Regular Paper

Generating Nested SQL Queries for Documentation

MOHAMED E. EL-SHARKAWI † and YAHIKO KAMBAYASHI ††

In this paper, we develop procedures to transform a user's query into a nested form suitable for query documentation, editing, and reusability purposes. We discuss generating nested form of the three query types : chain, tree, and cyclic. For tree and cyclic queries, there may be more than one nested form. Thus, we have to determine the order of writing query blocks, we use the notion of *join strength*. Joins are classified by their strength of relationship. Strong joins are selected to connect adjacent query blocks. For a tree query, the first blocks is selected by the output attribute, the second one is selected as that of strongest join with the previous block and so on. Chain queries are handled as a special case of tree queries. A cyclic query cannot be written in the nested form. A form which is in part nested and in part unnested, called seminested form is used. This form is generated by first finding a minimum spanning tree of the cyclic query graph. This tree contains most strong joins. We discuss also the case when the query output is obtained from multiple blocks. Since it is not possible to write such a query in the nested form, the query is converted into a modified nested query in which the output is obtained from only one block.

1. Introduction

Relational databases are heavily used as a component of information systems. Since the introduction of the relational data model by Codd[1], several commercial database management systems (DBMS's) based on the model have been developed[2)~4)]. One of the advantages of the model is supporting *non-procedural* query languages to manipulate the data. That is, a user describes the output of his query, not how to access data satisfying his requirements. That is in contrast of network and hierarchical data models. The original query languages suggested by Codd, relational algebra and relational calculus, are not user friendly. Several high level query languages have been proposed, SQL, QBE, QUEL to name some. Some of these languages are easy to use. For example, in QBE[5] it is easy to write queries through its visual representation of relations. In some other languages it is easy to understand a query's meaning. In SQL, a complex query may be written in the nested form as consecutive query blocks. In this way, editing and reuse of an existing query is easy.

Query understandability should be provided and enhanced in order to document, edit, and reuse queries. User interfaces usually concentrate on ease of written queries, neglecting understanding previously written queries. Studying query understandability issues is important for the following reasons :

(1) System documentation is one of the most important phases of building software systems. Documentation of database queries is also important.

(2) Generating understandable queries is a component of an integrated interface named ENLI[6]. ENLI has the following characteristics : 1- supports naive users with an English like query language. 2- uses QBE as an intermediate language that supports trained users, who write interactive simple queries, 3- generates for a given query its equivalent and understandable nested form using procedures in this paper, 4- generates English meanings of SQL queries[7], and 5- generates an understandable form of a query's output, by presenting the output in an unnormalized relation.

(3) Current DBMS's permit users to store and edit their queries. Editing nested SQL queries would be easier than editing unnested ones.

(4) Query modification is a frequent operation when editing queries. It is easy to modify a nested query. Dropping and adding joins to a query, splitting a query into two queries, or

† Department of Mathematics, Faculty of Science, Kuwait University
†† Integrated Media Environment Experimental Laboratory, Faculty of Engineering, Kyoto University

combining several queries into a single query can be performed easily on nested queries.

(5) Using procedures presented in this paper, a core for a new query editor can be build. The new query editor is shown in Fig. 1. Figure 1 (a) shows the current editing process. Figure 1 (b) shows the new editor. A query is edited using a text editor, procedures presented here are used to generate a more understandable form of the query. The procedures are also part of the query design facility which is necessary to realize a database workbench[8].

In this paper, we discuss how to convert a SQL query into a more understandable form. SQL is now considered a standard relational query language and is supported in many commercially available relational database systems. Our approach to generate SQL queries for documentation is based on transforming an unnested query into an equivalent nested form which is suitable for documentation. We shall define which nested form is suitable for documentation in Section 3. If the query is in the nested form, we first transform it into its equivalent unnested one since the nested form given by the user may not be the best for documentation.

The following example shows the understandability of the nested form over the unnested one.

Example : Consider a database for suppliers, parts, and projects. It consists of five relations :

Supplier (S #, SNAME, CITY #)
Part (P #, PNAME)
SP (S #, P #, QTY)
Project (PJ #, P #)
City (City #, CNAME)

Underlined attributes are the keys. Consider the query :

Find supplier names of suppliers in Tokyo who supply parts to project number DB 92.

The query is written in the unnested and nested forms as follows :

```
SELECT   SNAME
FROM     Supplier, Part, SP, Project, City
WHERE    Supplier. S #=SP. S # AND
         Supplier. CITY #=City. CITY #
         AND
         SP. P #=Project. P # AND
         Project. PJ #="DB 92" AND
         City. CNAME="TOKYO"
```

The nested form of the query is :

```
SELECT   SNAME
FROM     Supplier
WHERE    S #=
         SELECT     S #
         FROM       SP
         WHERE      P #=
             SELECT     P #
             FROM       Project
             WHERE      PJ
                        #="DB 92"
AND      Supplier. CITY #=
           SELECT     CITY #
           FROM       City
           WHERE      CNAME=
                            "TOKYO"
```

The paper is organized as follows. In the next section, basic concepts and related work are reviewed. Section 3 discusses basic conversion procedures. The notion of join strength is introduced, next we give a preprocessing step to convert a given query into an equivalent one that is written in a deep level of nesting, next procedures to handle chain and tree queries are given. Section 4 gives a procedure to transform cyclic

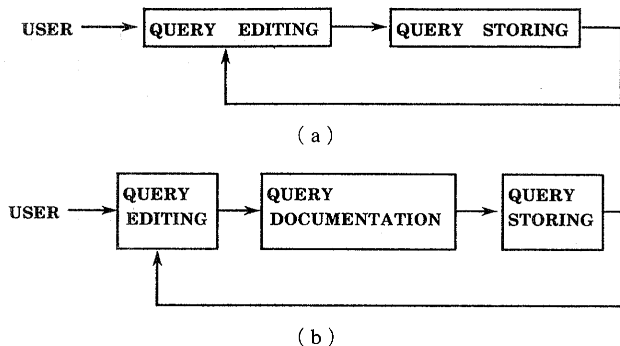


Fig. 1 Structure of a new query editor.

queries to nested form. Section 5 discusses the case when the output is obtained from more than one query block. Section 6 concludes the paper.

2. Basic Concepts

2.1 SQL Syntax

In this section we review the part of SQL syntax that is relevant to our discussions. A complete syntax of SQL can be found in [Date][9]. A basic SQL query consists of three clauses, two are mandatory SELECT and FROM and one is optional WHERE. These three clauses constitute a *query block*. The SELECT clause enumerates output attributes. FROM clause names relations involved in the query. The WHERE clause contains conditions (predicates) that should be satisfied by the output.

In the WHERE clause, predicates to be satisfied are:

(a) A simple predicate:

Attribute [Comparison Operator] Constant

(b) A join predicate:

Attribute1 [Comparison Operator] Attribute2

(c) A nested predicate:

Attribute \ Constant [Comparison Operator] Query Block

there are some other predicates which are not relevant to our discussions, these are set, minus, and union predicates. To write a query that contains any of these predicates, the query should be nested.

Comparison Operators considered are $=$, $\neq$, $>$, $\geq$, $<$, $\leq$. Predicates may be combined using AND and OR operators and negated using NOT.

To write a join query in SQL there are two forms, unnested and nested. In the unnested form the query is written as a single block. Names of all relations involved in the query are written in the FROM clause. Joins are represented as join predicates in the WHERE clause. In the nested form, joins are represented as nested predicates. SQL forms are defined as follows:

Definition 1: A query is said to be written in the *unnested form* when all the joins are written as join predicates.

Definition 2: A query is said to be written in the *pure nested form* when all the joins are written as nested predicates.

Definition 3: A query is said to be written in the *seminested form* when some of the joins are written as join predicates and some are written as nested predicates.

Throughout the paper, we use the term nested form to mean pure nested form.

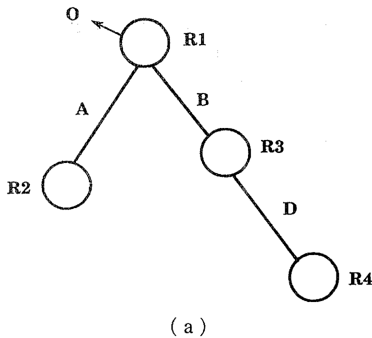
In this paper we consider Select-Project-Join (SPJ) queries. Queries that contain set, minus, or union predicates should be written in the nested form.

Definition 4: A query is an *SPJ query* if it consists of a select or a project operation (or both) and a join operation.

2.2 Query Graphs

To formalize our discussions, we use graph representation of queries.

Definition 5: A query graph $G=(N, E)$ is a labelled undirected graph. N is the set of nodes and E is the set of edges in the graph. Each node i in N corresponds to a query block. One block contains one or more relations. There is an edge between nodes i and j when a relation in the block represented by node i is joined with a relation in the block represented by node j. Each edge e_{ij} is labelled with the attribute that joins blocks i and j. If there is more than one attribute joining the two blocks, there is an edge corresponding to each attribute. Let n be the



(a)

```
SELECT R1.O
FROM R1,R2,R3,R4
WHERE  R1.A = R2.A
AND    R1.B=R3.B
AND    R3.D=R4.D
```

(b)

```
SELECT R1.O
FROM   R1
WHERE  R1.A =
   SELECT R2.A
   FROM   R2
AND    R1.B =
   SELECT R3.B
   FROM   R3
   WHERE  R3.D=
      SELECT R4.D
      FROM   R4
```

(c)

Fig. 2 SQL query represented in unnested and nested forms.

number of nodes, among these nodes one is distinguished as the *root* of the graph. The root is the node that corresponds to the query block in which the query output is specified. This node is called *output node*. If there is more than one output node, one is selected as the root.

Definition 6: A node i which is connected to K_i nodes is called to have *join degree* K_i. Among the K_i nodes a node NtR_i (Nearest to Root) is defined as the node with minimum distance to the root.

Note that for some nodes $NtR_i = i$ (i. e. node i is directly connected to the root).

Definition 7: The *WHERE-degree* of a node i is defined as the join degree of i reduced by one.

When there is no confusion degree is used to mean join degree.

We define three types of query graphs: chain, tree, and cyclic.

Definition 8: A graph is a *tree* graph if it has n nodes and $n-1$ edges. A node of degree 1 is called a *leaf node*. Nodes that are not leaves are called *inner nodes*.

Difinition 9: A graph is a *chain* graph if it is a tree graph such that there is only two nodes of degree one (the root and the leaf) and every other node has degree two.

Definition 10: A graph is a *cyclic* graph if it is not a tree graph.

Definition 11: The *distance* between two nodes i and j, denoted $d[i, j]$, is defined as the number of edges in the path that connects the two nodes.

Note that for a cyclic graph, there may exist two nodes u and v, such that there is more than one $d[u, v]$.

Throughout the paper we assume that every graph is connected.

Query graphs that correspond to chain, tree, and cyclic queries are given in Fig. 3(a), (b), and (c) respectively.

A query is a tree query if its corresponding graph is a tree graph, it is chain query if its corresponding graph is a chain graph, and it is cyclic query if its corresponding graph is a cyclic graph.

2.3 Related Work

The query conversion approach is widely used in the domain of query processing. A query is transformed to a new form that is more amenable for efficient processing (e. g. Refs. 10), 11)). Reference 12) shows transforming SQL queries into relational algebra generates wider optimization plans. Reference 13) transformed nested SQL queries into their equivalent unnested queries. An existing query optimizer is designed to efficiently process unnested queries. In Ref. 14), a set SQL query is transformed into a non-set query in order to minimize transmission cost in distributed environments.

Work on user interfaces aim at presenting users with easy ways to express their queries. They did not consider understandability of queries. The work in Refs. 15), 16) discussed generating English sentences that express the meaning of SQL queries. The problem with this approach is that the generated English meaning may not be used again.

Work in this paper is not concerned with efficient processing of queries. Our aim is to transform a query from its form to a more *understandable* form.

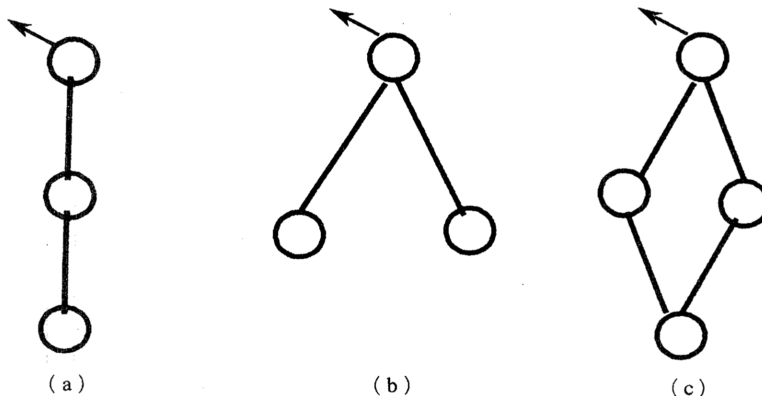


Fig. 3 Query graphs for chain, tree, and cyclic queries.

3. Basic Conversion Procedures

In this section we shall develop procedures for generating nested form for tree queries. The next section discusses cyclic queries. We assume that there is only one block from which the output is obtained, generalization will be given in Section 5.

3.1 Strength of Joins

Although the join structure of a query may be tree or cyclic, SQL requires a one dimensional representation. Thus, it is necessary to determine the order of writing query blocks. It is reasonable that two blocks which are joined *strongly* be placed adjacent to each other. We need, then, to determine join strength. For example, consider a tree query graph where some inner node i have WHERE-degree two (nodes j and k). We have to decide the order of writting query blocks corresponding to these two nodes. Generaly, for a query block corresponding to a node i, that has degree K_i, will have a WHERE clause with K_i-2 AND's. We need to decide which join will be written in the WHERE clause, which in the first AND, and so on.

We present some criteria by which the order of writing the joins is determined. The most strong join is written in the WHERE clause and the least strong join is written in the last AND. These criteria are based on perceiving database relations as representing real-world entities and relationships between the entities. We may think of a join between two relations as a way to connect between some real-world concepts. Some joins may represent strong connections or even collect information about a single entity. This information was distributed among several relations as a result of the normalization process[17]. In a query graph edges are assigned weights to represent join strength, the minimum the weight, the strong the join.

For some node i of WHERE-degree k_i, $k_i \geq 2$, criteria to decide strength of joins are as follows:

(1) If there is one and only one join such that the join attribute is the key of node i (a key of a node is a key of one of the relations in the block represented by the node), we consider this join as the most strong join. If all the joins are keys of relations in the block, the one which is the key of the relation from which the output is obtained is considered as the most strong join. The reason is that a key of a relation is used identify an entity or a relationship in the world. Thus joining two relations by a key of one of them is a way to collect together information on a single real-world entity.

(2) If there is more than one join such that the joining attributes are alternative keys of the node i and one of these attributes is in the output of the block, this join is considered as the most strong join. This attribute is playing an important role (the output of the block), and if it is the query output, the user is interested in it.

(3) If there is no join attribute which is the key of node i, and, however, one of the join attributes is a key of another node in k_i, this join is considered as the most strong join. If there are more than one attribute satisfying this condition, we select the one corresponding to the node of minimum degree.

(4) Consider a case where there is a node i that has WHERE-degree k_i and the node that has the most strong join with i is j. In the subgraph rooted at j, there is some node l such that d[j, l] is longer than all d[m, v], for any node m in k_i, and for any node v in the subgraph rooted at m. Here we have a tradeoff, if we write the join between i and j first, the rest of joins between i and any node in k_i will be written so fare from the block corresponding to node i. In this case, we combine (join) the two nodes, i and j, into one node (block) and then write the query. The following example illustrates this situation.

Example: Consider the graph in Fig. 4(a), for simplicity each node B_i corresponds to a relation named B_i. Assume that the join between blocks B_1 and B_2 is stronger than the join between blocks B_1 and B_3. According to the previous criteria, the join between B_1 and B_2 should be represented first, the query will be written as shown in Fig. 4(c). However, the path from the root node to the leaf node of B_2 is longer than the path containing B_3. It may be better then to combine nodes B_1 and B_2 together as in Fig. 4(b). After combining the two nodes, it is not needed to select which node among B_3 and B_4 to represent next, since B_3 is considered strong relative to the original output node B_1. The query is written as shown in Fig.

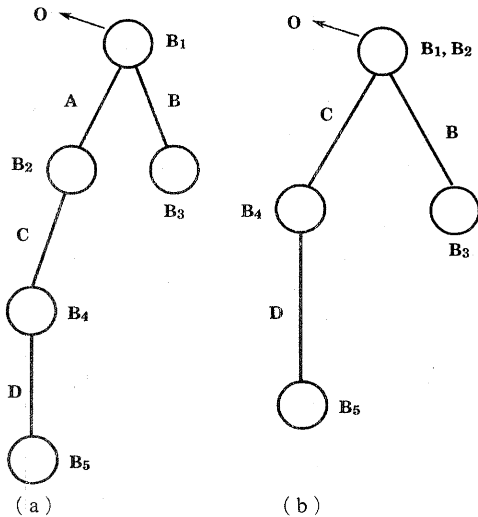


Fig. 4 Case when the strongest join *i* (b) the longest path.

```
SELECT  B1.O
FROM    B1
WHERE   B1.A =
     SELECT  B2.A
     FROM    B2
     WHERE   B2.C =
        SELECT  B4.C
        FROM    B4
        WHERE   B4.D=
           SELECT  B5.D
           FROM    B5
AND         B1.B =
     SELECT  B3.B
     FROM    B3
```

(c)

```
SELECT  B1.O
FROM    B1, B2
WHERE   B1.A = B2.A
AND        B1.B =
        SELECT  B3.B
        FROM    B3
AND        B2.C =
        SELECT  B4.C
        FROM    B4
        WHERE   B4.D=
           SELECT  B5.D
        FROM    B5
```

(d)

Fig. 4 (continued) SQL queries representing Fig. 4 (a) and (b).

4(d), combining B_1 and B_2 is the join between the two relations represented by the two nodes.
(5) If none of the join attributes satisfies the above criteria, all joins are considered to have same join strength. We consider the subtree with minimum weight as the most strong one and represent it before the others.

3. 2 Equivalence Transformation of Query Graphs

When p relations are joined by identical attribute, we can apply the equivalence transformation shown in Fig. 5.

We use this transformation as a preprocessing step to convert a graph into another equivalent one that will be written in deep level of nesting. For example, consider the query graph given in Fig. 6(a), it may be converted into the graph in Fig. 6(b).

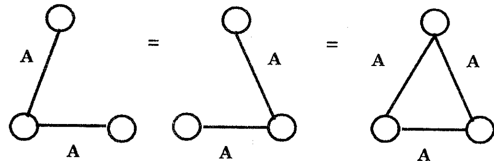


Fig. 5 Equivalent query graphs.

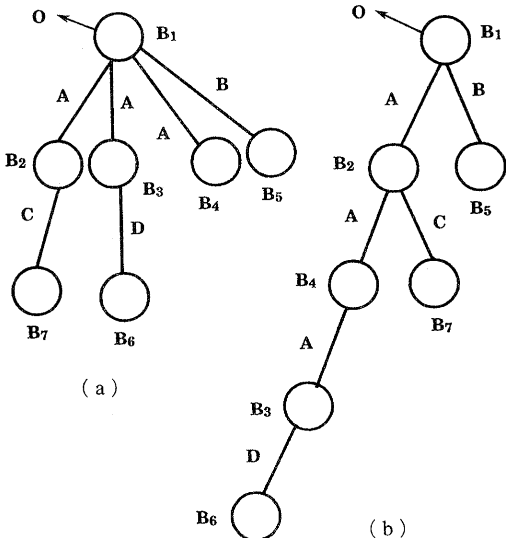


Fig. 6 Two equivalent tree graphs.

This situation can be stated as follows :

Given a query graph having some node i of join degree K_i (i. e. there are K_i nodes connected to i). Among the K_i nodes there are m nodes such that all edges that connect the m nodes to node i have same label L. We can use the mentioned transformation to convert this graph to another equivalent one whose corresponding query will be written in a deeper level of nesting than the original graph. The following procedure, RG, makes this transformation. The procedure performs as follows. First, calculates for each node j in m the distance between the node and the root, i. e. $d[j, r]$. We assume that there is a procedure that accepts a graph, two nodes, and calculates the distance between the two nodes. Next, if all the nodes in m have same distance from the root, the procedure constructs a chain of m nodes. If, however, there is some node u such that $d[u, r]$ is less than any $d[j, r]$, we construct a chain of $m-1$ nodes, the execluded node is node u.

```
Procedure RG
Input: A query graph.
output: A reconstructed form of the input query.
begin

  if the graph has some node i
  that is connected to m nodes,
  m≤K_i, such that all the edges that connect
  the node i to m nodes have same label
  then do
    for each node u in m do
      calculate d[u, r],
      where r is the root of the graph;
    end do
    if all the m nodes have same d[u, r] then do
      construct a chain consisting of the m
      nodes;
      append this chain to node i;
      /* i will be of degree K_i-m+1*/
    end do
    else
      if among the m nodes there is some node
      u' with
      minimum distance from the root then do
        construct a chain consisting of the m
        -1 nodes;
        /* the execluded node is u'*/
        append this chain to node i;
        /* i will be of degree K_i-m+2*/
      end do
      end if
    end if
  end if
end
```

The node that will be the root of the chain of m nodes is selected as follows:

(1) If the graph is tree, it is the node with most strong join.

(2) If the graph is tree and the m nodes have same join strength, select the one with minimum degree.

(3) If the graph is cyclic, the root of the chain is the node that participates in the cycle, if such a node exists.

The rest of the nodes in the chain are connected according to the node's degree. The node with minimum degree is connected directly to the root, and the one with maximum degree is the leaf of the chain. For example, in Fig. 6(b) node B_4 is connected directly to node B_2 rather than node B_3. That is because B_4 has degree less than B_3.

3.3 Generating Nested Form for Tree Queries

In this section, we give a procedure to convert an unnested tree query into its equivalent nested form which is the most suitable for our purposes. Before giving this procedure, we give a procedure to handle chain queries.

A chain query is a simple tree query in which every node has WHERE-degree one and there is only one leaf with zero WHERE-degree. In this case, we do not need to apply any criteria to arrange query blocks.

In the chain graph, we number the nodes from 1 to n, such that the root is numbered 1 and the only leaf is numbered n. There is an edge between nodes i and $i+1$, $i=1, \cdots, n-1$. Procedure CQ accepts a chain query in the unnested form and converts it into the nested form. The query output is specified in the SELECT clause of the block corresponding to the first node in the chain. The SELECT clause of any node i contains attribute joining i with $i-1$. The FROM clause in the block corresponding to node i contains relation(s) in the block. The WHERE clause of any node i, $i=1, \cdots, n-1$, contains attribute joining i with $i+1$. The WHERE clause of the leaf node does not contain any nested predicate.

```
Procedure CQ
Input: A chain query in the unnested form and its query graph.
output: The same query in the nested form.
begin

  for i=1 to n do

    if i=1 then

      SELECT clause of i contains output
      attributes;
    else
      SELECT clause of i contains the attri-
      bute joining i with i-1;
    end if
    FROM clause of i contains name(s) of
    relation(s) in block i
```

```
    if i≠n then
      WHERE clause of i contains the attribute
      joining i with i+1;
    else
      i has no join predicate in the WHERE
      clause;
    end if
  end for
end
```

Before rewriting a tree query in the nested form, we have to reconstruct the tree according to the strength of joins. For each node i that has a WHERE-degree $k_i \geq 1$, we sort the k_i nodes such that the leftmost node in k_i is the one with most strong join with node i, and the rightmost node is the one with least strong join with i. After reconstructing the tree, procedure TQ is applied to rewrite the query.

While procedure TQ is preorderly traversing the tree[18], it writes the query corresponding to each node. If the current node is the root, its SELECT clause contains output attributes. Otherwise, it checks its NtR node, if it is the node itself, the SELECT clause of the node contains attribute joining the node with the root. If, however, the NtR is not the node itself, the SELECT clause of the node contains attribute joining the node with its NtR node. Next, the degree of the node is reduced by one to reflect the previous step. The FROM clause of a node contains name(s) of relation(s) in the block. The third step is to write the WHERE clause of the node. If the node is a leaf, i. e. its k_i is zero, its WHERE clause does not contain a nested predicate. For a non-leaf node, the WHERE clause contains $k_i - 1$ AND subclauses. In the WHERE clause, the join between the node and the node with the most strong join, among the k_i nodes, is written. AND clause number j, $j=1, \cdots, k_i - 1$, contains join of strength $j+1$. The final step is to write the whole query. Blocks are written consecutively according to the list generated from the preorder traversal of the tree.

Procedure TQ

Input: A tree query and its reconstructed query graph.

output: The same query in the nested form.

While traversing the tree in a preorder traversal, write the query following the next algorithm.

```
begin
j=1;
  for each node i do
    if i is the root then
      SELECT clause of i contains output
      attributes;
    else
      if NTR_i≠i then do
        SELECT clause of i contains the
        attribute joining i with NTR_i;
        k_i=k_i-1;
      end do
      else do
        SELECT clause of i contains the
        attribute joining i with the root;
        k_i=k_i-1;
      end do
      end if
    end if
    FROM clause of i contains name(s) of
    relation(s) in block i
    if k_i>0, that is i is not a leaf node then do
      WHERE clause of i has k_i-1 ANDs;
      WHERE clause of i contains the most
      strong join among k_i;
      repeat
        AND clause number j contains the join
        of strength j+1;
      until j=k_i-1;
    end do
    else
      node i has no nested predicate in its
      WHERE clause;
    end if
  end for
write the query blocks according to the list
produced from the preorder traversal of the
tree;
end
```

4. Conversion Procedure for Cyclic Queries

A cyclic graph consists of n nodes and m edges such that $m \geq n$. The nested form is more understandable than the unnested form, however, it cannot represent cyclic queries. Another form called *seminested* form can represent cyclic queries. This form is more understandable than

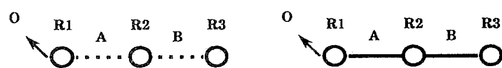
the unnested form. In the seminested form, all joins are written in the nested form except either one of the edges that constitute a cycle. This join is written as a join predicate.

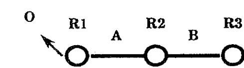
In a query graph, when an edge is dotted its corresponding join is written as a join predicate. When an edge is solid its corresponding join is written as a nested predicate. In a query graph, if some edges are dotted and some others are solid, its corresponding query is written in the seminested form. Figure 7 (a), (b), (c) show an example of a query written in unnested, nested, and seminested forms, respectively.

As in the case of tree queries, the criteria of join strength is used to select which join among those causing the cycle to be represented as a nested predicate and which as a join predicate. In some database systems, it is allowed to write a multiple attribute join in a WHERE clause, in this case we may combine (join) $n-1$ nodes in the cycle as a single node, the query becomes a tree query. When the number of nodes $n>3$, it is better not to use this transformation. The number of combined nodes is large and the query will not be easy to understand.

Before giving a procedure to handle cyclic queries, the following two definitions are required.

Definition 11: Given a connected cyclic graph $G_c(V_c, E_c)$, where V_c is the set of nodes and E_c is the set of edges. We define a *spanning tree* of





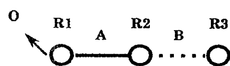

```
SELECT R1.O
FROM R1,R2,R3
WHERE R1.A = R2.A
AND R2.B=R3.B
```

(a)

```
SELECT R1.O
  FROM   R1
WHERE R1.A =
        SELECT R2.A
          FROM    R2
          WHERE R2.B =
             SELECT R3.B
             FROM    R3
```

(b)

```
SELECT R1.O
FROM   R1
WHERE R1.A =
   SELECT R2.A
   FROM   R2,R3
   WHERE  R2.B =R3.B
```



(c)

Fig. 7 A query written in the unnested, nested, and seminested forms.

the graph as the connected tree $T_s(V_s, E_s)$ such that $V_s=V_c$ and $E_s \subset E_c$.

Definition 12: Given a connected cyclic weighted graph, i. e. each edge is assigned certain weight. A spanning tree of the graph is called a *minimum spanning tree*, when the summation of the weights in tree is the minimum among the candidate spanning trees.

Procedure CYQ accepts a cyclic unnested query and generates it in the seminested form. First, if the underlaying SQL permits writing multiple attributes in a WHERE clause, and the number of nodes in the cycle is less than or equal three, the following two steps are done:
(1) Combine the output node with one of the nodes in the cycle. This node is the one that has the most strong join with the output node. Combining two nodes in a graph corresponds to joining relations represented by the nodes on the attribute labeling the edge between the two nodes. (2) Apply procedure TQ on the new graph.

Otherwise, it finds a minimum spanning tree of the graph. Edges in the tree are colored red and the remaining edges are colored blue. The minimum spanning tree is preorderly traversed. During the traversal process, for any node i which is connected by a blue edge to another node j, check whether node j has been already visited. If the node has been visited, write the join between relations i and j as a join predicate. This check is necessary to ensure that any relation which is referenced in a join predicate has been represented in a query block.

Procedure CYQ

Input: A cyclic query in the unnested form and its query graph.

output: The same query in the seminested form.

begin

if mutli-attribute join is allowed and the number of nodes in the cycle less than or equal three **then do**

Combine the output node with the one of the most strong join into one new node;
Apply procedure TQ on the new graph;

else do

```
SELECT R1.O
FROM   R1
WHERE  R1.A =
    SELECT R2.A
    FROM   R2
    WHERE  R2.D =
      SELECT R4.D
      FROM   R4
    AND    R2.C =
      SELECT R3.C
      FROM   R3
      WHERE  R3.B = R1.B
      AND    R3.F =
        SELECT R5.F
        FROM   R5
        WHERE  R5.E = R4.E
```

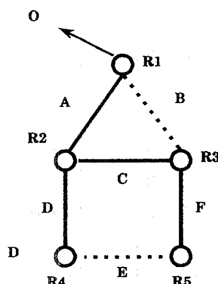


Fig. 8 A cyclic query written in the seminested form.

Preprocessing steps :

(1) Determine a minimum spanning tree and color all edges in the tree red and the remaining edges blue ;
(2) Reconstruct the graph as done in case of tree queries to reflect strength of joins ;

End Preprocessing steps :
while traversing the minimum spanning tree preorderly **do**
  Create a list of all nodes traversed so far ;
  if the current node *i* is connected to another node *j* with a blue edge **then do**

    if the relation corresponding to node *j* is in the list of traversed nodes **then** write the join between relations *i* and *j* as a join predicate ;
    end if
  end do
  end if
end do
end

Example : Figure 8 shows a cyclic query graph. The graph has two cycles. The first consists of nodes R1, R2, and R3, the second cycle consists of nodes R2, R4, R5, and R3. (We consider that writing multi-attribute join in one WHERE clause is not permitted.) The first step is to determine a minimum spanning tree. Assume that edges in the minimum spanning tree are the edges labelled A, D, C, and F. The other remaining edges B and E are execluded from the tree. Joins represented by the edges of the tree are written as nested predicates and other joins are represented as join predicates.

5. Queries with Multiple Output Relations

The nested form is more understandable, however, some queries may not be written in this form. If the query output is obtained from several relations, the query cannot be written in the nested form. If the query is simple, it may be understandable when written in the unnested form. In this section, we give two preprocessing procedures to convert graphs with multiple output nodes into equivalent graphs with only one output node. The first procedure TMO, handles tree graphs and the second, CMO, handles cyclic graphs.

First we describe TMO. If the underlying SQL permits writing multiple joins in a single join predicate, output nodes are combined into a single node. Output relations are combined by taking their Cartesian product. If it is not permitted to write multiple joins in one join predicate, combines output nodes along with any node in the path connecting two output nodes. In this case, relations corresponding to these nodes are joined. Next, the graph is reconstructed such that the new output node becomes its root node. For any node that was connected to any of the combined nodes, there will be an edge between this node and the new root having same label as before.

Procedure TMO
Input : A tree query graph with multiple output nodes
output : An equivalent graph with a single output node
begin

  /* Define JOIN as a store of nodes to be combined together. */
  if multiple attribute join is allowed **then**
    Combine output nodes into one node ;
    /* Combined in the query by taking their Cartisian product */
  else
    for each node *i* **do**
      if *i* in the path between any two output nodes **then**
        add *i* to JOIN ;
      end do
      constitute a relation that corresponds to the output node R_o as $R_o = \bowtie R_i$, *i*

$=1 \cdots m$,
where m is the number of relations in JOIN;
/* reconstruct the graph with R_o as the root */
end if
for each node j that was connected to any node in JOIN **do**
connect this node to the new root with an edge havig same label as before;
end do
end

Example: Consider the query graph shown in Fig. 9(a), applying procedure TMO generates the graph shown in Fig. 9(b). The node $B*_1$ corresponds to joining relations in blocks B_1, B_2, and B_4.

In case of tree queries, we have only one path to combine output nodes. In case of cyclic queries, however, if the path that connects the output nodes contains at least two nodes that are members in a cycle, we may have two ways to combine output nodes. We have the following criteria to select which set of nodes to combine:
(1) If the paths connecting output nodes have same length, apply the criteria of join strength.
(2) If the paths connecting output nodes have different lengths, select the path that contains minimum number of nodes to combine. In this case there is a tradeoff between selecting the path that contains minimum number of nodes to combine and writing query blocks according to strength of joins. When considering join strength, we may have to select large number of nodes to combine. We decided to combine minimum number of nodes sacrificing strength of joins in order to generate nestings deeply as possible.

To select such a path, we generate a new graph from the original one. Nodes in the new graph are output nodes in the original graph. Each path that connects any two output nodes i, j in the original graph is represented by a labeled edge between i, j in the new graph. The label of an edge connecting nodes u, v in the new graph is the distance between these two nodes in the original graph. For the new graph, we find the minimum spanning tree. This tree corresponds to the path that contain minimum number of nodes to combine.

Example: The graph in Fig. 10(a) is a cyclic graph with multiple output nodes. The graph in Fig. 10(b) shows the new generated graph. Solid edges in the new graph represent the minimum distance spanning tree. Figure 10(c) shows the original graph with only one output

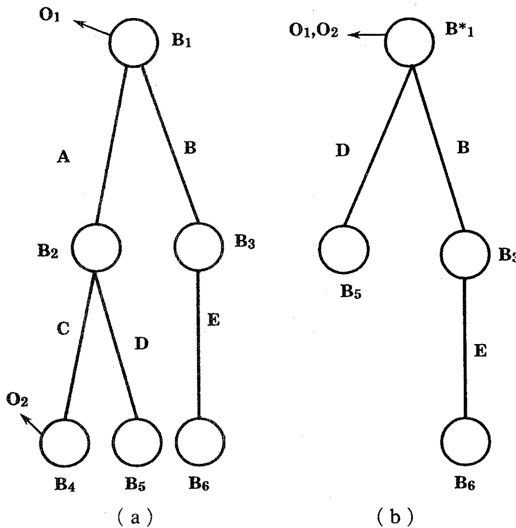


Fig. 9 A tree graph with multiple output nodes and its simplification.

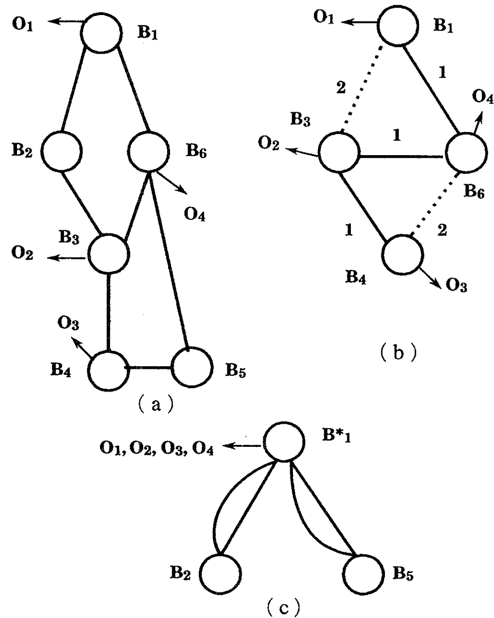


Fig. 10 Cyclic query graph with multiple output nodes and its simplification.

node $B*_1$, which is the combination of nodes B_1, B_6, B_3, and B_4.

Procedure CMO

Input: A cyclic query graph with multiple output nodes

output: An equivalent graph with a single output node

Condition:

The output node in the final graph is the combination of nodes in the input graph.

Method:

1- Build a new graph $G'(V', E')$ from the input graph as follows:
V′ is the set of output nodes in the original graph.
The set of edges E′ is constructed as follows:
for any two nodes i, j, if there is a path in the original graph is connecting these nodes,
it is represented by an edge between i, j in the new graph.
The label of this edge is distance between i, j in the original graph.
Associated with each edge an array that contains nodes in the original path between the two nodes.

2- Find the minimum spanning tree of the new graph.

3- Combine all the nodes in the arrays associated with edges in the minimum spanning tree into one node, say $B*_1$.

4- Build the output graph with node $B*_1$ as its output node.
For any node u, in the original graph, which is connected by an edge labelled l to any node in the combined nodes,
connect u to $B*_1$ via an edge with label l.

end

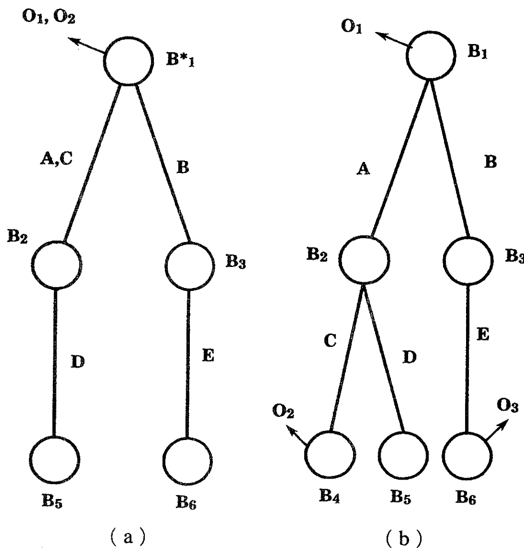


Fig. 11 Case when multiple attributes join is possible.

If the underlying SQL permits writing more than one join attribute in the WHERE clause, we can combine output nodes only. The combination is their Cartesian product. The graph shown in Fig. 9(a) is reconstructed as the graph shown in Fig. 11(a). The query block which corresponds to the node $B*_1$ in Fig. 11 (a) contains the Cartesian product of relations in blocks B_1 and B_4.

In the graph shown in Fig. 11(b), we need to combine output nodes in different paths, that is we need to combine nodes B_1, B_2, B_3, B_4, and B_6. If multi-attribute join is permitted we only need to combine nodes B_1, B_4, and B_6.

6. Conclusion

In this paper, we presented procedures to generate nested SQL queries for documentation and query reuse and editing. An unnested query is transformed to its equivalent nested one. Tree and cyclic queries are considered. For a tree or a cyclic query, there may be more than one nested form. We gave criteria to select the order of writing query blocks. These criteria are based on determining join strength. We also presented procedures to handle queries when the output is obtained from several blocks.

References

1) Codd, E. F.: A Relational Model for Large Shared Data Banks, *Comm. ACM*, Vol. 13, No. 6, pp. 377-387 (1970).
2) Astrahan, M. M. et al.: System R: Relational Approach to Database Management, *ACM Transactions on Database Systems*, Vol. 1, No. 2, pp. 97-137 (1976).
3) Chamberlin, D. D. et al.: SEQUEL2: A

Unified Approach to Data Definition, Manipulation, and Control, *IBM J. Res. Dev.*, Vol. 20, No. 6, pp. 560-575 (1976).
4) Unify Corporation: Unix Relational Data Base Management System, Reference Manual (1984).
5) Zlcof, M.: Query-by-Example: A Database Language, *IBM Systems J.*, Vol. 16, No. 6, pp. 324-343 (1977).
6) Kambayashi, Y.: An Overview of a Natural Language Assisted Database User Interface: ENLI, *Proc. IFIP*, pp. 1055-1060, Sept. (1986).
7) Kambayashi, Y. and Amano, H.: Transformations of Natural Language Expressions by Basic Relational Database Operations, *Trans. IPSJ* Vol. 30, No. 10, pp. 1316-1322 (1989) (in Japanese).
8) Kambayashi, Y.: Functions of the Database Workbench, *Proc. NCC*, pp. 547-553 (July 1984).
9) Date, C. J.: *A Guide to the SQL Standards*, 2nd Ed., Addison-Wesley (1989).
10) Bernstein, P. and Chiu, W.: Using Semi-Joins to Solve Relational Queries, *J. ACM*, Vol. 28, No. 1, pp. 25-40 (1981).
11) Kambayashi, Y.: Processing Cyclic Queries, *Query Processing in Database Systems*, Kim, W., David, S. and Don Batory, S. (eds.), Springer-Verlag (1985).
12) Ceri, S. and Gottlob, G.: Translating SQL into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries, *IEEE Software Engineering*, Vol. SE-11, No. 4, pp. 324-345 (1985).
13) Kim, W.: On Optimizing an SQL-like Nested Query, *ACM Transactions on Database Systems*, Vol. 7, No. 3, pp. 443-469 (1982).
14) El-Sharkawi, M. and Kambayashi, Y.: Efficient Processing of Distributed Set Queries, *Databases: Theory, Design, and Applications*, Rishe, N., Navathe, S. and Tal, K. (eds.), IEEE Computer Press. (1991).
15) Luk, W. S. and Kloster, S.: ELFS: English Language from SQL, *ACM Transactions on Database Systems*, Vol. 11, No. 4, pp. 447-472 (1986).
16) Amano, H. and Kambayashi, Y.: Translation of SQL Queries Containing Nested Predicates into Pseudonatural Language, *Proc. of DASFAA*, pp. 116-125 (1991).
17) Date, C. J. : *An Introduction to Database Systems*, 3rd Ed., Addison-Wesley (1981).
18) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

(Received July 3, 1989)
(Accepted September 9, 1993)

Mohamed E. El-Sharkawi received the B. Sc. degree in Computer and Systems Engineering from Al-Azhar University, Cairo, Egypt, M. Eng. and D. Eng. degrees from Kyushu University, Fukuoka, Japan, in 1981, 1988, and 1991, respectively. He spent one year as a researcher in the Advanced Software Technology and Mechatronics Research Institute of Kyoto (ASTEM/Kyoto). He was also a visiting researcher at the Integraed Media Environment Experimental Laborator at Kyoto University. Currently, he is Assistant Professor at the Department of Mathematics, Kuwait University. His research interests are in the areas of object-oriented databases and computer supported cooperative work.

Yahiko Kambayashi received the B. E., M. E., and Ph. D. degrees in electronic engineering from Kyoto University, Kyoto, Japan, in 1965, 1967, and 1970, respectively. During 1970-1971 he was a Research Associate at Kyoto University. From 1971 to 1973 he was a Visiting Research Associate at the University of Illinois, Urbana, U. S. A. During 1973-1984 he was with the Department of Information Science, Kyoto University. In 1984 he became a Professor at the Department of Computer Science and Comunication Engineering, Kyushu University, Fukuoka, Japan. Since 1990 he has been a Professor at Integrated Media Environment Experimental Laboratory of Kyoto University. In 1979, he was a Visiting Professor at McGill University, Montreal, Canada, and in 1984 he was a Visiting Professor at Wuhan University, Wuhan, China. His research interests include logic design and database theory. Dr. Kambayashi was a Chairman of SIGDBS (database systems) of the IPSJ, a Chairman of SIGCOMP (computation theory) of the IIEIEJ. He was also a member of the boad of IPSJ. He is currently a vice-chair of Japan Section of ACM.