QUERY PROCESSING UTILIZING DEPENDENCIES AND HORIZONTAL DECOMPOSITION

Yahiko Kambayashi
Masatoshi Yoshikawa

Department of Information Science
Faculty of Engineering
Kyoto University
Sakyo, Kyoto, 606 JAPAN

Abstract

Since join operations are expensive, usually join scheduling is very important for query processing. In this paper we will discuss new procedures to handle cyclic queries utilizing dependencies and horizontal decompositions. There are three known procedures for cyclic query processing: (1) Relation merging, (2) Tuple-wise processing, (3) Attribute addition. As join operations are applied to relations which are processed by selection operations, the number of tuples is usually less than the original relation and thus there are situations in which temporary FDs are satisfied. Such FDs can be used to simplify the given query. To convert a given cyclic query into a tree, some relations must satisfy a set of FDs. This can be attained by horizontal decomposition. Tuple-wise processing and attribute addition are shown to be special cases of the FD-based procedure. We have also developed MVD-based procedures which are generalized from the FD-based procedure.

This research was supported in part by the grant from the Ministry of Education, Japan.

1. Introduction

One of the advantages of relational database systems is the utilization of flexible high-level query languages. Query optimization is essential to support such languages. This paper discusses new query processing procedures utilizing data dependencies which include some of the previously known query processing procedures as special cases.

Since the cost for computing joins is often very high, there are many papers which focus on the processing of joins. With respect to natural joins, queries are classified into two classes, tree and cyclic [BERNG8111]. Tree queries can be processed efficiently [BERNG8111], but cyclic query processing is known to be difficult. It has been shown that any procedure for processing cyclic queries contains a procedure for converting them into tree queries [GOODS8203]. The following procedures realize the conversion.

(1) Relation merging.
(2) Tuple-wise processing.
(3) Attribute addition.

Examples of these procedures will be given in Section 3. Method (1) is the simplest. The conversion is realized by merging all relations forming a cycle. Method (2) is known as Wong's decomposition approach [WONGY7609], which can be regarded as a conversion process based on tuple-wise horizontal decomposition of a relation. By method (3) the conversion is realized by adding attributes to relations forming a cycle [KAMBY8206]. Although methods (2) and (3) look quite different, in this paper we will show that method (2) can be regarded as a special case of method (3), since horizontal decomposition of

relations can be handled by addition of an attribute which identifies each subrelation.

In this paper we will introduce a new procedure for cyclic query processing which utilizes horizontal and vertical decompositions of relations. Vertical decompositions can be realized by functional dependencies (FDs) and multivalued dependencies (MVDs). As join operations are applied to the relations which are processed by selections, the number of tuples is usually less than the original relations and thus there are chances that FDs are satisfied. For a given query, we first list possible dependencies to be utilized and then check these dependencies. Even if such dependencies are not satisfied by a relation we can always obtain a set of relations satisfying the dependencies by horizontal decomposition.

Since our purpose is to perform joins, it is natural to obtain larger relations at each intermediate steps like procedures (1) and (3). Our procedures, however, show that relation splitting is sometimes very useful to save the processing cost. For such relation splitting only a primitive method (procedure (2)) has been known. Our procedures utilize dependencies which are usually used for database design and give very general relation splitting methods which include procedures (2) and even procedure (3) as special cases.

In Section 2 basic definitions are given. Section 3 shows comparison of the above three procedures and the new procedures introduced in this paper using the same example in order to understand the idea underlying the procedures. Section 4 discusses a query simplification procedure using FDs, and a query processing procedure using FDs and horizontal decomposition. Tuple-wise processing and attribute addition are shown to be special cases of the second procedure. In Section 5 query processing procedures using MVDs and horizontal decomposition are shown. When degenerated MVDs are used, we can always get partial solutions for all relations. General MVDs do not have such a property. Section 6 discusses conditions under which such a property is satisfied using FD- and MVD-based procedures. Section 7 summarizes procedures for the decomposition of relations utilizing dependencies.

2. Basic Concepts

A relation scheme is a finite set of attributes A_i's$(1 \le i \le n)$ and is denoted by $\underline{R}(A_1,\ldots,A_n)$ or $\underline{R}$. Each attribute A_i is associated with a value set $dom(A_i)$ which is called a domain. A relation R (on the relation scheme $\underline{R}(A_1,\ldots,A_n)$) is a finite set of total mappings from the set of attributes to the corresponding domains. Each total mapping is called a tuple and is denoted by t. A tuple t can be viewed as a n-tuple $(d_1,\ldots,d_n)$, where $d_i \in dom(A_i)$, by proper ordering of attributes. Attribute A_i in relation R_j is denoted by $R_j.A_i$. A collection of relation schemes is called a database scheme. A collection of relations corresponding to the database scheme is called a database state and is denoted by $D(R_1,\ldots,R_n)$.

A projection of a tuple t on attribute set X is denoted by t[X], which is defined as a part of t corresponding to the attribute set X. The following notations of the relational algebra will be used.

Projection: $R[X] = \{t[X] \mid t \in R\}$

θ-selection: $\sigma_{A\theta c}R = \{t \mid t[A] \,\theta\, c,\ t \in R\}$

θ-restriction: $\sigma_{A\theta B}R = \{t \mid t[A] \,\theta\, t[B],\ t \in R\}$

(Here, θ is a comparison operator such as =, >, <, etc. ,and c is a constant value.)

Natural equi-join: $R_i \bowtie R_j$

$= \{t \mid t \in R,\ t[\underline{R}_1] \in R_1,\ t[\underline{R}_j] \in R_j,\ \underline{R} = \underline{R}_1 \cup \underline{R}_2\}$

Since join operation is associative, the natural equi-join of n relations $R_1,\ldots,R_n$ is denoted by

$R_1 \bowtie \ldots \bowtie R_n$

A query Q, which consists of a qualification q and a target attribute set TA, maps a database state $D(R_1,\ldots,R_n)$ into the following relation.

$$(\sigma_q(R_1 \times \ldots \times R_n))[TA] \qquad (2\text{-}1)$$

A query of which a qualification is a conjunction of clauses of the form $R_i.A_{ik}=R_j.A_{jh}$ is called an equi-join query. If $A_{ik}=A_{jh}$ is satisfied for every clause $R_i.A_{ik}=R_j.A_{jh}$, an equi-join query is called sub-natural. Sub-natural queries of which qualifications contain a clause $R_i.A_{ik}=R_j.A_{jh}$ for every possible combination of A_{ik} and A_{jh} satisfying $A_{ik}=A_{jh}$ are called natural. We will only consider natural equi-join queries in this paper,

since any equi-join query can be transformed into a natural one by proper renaming of attributes. The concept of sub-natural is necessary for equivalent transformation of natural equi-join queries. For natural equi-join queries, the relation (2-1) becomes equivalent to the following one.

$$(R_1 \bowtie, \ldots, \bowtie R_n)[TA] \qquad (2\text{-}2)$$

Let $(R_1 \bowtie, \ldots, \bowtie R_n)[R_i]$ be a partial solution of the join for R_i. In this paper, we will develop procedures for joins which will obtain partial solutions for all relations involved in the join. Since target attribute sets are not required to be considered in our problem, we will use q to represent a query. If the complete result of the join is required, the process to obtain the partial solutions can be used as a preprocessing step.

A query graph $G_q=(V, E, L)$ corresponding to a sub-natural query q is a labeled undirected graph. V is a set of vertices, where v_i in V corresponds to relation R_i refered in q. Two vertices v_i and v_j corresponding to R_i and R_j are connected by an edge iff there is a clause $R_i.A=R_j.A$ in q. The label of the edge is the union of all such A. E is the set of edges and L is the set of labels for E.

Two queries are said to be equivalent if both will produce the same result for any database state. Two query graphs are equivalent if the two corresponding queries are equivalent.

A query is called a tree query if it is equivalent to a query whose query graph is circuit-free; otherwise it is called cyclic [BERNG8111]. There are tree queries whose query graphs have cycles [BERNG8111].

Example 1 : Consider the following four relations.

SUPERVISOR(Professor, Student, Approver)
ENROLLMENT(Student, Subject, Approver)
CLASS(Subject, Professor, Approver)
OFFICE(Professor, Room)

Here, we use P, S, J, A and R for attributes Professor, Student, Subject, Approver and Room, respectively. The meaning of these relations is self-evident. We have approvers in the first three relations. The query graph of the natural equi-join query for this database scheme is shown in Fig.1(a). The edge between CLASS and OFFICE can be removed since there are edges labeled by P between CLASS and SUPERVISOR and between SUPERVISOR and OFFICE. The query is a cyclic query whose cyclic part consists of three relations SUPERVISOR, ENROLLMENT and CLASS. One of the labels A on edges connecting three relations is redundant.

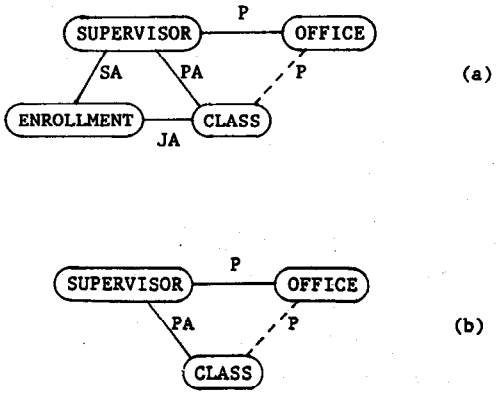


Fig.1 A cyclic query and a tree query

If only three relations except for ENROLLMENT are involved in the query, the query graph becomes the one in Fig.1(b) which represents a tree query.

There exists an efficient procedure to test whether or not a given query is a tree query [YU--07911] [BEERF8105]. A special case is treated by Bernstein and Chiu [BERNC8101].

A semi-join of R_i by R_j is denoted by $R_i \ltimes R_j$ and defined as

$$R_i \ltimes R_j = (R_i \bowtie R_j)[R_i]$$
$$= R_i \ltimes R_j[R_i \cap R_j]$$

For tree queries, there exists an efficient procedure to calculate partial solutions for all relations using semi-joins only [BERNG8111]. If no further application of semi-join changes the contents of a relation, that relation is called irreducible.

Data dependencies are important time-independent semantic constraints concerned with attribute relationships. A functional dependency (FD):X->Y is satisfied in relation R if and only if for any pair of tuples t and t', t[X]=t'[X] implies

t[Y]=t'[Y]. Throughout this paper, we assume X∩Y=∅ for an FD: X->Y. A multivalued dependency (MVD): X->>Y|Z is satisfied in relation R (where XYZ = R and Y∩Z=∅) if and only if

R = R[XY] ⋈ R[XZ]

A degenerated MVD (DMVD) [ARMSD8012] [SAGIF7903] [TANAL7808]: X=>>Y|Z is satisfied in relation R (where X ∩ Y = ∅ and X∩Z = ∅) if and only if R = $R_1 \cup R_2$ such that the FD: X->Y is satisfied in R_1 and the FD: X->Z is satisfied in R_2.

FD and MVD are constraints on R which must be satisfied at all times. A dependency which cannot be derived from the set of such constraints may be satisfied by snapshot data. For example, even if there is no restriction on the number of jobs which each person can have, at a certain instance we may have a relation satisfying the condition that each person has exactly one job. We say that there exists a temporary FD: person->job. If a snapshot relation satisfies the condition of an FD (or an MVD), we call it a temporary FD (or a temporary MVD, respectively). Note that an FD (or an MVD) is a temporary FD (a temporary MVD, respectively). Such temporary dependencies are especially important for query processing as will be shown later.

3. Elementary Consideration on Cyclic Query Processing

In this section we will show the basic idea used in our new procedure together with the three basic procedures using one simple example query shown in Fig.1(a). Let us assume the database state shown in Fig.2 is given.

SUPERVISOR

P	S	A
p_1	s_1	a_1
p_2	s_2	a_1
p_3	s_2	a_2
p_3	s_3	a_2

ENROLLMENT

S	J	A
s_1	j_1	a_1
s_2	j_2	a_1
s_2	j_3	a_2
s_3	j_3	a_3

CLASS

J	P	A
j_1	p_2	a_1
j_2	p_1	a_1
j_3	p_3	a_2
j_4	p_3	a_3

OFFICE

P	R
p_2	r_1
p_3	r_2
p_4	r_2

(a)

Fig.2 A database state of the query in Fig.1(a)

We will first show how this query can be processed by the three elementary query processing procedures. Since an efficient procedure for tree queries is known, we will only show how the query can be transformed into a tree.

(1) Relation merging : This method transforms the given query into a tree by merging all relations in each cyclic part of the query. Since the three relations SUPERVISOR, ENROLLMENT and CLASS form a cycle, we first apply the following join.

SUPERVISOR ⋈ ENROLLMENT ⋈ CLASS

The resulting query graph is shown in Fig.3, which is a tree.

Fig.3 Conversion to a tree query by relation merging

(2) Tuple-wise processing : By this method we first select one relation in a cycle and the query is decomposed into queries each of which processes just one tuple of the selected relation. This is known as the query decomposition method [WONGY7609]. For example, if we select CLASS, the following union of queries is equivalent to the given query.

$\bigcup_{t_i \in \text{CLASS}} (t_i \bowtie \text{SUPERVISOR} \bowtie \text{ENROLLMENT} \bowtie \text{OFFICE})$

Each query in the union is a tree query as shown in Fig.4. Here, relation CLASS is regarded as a single tuple relation and it can be decomposed into two relations t_i[PA] and t_i[JA]. The edge labeled by A

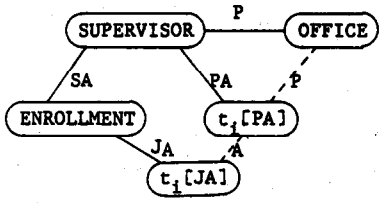


Fig.4 Each query of the tuple-wise processing

connecting the two relations is redundant since there exists another path between the two relations each of whose edge label contains A.

(3) Attribute addition : If we get the query graph shown in Fig.5 by adding attribute labels to edges then the edge between ENROLLMENT and CLASS becomes redundant, that is, the query is converted into a tree. Addition of attributes is realized during the processing of the query. That is, S, A, J values of ENROLLMENT and P, A, J values of CLASS are required to be transmitted to SUPERVISOR. In this particular example the method is equivalent to the relation merging (usually the method is better than the relation merging). The difference is caused by the fact that we can encode J values. By a mutual exchange of J values between ENROLLMENT and CLASS, we can use 1,2,... instead of the real J values in the intersection (see Fig.5 (b)). The method is very much effective when there are few J values in common in both relations. A general procedure is given in [KAMBY8206].

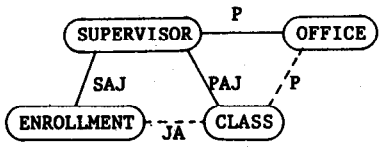


Fig.5(a) Conversion to a tree query by adding attributes

ENROLLMENT

S	J	A
s_1	1	a_1
s_2	2	a_1
s_2	3	a_2
s_3	3	a_3

CLASS

J	P	A
1	p_2	a_1
2	p_1	a_1
3	p_3	a_2

Fig.5(b) Numerical data compression of relations

(4) Use of dependencies : This method is newly introduced in this paper. In this section, we will only show the basic idea using the same example used for other procedures. If FD:A->J is satisfied by CLASS, it can be decomposed into two relations CLASS[A,J] and CLASS[A,P]. The resultant query

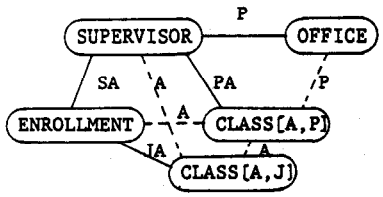


Fig.6 Use of a dependency for query conversion

graph is shown in Fig.6. The query is a tree, since the edge labeled by A between the two projections of CLASS is redundant. By the conventional procedure for tree query processing, we can get all partial solutions for relations in Fig.6. The partial solution for CLASS can be obtained by joining the results for CLASS[A,P] and CLASS[A,J].

Such a decomposition of CLASS is possible when MVD:A->>P|J is satisfied. FD:A->P and FD:A->J are special cases of the MVD. Degenerated MVD:A=>>P|J is another special case. Note that if MVD:A->>P|J which is not FD or DMVD is used, we may not be able to get the partial solution for CLASS by joining the two projections (Details about this problem will be discussed in Sections 5 and 6).

4. Query Processing Utilizing Functional Dependencies

In this section we will discuss query processing procedures using FDs, and show that the procedure is a generalized version of attribute addition and tuple-wise processing.

4.1 Query Simplification Utilizing FDs

Theorem 1 : If a query graph G contains an edge $e = \langle R_i, R_j \rangle$ with label Z such that
(1) FD: X->Y is satisfied by both R_i and R_j, and
(2) $X \subseteq Z$,
then the label of e can be replaced by Z-Y without loss of the equivalence of the query.

Proof : Let G' be the query graph after the replacement of the label of e, also let q and q' be queries corresponding to G and G', respectively. It is suffice to show the both sides' implication of q and q'.

(q => q') Trivial.

(q' => q) Let $R_i^{q'}$ and $R_j^{q'}$ be partial solutions of R_i and R_j for q', respectively. To prove the implication, we need to show that $R_i^{q'}[XY] = R_j^{q'}[XY]$ holds. $R_i^{q'}[X] = R_j^{q'}[X]$ holds from the assumption (2). (Remind that our definition of an FD: X->Y required the condition $X \cap Y = \emptyset$.) Also

$(R_i^{q'}[X] = R_j^{q'}[X]) \Rightarrow (R_i^{q'}[XY] = R_j^{q'}[XY])$ holds from the assumption (1). □

As a special case, when $X=\emptyset$, Y can be eliminated. Using the above theorem we can simplify a given query graph utilizing FDs. When the FD is satisfied by only one of R_i or R_j (say R_i), the assumption of Theorem 1 is fullfilled by performing a semi-join $R_j \ltimes R_i$. Therefore, when proper FDs hold in relations, there are cases in which the partial solutions of cyclic queries are obtained using only semi-joins.

Besides the FDs given as constraints on the database, we can use various FDs, which is summarized in the following property.

Property 1: The following FDs can be used for query simplification shown in Theorem 1.

(1) FDs in the set of database constraints.

(2) FDs produced by relational operations.

(2-1) If there is a selection operation $\sigma_{A='a'}R$ then there exists an FD:$\emptyset$->A, where 'a' is a constant value.

(2-2) If there is a restriction operation $\sigma_{A=B}R$ then there exist FDs:A->B and B->A.

(3) FDs produced by an intersection of the two relations to be joined.

(4) Temporary FDs produced during query processing can be utilized.

In case (3) we need to store the correspondence of X and Y in order to record Y values from the partial solution for a relation whose attribute set contains X.

We will show examples which use Theorem 1.

Example 2:

(1) Consider the cyclic query shown in Fig.7 (a). If a selection operation $\sigma_{B='b'}R_2$ is applied then FD: $\emptyset$->B is satisfied in R_2. Therefore by performing a semi-join $R_3 \ltimes R_2$, the label B on the edge between R_2 and R_3 can be removed. Since the resultant edge label is $\emptyset$, the edge is removed and the query becomes a tree (see Fig.7 (b)).

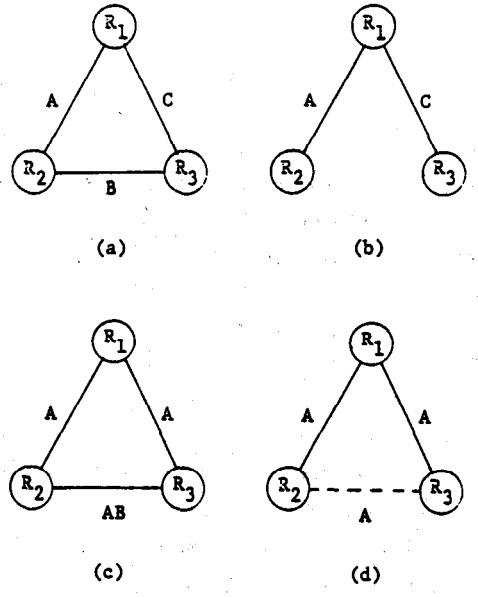


Fig.7 Simplification utilizing FDs

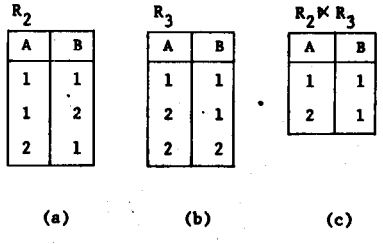


Fig.8 FD satisfied in a joined relation

(2) Consider the query shown in Fig.7(c). If a restriction operation $\sigma_{A=B}R_2$ is applied then there exists an FD:A->B. After performing a semi-join $R_3 \ltimes R_2$, we apply Theorem 1 to the edge between R_2 and R_3. As label B is eliminated we have the query graph shown in Fig.7 (d) which is a tree.

(3) If R_2 and R_3 are given as in Fig.8 (a) and (b), the FD:A->B is not satisfied. However, by performing a semi-join $R_2 \ltimes R_3$ we can get the relation R_2' shown in Fig.8(c) where the FD is satisfied. Further semi-join $R_3 \ltimes R_2$ propergates the FD to the relation R_3. Thus, applying Theorem 1, the query shown in Fig.7(c) can be transformed into the one in Fig.7(d).

In the above discussion we considered the situation when FDs are satisfied we will try to use them to simplify the given query. Another approach is first to find a proper set of FDs to simplify the query and then select a proper process such that these FDs are satisfied in the relations. This can be realized by horizontal decomposition of a relation. An example is as follows.

Example 3 : We will consider the example used in Section 3 (see Fig.1(a)). If an FD:A->J holds in the relation CLASS, we can eliminate J and the query graph shown in Fig.9 is obtained which is a tree. It is another interpretation of the process discussed in Section 3 (4). We will consider the problem that we know the FD is useful for query simplification and it is not satisfied by CLASS. Fig.10 shows an example of CLASS which does not satisfy the FD. If we decompose the relation into two relations such that one relation R_1 has the first three tuples and the second relation R_2 has the next two tuples and the third relation R_3 has the last tuple, each of the relations satisfies the FD. We can decompose the query into three subqueries, such that in each subquery CLASS is replaced by R_1, R_2 or R_3. The partial solution for the original query is the union of the partial solutions for three subqueries.

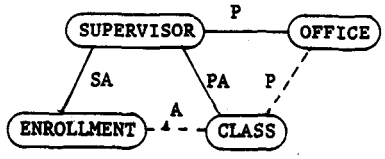


Fig.9 Use of an FD : A→J for query conversion

CLASS

J	P	A
j_1	p_2	a_1
j_4	p_1	a_2
j_2	p_2	a_3
j_2	p_1	a_1
j_3	p_1	a_3
j_1	p_2	a_3

Fig.10 Horizontal decomposition of a relation

We have the following procedure utilizing FDs and horizontal decomposition.

Procedure 1 : (Conversion of a query into a tree utilizing FDs and horizontal decomposition)

(1) For each cyclic part of the query graph apply the following steps.

(2) Select one edge labeled Z.

(3) Let X be the intersection of the labels of edges in the cycle and Z=XY (see Fig.11(a)).

(4) Since if FD: X->Y is satisfied the cycle is removed by performing a semi-join (see Fig.11 (b)), we apply horizontal decomposition to one of the two relations corresponding to the terminal nodes of the edge so that each subrelation satisfies the FD. (The method to decompose a relation based on FDs is shown in Section 7.)

If the given query has more than one cycle, selection of edges at step (2) can be done by finding all edges not contained in one particular spanning tree. Basic consideration on the selection of such a spanning tree is given in [KAMBY8206]. For a large cycle, X at step (3) tends to be ∅. When X=∅ this procedure is equivalent to the attribute addition method, as will be shown in Section 4.2.

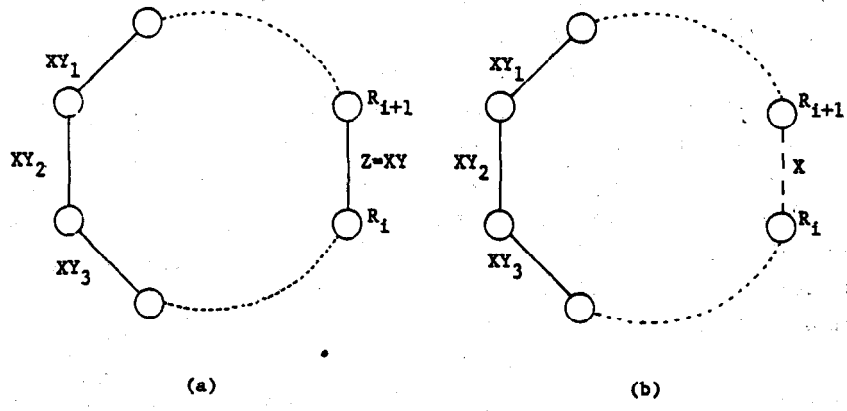


Fig.11 Removeal of a cycle

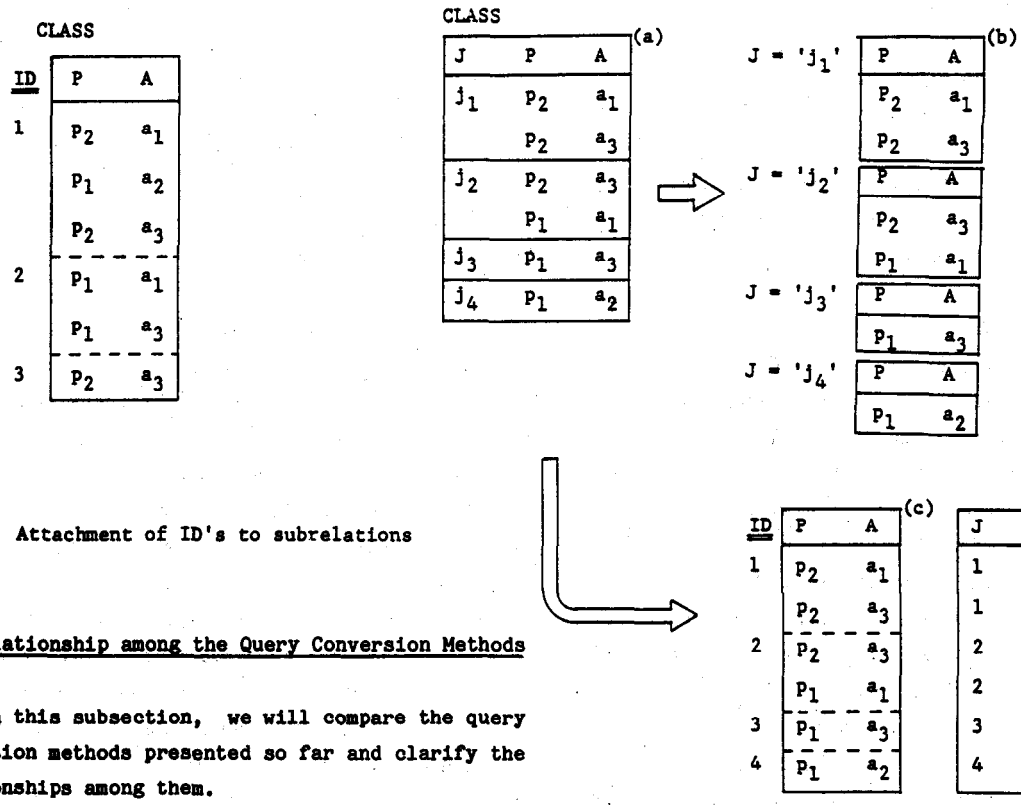


Fig.12 Attachment of ID's to subrelations

Fig.13
Horizontal decomposition method utilizing $\phi \rightarrow J$ and attribute addition method

4.2 Relationship among the Query Conversion Methods

In this subsection, we will compare the query conversion methods presented so far and clarify the relationships among them.

We use the relation in Fig.10 as example again. When the relation is decomposed horizontally so that each of the subrelations satisfies the FD:A->J, each subquery becomes a tree query and processed separately. However if we handle the subqueries simultaneously, we need to distinguish them. Thus identifiers of subqueries must be attached when semi-joins are performed. For example, when performing a semi-join SUPERVISOR $\underset{FA}{\ltimes}$ CLASS, a relation with identifiers (IDs) shown in Fig.12 is transmitted to relation SUPERVISOR. Note that the IDs also identify different FDs of the same syntactical form A->J.

Next we will show that when the left hand side of the FD is ϕ, this horizontal decomposition approach is essentially the same as the attribute addition method. When the relation CLASS satisfies the FD:ϕ->J, the same discussion as in the case of FD:A->J holds since the former FD implies the latter. Thus the query can be easily transformed into the tree query shown in Fig.9. If the FD:ϕ->J does not hold on CLASS, we decompose it horizontally again so that each subrelation satisfies the FD. This decomposition is easily achieved by first performing GROUP-BY[J] on CLASS and then regarding each subrelation as a separate relation (see Fig.13 (a), (b)). When we handle these subqueries simultaneously, we require IDs to distinguish them. However, in this case, values of attribute J can be used as IDs since there is a one-to-one correspondence between subrelations and J-values. Thus J-values are attached to the CLASS[P, A] relation when performing the semi-join SUPERVISOR $\underset{FA}{\ltimes}$ CLASS (see Fig.13 (c), (d); Data compression technique shown in Fig.5 is applied in the relation of Fig.13 (d)). This method, however, is the attribute addition approach.

Thus the only difference between horizontal decomposition approach and attribute addition approach when the left hand side of the FD is ϕ, is that each subquery is considered to be processed separately in the former approach while all of them are regarded to be processed simultaneously in the latter approach. The horizontal decomposition by FDs with nonempty left hand side is usually more economical than attribute addition, since the

number of subrelations usually decreases.

Wong's decomposition approach may be interpreted to mean that we use the horizontal decomposition where each subrelation consists of only one tuple. Since arbitrary FDs are satisfied in these subrelations (∅->X is satisfied for a relation with attribute set X, thus any ∅->Y (Y⊆X) is satisfied which is required by the attribute addition or equivalently the FD-based procedure), we can convert the original query into a tree query if we select at least one relation to be decomposed for each cycle in the query graph.

5. Query Processing Utilizing Multivalued Dependencies

In Section 4 it was shown that Procedure 1 is equivalent to the attribute addition method in many cases. We will generalize the procedure, utilizing MVDs in this section. These procedures are entirely new and subsume Procedure 1 as a special case.

5.1 Conversion of Queries into Tree Using MVDs

The conversion procedure from a cyclic query to a tree query utilizing FDs (see Fig. 11) can be easily generalized to the case utilizing MVDs. Consider the cycle shown in Fig.14 (a).

(1) Select a relation, say R_0, in the cycle.

(2) Let X be the intersection of labels of edges in the cycle. If an MVD X'->>Y|Z holds on R_0, where R_0=X'YZ and X'⊆X, then decompose R_0 into two relations R_0^1(X'Y) and R_0^2(X'Z).

(3) Since the edge between R_0^1 and R_0^2 is redundant (see Fig.14 (b)), the original cycle is eliminated. We can use a spanning tree for the selection of relations to be decomposed, when there are many cycles. A general procedure is shown below.

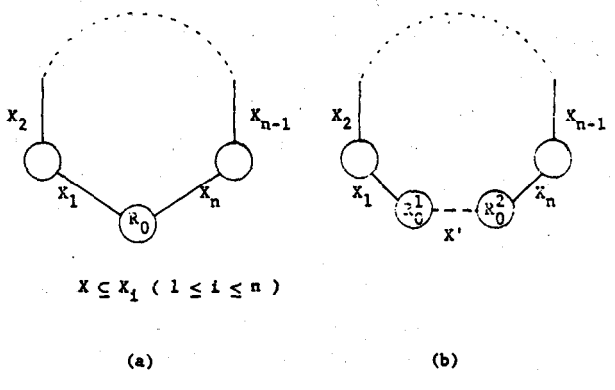


Fig.14 Removal of a cycle by an MVD-based decomposition

Procedure 2 : (Conversion of a cyclic query into tree utilizing MVDs)

(1) Select a spanning tree in the query graph.

(2) For each edge not contained in the spanning tree, select one relation between relations corresponding to the two nodes incident to the edge.

(3) Determine the MVD which should hold on the relation corresponding to the node chosen in (2), in order to remove the cycle.

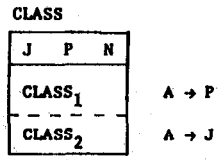
(4) Decompose the relation using the MVD.

(5) Repeat (2)-(4) for every edge not contained in the spanning tree.

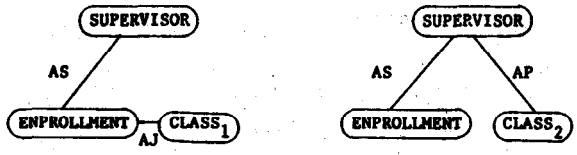
Similar to the case of FDs, the left hand side of the MVDs selected at step (3) becomes ∅ in many cases. Thus we need to use the horizontal decomposition in Procedure 1.

5.2 Query Processing Using DMVDs

In this subsection, a new query processing approach using DMVDs is introduced. First an example is given. We will use the query which consists of three relations SUPERVISOR, ENROLLMENT and CLASS in Example 1. Assume a DMVD:A=>>P|J is satisfied in the relation CLASS. Also let $CLASS_1$ and $CLASS_2$ be horizontally partitioned subrelations in which FDs:A->P and A->J are satisfied, respectively (Fig.15 (a)). Following the discussions of subsection 5.1, the original query can be converted to tree queries of Fig.15 (b) and (c). Each query graph corresponds to $CLASS_1$ and $CLASS_2$, so the result of the original query is the union of the results of these two queries. As in the case of FDs, we can generalize the idea for the situation where the original relation does not satisfy a DMVD. Consider a cyclic query shown in Fig.16 (a). Horizontally partitioning the relation R_0 so that each subrelation satisfies an FD:∅->A or

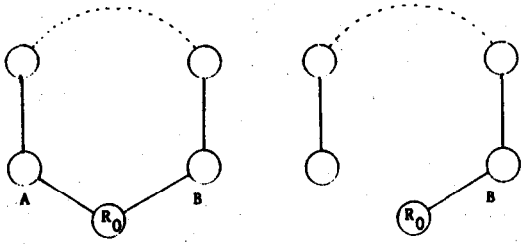


(a)

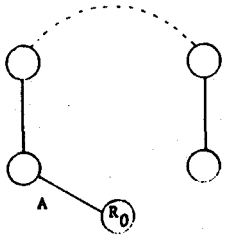


(b) (c)

Fig.15 Query processing using a DMVD



(a) (b)



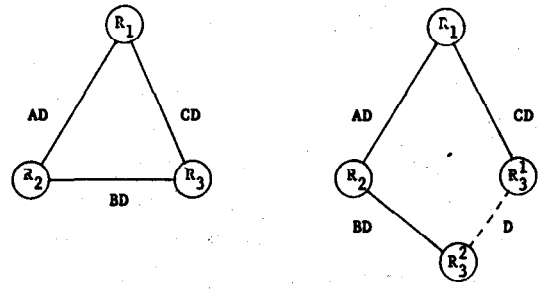
(c)

Fig.16 Conversion of a Query by a DMVD

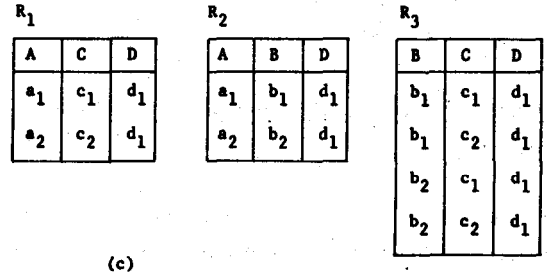
ϕ->B, we can obtain subqueries as shown in Fig.16 (b) and (c). The number of subqueries is equal to that of subrelations. A procedure for such a decomposition will be shown in Section 7.

5.3 Query Processing Using MVDs

Although MVDs can also be utilized for query conversion, there is a qualitative difference between the usage of FDs (or DMVDs) and that of MVDs. The difference is whether the partial solutions of all relations can be obtained using only semi-joins or not. When MVDs are utilized for



(a) (b)



(c)

Fig.17 The case when not all the partial solutions are obtained by semi-joins only

the conversion, the partial solution for the relation decomposed by an MVD cannot in general be obtained by semi-joins only. The following example clarify this.

Example 4 : Consider a query shown in Fig.17 (a). If MVD: D->>B|C is satisfied on R_3, it can be decomposed into two relations R_3^1(CD) and R_3^2(BD). Thus the query can be converted into the tree as shown in Fig.17 (b). Therefore the partial solutions of R_1, R_2, R_3^1 and R_3^2 can be easily obtained, while that of R_3 is not. To confirm this, just observe the contents of the relation in Fig.17 (c) where all three relations are irreducible and R_3 satisfies the MVD:D->>B|C, yet R_3 is not the partial solution. Therefore the method can be used, if the target attributes do not contain both B and C.

In the case of FDs we used horizontal decomposition when FDs are not satisfied. For MVDs we can use overlapped decomposition called cover. If an MVD is not satisfied in the original relation, the covering of the relation is obtained such that each cover satisfies an MVD (see Fig.18).

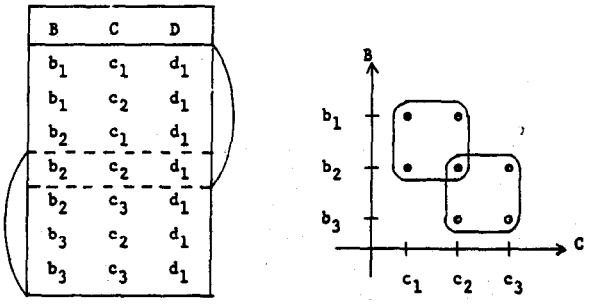


Fig.18 Covering of the relation

6. Obtainability of Partial Solutions of Relations

As shown in previous sections, when FDs and DMVDs are used for query conversion, partial solutions of all relations can be obtained by semi-join only, while if MVDs are used that is not always guaranteed. In this section we consider more general situations where some FDs and MVDs are given on each relation. The problem is whether the partial solution of a particular relation can be obtained using only semi-joins. A sufficient condition for this problem and a comprehensive example will be given.

The decision is made by transformations of a given query graph using FDs and MVDs. The following two rules are used for the transformation.

FD-rule : If an FD:X->Y is satisfied at a relation R and if there is an edge with label Z ($\supseteq$ X) incident to the node in the query graph, replace the label Z by Z-Y (Note that this rule is an application of Theorem 1 in Section 4). By the replacement of labels, there may arise a situation that some relations have the common attribute, yet they are not connected by edges with the label containing the common attributes in the resultant query graph. In that case, rename the occurrences of attributes properly so that the resultant graph represents a query graph.

MVD-rule : If an MVD:X->>Y|Z is satisfied at a relation R, split the node into two nodes corresponding to R[XY] and R[XZ]. Rearrange edges incident to the node R so that the resultant graph represents a query graph.

Let $f_{i1},\ldots,f_{ik_i}$ be FDs and m_i be MVDs holding on the relation R_i $(1\leq i\leq n)$ (We assume that up to one MVD is satisfied at each relation, for simplicity). Let F_{ij} and M_i represent the transformation process according to the FD:f_{ij} and the MVD:m_i, respectively. The following theorem holds.

Theorem 2 : If the query graph after the application of a sequence of transformations $M_1,\ldots,M_{i-1},M_{i+1},\ldots,M_n,F_{11},\ldots,F_{nk_n}$ represents a tree query, then the partial solution of relation R_i can be obtained only using semi-joins.

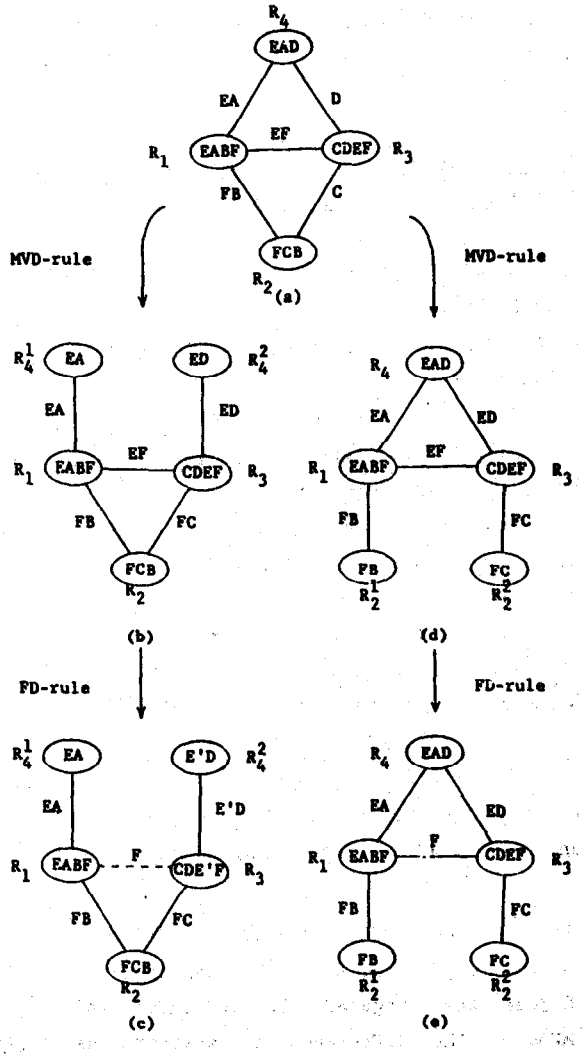


Fig.19 Transformation of a query using FDs and MVDs

Example 5 : Consider a query whose query graph is shown in Fig.19 (a). Assume that the following dependencies hold.

f_1 : F->E at R_1
m_2 : F->>B|C at R_2
m_4 : E->>A|D at R_4

First let us decide whether the partial solution of R_2 can be obtained using only semi-joins or not. Applying M_4, the query graph in Fig.19 (b) is obtained. Then F_1 is applied to yield the final query graph in Fig.19 (c) (Note that attribute E of relations R_3 and R_4^2 are renamed to E'). Since the final query graph represents a tree query, the partial solution of R_2 can be obtained only by semi-joins. However in the case of R_4, this is not true since the final query graph after the transformations M_2 and F_1 does not represent a tree query (see Fig.19 (e)).

7. Procedures for Horizontal Decomposition of a Relation based on FDs and MVDs

In this section, we will give procedures to detect temporary dependencies together with procedures for horizontal decomposition of relations.

Detection of temporary FDs is considered to be expensive. We will show, however, the cost is very small when semi-joins are used.

Semi-join requires an intersection of two sets. It is usually performed by the following two methods.

(1) Sorting of values of the join attributes.
(2) Use of hash values.

The both methods can be used to check the existence of temporary FDs. We will consider the case shown in Fig.11 (a), where Z=XY is a lable on the edge between R_i and R_{i+1} and the existence of an FD X->Y is required to be checked.

Procedure 3 : (Checking of a temporary FD X->Y utilizing the sorting process by Z=XY)

(1) For sorting by Z, we put more weight on X-values than Y-values. That is, tuples are sorted first by X-values and then by Y-values.

(2) If the FD X->Y is not satisfied, there are tuples with different Y-values and the same X-values. Since these tuples are consecutively arranged, checking of the existence of the FD is not expensive (linear time).

Checking of a temporary FD utilizing the hashing process for Z is also simple. Instead of using Z-values for hashing, we need to use X-values for hashing so that tuples with the same x-values will be put into the same bucket.

When the FD is not satisfied we need to decompose a relation into subrelations satisfying the FD. The process is not expensive either.

Procedure 4 : (Horizontal decomposition of a relation into subrelations satisfying an FD X->Y) (Symbols in Fig. 11 are used)

(1) Find tuples which do not satisfy the FD condition in R_i. Each tuple has at least one other tuple having the same X-value.

(2) Transmit the Z-values of the tuples to the site of R_{i+1} and perform a semi-join.

(3) Find tuples not satisfying the FD condition in the result. These tuples will violate the FD condition in the result of joining R_i and R_{i+1}. Classify these tuples with X-values such that tuples in the same set have the same X-value. Each set has at least two elements.

(4) Decompose R_{i+1} into subrelations such that each subrelation contains at most one tuple from each set defined at step (3). The number of subrelations is the cardinality of the largest set.

When DMVDs from ϕ are considered, the following heuristic procedure can be used to reduce the number of subqueries.

Procedure 5 : (Horizontal decomposition of a relation into subrelations satisfying a DMVD)

(1) Let ϕ=>>A|B be a DMVD to be satisfied at relation R.

(2) Count the number of tuples for A='a' or B='b'.

(3) Let a_i (resp. b_j) be the A (resp. B) -value which have the largest number $||a_i||$ (resp. $||b_j||$) of tuples for A='a' (resp. B='b').

(4) If $||a_i|| \geq ||b_j||$ (resp. $||a_i|| < ||b_j||$), let R[A='a_i'] (resp. R[B='b_j']) be a new subrelation and separate it from R.

(5) Repeat (1)-(4) until R contains no tuple.

To find the minimum horizontal decomposition using MVDs whose left hand side is ϕ is NP-complete, since it is equivalent to find a cover consists of the minimum number of complete bipartite subgraph in a given bigraph [GAREJ79].

In distributed databases, there are in general several relations located at one site. Therefore, there may be situations when the local processing is first performed which takes a join of relations. An MVD is satisfied in the relation generated by joining two relations. Such MVDs can be utilized for the query processing. In this case we do not need an expensive procedure to detect MVDs.

Acknowledgments

The authors are grateful to Professor Shuzo Yajima and members of Yajima Lab. for helpful discussions.

References

[ARMSD8012] Armstrong,W.W. and Delobel,C., "Decompositions and Functional Dependencies in Relations.", ACM TODS, Vol.5, No.4, pp.404-430, Dec. 1980.

[BEERF8105] Beeri,C., Fagin,R., Maier,D., Mendelzon,A., Ullman,J. and Yannakakis,M., "Properties of Acyclic Database Scheme", Proc. of 13th Annual ACM Symposium on Theory of Computing, pp.355-362, May 1981.

[BERNC8101] Bernstein,P.A. and Chiu,D.M., "Using Semi-Joins to Solve Relational Queries", JACM, Vol.28, No.1, pp,25-40, Jan. 1981.

[BERNG8111] Bernstein,P.A. and Goodman,N., "Power of Natural Semijoins", SIAM J. Comput., Vol.10, No.4, pp.751- 771, Nov. 1981.

[GAREJ79] Garey,M.R. and Johnson,D.S., Computers and Intractability: A Guide to The Theory of NP-Completeness, W.H.Freeman and Company, 1979.

[GOODS8203] Goodman,N. and Shmueli,O., "The Tree Property is Fundamental for Query Processing (Extended Abstract)." Proc.1st ACM SIGACT-SIGMOD Symposium on PODS, pp.40-48, Mar.1982.

[KAMBY8206] Kambayashi,Y., Yoshikawa,M. and Yajima,S., "Query Processing for Distributed Databases Using Generalized Semi-Joins.", Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp.151-160, June 1982.

[SAGIF7903] Sagiv,Y. and Fagin,R., "An Equivalence between Database Dependencies and a Subclass of Propositional Logic.", IBM Res. Rep., RJ2500, Mar. 1979.

[TANAL7808] Tanaka,K., Le Viet,C., Kambayashi,Y. and Yajima,S., "A File Organization Suitable for Relational Database Operations.", Lecture Notes in Computer Science 75, pp.193-227, Aug. 1978.

[WONGY7609] Wong,E. and Youseffi,K., "Decomposition - A Strategy for Query Processing.", ACM TODS, Vol.1, No.3, pp223-241, Sep. 1976.

[YU--07911] Yu,C.T. and Ozsoyoglu,M.Z., "An Algorithm for Tree-Query Membership of a Distributed Query", Proc. of 3rd International Computer Software and Applications Conference (COMPSAC), pp.306-312, Nov. 1979.