Chapter 19: Optimization

CS-6360 Database Design

Chris Irwin Davis, Ph.D.

Email: chrisirwindavis@utdallas.edu
Phone: (972) 883-3574
Office: ECSS 4.705

Introduction to Query Processing

- **Query optimization**:
  - The process of choosing a suitable execution strategy for processing a query.
- Two high-level approaches:
  - **Heuristic Estimate**
  - **Cost-based Estimate**

19.1 – Query Trees and Heuristics for Query Optimization

Using Heuristics in Query Optimization

- Process for heuristics optimization
  1. The parser of a high-level query generates an initial internal representation;
  2. Apply heuristics rules to optimize the internal representation.
  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 first apply the operations that reduce the size of intermediate results.
  - e.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

Using Heuristics in Query Optimization

- 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.

- Relation algebra:

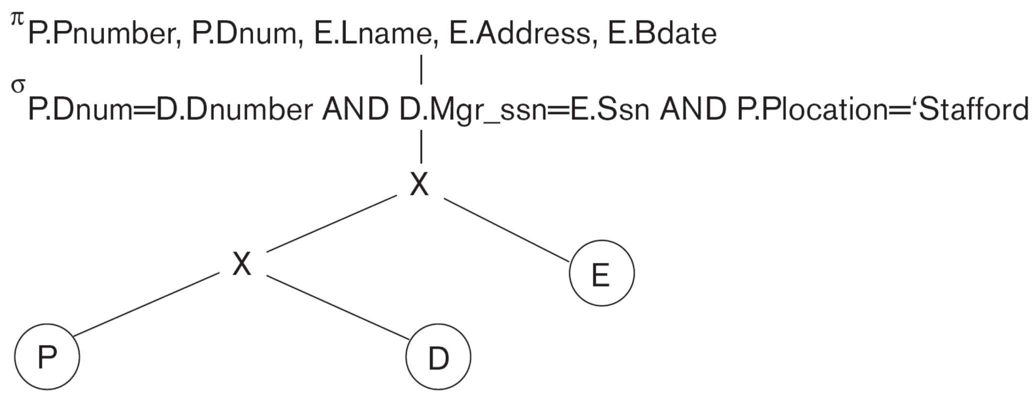
$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}} (((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}} (\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}} (\text{EMPLOYEE}))$$

- SQL query:

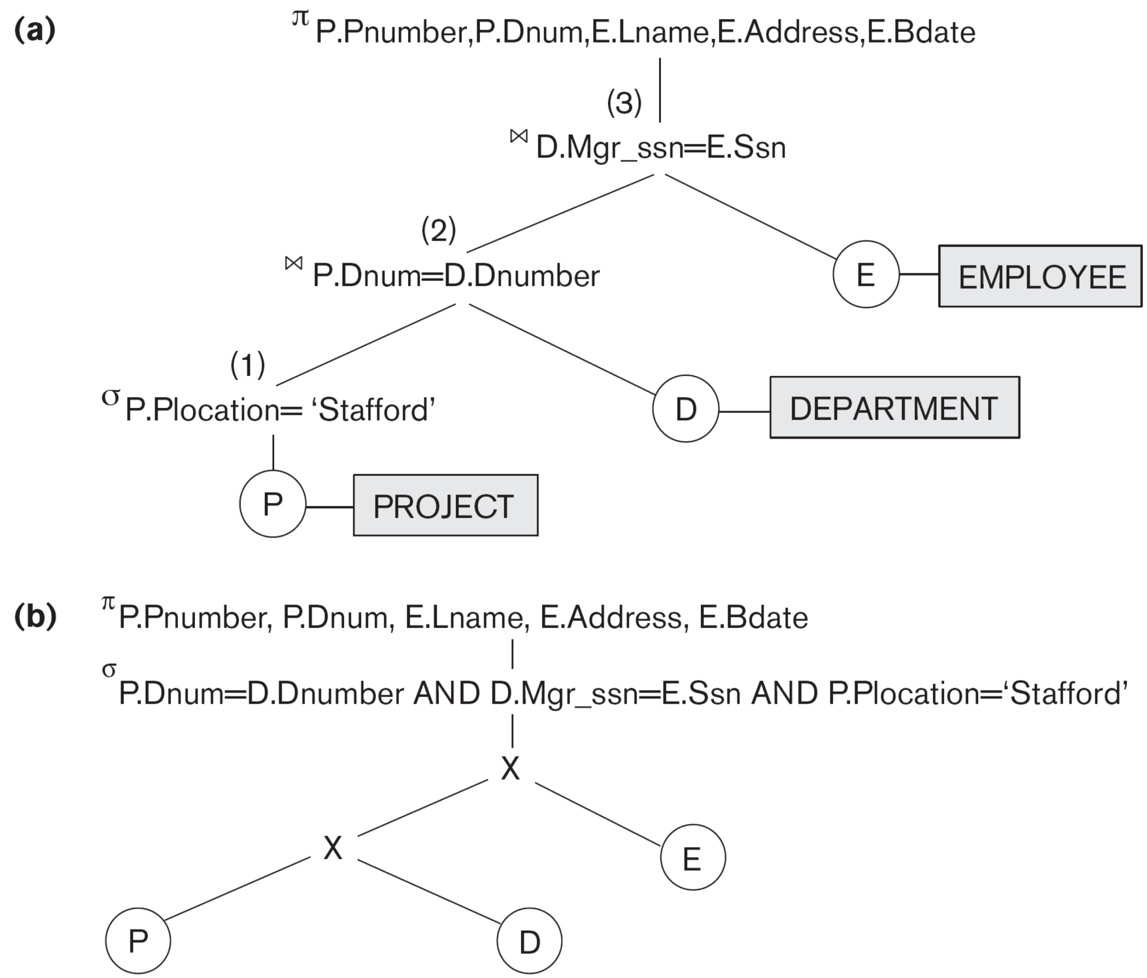
```
Q2:  SELECT   P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
     FROM     PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E
     WHERE    P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND
              P.Plocation= 'Stafford';
```

Using Heuristics in Query Optimization

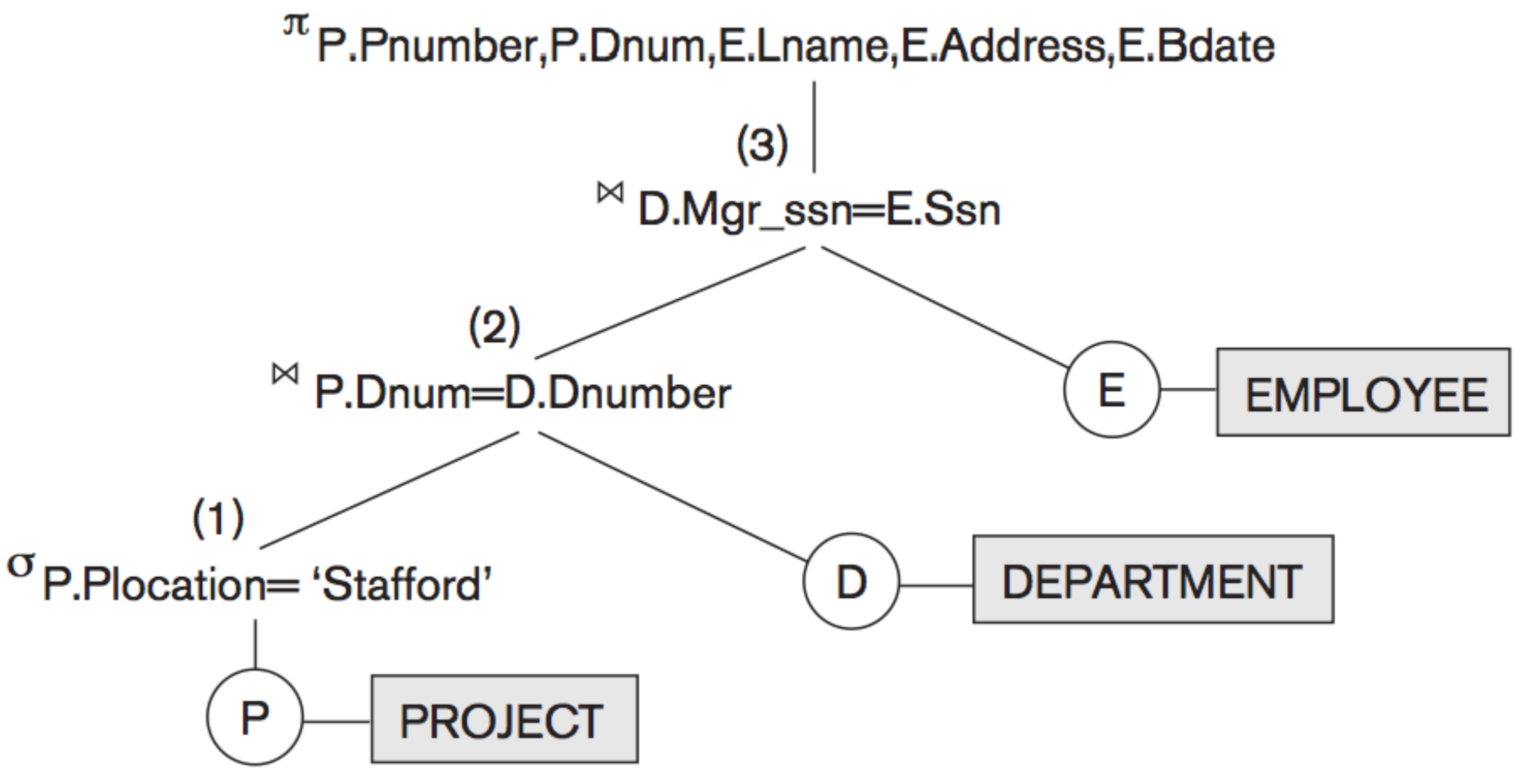
(b)



Using Heuristics in Query Optimization



Using Heuristics in Query Optimization



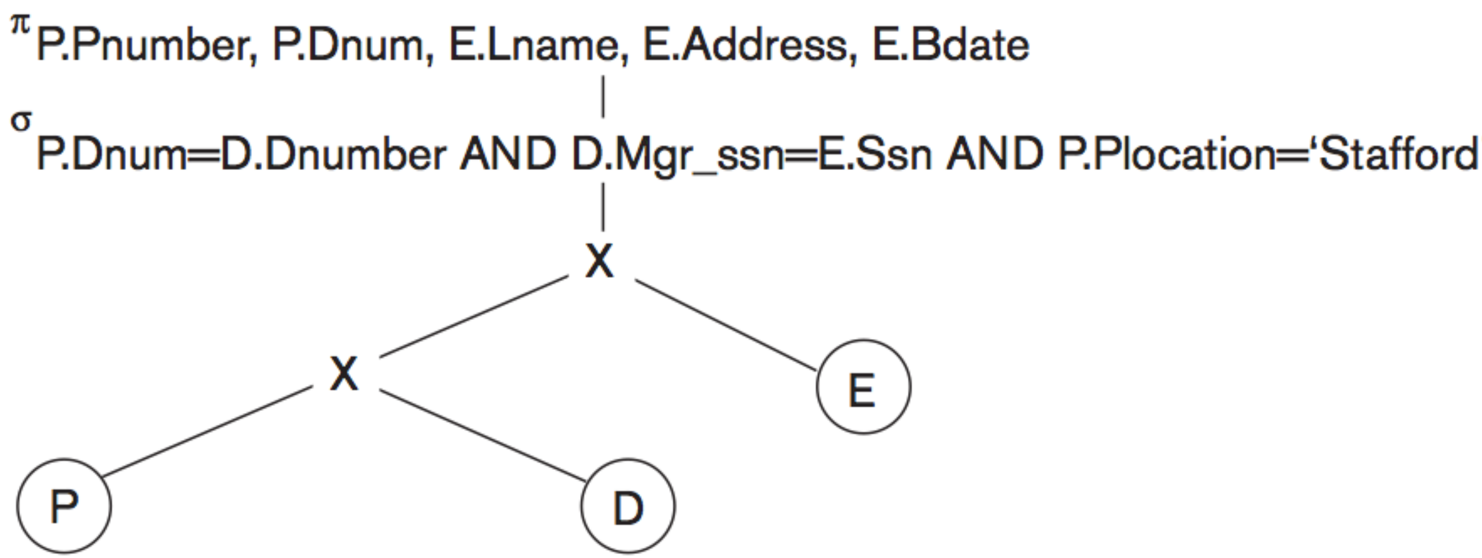
Query tree corresponding to the relational algebra expression for Q2.

Using Heuristics in Query Optimization

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

Using Heuristics in Query Optimization

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



Initial (canonical) query tree for SQL query Q2.

Using Heuristics in Query Optimization

- 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.
- Example:

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

Left-deep Tree Examples

Figure 19.7
Two left-deep (JOIN) query trees.

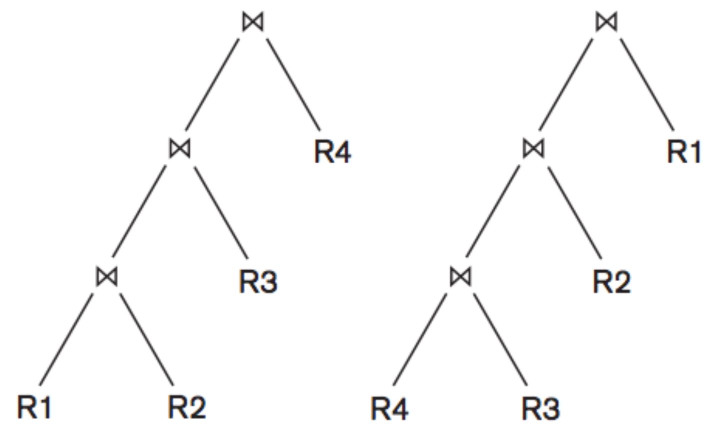
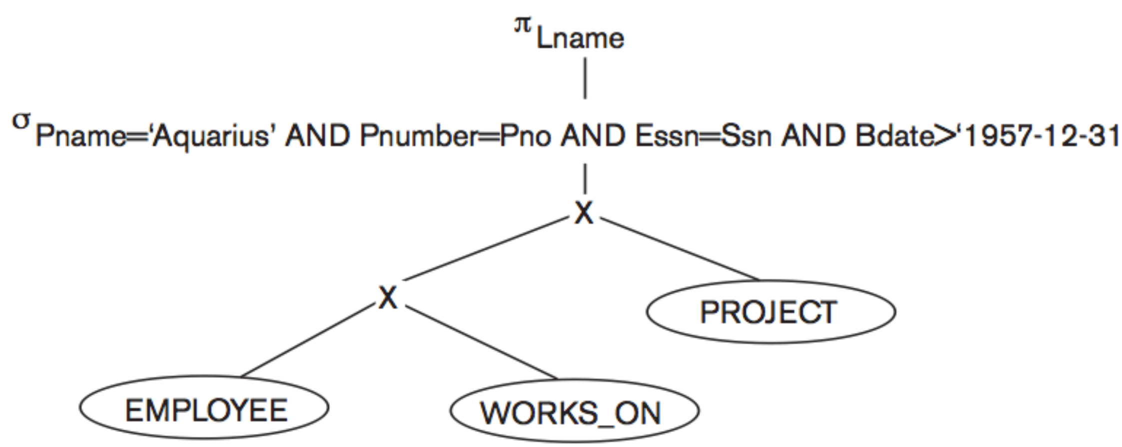
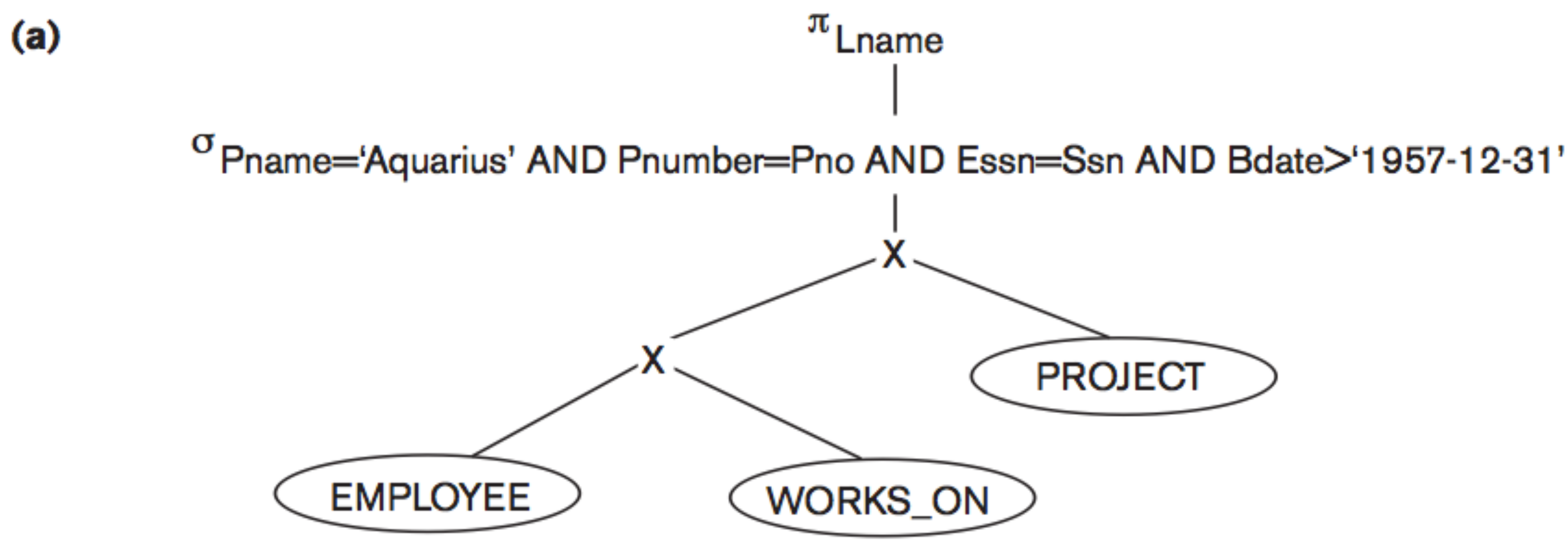


Figure 19.5 (a)
Query Q

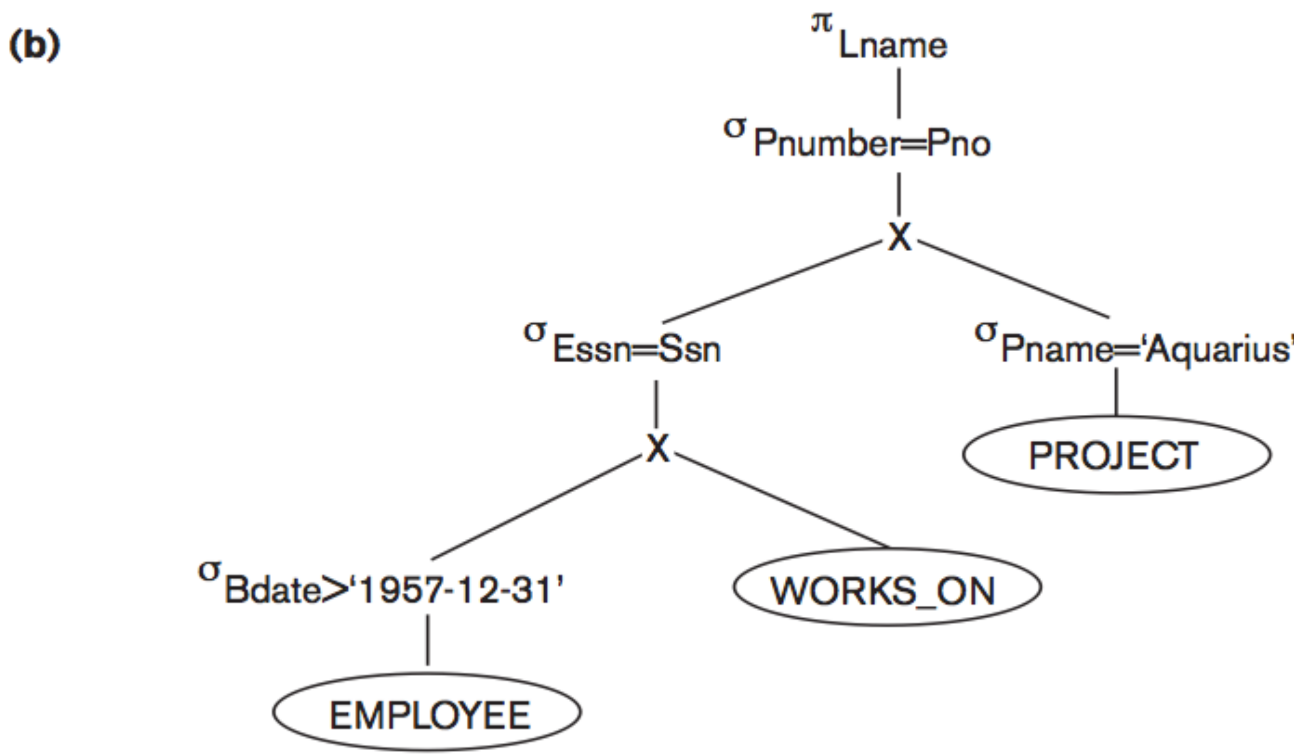


Steps in converting a query tree during heuristic optimization



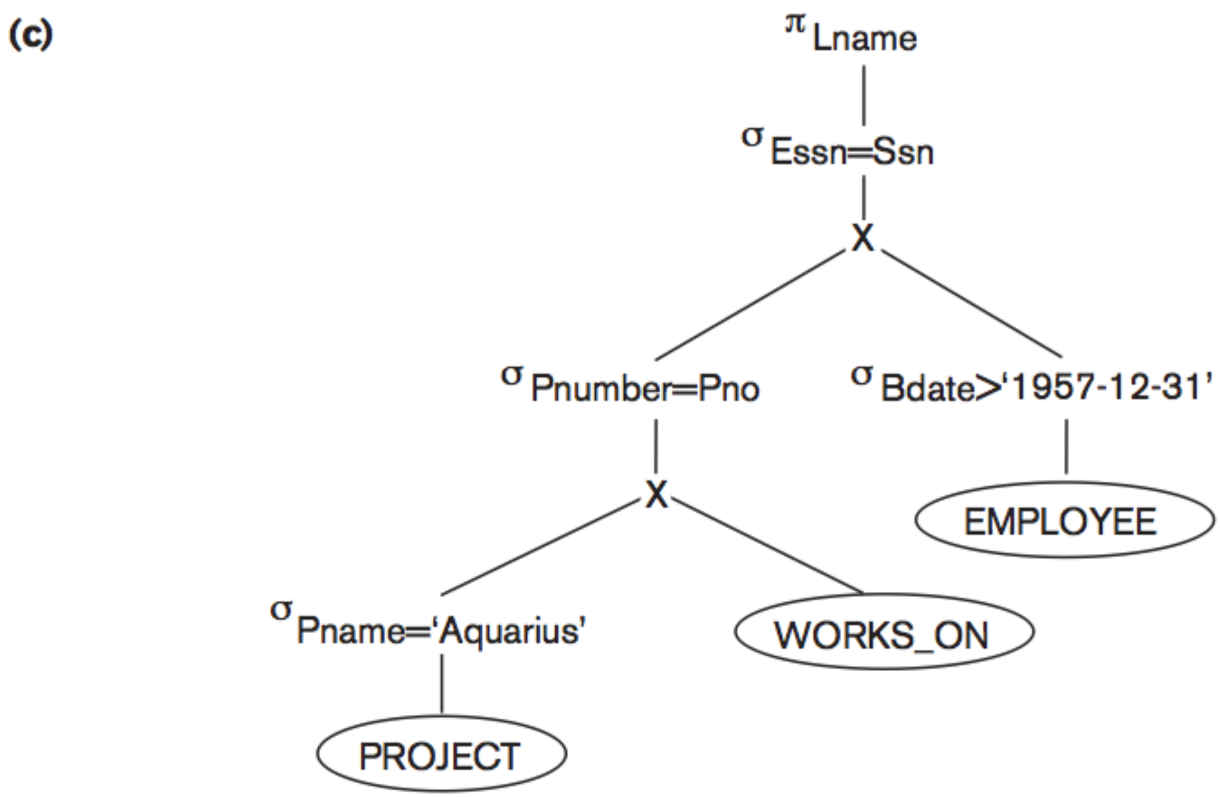
(a) Initial (canonical) query tree for SQL query Q.

Steps in converting a query tree during heuristic optimization



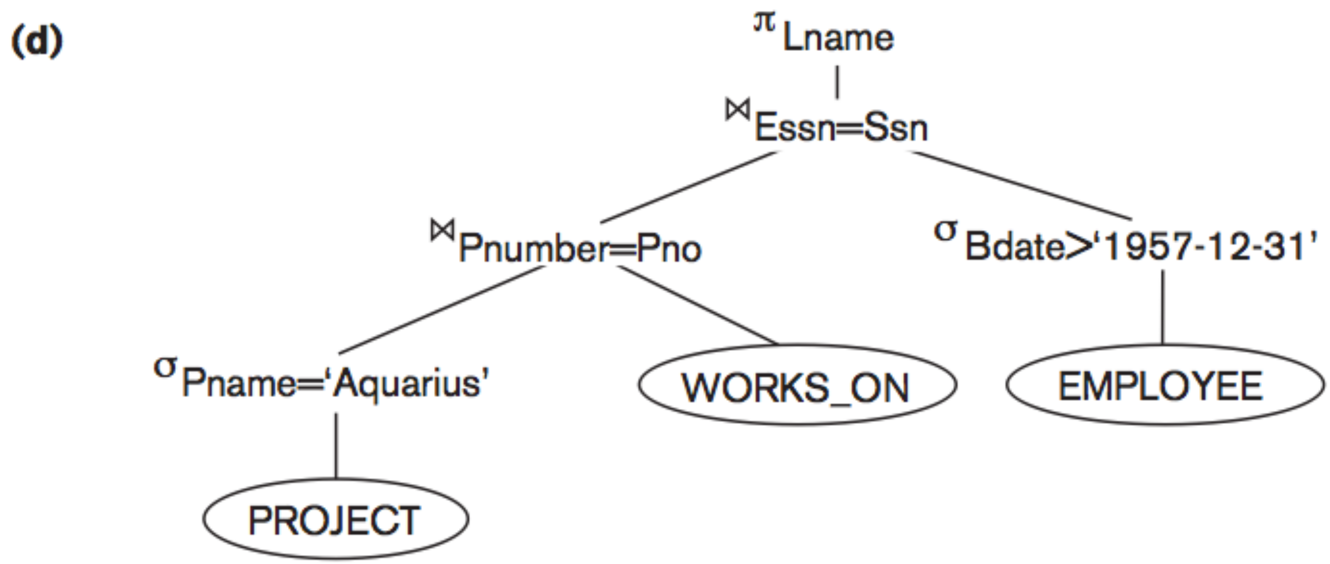
(b) Moving SELECT operations down the query tree.

Steps in converting a query tree during heuristic optimization



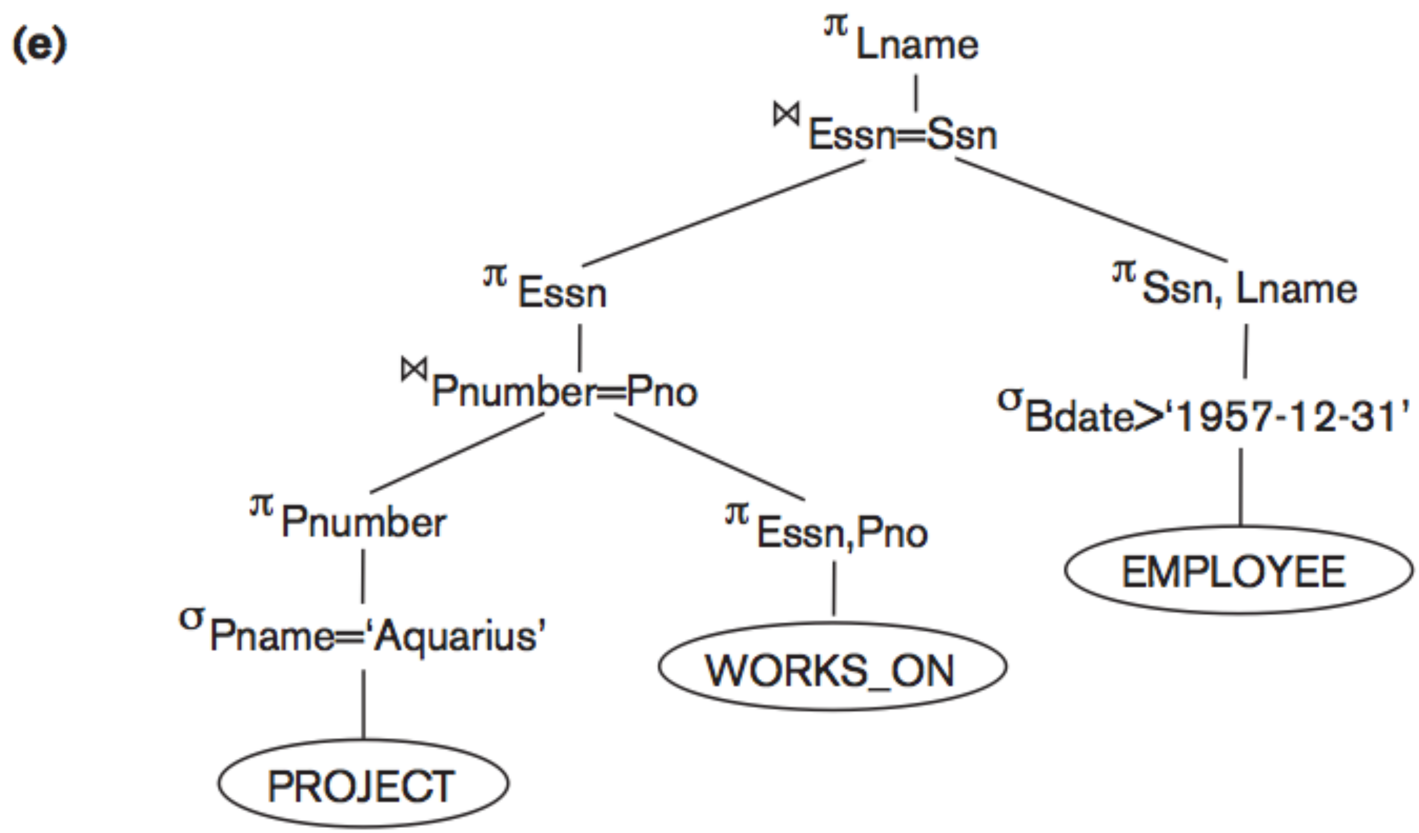
(c) Applying the more restrictive SELECT operation first.

Steps in converting a query tree during heuristic optimization



(d) Replacing CARTESIAN PRODUCT + SELECT with JOIN operations.

Steps in converting a query tree during heuristic optimization



(e) Moving PROJECT operations down the query tree.

19.8 – Using Selectivity and Cost Estimates in Query Optimization

Using Selectivity and Cost Estimates in Query Optimization

- **Cost-based query optimization**:
  - Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.
  - (Compare to heuristic query optimization)
- Considerations and Issues
  - Cost function
  - Number of execution strategies to be considered

Using Selectivity and Cost Estimates in Query Optimization

- **Cost Components for Query Execution**
  1. Access cost to secondary storage
  2. Storage cost
  3. Computation cost
  4. Memory usage cost
  5. Communication cost

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

Cost Components for Query Execution

- **1) Access cost to secondary storage**
  - This is the cost of transferring (reading and writing) data blocks between secondary disk storage and main memory buffers.
  - This is also known as disk I/O (input/output) cost.
  - The cost of searching for records in a disk file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes.
  - In addition, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.

Cost Components for Query Execution

- **2) Disk storage cost**
  - This is the cost of storing any intermediate files on disk that are generated by an execution strategy for the query.
- **3) Computation cost**
  - This is the cost of performing in-memory operations on the records within the data buffers during query execution.
  - Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values.
  - This is also known as *CPU (central processing unit)* cost.

Cost Components for Query Execution

- **4) Memory usage cost**
  - This is the cost pertaining to the number of main memory buffers needed during query execution.
- **5) Communication cost**
  - This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated.
  - In distributed databases (see Chapter 25), it would also include the cost of transferring tables and results among various computers during query evaluation.

Using Selectivity and Cost Estimates in Query Optimization

- Catalog Information used in Cost Functions
  - To estimate the costs of various execution strategies, we must keep track of any information that is needed for the cost functions.
  - This information may be stored in the DBMS catalog, where it is accessed by the query optimizer.
  - **First**, we must know the size of each file. For a file whose records are all of the same type, the
    - **number of records (tuples) (*r*)**,
    - the (average) **record size (*R*)**, and
    - the **number of file blocks (*b*)** (or close estimates of them) are needed.
    - The **blocking factor (*bfr*)** for the file may also be needed.

Using Selectivity and Cost Estimates in Query Optimization

- Catalog Information used in Cost Functions
- Information about the size of a **file**
  - B – block size (in kb)
  - R – record size (in kb)
  - r – number of records (tuples)
  - b – number of blocks
  - bfr – blocking factor ($\lfloor B/R \rfloor$ records per block)

Using Selectivity and Cost Estimates in Query Optimization

- Catalog Information used in Cost Functions
- Information about **indices** and **indexing attributes** of a file
  - *r* – Number of tuples (rows)
  - *sl* – Selectivity of an attribute
    - fraction of records satisfying an equality condition
  - *s* – Selection cardinality of an attribute. (s = sl * r)
  - *d* – Number of distinct values of an attribute
  - *x* – Number of levels of each multilevel index
  - b_{I1} – Number of first-level index blocks
    - i.e. "root" level

Using Selectivity and Cost Estimates in Query Optimization

- Catalog Information used in Cost Functions
- Information about **joins**
  - $|R \bowtie S|$, $|R * S|$, $|R \times S|$ – join cardinality
  - js – join selectivity
    - The ratio of the expected size of the join result divided by the maximum size $n_R * n_S$ of the join

Using Selectivity and Cost Estimates in Query Optimization

Examples of Cost Functions for SELECT

- **S1**. Linear search (brute force) approach
  - $C_{S1a} = b$;
  - For an equality condition on a key, $C_{S1a} = (b/2)$ if the record is found; otherwise $C_{S1a} = b$.
- **S2**. Binary search:
  - $C_{S2} = \log_2 b + \lceil (s/bfr) \rceil - 1$
  - For an equality condition on a unique (key) attribute, $C_{S2} = \log_2 b$
- **S3**. Using a primary index (S3a) or hash key (S3b) to retrieve a single record
  - $C_{S3a} = x + 1$; $C_{S3b} = 1$ for static or linear hashing;
  - $C_{S3b} = 1$ for extendible hashing;

Examples of Cost Functions for SELECT (contd.)

- **S4**. Using an ordering index to retrieve multiple records:
  - For the comparison condition on a key field with an ordering index, $C_{S4} = x + (b/2)$
- **S5**. Using a clustering index to retrieve multiple records:
  - $C_{S5} = x + \lceil (s/bfr) \rceil$
- **S6**. Using a secondary (B+-tree) index:
  - For an equality comparison, $C_{S6a} = x + s$;
  - For an comparison condition such as >, <, >=, or <=,
  - $C_{S6a} = x + (b_{I1}/2) + (r/2)$

Using Selectivity and Cost Estimates in Query Optimization

Examples of Cost Functions for SELECT (contd.)

- **S7**. Conjunctive selection:
  - Use either S1 or one of the methods S2 to S6 to solve.
  - For the latter case, use one condition to retrieve the records and then check in the memory buffer whether each retrieved record satisfies the remaining conditions in the conjunction.
- **S8**. Conjunctive selection using a composite index:
  - Same as S3a, S5 or S6a, depending on the type of index.
- Examples of using the cost functions.

Using Selectivity and Cost Estimates in Query Optimization

Examples of Cost Functions for JOIN (contd.)

- **J1**. Nested-loop join:
  - Using R for outer loop
  - $C_{J1} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$
- **J2**. Single-loop join (using an access structure to retrieve the matching record(s))
  - If an index exists for the join attribute B of S with index levels x_B, we can retrieve each record s in R and then use the index to retrieve all the matching records t from S that satisfy $t[B] = s[A]$.
  - The cost depends on the type of index.

Using Selectivity and Cost Estimates in Query Optimization

Examples of Cost Functions for JOIN (contd.)

- **J2**. Single-loop join (contd.)
  - For a **secondary index** on *S*,
    - $C_{J2a} = b_R + (|R| * (x_B + 1 + s_B)) + ((js * |R| * |S|)/bfr_{RS})$
  - For a **clustering index** on *S*,
    - $C_{J2b} = b_R + (|R| * (x_B + (s_B / bfr_B))) + ((js * |R| * |S|)/bfr_{RS})$;
  - For a **primary index** on *S*,
    - $C_{J2c} = b_R + (|R| * (x_B + 1)) + ((js * |R| * |S|)/bfr_{RS})$
  - If a hash key exists for one of the two join attributes — *B* of *S*
    - $C_{J2d} = b_R + (|R| * h) + ((js * |R| * |S|)/bfr_{RS})$
- **J3**. Sort-merge join:
    - $C_{J3a} = C_S + b_R + b_S + ((js * |R| * |S|)/bfr_{RS})$
    - (C_S: Cost for sorting files)

Example: Join Cost

- Suppose that we have the EMPLOYEE file described in the example in the previous section (of the textbook, per p.717), and
- Assume that the DEPARTMENT file in Figure 3.5 consists of $r_D = 125$ records stored in $b_D = 13$ disk blocks
  - EMPLOYEE cost values are from the table in Figure 19.8 (book never says so)
  - DEPARTMENT cost values are given (different than Figure 19.8)
- Consider the following two join operations:
  - OP6: EMPLOYEE $\bowtie_{\text{Dno=Dnumber}}$ DEPARTMENT
  - OP7: DEPARTMENT $\bowtie_{\text{Mgr\_ssn=Ssn}}$ EMPLOYEE

Example: Join Cost

- Suppose that we
  - primary index on Dnumber of DEPARTMENT with $x_{\text{Dnumber}} = 1$ level
  - secondary index on Mgr_ssn of DEPARTMENT with selection cardinality $s_{\text{Mgr\_ssn}} = 1$ and levels $x_{\text{Mgr\_ssn}} = 2$.
- Assume that the join selectivity for OP6 is $js_{\text{OP6}} = (1 / |\text{DEPARTMENT}|) = 1/125$
  - because Dnumber is a key of DEPARTMENT.
- Also assume that the blocking factor for the resulting join file is $bfr_{\text{ED}} = 4$ records per block

1. Using method J1 with EMPLOYEE as outer loop:

$$C_{J1} = b_E + (b_E * b_D) + ((js_{OP6} * r_E * r_D)/bfr_{ED})$$
$$= 2000 + (2000 * 13) + (((1/125) * 10{,}000 * 125)/4) = 30{,}500$$

2. Using method J1 with DEPARTMENT as outer loop:

$$C_{J1} = b_D + (b_E * b_D) + ((js_{OP6} * r_E * r_{D)}/bfr_{ED})$$
$$= 13 + (13 * 2000) + (((1/125) * 10{,}000 * 125/4) = 28{,}513$$

3. Using method J2 with EMPLOYEE as outer loop:

$$C_{J2c} = b_E + (r_E * (x_{\text{Dnumber}} + 1)) + ((js_{OP6} * r_E * r_D)/bfr_{ED}$$
$$= 2000 + (10{,}000 * 2) + (((1/125) * 10{,}000 * 125/4) = 24{,}500$$

4. Using method J2 with DEPARTMENT as outer loop:

$$C_{J2a} = b_D + (r_D * (x_{\text{Dno}} + s_{\text{Dno}})) + ((js_{OP6} * r_E * r_D)/bfr_{ED})$$
$$= 13 + (125 * (2 + 80)) + (((1/125) * 10{,}000 * 125/4) = 12{,}763$$