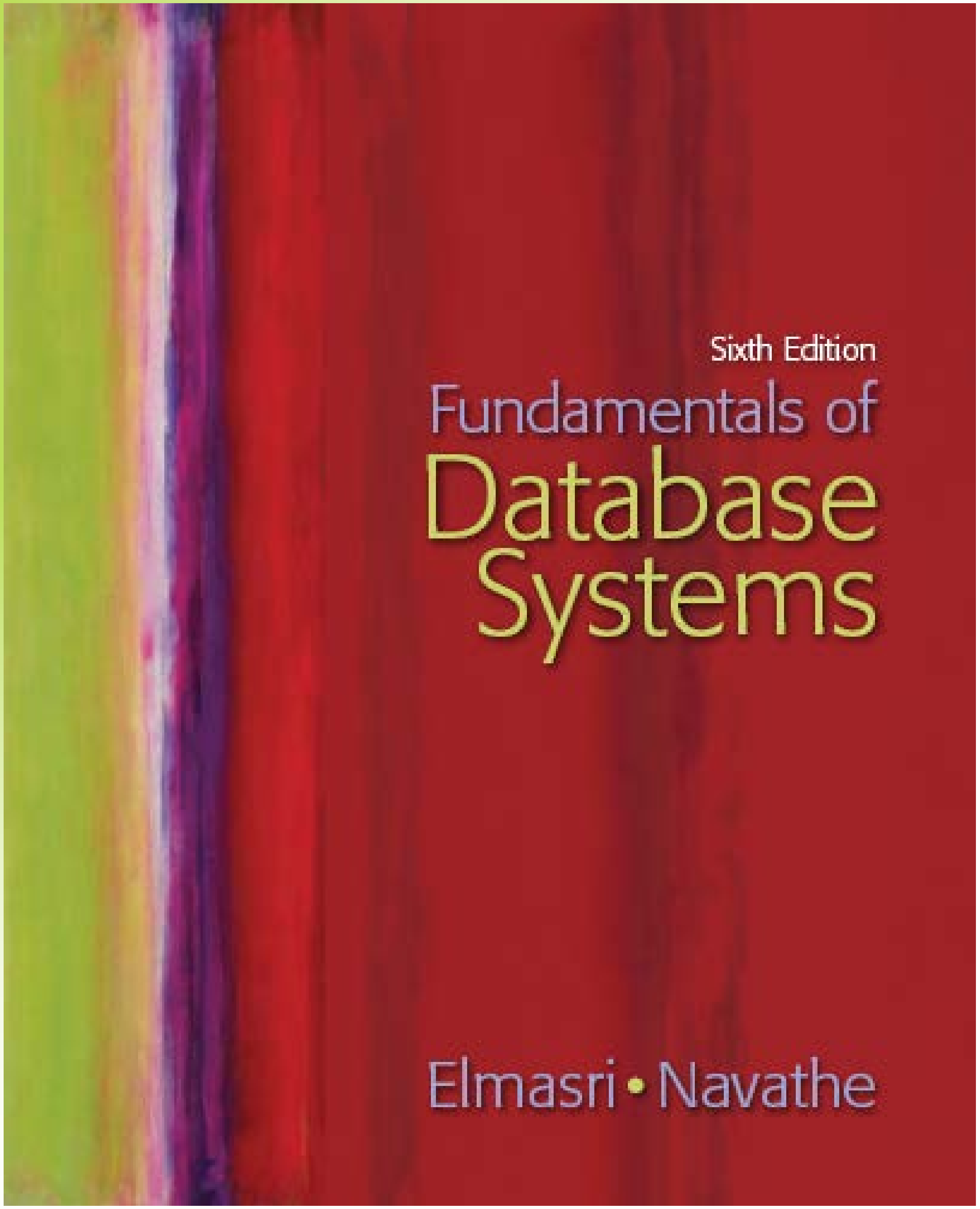
Chapter 19

Algorithms for Query Processing and Optimization



Addison-Wesley is an imprint of PEARSON

0. Introduction to Query Processing (1)

- **Query processing:**
  - Process of executing a query in a DBMS
- **Query optimization**:
  - Process of choosing a suitable execution strategy for processing a query

Introduction to Query Processing (2)

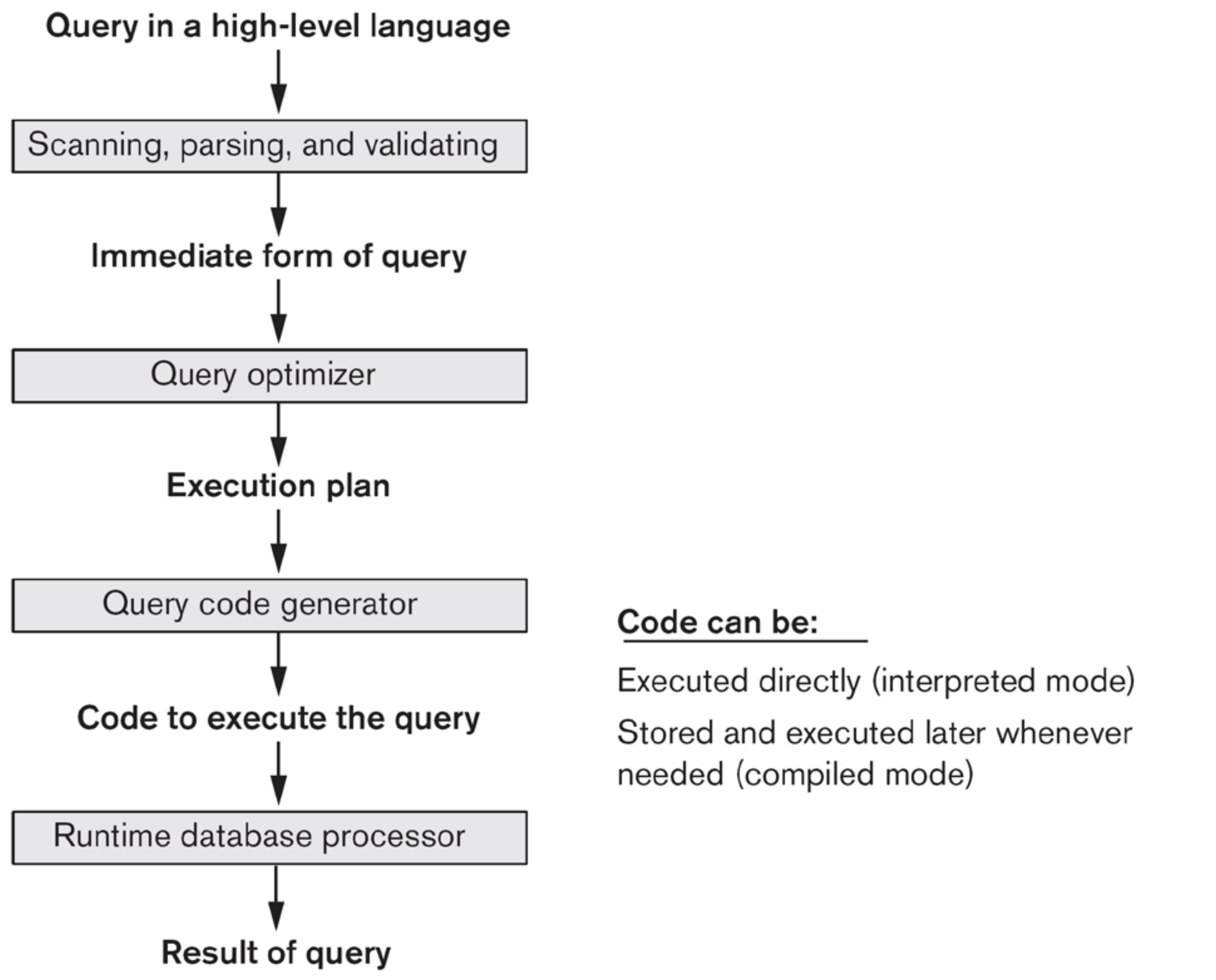

Query in a high-level language
Scanning, parsing, and validating
Immediate form of query
Query optimizer
Execution plan
Query code generator
Code to execute the query
Runtime database processor
Result of query
Code can be:
Executed directly (interpreted mode)
Stored and executed later whenever needed (compiled mode)

Figure 19.1
Typical steps when processing a high-level query.

1. Translating SQL Queries into Relational Algebra (1)

- **Query block**:
  - A basic unit that can be translated into the algebraic operators and optimized
  - Contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause
- **Nested queries** within a query
  - Identified as separate query blocks

Translating SQL Queries into Relational Algebra (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
```

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

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

2. Algorithms for External Sorting

- **External sorting**:
  - Suitable for large files of records stored on disk
  - Because most database files do not fit entirely in main memory
- **Sort-Merge strategy**:
  - Starts by sorting small subfiles (**runs**) of the main file
  - Then, merges the sorted runs, creating larger sorted subfiles that are merged in turn

3. Algorithms for SELECT and JOIN Operations (1)

- Implementing the SELECT Operation
- Examples:
  - (OP1): $\sigma_{\text{SSN='123456789'}}$ (EMPLOYEE)
  - (OP2): $\sigma_{\text{DNUMBER>5}}$(DEPARTMENT)
  - (OP3): $\sigma_{\text{DNO=5}}$(EMPLOYEE)
  - (OP4): $\sigma_{\text{DNO=5 AND SALARY>30000 AND SEX=F}}$(EMPLOYEE)
  - (OP5): $\sigma_{\text{ESSN=123456789 AND PNO=10}}$(WORKS_ON)

Algorithms for SELECT and JOIN *Operations* (2)

- Implementing the SELECT Operation (contd.):
- Search Methods for Simple Selection:
  - S1 **Linear search** (brute force):
    - Retrieve every record in the file, and test whether its attribute values satisfy the selection condition.
  - S2 **Binary search**:
    - If the selection condition involves an equality comparison on a key attribute on which *the file is ordered*, binary search (which is more efficient than linear search) can be used. (See OP1).
  - S3 **Using a primary index or hash key to retrieve a single record**:
    - If the selection condition involves an equality comparison on a key attribute *with a primary index (or a hash key),* use the primary index (or the hash key) to retrieve the record.

Algorithms for SELECT and JOIN Operations (3)

- Implementing the SELECT Operation (contd.):
- Search Methods for Simple Selection:
  - S4 **Using a primary index to retrieve multiple records**:
    - If the comparison condition is >, ≥, <, or ≤ on a key field with a primary index, use the index to *find the record satisfying the first corresponding equality condition*, then retrieve all subsequent records in the (ordered) file.
  - S5 **Conjunctive selection**:
    - If 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 S4, *use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions* in the conjunctive condition.

Algorithms for SELECT and JOIN Operations (4)

- Implementing the SELECT Operation (contd.):
- Search Methods for Complex Selection:
  - S6 **Conjunctive selection by intersection of record pointers**:
    - This method is possible if secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the conjunctive condition and if the indexes include record pointers (rather than block pointers).
    - 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.

Algorithms for SELECT and JOIN Operations (5)

- Implementing the JOIN Operation:
  - Join (EQUIJOIN, NATURAL JOIN)
    - two–way join: a join on two files
    - e.g. $R \bowtie_{A=B} S$
    - multi-way joins: joins involving more than two files.
    - e.g. $R \bowtie_{A=B} S \bowtie_{C=D} T$
- Examples
  - (OP6): EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT
  - (OP7): DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

Algorithms for SELECT and JOIN Operations (6)

- Implementing the JOIN Operation (contd.):
- Methods for implementing joins:
  - J1 **Nested-loop join** (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 **Single-loop join** (Using an access structure to retrieve the matching records):
    - If an index (or hash key) 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 retrieve directly all matching records s from S that satisfy s[B] = t[A].

Algorithms for SELECT and JOIN Operations (7)

- Implementing the JOIN Operation (contd.):
- Methods for implementing joins:
  - J3 **Sort-merge join**:
    - If the records of R and S are *physically sorted* (*ordered*) by value of the join attributes A and B, respectively, we can implement the join in the most efficient way possible.
    - 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, the records of each file are scanned only once each for matching with the other file—unless both A and B are non-key attributes, in which case the method needs to be modified slightly.

Algorithms for SELECT and JOIN Operations (8)

- Implementing the JOIN Operation (contd.):
- Methods for implementing joins:
  - J4 **Hash-join**:
    - The records of files R and S are both hashed to the *same hash file*, using the *same hashing function* on the join attributes A of R and B of S as hash keys.
    - A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets.
    - 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.

4. Algorithms for PROJECT and SET Operations (1)

- Algorithm for PROJECT operations (Figure 15.3b)

  $\pi_{\text{<attribute list>}}(R)$

  1. If <attribute list> has a key of relation R, extract all tuples from R with only the values for the attributes in <attribute list>.
  2. If <attribute list> does NOT include a key of relation R, *duplicated tuples must be removed from the results.*

- Methods to remove duplicate tuples
  1. Sorting
  2. Hashing

Algorithms for PROJECT and SET Operations (2)

- **Algorithm for SET operations**
- **Set operations**:
  - UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN PRODUCT
- **CARTESIAN PRODUCT** of relations R and S include all possible combinations of records from R and S. The attribute of the result include all attributes of R and S.
- **Cost analysis** of CARTESIAN PRODUCT
  - If R has n records and j attributes and S has m records and k attributes, the result relation will have n*m records and j+k attributes.
- CARTESIAN PRODUCT operation is **very expensive** and should be avoided if possible.

Algorithms for PROJECT and SET Operations (3)

- **Algorithm for SET operations (contd.)**
- **UNION** (See Figure 19.3c)
  - Sort the two relations on the same attributes.
  - Scan and merge both sorted files concurrently, whenever the same tuple exists in both relations, only one is kept in the merged results.
- **INTERSECTION** (See Figure 19.3d)
  - Sort the two relations on the same attributes.
  - Scan and merge both sorted files concurrently, keep in the merged results only those tuples that appear in both relations.
- **SET DIFFERENCE R-S** (See Figure 19.3e)
  - Keep in the merged results only those tuples that appear in relation R but not in relation S.

5. Implementing Aggregate Operations

- Implementing Aggregate Operations:
- **Aggregate operators**:
  - **MIN, MAX, SUM, COUNT** and **AVG**
- Options to implement aggregate operators:
  - **Table Scan**
  - **Index**
- Example
  - SELECT MAX (SALARY)
  - FROM EMPLOYEE;
- If an (ascending) index on SALARY exists for the employee relation, then the optimizer could decide on traversing the index for the largest value, which would entail following the right most pointer in each index node from the root to a leaf.

Implementing Aggregate Operations

- Implementing Aggregate Operations (contd.):
- **SUM, COUNT and AVG**
- For a **dense index** (each record has one index entry):
  - Apply the associated computation to the values in the index.
- For other cases:
  - Actual number of records associated with each index entry must be accounted for

7. Using Heuristics in Query Optimization (1)

- 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

- 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.

Using Heuristics in Query Optimization (2)

- **Query tree**:
  - A tree 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.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

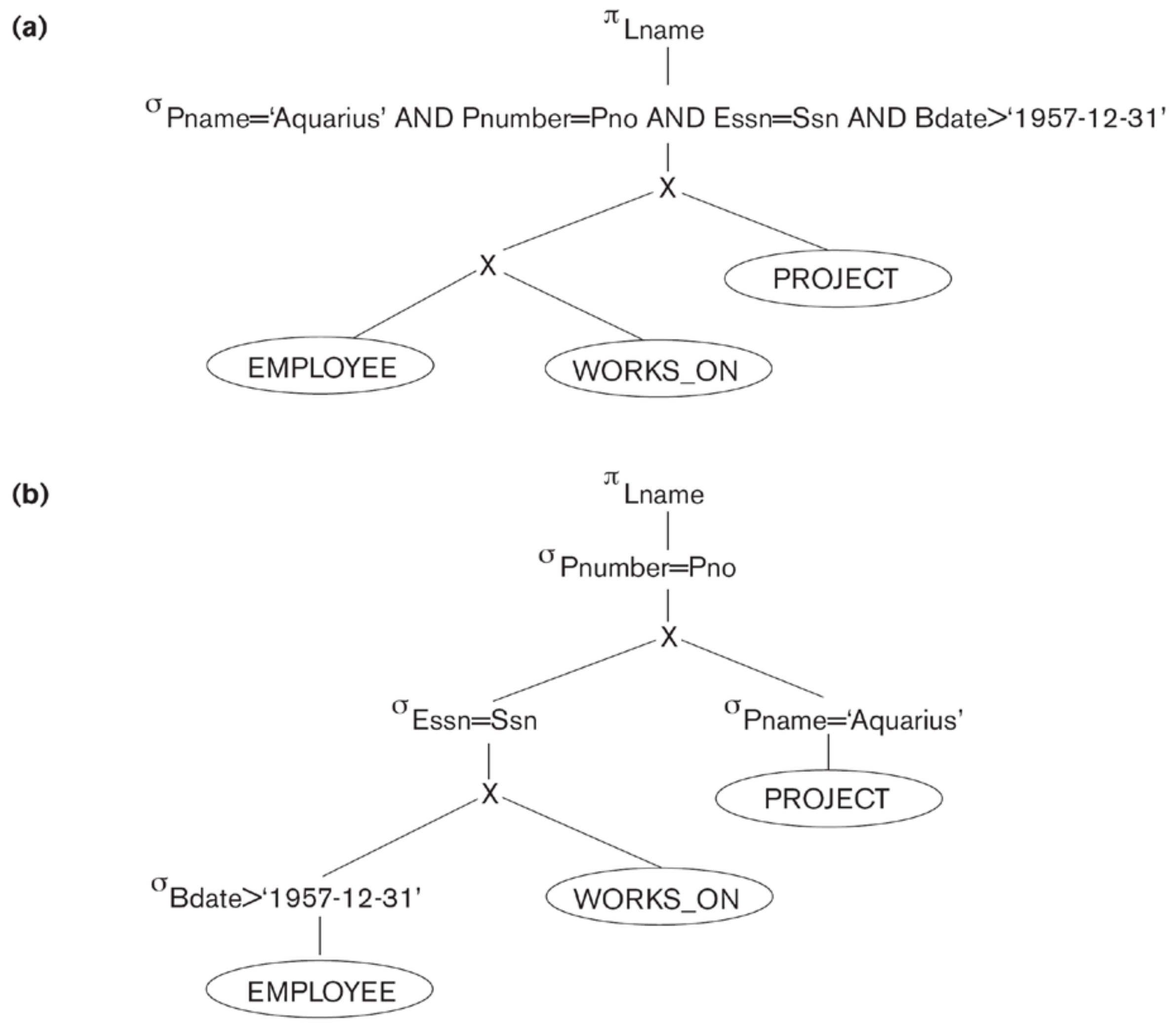
Using Heuristics in Query Optimization (3)

- 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
                PNMUBER=PNO AND ESSN=SSN
                AND BDATE > ‘1957-12-31’;
```

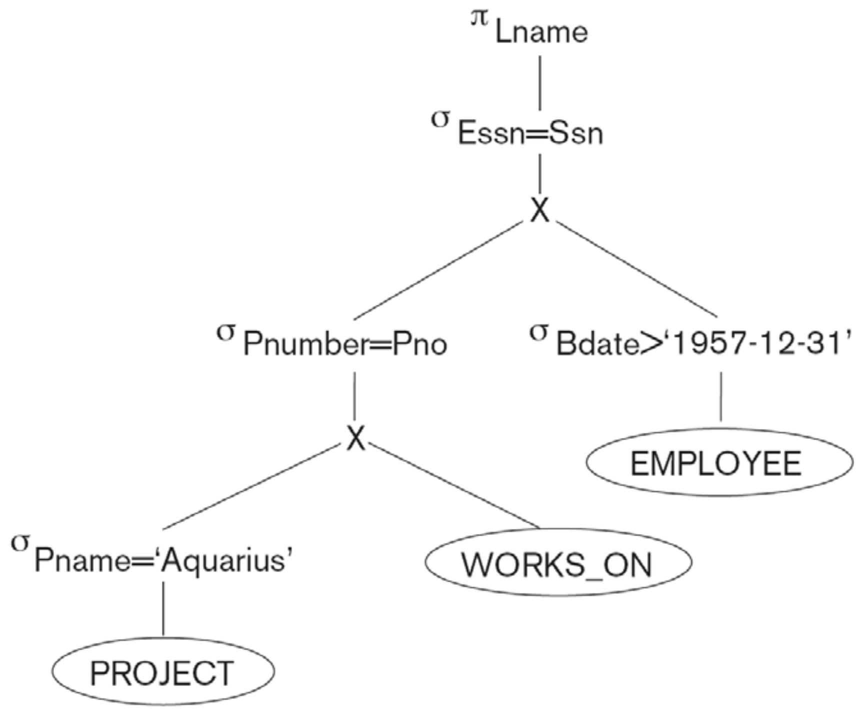
Using Heuristics in Query Optimization

Figure 19.5
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.

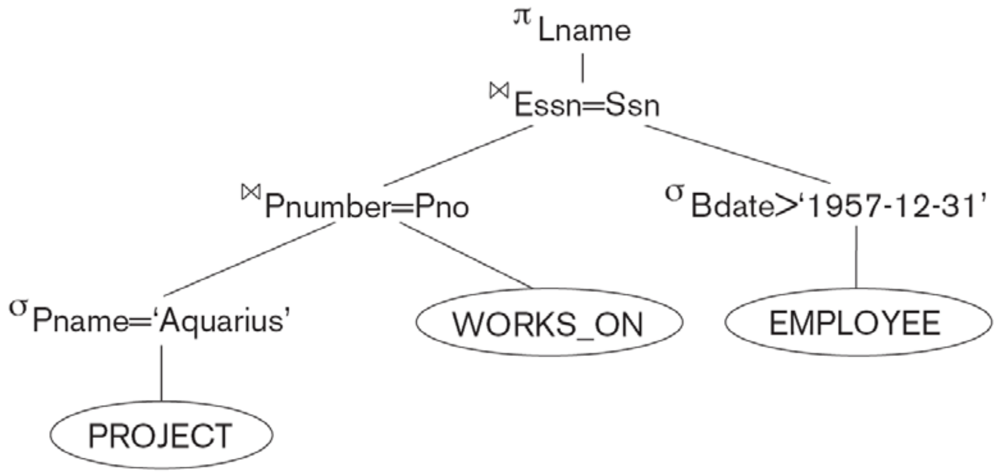


Using Heuristics in Query Optimization

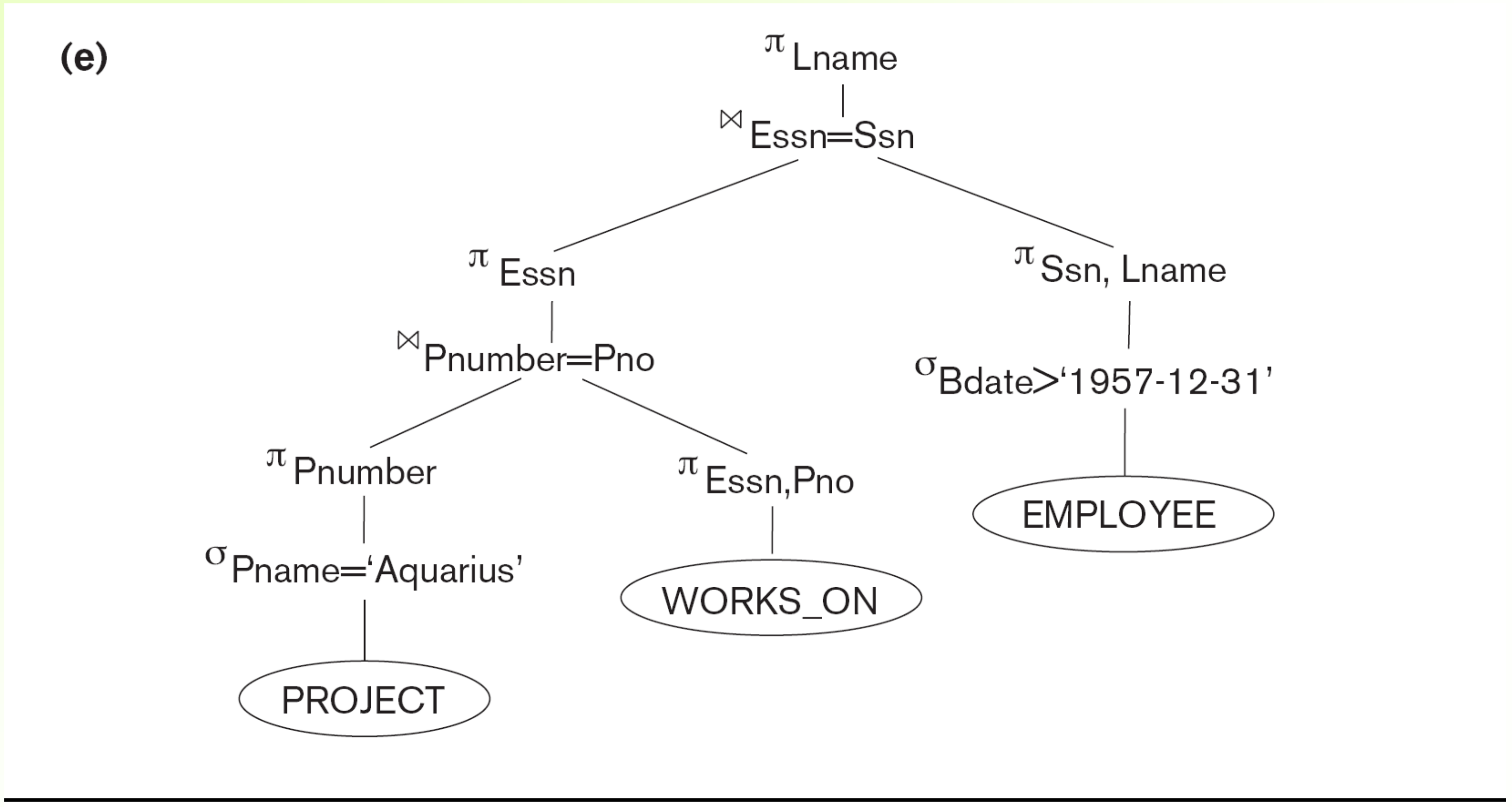
(c)



(d)



Using Heuristics in Query Optimization



8. Using Selectivity and Cost Estimates in Query Optimization (1)

- **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)
- Issues
  - Cost function
  - Number of execution strategies to be considered

Using Selectivity and Cost Estimates in Query Optimization (2)

- 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.

Summary

0. Introduction to Query Processing
1. Translating SQL Queries into Relational Algebra
2. Algorithms for External Sorting
3. Algorithms for SELECT and JOIN Operations
4. Algorithms for PROJECT and SET Operations
5. Implementing Aggregate Operations and Outer Joins
6. Using Heuristics in Query Optimization
7. Using Selectivity and Cost Estimates in Query Optimization