Fundamentals of

Database Systems

5th Edition

Elmasri / Navathe

Algorithms for Query Optimization

Fundamentals of Database Systems

5th Edition

Elmasri / Navathe

Query Optimization

- **Query:** A query is a request for information from a database.
- **Query Plans:** A query plan (or query execution plan) is an ordered set of steps used to access data in a SQL relational database management system.
- **Query Optimization:**
  - A single query can be executed through different algorithms or re-written in different forms and structures.
  - Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal?

Query Optimizer

- A query optimizer is a critical database management system (DBMS) component that analyzes Structured Query Language (SQL) queries and determines efficient execution mechanisms.
- A query optimizer generates one or more query plans for each query, each of which may be a mechanism used to run a query. The most efficient query plan is selected and used to run the query.

Query optimization – 2 ways

- Analyze and transform equivalent relational expressions:
  - Try to minimize the tuple and column counts of the intermediate and final query processes.
- Using different algorithms for each operation:
  - These underlying algorithms determine how tuples are accessed from the data structures they are stored in, indexing, hashing, data retrieval and hence influence the number of disk and block accesses

Query Optimization Types

- Query optimization divided into two types: **Heuristic** (sometimes called Rule based) and **Systematic** (Cost based).
- **Heuristic Query Optimization**
  - Oracle calls this Rule Based optimization.
  - A query can be represented as a tree data structure. Operations are at the interior nodes and data items (tables, columns) are at the leaves.
  - The query is evaluated in a *depth-first* pattern.

- Analyze and transform equivalent relational expressions

Using Heuristics in Query Optimization

- Uses heuristic algorithms to evaluate relational algebra expressions
- Process for heuristics optimization
  1. The parser of a high-level query generates an initial internal representation (*estimating the cost of a relational algebra expression*)
  2. Apply heuristics rules to optimize the internal representation (*transforming one relational algebra expression to an equivalent one*).
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.
- The main heuristic is to apply first the operations that reduce the size of intermediate results.
  - E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

Equivalence Preserving Transformation

- To transform a relational expression into another equivalent expression we need transformation rules that preserve equivalence known as 'Equivalence Rules'
- These generate equivalent expressions for a query written in relational algebra.
- To optimize a query, convert the query into its equivalent form as long as an equivalence rule is satisfied.

General Transformation Rules for Relational Algebra Operations

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

- $\sigma_{\text{c1 AND c2 AND ... AND cn}}(R) = \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$
- **Explanation:**
  - Applying condition c1 intersection c2 is expensive. Instead, filter out tuples satisfying condition (inner selection) and then apply condition (outer selection) to the resulting fewer tuples.
  - This leaves us with less tuples to process the second time. This can be extended for two or more intersecting selections. Since we are breaking a single condition into a series of selections or cascades, it is called a "cascade".

General Transformation Rules for Relational Algebra Operations (cont.)

2. Commutativity of σ: The σ (selection) operation is commutative:

- $\sigma_{c1} (\sigma_{c2}(R)) = \sigma_{c2} (\sigma_{c1}(R))$
- **Explanation:**
  - condition is commutative in nature. This means, it does not matter whether we apply c1 first or c2 first. In practice, it is better and more optimal to apply that selection first which yields a fewer number of tuples. This saves time on our outer selection.

3. Cascade of π: In a cascade (sequence) of π operations, all following projections can be omitted, only the first projection is required. This is called a pi-cascade.

- $\pi_{List1} (\pi_{List2} (...(\pi_{Listn}(R))...)) = \pi_{List1}(R)$
- **Explanation:**
  - A cascade or a series of projections is meaningless. This is because in the end, we are only selecting those columns which are specified in the last, or the outermost projection. Hence, it is better to collapse all the projections into just one i.e. the outermost projection.

General Transformation Rules for Relational Algebra Operations (cont.)

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:

- $\pi_{A1, A2, ..., An} (\sigma_c (R)) = \sigma_c (\pi_{A1, A2, ..., An} (R))$
- Explanation:
  - Permitted if attr in $\pi \supseteq$ all attributes in σ_c

5. Commutativity of $\bowtie$ (and x): The $\bowtie$ operation is commutative as is the x operation:

- $R \bowtie_C S = S \bowtie_C R$; $R \times S = S \times R$
- Explanation:
  - although the order of attributes may not be the same in the relations resulting from the two joins (or two Cartesian products), the *meaning* is the same because the order of attributes is not important in the alternative definition of relation.

General Transformation Rules for Relational Algebra Operations (contd.):

6. Commuting σ with $\bowtie$ (or x): 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:
   - $\sigma_c (R \bowtie S) = (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$

General Transformation Rules for Relational Algebra Operations (contd.):

7. Commuting π with $\bowtie$ (or x): 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.

General Transformation Rules for Relational Algebra Operations (contd.):

8. Commutativity of set operations: The set operations ∪ and ∩ are commutative but "–" is not.

9. Associativity of $\bowtie$, x, ∪, 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
   - $\sigma_c (R \; \theta \; S) \; = \; (\sigma_c (R)) \; \theta \; (\sigma_c (S))$

General Transformation Rules for Relational Algebra Operations (contd.):

11. The π operation commutes with $\cup$.

$$\pi_L (R \cup S) = (\pi_L (R)) \cup (\pi_L (S))$$

12. Converting a (σ, x) sequence into $\bowtie$: If the condition c of a σ that follows a x Corresponds to a join condition, convert the (σ, x) sequence into a $\bowtie$ as follows:

$$(\sigma_C (R \times S)) = (R \bowtie_C S)$$

Examples

- Assume the following tables:

  instructor(ID, name, dept_name, salary)
  teaches(ID, course_id, sec_id, semester, year)
  course(course_id, title, credits)

Query 1: Find the names of all instructors in the Music department, along with the titles of the courses that they teach

$$\pi_{name,title}(\sigma_{dept\_name=\text{``Music''}}(instructor \bowtie (teaches \bowtie \pi_{course\_id,title}(course))))$$

Here, dept_name is a field of only the instructor table. Hence, we can select out the Music instructors before joining the tables, hence reducing query time.

Examples (contd..)

$$\pi_{name,title}(\sigma_{dept\_name=\text{"}Music\text{"}}(instructor \bowtie (teaches \bowtie \pi_{course\_id,title}(course))))$$

Here, dept_name is a field of only the instructor table. Hence, we can select out the Music instructors before joining the tables, hence reducing query time.

Optimized Query:

Using rule 6a, and Performing the selection as early as possible reduces the size of the relation to be joined.

$$\pi_{name,title}((\sigma_{dept\_name=\text{"}Music\text{"}}(instructor) \bowtie (teaches \bowtie \pi_{course\_id,title}(course)))$$

Outline of a Heuristic Algebraic Optimization Algorithm:

1. Using rule 1, break up any select operations with conjunctive conditions into a cascade of select operations.
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.
4. Using Rule 12, combine a Cartesian product operation with a subsequent select operation in the tree 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.

Using Heuristics in Query Optimization (2)

- **Query tree**:
  - A tree data structure that corresponds to a relational algebra expression.
  - It represents the input relations of the query as **leaf nodes** of the **tree**, and represents the relational algebra operations as internal nodes.
  - The root relation represents the answer to the query
  - Two query trees are equivalent if their root relations are the same
- **Query graph**:
  - A graph data structure that corresponds to a relational calculus expression. It does *not* indicate an order on which operations to perform first. There is only a *single* graph corresponding to each query.

Query Tree

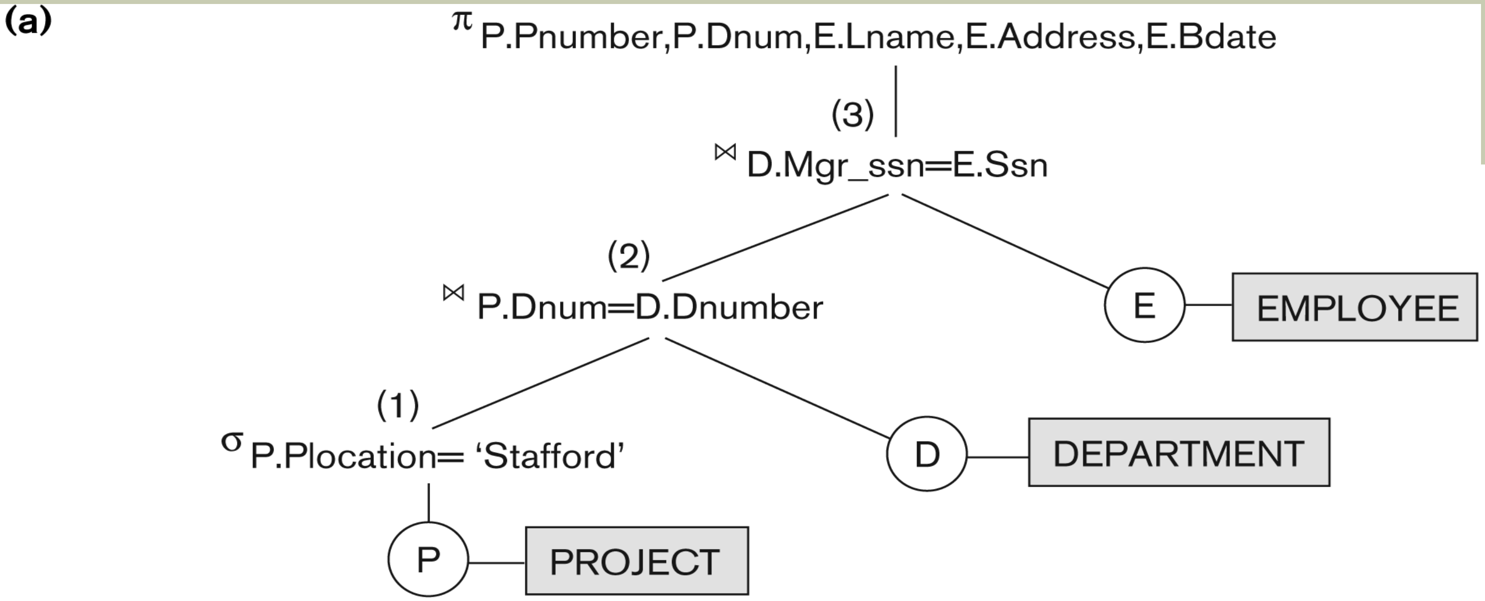
- Example:
  - For every project located in ‘Stafford’, retrieve the project number, the controlling department number and the department manager’s last name, address and birthdate.
- SQL query:

```
Q2:        SELECT     P.NUMBER,P.DNUM,E.LNAME,
                      E.ADDRESS, E.BDATE
           FROM       PROJECT AS P,DEPARTMENT AS D,
                           EMPLOYEE AS E
           WHERE      P.DNUM=D.DNUMBER AND
                      D.MGRSSN=E.SSN AND
                      P.PLOCATION=‘STAFFORD’;
```

- Relation algebra:

$$\pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}} (((\sigma_{\text{PLOCATION='STAFFORD'}}(\text{PROJECT})) \bowtie_{\text{DNUM=DNUMBER}} (\text{DEPARTMENT})) \bowtie_{\text{MGRSSN=SSN}} (\text{EMPLOYEE}))$$

Query Tree



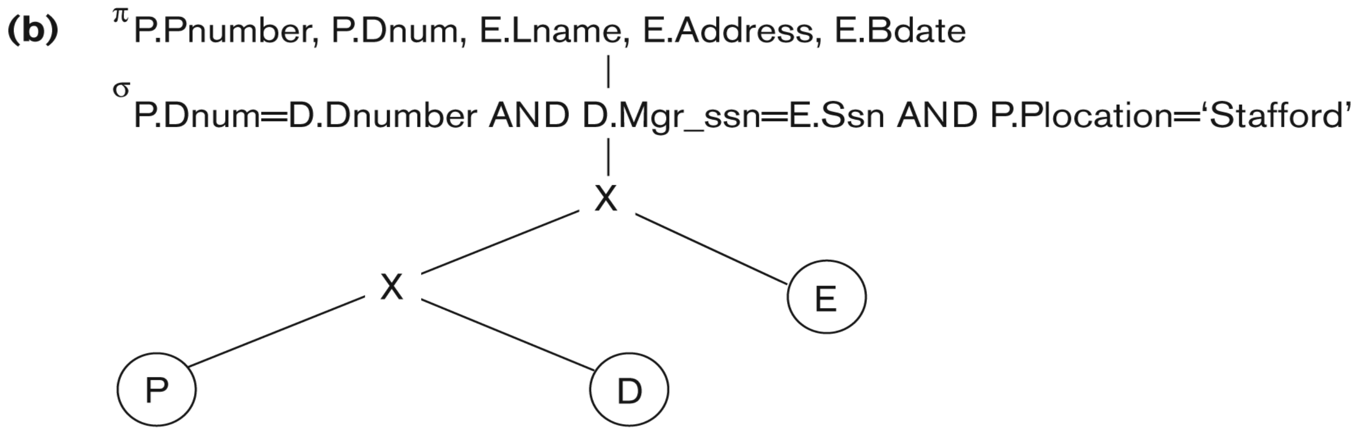

(b)
π P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
σ P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation='Stafford'
X
X
E
P
D

Figure 15.4
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. (c) Query graph for Q2.

Query Graph

- Nodes represents Relations
- Edges represents Join & Selection conditions
- Attributes to be retrieved from relations represented in square brackets.
- *Drawback* :- Does not indicate an order on which operations are performed

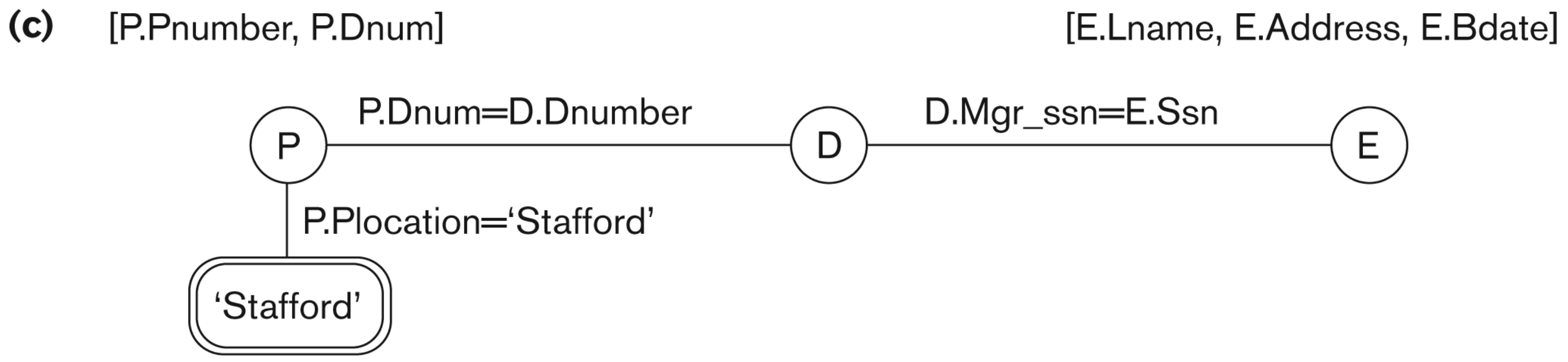


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Query Tree Optimization Example

- Heuristic Optimization of Query Trees:
  - The same query could correspond to many different relational algebra expressions — and hence many different query trees.
  - The task of heuristic optimization of query trees is to find a **final query tree** that is efficient to execute.

Query Tree Optimization Example

- Example: Find the last names of employees born after 1957 who work on a project named ‘Aquarius’.

```
Q:  SELECT      LNAME
    FROM        EMPLOYEE, WORKS_ON, PROJECT
    WHERE       PNAME = ‘AQUARIUS’ AND
                PNMUBER=PNO AND ESSN=SSN
                AND BDATE > ‘1957-12-31’;
```

Steps in converting a query tree during heuristic optimization.

Initial (canonical) query tree for SQL query Q.

Moving SELECT operations down the query tree.

Applying the more restrictive SELECT operation first.

Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.

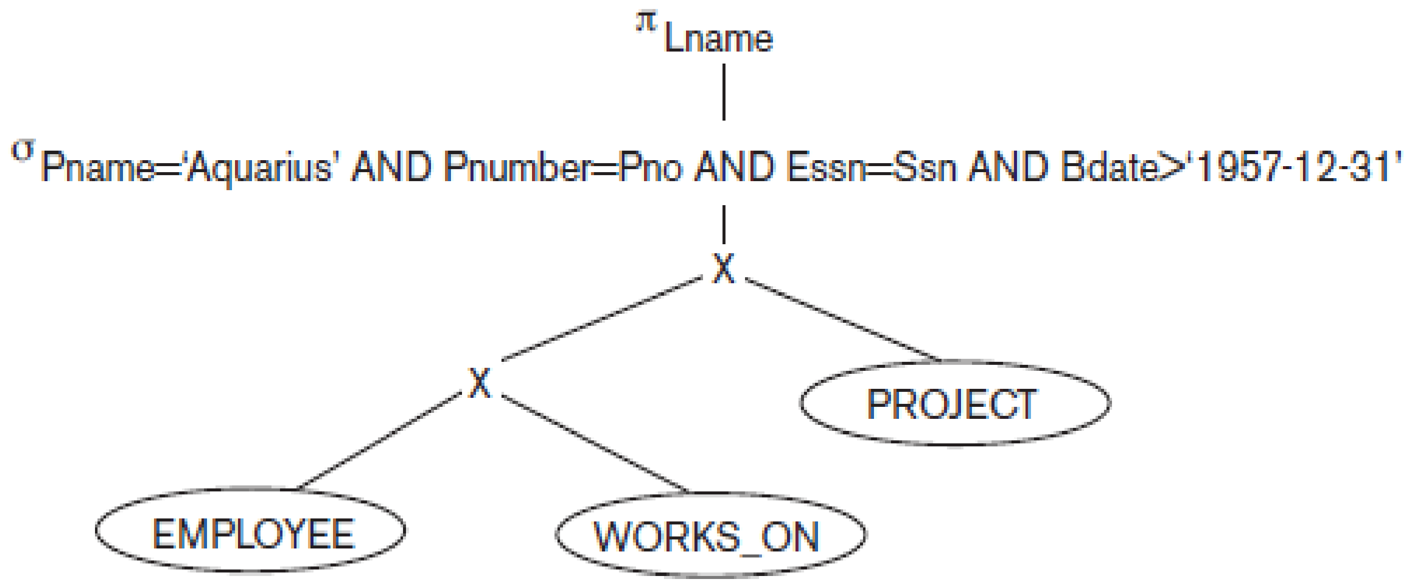
Moving PROJECT operations down the query tree.

Query Tree Optimization Example

- **INITIAL QUERY TREE**

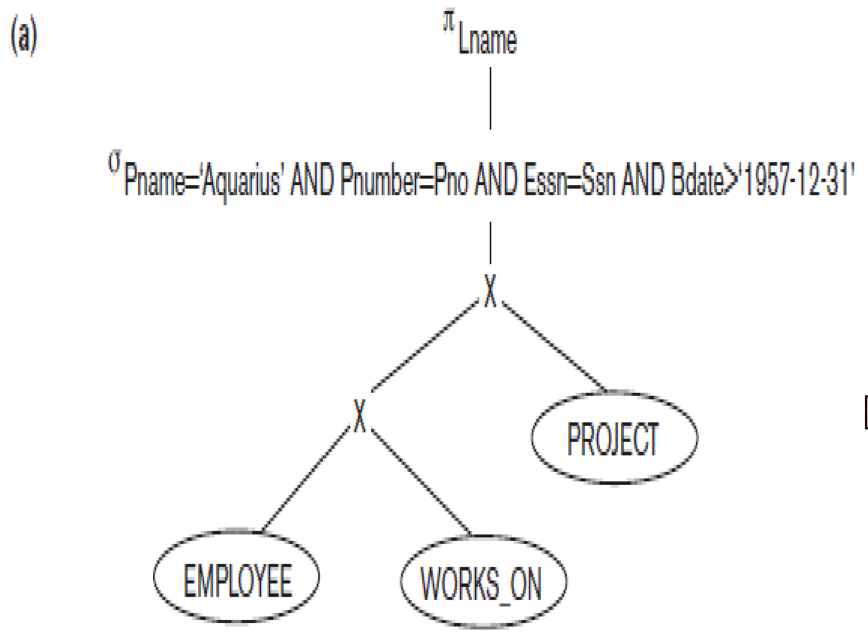
```
Q:  SELECT      LNAME
    FROM        EMPLOYEE, WORKS_ON, PROJECT
    WHERE       PNAME = ‘AQUARIUS’ AND
          PNMUBER=PNO AND ESSN=SSN
          AND BDATE > ‘1957-12-31’;
```

(a)

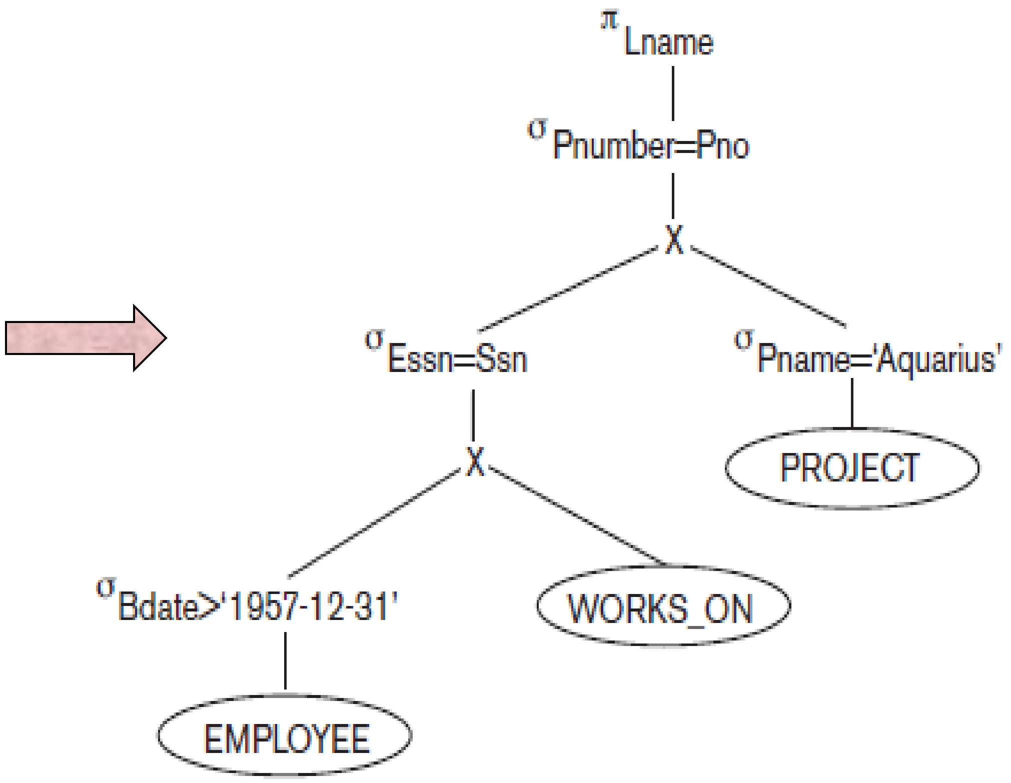

π Lname
σ Pname='Aquarius' AND Pnumber=Pno AND Essn=Ssn AND Bdate>'1957-12-31'
X
X
PROJECT
EMPLOYEE
WORKS_ON

Query Tree Optimization Example (cont.)

- MOVE SELECT DOWN THE TREE USING CASCADE & COMMUTATIVITY RULE OF SELECT OPERATION



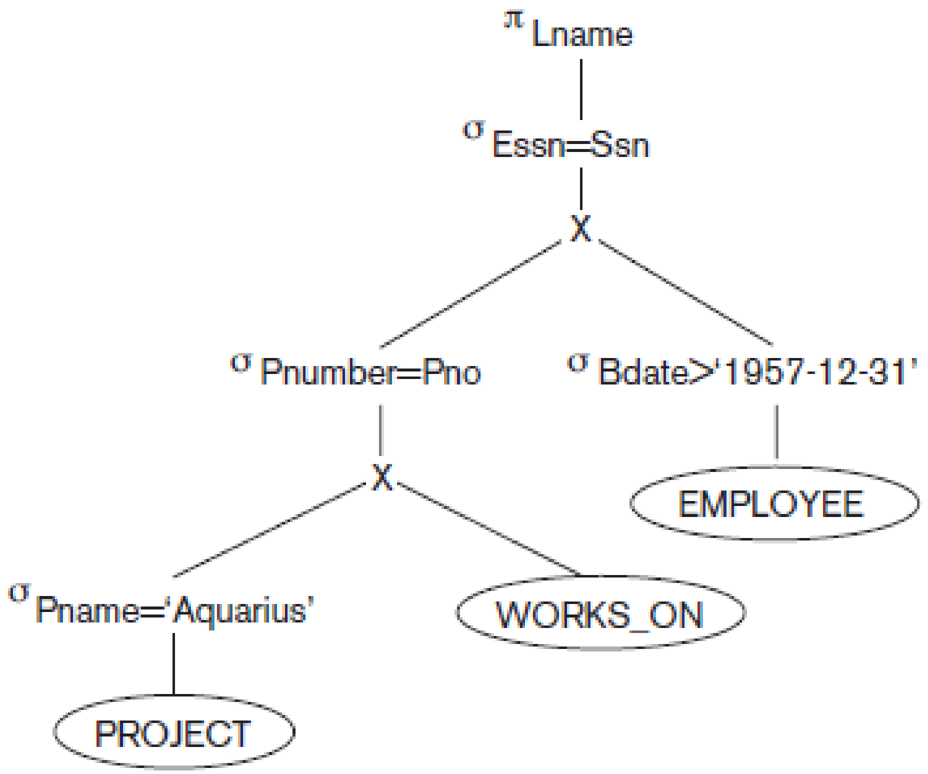
(a)
π_{Lname}
$\sigma_{Pname='Aquarius' \text{ AND } Pnumber=Pno \text{ AND } Essn=Ssn \text{ AND } Bdate>'1957-12-31'}$
X
X
PROJECT
EMPLOYEE
WORKS_ON



Query Tree Optimization Example (cont.)

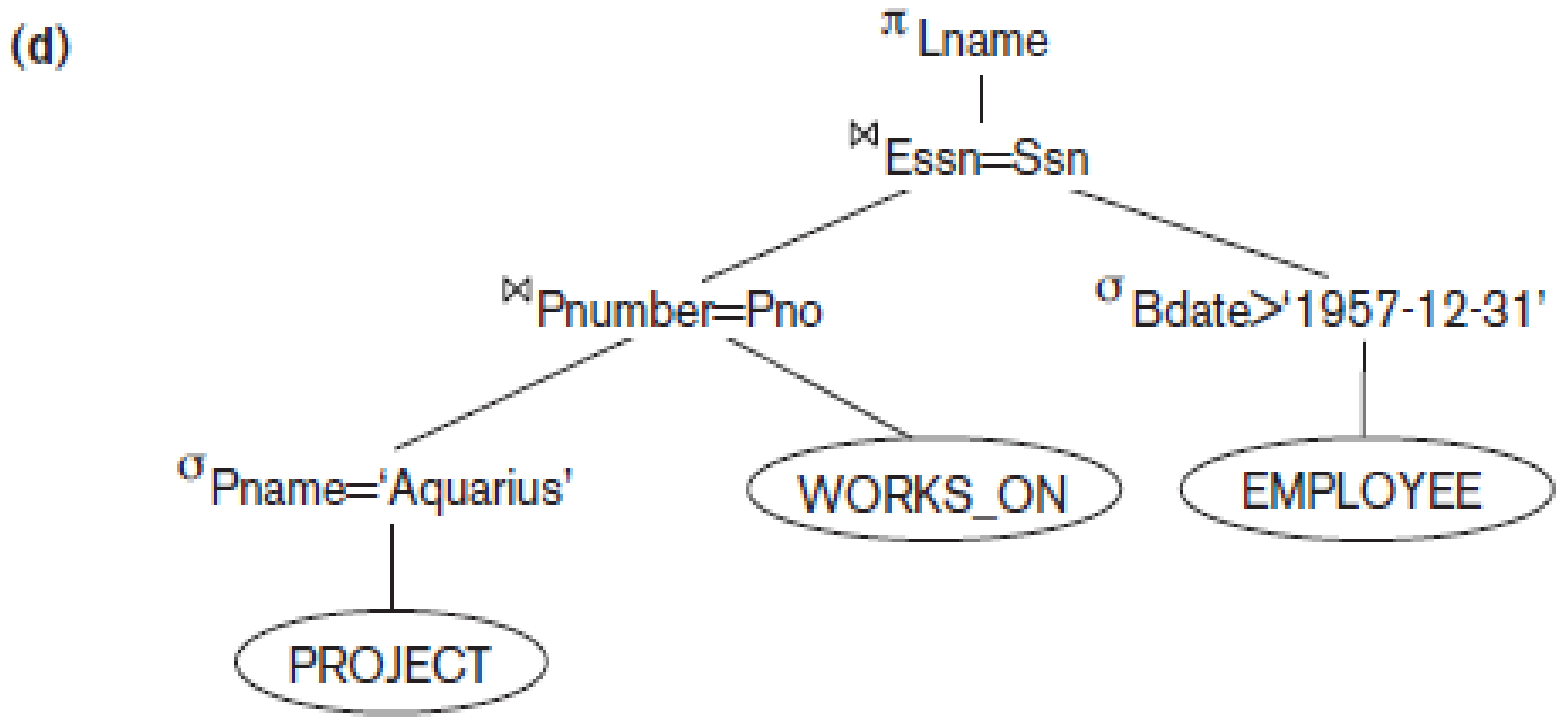
- REARRANGE OF LEAF NODES, USING COMMUTATIVITY & ASSOCIATIVITY OF BINARY OPERATIONS.

(c)



Query Tree Optimization Example (cont.)

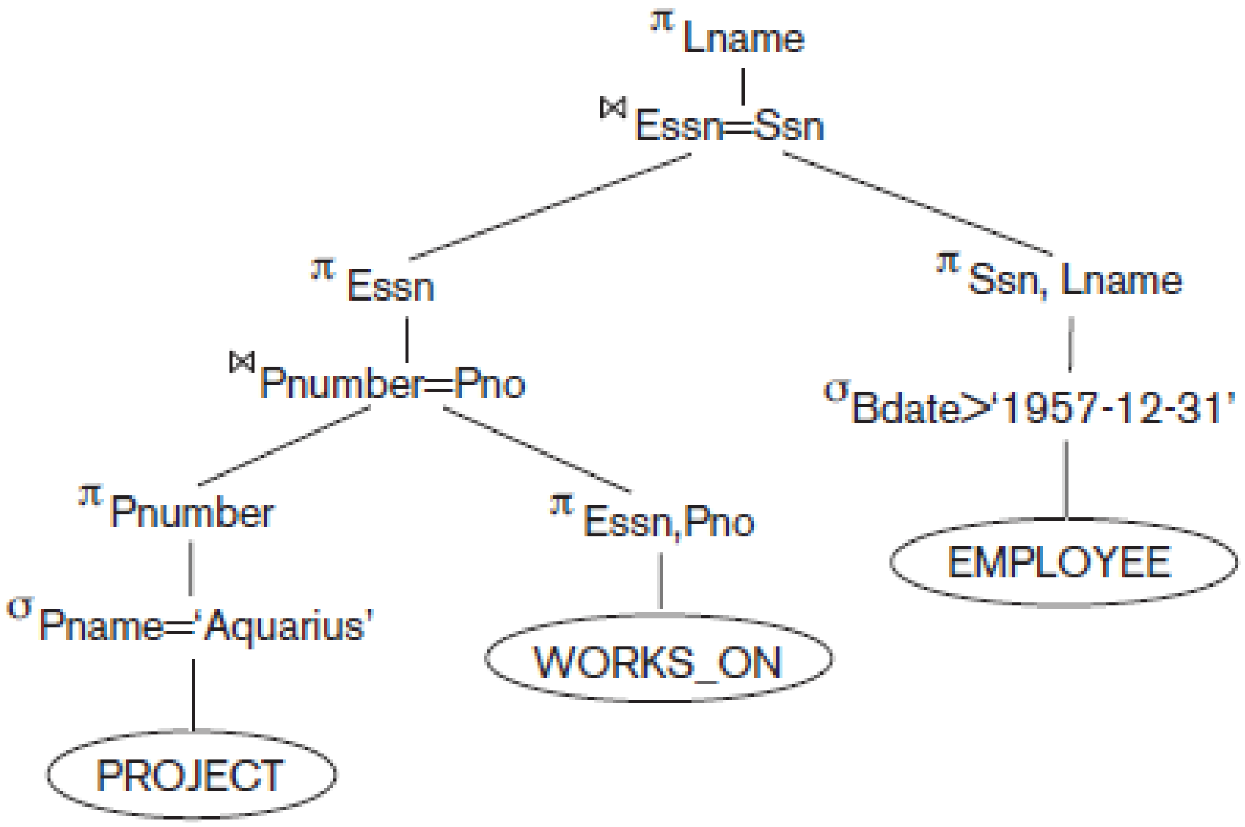
- **CONVERTING SELECT & CARTESIAN PRODUCT INTO JOIN**



Query Tree Optimization Example (cont.)

- **BREAK-MOVE OF PROJECT USING CASCADE & COMMUTING RULES OF PROJECT OPERATIONS.**

(e)



Example of heuristic query optimization

- PROJECT TABLE
  - PNAME PNUMBER PLOCATION DNUM
- DEPARTMENT TABLE:
  - DNAME DNUMBER MGREMPID MGRSTARTD
- EMPLOYEE TABLE
  - FNAME MI LNAME EMPID BDATE ADDRESS S SALARY SUPERMPID DNO

Query:

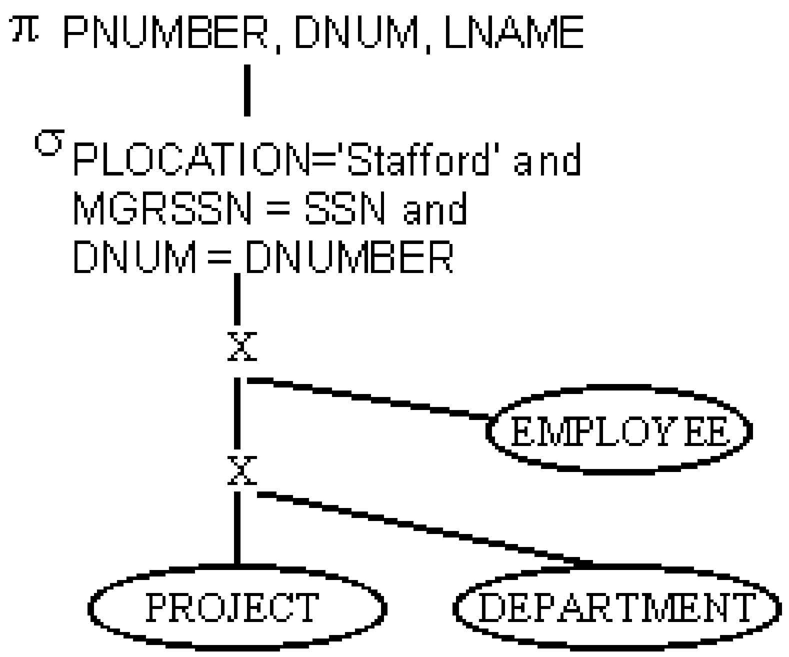
- SELECT PNUMBER, DNUM, LNAME FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE DNUM=DNUMBER and MGREMPID=EMPID and PLOCATION = 'Stafford';

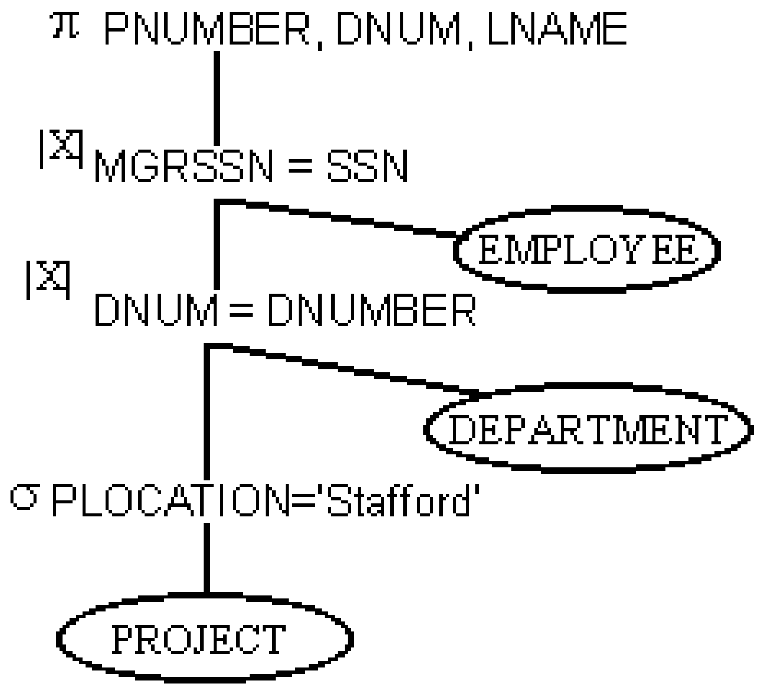
Example of heuristic query optimization(contd..)

Relational Algebra:

$$\pi_{\text{pnumber, dnum, lname}}\left(\sigma_{\text{plocation} = \text{'Stafford'}}\left(\sigma_{\text{mgrssn=ssn}}\left(\sigma_{\text{dnum=dnumber}}\left(P \times (D \times E)\right)\right)\right)\right)$$

Which of the following query tree is more efficient?



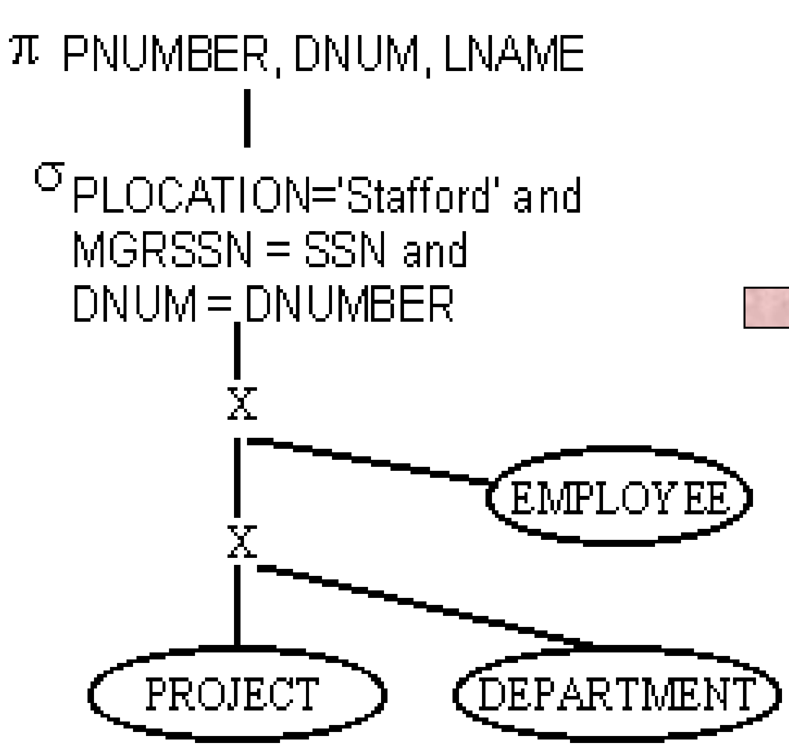


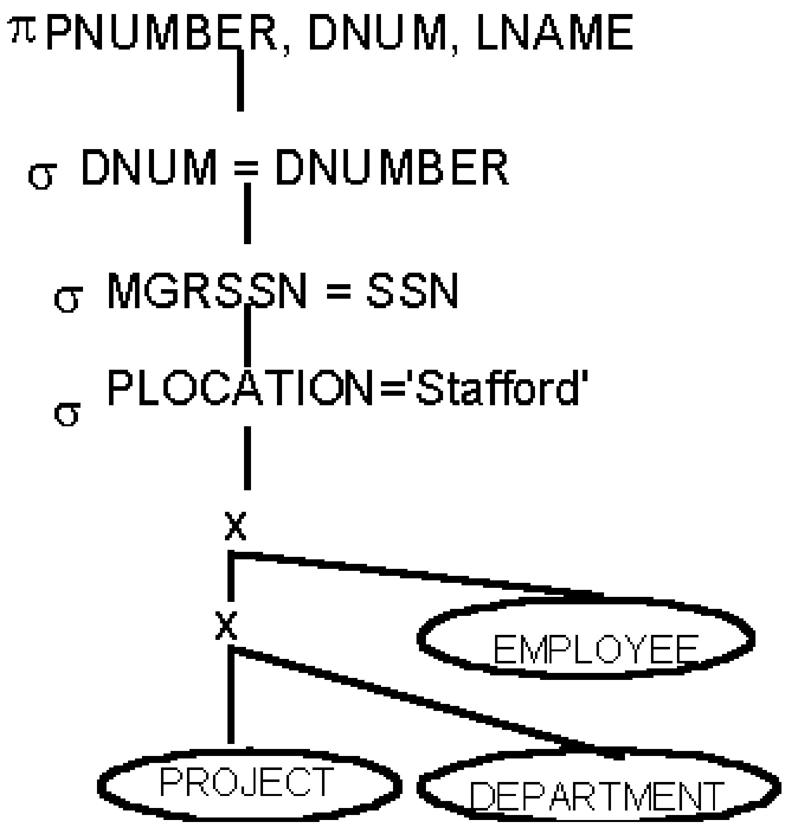
Example of heuristic query optimization(contd..)

- Note the two cross product operations. These require lots of space and time (nested loops) to build.
- After the two cross products, we have a temporary table with 144 records (6 projects * 3 departments * 8 employees).
- An overall rule for heuristic query optimization is to perform as many select and project operations as possible before doing any joins.

Example of heuristic query optimization(contd..)

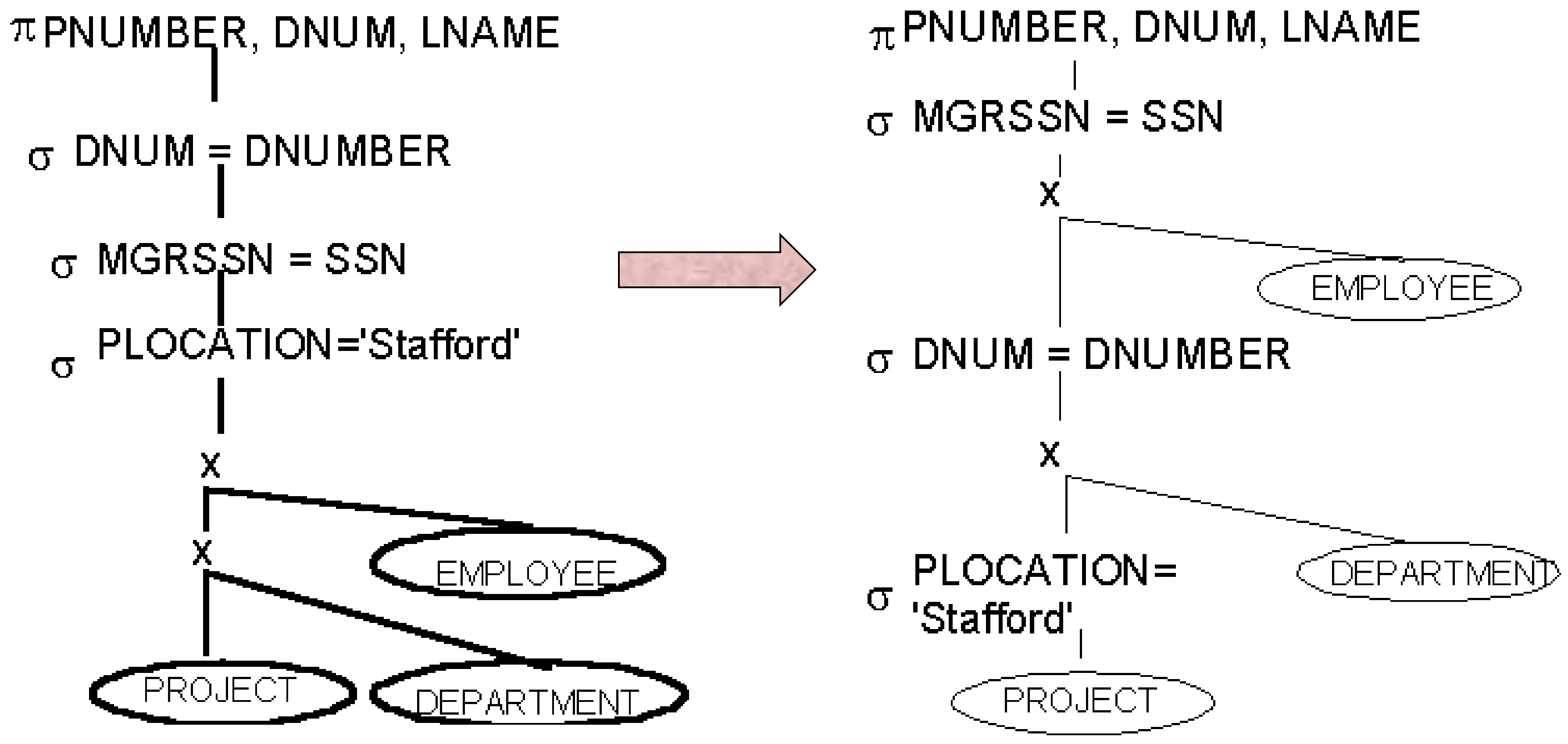
Use Rule 1 to Break up Cascading Selections





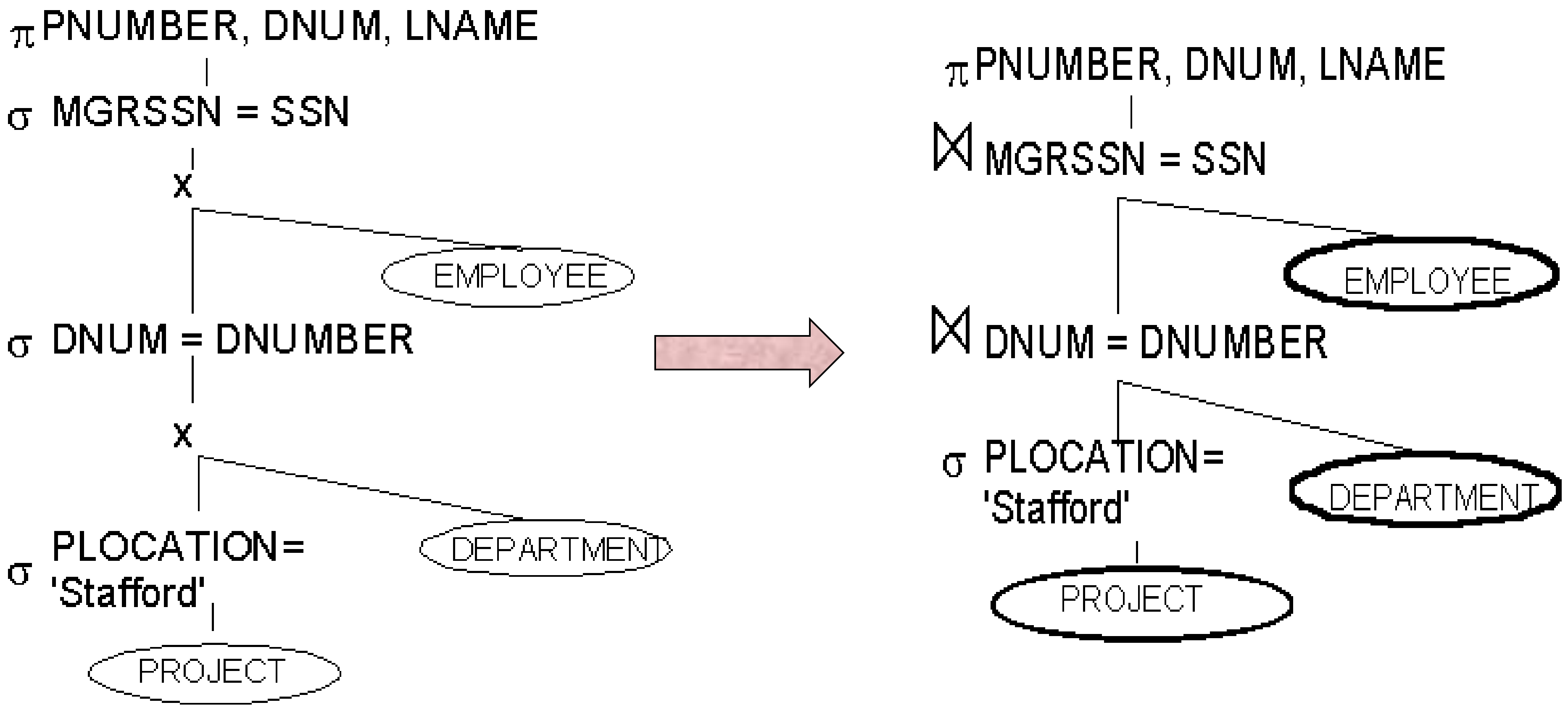
Example of heuristic query optimization(contd..)

Use Rule 2 Commute Selection with Cross Product



Example (contd..)

Combine Cross Product and Selection to form Joins



Summary of Heuristics in Query Optimization

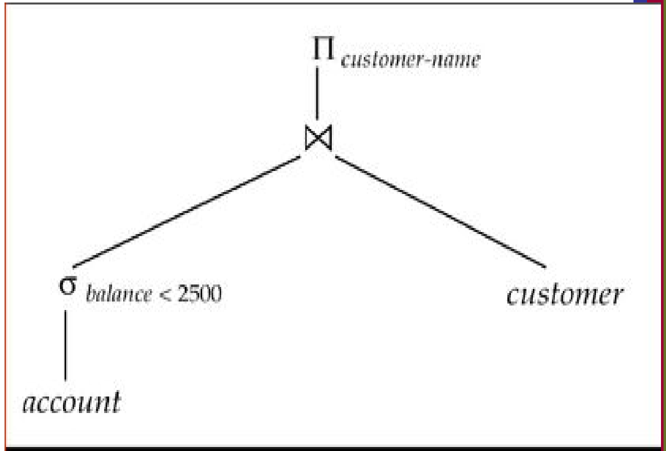
- Summary of Heuristics for Algebraic Optimization:
  1. The main heuristic is to apply first the operations that reduce the size of intermediate results.
  2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible.)
  3. The select and join operations that are most restrictive 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.)

Query Execution Plans

- An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.
- **Two approaches for Execution Plan**
  - **Materialized evaluation**
  - **Pipelined evaluation**

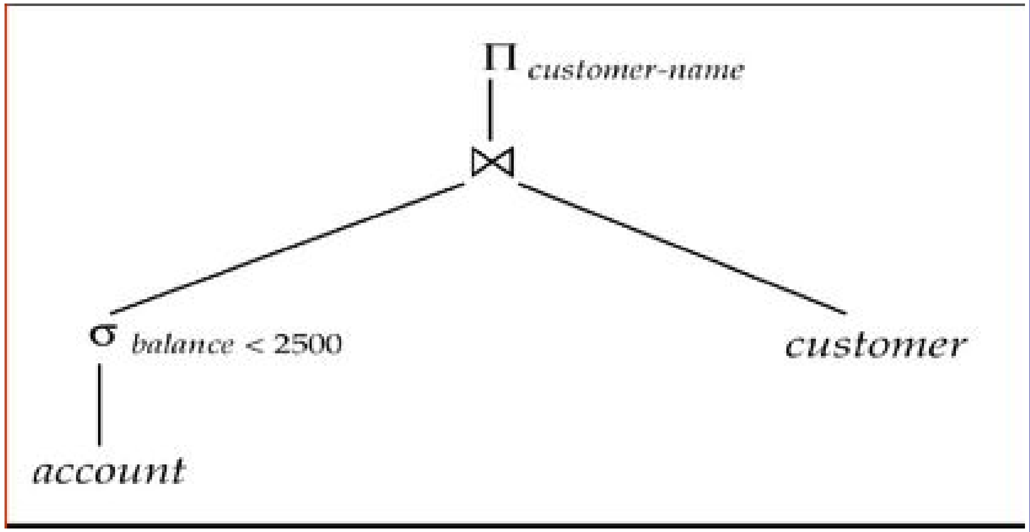
Using Heuristics in Query Optimization

- Query Execution Plans -
  - **Materialized evaluation**: the result of an operation is stored as a temporary relation.
  - Evaluate one operation at a time, starting at the lowest- level.
  - Use intermediate results materialized into temporary relations to evaluate next-level operations.
  - The cost of materialization is the sum of the individual operations plus the cost of writing the intermediate results to disk
  - Example:
    - Evaluate the expression (selection condition) in the figure below. Compute and store
    - Then compute and store its join with customer
    - Finally, compute the projections on customer-name.



Using Heuristics in Query Optimization

- Query Execution Plans -
  - **Pipelined evaluation**: as the result of an operator is produced, it is forwarded to the next operator in sequence.
  - Example:
    - Do not store the result of (selection condition) Instead, pass tuples directly to the join.
    - Similarly, don't store result of join, pass tuples directly to projection.



Cost-based query optimization Measures of Query Cost:

- **Cost-based query optimization**:
  - The main aim of query optimization is to choose the most efficient way of implementing the relational algebra operations at the lowest possible cost.
  - Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.
  - The method of optimizing the query by choosing a strategy those results in minimum cost is called cost-based query optimization

Cost-based query optimization Measures of Query Cost:

- **Cost Components for Query Execution**
  - Access cost to secondary storage
    - Disk I/O Cost – cost of transferring blocks to and from disk to memory
    - The cost of searching for records in a disk file (access structure)
    - Contiguous/non-contiguous blocks allocation
  - Storage cost
    - cost of storing on disk any intermediate files that are generated by an execution strategy for the query
  - Computation cost
    - Cost of performing in-memory operations on the records within the data buffers during query execution

Cost-based query optimization Measures of Query Cost:

- **Cost Components for Query Execution (contd..)**
  - Memory usage cost
    - Cost pertaining to the number of main memory buffers needed during query execution
  - Communication cost
    - The cost of shipping the query and its results from the database site to the site or terminal where the query originated

**Note: Different database systems may focus on different cost components.

Cost-based query optimization

- **Catalog Information Used in Cost Functions**
  - Information about the size of a file
    - number of records (tuples) (r),
    - record size (R),
    - number of blocks (b)
    - blocking factor (bfr)
  - Information about indexes and indexing attributes of a file
    - Number of levels (x) of each multilevel index
    - Number of first-level index blocks (bI1)
    - Number of distinct values (d) of an attribute

Cost-based query optimization

- **Catalog Information Used in Cost Functions (contd..)**
  - Information about indexes and indexing attributes of a file (contd..)
    - Selectivity (sl) of an attribute, the fraction of records satisfying an equality condition on the attribute,
      - For key attribute, sl = 1/r . For non-key, sl = 1/d (*d distinct values are* uniformly distributed among the records)
    - Selection cardinality (s) of an attribute, *average number of records that will satisfy an* equality selection condition on that attribute.
      - S = sl * r
- (For a nonkey attribute with *d distinct values, it is often the case that the records are* not uniformly distributed among these values
  - For example, suppose that a company has 5 departments numbered 1 through 5, and 200 employees who are distributed among the departments as follows: (1, 5), (2, 25), (3, 70), (4, 40), (5, 60)
  - In such cases, the optimizer can store a **histogram** that reflects the distribution of employee records over different departments in a table with the two attributes (Dno, Selectivity), which would contain the following values for our example:

    (1, 0.025), (2,0.125), (3, 0.35), (4, 0.2), (5, 0.3).)

Reference

- Navathe