UNIT-4

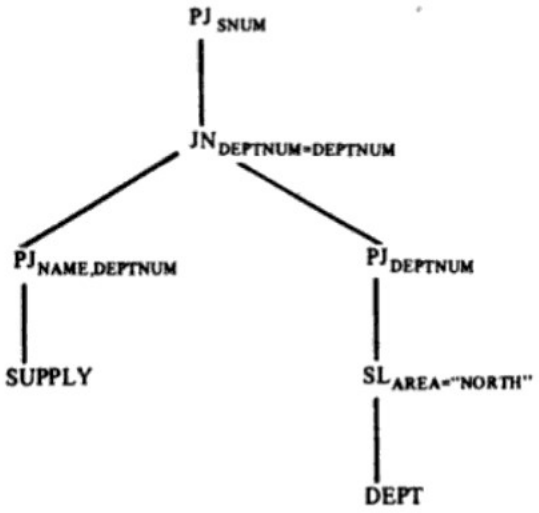
4.1 TRANSFORMING GLOBAL QUERIES INTO FRAGMENT QUERIES

A. Canonical Expression of a Fragment Query

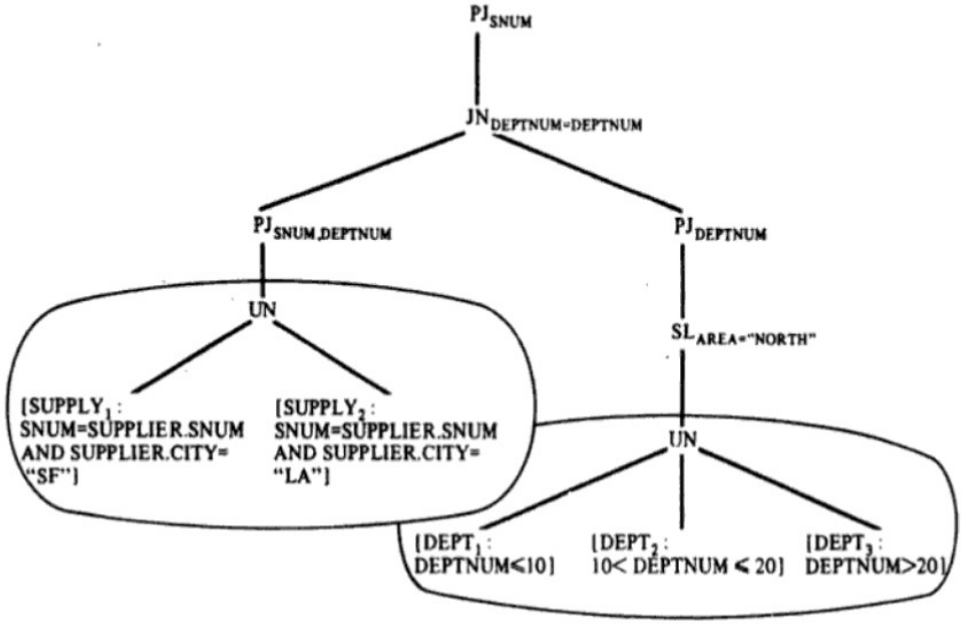
A canonical expression for the algebraic expression over the global schema is obtained as follows:

"Reconstruction of global relation is obtained from fragments. We map an operator tree on the global schema, in which collecting the fragments of all global relations into temporary relations and then applying to these temporaries the global query. "

Example: Operator tree of Q1 as follows



The following figure shows the canonical expression of above tree



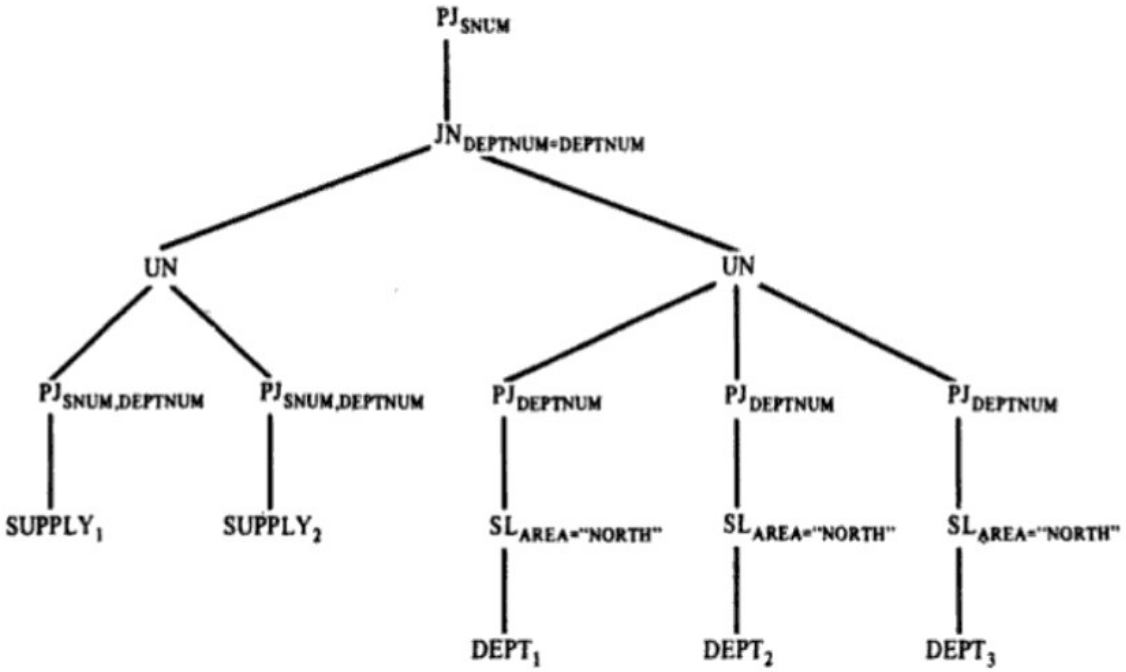
(*a*) Canonical form of query Q1.

We apply following criteria 1 and criteria 2 for further simplification of above canonical expression.

Criterion 1 : Use Idempotence of selection and projection to generate appropriate selections and projections for each operand relation

Criterion 2 : Push selections and projections down in the tree as far as possible.

After applying these two criterions, the resulting operator are described as follows:



(*b*) Pushing selections and projections down in the operator tree.

B. Algebra of Qualified Relations

A qualified relation is a relation extended by a qualification and the qualification possessed by all the tuples of the relation.

Qualified relation can be denoted by a pair **[R :qR],** where **R** is a relation called the **body of the qualified relation and qR is a predicate called the qualification of the qualified relation. Horizontal fragments are** typical examples of **qualified relations**.

The following rules define the result of applying the operations of relational algebra to qualified relations:

Rule 1 $\quad \mathbf{SL}_F[R : q_R] \Rightarrow [\mathbf{SL}_F R : F \text{ AND } q_R]$

This rule states that the application of a selection $\mathbf{SL}_F$ to a qualified relation $[R : q_R]$ produces a qualified relation having the relation $\mathbf{SL}_F R$ as its body and the predicate F AND q_R as its qualification. The extension of the qualification to F AND q_R after a selection reflects the fact that F holds on all the selected tuples as well as q_R.

Rule 2 $$\mathbf{PJ}_A[R : q_R] \Rightarrow [\mathbf{PJ}_A R : q_R]$$

The qualification of the result of a projection remains unchanged, even if the projection eliminates some of the attributes upon which the qualification was evaluated.

Rule 3 $$[R : q_R]\ \mathbf{CP}\ [S : q_S] \Rightarrow [R\ \mathbf{CP}\ S : q_R \text{ AND } q_S]$$

The extension of the qualification to q_R AND q_S is rather intuitive; notice that the two qualifications apply to disjoint attributes of $R\ \mathbf{CP}\ S$.

Rule 4 $$[R : q_R]\ \mathbf{DF}\ [S : q_S] \Rightarrow [R\ \mathbf{DF}\ S : q_R]$$

However, the following problem arises with rule 4: consider the intersection operation of traditional relational algebra, defined as $R\ \mathbf{IN}\ S = R\ \mathbf{DF}\ (R\ \mathbf{DF}\ S)$. From the definition, it is easy to show that intersection is commutative; i.e.,

$$R\ \mathbf{IN}\ S = S\ \mathbf{IN}\ R.$$

If we apply the definition of extended algebra, we come to a surprising result:

$$[R : q_R]\ \mathbf{IN}\ [S : q_S] \Rightarrow [R : q_R]\ \mathbf{DF}\ ([R : q_R]\ \mathbf{DF}\ [S : q_S]) \Rightarrow$$
$$[R : q_R]\ \mathbf{DF}\ [R\ \mathbf{DF}\ S : q_R] \Rightarrow [R\ \mathbf{DF}\ (R\ \mathbf{DF}\ S) : q_R] \Rightarrow$$
$$[R\ \mathbf{IN}\ S : q_R] \qquad (5.4a)$$

$$[S : q_S]\ \mathbf{IN}\ [R : q_R] \Rightarrow [S : q_S]\ \mathbf{DF}\ ([S : q_S]\ \mathbf{DF}\ [R : q_R]) \Rightarrow$$
$$[S : q_S]\ \mathbf{DF}\ [S\ \mathbf{DF}\ R : q_S] \Rightarrow [S\ \mathbf{DF}\ (S\ \mathbf{DF}\ R) : q_S] \Rightarrow$$
$$[S\ \mathbf{IN}\ R : q_S] \qquad (5.4b)$$

In fact, the result that we would like to find is

$$[R\ \mathbf{IN}\ S : q_R \text{ AND } q_S]$$

Rule 5 $$[R : q_R]\ \mathbf{UN}\ [S : q_S] \Rightarrow [R\ \mathbf{UN}\ S : q_R \text{ OR } q_S]$$

The union is extended by taking the disjunction of qualifications.

Given the above rules, let us see how derived operations of algebra, such as join and semi-joins, are extended. We have two additional rules:

Rule 6 $$[R : q_R]\ \mathbf{JN}_F\ [S : q_S] \Rightarrow [R\ \mathbf{JN}_F\ S : q_R \text{ AND } q_S \text{ AND } F]$$

Rule 7 $$[R : q_R]\ \mathbf{SJ}_F\ [S : q_S] \Rightarrow [R\ \mathbf{SJ}_F\ S : q_R \text{ AND } q_S \text{ AND } F]$$

We give following two criterion 3 and 4 based on qualification of relations in further simplification.

Criterion 3. Push selections down to the leaves of the tree, and then apply them using the algebra of qualified relations; substitute the selection result with the empty relation if the qualification of the result is contradictory.

Criterion 4. Use the algebra of qualified relations to evaluate the qualification of operands of joins; substitute the subtree, including the join and its operands, with the empty relation if the qualification of the result of the join is contradictory.

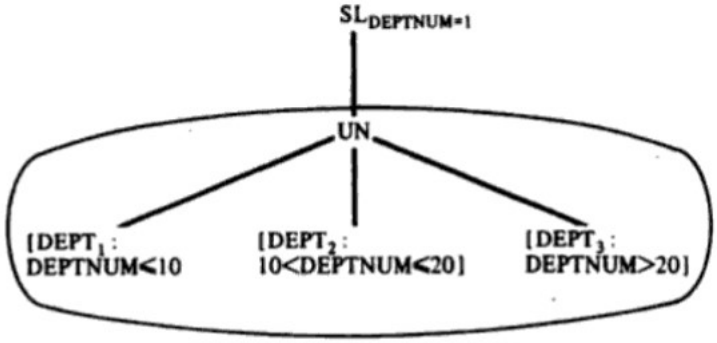
These two criterions are used in simplification of horizontally fragmented relations and joins between horizontally fragmented relations.

C. Simplification of Horizontally Fragmented Relations

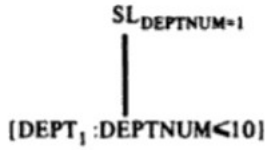
We show this kind of simplification with an example. Let us consider query Q3 on relation *DEPT*, which is horizontally fragmented:

$$Q3 : \mathbf{SL}_{DEPTNUM=1}\ DEPT$$

The canonical form of the query is shown in Figure 5.5a. We push the selection toward the leaves by distributing it with respect to union. Then we apply selections according to criterion 3; the qualification of results of the selections on $DEPT_2$ and $DEPT_3$ is a contradiction;



(a) Canonical form of query Q3.



(b) Simplification of query Q3.

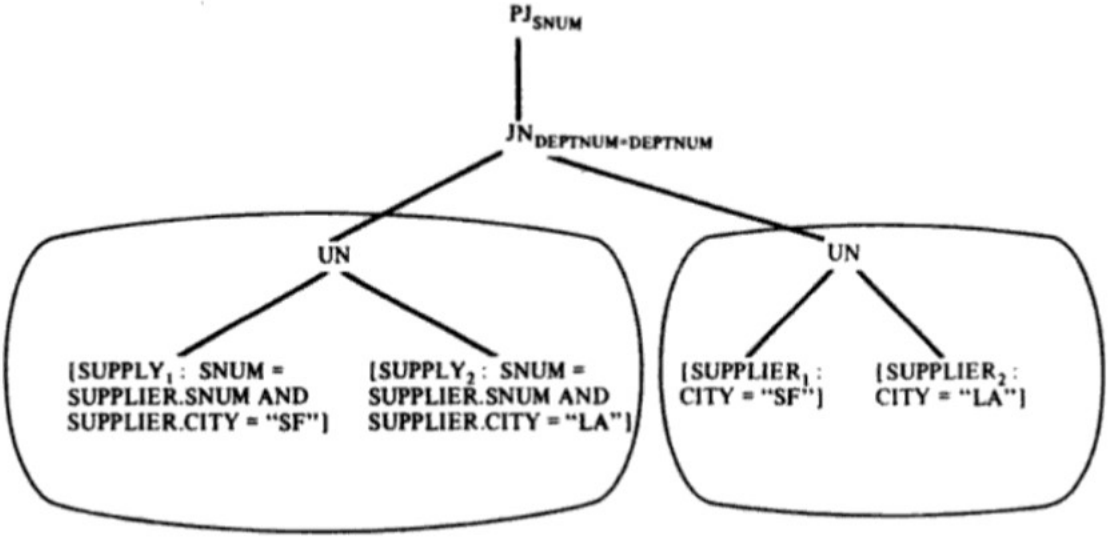
D. Simplification of Joins between Horizontally Fragmented Relations

Let us consider, for simplicity, the join between two fragmented relations R and S. There are two distinct possibilities of joining them; the first one requires collecting all the fragments of R and S before performing the join. The second one consists of performing the join between fragments and then collecting all the results into the same result relation; we refer to this second case as "distributed join."

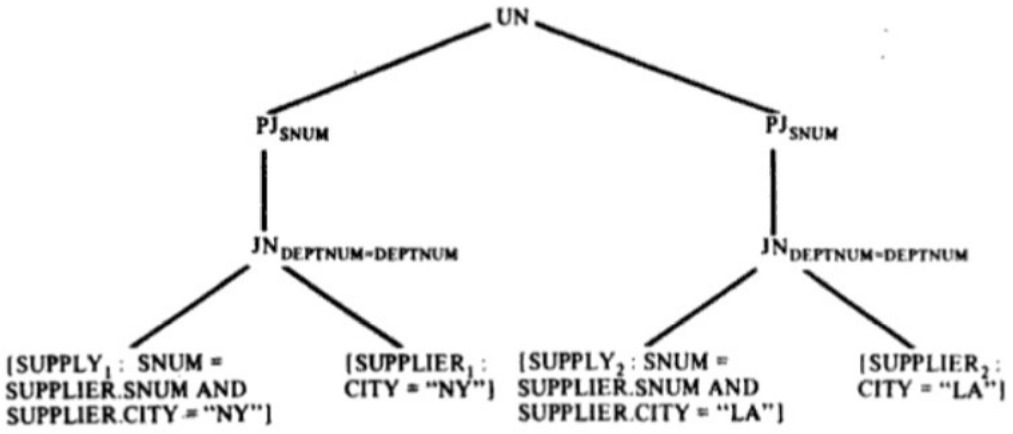
Criterion 5. In order to distribute joins which appear in the global query, unions (representing fragment collections) must be pushed up, beyond the joins that we want to distribute.

Building a join graph requires, then, applying criterion 5 (for distributing the join) followed by criterion 4

Figure 5.6a shows the canonical form of the query. We recall that the fragmentation of *SUPPLY* is derived from the fragmentation of *SUPPLIER*; i.e., each tuple of *SUPPLY* is stored either in fragment $SUPPLY_1$, if it refers to a supplier of San Francisco, or in fragment $SUPPLY_2$, if it refers to a supplier of Los Angeles. Applying criterion 5, we push the two unions up beyond the join; thus, we generate four joins between fragments. We then apply criterion 4, and we discover that two of them are intrinsically empty because their qualification is contradictory. The empty joins are those of $SUPPLIER_1$ (in "SF") with $SUPPLY_2$ (of "LA" suppliers), and likewise of $SUPPLIER_2$ (in "LA") with $SUPPLY_1$ (of "SF" suppliers). Thus, the operator tree reduces to that of Figure 5.6*b*. Assuming that fragments with the same index are placed at the same site (i.e., that data about *SUPPLY* are stored together with data about *SUPPLIERS*), this operator tree corresponds to an efficient way of evaluating the query, because each join is local to one site.



(*a*) Canonical form of query Q4.



(*b*) Distributed join for query Q4.

Figure 5.6 Simplification of joins between horizontally fragmented relations.

E. Using Inference for Further Simplifications

Assume that the following knowledge is available to the query optimizer:

1. The North area includes only departments 1 to 10.
2. Orders from departments 1 to 10 are all addressed to suppliers of San Francisco.

We use the above knowledge to "infer" contradictions that allow eliminating subexpressions.

a. From 1 above, we can write the following implications:

$$AREA = \text{"North"} \Rightarrow \text{NOT}\,(10 < DEPTNUM \leq 20)$$
$$AREA = \text{"North"} \Rightarrow \text{NOT}\,(DEPTNUM > 20)$$

Using criterion 3, we apply the selection to fragments $DEPT_1$, $DEPT_2$, and $DEPT_3$ and evaluate the qualification of the results. By virtue of the above implications, two of them are contradictory. This allows us to eliminate the subexpressions for fragments $DEPT_2$ and $DEPT_3$. Thus, the operator tree of Figure 5.4*b* reduces to that of Figure 5.7*a*.

b. We then apply criterion 5 for distributing the join; in principle, we would need to join the subtree including $DEPT_1$ with both subtrees including $SUPPLY_1$ and $SUPPLY_2$. But from 1 above, we know that:

to join the subtree including $DEPT_1$ with both subtrees including $SUPPLY_1$ and $SUPPLY_2$. But from 1 above, we know that:

$$AREA = \text{"North"} \Rightarrow DEPTNUM \leq 10$$

and from 2 above we know that:

$$DEPTNUM \leq 10 \Rightarrow$$
$$\text{NOT}\,(SNUM = SUPPLIER.SNUM \text{ AND } SUPPLIER.CITY = \text{"LA"})$$

By applying criterion 4, it is possible to deduce that only the subtree including $SUPPLY_1$ needs to be joined. The final operator tree for query Q1 is shown in Figure 5.7*b*.

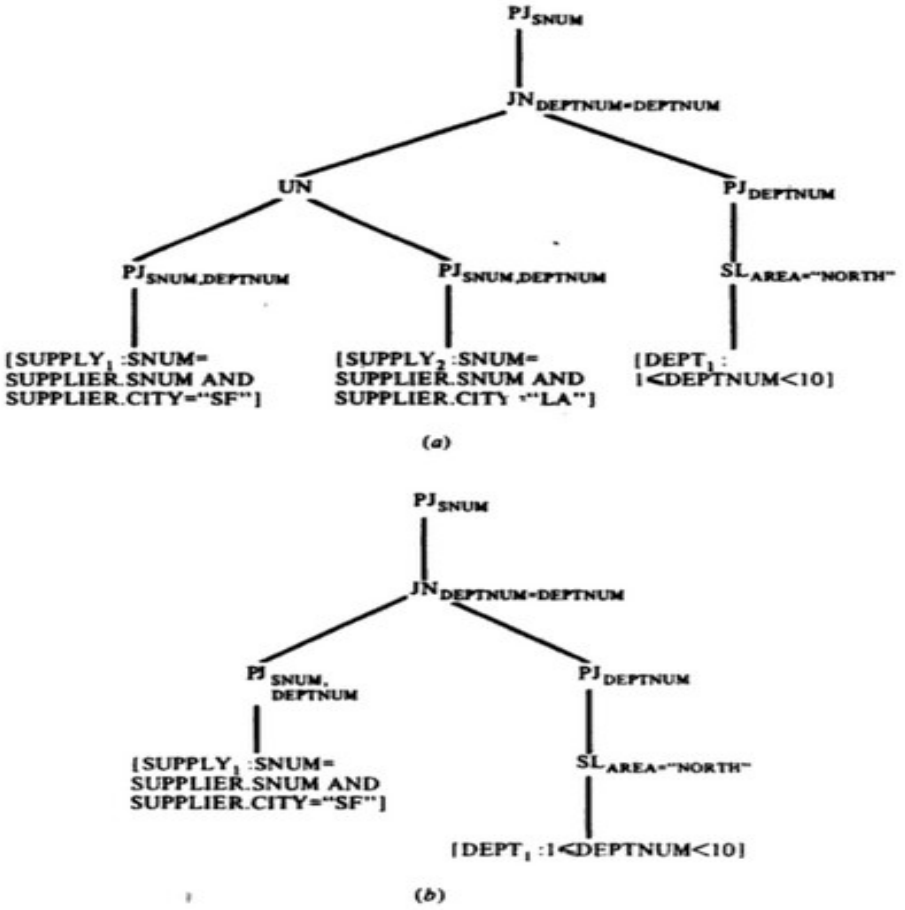


Figure 5.7 Simplification of an operator tree using inference.

F. Simplification of Vertically Fragmented Relations

Simplification is to determine a proper subset of the fragments which is sufficient for answering the query, then eliminate all other fragments from the query expression, as well as the joins which are used in the inverse of the fragmentation schema for reconstructing global relations.

For the following query, simplification of vertical fragmented relations are as follows:

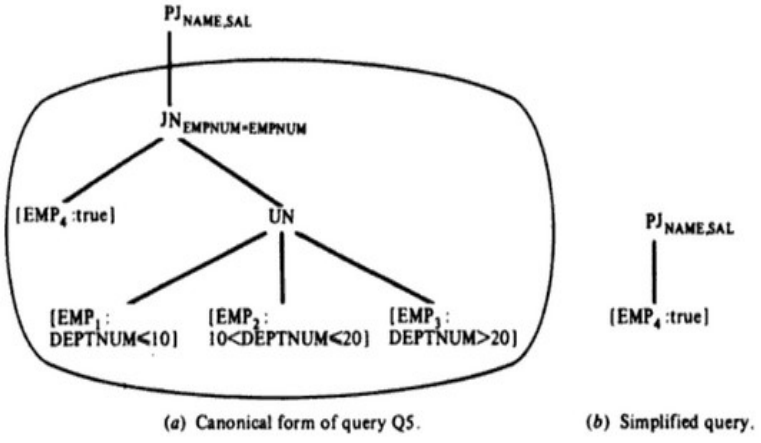


Figure 5.8 Simplification of vertically fragmented relations.

4.2 EQUIVALENCE TRANSFORMATIONS FOR QUERIES:

It is possible to transform most SQL queries into equivalent expressions of relational algebra.

4.2.1 Operator Tree of a query:

In order to have a more practical representation of queries, in which expression manipulation is easier to follow, we introduce **operator trees**. Let us consider query Q1, which requires the supplier number of suppliers that have issued a supply order in the North area of our company. Query Q1 corresponds to the following expression of the relational algebra:

$$\text{Q1} : \mathbf{PJ}_{SNUM}\, \mathbf{SL}_{AREA=\text{"North"}}(SUPPLY\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\ DEPT)$$

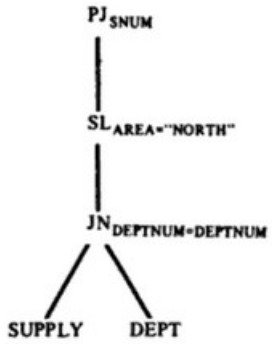


Figure 5.1 An operator tree for query Q1.

The operator tree of an expression of relational algebra can be regarded as the parse tree of the expression itself, assuming the following grammar:

$R \rightarrow$ identifier
$R \rightarrow (R)$
$R \rightarrow un\_op\ R$
$R \rightarrow R\ bin\_op\ R$
$un\_op \rightarrow \mathbf{SL}_F \mid \mathbf{PJ}_A$
$bin\_op \rightarrow \mathbf{CP} \mid \mathbf{UN} \mid \mathbf{DF} \mid \mathbf{JN}_F \mid \mathbf{NJN}_F \mid \mathbf{SJ}_F \mid \mathbf{NSJ}_F$

4.2.2 Equivalent transformations for the Relational Algebra

Equivalence transformations can be given systematically for small expressions, i.e., expressions of two or three operand relations. These transformations are classified into categories according to the type of the operators involved. Let U and B denote unary and binary algebraic operations, respectively. We have:

- **Commutativity** of unary operations:

$$U_1\ U_2\ R \leftrightarrow U_2\ U_1\ R$$

- **Commutativity** of operands of binary operations:

$$R\,B\,S \leftrightarrow S\,B\,R$$

- **Associativity** of binary operations:

$$R\,B\,(S\,B\,T) \leftrightarrow (R\,B\,S)\,B\,T$$

- **Idempotence** of unary operations:

$$U\,R \leftrightarrow U_1\,U_2 R$$

- **Distributivity** of unary operations with respect to binary operations:

$$U(R\,B\,S) \rightarrow U(R)\,B\,U(S)$$

- **Factorization** of unary operations (this transformation is the inverse of distributivity):

$$U(R)\,B\,U(S) \rightarrow U(R\,B\,S)$$

The tables contain in each position a **validity indicator**. A validity indicator "Y" means that the property can always be applied; "N" means that it cannot be applied. For example, the validity indicator "Y" in the first row and first column of Table 5.1 means that the following transformation is correct

$$\mathbf{SL}_{F1}\,\mathbf{SL}_{F2}R \rightarrow \mathbf{SL}_{F2}\mathbf{SL}_{F1}R$$

where $F1$ and $F2$ are two generic selection specifications.

Validity indicators can also be "SNC," specifying a condition which is necessary and sufficient for the application of the property. For example, the validity indicator SNC_1 in the second row and first column of Table 5.1 means that the transformation

$$\mathbf{PJ}_{A1}\mathbf{SL}_{F2}R \rightarrow \mathbf{SL}_{F2}\mathbf{PJ}_{A1}R$$

Table 5.1 Commutativity of unary operations

	$\mathbf{SL}_{F2}$	$\mathbf{PJ}_{A2}$
$\mathbf{SL}_{F1}(*(R)) \rightarrow *(\mathbf{SL}_{F1}(R))$	Y	Y
$\mathbf{PJ}_{A1}(*(R)) \rightarrow *(\mathbf{PJ}_{A1}(R))$	SNC_1	SNC_2

$SNC_1 : Attr(F2) \subseteq A1$

$SNC_2 : A1 \equiv A2$

Table 5.2 Commutativity of operands and associativity of binary operations

	UN	DF	CP	$\mathbf{JN}_F$	$\mathbf{SJ}_F$
$R * S \rightarrow S * R$	Y	N	Y	Y	N
$(R * S) * T \rightarrow R * (S * T)$	Y	N	Y	SNC_1	N

SNC_1 for $(R\,\mathbf{JN}_{F1}\,S)\mathbf{JN}_{F2}\,T \rightarrow R\,\mathbf{JN}_{F1}(S\,\mathbf{JN}_{F2}\,T) : Attr(F2) \subseteq Attr(S) \cup Attr(T)$

Table 5.3 Idempotence of unary operations

$\mathbf{PJ}_A(R) \rightarrow \mathbf{PJ}_{A1}\,\mathbf{PJ}_{A2}(R)$	$SNC : A \equiv A1, A \subseteq A2$
$\mathbf{SL}_F(R) \rightarrow \mathbf{SL}_{F1}\,\mathbf{SL}_{F2}(R)$	$SNC : F = F1 \wedge F2$

In nondistributed databases, general criteria have been given for applying equivalence transformations for the purpose of simplifying the execution of queries:

Criterion 1. Use idempotence of selection and projection to generate appropriate selections and projections for each operand relation.

Criterion 2. Push selections and projections down in the tree as far as possible.

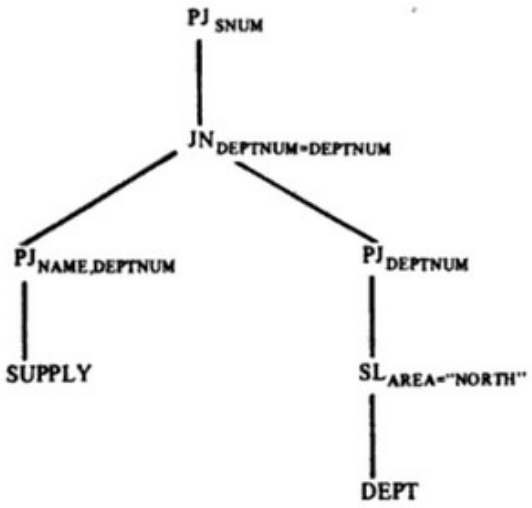


Figure 5.2 A modified operator tree for query Q1.

Figure 5.2 shows a modified operator tree for query Q1, in which the following transformations have been applied:

1. The selection is distributed with respect to the join; thus, the selection is applied directly to the *DEPT* relation.
2. Two new projection operations are generated and are distributed with respect to the join.

Notice that criterion 1 (use of idempotence) has been applied to the projection operation before distributing it; the selection operation, instead, has been directly moved to the *DEPT* relation.

4.2.3 Operator Graph and Determination of Common Sub-Expressions

An important issue in query expression is to discover its common sub-expressions; i.e., sub-expressions which appear more than once in the query; it shows that, saving the time and space if common sub-expressions are evaluated only once.

Ex:

$$Q2 : \mathbf{PJ}_{EMP.NAME}((EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\ \mathbf{SL}_{MGRNUM=373}DEPT)\mathbf{DF}\ (\mathbf{SL}_{SAL>35000}EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\mathbf{SL}_{MGRNUM=373}DEPT))$$

In this, common subexpression is as follows:

$$EMP\ \mathbf{JN}_{DEPTNUM=DEPTNUM}\ \mathbf{SL}_{MGRNUM=373}\ DEPT$$

Once common subexpressions are identified, we can use the following properties to further simplify an operator tree:

$$R\ \mathbf{NJN}\ R \leftrightarrow R$$
$$R\ \mathbf{UN}\ R \leftrightarrow R$$
$$R\ \mathbf{DF}\ R \leftrightarrow \emptyset$$

$$R\ \mathbf{NJN}\ \mathbf{SL}_F\ R \leftrightarrow \mathbf{SL}_F\ R$$
$$R\ \mathbf{UN}\ \mathbf{SL}_F\ R \leftrightarrow R$$
$$R\ \mathbf{DF}\ \mathbf{SL}_F\ R \leftrightarrow \mathbf{SL}_{\mathbf{NOT}\ F}\ R$$

$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{NJN}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathbf{AND}\ F_2}\ R$$
$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{UN}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathbf{OR}\ F_2}\ R$$
$$(\mathbf{SL}_{F_1}\ R)\ \mathbf{DF}\ (\mathbf{SL}_{F_2}\ R) \leftrightarrow \mathbf{SL}_{F_1\ \mathbf{AND\ NOT}\ F_2}\ R$$

4.3 DISTRIBUTED GROUPING AND AGGREGATE FUNCTION EVALUATION

Database applications having database access operations that can be expressed in query languages rather than specifying with relational algebra. Grouping tuples into disjoint subsets of relations and evaluating aggregate functions over them

We first give some examples, using SQL, of queries which use the above features.

Q6: Select AVG(*QUAN*)
from *SUPPLY*
where *PNUM* = "P1"

This query retrieves into a result relation having one attribute and one tuple the average quantity of supply orders for product "P1."

Q7: Select *PNUM*, *SNUM*, SUM(*QUAN*)
from *SUPPLY*
group by *SNUM*, *PNUM*

This query corresponds to partitioning the relation *SUPPLY* into groups having the same value of *SNUM* and *PNUM* (but different values of *DEPTNUM* and *QUAN*), evaluating for each such group the sum of the quantities, and retrieving *SNUM*, *PNUM*, and the sum of the quantities of each group into the result relation.

Q8: Select *SNUM*, *PNUM*, SUM(*QUAN*)
from *SUPPLY*
group by *SNUM*, *PNUM*
having SUM(*QUAN*) > 300

Extensions with Relational Algebra

Relational algebra is extended with the following **group-by** $\mathbf{GB}_{G,AF}\ R$ such that:

- G are the attributes which determine the grouping of R.
- AF are aggregate functions to be evaluated on each group.
- $\mathbf{GB}_{G,AF}R$ is a relation having:
  A relation schema made by the attributes of G and the aggregate functions of AF.

With the above operation, it is possible to write in algebra queries Q6, Q7, and Q8. We have

$$Q6 : \mathbf{GB}_{\text{AVG}(QUAN)}\ \mathbf{SL}_{PNUM=\text{"P1"}}\ SUPPLY$$
$$Q7 : \mathbf{GB}_{SNUM,PNUM,\text{SUM}(QUAN)}\ SUPPLY$$
$$Q8 : \mathbf{SL}_{\text{SUM}(QUANT)>300}\ \mathbf{GB}_{SNUM,PNUM,\text{SUM}(QUAN)}\ SUPPLY$$

Some comments are in order. The G part corresponds to the "group-by" clause, and the AF part corresponds to the aggregate functions whose computation is required. Typically in these queries attributes upon which grouping is made are also retrieved; thus, the attributes which appear in the "group-by" clause also appear in the "select" clause.

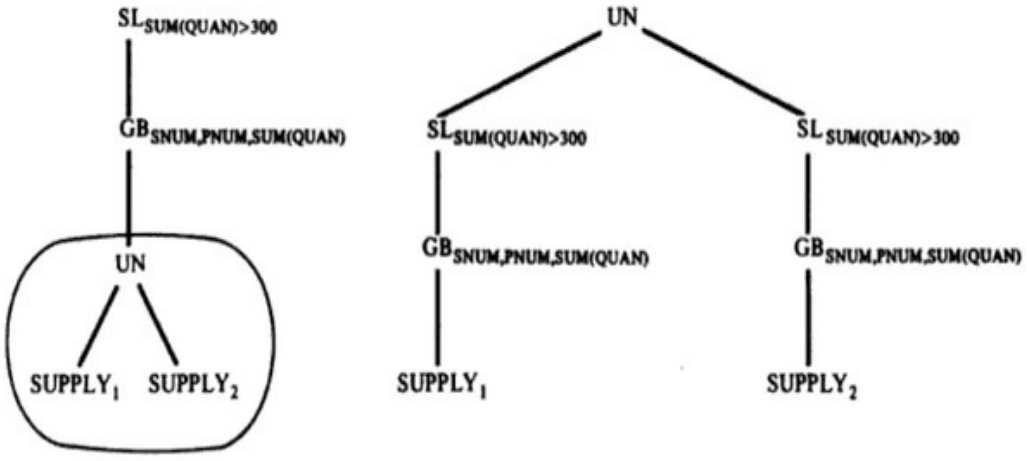
In query Q6, the G part is left empty, as the function is applied to all the tuples of *SUPPLY*. In query Q8, the usual selection operation is applied to the result of the **GB** operation; this selection corresponds to the "having" clause of SQL.

Properties of Group By Operations:

In this section, we give an equivalence property for the new operation, and we discuss, in general, the possibility of evaluating aggregate functions in a distributed way.

The property in which we are interested is the distributivity of **GB** with respect to union:

$$\mathbf{GB}_{G,AF}(R_1\ \mathbf{UN}\ R_2) \rightarrow (\mathbf{GB}_{G,AF}R_1)\mathbf{UN}(\mathbf{GB}_{G,AF}R_2), \qquad \text{Val. ind.: } SNC$$



(*a*) Canonical form of query Q8. (*b*) Distributed version of query Q8.

Figure 5.10 A query with grouping and aggregate functions.

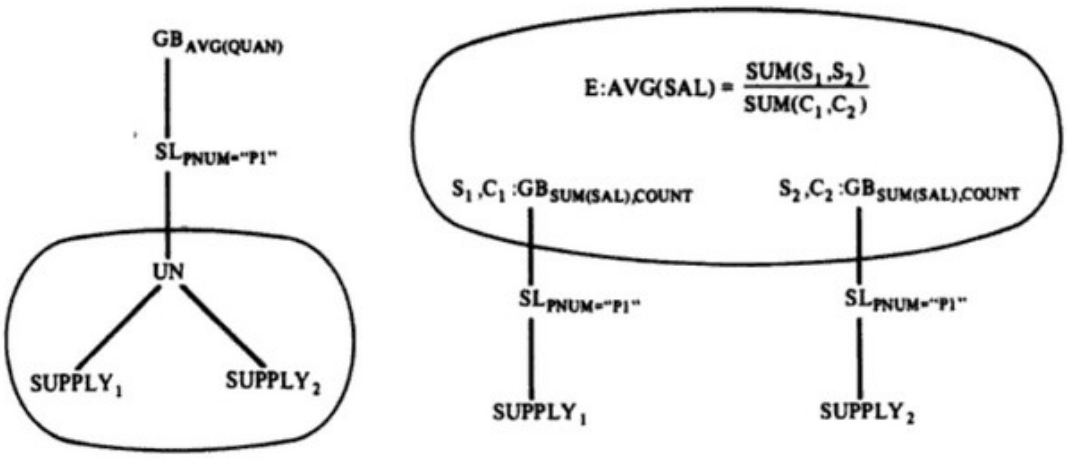
Criterion 6. In order to distribute grouping and aggregate function evaluations appearing in a global query, unions (representing fragment collections) must be pushed up, beyond the corresponding group-by operation.

An aggregate function for which it is possible to find the functions F_i and the expression $E(F_i)$ is the function average

$$\text{AVG}(S) = \frac{\text{SUM}(\text{SUM}(S_1), \text{SUM}(S_2), \ldots, \text{SUM}(S_n))}{\text{SUM}(\text{COUNT}(S_1), \text{COUNT}(S_2), \ldots, \text{COUNT}(S_n))}$$

Similarly, we have

$$\text{MIN}(S) = \text{MIN}(\text{MIN}(S_1), \text{MIN}(S_2), \ldots, \text{MIN}(S_n))$$
$$\text{MAX}(S) = \text{MAX}(\text{MAX}(S_1), \text{MAX}(S_2), \ldots, \text{MAX}(S_n))$$
$$\text{COUNT}(S) = \text{SUM}(\text{COUNT}(S_1), \text{COUNT}(S_2), \ldots, \text{COUNT}(S_n))$$
$$\text{SUM}(S) = \text{SUM}(\text{SUM}(S_1), \text{SUM}(S_2), \ldots, \text{SUM}(S_n))$$



(a) Canonical form of query Q6. (b) Distributed version of query Q6.

Figure 5.11 Distributed evaluation of aggregate functions.

4.4 PARAMETRIC QUERIES

The selection criteria of queries include parameters whose values are not known when the query is compiled. When parametric queries are executed, the user provides values which are bound to parameters, thus parametric queries allow the repeated execution of queries for different values of the parameters and return different values at each execution.

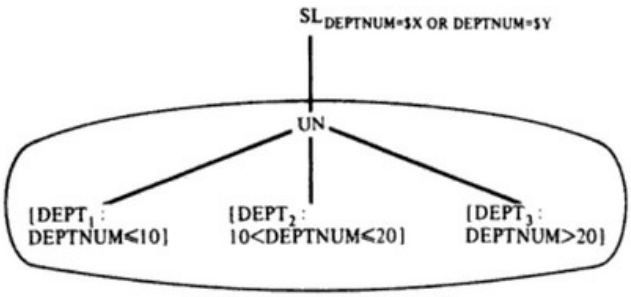
4.4.1 Simplification of Parametric queries and extension of algebra

As an example of a parametric query, let us consider query Q9, selecting tuples of the global relation *DEPT* having given department numbers. Let the selection on *DEPTNUM* be parametric:

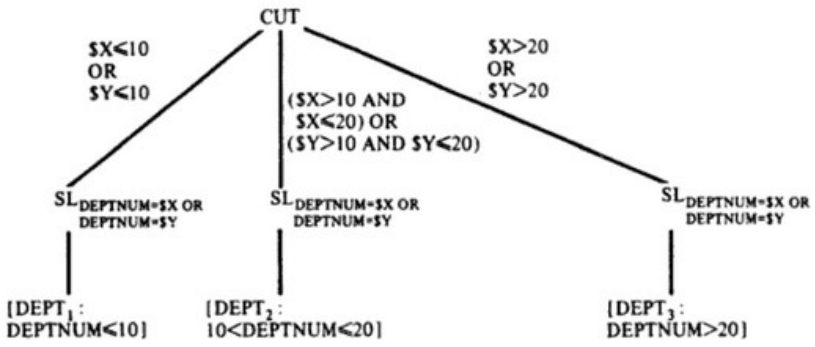
$$\text{Q9} : \text{SL}_{DEPTNUM=\$X \text{ OR } DEPTNUM=\$Y}\ SUPPLY$$

At run time, the actual values are assigned to the parameters $X and $Y by the program which issues the query.

The simplification of parametric queries has some distinguishing features. Consider the above query, whose canonical form is shown in Figure 5.12a.



(*a*) Canonical form of query Q9.



(*b*) Query tree with the CUT operation.

Figure 5.12 Use of CUT in a parametric query.

time, we don't know which of the fragments of the global relation *DEPT* will be addressed. However, we know that *at most two* of them will be involved in the query. At run time, when the query will be activated, it will also be possible to know, given the value of the parameters, which ones of the fragments will be involved in the query. This shows that part of the simplification of a parametric query can be done at compile time, but part of it needs to be done at run time.

Figure 5.12*b* shows the new operator tree for the parametric query. In this example, the selection is pushed below the union operation, applying criterion 2, and then the union is substituted by the **CUT** operation. The three formulas for the **CUT** operation (one for each operand of the union operation) are

$$F1 : \$X \leq 10 \text{ OR } \$Y \leq 10$$
$$F2 : (\$X > 10 \text{ AND } \$X \leq 20) \text{ OR } (\$Y > 10 \text{ AND } \$Y \leq 20)$$
$$F3 : \$X > 20 \text{ OR } \$Y > 20$$

At run time, the **CUT** operation is the first one to be evaluated; the query, for $\$X = 1$ and $\$Y = 13$, reduces to the first and second branch of the operator tree; for $\$X = 1$ and $\$Y = 5$, the query reduces to the first branch of the tree.

4.4.2 Using temporaries in Multiple Activations of Parametric Queries

For the case of repeated executions and lower the costs, it might be useful to build temporary relations at the site of origin of the query, which store a superset of the data required at each iteration.

$$Q10 : \mathbf{PJ}_{NAME}\,\mathbf{SL}_{MGRNUM=\$X \text{ AND } DEPTNUM=12}\,EMP$$

First, we modify the query so that the operator tree of Figure 5.13a is obtained. The general goal of this transformation is to insulate subtrees which are **not parametric**, as they will constitute the required temporaries.

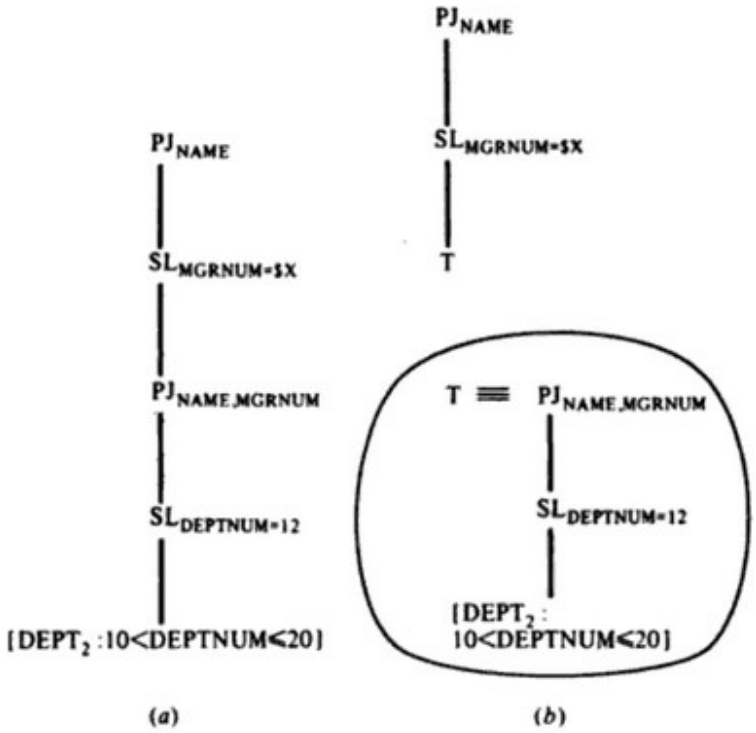


Figure 5.13 Use of temporary relations for parametric queries.

$$\mathbf{PJ}_{NAME,MGRNUM}\,\mathbf{SL}_{DEPTNUM=12}\,EMP$$

We then call T the temporary given by the above expression, and divide the operator tree into two pieces in correspondence to T (see Figure 5.13*b*). T represents the temporary relation that will be used for repeated executions of query Q10.