データベース・システム 64－9
（１９８８. ３. １５）

ドキュメンテーションのための入れ子SQL質問の生成

モハメッド　エルシルケウイ　　上林弥彦

九州大学工学部

データベース質問のドキュメンテーションは利用者インタフェースやデータベースワークベンチなど利用者を援助するシステムの実現に必要である。本稿ではドキュメンテーションに適したSQL質問の構造の生成問題について議論する。SQLでは複雑な質問を入れ子構造などの理解しやすい形で書くことができる。しかし、入れ子構造で表現できるるのは非巡回質問のみであり、巡回質問に対しては入れ子を用いない構造より理解しやすい準入れ子と呼ばれる構造を生成する。木質問や巡回質問を入れ子や準入れ子の構造で記述する方法は一意ではない。ドキュメンテーションに最も適した構造を選ぶため“結合の強さ”の概念を用いる。この概念は生成された構造の質問ブロックの順序変更に用いる。結合の強さが強いほど質問ブロックが近くなる。

Generating Nested SQL Queries for Documentation

Mohamed El-Sharkawi Yahiko Kambayashi

Dept. of Computer Sci. and Communication Eng.

Kyushu University

Documenting database queries is necessary to realize database users' aid systems, like user interfaces and database workbench. In this paper, we consider the problem of generating forms of SQL queries suitable for documentation. In SQL, a complex query can be written in an easy to understand form. It can be written in the nested form, i.e. as simple query blocks connected together. In this paper, we discuss how to convert a given query to a nested form suitable for documentation. The nested form, however, can represent only acyclic queries. For cyclic queries, we generate a form called seminested form which more understandable than the unnested form. There may be more than one way to write a tree or cyclic query in the nested or the seminested form. To select among these alternatives the most suitable form for documentation, we use the notion of "Strength of Joins". This notion is used to arrange query blocks in the generated form. The stronger the join between two relations, the closer their corresponding query blocks.

1- Introduction

Because of the advantages of the relational model of databases [4], we find computerized applications that use DBMSs based on that model. One of the important phases to realize a computerized application is the documentation phase. System design and system operation have to be well documented to help understanding, evaluating, and modifying the system. Therefore, it is also necessary to consider documenting database queries. Documenting database queries is not only necessary in such systems, but also to build users aid systems, such as user interfaces [7] and database workbench [8]. In this paper, we consider the problem of documenting queries. We discuss the case when queries are written in SQL (Structured Query Language) [1,3]. SQL is, now, considered as a standard relational query language. In SQL, it is easy to write and understand complex queries. Using the notion of nested queries, i.e. a set of simple queries nested together, a complex query may be written in understandable form (see Fig.1). Sometimes, the query submitted to the system is not suitable for documentation, hence, we transform the given SQL query into another, semantically equivalent, form which is suitable for documentation. Our approach to generate SQL queries for documentation is based on transforming an unnested query into the nested form which is suitable for documentation. If the given query is nested we first transform it into its equivalent unnested one, since the nested form given by a user may not be the best for documentation purpose. Transforming an SQL query from one form to another was also presented in [10], to optimize processing of nested SQL-like queries. A nested query is transformed into an unnested one to exploit the existing query optimizer, which is well suited for optimizing unnested queries.

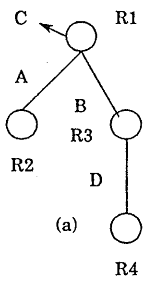
2- Basic Concepts

2-1 SQL Syntax

In this section we describe the syntax of SQL that is relevant to our discussions. Full description of SQL can be found in [1,3,5]. In SQL the basic query consists of three clauses: SELECT, FROM, and WHERE. These three clauses constitute a query block. The SELECT clause enumerates output attributes. FROM clause contains names of relations involved in the query. SELECT and FROM clauses are mandatory to write a query. WHERE clause, however, is optional. It contains condition that must be satisfied by the query. A condition may contain one of the comparison operators $=,\neq,<,\leqq,>,\geqq$. If there is more than one condition to be satisfied, they may be combined by logical operations AND, OR, and NOT. Beside those comparison operators, there are also set operators. In the WHERE clause predicates, as relevant to our discussions, to be satisfied may be one of the following:

(a) A simple predicate, has the form:
Attribute <Comparison operator> Constant value.
(b) A join predicate, has the form:
Attribute 1 <Comparison operator> Attribute 2.
(c) A nested predicate, has the form:
Attribute/Constant value <Comparison operator> Query block.

To write a join query in SQL there are two forms, the unnested and the nested form. In the unnested form the query is written as a single query block in which the FROM clause contains names of relations in the query, and joins are represented as join predicates. In the nested form, joins are represented as nested predicates. The nested form simplifies writing complex queries. A complex query may be written as a set of nested query blocks using nested predicates. Fig.1 (a), (b), and (c) show a query graph, its corresponding query in the unnested and nested forms, respectively. We define SQL forms as follows.



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

(b)

```
SELECT R1.C
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.1 SQL query represented in unnested and nested forms

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

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

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

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

2-2 Query Graphs

To formalize our discussions, we use query graphs to express queries. We need the following definitions.

Definition 4: A query graph G=(N,E) is a labeled undirected graph, where 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 relation. There is an edge between nodes i and j iff a relation in the block represented by node i is joined with another relation in the block represented by node j. Each edge e_{ij} is labeled with the attribute that is joining blocks i and j. If there is more than one attribute that join the two blocks, there will be an edge corresponds to each attribute. Let n be the number of nodes and k_i be the degree of node i (the number of edges connected to node i). Among the N nodes one is distinguished as the root of the graph. The root is considered as the node that corresponds to the block from which the output is obtained, this node will be called output node. If there is more than one output node, one is selected as the root.

We define three types of query graphs, chain, tree, and cyclic query graphs [2]. A query has the same type of its corresponding query graph.

Definition 5: A connected graph is a tree graph if it has n nodes and n-1 edges. That is the graph is circuit-free. A node that is of degree 1, is called a leaf node. When n is greater than 2, one node is selected as a root. Other nodes are called inner nodes.

Definition 6: A query graph is a chain graph, if it is a tree graph such that, there is only two nodes with degree 1 and every other node satisfies $k_i = 2$.

(Note that, one of the leaf is the root.)

Definition 7: A query graph is cyclic if it is not a tree graph.

Definition 8: The distance between two nodes i, j, d[i,j], is defined as the number of edges in the path that is connecting the two nodes.

(Note that, for a cyclic graph, it may exist two nodes u, v, such that there are more than one d[u,v].)

Throughout the paper we assume that every graph is connected.

Query graphs that are corresponding to chain, tree, and cyclic queries are given in Fig. 2(a), (b), and (c) respectively.

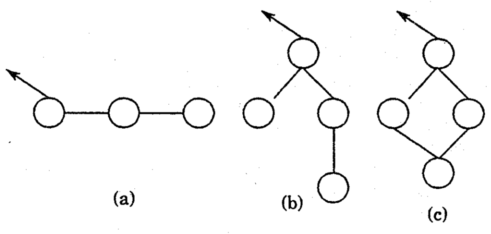


Fig.2 Query graphs for chain, tree, and cyclic queries

3- Basic Conversion Procedures

In this section we will develop procedures for tree queries which will be generalized in the following section. We assume that there is only one block from which the query output is obtained, generalization will be given in Section 5.

3-1 Strength of Joins

Although the join structure of a query can be as a tree or a cyclic graph, SQL representation requires one dimensional representation. Thus it is necessary to determine the order of query blocks. It is reasonable that two blocks which are joined strongly are adjacently placed. Thus, we need conditions of strength of joins. For example, in a tree graph some inner nodes may have degree greater than 2. Query block corresponds to such a node i has a WHERE clause with k_i-1 ANDs, thus we need to decide which join will be represented in the WHERE clause, which in the first AND, and so on. Here, we present selection criteria by which the order of joins is determined. The most strongest join is written in the WHERE clause, and the join with least strength is written in the last AND. These criteria are based on perceiving the database relations as representing real-world entities and relationships between those entities; we may think of a join between two relations as a way to describe connections between real-world components (i.e. entities and relationships). Some joins may represent strong connections or even collect information, about an entity, distributed among several relations. All edges, then, are assigned weights represent the join strength, the minimum the strong. Criteria to decide strength of joins are as follows:

(1) For some node i of degree k_i, $k_i > 1$, if there is one and only one join such that the join attribute is the key of that node i (a key of a node is a key of one of the relations in the block represented by that node), we consider that join as the strongest one. If all the joins are keys of relations in the block represented by the node, the one which is key of a relation from which the output of the block is obtained is considered as the strongest join. The rationale is, a key of a relation is used to identify an entity or a relationship in the world, thus joining two relations by a key of one of them is considered as collecting together information about that entity. This information, due to requirements of database deign, is distributed among several relations.

(2) If there is more than one join such that the joining attributes are alternative keys of the node i and one among them is in the output of the block, we consider that one as the most important join. Since that key is playing an important role, and if it is the query output, the user is interested in information concerning it.

(3) If there is no join attribute, among all the joins, which is a key of the node i, and, however, one among them is a key of another node, then that one is represented as the strongest join. If there are more than one, we select the one whose block is of minimum degree.

(4) Consider a case where there is a node i need to be joined with k_i nodes and the node has strongest 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 far from the block of node i. In this case, we combine the two nodes, i and j, into one and then write the query. Fig. 3 shows an example. The join between relations in block B_1 and relations in block B_2, in Fig. 3(a), is

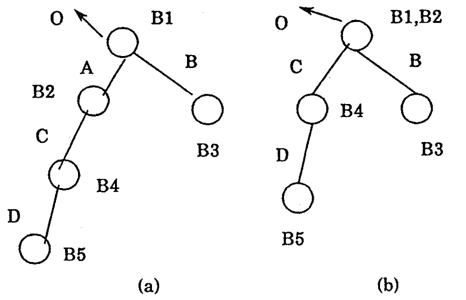


Fig. 3 Case when the strongest join is in the longest path

strongest than that between B_1 and B_3. According to previous criteria this join is selected to be represented first. However, the path from the root node to the leaf node that contains B_2 is longer than that containing B_3. It may be better to combine nodes B_1 and B_2 together as in Fig.3(b). After combining the two nodes, it is not needed to select among B_3 and B_4, since B_3 is considered strong relative to the original output node.

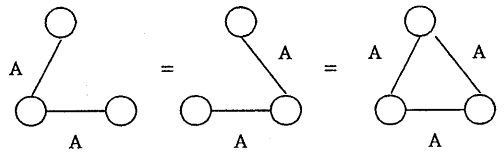


Fig. 4 Equivalent query graphs

(5) If no of the join attributes satisfies the above criteria, then all joins are considered have the same strength, we consider the subtree with minimum weight as the most strongest one and represent it before the others.

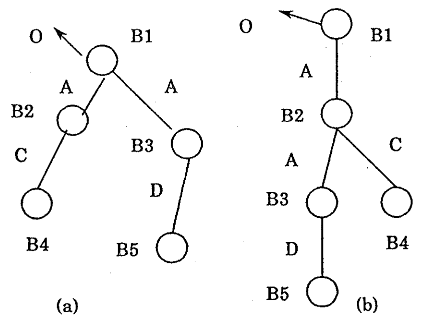


Fig. 5 Two equivalent tree graphs

3-2 Equivalence Transformation of Query Graphs

When p relations are joined by identical attribute we can apply the equivalence transformation shown in Fig. 4. We use such a transformation as a preprocessing step to convert a graph into its equivalent one that will be written in deep level of nesting. For example, consider the query graph given in Fig. 5(a), it may be converted into the graph in Fig. 5(b). This situation can be stated as follows: Given a query having some node i of degree k_i. Among the k_i nodes there are m *nodes* such that all the edges that connect the m nodes to node i have same label, L. Then we can use the previously mentioned transformation to convert the graph to another equivalent one that will be written in a deeper level of nesting than the original one. We first calculate for each node j in m d[j,r], where r is the root node. If 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 excluded node is node u. If, however, all the nodes in m have same distance from the root, we construct a chain of m nodes.

The following procedure, RG, does the job. We assume that there is a procedure that accepts a graph, two nodes m and n, and calculates d[m,n].

```
Procedure RG
Input : A query graph.
Output : A reconstructed form of the input query.
begin
  if the graph has some node $i$ having $m$, $m \leqq k_i$,
  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 for 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 excluded 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 cyclic, it is the node that participate in the cycle.

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

3-3 Converting Tree Queries for Documentation

Before we give the procedure to convert an unnested tree query into its equivalent nested one, we give procedure to handle chain queries. A chain query is a simple case of tree queries. In this case it is not needed to apply the selection criteria, any node is at most of degree two. 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 $i+1$, $i=1,...,n$-1.

Procedure CQ

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

Output: The same query written 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 attributes joining $i$ with $i$-1;
    end if
  FROM clause of $i$ contains name of relation $R_i$;
    if $i \neq n$ then
      WHERE clause of $i$ contains the attributes 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 has $k_i \geqq 1$, we sort the k_i nodes such that the leftmost to be the most strong join and the rightmost be the least strongest one. After reconstructing the tree we apply the following procedure to rewrite the query. We need the following definition.

Definition 9: For each node i which is connected to C_i nodes, we define among these C_i nodes a node NtR_i (Nearest to Root) as that node with minimum distance to the root. For some nodes $NtR_i = i$.

Procedure TQ

Input: A tree query written in the unnested form with its reconstructed graph.

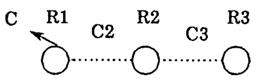
Output: The same query written in the nested form

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 $NtR_i \neq i$ then do
        SELECT clause of $i$ contains attribute joining $i$ with $NtR_i$;
        $ki=ki$-1;
        end do
      else SELECT clause of $i$ contains
          attribute joining $i$ with the root;
      end if
  FROM clause of $i$ contains name of relation $R_i$;
  if $i$ is not a leaf node then do
    WHERE clause of $i$ has $k_i$-1 ANDs;
    WHERE clause of $i$ contains the strongest 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 join predicate in the WHERE clause
  end if
 end for
end
```

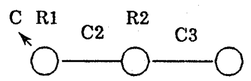
4- Conversion Procedure for General Cyclic Queries

A cyclic graph consists of n nodes and m edges, such that $m \geqq n$. While the nested form of SQL is very much understandable, it cannot represent cyclic queries. The seminested form, however, can represent cyclic queries, and is easier to understand than the unnested form. In this form, all the joins between relations are written in the nested form, but either one of those whose edges constitute the cycle. In the query graph, if all the edges are dotted, the query will be written in the unnested form, and if all the edges are solid, the query will be written in the nested form. If some edges are dashed and the others are solid the query will be written in the seminested form. That is, relations connected by dotted lines are merged into a block. Fig.6 (a), (b), and (c) show an example of a query written in unnested , nested, and seminested forms, respectively. As in case of tree q



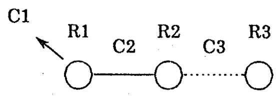
```
SELECT $R_1.C_1$
FROM $R_1,R_2,R_3$
WHERE $R_1.C_2 = R_2.C_2$
AND   $R_2.C_3 = R_3.C_3$
```

(a)



```
SELECT $R_1.C_1$
FROM $R_1$
WHERE $R_1.C_2$ =
    SELECT $R_2.C_2$
    FROM $R_2$
    WHERE $R_2.C_3$ =
        SELECT $R_3.C_3$
        FROM $R_3$
```

(b)



```
SELECT $R_1.C_1$
FROM $R_1$
WHERE $R_1.C_2$ =
    SELECT $R_2.C_2$
    FROM $R_2,R_3$
    WHERE $R_2.C_3 = R_3.C_3$
```

(c)

Fig.6 A query written in the unnested, nested, and seminested forms

ueries, we need to select which join among those causing the cycle to be represented in the nested form and which in the unnested form. We use same criteria as in tree queries. When multiple attributes join is allowed we may combine n-1 nodes in the cycle into one node, and then the query becomes a tree one. When $n > 3$, it is better to use the approach used when multiple attributes join is not permitted, since the number of nodes needed to be combined is large making the query not easy to understand.

In this context, we need to define a spanning tree of a cyclic graph.

Definition 10: 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 that graph as the connected tree $T_S(V_S,E_S)$ such that $V_S = V_c$ and $E_S \subsetneq E_c$.

Definition 11: Given a connected weighted cyclic graph, i.e. a graph such that each edge is assigned certain weight. A spanning tree of the graph is called *minimum spanning tree*, when summation of weights in the tree is the minimum among all candidate spanning trees.

```
Select O
From R1
Where R1.A =
    Select R2.A
    From R2
    Where R2.B =
      Select R3.B
      From R3
    And R2.C =
      Select R4.C
      From R4
      Where R4.D =
        Select R5.D
        From R5
        Where R5.G = R1.G
```

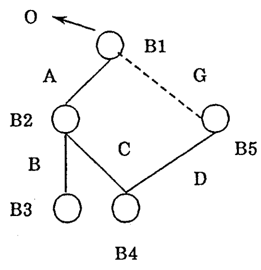


Fig.7 A cyclic query written in the seminested form

It is known that there is an efficient procedure to obtain a minimum spanning tree for a given graph.

Procedure CYQ

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

Output: The same query written in the seminested form

```
begin
  If multiattributes join is allowed and the nodes in
  the cycle ≦ 3 then do
    Combine the output node with that one of the
    strongest join into one new node;
    Apply procedure TQ;
  else do
    Preprocessing step: In the graph determine
    a minimum spanning tree that has all edges
    are the strong joins and color all edges in the
    tree red and the rest of edges blue.
      For each node i do
        if i is not connected by blue edge to any
        node then
          use procedure TQ to write its
          corresponding query block;
        else do
          write the join corresponding to the blue
          edge in the unnested form;
          write the rest of joins associated with i
          using procedure TQ;
        end
        end if
      end
    end if
end
```

For example of writing a cyclic query in the seminested form consider the query graph shown in Fig.7. The solid edges represent the minimum spanning tree. The dashed edges represent joins that are written in the unnested form.

In the previous discussion, the cycles in the graph are not related to each other. In some cases, however not common, a user may write a complex query that contains more than one cycle such that some of them have a common component, i.e. a common node or a common edge. Fig. 8 show examples of graphs with common components. . There are four cases of cycles

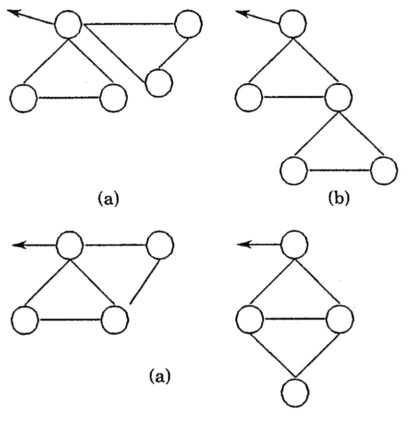


Fig.8 Cycles with common components

with a common component:

case (1): two cycles with a common node which is the output of the both cycles, Fig.8 (a).

case (2): two cycles with a common node which is not the output of the both cycles, Fig.8 (b).

case (3): two cycles with a common edge which contains the output node, Fig.8 (c).

case (4): two cycles with a common edge which does not contain the output node, Fig.8 (d).

Next, we discuss how to handle each case.

Case (1): We calculate the weight of each cycle, and write the query such that the cycle with minimum weight is written in the WHERE clause of the block that contains the output node. The second cycle is written in the AND subclause. Note that, within each cycle we apply procedure CYQ.

Case (2): Let us name the cycle that contains the output node as the main cycle and the other cycle

that contains the common node as the secondary cycle. We start writing the cycles by writing the main cycle, and when we are in the position to write the common node; we have the following two choices:
(a) Write the secondary cycle before completing the main cycle.
(b) Write the secondary cycle after completing the main cycle.
The first choice arises when the join between the common node and one of the nodes in the secondary cycle is strongest than that join between the common node and a node in the main cycle. The second choice arises when the only join required to complete writing the main cycle is between the common node and the output node; or the join between the common node and a node in the main cycle is strongest than that joins between the common node and any node of the nodes in the secondary cycle.
Case (3): We combine the two nodes connected by the common edge to have a graph as the one in case (1).
Case (4): We combine the two nodes connected by the common edge to generate a graph similar to that one in case (2).

5- Queries with Multiple Output Relations

From the definition of SQL, it is not possible to write a query that has the output is obtained from more than one block in the nested form, if the query is simple, it is easy to understand such a query written in the unnested form. We give two preprocessing procedures that convert a graph with multiple output nodes into another one with only one output node. The first procedure, TMO, handles tree queries and the second, CMO, handles cyclic queries. Before describing procedure TMO, we explain the idea through an example.

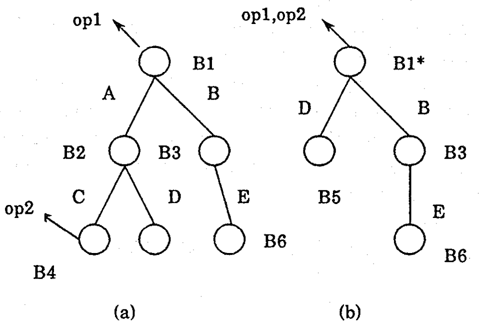


Fig. 9 A tree query graph with multiple output nodes and its simplification

Consider the query graph in Fig. 9(a), applying procedure TMO produces the graph in Fig. 9(b). The node B^*_1 corresponds to joining relations in blocks B_1, B_2, B_4.

Procedure TMO

Input: A tree query graph with multiple output nodes.
Output: A corresponding graph with only one output node.

```
begin
  if multiple attributes join is allowed then
      Combine output nodes into one node;
  else
          for each node $i$ do;
            if $i$ is in the path between any two
            output nodes then add $i$ to JOIN;
          end do
            constitute a relation that
            corresponds to the output node $R_0$ as
            follows:
            $R_0 = \infty R_i$, $i=1,\dots,m$, where $m$ is the
            number of relations in JOIN;
            /*reconstruct the graph with $R_0$ as
            the root node*/
        end if
  end if
  for   each node $j$ such that $j$ was connected to
        any one of the combined nodes do;
         connect that node to the new node with
        an edge having same label as before;
        end do
end
```

In case of tree queries, we have only one way to combine output relations together. In case of cyclic queries, however, we may have two ways to combine the output nodes if the path connecting the two output nodes contains a node which is a member in a cycle. We have the following criteria to select which set of nodes to be combined together:
(1) Select the path that contains minimum number of nodes need to be combined to have a graph with only one output node. To select such path, we generate from the graph another new graph. In which nodes are the 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 nodes 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 generated graph we find the minimum spanning tree, which is the desired path that contains minimum number of nodes need to be combined to have a graph with only one output node. For example, consider the graph given in Fig. 10(a) which is a cyclic graph with multiple output nodes, Fig.10(b), shows the new generated graph where edges represented by solid lines is the minimum spanning tree. Fig. 10(c) shows the original graph with only one output node B^*_1, which is the combination of nodes B1, B_6, B_3, and B_4. The procedure doing this conversion is as follows:

Procedure CMO

Input: A cyclic graph with multiple output nodes.
Output: An equivalent graph with only one output node.
Condition: The output node in the output graph is the merge of minimum number of nodes in the input graph.

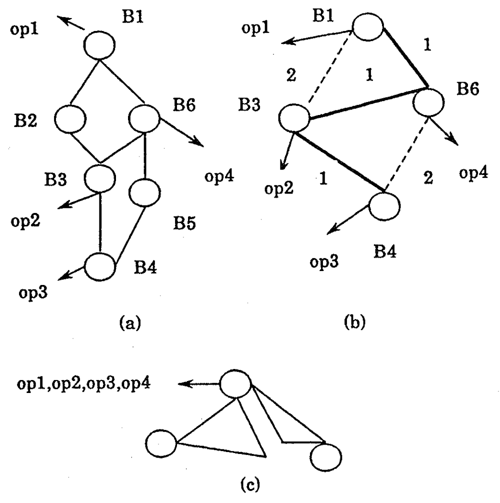


Fig. 10 A cyclic query graph with multiple output nodes

Method:

1- Build a new graph G'(V',E') from the input graph where, 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 path, in the original graph, connecting these two nodes, it is represented by an edge between *i*, *j* in the new graph. The label of this edge is the distance between the two nodes *i*, *j* in the original graph. Associated with each edge an array contains nodes in the original path represented by this edge.

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

3- Combine all the nodes in 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 the only output node. For any node *u*, in the original graph, that is connected to any node in the combined nodes, by an edge labeled *l'*. Connect the node *u* with B^*_1 via an edge with label *l'*.

end

(2) If the paths connecting output nodes have same

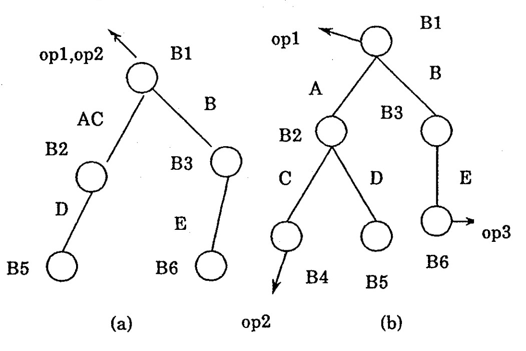


Fig. 11 Case when multiple attribute join is possible

lengths. Apply the criteria of selecting the strongest joins.

In the previous discussions, if any two blocks are joined with more than one attribute, for each join attribute there was an edge labeled with the attribute name, connecting nodes correspond to those blocks. This is done due to an implication in SQL syntax. That is, in the nested form, the WHERE clause of the outer block and the SELECT clause of the inner block, contain at most one attribute. In some new implementations of SQL this restriction is released. Typical systems are SQL/RT of IBM and UNIFY databases. Releasing that restriction permits combining output nodes only not necessarily intermediate nodes that are connecting them. Hence, the graph in Fig. 9(a) is reconstructed as shown in Fig. 11(a), where the block represented by node B^*_1 corresponds to the Cartesian product of relations in blocks B_1 and B_4. This is, also, very suitable when there are more than two output nodes and to combine them it is necessary to combine nodes in different paths, Fig.11 (b) shows an example. In this case, we have to combine nodes B_1, B_2, B_3, B_4, and B6 if multiattributes join is not permitted, while we need to combine nodes B_1, B_4, and B_6 if it is permitted.

6- Conclusion

In this paper, we suggested procedures to generate SQL queries for documentation. For an unnested acyclic query, we generate its corresponding nested one which is suitable for documentation. For cyclic queries, that cannot be expressed by the nested form, an unnested cyclic query is converted into another one that is suitable for documentation.

References

[1] Astrahan,M., et al., ACMTODS, Vol.1, No.2, pp.97-137.
[2] Bernstein,P., Chiu,W., JACM, Vol.28, No.1, pp.25-40.
[3] Chamberlin,D., et al., IBM J. Res. Dev., pp.560-575.
[4] Codd,E.F., Comm. ACM, Vol.13, No.6, pp.377-387.
[5] Date,C.J., 3rd edition, Addison-Wesely, 81.
[6] Goodman,N., Shmueli,O., ACMTODS, Vol.7, No.4, pp.653-677.
[7] Kambayashi,Y., Proc. IFIP, pp.1055-1060, Sept.1986.
[8] Kambayashi,Y., Proc. NCC, pp.547-553, July 1984.
[9] Kambayashi,Y., Yoshikawa,M., Yajima,S., Proc. ACM SIGMOD, June 1982, pp.151-160.
[10] Kim,W., ACMTODS, Vol.7, No.3, pp.443-469.