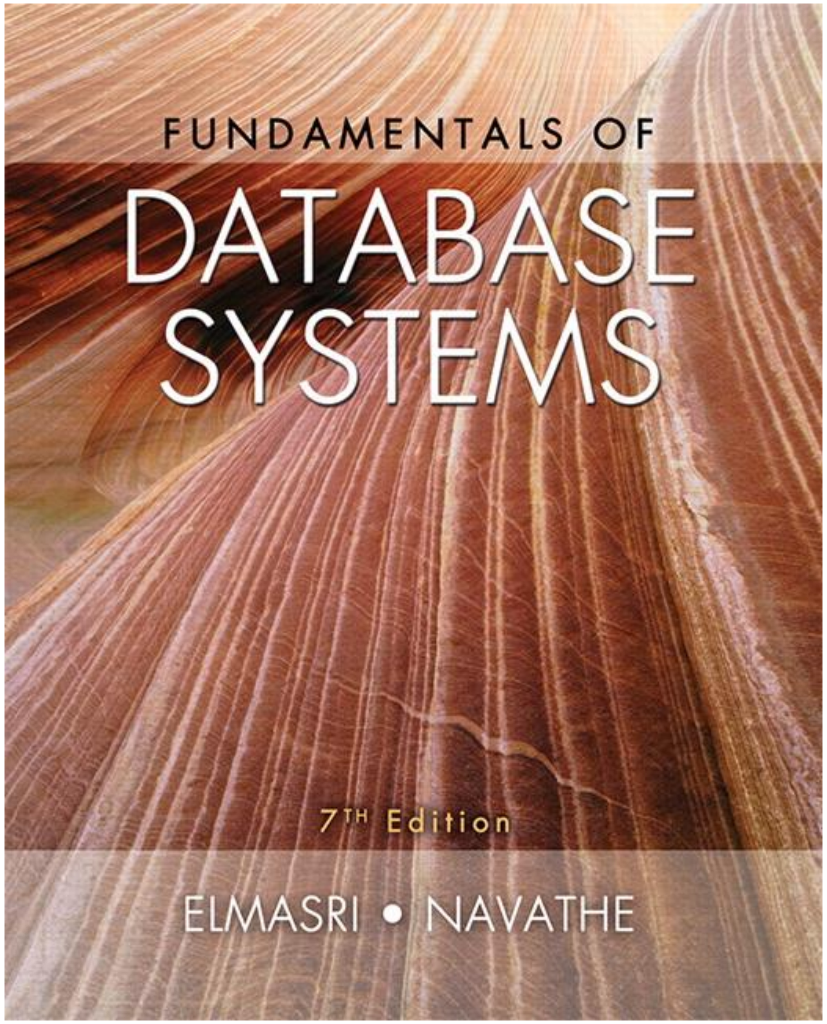

FUNDAMENTALS OF
DATABASE SYSTEMS
7TH Edition
ELMASRI • NAVATHE

CHAPTER 19

Query Optimization

Introduction

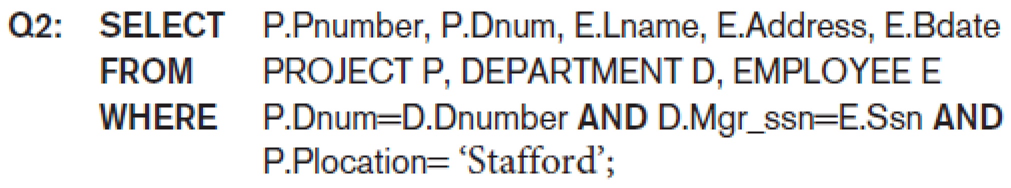
- Query optimization
  - Conducted by a query optimizer in a DBMS
  - Goal: select best available strategy for executing query
    - Based on information available
- Most RDBMSs use a tree as the internal representation of a query

19.1 Query Trees and Heuristics for Query Optimization

- Step 1: scanner and parser generate initial query representation
- Step 2: representation is optimized according to heuristic rules
- Step 3: query execution plan is developed
  - Execute groups of operations based on access paths available and files involved

Query Trees and Heuristics for Query Optimization (cont'd.)

- Example heuristic rule
  - Apply SELECT and PROJECT before JOIN
    - Reduces size of files to be joined
- Query tree
  - Represents relational algebra expression
- Query graph
  - Represents relational calculus expression
- Example for Q2 on next slide



```
Q2:  SELECT   P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
     FROM     PROJECT P, DEPARTMENT D, EMPLOYEE E
     WHERE    P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND
              P.Plocation= 'Stafford';
```

Query Trees and Query Graph Corresponding to Q2

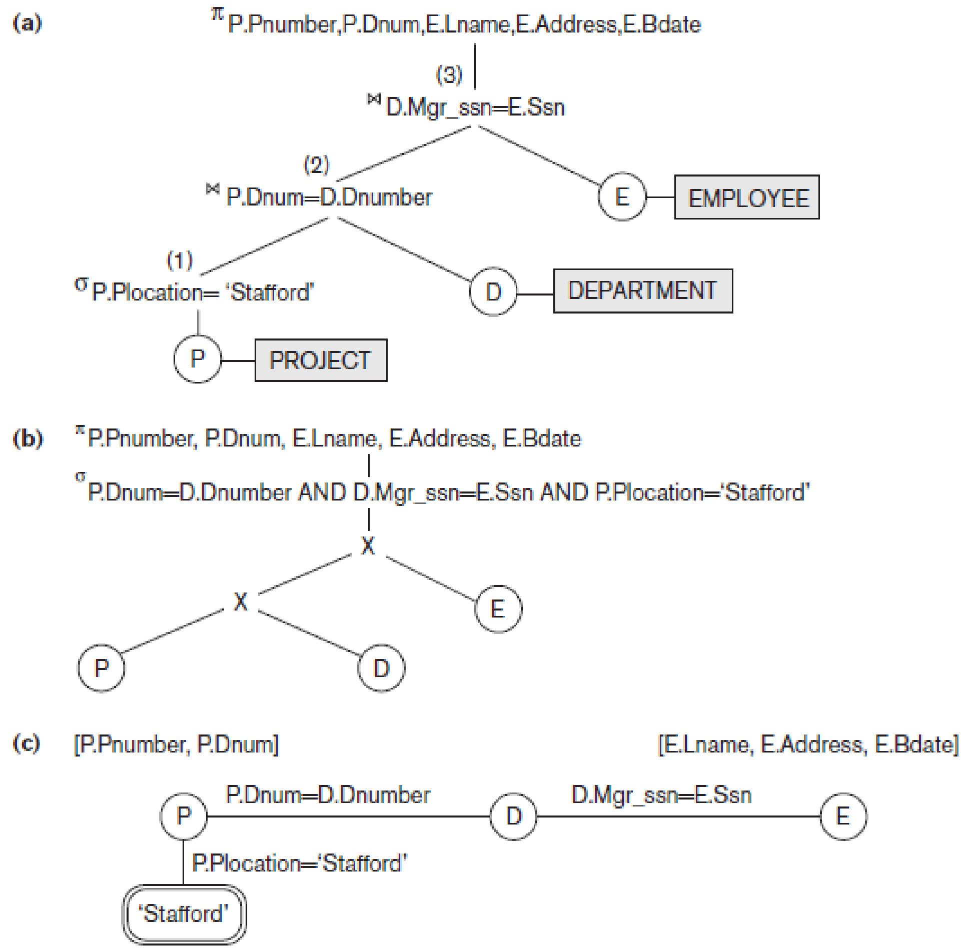

(a)
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
(3)
$\bowtie$ D.Mgr_ssn=E.Ssn
(2)
$\bowtie$ P.Dnum=D.Dnumber
E
EMPLOYEE
(1)
σ P.Plocation= 'Stafford'
D
DEPARTMENT
P
PROJECT
(b)
π P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
σ P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation='Stafford'
X
X
E
P
D
(c)
[P.Pnumber, P.Dnum]
[E.Lname, E.Address, E.Bdate]
P.Dnum=D.Dnumber
D.Mgr_ssn=E.Ssn
P
D
E
P.Plocation='Stafford'
'Stafford'

Figure 19.1 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. (c) Query graph for Q2.

Query Trees and Heuristics for Query Optimization (cont'd.)

- Query tree represents a specific order of operations for executing a query
  - Preferred to query graph for this reason
- Query graph
  - Relation nodes displayed as single circles
  - Constants represented by constant nodes
    - Double circles or ovals
  - Selection or join conditions represented as edges
  - Attributes to be retrieved displayed in square brackets

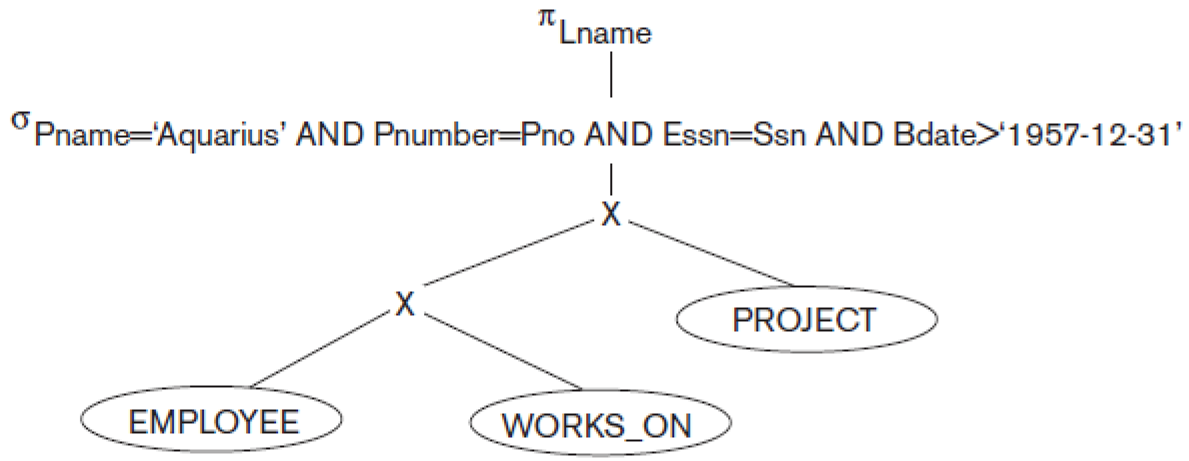
Heuristic Optimization of Query Trees

- Many different query trees can be used to represent the query and get the same results
- Figure 19.1b shows initial tree for Q2
  - Very inefficient - will never be executed
  - Optimizer will transform into equivalent final query tree

Query Transformation Example

```
Q:    SELECT   E.Lname
      FROM     EMPLOYEE E, WORKS_ON W, PROJECT P
      WHERE    P.Pname='Aquarius' AND P.Pnumber=W.Pno AND E.Essn=W.Ssn
               AND E.Bdate > '1957-12-31';
```

(a)



π_{Lname}
$\sigma_{\text{Pname='Aquarius' AND Pnumber=Pno AND Essn=Ssn AND Bdate>'1957-12-31'}}$
X
X
PROJECT
EMPLOYEE
WORKS_ON

Figure 19.2 Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q.

Query Transformation Example (cont'd.)

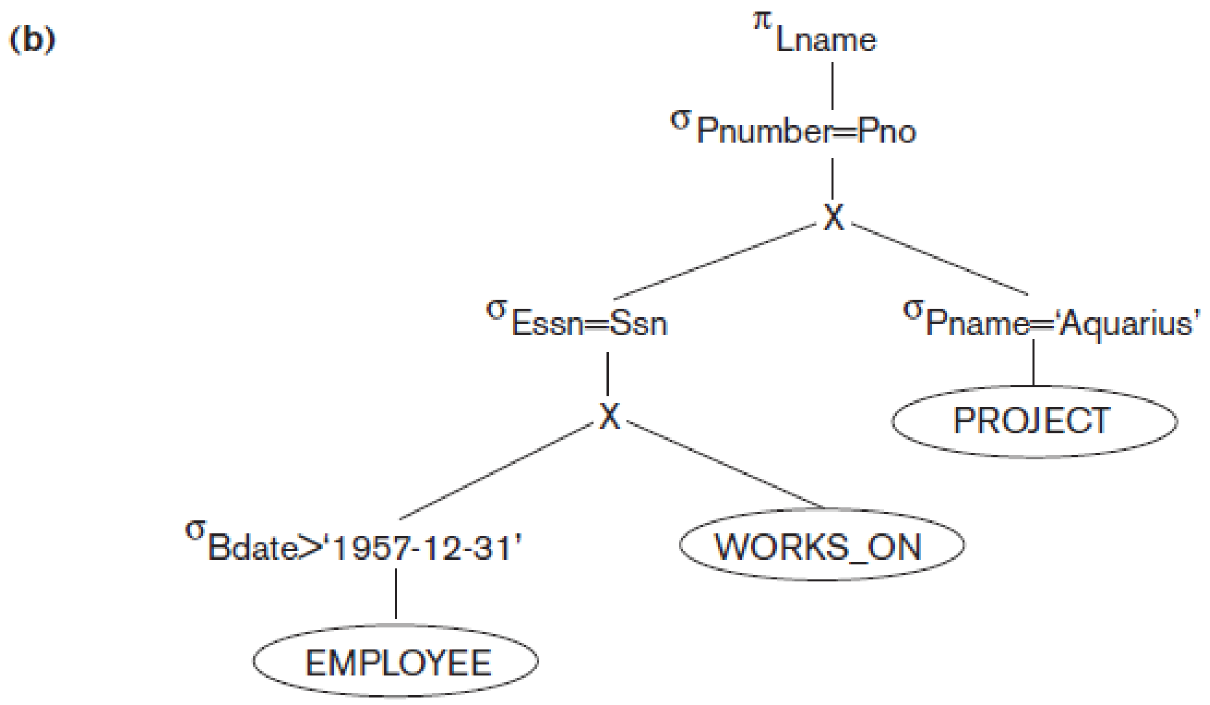

(b)
π_{Lname}
$\sigma_{\text{Pnumber=Pno}}$
X
$\sigma_{\text{Essn=Ssn}}$
$\sigma_{\text{Pname='Aquarius'}}$
X
PROJECT
$\sigma_{\text{Bdate>'1957-12-31'}}$
WORKS_ON
EMPLOYEE

Figure 19.2 Steps in converting a query tree during heuristic optimization
(b) Moving SELECT operations down the query tree.

Query Transformation Example (cont'd.)

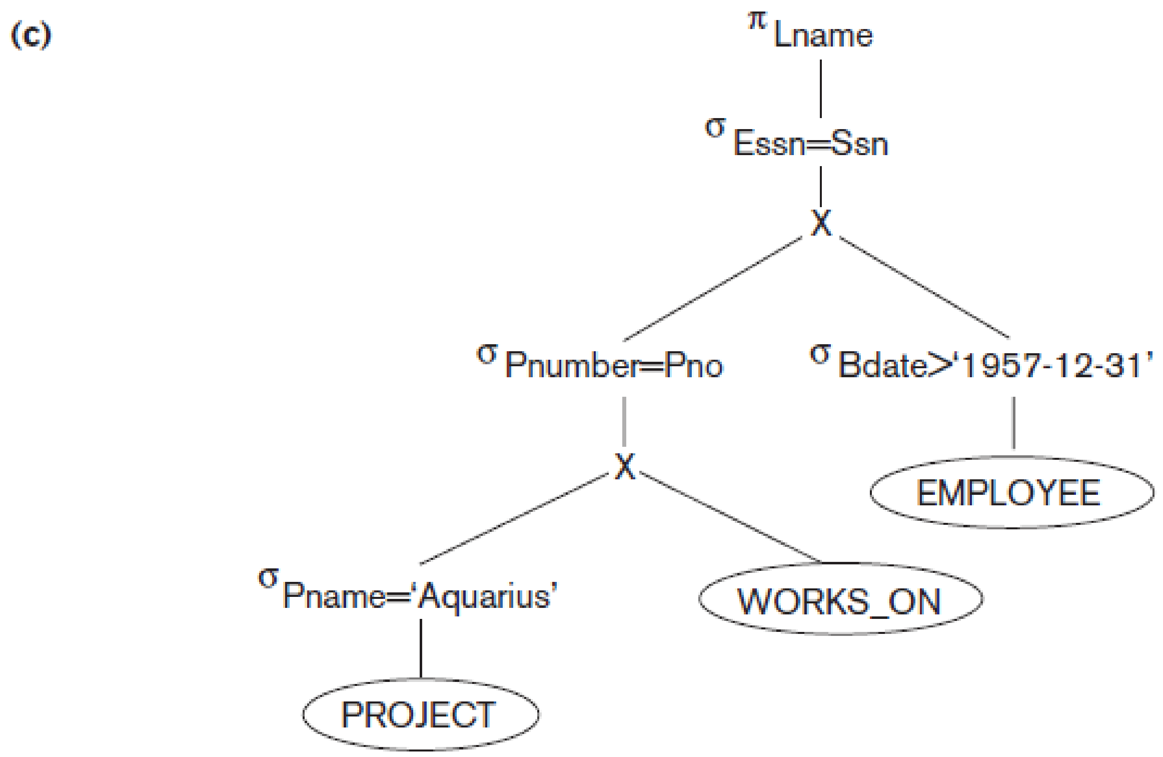


Figure 19.2 Steps in converting a query tree during heuristic optimization (c) Applying the more restrictive SELECT operation first.

Query Transformation Example (cont'd.)

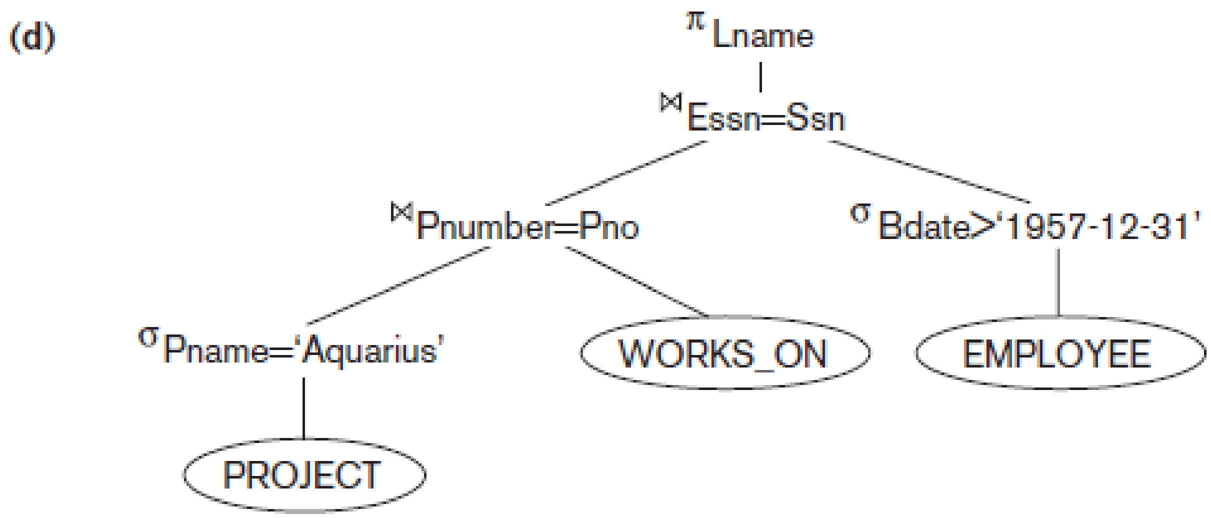


Figure 19.2 Steps in converting a query tree during heuristic optimization
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.

Query Transformation Example (cont'd.)

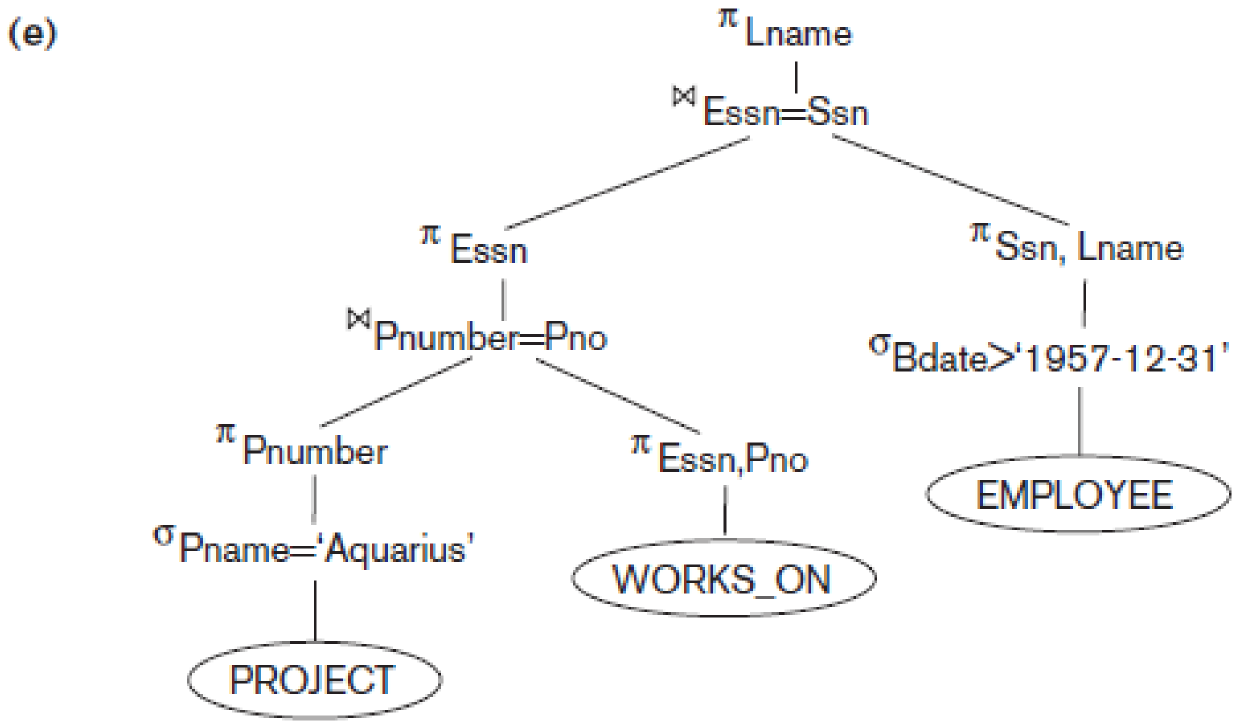


Figure 19.2 Steps in converting a query tree during heuristic optimization
(e) Moving PROJECT operations down the query tree.

General Transformation Rules for Rational Algebra Equations

- Some transformation rules useful in query optimization
  - Cascade of σ
    - Conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations
  - Commutativity of σ
  - Cascade of π
    - In a sequence of π operations, all but the last one can be ignored
  - Commuting σ with π

Summary of Heuristics for Algebraic Optimization

- Apply first the operations that reduce the size of intermediate results
  - Perform SELECT and PROJECT operations as early as possible to reduce the number of tuples and attributes
  - The SELECT and JOIN operations that are most restrictive should be executed before other similar operations

19.2 Choice of Query Execution Plans

- Materialized evaluation
  - Result of an operation stored as temporary relation
- Pipelined evaluation
  - Operation results forwarded directly to the next operation in the query sequence

Nested Subquery Optimization

- Unnesting
  - Process of removing the nested query and converting the inner and outer query into one block
- Queries involving a nested subquery connected by IN or ANY connector can always be converted into a single block query
- Alternate technique
  - Creating temporary result tables from subqueries and using them in joins

Subquery (View) Merging Transformation

- Inline view
  - FROM clause subquery
- View merging operation
  - Merges the tables in the view with the tables from the outer query block
  - Views containing select-project-join operations are considered simple views
    - Can always be subjected to this type of view-merging

Subquery (View) Merging Transformation (cont'd.)

- Group-By view-merging
  - Delaying the Group By operation after performing joins may reduce the data subjected to grouping in case the joins have low join selectivity
  - Alternately, performing Group By early may reduce the amount of data subjected to subsequent joins
  - Optimizer determines whether to merge GROUP-BY views based on estimated costs

Materialized Views

- View defined in database as a query
  - Materialized view stores results of that query
    - May be stored temporarily or permanently
- Optimization technique
  - Using materialized views to avoid some of the computation involved in the query
  - Easier to read it when needed than recompute from scratch

Incremental View Maintenance

- Update view incrementally by accounting for changes that occurred since last update
  - Join
  - Selection
  - Projection
  - Intersection
  - Aggregation

19.3 Use of Selectives in Cost-Based Optimization

- Query optimizer estimates and compares costs of query execution using different strategies
  - Chooses lowest cost estimate strategy
  - Process suited to compiled queries
- Interpreted queries
  - Entire process occurs at runtime
  - Cost estimate may slow down response time

Use of Selectives in Cost-Based Optimization (cont'd.)

- Cost-based query optimization approach
  - For a given query subexpression, multiple equivalence rules may apply
  - Quantitative measure for evaluating alternatives
    - Cost metric includes space and time requirements
  - Design appropriate search strategies by keeping cheapest alternatives and pruning costlier alternatives
  - Scope of query optimization is a query block
    - Global query optimization involves multiple query blocks

Use of Selectives in Cost-Based Optimization (cont'd.)

- Cost components for query execution
  - Access cost to secondary storage
  - Disk storage cost
  - Computation cost
  - Memory usage cost
  - Communication cost

Catalog Information Used in Cost Functions

- Information stored in DBMS catalog and used by optimizer
  - File size
  - Organization
  - Number of levels of each multilevel index
  - Number of distinct values of an attribute
  - Attribute selectivity
    - Allows calculation of selection cardinality
      - Average number of records that satisfy equality selection condition on that attribute

Histograms

- Tables or data structures that record information about the distribution of data
- RDBMS stores histograms for most important attributes

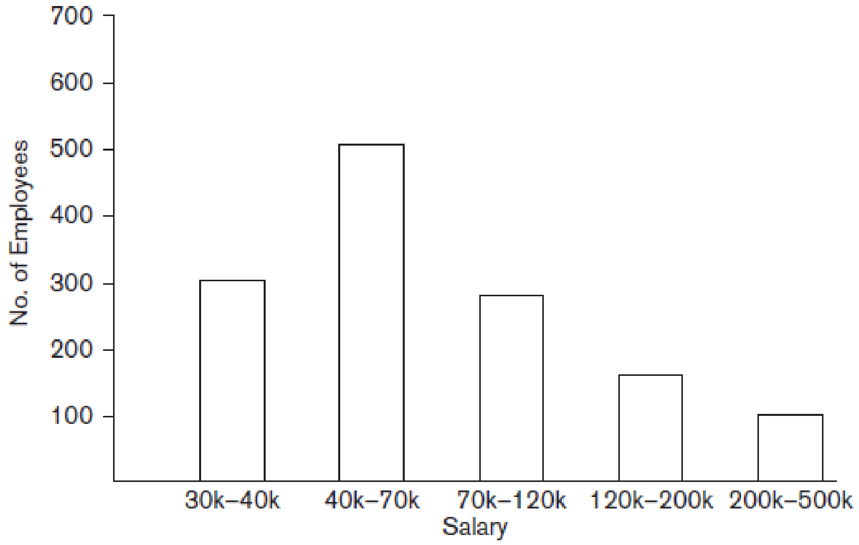


Figure 19.4 Histogram of salary in the relation EMPLOYEE

19.4 Cost Functions for SELECT Operation

- Notation used in cost formulas

C_{Si}: Cost for method *Si* in block accesses
r_X: Number of records (tuples) in a relation X
b_X: Number of blocks occupied by relation X (also referred to as b)
bfr_X: Blocking factor (i.e., number of records per block) in relation X
sl_A: Selectivity of an attribute A for a given condition
sA: Selection cardinality of the attribute being selected ($= sl_A * r$)
xA: Number of levels of the index for attribute A
$b_{I1}A$: Number of first-level blocks of the index on attribute A
NDV (A, X): Number of distinct values of attribute A in relation X

Cost Function for SELECT Operation (cont'd.)

- S1: Linear search (brute force approach)
  - Search all file blocks to retrieve all records

    $C_{S1a}=b$
  - For equality condition on a key attribute, on average one-half the records are searched

    $C_{S1b}=\frac{b}{2}$
- S2: Binary search

  $C_{S2}=\log_2 b+[\frac{s}{bfr}]-1$
  - Reduces to $\log_2 b$ if equality condition is on a key attribute

Cost Function for SELECT Operation (cont'd.)

- S3a: Using a primary index to retrieve a single record

  $C_{S3a} = x + 1$

- S3b: Using a hash key to retrieve a single record

  $C_{S3b} = 1$

- S4: Using an ordering index to retrieve multiple records

  $C_{S4} = x + \frac{b}{2}$

Cost Functions for SELECT Operation (cont'd.)

- S5: Using a clustering index to retrieve multiple records

  $C_{S5} = x + \lceil \frac{s}{bfr} \rceil$

- S6: Using a secondary (B+ tree) index

  $C_{S6a} = x + 1 + s$ (worst case)

  $C_{S6b} = x + \frac{b_{I1}}{2} + \frac{r}{2}$ (for range queries)

Cost Functions for SELECT Operation (cont'd.)

- Dynamic programming
  - Cost-based optimization approach
  - Subproblems are solved only once
  - Applies when a problem has subproblems that themselves have subproblems

19.5 Cost Functions for the JOIN Operation

- Cost functions involve estimate of file size that results from the JOIN operation
- Join selectivity
  - Ratio of the size of resulting file to size of the CARTESIAN PRODUCT file
  - Simple formula for join selectivity

$$js = 1 / \max(\mathrm{NDV}(A, R), \mathrm{NDV}(B, S))$$

- Join cardinality

$$jc = |(R_c\ S)| = js * |R| * |S|$$

Cost Functions for the JOIN Operation (cont'd.)

- J1: Nested-loop join
  - For three memory buffer blocks:

$$C_{J1} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$$

  - For n_B memory buffer blocks:

$$C_{J1} = b_R + (\lceil b_R/(n_B - 2)\rceil * b_S) + ((js * |R| * |S|)/bfr_{RS})$$

- J2: Index-based nested-loop join
  - For a secondary index with selection cardinality S_B for join attribute B of S:

$$C_{J2a} = b_R + (|R| * (x_B + 1 + s_B)) + ((js * |R| * |S|)/bfr_{RS})$$

Cost Functions for the JOIN Operation (cont'd.)

- J3: Sort-merge join
  - For files already sorted on the join attributes

$$C_{J3a} = b_R + b_S + ((js * |R| * |S|)/bfr_{RS})$$

  - Cost of sorting must be added if sorting needed
- J4: Partition-hash join

$$C_{J4} = 3 * (b_R + b_S) + ((js * |R| * |S|)/bfr_{RS})$$

Cost Functions for the JOIN Operation (cont'd.)

- Join selectivity and cardinality for semi-join

```
SELECT COUNT(*)
FROM T1
WHERE T1.X IN (SELECT T2.Y
      FROM T2);
```

  - Unnesting query above leads to semi-join

```
SELECT COUNT(*)
FROM T1, T2
WHERE T1.X S= T2.Y;
```

- Join selectivity

$$js = \text{MIN}(1, \text{NDV}(Y, T2)/\text{NDV}(X, T1))$$

- Join cardinality

$$jc = |T1| * js$$

Cost Functions for the JOIN Operation (cont'd.)

- Join selectivity and cardinality for anti-join

```
SELECT COUNT (*)
FROM T1
WHERE T1.X NOT IN (SELECT T2.Y
FROM T2);
```

  - Unnesting query above leads to anti-join

```
SELECT COUNT(*)
FROM T1, T2
WHERE T1.X A= T2.Y;
```

- Join selectivity

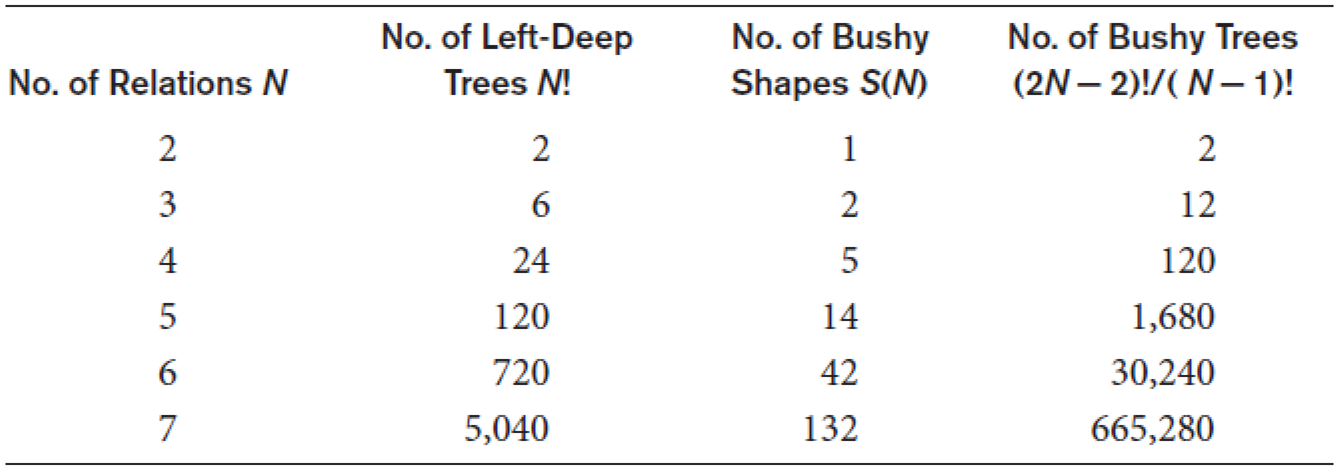
$$js = 1 - \text{MIN}(1, \text{NDV}(\text{T2.y})/\text{NDV}(\text{T1.x}))$$

- Join cardinality

$$jc = |\text{T1}| * js$$

Cost Functions for the JOIN Operation (cont'd.)

- Multirelation queries and JOIN ordering choices
  - Left-deep join tree
  - Right-deep join tree
  - Bushy join tree



No. of Relations N	No. of Left-Deep Trees $N!$	No. of Bushy Shapes $S(N)$	No. of Bushy Trees $(2N-2)!/(N-1)!$
2	2	1	2
3	6	2	12
4	24	5	120
5	120	14	1,680
6	720	42	30,240
7	5,040	132	665,280

Table 19.1 Number of permutations of left-deep and bushy join trees of *n* relations

Cost Functions for the JOIN Operation (cont'd.)

- Physical optimization involves execution decision at the physical level
  - Cost-based physical optimization
    - Top-down approach
    - Bottom-up approach
- Certain physical level heuristics make cost optimizations unnecessary
  - Example: for selections, use index scans whenever possible

Cost Functions for the JOIN Operation (cont'd.)

- Left-deep trees generally preferred
  - Work well for common algorithms for join
  - Able to generate fully pipelined plans
- Characteristics of dynamic programming algorithm
  - Optimal solution structure is developed
  - Value of the optimal solution is recursively defined
  - Optimal solution is computed and its value developed in a bottom-up fashion

19.6 Example to Illustrate Cost-Based Query Optimization

- Example: Consider Q2 below and query tree from Figure 19.1(a) on slide 6

```
Q2:  SELECT  Pnumber, Dnum, Lname, Address, Bdate
     FROM    PROJECT, DEPARTMENT, EMPLOYEE
     WHERE   Dnum=Dnumber AND Mgr_ssn=Ssn AND
             Plocation='Stafford';
```

- Information about the relations shown in Figure 19.6 (next slide)

Figure 19.6 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	Mgr_ssn	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	Uniqueness	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.

Example to Illustrate Cost-Based Query Optimization (cont'd.)

- Assume optimizer considers only left-deep trees
- Evaluate potential join orders
  - PROJECT $\bowtie$ DEPARTMENT $\bowtie$ EMPLOYEE
  - DEPARTMENT $\bowtie$ PROJECT $\bowtie$ EMPLOYEE
  - DEPARTMENT $\bowtie$ EMPLOYEE $\bowtie$ PROJECT
  - EMPLOYEE $\bowtie$ DEPARTMENT $\bowtie$ PROJECT

19.7 Additional Issues Related to Query Optimization

- Displaying the system's query execution plan
  - Oracle syntax
    - EXPLAIN PLAN FOR <SQL query>
  - IBM DB2 syntax
    - EXPLAIN PLAN SELECTION [additional options] FOR <SQL-query>
  - SQL server syntax
    - SET SHOWPLAN_TEXT ON or SET SHOWPLAN_XML ON or SET SHOWPLAN_ALL ON

Additional Issues Related to Query Optimization (cont'd.)

- Size estimation of other operations
  - Projection
  - Set operations
  - Aggregation
  - Outer join
- Plan caching
  - Plan stored by query optimizer for later use by same queries with different parameters
- Top k-results optimization
  - Limits strategy generation

19.8 An Example of Query Optimization in Data Warehouses

- Star transformation optimization
  - Goal: access a reduced set of data from the fact table and avoid using a full table scan on it
  - Classic star transformation
  - Bitmap index star transformation
- Joining back

19.9 Overview of Query Optimization in Oracle

- Physical optimizer is cost-based
- Scope is a single query block
- Calculates cost based on object statistics, estimated resource use and memory needed
- Global query optimizer
  - Integrates logical transformation and physical optimization phases to generate optimal plan for entire query tree
- Adaptive optimization
  - Feedback loop used to improve on previous decisions

Overview of Query Optimization in Oracle (cont'd.)

- Array processing
- Hints
  - Specified by application developer
  - Embedded in text of SQL statement
  - Types: access path, join order, join method, enabling or disabling a transformation
- Outlines used to preserve execution plans
- SQL plan management

19.10 Semantic Query Optimization

- Uses constraints specified on the database schema
- Goal: modify one query into another that is more efficient to execute

19.11 Summary

- Query trees
- Heuristic approaches used to improve efficiency of query execution
- Reorganization of query trees
- Pipelining and materialized evaluation
- Cost-based optimization approach
- Oracle query optimizer
- Semantic query optimization