CS3402 Database Systems:
Query Optimization

Overview

- **Query optimization**
  - The process of choosing a suitable execution strategy for processing a query.
- Two internal representations of a query:
  - **Query Tree**
  - **Query Graph**

The Query Go Through…

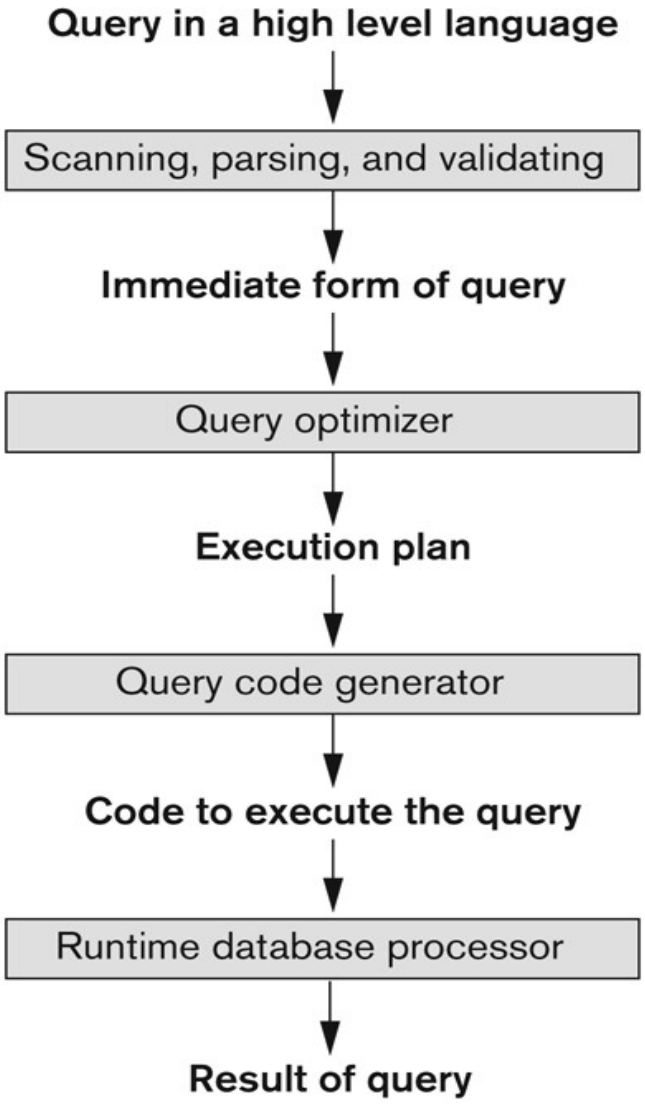


Figure 15.1
Typical steps when processing a high-level query.

Code can be:

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

Translating SQL Queries into Relational Algebra

- **Query block**:
  - The basic unit that can be translated into the algebraic operators and optimized.
- 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** within a query are identified as separate query blocks.
- Aggregate operators in SQL must be included in the extended algebra.

Translating SQL Queries into Relational Algebra

```
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
```

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

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

Implementing the Select Operation

Examples:

(OP1): $\sigma_{SSN='123456789'}$ (EMPLOYEE)

(OP2): $\sigma_{DNUMBER>5}$ (DEPARTMENT)

(OP3): $\sigma_{DNO=5}$ (EMPLOYEE)

(OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}$ (EMPLOYEE)

(OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}$ (WORKS_ON)

Search Methods for Selection

S1. Linear search (brute force)

Retrieve *every record* in the file;

Test whether its attribute values satisfy the selection condition.

S2. Binary search

Condition: the selection condition involves an equality comparison on a key attribute on which the file is ordered

An example is op1 if ssn is the ordering attribute for the employee file.

S3. Using a primary index or hash key to retrieve a single record

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, we can use the primary index (or the hash key) to retrieve the record.

Search Methods for Selection

S4. Using a primary index to retrieve multiple records

Condition: the comparison condition is >, ≥, <, or ≤ on a key field with a primary index

For example, dnumber > 5 in op2, we 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.

S5. Using a clustering index to retrieve multiple records

Condition: the selection condition involves an equality comparison on a nonkey attribute with a clustering index

for example, dno = 5 in op3, we use the clustering index to retrieve all the records satisfying the selection condition.

Search Methods for Selection

S6. Using a secondary index or B^+ tree

Condition: the indexing field has unique values (is a key) or when we want to retrieve multiple records if the indexing field is not a key. In addition, we can retrieve records on conditions involving >,>=, <, or <=. (FOR **RANGE QUERIES**)

RANGE QUERY EXAMPLE:

30000<=salary<=35000.

CONJUNCTIVE QUERY EXAMPLE:

(OP4): σ DNO=5 AND SALARY>30000 AND SEX=F (EMPLOYEE)

Search Methods for Selection

S7. Conjunctive selection

Condition: an attribute involved in any single *simple condition* in the conjunctive condition has an access path that permits the use of one of the methods S2 to S6

Method: use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition

S8. Conjunctive selection using a composite index

Condition: two or more attributes are involved in equality conditions in the conjunctive condition and a composite index exists on the combined fields

For example, if an index has been created on the composite key (essn, pno) of the **works_on** file for op5 we can use the index directly.

Search Methods for Selection

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)

Method:

Each index can be used to retrieve the *record pointers* that satisfy the individual condition.

The *intersection* of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly.

If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.

On Selection Optimization

- Whenever a single condition specifies the selection such as op1, op2, or op3, we only check whether an access path exists on the attribute involved in that condition.
  - If an access path exists, the method corresponding to that access path is used;
  - otherwise, the "brute force" linear search approach of method S1 is used.

- Query optimization for a select operation is needed mostly for conjunctive select conditions whenever *more than one* of the attributes involved in the conditions have an access path. The optimizer should choose the access path that *retrieves the fewest records* in the most efficient way.

On Selection Optimization

- 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).
  - May need to be obtained through “estimation”

- The smaller the selectivity, the fewer tuples the condition selects and the higher the desirability of using that condition first to retrieve records.

On Selection Optimization

- a **disjunctive condition** (where simple conditions are connected by the OR logical connective rather than by AND) is much harder to process and optimize.

  For example, consider OP4':

  (OP4'): $\sigma_{\text{DNO=5 OR SALARY>30000 OR SEX=F(EMPLOYEE)}}$

  When can we optimize?

  - If any *one* of the conditions does not have an access path, we have to use the brute force linear search approach.
  - When 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

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

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

- Join **Department** table with **Employee** table to retrieve "Work for" relationship
- If there is a clustering index on "Dno" in Employee table
- What can you do?

Implementing the Join Operation

J3. **Sort-merge join**

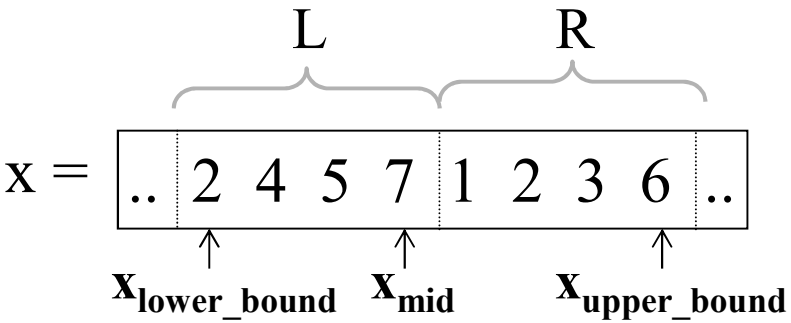
Condition: the records of R and S are *physically sorted* (ordered) by value of the join attributes A and B, respectively

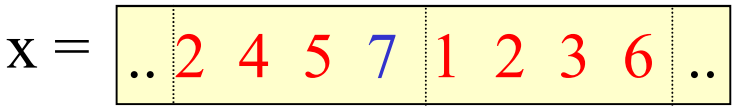
Method:

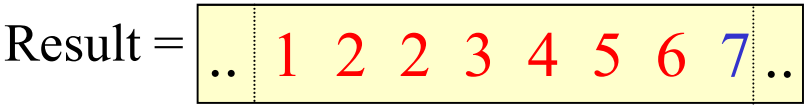
- Both files are scanned in order of the join attributes, matching the records that have the same values for A and B.
- In this method, if the joining attribute is a key, the records of each file are scanned only once each for matching with the other file
- How about non-key?

Sort-Merge Example

- Numbers are key values
- How to deal with non-key?







Implementing the Join Operation

J4. **Hash-join**

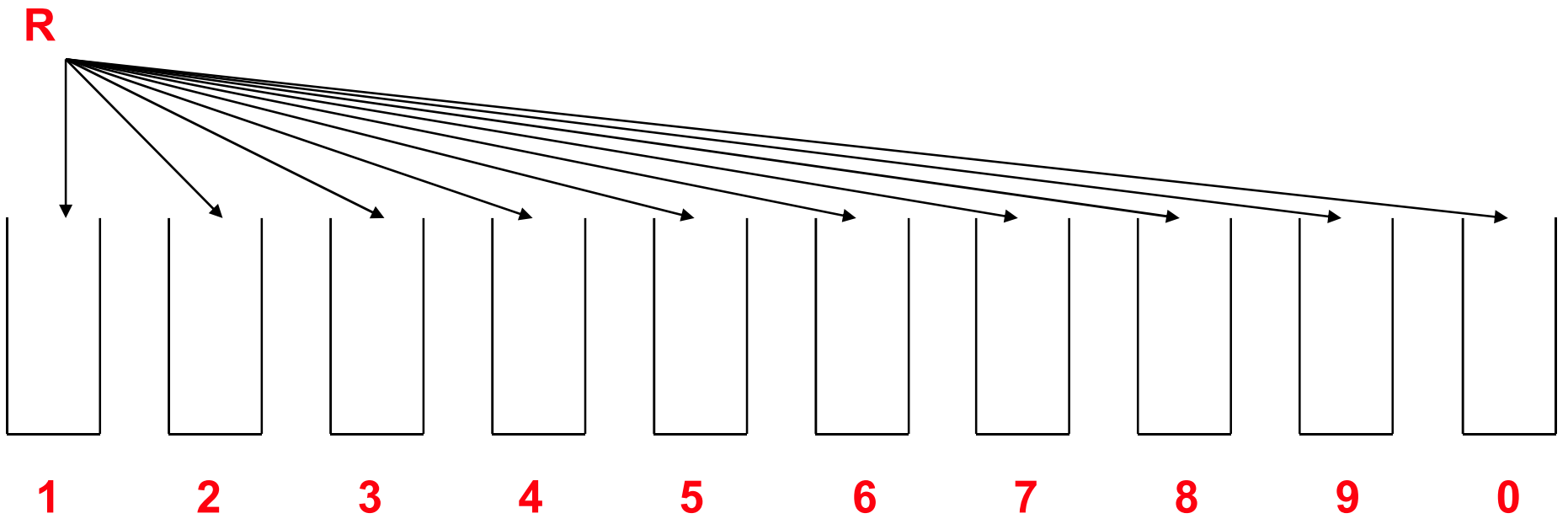
The records of files R and S are hashed using the *same hashing function* on the join attributes A of R and B of S.

Step 1. A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets.

Step 2. 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.

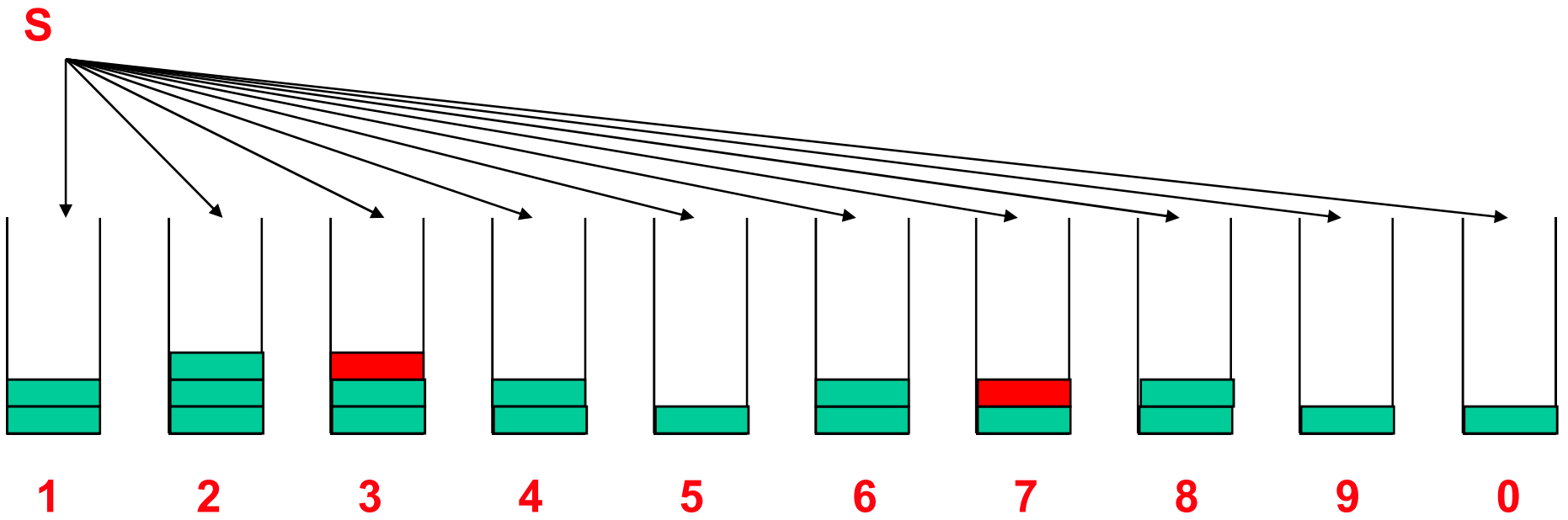
Hash Join Example

- Hash records of R into the buckets



Hash Join Example

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



Comparison between Two Ways

- Hash R first, and then S
  - How many comparisons?
  - How many blocks to read?
- Hash S first, and then R
  - How many comparisons?
  - How may blocks to read?

Implementing the Join Operation

WHAT ABOUT the MULTI-WAY JOIN?

$$U = (R *_{A=B} S *_{C=D} T)$$

[*hint*: U = (R*S)*T = R*(S*T) = (R*S)*(S*T)]

Implementing Aggregate Operation

The aggregate operators: min, max, sum, count and avg

Methods:

(a) Table Scan

(b) Index

For example, consider the following SQL query:

```
SELECT MAX(SALARY)
FROM EMPLOYEE;
```

If an index (say, B^+ tree) on SALARY exists for the employee relation, then the optimizer could decide on traversing the index for the largest value, which follows the right most pointer in each index node from the root to a leaf.

- In most cases, this would be more efficient than a full table scan of the EMPLOYEE relation.
- How about MIN?

Implementing Aggregate Operation

SUM, COUNT and AVG

Methods:

(a) for a dense index (each record has one index entry) -- apply the computation to the values in index.

(b) for a non-dense index: actual number of records associated with each index entry are considered.

GROUP BY aggregates: this operator is applied to subsets of a table.
Employee relation is hashed or sorted to partition the file.

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

Index on Salary : Cannot give any benefit.

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

Heuristic Optimization

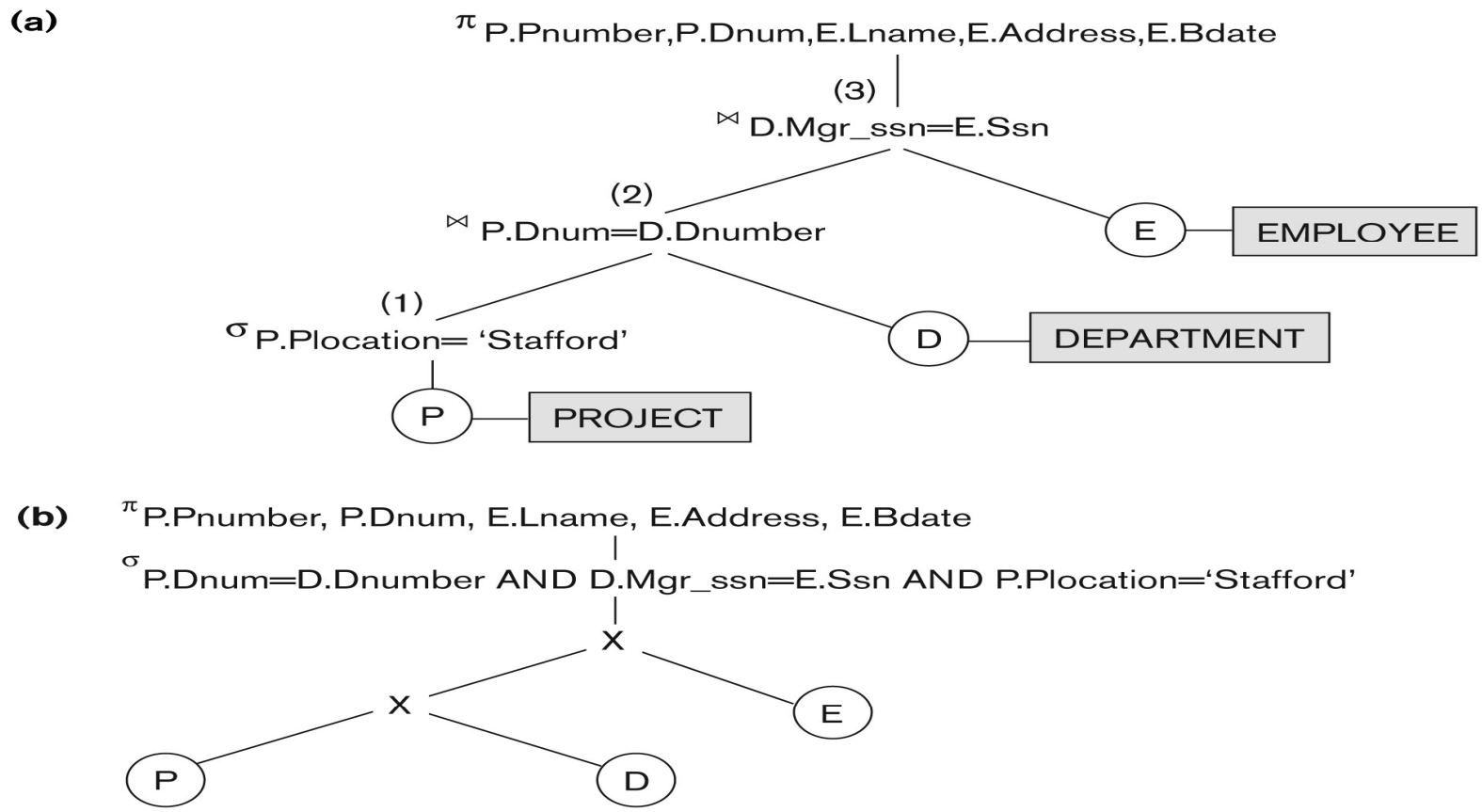


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Heuristic Optimization

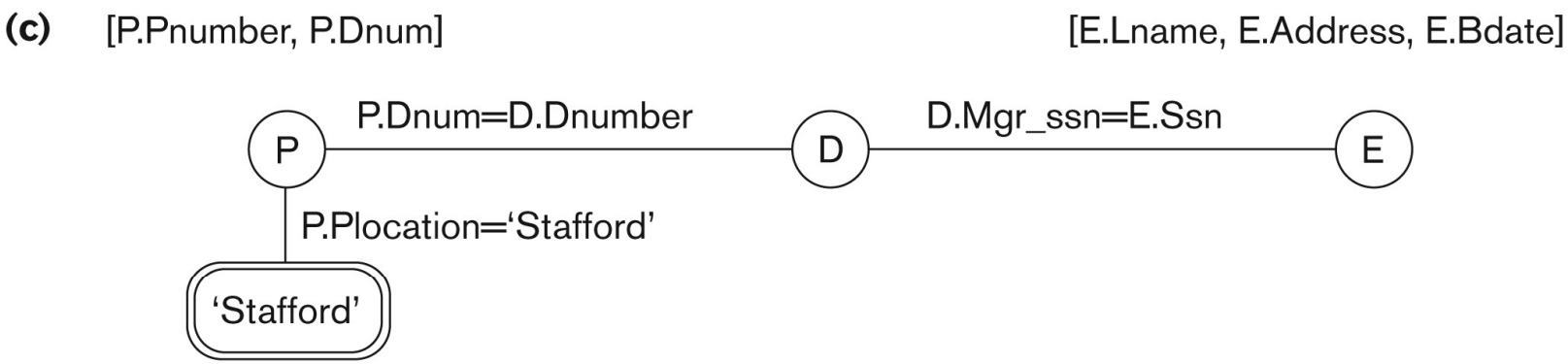


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

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 30000?

Heuristic Optimization

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:

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

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

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

Heuristic Optimization

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

$$\pi \text{ List1 } (\pi \text{ List2 } (...(\pi \text{ Listn}(R))...)) = \pi \text{ List1}(R)$$

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:

$$\pi \text{ A1, A2, ..., An } (\sigma \text{ c } (R)) = \sigma \text{ c } (\pi \text{ A1, A2, ..., An } (R))$$

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

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:

   $$\sigma_c (R * S) = (\sigma_c (R)) * S$$

   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:

   $$\sigma_c (R * S) = (\sigma_{c1} (R)) * (\sigma_{c2} (S))$$

   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

8. **Commuting π with $\bowtie_c$ (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. 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:

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

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

For X, there is no condition c, so the first transformation rule always applies by replacing $\bowtie_c$ with X.

Heuristic Optimization

9. Associativity of *, X, U, and ∩: 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)

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

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)

We discuss next how these rules are used in heuristic optimization.

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. 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.

Outline of a Heuristic Algebraic Optimization Algorithm

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. Combine a Cartesian product operation with a subsequent select operation whose condition represents a join condition into a join operation.

Outline of a Heuristic Algebraic Optimization Algorithm

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.

Example

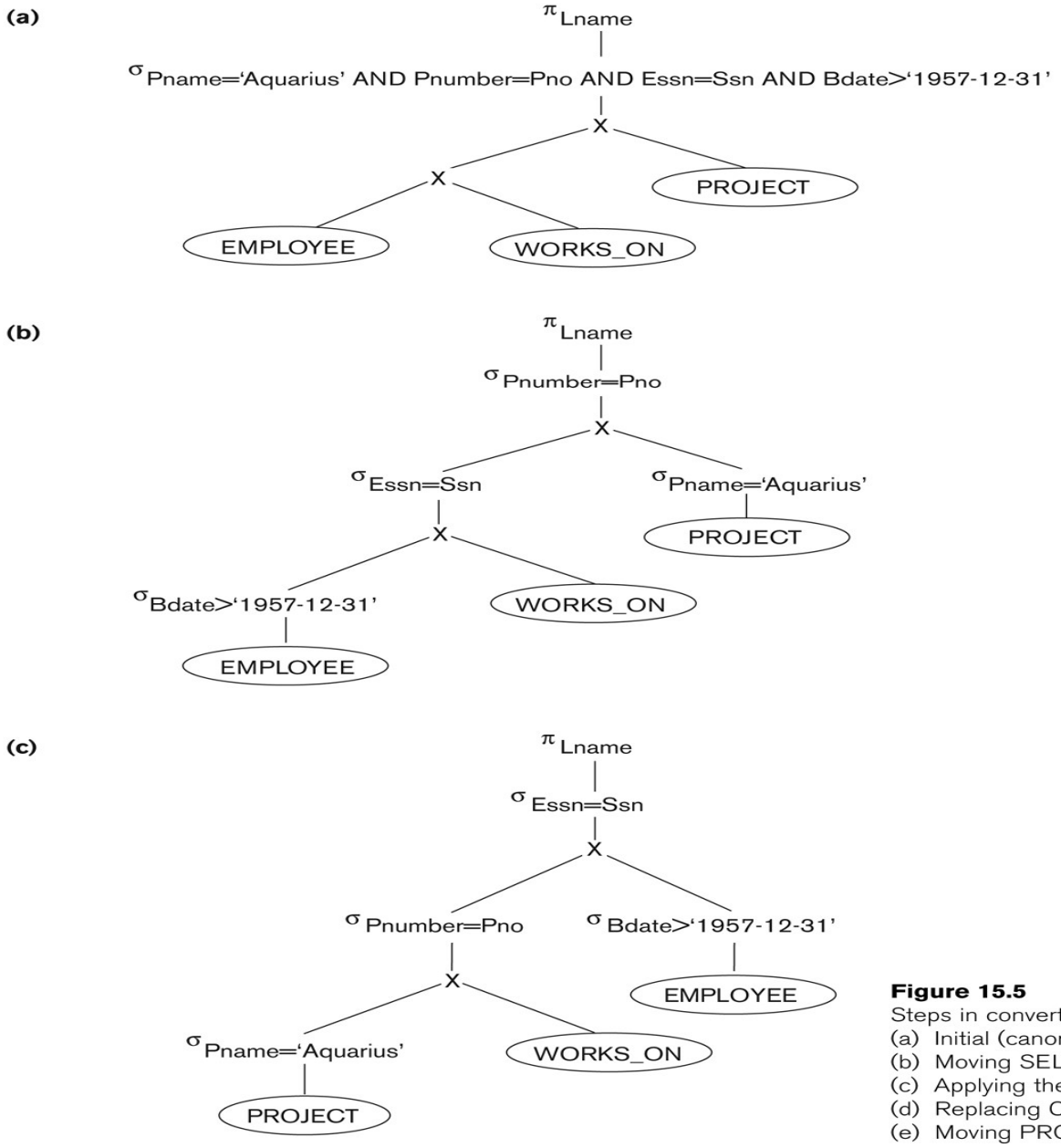
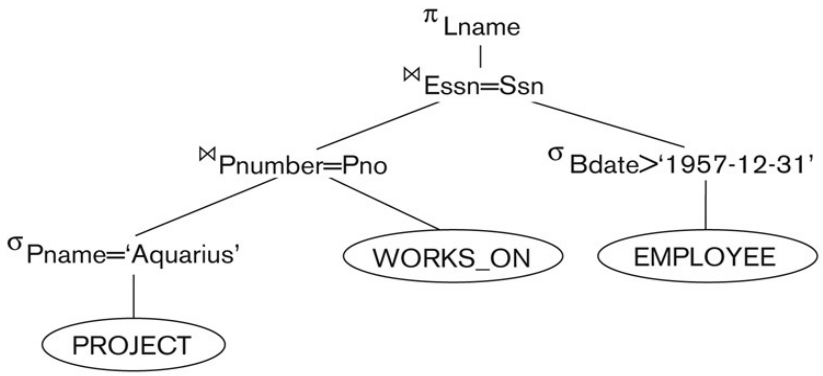


Figure 15.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

Example

(d)



(e)

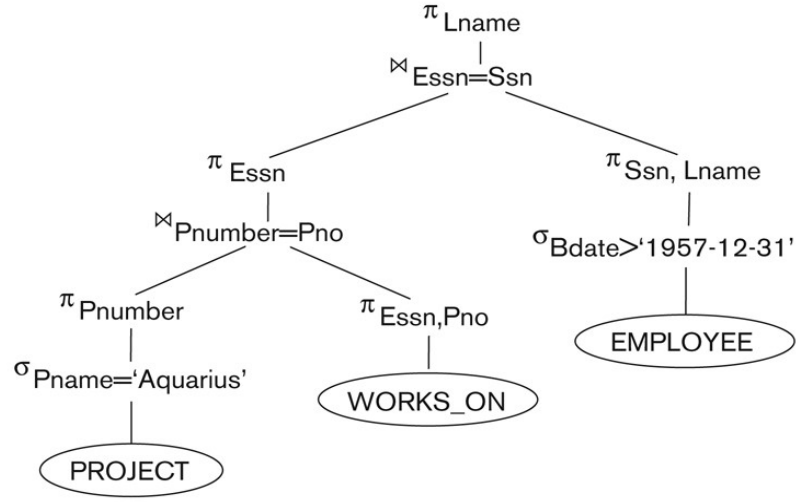


Figure 15.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

Example

- Figure 15.5(b) shows the tree of Figure 15.5(a) after applying steps 1 and 2 of the algorithm;
- Figure 15.5(c) shows the tree after applying step 3;
- Figure 15.5(d) after applying step 4;
- and Figure 15.5(e) after applying step 5.
- In step 6 we may group together the operations in the subtree whose root is the operation PNUMBER=PNO into a single algorithm.
- We may also group the remaining operations into another subtree, where the tuples resulting from the first algorithm replace the subtree whose root is the operation PNUMBER=PNO, because the first grouping means that this subtree is executed first.

Summary of Heuristic Optimization

- The main heuristic is to first apply the operations that reduce the size of intermediate results.
  - This includes performing select operations as early as possible to reduce the number of tuples, and
  - performing 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.
- In addition, the select and join operations that are most restrictive—that is, result in relations with the fewest tuples or with the smallest absolute size—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.

Learning Objectives

1. Different ways of implementing “Select”. Able to select suitable ways when certain conditions are given.
2. Different ways of implementing “Join”. Able to select suitable ways when certain conditions are given; Able to compare different methods (which one is better) given certain condition.
3. Be familiar with the ways to implement aggregate functions utilizing index structure.
4. Understand how to improve the execution of queries, and the reason to use those 12 heuristic rules