12

On Reducing the Set of Feasible Query Trees in Distributed Query Processing Optimization

1.0 INTRODUCTION

The result of a query in a relational database can be expressed by operations with series and parallel relationship. These relationships can be depicted by graphs called query trees [HEVNEA1]. Given the permutability properties of relational operators, in most non trivial cases there will be more than one query tree that produces the correct answer. Furthermore, if the query contains a large number of operations there will be many query trees, herein called the set of feasible query trees, that produce the correct answer. Examining them all take up time and storage, and is therefore undesirable. This paper focuses on identifying rules that will reduce the total number of possible feasible query trees used in determining query processing policies for distributed database systems as proposed in [CHUW1].

In distributed database systems we are generally interested in determining the best policy for processing a particular query or class of queries. A number of query optimization algorithms, that derive optimal distribution strategies for queries have recently been reported in the literature.

The first algorithm for distributed query processing [WONGE1] translated the query into a sequence of move and process operations. The algorithm moves all the relations referenced by the query to an initial site, selected a priori, for processing. The final distribution strategy is arrived at by improving upon this initial solu-

tion with a sequence of move and process commands until the lower cost sequence is found. A number of refinements to the algorithm followed, notably those incorporating the notion of reducer and the semijoin operator [BERNSP1]. They avoided the need for an initial a priori processing site and enhanced both the basic algorithm and the cost estimation techniques.

Algorithms to minimize the response time and the total time for distributed queries were first proposed by [HEVNEA1]. Improvements and refinements to these have recently been reported in [APERSP1].

Distributed query optimization has also been examined for particular network configurations. In [KERSCL1] a query optimizer for star configured computer networks is presented. The results of experiments using the optimizer in a simulated microcomputer based network were reported. They were subsequently used as input in determining strategies with minimal response time processing for queries involving the select, project and join commands.

Most proposed algorithms [WONGE1], [HEVNEA1], [APERSP1] emphasize minimizing communication cost. In [CHUW1], however, a method for generating the optimal query processing policy that considers communication and processing costs is described. Queries are first partitioned into subqueries expressed by relational algebra operations. Their serial and parallel relationships and processing sequences are initially represented by a query tree. Based on permutability properties a number of equivalent query trees may be generated, called the set of feasible query trees. If the number of operations in the initial query tree and/or the number of transfor-

mations that can be applied to the tree are large, the processing and storage requirements for Chu's model [CHUW1] or any other model using the set of feasible query trees increases rapidly.

The primary goal of this paper is to propose heuristic rules to substantially reduce the total number of feasible query trees representing a given query. Recognizing that a fair amount of groundwork is needed, Section 2.0 presents a brief discussion of the works of [SMITHJ1] and [CHUW1] in defining permutability properties for relational operators. Section 3.0 describes two simple databases used to illustrate the results obtained when using semijoins with other operators in a particular order. The complete set of permutability properties of semijoins with other relational operators is given in Section 4.0.

2.0 PERMUTABILITY PROPERTIES OF RELATIONAL OPERATORS

Initially any query can be represented by different series of sequences producing the same correct answer. For instance, given one such series of sequences we can generate the complete set of feasible query trees by using the permutability properties of the relational algebra operations. The implementation of permutability properties for queries involving the projection, selection and join operations have been discussed in [CHUW1]. We can find permutability properties of different algebraic operators by applying commutative, associative and distributive properties. We take into account the many operators (selection, projection) and binary operations (union, join, difference, intersection, division and semijoin). Therefore, we have permutability properties by combining the following operations:

(1) Two adjacent unary operations,

(2) Adjacent unary and binary operations

(3) Two identical binary operations in sequence

(4) Two different binary operations in sequence

Earlier work by [SMITHJ1] identified the permutability properties of many operations and of many operations with binary operations. For instance, they demonstrated that, in genera, Projections (π_s) and Selections (σ_p) are permutable, but the transformation of Selection preceeding Projection to Selection succeeding Projection is permissible only if all the attributes of condition p are contained in the set s.

More recently [CHUW1] added these by defining the commutative, associative and distributivity properties of the difference, division, union, intersection and join operations.

The permutability properties of relational operators developed so far do not included the semijoin operator [BERNSP1]. The semijoin of a relation R_1 by relation R_2 is defined to be the join of R_1 and R_2 projected back onto attributes of R_1 . In other words, the semijoin retrieves all tuples for R_1 that join with any tuple of R_2 .

The extensive use of this new operator in database theory indicates that it's importance is not in doubt anymore. Therefore, in Section 3.0 a complete set of permutability properties for this operator is given following the works of [SMITHJ1] and [CHUW1].

3.0 TWO SAMPLE RELATIONAL DATABASES

Let's now describe two small, sample relational databases to be used throughout this paper to illustrate the permutability properties of relational operators.

The first database consists of tree relations: dealers, cars and dealer-car (DC). Each dealer consists of a dealer number (D), dealer name (DNAME), city (CITY) and the average price of a car (PRICE). For the relation cars we wish to record a car under car number (C), car name (CNAME), the model (MODEL) and it's price (PRICE). We assume that each dealer, as well as car, is identified by a unique number and name. The significance of an DC record is that the specified cars (C) are being supplied to the specified dealers (D) in the specified quantities (QTY).

The second sample database consists of three relations (D1, D2 and D3) with car name (CNAME) and quantity (QTY) as their attributes.

Dealers

D#	DNAME	CITY	PRICE
D1	AUTOLAND	Falls Church	13K
D2	Britt	Fairfax	7K
D3	Marshall	Arlington	11K
D4	Tysons	McLean	9K
D5	HOV	Arlington	10K

Cars

C#	CNAME	MODEL	PRICE
C1	Mazda	GLC	6K
C2	Escort	L	7K
C3	Toyota	Tercel	9K
C4	VW	WagonL	10K
C5	Chevrolet	Chevette	5.5K
C6	Buick	Regal	9K

D.C.

D#	C#	QTY
D1	C1	20
D1	C2	25
D1	C3	30
D1	C4	5
D1	C5	20
D1	C6	5
D2	C1	20
D2	C2	5
D3	C2	25
D3	C5	20
D4	C2	38
D4	C4	71
D4	C5	12

Table 1 The Dealers-and-Cars Database

D1

CNAME	QTY
Mazda	20
Escort	25
Toyota	30
VW	5
Chevrolet	20
Buick	5

D2

CNAME	QTY
Mazda	20
Escort	5

D3

CNAME	QTY
Escort	5
Chevrolet	20

Table 2 Dealers Database

4.0 PERMUTABILITY PROPERTIES OF SEMIJOINS

The permutability properties of semijoins with other unary and binary operations are developed by combining unary operators (selection, projection) and binary operators (difference, division, union, intersection, join and semijoin) in a particular order.

The notation to be used is as follows:

Relations: A, B and C

PROJECTION (π): projects its operand into the set of attributes s.

SELECTION (σ_p): selection condition p.

DIFFERENCE (-): left hand side relation minus right hand side relation

DIVISION ($\div$): left hand side relation divided by the right had side relation

UNION ($\cup$)

INTERSECTON ($\cap$)

NATURAL JOIN ($A \bowtie B$): applicable only when both A and B have columns that are named by attributes.

SINGLE DOMAIN SEMIJOIN $(A \ltimes B) = \pi_A(A \bowtie B)$

$(A \rtimes B) = \pi_B(A \bowtie B)$

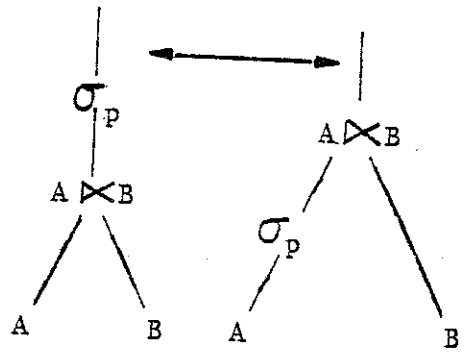
3.1 Semijoin with Adjacent Unary Operations

Let us consider the selection and projection operations

(i) Semijoin with adjacent selection: When a selection is followed by a semijoin we can exchange them without modifying the resultant query. If a semijoin on A, i.e., ($A \ltimes B$), follows a selection condition p an exchange is feasible only if all the attributes in condition p are attri-

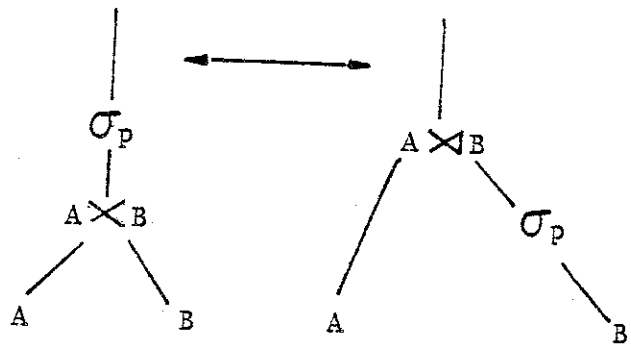
butes of relation A. While, if a semijoin on B, i.e., (A ⋉ B), follows a selection condition p an exchange is feasible only if all the attributes in condition p are attributes of relation B. Both of these bidirectional transformations are illustrated in Figure 1.

(ii) Semijoin with adjacent projection: When a projection is followed by a semijoin we can exchange them without modifying the resultant query. The bidirectional transformation shown in Figure 2 can be performed only if all the attributes deleted in the projection do not intersect with the semijoin's selection attributes.



The left to right transformation above can be performed only if all the attributes in condition p are attributes of relation A.

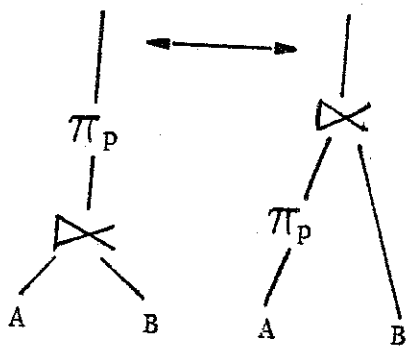
(a)



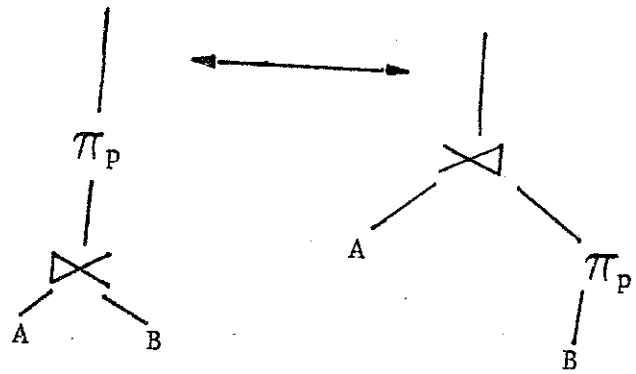
The left to right transformation above can be performed only if all the attributes in condition p are attributes of relation B.

(b)

FIGURE 1 Semijoin with adjacent selection



(a)

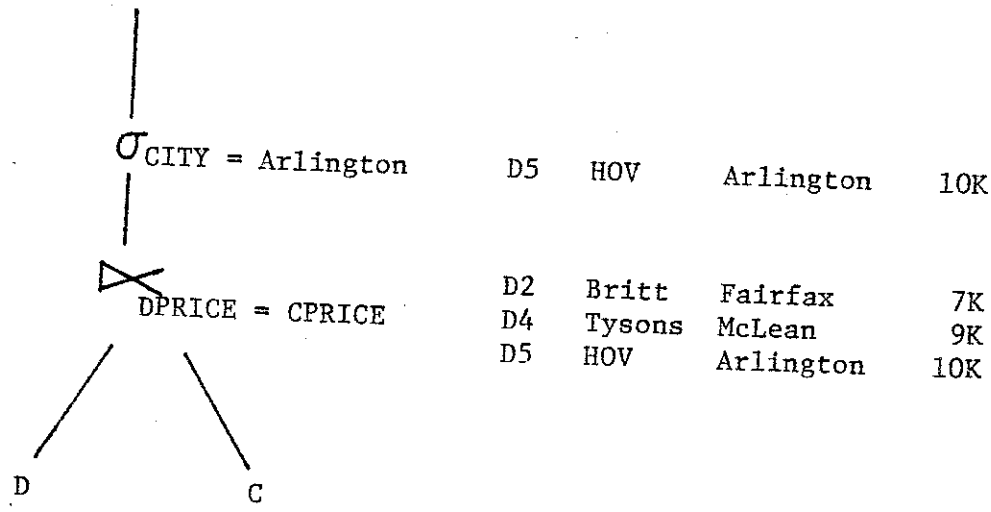


(b)

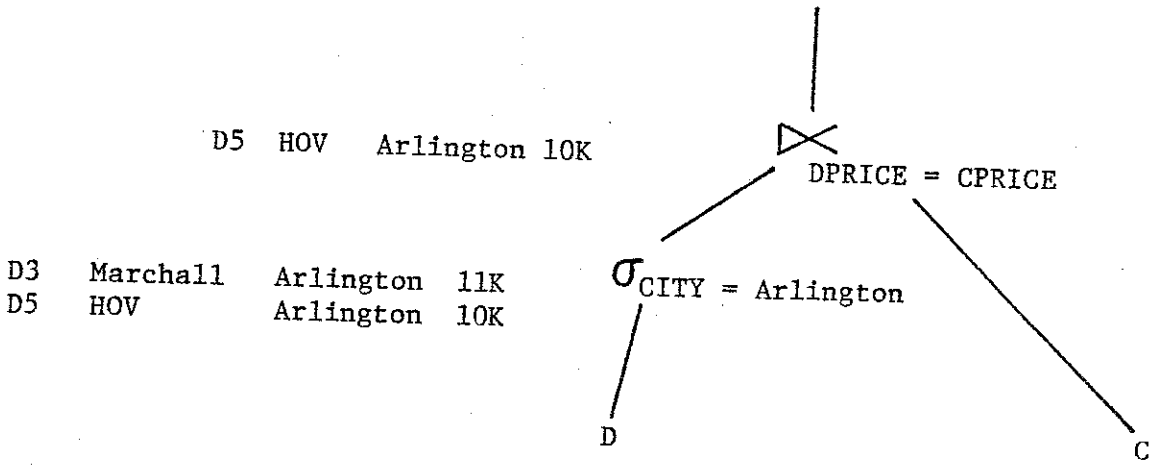
The left to right transformations can be performed only if all the attributes deleted in the projection do not intersect with the semijoin's selection attributes.

FIGURE 2 Semijoins with adjacent projection

Next, we illustrate, using the Dealers-and-Cars database, the permutability properties of semijoins with adjacent selection and projection operations. The results are given in Figure 3 and 4 respectively.

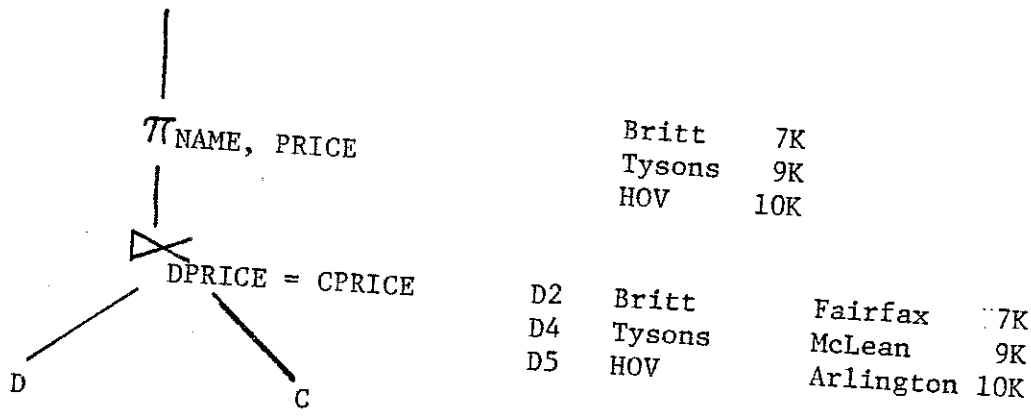


(a)

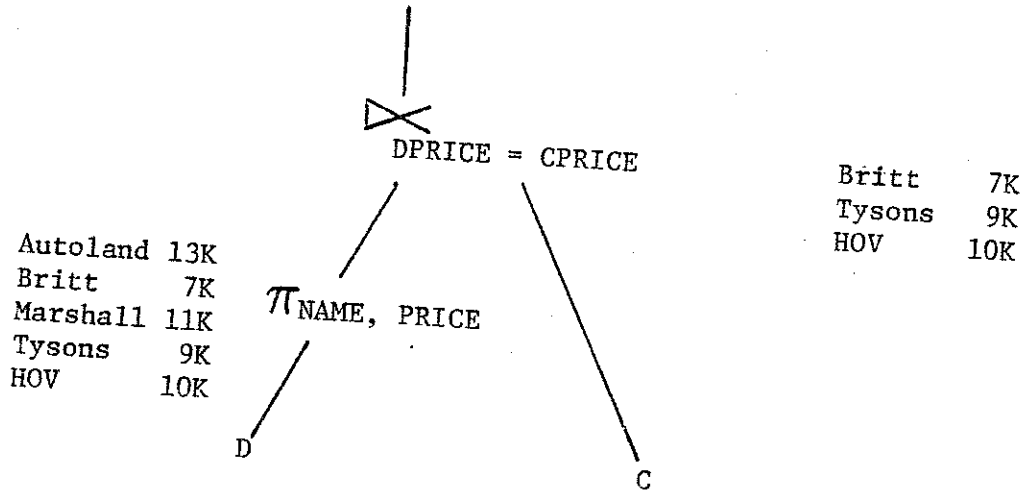


(b)

FIGURE 3 Semijoin with adjacent selection



(a)



(b)

FIGURE 4 Semijoin with adjacent projection

3.2 Two Identical Binary Operations in Sequence

(i) Two semijoin operations in sequence: Since semijoin operations are commutative and associative, if we project on the same set of attributes on both operations we can exchange the relations that project back after the join operation. In Figure 5 they correspond to relations B and C. The bidirectional transformation holds for both equality and inequality semijoins.

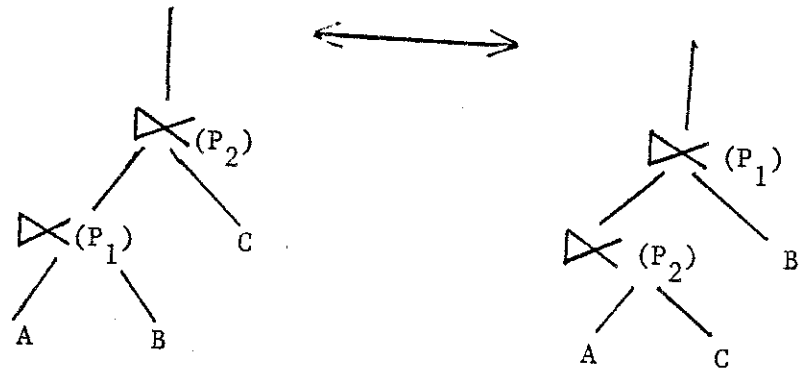


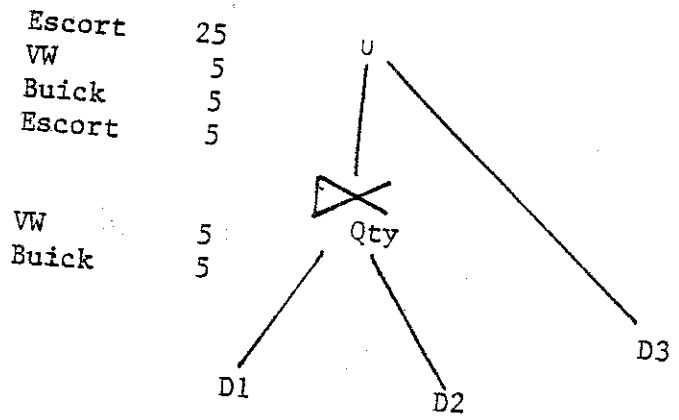
FIGURE 5 Semijoin operations in sequence

3.3 Semijoin operations with other binary operators

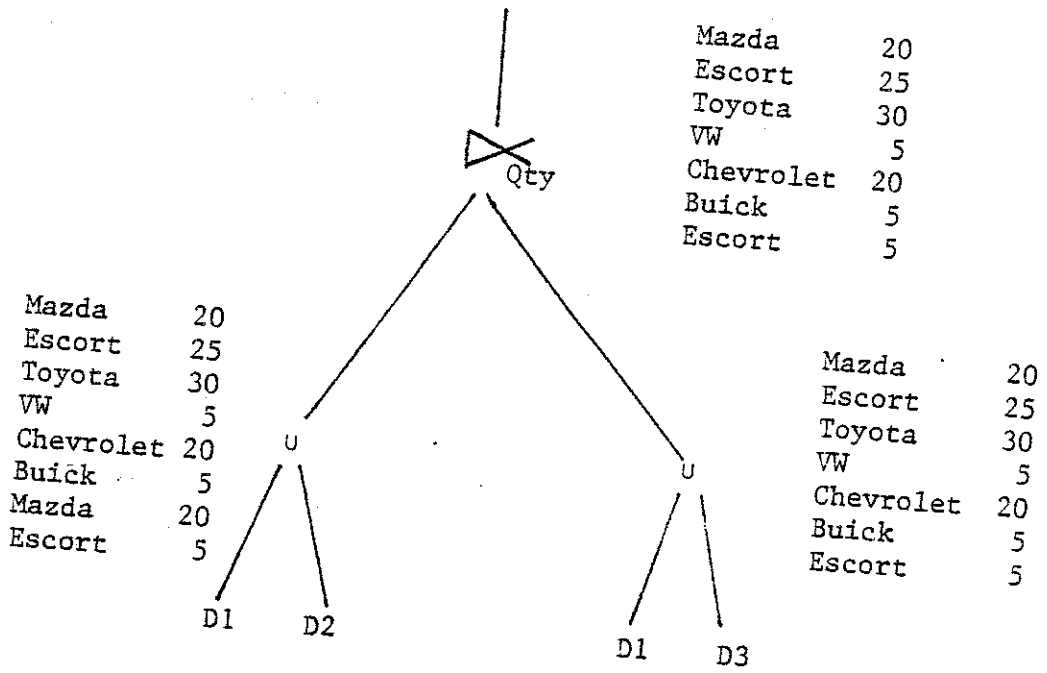
(i) Semijoin and union: Semijoin can not always be permuted with a subsequent union operation, i.e.:

$$(A \ltimes B) \cup C \neq (A \cup B) \ltimes (A \cup C)$$

If a union operation is operated before a semijoin, the sequence of operations results in a larger set than if we perform the semijoin. The reason is that large files are produced after a union operation which when accompanied by a semijoin gives a large set of tuples. The sequence of operations mentioned above may be equal in some cases but it is not universal. An example of semijoin and union exchange that generates a different response to the same query is given by Figure 6.



(a)



(b)

FIGURE 6 An Example of Semijoin and Union Exchange Generating a Different Query Response

3.4 Semijoin and Intersection

Semijoin is not distributive over intersection, by Theorem. Thus,

$$(A \ltimes B) \cup C \neq (A \cup B) \ltimes (A \cup C)$$

Figure 7 shows an example of the above property using the sample database given by Table 2.

Furthermore, a semijoin if permuted with an intersection gives, in most instances, a different query response. Thus,

$$(A \ltimes B) \cap C \neq (A \cap C) \ltimes B$$

and $$(A \ltimes B) \cap C = (A \ltimes B) \cap (C \ltimes B)$$

Both cases are illustrated by Figures 8 and 9 respectively.

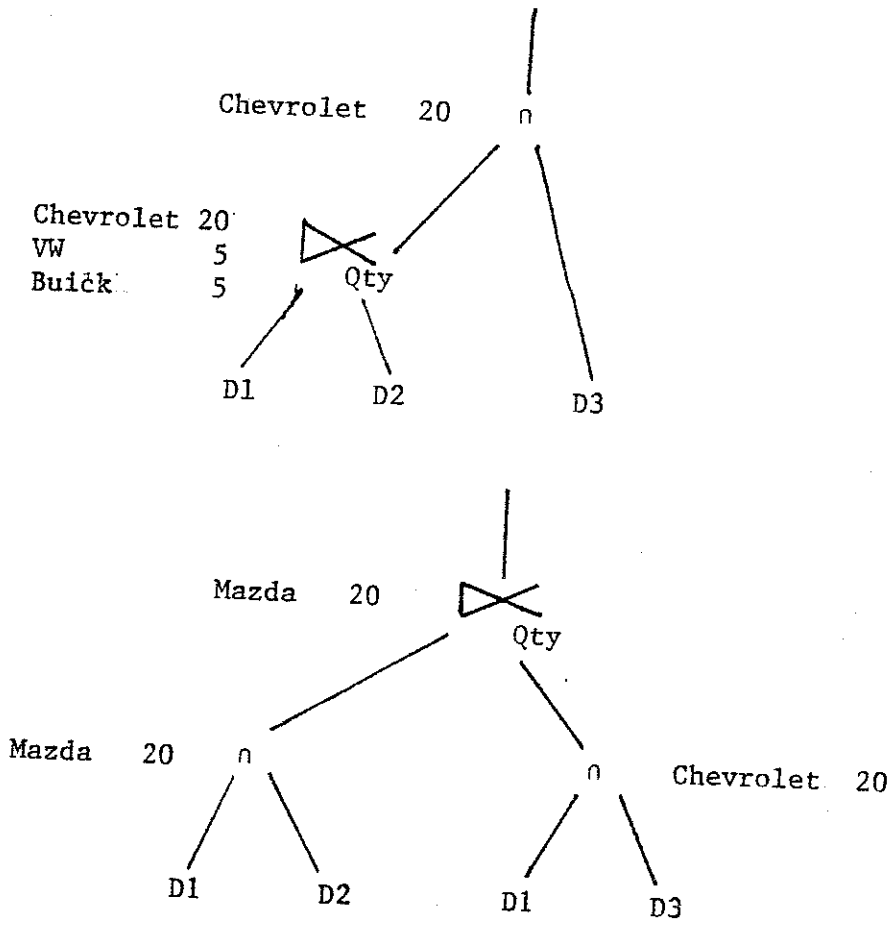
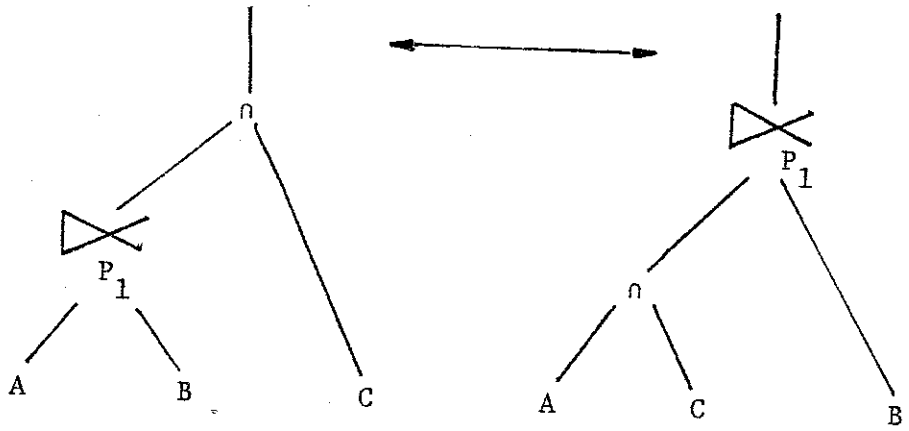
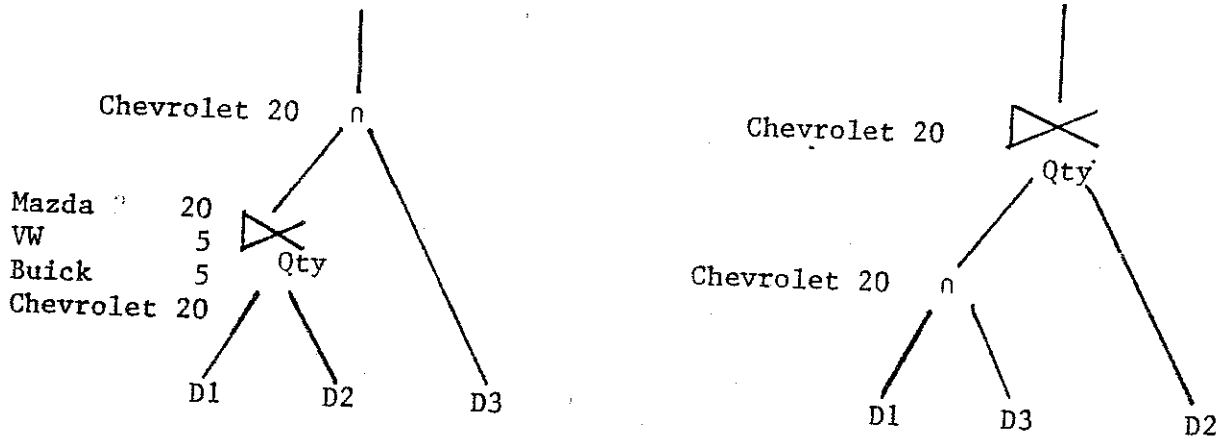


FIGURE 7 An Example of Semijoin and Intersection Exchange Generating a Different Query Response

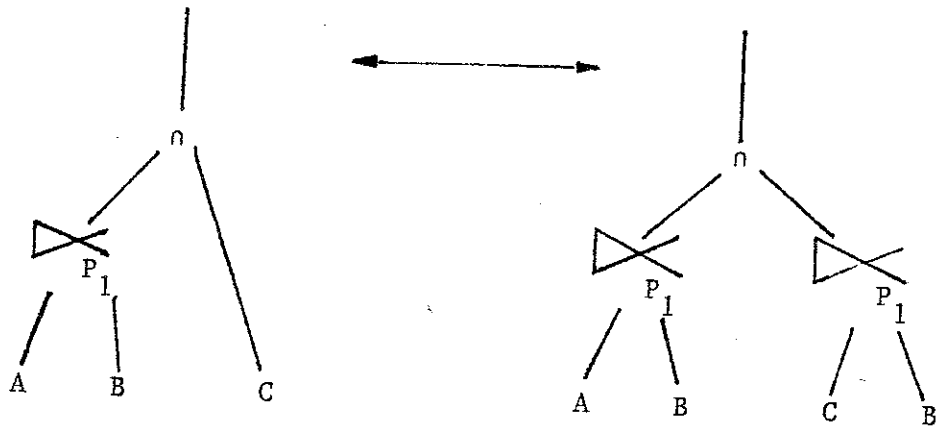


(a)

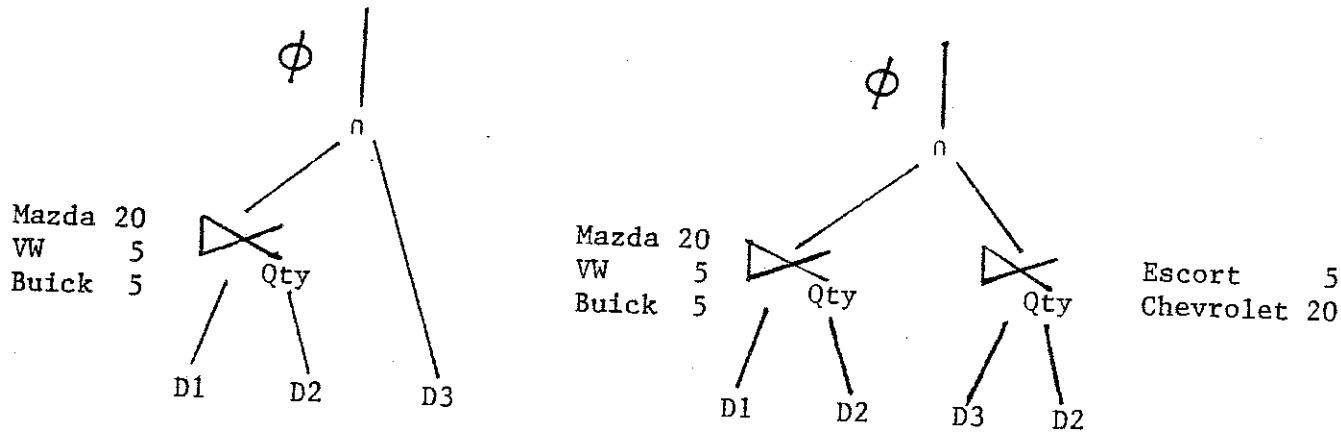


(b)

FIGURE 8 Semijoin permuted with Intersection



(a)



(b)

FIGURE 9 Semijoin permuted with Intersection

3.5 Semijoin and Difference

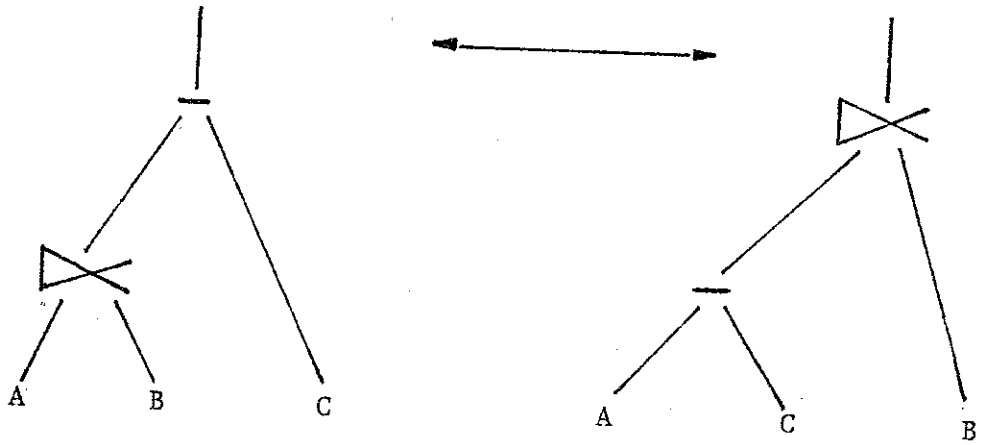
Semijoin can be permuted with difference, as demonstrated by Theorem. Thus,

(i) $(A \ltimes B) - C = (A - C) \ltimes B$

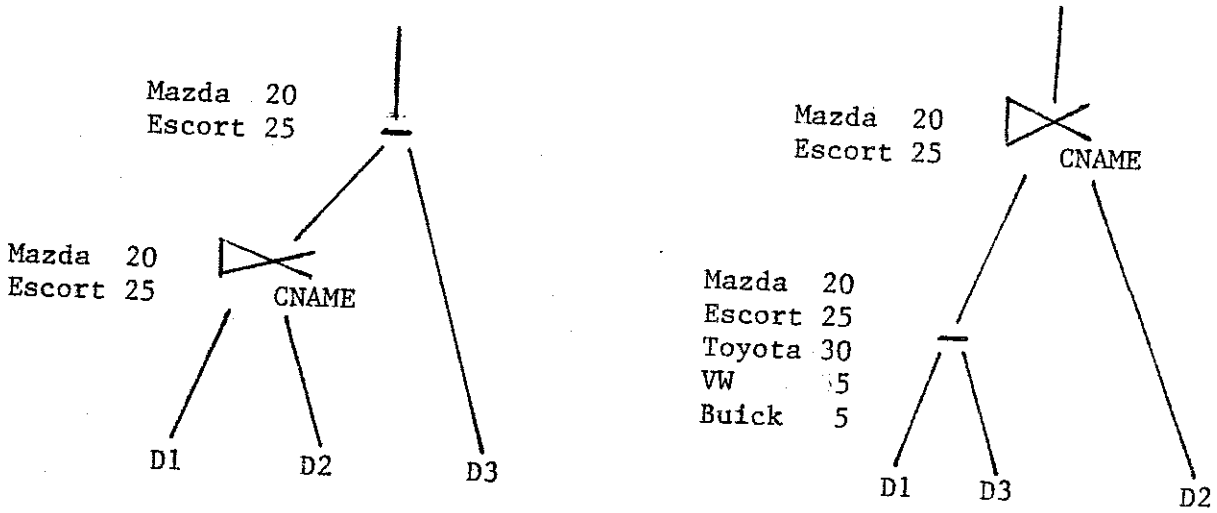
(ii) $(A \ltimes B) - C = (A - C) \ltimes (B - C)$

(iii) $(A \ltimes B) - C = (A \ltimes B) - (C \ltimes B)$

All three cases are depicted graphically, and examples given by Figures 10, 11 and 12 respectively.

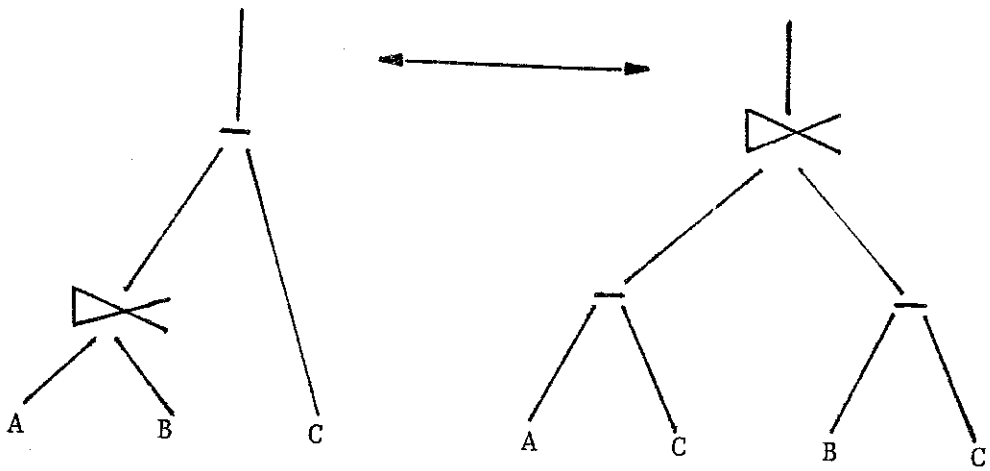


(a)

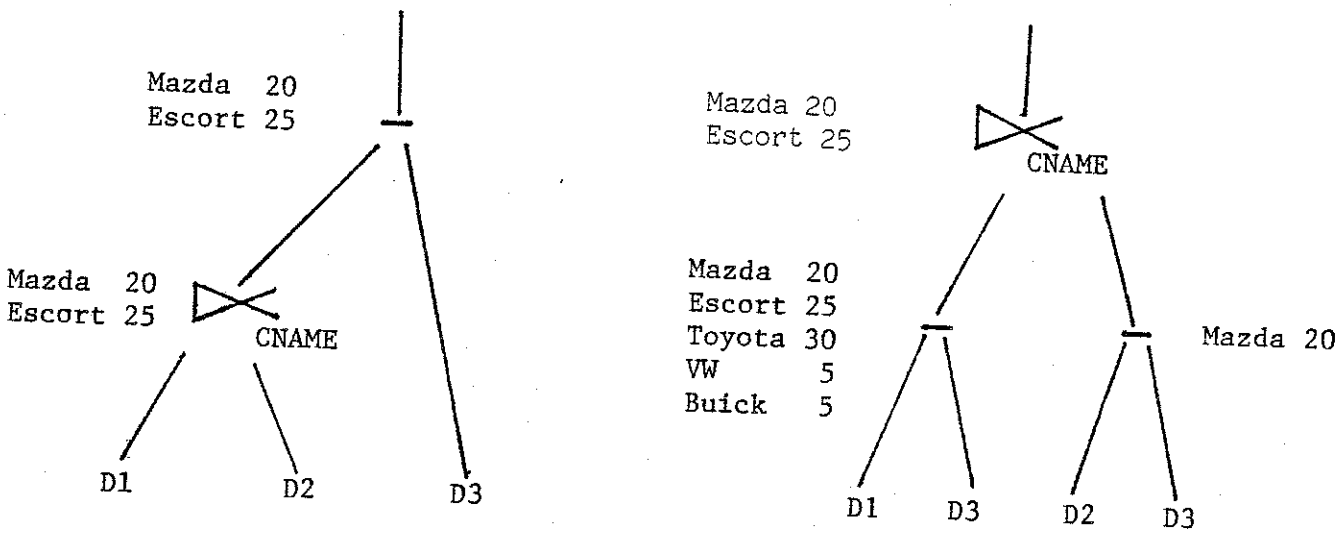


(b)

FIGURE 10 Semijoin with Difference

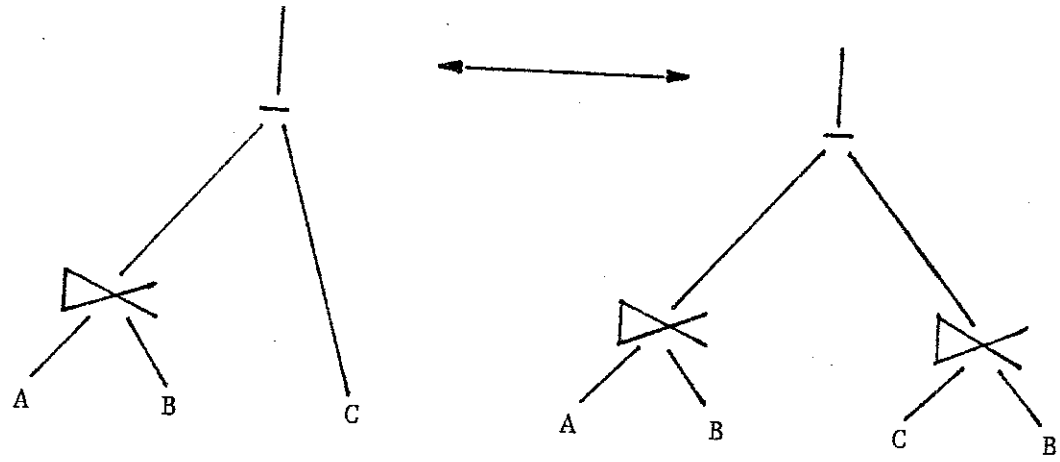


(a)

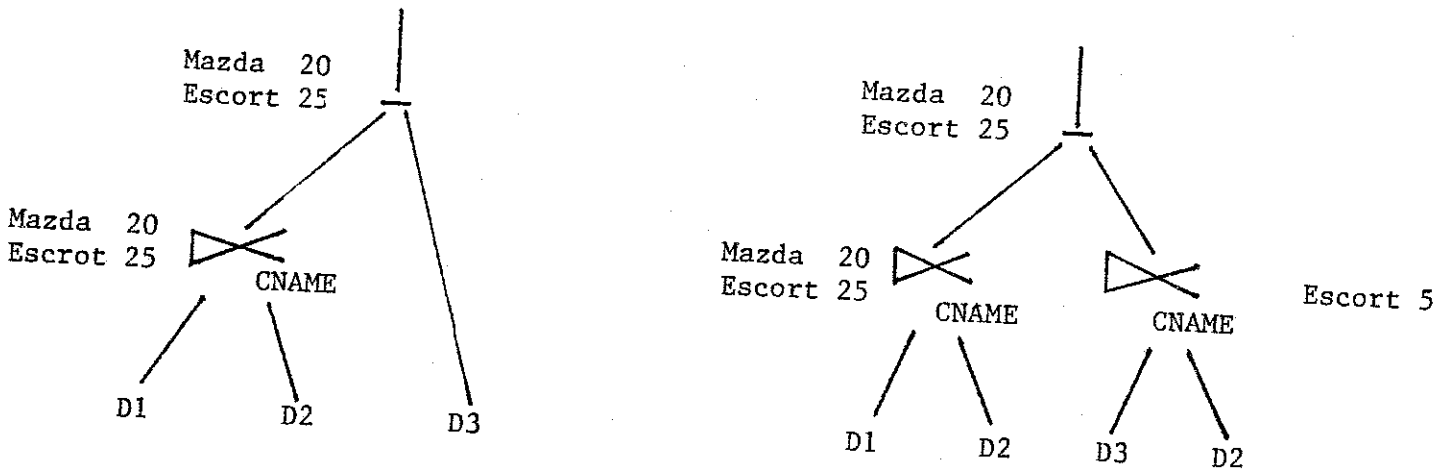


(b)

FIGURE 11 Semijoin with Difference



(a)



(b)

FIGURE 12 Semijoin with Difference

3.6 Semijoin and Join

Semijoin can be permutted with join, by Theorem if we apply the same condition p. Thus,

$$(A \ltimes B) \bowtie C = (A \bowtie C) \ltimes B$$

Figure 13 illustrates, using the database given by Table 2, the above property.

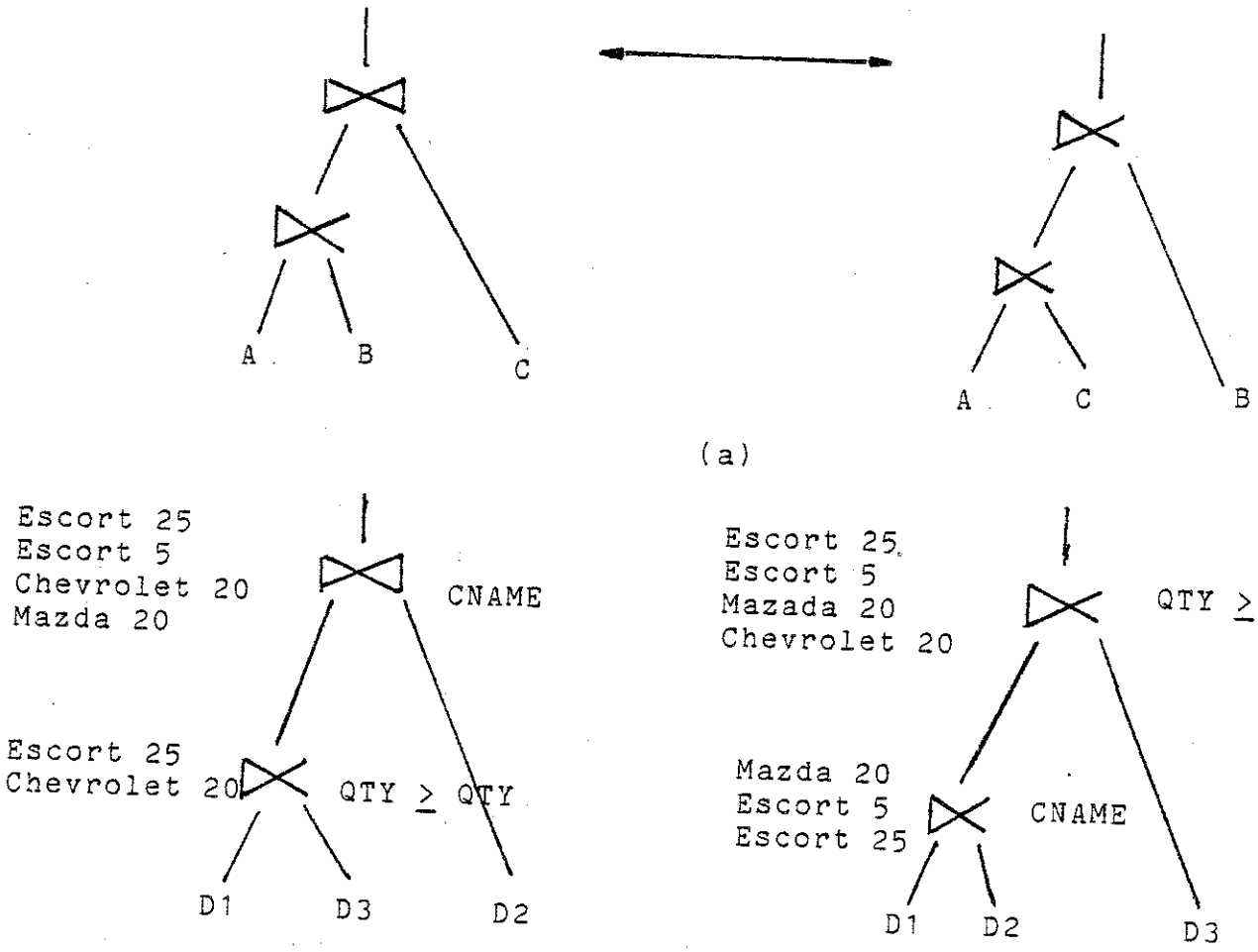


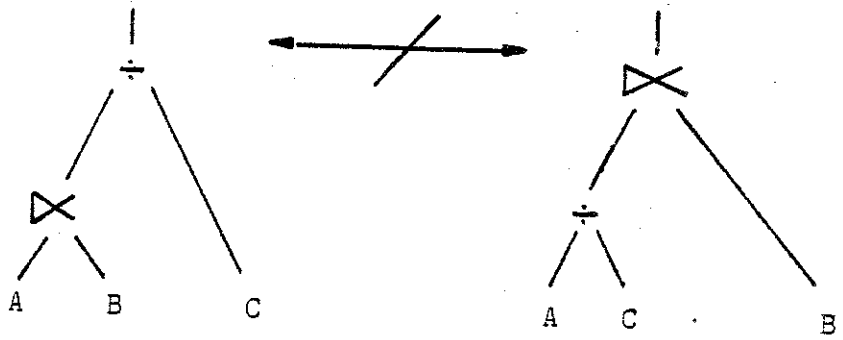
FIGURE 13 Semijoin and Join

3.7 Semijoin and Division

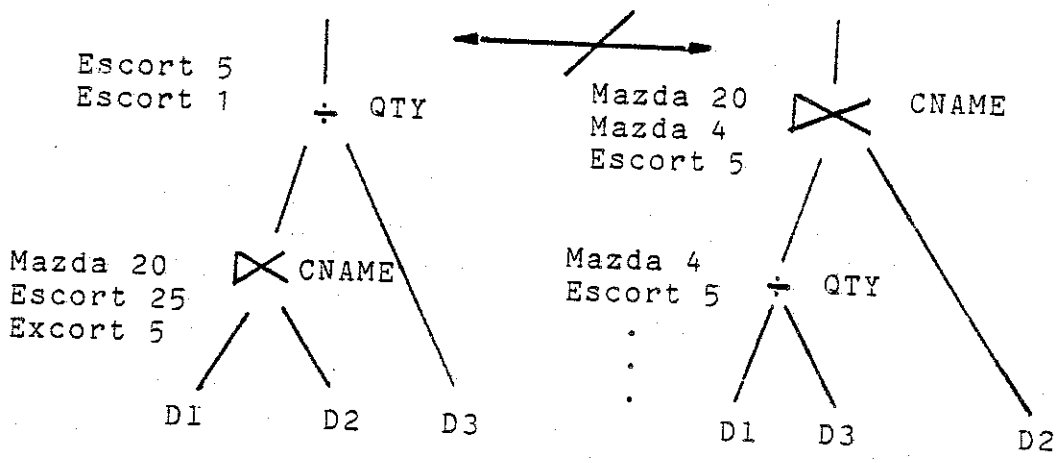
If a semijoin is permutted with a subsequent division the result of the two operations will have fewer tuples than if the semijoin preceeds the division. This occurs because the division operator eliminates some tuple values from the operand. Therefore, semijoins can not be permuttedwith division.

$$(A \ltimes B) \div C \quad (A \div C) \ltimes B$$

Figure 14 depicts a case, using the sample database given by Table 2, where the permutability property of semijoin and division is violated.



(a)



(b)

FIGURE 14 Semijoin and Division

References

Query Processing Optimization in Distributed Systems

[KERSCL1] Kerschberg L., Ting P. and Yao B. "Query Optimization in Star Computer Networks" ACM TODS Vol. 7, No. 4, December 82.

[BERNSP1] Bernstein, P. A., Goodman, N., Wong, E., Reeve, C. and Rothnie, J., "Query Processing in a System for Distributed Databases" ACM TODS, Vol. 6, No. 4, December 1981.

[SMITHJ1] Smith, J. M. and Chang, P.Y. "Optimizing the Performance of a Relational Algebra Database Interface" Communications of the ACM, Vol. 18, No. 10, October, 1975.

[HEVNEA1] Hevner, A.R. and Yao, S. B. "Query Processing in Distributed Database Systems" IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May, 1979.

[CHUW1] Chu, W. W. and Hurley, P. "Optimal Query Processing for Distributed Database Systems" IEEE Transactions on Computers, Vol. C-31, No. 9, September, 1982.

[WONGE1] Wong, E. "Retrieving dispersed data from SDD-1" Prac. Berkeley Workshop Distributed Data Mangement and Computer Netowrks, May, 1977.

[APERSP1] Apers, P., Hevner, A. and Yao, B. "Optimization Algorithms for Distributed Queries" IEEE Transactions on Software Engineering, Vol. SE-9, No. 1, January, 1983.

[BERNSP1] Bernstein, P. A. and Chin D. W. "Using Semijoins to Solve Relational Qeries" JACM 28(1981) pp 25-40.

[YUC1] Yu, C. T. and Chang, C. C. "On the Design of a Query Processing Strategy in a Distributed Database Environment" Proceedings: Database Week, SIGMOD Record Volume 13, No. 4, San Jose, Ca. May 23-26, 1983.

[ROTHNJ1] Rothnie, J. B. and Goodman, N. An overview of the preliminary design of SDD-1. In Proc. Berkeley Workshop Distributed Data Management and Computer Networks, (May 1977).