Lecture 11: Query Optimization

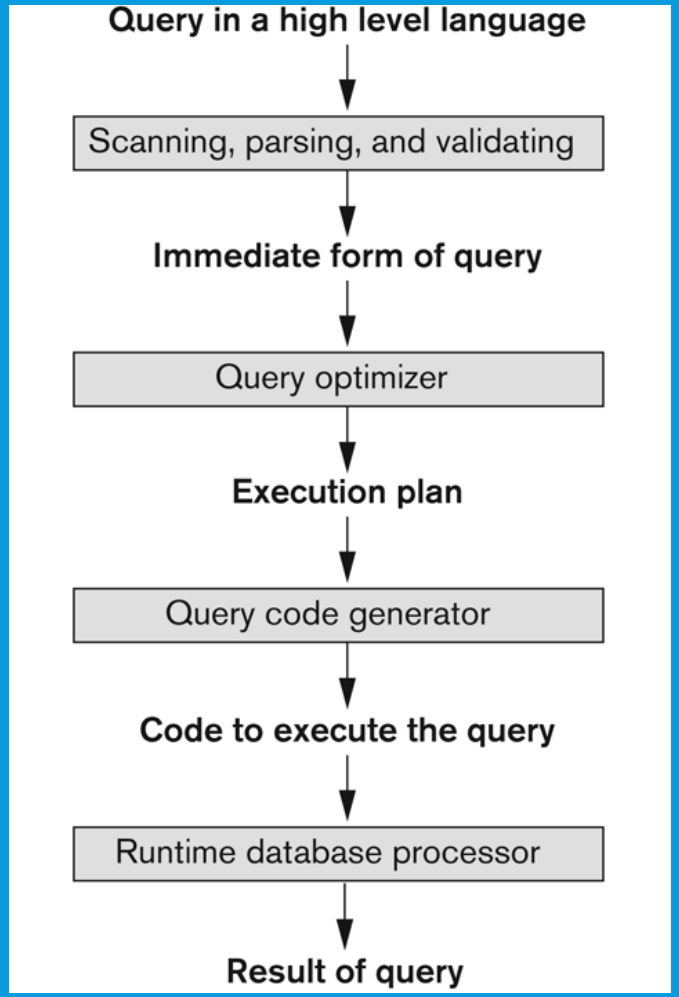
CS3402 Database Systems

Introduction to Query Optimization (1/2)

- Query optimization is the process of choosing a suitable execution strategy for processing a query
- It may not be optimal but is a reasonably efficient strategy (better)
- A query, e.g., a SQL, first be scanned, parsed and validated
  - The scanner identifies the query tokens, e.g., the keywords, attribute names and relation names
  - The parser checks the query syntax
  - The validation checks that all attributes and relation names are valid
- Two internal representations of a query: Query Tree and Query Graph

Introduction to Query Optimization (2/2)

- Code can be
  - Executed directly (interpreted mode)
  - Stored and executed later whenever needed (compiled mode)
- The figure depicts typical steps when processing a high-level query



Translating SQL Queries into Relational Algebra (1/2)

- Query block is the basic unit that can be translated into the algebraic operators
- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block
- Nested queries (or sub-queries) within a query are identified as separate query blocks

Translating SQL Queries into Relational Algebra (2/2)

```
SELECT  LNAME, FNAME
FROM    EMPLOYEE
WHERE   SALARY  >  ( SELECT MAX (SALARY)
                     FROM    EMPLOYEE
                     WHERE   DNO = 5 );
```

```
SELECT  LNAME, FNAME
FROM    EMPLOYEE
WHERE   SALARY > C
```

```
SELECT  MAX (SALARY)
FROM    EMPLOYEE
WHERE   DNO = 5
```

$\pi_{LNAME,\ FNAME}\ (\sigma_{SALARY>C}(EMPLOYEE))$

$\mathcal{F}_{MAX\ SALARY}\ (\sigma_{DNO=5}\ (EMPLOYEE))$

Implementing the Select Operation

- Simple SELECT statement, e.g.,
  - (OP1): $\sigma_{SSN='123456789'}$ (EMPLOYEE)
  - (OP2): $\sigma_{DNUMBER>5}$ (DEPARTMENT)
  - (OP3): $\sigma_{DNO=5}$ (EMPLOYEE)
- Conjunctive SELECT, e.g.,
  - (OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}$ (EMPLOYEE)
  - (OP5): $\sigma_{ESSN='123456789'\ AND\ PNO=10}$ (WORKS_ON)
- Disjunctive SELECT, e.g.,
  - (OP4'): $\sigma_{DNO=5\ OR\ SALARY>30000\ OR\ SEX=F}$ (EMPLOYEE)

Search Methods for Selection (1/8)

- S1: Linear search (unordered file)
  - Retrieve every record in the file sequentially
  - Test whether its attribute value satisfies the selection condition
  - Each disk block is read into a main memory buffer and then a search through the records within the main memory buffer
- S2: Binary search (ordered file)
  - Condition: the selection condition involves an equality comparison on a key attribute on which the file is ordered
  - The records of the file are sorted according to the value of the key
  - Op1: $\sigma_{SSN='123456789'}$ (EMPLOYEE) if ssn is the ordering attribute for the employee file

Search Methods for Selection (2/8)

- S3: Using a primary index or hash key to retrieve a single record
  - The index/hash function tells the locations of the records
  - Condition: the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key)
  - For example, ssn='123456789' in OP1 (EMPLOYEE), we can use the primary index (or the hash key) to directly retrieve the record

Search Methods for Selection (3/8)

- S4: Using a primary index to retrieve multiple records
  - The index tells the locations of the records (a sparse index on an ordered file)
  - Condition: the comparison condition is >, ≥, <, or ≤ on a key field with a primary index
  - For example, dnumber > 5 in OP2, $\sigma_{DNUMBER>5}$ (DEPARTMENT)
  - Use the index to find the record satisfying the corresponding equality condition (dnumber = 5); then retrieve all subsequent records in the (ordered) file
  - For the condition dnumber < 5, retrieve all the preceding records

Search Methods for Selection (4/8)

- S5: Using a clustering index to retrieve multiple records
  - Clustering index: the file records are physically ordered on a non-key field (duplicated values)
  - The index tells the locations of the records
  - Condition: the selection condition involves an equality comparison on a non-key attribute with a clustering index
  - For example, dno = 5 in OP3, $\sigma_{DNO=5}$ (EMPLOYEE)
  - Note dno is NOT a key for EMPLOYEE
  - Use the clustering index to retrieve all the records satisfying the selection condition

Search Methods for Selection (5/8)

- S6: Using a secondary index or B^+-tree
  - The secondary index may be created on a field that is a candidate key and has a unique value in every record, or a non-key field with duplicated values
  - The index tells the location of the records
  - We can retrieve records on conditions involving >,>=, <, or <=. (e.g., range queries)
  - Range query example: 30000<=salary<=35000

Example of a Secondary Index

- Using blocks of record pointers for handling same value non-key records

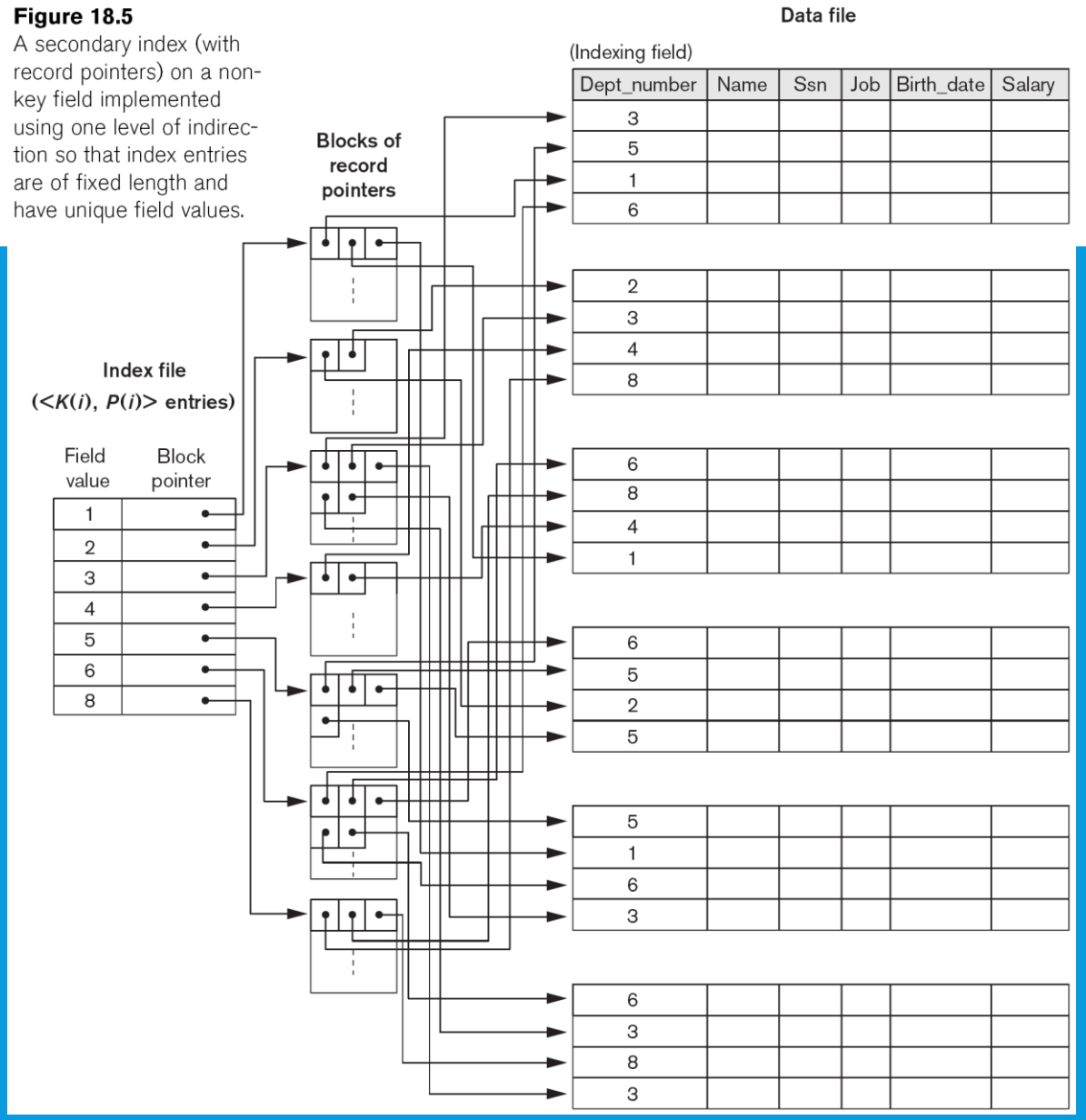


Figure 18.5
A secondary index (with record pointers) on a non-key field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

Search Methods for Selection (6/8)

- S7: Conjunctive selection
  - A conjunctive condition contains several simple conditions connected with AND
  - Condition: an attribute has an access path that permits the use of one of the methods S2 to S6
    - Binary search
    - Hash key
    - Primary index, Clustering index, Secondary index
    - B-tree and B^+-tree index
  - Use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition
  - (OP4): $\sigma_{\text{DNO=5 AND SALARY>30000 AND SEX=F}}$ (EMPLOYEE)

Search Methods for Selection (7/8)

- S8: Conjunctive selection using a composite index
  - E.g., if an index has been created on the composite key (e.g., Essn, Pno) of the WORK_ON file
  - Condition: two or more attributes are involved in equality conditions in the conjunctive condition and a composite index exists on the combined fields
  - If an index has been created on the composite key (essn, pno) of the WORK_ON file, we can use the index directly
  - $\sigma_{\text{ESSN='123456789' AND PNO=10}}(\text{WORKS\_ON})$

Search Methods for Selection (8/8)

- S9: Conjunctive selection by intersection of record pointers
  - Condition: secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the conjunctive condition and the indexes include record pointers (rather than block pointers)
  - Index pointing to the records (dense index)
  - Method:
    - Each index can be used to retrieve the record pointers that satisfy the individual condition
    - The intersection (common) of these sets of record pointers gives the record pointers that satisfy the conjunctive condition
    - E.g., use the indexes to get dno > 5 and salary > 30000

Selection Optimization (1/2)

- In choosing between multiple simple conditions in a conjunctive select condition, it is important to consider the selectivity of each condition
  - Defined as the ratio of the number of records (tuples) that satisfy the condition to the total number of records (tuples) in the file (relation)
  - Smaller selectivity $\rightarrow$ more desirable

Selection Optimization (2/2)

- A disjunctive condition (where simple conditions are connected by OR) is much harder to process and optimize
  For example, (OP4'): $\sigma_{\text{DNO=5 OR SALARY>30000 OR SEX=F}}$ (EMPLOYEE)
- How to optimize?
  - If any one of the conditions does not have an access path, we have to use the brute force linear search approach
  - If an access path exists on *every* condition, we can optimize the selection by retrieving the records satisfying each condition and then applying the union operation to remove duplicate records
  - If the appropriate access paths that provide record pointers exist for every condition, we can union record pointers instead of records

Implementing the Join Operation (1/4)

- The JOIN operation is one of the most time-consuming operations in query processing
- Two-way JOIN:

$$R *_{A=B} S$$

- Multi-way JOIN:

$$R *_{A=B} S *_{C=D} T .....$$

- Example operations:
  - (OP6): EMPLOYEE $*_{DNO=DNUMBER}$ DEPARTMENT
  - (OP7): DEPARTMENT $*_{MGRSSN=SSN}$ EMPLOYEE

Implementing the Join Operation (2/4)

- J1: Nested (inner-outer) loop approach (brute force)
  - For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B]
- J2: Using an access structure to retrieve the matching record(s)
  - If an index exists for one of the two join attributes, say, B of S, retrieve each record t in R, one at a time, and
  - Then use the access structure to directly retrieve all matching records s from S that satisfy s[B] = t[A]

Example: Choice of Outer-Loop (1/2)

- Nested-loop approach (J1)
- (OP6): EMPLOYEE * $_{DNO=DNUMBER}$ DEPARTMENT
- Number of buffers in main memory $n_B = 7$ blocks
- DEPARTMENT file consists of $r_D = 50$ records in $b_D = 10$ blocks
- EMPLOYEE file consists of $r_E = 6{,}000$ records in $b_E = 2{,}000$ blocks
- Read one block at a time for the inner-loop file and use its records to check the outer-loop blocks that are in main memory for matching (5 memory blocks)
- An extra block in main memory is needed to contain the results (1 memory block)

Example: Choice of Outer-Loop (2/2)

- EMPLOYEE or DEPARTMENT as outer-loop?
- If EMPLOYEE is chosen for the outer-loop and each block of EMPLOYEE is read once and the entire DEPARTMENT file is read once for each time we read in (n_B – 2) blocks of EMPLOYEE file
- Total number of blocks accessed for outer-loop file = b_E
- No. of times (n_B – 2) blocks of outer file are loaded into main memory = $b_E/(n_B – 2)$ (i.e., the number of execution times of the inter-loop)
- Total number of blocks accessed for inner-loop file = $b_D * [b_E/(n_B – 2)]$
- Total number of block read accesses = $b_E + b_D * [b_E/(n_B – 2)] = 2,000 + 10 * 2,000/5 = 6,000$ block accesses
- If we use DEPARTMENT in the outer-loop, then $b_D + b_E * [b_D/(n_B – 2)] = 10 + 2,000 * 10/5 = 4,010$ block accesses

Join Selection Factor (1/2)

- Join selection factor: The fraction of records in a file that will be joined with records in the other file
- (OP7): DEPARTMENT * $_{MGRSSN=SSN}$ EMPLOYEE
- Suppose the secondary indexes exist on both the attributes Ssn of EMPLOYEE and Mgr_ssn of DEPARTMENT with the number of index levels $X_{Ssn} = 4$ and $_{XMgr\_ssn} = 2$ (different no. of records, different levels)
- First retrieves each EMPLOYEE record and uses the index on Mgr_ssn of DEPARTMENT to find a matching DEPARTMENT record
- The no. of block accesses = $b_E + (r_E * (x_{Mgr\_ssn} + 1) = 2{,}000 + (6{,}000 * 3) =$ 20,000 block accesses

Join Selection Factor (2/2)

- Retrieves each DEPARTMENT record and uses the index on Ssn of EMPLOYEE to find a matching record
- The no. of block accesses = $b_D + (r_D * (x_{Ssn} + 1) = 10 + (50 * 5) = 260$ block accesses
- Join selection factor of Ssn=Mgr_ssn is 1

Implementing the Join Operation (3/4)

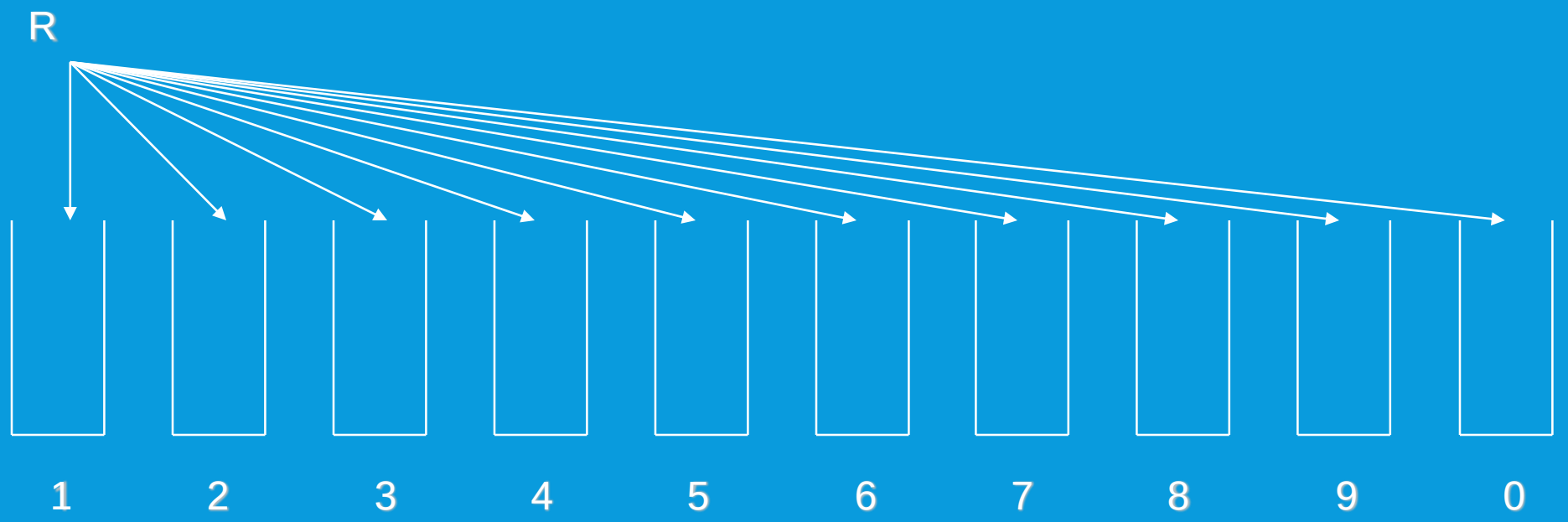
- J3: Sort-merge join
  - Condition: the records of R and S are physically sorted (ordered) by value of the join attributes A and B, respectively
  - Method:
    - Both files are scanned in order of the join attributes, matching the records that have the same values for A and B
    - In this method, if the joining attribute is sorted, the records of each file are scanned only once each for matching with the other file
    - Otherwise, sort the records first before matching. Sorting cost = O(n log n)

Implementing the Join Operation (4/4)

- J4: Hash-join
  - The records of files R and S are hashed using the same hashing function on the join attributes A of R and B of S
- Step 1 (partitioning phase). A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets
- Step 2 (probing phase). A single pass through the other file (S) then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from R

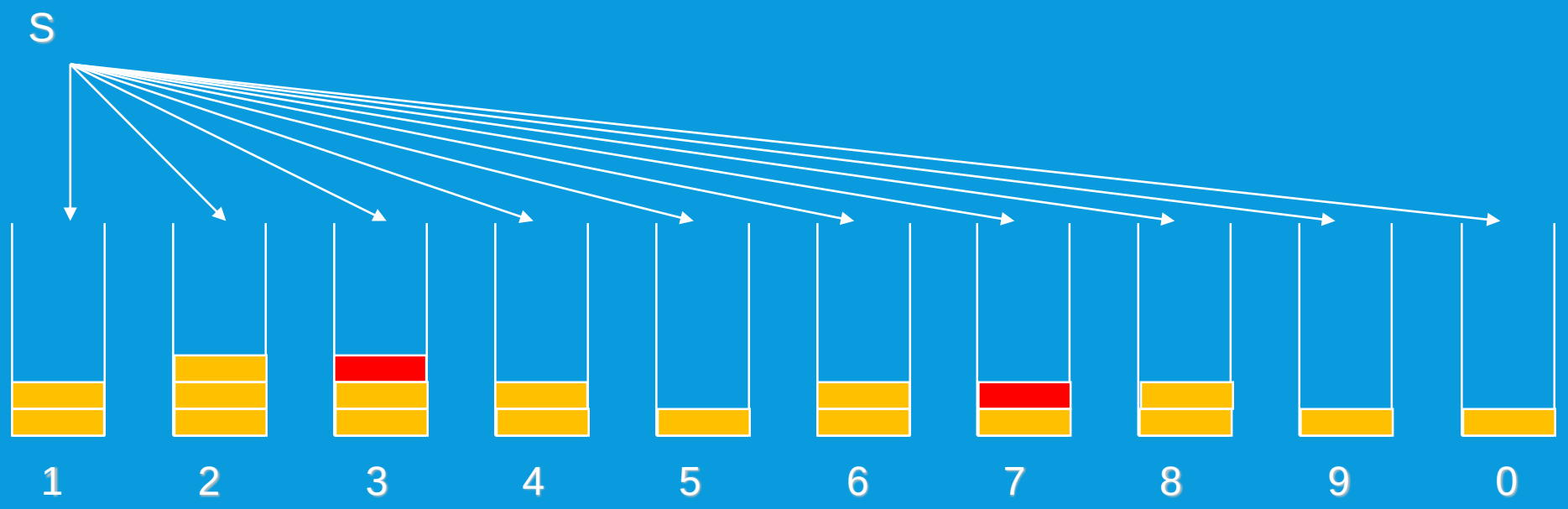
Hash Join: Example (1/2)

- Hash records of R into the buckets



Hash Join: Example (2/2)

- Hash records of S into the buckets to compare with the records of R in the same bucket



Implementing Aggregate Operation

- SUM, COUNT and AVG
- Methods
  - For a dense index: apply the computation to the values in index
  - For a non-dense index: actual number of records associated with each index entry are used for computation (multiple records indexed by an index entry)
- GROUP BY: this operator is applied to subsets of a table. Employee relation is hashed or sorted to partition the file into groups such that each group has the same grouping attribute

```
SELECT      DNUM, AVG(SALARY)
FROM        EMPLOYEE
GROUP BY DNUM;
```

- With clustering index on the grouping attribute: records are already partitioned (grouped) on that attribute

Using Heuristics in Query Optimization (1/4)

- Heuristic rule may be applied to modify the internal representation of a query (e.g., a query tree) to improve performance
- Process for heuristics optimization
  1. The parser 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

Using Heuristics in Query Optimization (2/4)

- The main heuristic is to apply the operations that reduce the size of intermediate results first
  - E.g., SELECT and PROJECT before JOIN or other binary operations
  - The size of the resulting file from a binary operation (e.g., JOIN) is usually a multiplicative function of the sizes of the input files
  - The SELECT and PROJECT operations reduce the size of a file

Query Tree

- A query tree is a tree data structure that corresponds to a relational algebra expression
- The query as leaf nodes represents the input relations
- The order of execution of operations starts at the leaf nodes and ends at the root node
- Example: for every project located in “Stafford”, retrieve the project number, the controlling department number, and the manager’s last name, address and birthdate

Using Heuristics in Query Optimization (3/4)

- Relation algebra:

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

- 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';
```

Using Heuristics in Query Optimization (4/4)

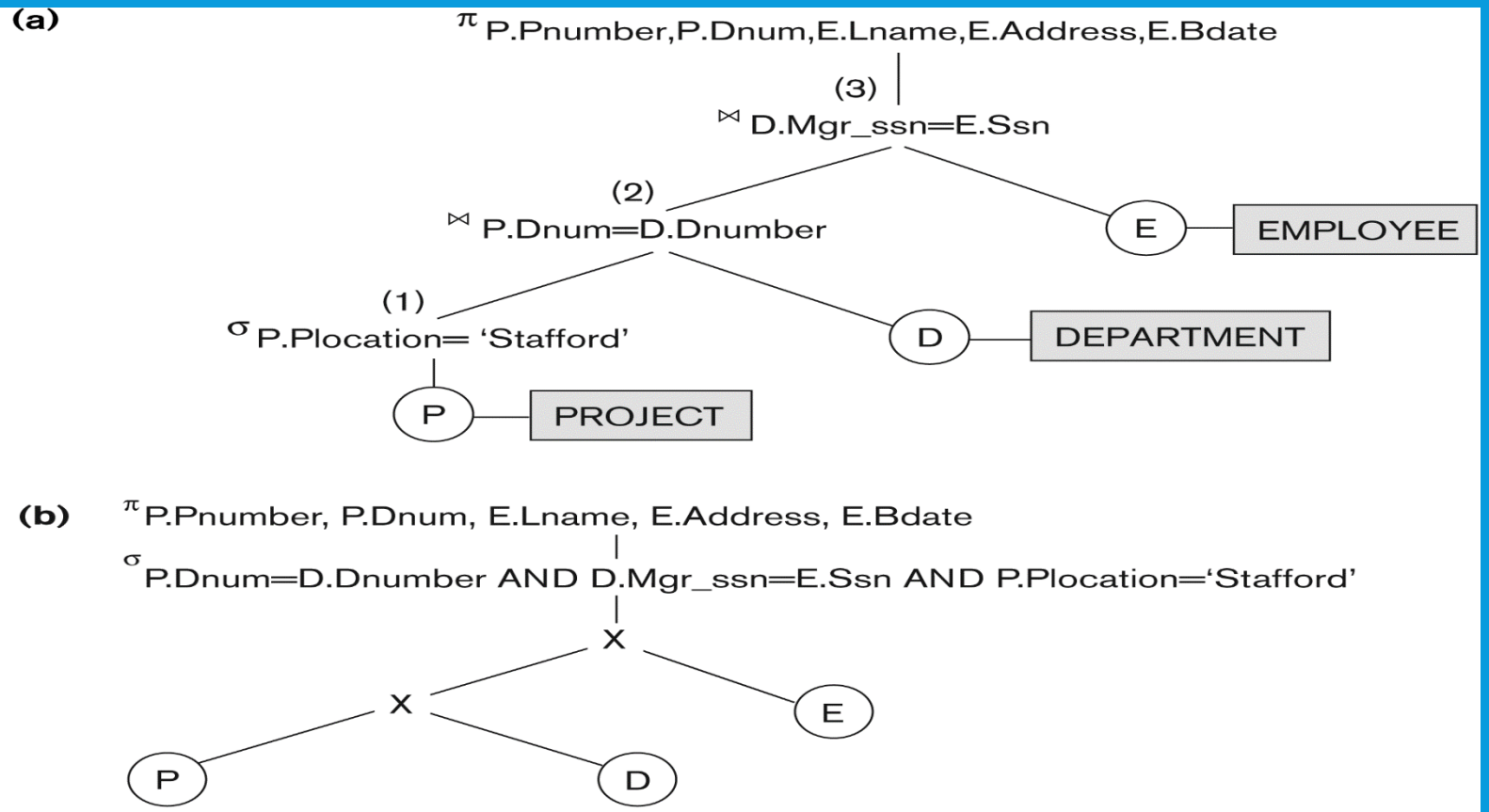


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Why do Heuristic Optimization?

- Example:
  - $\sigma_{Salary>30000}$ (EMPLOYEE * $_{(SSN=MGRSSN)}$ DEPT)
  - ($\sigma_{Salary>30000}$ (EMPLOYEE)) *$_{(SSN=MGRSSN)}$ DEPT
- Which one is better if most of the employees in the company has salary below 30,000?

Heuristic Optimization (1/6)

- 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.)

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

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

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

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

Heuristic Optimization (2/6)

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

   $$\pi \text{ List}_1 (\pi \text{ List}_2 (...(\pi \text{ List}_n(R))...)) = \pi \text{ List}_1(R)$$

4. Commuting σ with π : If the selection condition c involves only the attributes A_1, ..., A_n in the projection list, the two operations can be commuted:

   $$\pi A_1, A_2, ..., A_n (\sigma c (R)) = \sigma c (\pi A_1, A_2, ..., A_n (R))$$

5. Commutativity of * (or $\bowtie$): The * operation is commutative:

   $$R * S = S * R$$

   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 $\bowtie$ (and $\bowtie_c$) operation is commutative in the same sense as the * operation.

Heuristic Optimization (3/6)

6. Commuting σ with * (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:

   σ c (R * S) = (σ c (R)) * 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:

   σ c (R * S) = (σ c1 (R)) * (σ c2 (S))

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

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

Heuristic Optimization (4/6)

8. Commuting π with $\bowtie_c$ (or X): Suppose that the projection list is L = {A_1, ..., A_n, B_1, ..., B_m}, where A_1, ..., A_n are attributes of R and B_1, ..., B_m 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 A_{n+1}, ..., A_{n+k} of R and B_{m+1}, ..., B_{m+p} of S are involved in the join condition c but are not in the projection list L, the operations commute as follows:

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

$$\pi_L ((\pi_{A_1,\ldots,A_n,A_{n+1},\ldots,A_{n+k}}(R)) \bowtie_c (\pi_{B_1,\ldots,B_m,B_{m+1},\ldots,B_{m+p}} (S)))$$

Heuristic Optimization (5/6)

9. Associativity of *, X, U, and ∩ (X: cross product): These four operations are individually associative; that is, if q stands for any one of these four operations (throughout the expression), we have

   (R q S) q T = R q (S q T)

   E.g., (R U S) U T = R U (S U T)

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

    σ c (R q S) = (σ c (R)) q (σ c (S))

11. The π operation commutes with U. If q stands for U, we have

    π_L (R q S) = (π_L (R)) q (π_L (S))

Heuristic Optimization (6/6)

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):

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

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

Outline of a Heuristic Algebraic Optimization Algorithm (1/2)

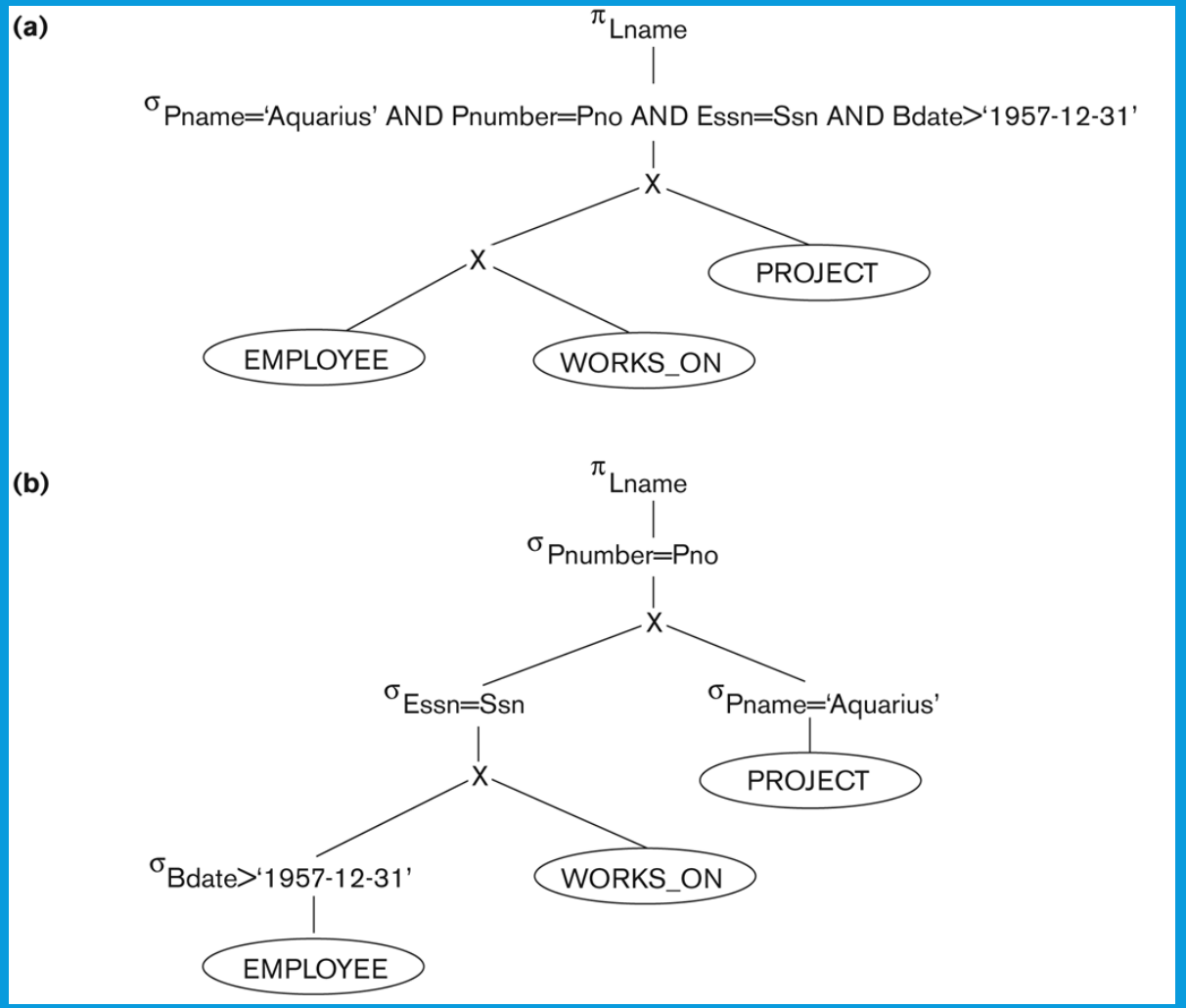
1. Using rule 1, break up any SELECT operations with conjunctive conditions into a cascade of SELECT operations
   - This permits a greater degree of freedom in moving select operations down different branches of the tree
2. Using rules 2, 4, 6, and 10 concerning the commutativity of SELECT with other 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

Outline of a Heuristic Algebraic Optimization Algorithm (2/2)

4. Combine a CROSS PRODUCT operation with a subsequent select operation whose condition represents a join condition into a join operation
5. Using rules 3, 4, 8, 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

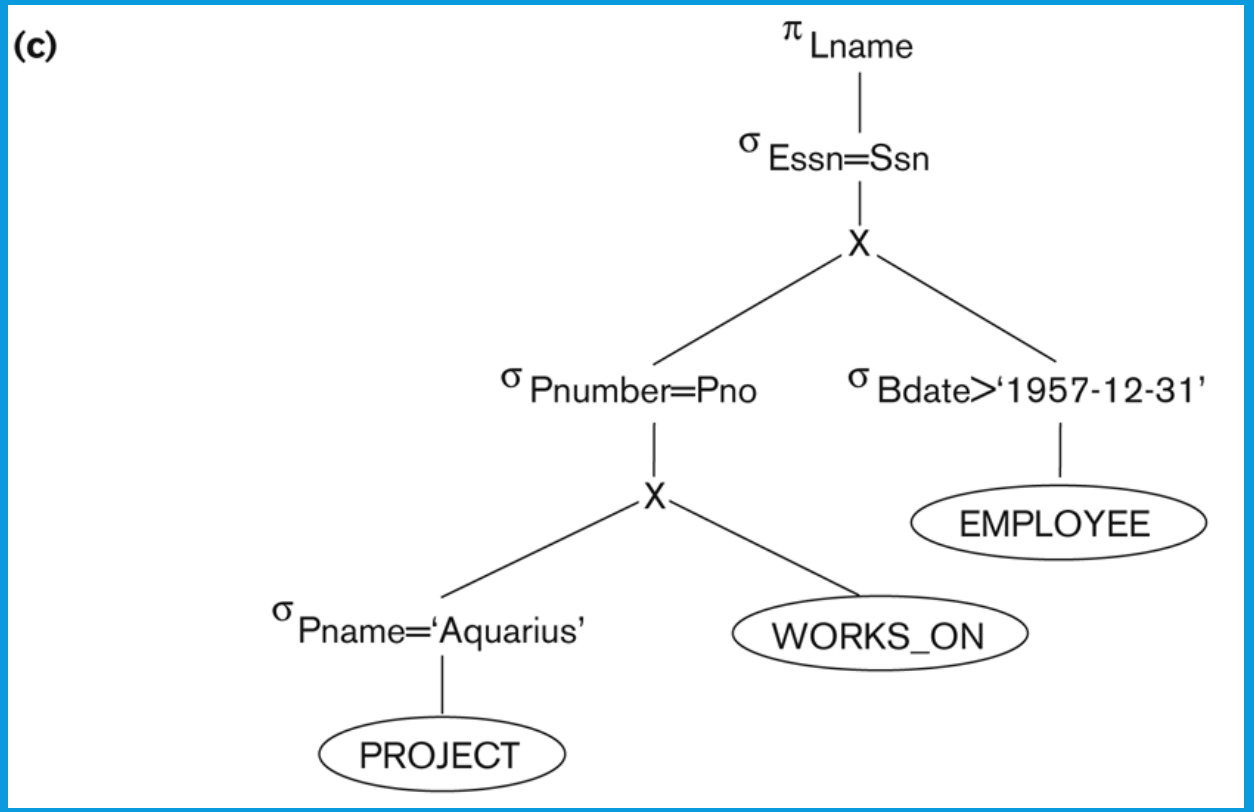
Heuristic Optimization: Example (1/4)

- Figure (a) sows the initial (canonical) query tree for a SQL query
- Figure (b) shows the query tree of after applying Steps 1 and 2 of the algorithm
  - Moving SELECT operations down the query tree



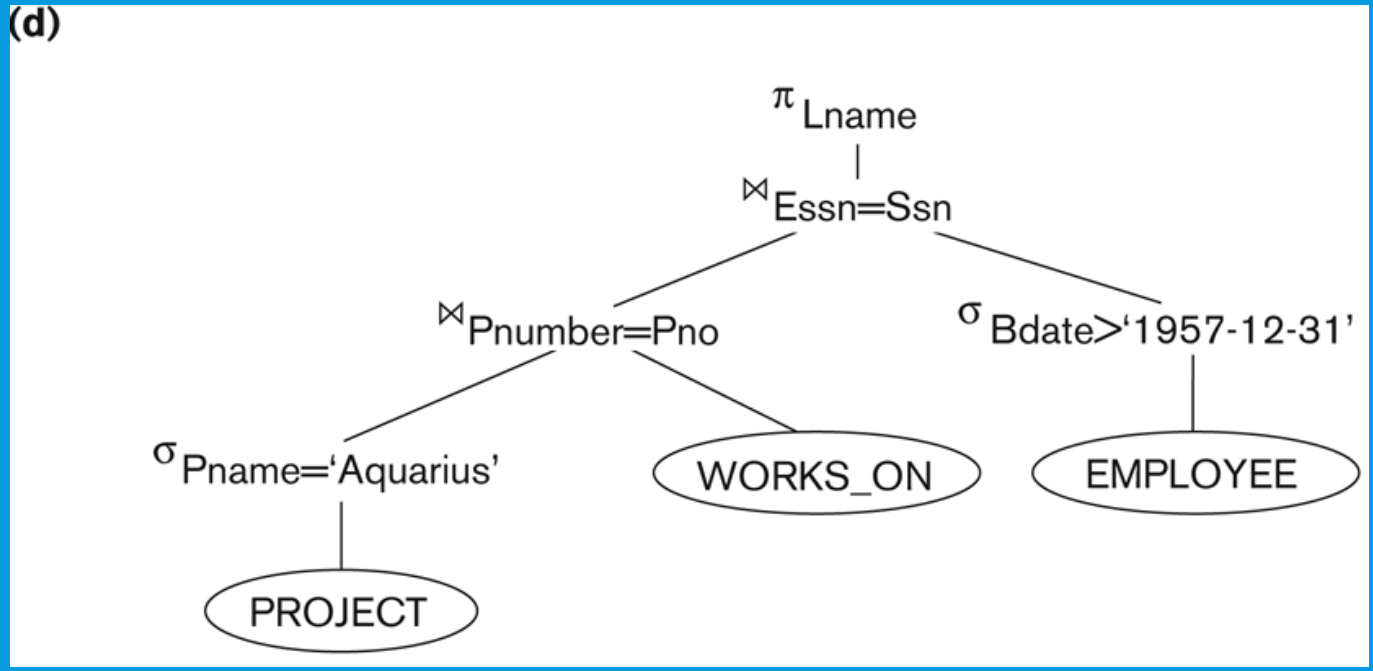
Heuristic Optimization: Example (2/4)

- Figure (c) shows the tree after applying Step 3
  - Applying the more restrictive SELECT operation first



Heuristic Optimization: Example (3/4)

- Figure (d) shows the query tree after applying Step 4
  - Replacing CROSS PRODUCT and SELECT with JOIN operation



Heuristic Optimization: Example (4/4)

- Figure (e) after applying Step 5
  - Moving PROJECT operations down the query tree