General Transformation Rules for Relational Algebra Operations.

There are many rules for transforming relational algebra operations into equivalent ones. Here we are interested in the meaning of the operations and the resulting relations. Hence, if two relations have the same set of attributes in a *different order* but the two relations represent the same information, we consider the relations equivalent. In Section 7.1.2 we gave an alternative definition of *relation* that makes order of attributes unimportant; we will use this definition here. We now state some transformation rules, without proving them:

1. Cascade of σ: A conjunctive selection condition can be broken up into a cascade (sequence) of individual s operations:

   $\sigma_{c1 \text{ AND } c2 \text{ AND } \ldots \text{ AND } cn}(R) + \sigma_{c1} (\sigma_{c2} (\ldots(\sigma_{cn}(R))\ldots))$

2. Commutativity of σ: The σ operation is commutative:

   $\sigma_{c1} (\sigma_{c2}(R)) + \sigma_{c2} (\sigma_{c1}(R))$

3. Cascade of π: In a cascade (sequence) ofp operations, all but the last one can be ignored:

   $\pi_{List1} (\pi_{List2} (\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List1}(R)$

4. Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:

   $\pi_{A1, A2, \ldots, An} (\sigma_c (R)) [\sigma_c (\pi_{A1, A2, \ldots, An} (R))$

5. Commutativity of ⋈ (or ∞): The ⋈ operation is commutative:

   $R \bowtie_c S + S \bowtie_c R$

   Notice that, although the order of attributes may not be the same in the relations resulting from the two joins, the "meaning" is the same because order of attributes is not important in the alternative definition of *relation* that we use here. The ∞ operation is commutative in the same sense as the ⋈ operation.

6. Commuting σ with ⋈ (or ∞): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:

   $\sigma_c (R \bowtie S) + (\sigma_c (R)) \bowtie S$

   Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:

   $\sigma_c (R \bowtie S) + (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$

The same rules apply if the ⋈ is replaced by a ∞ operation. These transformations are very useful during heuristic optimization.

7. Commuting π with ⋈ (or ∞): Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

   $\pi_L (R \bowtie_c S) + (\pi_{A1, ..., An} (R)) \bowtie_c (\pi_{B1, ..., Bm} (S))$

   If the join condition c contains additional attributes not in L, these must be added to the projection list, and a final π operation is needed. For example, if attributes An + 1, ..., An + k of R and Bm + 1, ..., Bm + p of S are involved in the join condition c but are not in the projection list L, the operations commute as follows:

   $\pi_L (R \bowtie_c S) +$

   $\pi_L ((\pi_{A1,. . .,An,An+1,. . .,An+k}(R)) \bowtie_c (\pi_{B1,...,Bm,Bm+1,...,Bm+p} (S)))$

   For ∞, there is no condition c, so the first transformation rule always applies by replacing $\bowtie_c$ with ∞.

8. Commutativity of set operations: The set operations ≈ and ↔ are commutative but – is not.

9. Associativity of ⋈, ∞, ≈, and ↔: These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have

   $(R \theta S) \theta T + R \theta (S \theta T)$

10. Commuting σ with set operations: The σ operation commutes with ≈, ↔, and –. If θ stands for any one of these three operations, we have

    $\sigma_c (R \theta S) + (\sigma_c (R)) \theta (\sigma_c (S))$

11. The π operation commutes with ≈. If θ stands for ≈, we have

    $\pi_L (R \theta S) + (\pi_L (R)) \theta (\pi_L (S))$

12. Other transformations: There are other possible transformations. For example, a selection or join condition c can be converted into an equivalent condition by using the following rules (known as DeMorgan's laws):

    c ≡ **NOT** (c1 **AND** c2) ≡ (**NOT** c1) **OR** (**NOT** c2)

    c ≡ **NOT** (c1 **OR** c2) ≡ (**NOT** c1) **AND** (**NOT** c2)

Additional transformations discussed in Chapter 7 are not repeated here. We discuss next how these rules are used in heuristic optimization.

Outline of a Heuristic Algebraic Optimization Algorithm.

We can now outline the steps of an algorithm that utilizes some of the above rules to transform an initial query tree into an optimized tree that is more efficient to execute (in most cases). The steps of the algorithm will lead to transformations similar to those discussed in our example of Figure 18.4. The steps of the algorithm are as follows:

1. Using rule 1, break up any SELECT operations with conjunctive conditions into a cascade of SELECT operations. This permits a greater degree of freedom in moving select operations down different branches of the tree.
2. Using rules 2, 4, 6, and 10 concerning the commutativity of SELECT with other operations, move each SELECT operation as far down the query tree as is permitted by the attributes involved in the select condition.
3. Using rule 9 concerning associativity of binary operations, rearrange the leaf nodes of the tree so that the leaf node relations with the most restrictive SELECT operations are executed first in the query tree representation. By "most restrictive SELECT operations," we mean the ones that produce a relation with the fewest tuples or with the smallest absolute size. Either definition can be used, since these rules are heuristic. Another possibility is to define the most restrictive SELECT as the one with the smallest selectivity; this is more practical because estimated selectivities are often available in the catalog.
4. Combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation whose condition represents a join condition into a JOIN operation.
5. Using rules 3, 4, 7, and 11 concerning the cascading of PROJECT and the commuting of PROJECT with other operations, break down and move lists of projection attributes down the tree as far as possible by creating new PROJECT operations as needed.
6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.

In our example, Figure 18.4(b) shows the tree of Figure 18.4(a) after applying steps 1 and 2 of the algorithm; Figure 18.4(c) shows the tree after applying step 3; Figure 18.4(d) after applying step 4; and Figure 18.4(e) after applying step 5. In step 6 we may group together the operations in the subtree whose root is the operation $\bowtie_{\text{PNUMBER=PNO}}$ into a single algorithm. We may also group the remaining operations into another subtree, where the tuples resulting from the first algorithm replace the subtree whose root is the operation $\bowtie_{\text{PNUMBER=PNO}}$, because the first grouping means that this subtree is executed first.

Summary of Heuristics for Algebraic Optimization.

We now summarize the basic heuristics for algebraic optimization. The main heuristic is to apply first the operations that reduce the size of intermediate results. This includes performing SELECT operations as early as possible to reduce the number of tuples and performing PROJECT operations as early as possible to reduce the number of attributes. This is done by moving SELECT and PROJECT operations as far down the tree as possible. In addition, the SELECT and JOIN operations that are most restrictive—that is, result in relations with the fewest tuples or with the smallest absolute size—should be executed before other similar operations. This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.