Query Processing and Optimization

Aims:

At the end of this group of two lectures you should be able to understand the steps of query processing and techniques for query optimization.

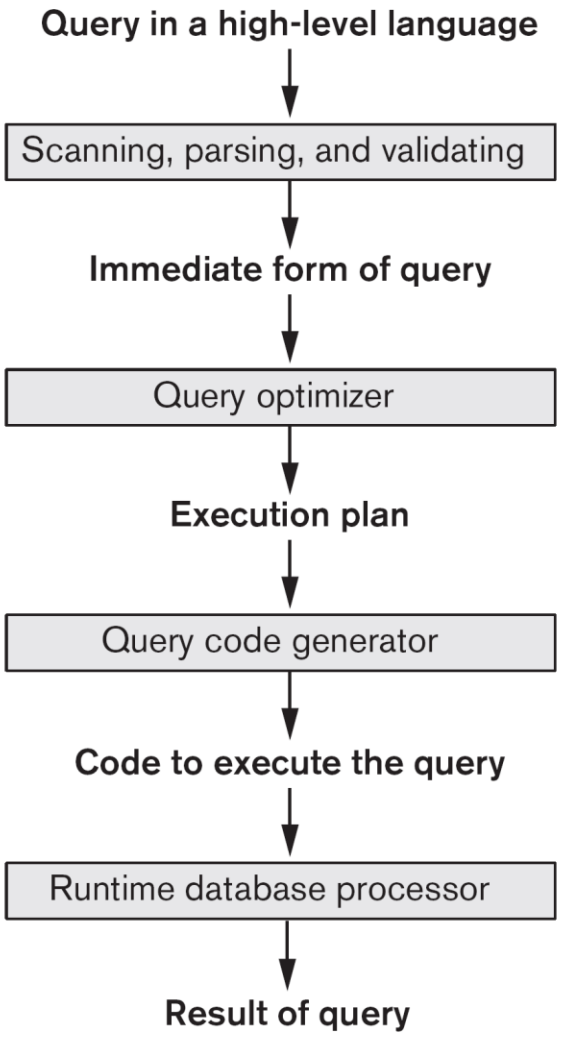
Reading: Elmasri & Navathe, Chapter 19 (6th ed.)

or Chapters 18 & 19 (7th ed.)

Overview

- Query processing
- Translating SQL queries into relational algebra
- Algorithms for executing query operations
- Query optimization
  - Heuristic
  - Cost-based
  - Semantic
- Query optimization in Oracle

Query Processing



Code can be:

Executed directly (interpreted mode)

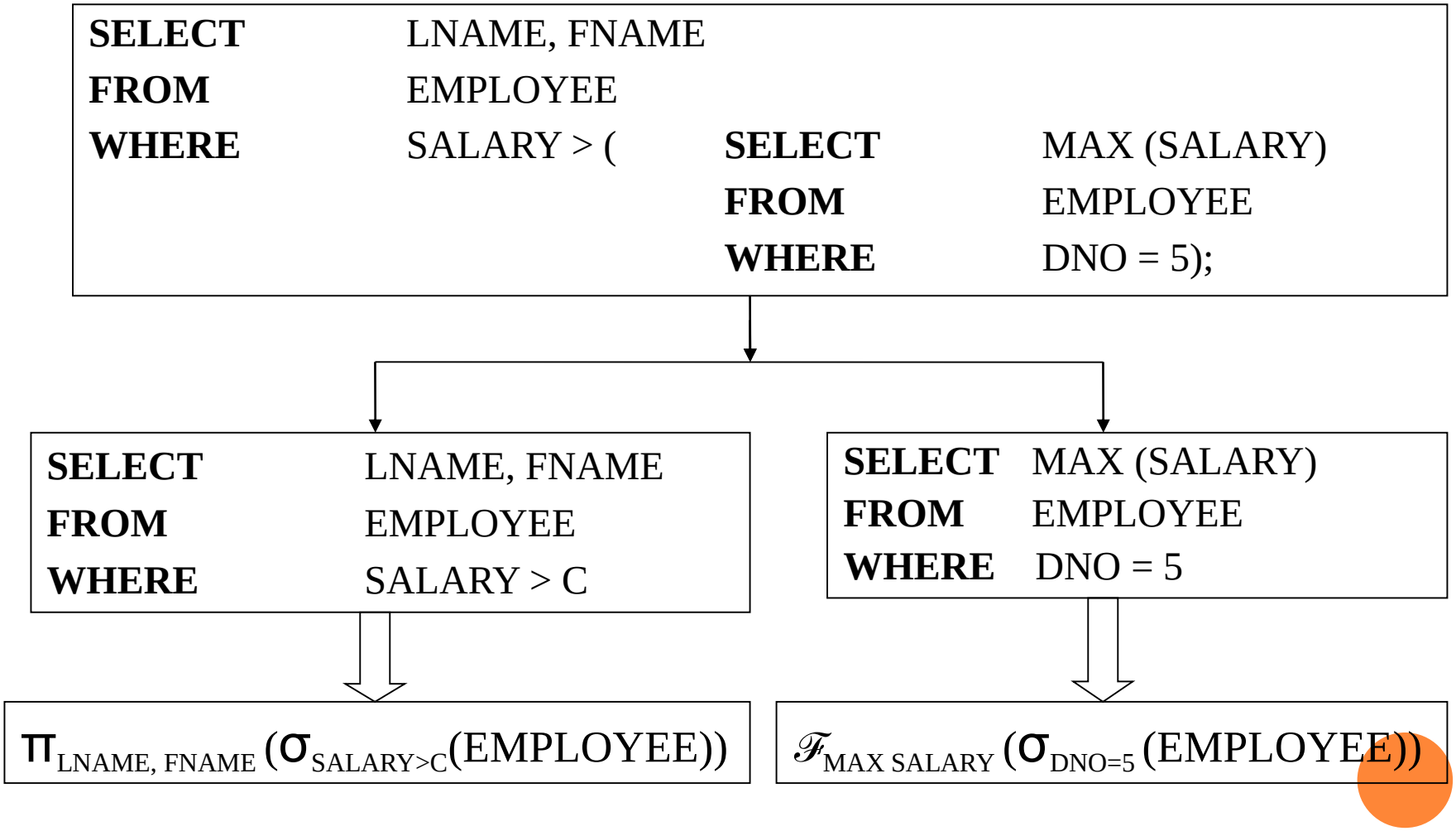
Stored and executed later whenever needed (compiled mode)

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

Translating SQL Queries into Relational Algebra

- **Query block**:
  - The basic unit that can be translated into the algebraic operators and optimized.
- A single SELECT-FROM-WHERE expression (potentially including GROUP BY and HAVING)
- **Nested queries** within a query are identified as separate query blocks

TRANSLATING SQL INTO RELATIONAL ALGEBRA



External Sorting

- Suitable for large files that do not fit entirely in main memory
- **Sort-Merge strategy**:
  - Starts by sorting small subfiles (**runs**) of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.
  - Sorting phase: $n_R = \lceil (b/n_B) \rceil$
  - Merging phase: $d_M = \text{Min}\ (n_B\text{-}1, n_R)$; $n_P = \lceil (\log_{dM}(n_R)) \rceil$
  - n_R: number of initial runs
  - b: number of file blocks
  - n_B: available buffer space
  - d_M: degree of merging
  - n_P: number of passes

ALGORITHM FOR EXTERNAL SORTING

```
set     i ← 1;
        j ← b;          {size of the file in blocks}
        k ← n_B;        {size of buffer in blocks}
        m ← ⌈(j/k)⌉;

{Sorting Phase}
while (i ≤ m)
do {
        read next k blocks of the file into the buffer or if there are less than k blocks
            remaining, then read in the remaining blocks;
        sort the records in the buffer and write as a temporary subfile;
        i ← i + 1;
}

{Merging Phase: merge subfiles until only 1 remains}
set     i ← 1;
        p ← ⌈log_{k−1} m⌉   {p is the number of passes for the merging phase}
        j ← m;
while (i ≤ p)
do {
        n ← 1;
        q ← ⌈j/(k−1)⌉;  {number of subfiles to write in this pass}
        while (n ≤ q)
        do {
            read next k−1 subfiles or remaining subfiles (from previous pass)
                one block at a time;
            merge and write as new subfile one block at a time;
            n ← n + 1;
        }
        j ← q;
        i ← i + 1;
}
```

Figure 19.2
Outline of the sort-merge algorithm for external sorting.

Algorithms for SELECT

- Simple selection
- Complex selection
- 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): $\sigma_{ESSN=123456789\ AND\ PNO=10}(WORKS\_ON)$
- Selectivity of a search condition – the ratio of the number of tuples that satisfy the condition to the total number of tuples in the relation (between 0 and 1)

ALGORITHMS FOR SIMPLE SELECTION

- File/index scans

- Linear search (brute force): no index, unstructured data
- Binary search: no index, sorted data
- Using a primary index or hash key to retrieve a single record
- Using a primary index to retrieve multiple records (>, ≥, <, ≤)
- Using a clustering index to retrieve multiple records (=)
- Using a secondary index

ALGORITHMS FOR COMPLEX SELECTION

- Conjunctive selection using an index
  - If possible, use an index for a simple condition
  - Check whether each retrieved tuple satisfies other conditions
- Conjunctive selection using a composite index
  - A compound index exists for two or more attributes involved in conditions
  - Example: A composite index exists on (Star, Movie)

    *select * from stars where star = 20 and movie = 10;*
- Conjunctive selection using intersection of record pointers
  - Secondary indexes available for (some) attributes, providing pointers to records (not blocks)
  - Retrieve record pointers
  - Find intersection
  - Check other conditions
- Disjunctive conditions (union)
  - If there are no indexes, brute force

ALGORITHMS FOR JOIN

- Time-consuming
- EQUIJOIN, NATURAL JOIN
- $R \bowtie_{A=B} S$
- Examples
  - (OP6): EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT
  - (OP7): DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE
- Factors affecting JOIN performance
  - Available buffer space
  - Join selection factor: the fraction of records in one file that will be joined with records in the other file
  - Choice of inner vs outer relation

Algorithms for JOIN

- **Nested-loop join** (brute force)

  for each tuple r from R do

  for each tuple s from S do

  if r.A = s.B then add <r,s> to result

- **Single-loop join**

  Using an access structure to retrieve the matching record(s)

  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 JOIN

- **Sort-merge join**
  - If R and S are *physically sorted* by the join attributes A and B
  - Scan both files in order of the join attributes and match
  - most efficient
- **Hash-join**
  - R and S are both hashed to the *same hash file*, using the *same hashing function* on the join attributes A and B as hash keys
  - Partitioning phase (with the smaller file R): hash records into buckets
  - Probing phase: single pass through S then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from R.

ALGORITHMS FOR PROJECT

- π <attribute list>(R)
- Easy if the attribute list contains the candidate key: remove unwanted attributes
- If not, duplicates must be removed

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

ALGORITHMS FOR SET OPERATIONS

- **CARTESIAN PRODUCT** is expensive:
  - 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
- **UNION**
  - 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
- **INTERSECTION**
  - 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**
  - Keep in the merged results only those tuples that appear in relation R but not in relation S

AGGREGATE FUNCTIONS

- **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

AGGREGATE FUNCTIONS

- **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 a **non-dense index**:
  - Actual number of records associated with each index entry must be accounted for
- With **GROUP BY**: the aggregate operator must be applied separately to each group of tuples
  - Use sorting or hashing on the group attributes to partition the file into the appropriate groups;
  - Computes the aggregate function for the tuples in each group.

IMPLEMENTING OUTER JOINS

- Left, Right, Full outer join
- The full outer join produces a result which is equivalent to the union of the results of the left and right outer joins
- Example:

```
SELECT    FNAME, DNAME
FROM      (EMPLOYEE LEFT OUTER JOIN DEPARTMENT
          ON DNO = DNUMBER);
```

- Nested Loop or Sort-Merge joins can be modified to implement outer join
  - For left outer join, use the left relation as outer relation and construct result from every tuple in the left relation
  - If there is a match, the concatenated tuple is saved in the result
  - However, if an outer tuple does not match, then the tuple is still included in the result but is padded with a null value(s)

QUERY OPTIMIZATION

- Minimizing the cost of a query
- Not necessarily optimal, but efficient plan
- Query optimization
  - Heuristic
  - Cost-based
  - Semantic

QUERY REPRESENTATION

- **Query tree**: a tree data structure that corresponds to a relational algebra expression
  - relations - leaf nodes
  - relational algebra operations - 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 resulting relation
- **Canonical tree:** a standard tree that corresponds to an SQL query without optimization

Example

```
SELECT lname, fname
FROM star, stars, movie
WHERE snumber=star AND movie=mnumber
        AND  title='Gone with the wind';
```

Relational algebra:

Another 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

  Select PNumber, DNUM, LName, Address, BDate

  From Project P, Department D, Employee E

  Where P.DNUM=D.DNumber and D.MGRSSN=E.SSN and P.Location=‘Stafford’;

- Relational algebra

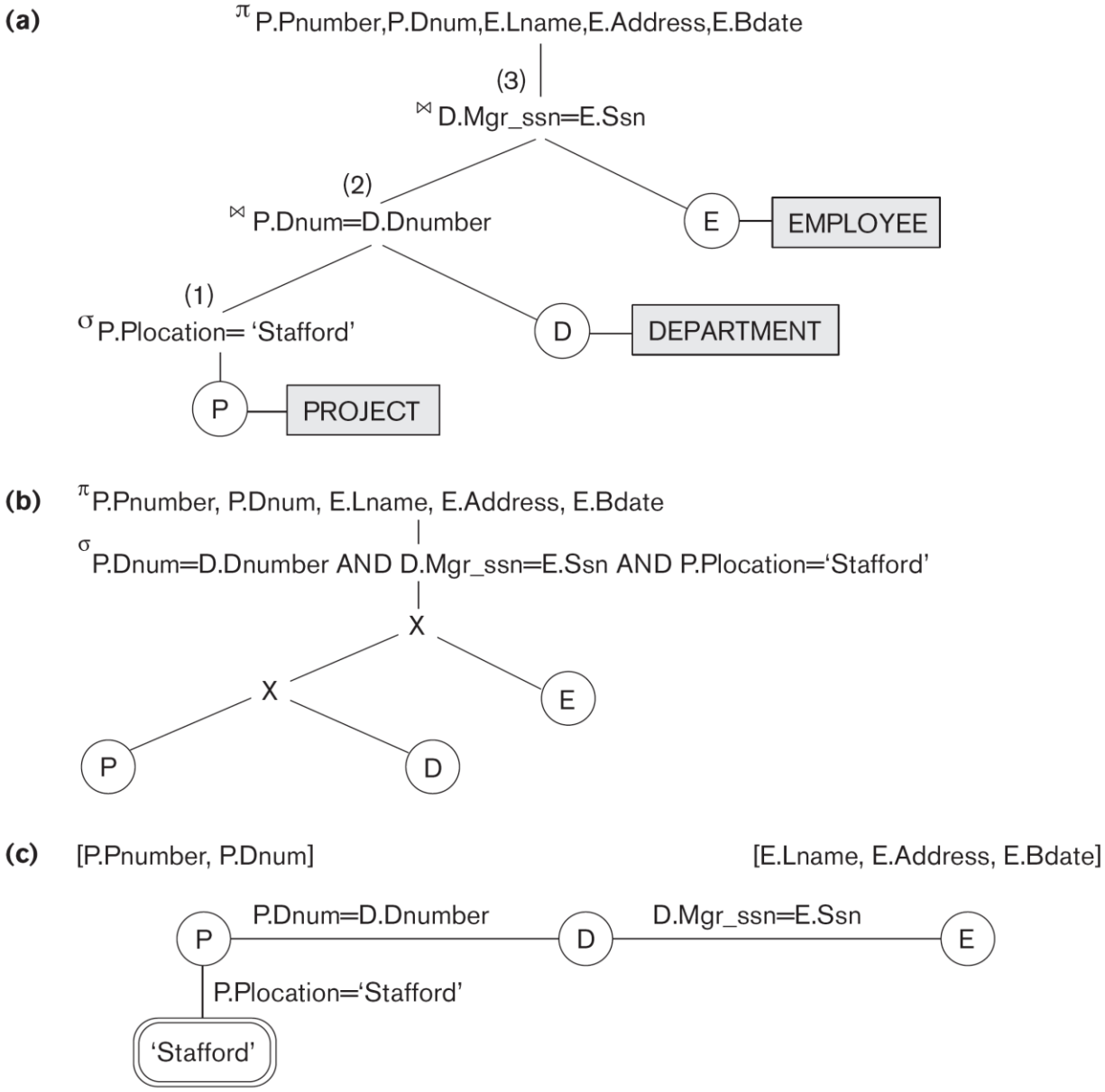


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

HEURISTIC QUERY 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 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.

HEURISTIC QUERY OPTIMIZATION

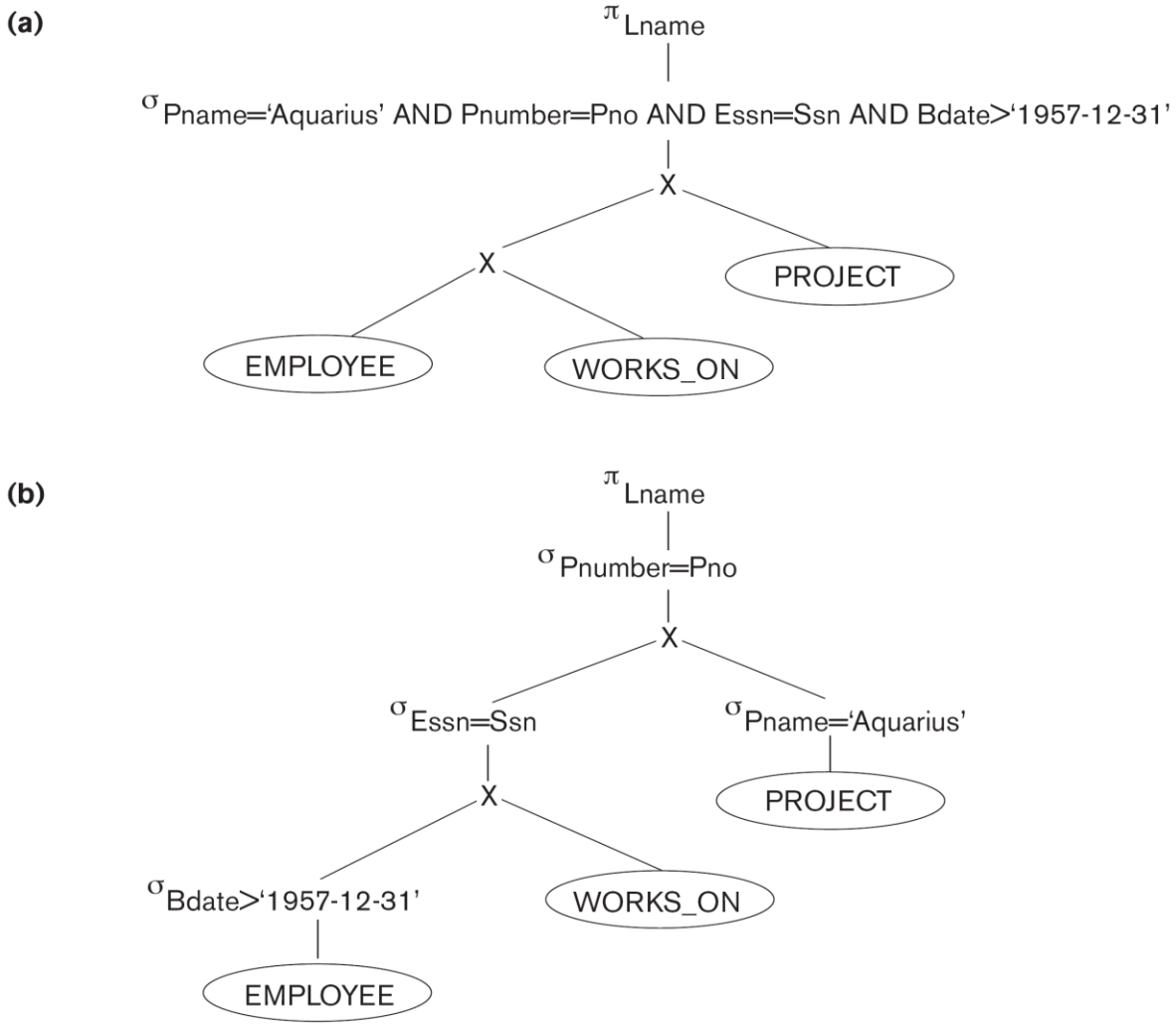
- The same query could correspond to many different relational algebra expressions — and hence many different query trees
- The task of heuristic optimization of query trees is to find a **final query tree** that is efficient to execute.
- Example:

Q: SELECT LNAME

FROM EMPLOYEE, WORKS_ON, PROJECT

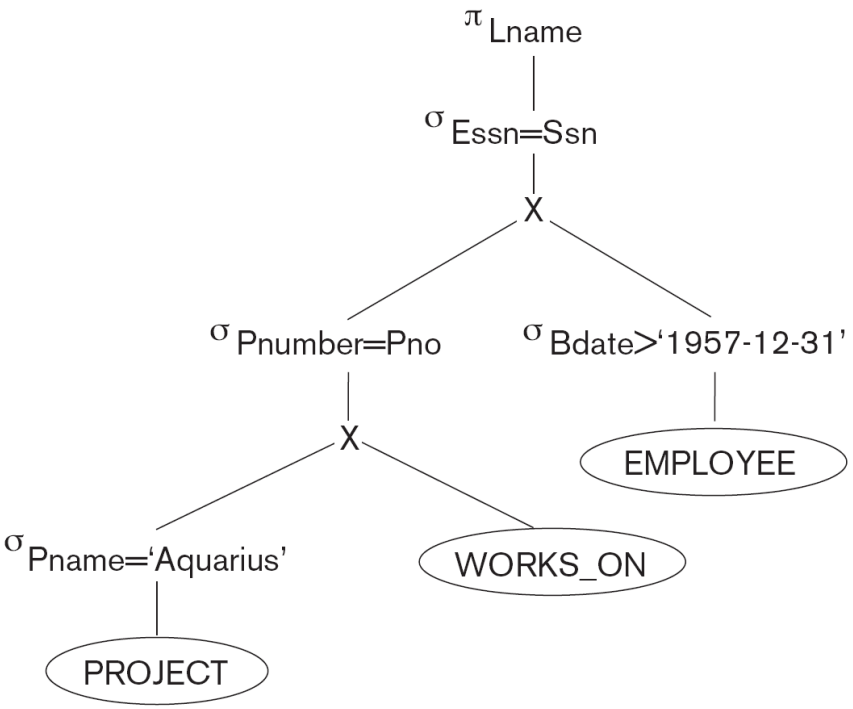
WHERE PNAME = 'Aquarius' AND
PNUMBER=PNO AND ESSN=SSN
AND BDATE > '1957-12-31';

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.



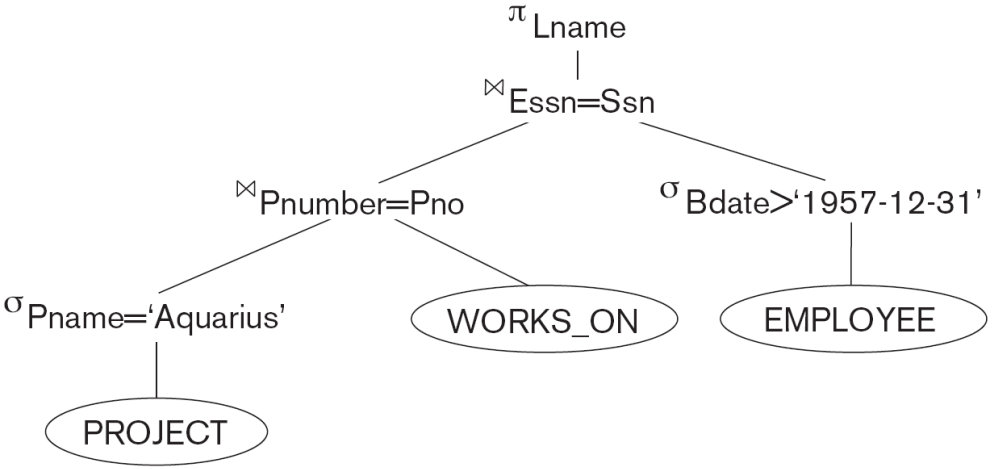
(c)

Apply more selective condition first



(d)

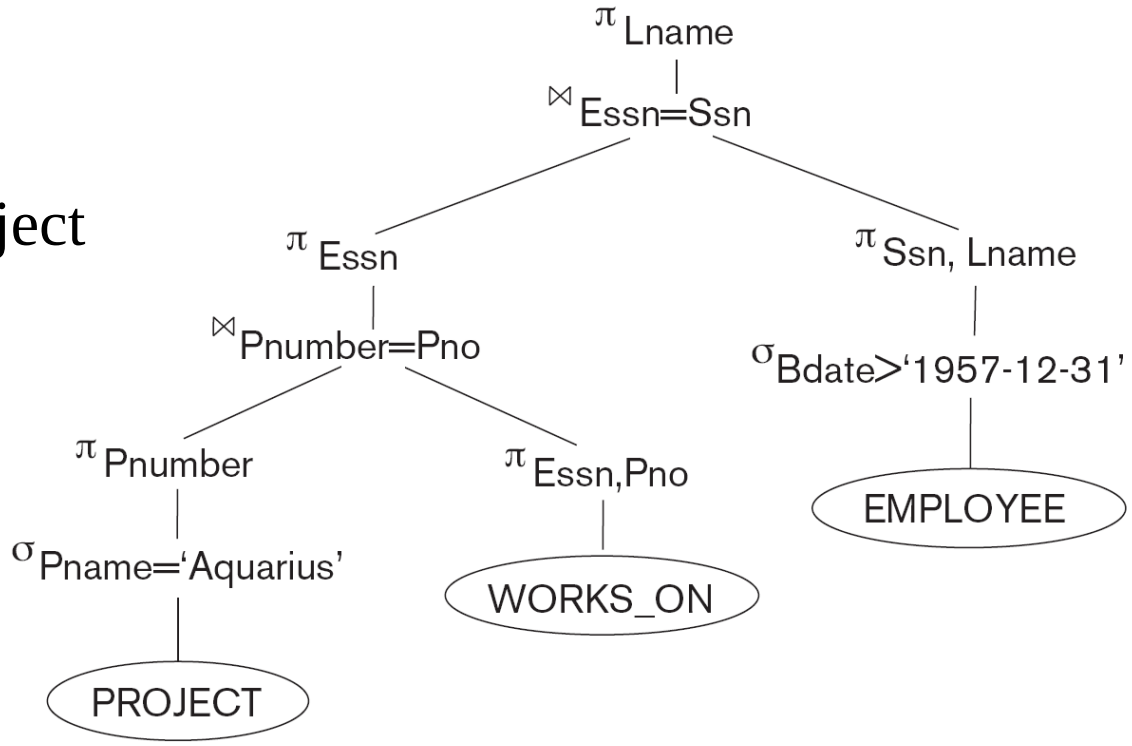
Use joins



Using Heuristics in Query Optimization

(e)

Move Project down



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:
   - $\sigma_{c1 \text{ AND } c2 \text{ AND } ... \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$
2. Commutativity of σ: The σ operation is commutative:
   - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
3. Cascade of π: In a cascade (sequence) of π operations, all but the last one can be ignored:
   - $\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) = \pi_{List1}(R)$
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:
   - $\pi_{A1, A2, ..., An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, ..., An}(R))$

Transformation Rules (cont)

5. Commutativity of join and Cartesian product

   $R \bowtie_C S = S \bowtie_C R$; $R \times S = S \times R$

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

Transformation Rules (cont)

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.

Transformation Rules (cont)

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

9. Associativity of ⋈, 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

- $\sigma_c (R \theta S) = (\sigma_c (R)) \theta (\sigma_c (S))$

Transformation Rules (cont)

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 $\bowtie$

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

HEURISTIC QUERY OPTIMIZATION

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

- The main heuristic is to apply first the operations that reduce the size of intermediate results
- 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.)
- 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 Plan

- An execution plan for a rel. algebra query consists of a combination of the 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.
- **Materialized evaluation**: the result of an operation is stored as a temporary relation
- **Pipelined evaluation**: as the result of an operator is produced, it is forwarded to the next operator in sequence

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

Cost-based 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

- Different DBMSs 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
    - Number of levels (x) of each multilevel index
    - Number of first-level index blocks (b_{I1})
    - Number of distinct values (d) of an attribute
    - Selectivity (sl) of an attribute
    - Selection cardinality (s) of an attribute. (s = sl * r)

Examples of Cost Functions for SELECT

- Linear search (brute force) approach
  - C = b;
  - For an equality condition on a key, C_{S1a} = (b/2) if the record is found; otherwise C_{S1a} = b.
- Binary search:
  - $C = \log_2 b + \lceil (s/bfr) \rceil - 1$
  - For an equality condition on a unique (key) attribute, $C = \log_2 b$
- Using a hash key to retrieve a single record:
  - C = 1

SEMANTIC QUERY OPTIMIZATION

- **Semantic Query Optimization**:
  - Uses constraints specified on the database schema in order to modify one query into another query that is more efficient to execute.
- Consider the following SQL query,

  Select E.LNAME, M.LNAME

  From EMPLOYEE E M

  Where E.SUPERSSN=M.SSN AND E.SALARY>M.SALARY
- Constraint: No employee can earn more than his or her direct supervisor
- If the semantic query optimizer checks for the existence of this constraint, it need not execute the query at all because it knows that the result of the query will be empty. Techniques known as theorem proving can be used for this purpose.

QUERY OPTIMIZATION IN ORACLE

- **Rule-based query optimization**: the optimizer chooses execution plans based on heuristically ranked operations. (earlier versions)
- **Cost-based query optimization**: the optimizer examines alternative access paths and operator algorithms and chooses the execution plan with lowest estimate cost.

  The query cost is calculated based on the estimated usage of resources such as I/O, CPU and memory needed.
- Application developers could specify hints to the ORACLE query optimizer. (The idea is that an application developer might know more information about the data)
- Possible to see the plan
  - EXPLAIN PLAN for select title from movie;
  - SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());