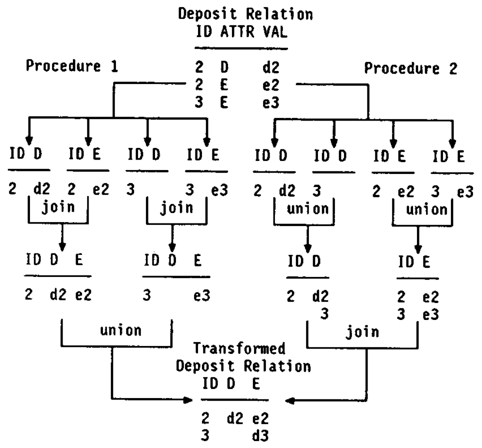

Figure 4. Transformation procedures for the deposit relation

for $i \in I$, $j_1, j_2 \in J$, we get transformed deposit relation T of D:

$$T = \bigcup_{i \in I} D_{i,\bullet} = \underset{j \in J}{\bowtie} D_{\bullet,j} \qquad (1)$$

Equation (1) shows two procedures to transform the deposit relation, which are shown in Figure 4. The cost of these two transformation procedures is further discussed in Section 5.

4.2 Definition of Hybrid Relations

According to the constraint on the deposit relation: $\pi_{ID} D \subseteq \pi_{ID} B$, in definition 1, each value of attribute ID in a deposit relation must also appears as a tuple identifier in a base relation, while not all identifiers in the base relation appear in the deposit relation. The hybrid relation preserves all tuples in the base relation that have no parts counter in the deposit relation, by performing outerjoin between the base relation and deposit relation.

Outerjoin of relation R and S preserving information in R, denoted as $R \overset{\leftarrow}{\bowtie} S$, is given [10] by:

$$R \overset{\leftarrow}{\bowtie} S = R \bowtie (S \cup (\pi_K R - \pi_K S) \times \perp^L)$$

where $K = atr(R) \cap atr(S)$, $L = \text{Degree}(S) - |K|$, and $\perp^1 = \{null\}$, $\perp^L = \perp^{L-1} \times \perp^1$

Definition 4 *(Outerjoin with transform)*: ***Outerjoin with transform*** is an operator, denoted as $\blacktriangleright$, that applies transformation τ, followed by applying outerjoin $\overset{\leftarrow}{\bowtie}$

Definition 5 *(Hybrid relation)*: Let B be a base relation and D be a deposit relation, and $H = B \blacktriangleright D$. H is called ***hybrid relation*** of B and D.

Hybrid relation H of B and D is expressed using algebraic operators as following equation:

$$H = B \overset{\leftarrow}{\bowtie} (\tau D) = B \overset{\leftarrow}{\bowtie} (\bigcup_{i \in I} D_{i,\bullet}) = B \overset{\leftarrow}{\bowtie} (\underset{j \in J}{\bowtie} D_{\bullet,j})$$

4.3 Relational Operators on Hybrid Relations

Selection: Let H be a hybrid relation of B and D, α be a constant, and $A_0 \in \pi_{ATTR}D$.

$$\sigma_{A_0=\alpha}H = B \bowtie \tau(D \bowtie (\sigma_{ATTR=A_0 \wedge VAL=\alpha}D)) \quad (2)$$

(Proof)
Let $j_0 \in J$, $A_{j_0} = A_0$, and $D' = D \bowtie \pi_{ID}\sigma_{ATTR=A_0 \wedge VAL=\alpha}D$.

$$d'_{i,j} = d_{i,j} \bowtie \pi_{ID}\sigma_{ATTR=A_0 \wedge VAL=\alpha}d_{i,j_0}$$

$$D'_{i,j} = D_{i,j} \bowtie (\pi_{ID}\delta_{VAL \leftarrow A_0}\pi_{ID,A_0} \sigma_{ATTR=A_0 \wedge VAL=\alpha}d_{i,j_0})$$

$$= D_{i,j} \bowtie (\pi_{ID}\delta_{VAL \leftarrow A_0}\pi_{ID,A_0} \sigma_{ATTR=A_0 \wedge VAL=\alpha}\sigma_{ID=s_i \wedge ATTR=A_j}d_{i,j})$$

$$= D_{i,j} \bowtie \pi_{ID}\sigma_{A_0=\alpha}D_{i,j} = \pi_{ID}\sigma_{A_0=\alpha}D_{i,j}$$

$$\therefore \sigma_{A_0=\alpha}\tau D = \tau(D \bowtie (\sigma_{ATTR=A_0 \wedge VAL=\alpha}D)) \; \blacksquare$$

Projection: Let $A \subseteq \pi_{ATTR}D$.

$$\pi_{ID,A}H = (\pi_{ID}B) \overset{\leftarrow}{\bowtie} \tau(\sigma_{ATTR \in A}D) \quad (3)$$

(Proof)
Let D' be $\sigma_{ATTR}D$. If $A_j \in A$, then $D'_{i,j} = D_{i,j}$ because $d'_{i,j} = d_{i,j}$. Otherwise, $d'_{i,j} = \phi$, and $D'_{i,j} = \{< s_i, \perp >\}$. Then,

$$\pi_{ID,A}\tau D = \bigcup_{i \in I} \pi_{ID,A}((\bowtie_{j \in J, A_j \in A} D_{i,j}) \bowtie (\bowtie_{j \in J, A_j \notin A} D_{i,j}))$$

$$= \bigcup_{i \in I} \pi_{ID,A}(\bowtie_{j \in J, A_j \in A} D_{i,j})$$

$$= \bigcup_{i \in I} \bowtie_{j \in J} D'_{i,j} = \tau D'.$$

$$\therefore \pi_{ID,A}(B \overset{\leftarrow}{\bowtie} \tau D) = (\pi_{ID}B) \overset{\leftarrow}{\bowtie} (\pi_A \tau D)$$

$$= (\pi_{ID}B) \overset{\leftarrow}{\bowtie} \tau(\sigma_{ATTR \in A}D) \; \blacksquare$$

4.4 Query Processing Tactics

The result of a query on the hybrid relation is calculated by applying the selection and projection operators specified in the query, after joining the base relation and the transformed deposit relation. This straightforward processing, shown in Figure 2, is not efficient, because the transformed deposit relation need to be computed for each query by accessing all tuples in the deposit relation. For example, if condition $ID = 2$ is specified in a query, it is better to transform the deposit relation after restricting it with this condition.

According to the properties (2) and (3) of the transformation operator, the cost of transforming a deposit relation can be reduced by applying the following restricting rules to the deposit relation before the transformation.

1. *Restriction by attribute names:* The transformed deposit relation need not have other attributes than those referred to in a query, because these attributes will be removed in the query result. Thus, tuples in the deposit relation can be restricted to those in which attribute names are specified in the target list or in the search conditions of the query. For example, the transformed deposit relation should have only the attributes *ID*, *A*, and *E* for a query with a target list *ID*, *A* and a search condition $E = e_2$.
2. *Restriction by values:* The transformed deposit relation need not have other tuples than those that satisfy the search conditions in a query, because these tuples will be removed in the query result. If the attribute name referred to in a condition term is not defined in the base relation, the term is translated to two terms on *ATTR* and *VAL* to restrict the number of tuples in the in the deposit relation. Note that this condition on the hybrid relation can also be removed from the original query, once the deposit relation has been transformed. For example, the term $E = e_1$ in the search conditions on the hybrid relation is converted to the terms $ATTR = E$ and $VAL = e_1$ to restrict the number of tuples in the deposit relation. Then, the term $E = e_1$ is removed from the search conditions in the original query.

5 Implementation Consideration

5.1 Data Type Support for the Deposit Relation

When the *VAL* attribute of a deposit relation has a specific data type such as integer or character, it is usually necessary to prepare as many deposit relations as the number of data types in order to store the values of different data types. To avoid this complicated situation, we designed the data format of the deposit relation, shown in Figure 5, in such a way that dose not change the deposit relation structure when storing values of different data types. The attribute *VAL* has three parts: *type* specifies the number corresponding to each data type such as integer or character, *length* specifies the byte length of the data, and *data* specifies the actual data.

ID	ATTR	VAL		
		TYPE 2 Bytes	LENGTH 2 Bytes	DATA Variable Length

Figure 5. Data format for the deposit relation

5.2 Outerjoin with Transform Operator

$D_{i,\bullet}$ and $D_{\bullet,j}$ are horizontal and vertical partitions of transformed deposit relation of D, respectively. The transformation operator τ is efficiently implemented by computing each $D_{\bullet,j}$ for all i, rather than computing each $D_{i,\bullet}$ for all j. This is because each $D_{\bullet,j}$ is a singlton relation, and it can be processed within the main memory. On the contrary, it is unlikely that buffer space is larger enough to accommodate each $D_{i,\bullet}$.

Following shows two different implementations of the outerjoin with transform operator based on two join algorithms; 1) nested loop, and 2) merge join [2] , respectively.

```
Nested Scan Algorithm
  A := set of n distinct attribute names of D
  For each block B[i] in base relation B
    For each record b[i,j] in B[i]
      t[A]:=null  (* n-ary null tuple *)
      For each block D[k] in deposit relation D
        For each record d[k,l] in D[k]
          If d[k,l].Id=b[i,j].Id
            Then t[d[k,l].ATTR] :=d[k,l].VAL
        End
      End
      Write(b[i,j] || t[A])
    End
  End

Merge-Based Algorithm
  A := set of n distinct attribute names of D
  Sort base relation B and deposit relation D by ID
  nb :=1  (* block number for D; nb ≤ max_nb *)
  nr :=1  (* record number in one block; nr ≤ max_nr *)
  For each block B[i] in B
    For each record b[i,j] in B[i]
      t[A]:=null  (* n-ary null tuple *)
      Do while(d[nb,nr].ID=b[i,j].ID & nb ≤ max_nb)
        If nr > max_nr Then Do nb :=nb+1; nr :=1; End
        Else If d[nb,nr].Id=b.Id[i,j]
          Then t[d[nb,nr].ATTR] :=d[nb,nr].VAL
      End
      Write(b[i,j] || t[A])
    End
  End
```

In each implementation, block access cost for outerjoin with transform is same as the cost for join operation. If the base relation B and the deposit relation D comprise N_B and N_D physical blocks respectively, then

$$Cost_{\text{Nested-Scan}} = N_B + N_B N_D$$
$$Cost_{\text{Merge}} = N_B + N_D + \text{(sorting cost)}$$

6 Concluding Remarks

In this paper, the hybrid relation has been proposed as a means of altering an existing relation by adding new attributes without reorganizing the database schema. An application system for the hybrid relations is under development on an IBM S/370 with an IBM Personal System/55 and an IBM 5080 graphic display for browsing images of museum artifacts. Bibliographical data on the images, such as image identifier and name of artifacts, are given in the base relation. Users are allowed to annotate images by updating the hybrid relation while browsing the images on the graphic display, and these annotations can be referred to in later queries.

By using the relational interface on the hybrid relations, users will not be subject to suspension of database use during the database schema evolution. The hybrid relation described in this paper can be applied to areas, such as image databases, where views of data differ from one user to another, and where the database schema is continuously growing.

Planed enhancements include transformation rules based on physical aspects, such as using indexes, for more efficient query processing. Query optimizer still remains to be investigated. Another plan is for a semantic interface layer for intelligent retrieval. Users may want to access the hybrid relations by using more familiar attribute names and values than those actually stored in the database. This layer will allow users to refer to hybrid relations without specifying exact attribute names and values in a query.

Acknowledgement

The author is very grateful to Dr. H. Takahashi and Dr. J-K. Hong, who are our collaborators, for their many suggestions and their encouragement in this work.

References

[1] Banerjee, J. and Kim, W., "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," *Proc. of ACM SIGMOD*, 1987, pp. 311-322.

[2] Blasgen,M. and Eswarn,K., "Storage and Access in Relational Databases," *IBM System Journal*, Vol. 4, No. 4, 1977, pp. 363-377.

[3] Borgida, A. and Williamson, K., "Accommodating Exceptions in Databases, and Refining the Schema by Learning from Them," *Proc. of VLDB*, 1985, pp. 72-81.

[4] Date, C.J., "The Outer Join," *Proc. of the 2nd Int. Conf. on Databases*, 1983.

[5] IBM Corp., "SQL/DS Concept and Facilities," 1983.

[6] Joseph, J. et al., "Report on the Object-oriented Database Workshop," *ACM SIGMOD Record*, Vol. 18, No. 3, 1989, pp. 143-158.

[7] Kunii, T. et al., "A Relational Data Base Schema for Describing Complex Pictures with Color and Texture," *Proc. IEEE 2nd Int. Joint Conf. on Pattern Recognition*, 1974, pp. 310-316.

[8] Lehman, T., "The Starburst Long Field Manager," IBM Corp., RJ6899, 1989.

[9] Motro, A., "Browsing in a Loosely Structured Database," *Proc. of ACM SIGMOD*, 1984, pp. 197-207.

[10] Ozsoyoglu,G., Matos,V., and Ozsoyoglu,M., "Query Processing Technique in the Summary-Table-Example Database Query Language," *ACM Transaction on Database Systems*, Vol. 14, No. 4, 1989, pp. 526-574.

[11] Roussopoulos, N. and Leifker, D., "An Introduction to PSQL: A Pictorial Structured Query Language," *Proc. IEEE Workshop on Visual Language*, 1984, pp. 88-87.

[12] Sato, M. et al., "Color Image Retrieval System for Ethnological Studies," *Transactions of the Information Processing Society of Japan*, Vol. 29, No. 12, 1989, pp. 1108-1118.

[13] Schek, H.J. and Pistor, P., "Data Structures for an Integrated Data Base Management and Information Retrieval System," *Proc. of VLDB*, 1982, pp. 197-207.

[14] Stonebraker, M. et al., "Document Processing in a Relational Database System," *ACM Transactions on Office Information Systems*, Vol. 1, No. 2, 1984, pp. 143-158.