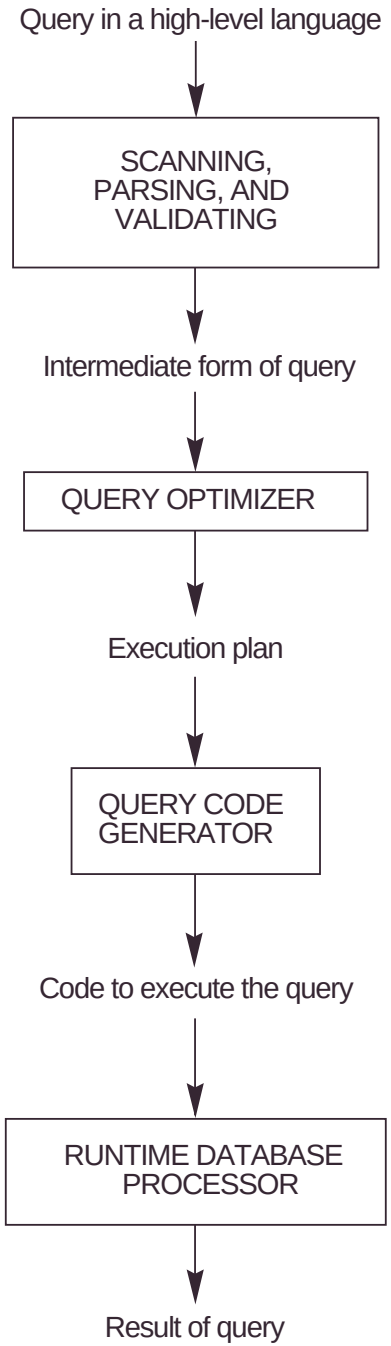
Figure 18.1 Typical steps when processing a high-level query.



Code can be:

- Executed directly (interpreted mode)
- Stored and executed later whenever needed (compiled mode)

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.2 Outline of the sort-merge algorithm for external sorting.

```
set     i ← 1;
        j ← b;      {size of the file in blocks}
        k ← n_B;    {size of buffer in blocks}
        m ← ⌈(j/k)⌉;
{Sort Phase}
while (i <= m)
        do {
            read next k blocks of the file into the buffer or if there are less than k blocks remaining,
                then read in the remaining blocks;
            sort the blocks in the buffer and write as a temporary subfile;
            i ← i + 1;
            }
{Merge Phase: merge subfiles until only 1 remains}
set     i ← 1;
        p ← log_{k–1} m;   {p is the number of passes for the merging phase}
        j ← m;
while (i <= p)
        do {
            n ← 1;
            q ← ⌈(j/ (k–1))⌉ ;   {number of subfiles to write in this pass}
            while (n <= q)
                do {
                    read next k–1 subfiles or remaining subfiles (from previous pass) one block at a time;
                    merge and write as new subfile;
                    n ← n + 1;
                    }
            j ← q;
            i ← i + 1;
                    }
```

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.3 Implementing JOIN, PROJECT, UNION, INTERSECTION, and SET DIFFERENCE by using sort-merge, where R has n tuples and S has m tuples. (a) Implementing the operation $T \leftarrow R \bowtie_{A=B} S$. (b) Implementing the operation $T \leftarrow \pi_{<\text{attribute list}>}(R)$.

```
(a)  sort the tuples in R on attribute A;  (*assume R has n tuples (records) *)
     sort the tuples in S on attribute B;  (*assume S has m tuples (records) *)
     set i ← 1, j ← 1;
     while (i ≤ n) and (j ≤ m)
     do{    if R(i)[A] > S(j)[B]
                then        set j ← j+1
            elseif R(i)[A] < S(j)[B]
                then        set i ← i+1
                else  {     (* R(i)[A] < S(j)[B], so we output a matched tuple*)
                            output the combined tuple <R(i), S(j)> to T;
                                    (*output other tuples that match R(i), if any*)
                                    set l ← j+1;
                                    while (l ≤ m) and (R(i)[A] < S(l)[B])
                                    do    {         output the combined tuple <R(i), S(l)> to T;
                                                    set l ← l+1
                                          }
                                    (*output other tuples that match S(j), if any*)
                                    set k ← i+1;
                                    while (k ≤ n) and (R(k)[A] < S(j)[B])
                                    do    {         output the combined tuple <R(k), S(j)> to T;
                                                    set k ← k+1
                                    }
                                   set i ← k, j ← l
                }
     }

(b)  create a tuple t[<attribute list>] in T' for each tuple t in R;
            (*T' contains the projection result before duplicate elimination*)
     if <attribute list> includes a key of R
            the T ← T'
            else    {     sort the tuples in T';
                          set i ← 1, j ← 2;
                          while i ≤ n
                                do    {     output the tuple T'[i] to T;
                                            while T'[i] = T'[j]  and j ≤ n do j ← j+1; (*eliminate duplicates*)
                                            i ← j; j ← i+1
                                      }
                    }
     (* T contains the projection result after duplicate elimination *)
```

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.3 Implementing JOIN, PROJECT, UNION, INTERSECTION, and SET DIFFERENCE by using sort-merge, where R has n tuples and S has m tuples. (c) Implementing the operation T $\leftarrow R \cup$ S. (d) Implementing the operation T $\leftarrow$ R $\cap$ S. (e) Implementing the operation T $\leftarrow R - S$.

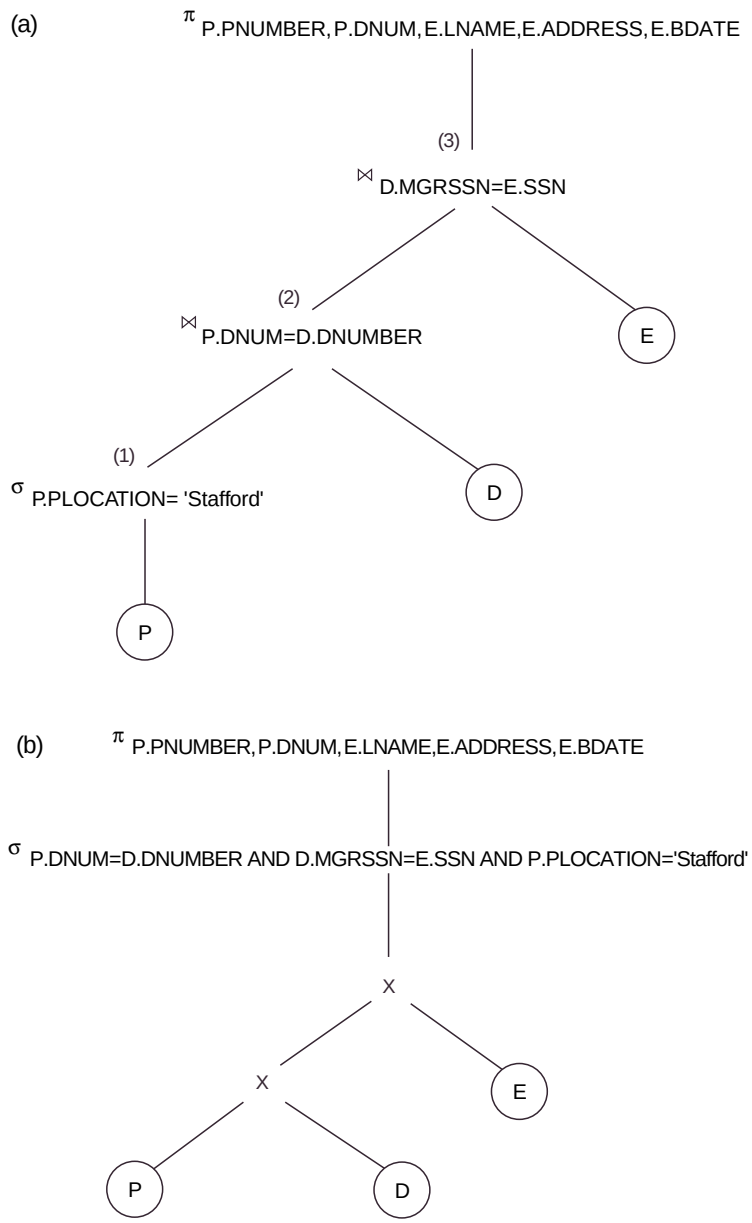
```
(c)  sort the tuples in R and S using the same unique sort attributes;
     set i ←1, j ←1;
     while (i ≤ n) and (j ≤ m)
     do   {     if R(i) > S(j)
                then   {     output S(j) to T;
                             set j←j+1
                       }
                elseif R(i) > S(j)
                then   {     output R(i) to T;
                             set i←i+1
                       }
                else set j←j+1 (*R(i)=S(j), so we skip one of the duplicate tuples*)
          }
     if (i ≤ n) then add tuples R(i) to R(n) to T;
     if (j ≤ m) then add tuples S(j) to S(m) to T;

(d)  sort the tuples in R and R using the same unique sort attributes;
     set i←1, j←1;
     while (i ≤ n) and (j ≤ m)
     do   {     if R(i) > S(j)
                then          set j←j+1
                elseif R(i) < S(j)
                then          set i←i+1
                else   {      output R(i) to T; (* R(i)=S(j), so we output the tuple *)
                              set i←i+1, j←j+1
                       }
          }

(e)  sort the tuples in R and S using the same unique sort attributes;
     set i←1, j←1;
     while (i ≤ n) and (j ≤ m)
     do   {     if R(i) > S(j)
                then          set j←j+1
                elseif R(i) < S(j)
                then   {      output R(i)to T; (* R(i) has no matching S(j), so output R(i)*)
                              set i←i+1
                       }
                else          set i←i+1, j←j+1
          }
     if (i ≤ n) then add tuples R(i) to R(n) to T;
```

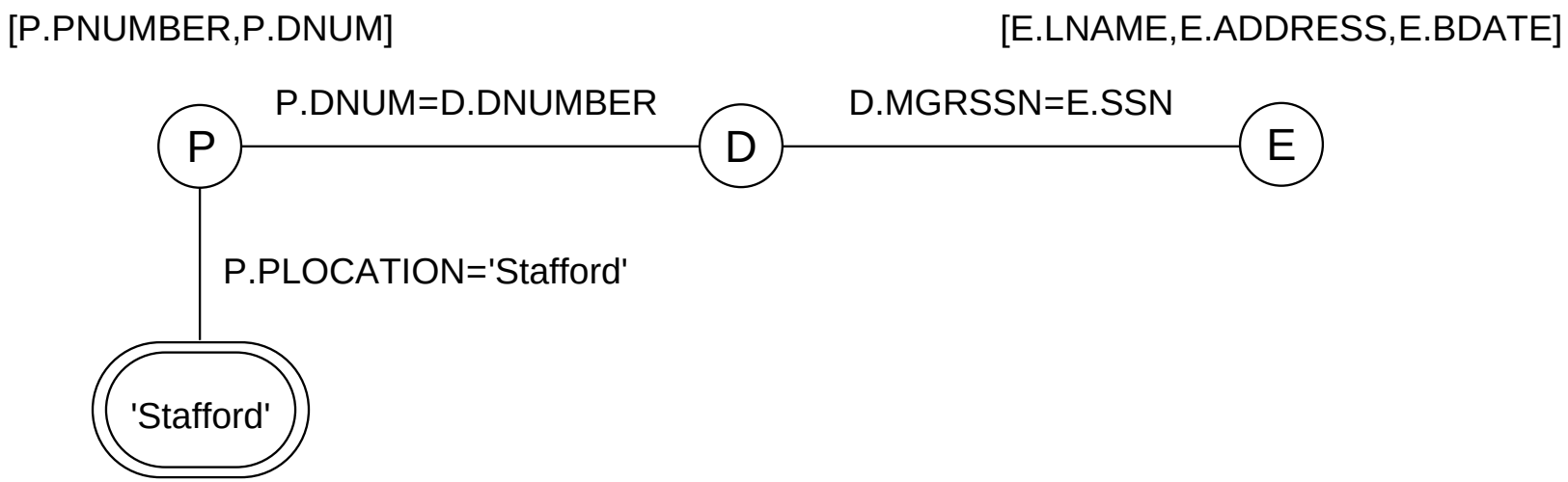
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.4 Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2.



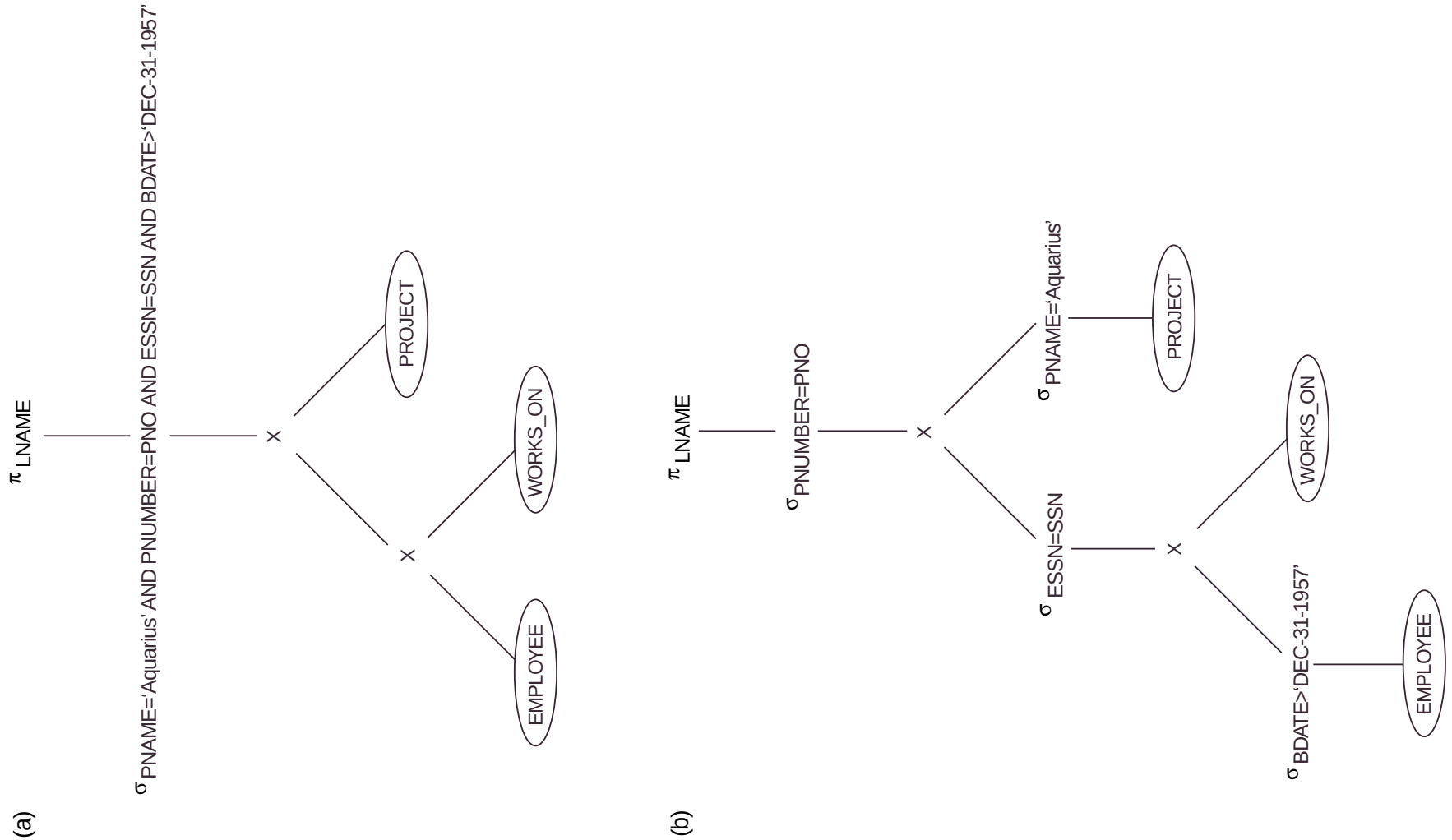
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.4 (c) Query graph for Q2.



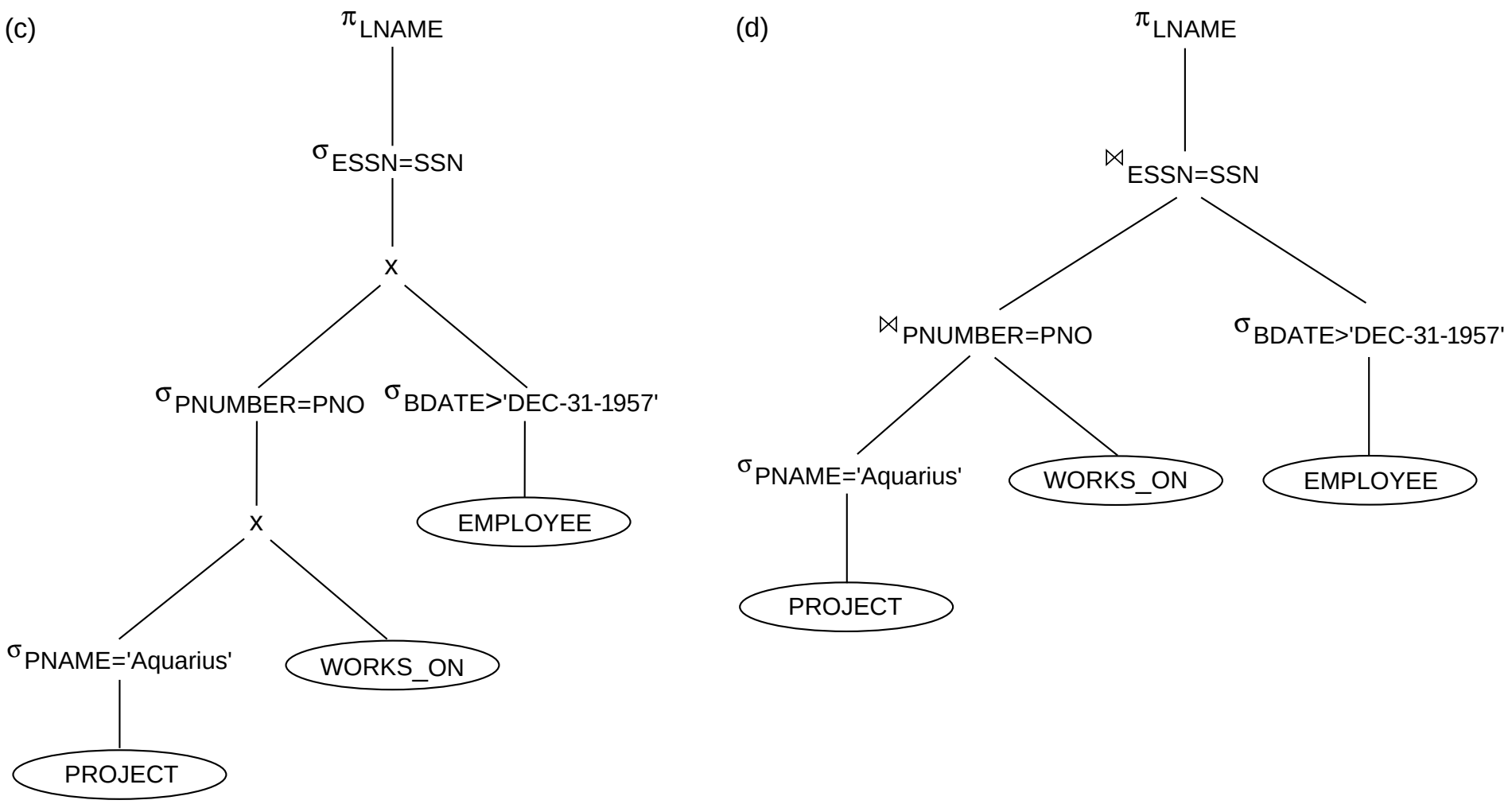
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.5 Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q. (b) Moving SELECT operations down the query tree.



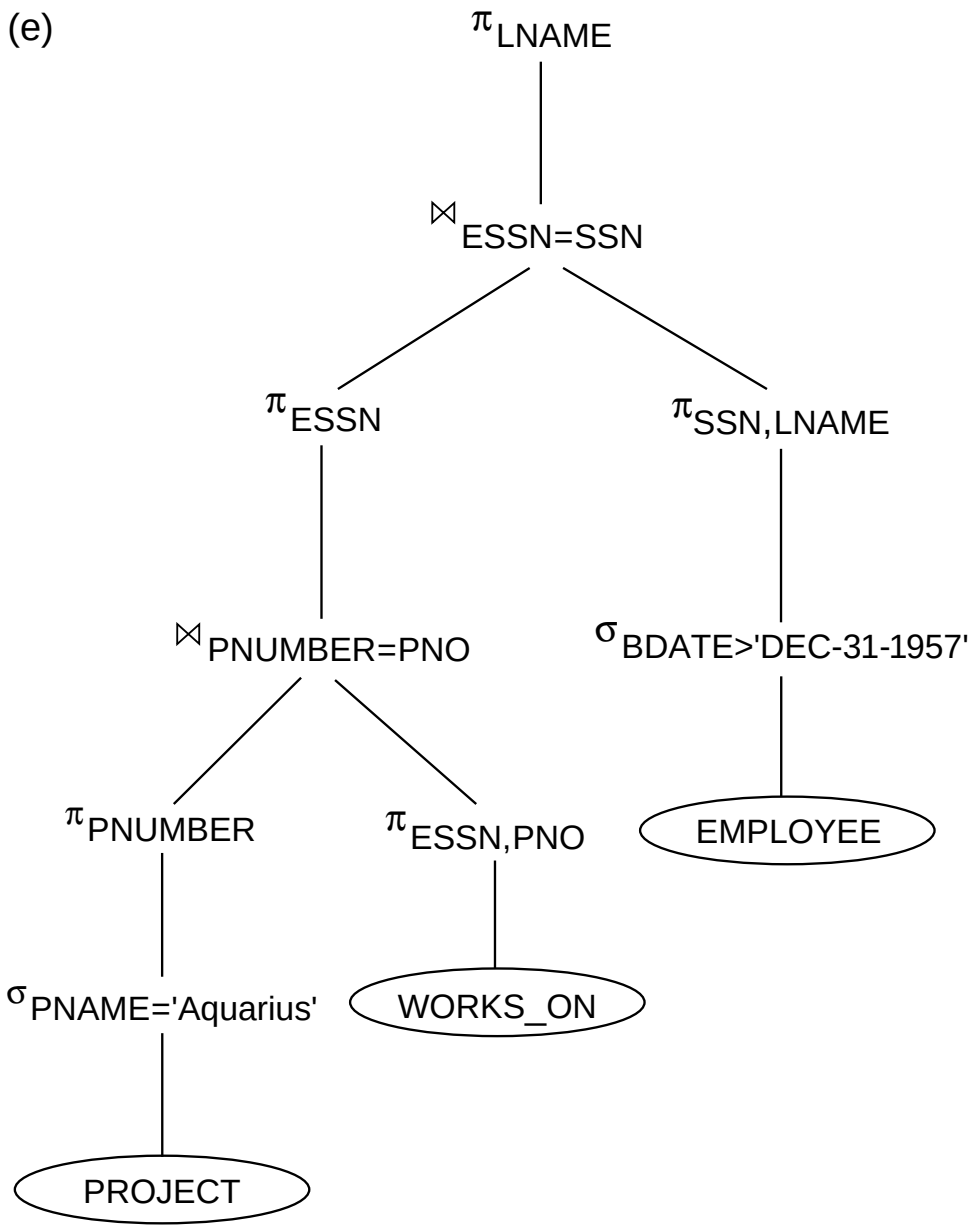
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.5 Steps in converting a query tree during heuristic optimization. (c) Applying the more restrictive SELECT operation first. (d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.



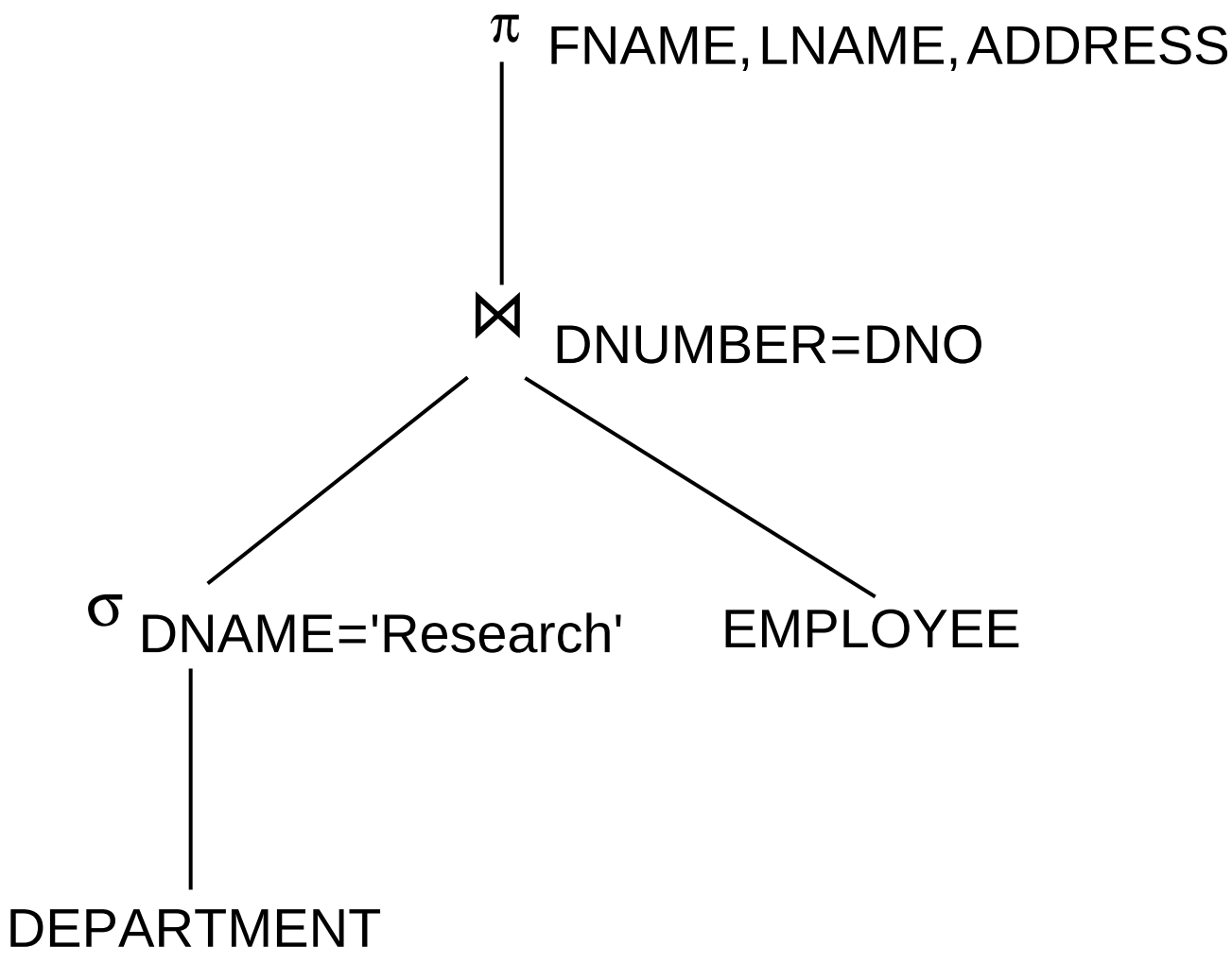
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.5 Steps in converting a query tree during heuristic optimization. (e) Moving PROJECT operations down the query tree.



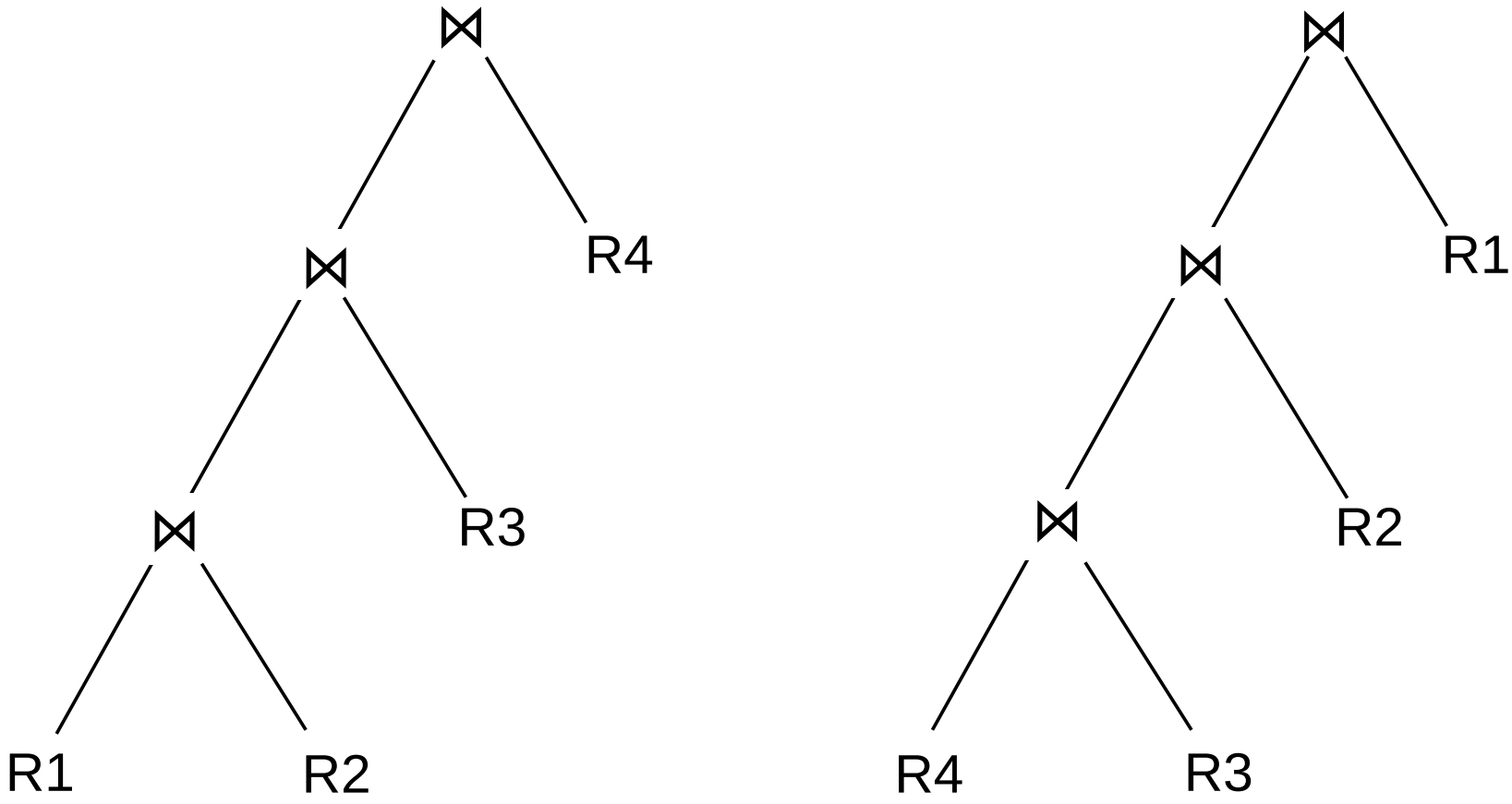
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.6 A query tree for query Q1.



© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.7 Two left-deep (join) query trees.



© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 18.8 Sample statistical information for relations in Q2. (a) Column information. (b) Table information. (c) Index information.

(a)

TABLE_NAME	COLUMN_NAME	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
PROJECT	PLOCATION	200	1	200
PROJECT	PNUMBER	2000	1	2000
PROJECT	DNUM	50	1	50
DEPARTMENT	DNUMBER	50	1	50
DEPARTMENT	MGRSSN	50	1	50
EMPLOYEE	SSN	10000	1	10000
EMPLOYEE	DNO	50	1	50
EMPLOYEE	SALARY	500	1	500

(b)

TABLE_NAME	NUM_ROWS	BLOCKS
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

(c)

INDEX_NAME	UNIQUENES	BLEVEL*	LEAF_ BLOCKS	DISTINCT _KEYS
PROJ_PLOC	NONUNIQUE	1	4	200
EMP_SSN	UNIQUE	1	50	10000
EMP_SAL	NONUNIQUE	1	50	500

*BLEVEL is the number of levels without the leaf level.

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition