Lecture 8

Query Optimization

Lecture Sources

- These slides are based on
  - Chapters 19 in:

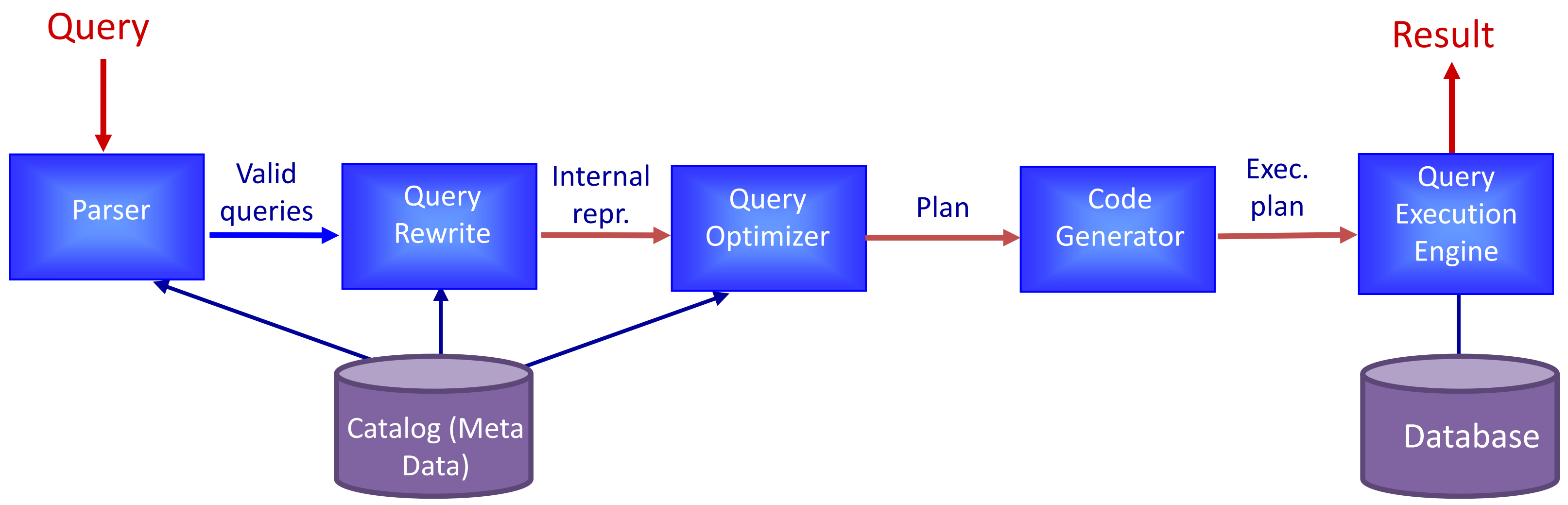
    El Masri, R and Navathe, S., "*Fundamentals of Database Systems*", 7th Edition, Pearson Int. Edition, Addison Wesley, ISBN: 0-321-4150-X (2007).

  - Chapter s 13 and 14 in:

    Henry F. Korth, Abraham Silberschatz, *Database System Concepts*, *(5th Ed.).* McGraw-Hill, 2006

Query Processing

- Deals with developing algorithms that analyze queries to generate a *good* execution plan that defines a sequence of steps for query evaluation, each step corresponds to one relational operation and its evaluation method.



Phases of Query Processing

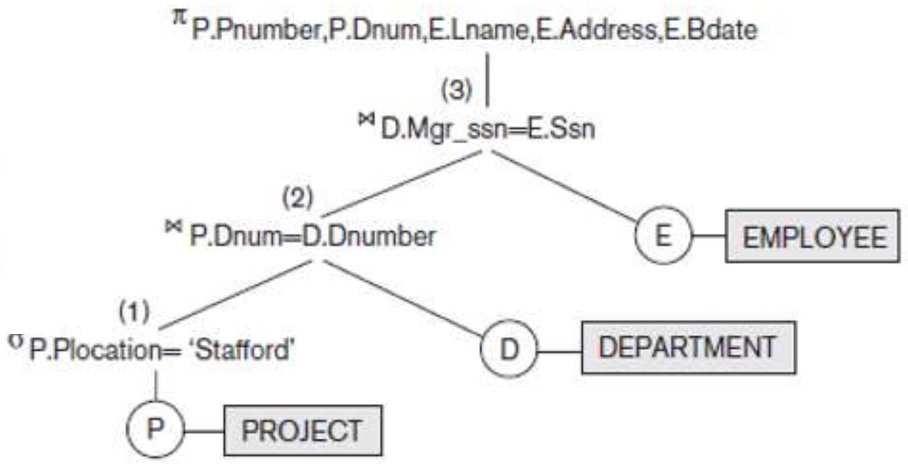
The Process of Query Optimization

- **Query optimization**: the process of choosing a suitable execution strategy for processing a query.

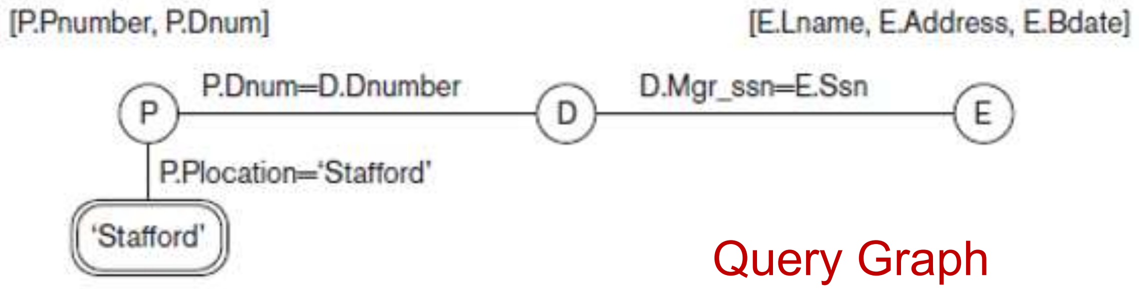
- Two internal representations of a query
  - **Query Tree**
  - **Query Graph**

Query Trees and Query Graph

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



Query Tree



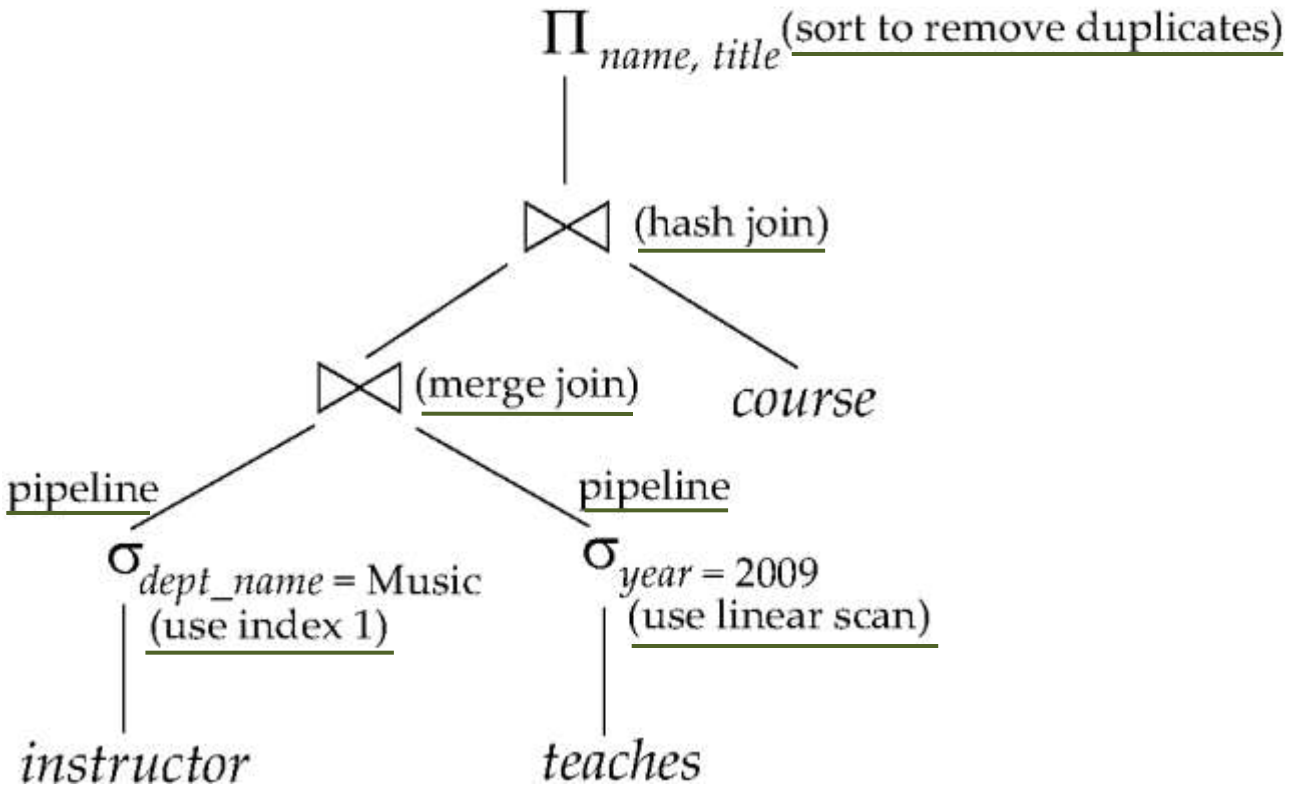
Query Graph

Query Tree

- A tree data structure 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*.
- 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.

Query Evaluation Plan (QEP)

- A **query evaluation plan** (**QEP**) defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated


$\Pi_{name,\ title}$ (sort to remove duplicates)
$\bowtie$ (hash join)
$\bowtie$ (merge join)
course
pipeline
pipeline
$\sigma_{dept\_name = \text{Music}}$ (use index 1)
$\sigma_{year = 2009}$ (use linear scan)
instructor
teaches

Query Optimization Problem

- For each query there is a number of equivalent execution plans that can produce the same result.

  - Search Space

  For a query with ***m*** operations, each operation can be evaluated in ***k*** different ways, the search space can be as many as **(m!) . k** different execution plans.

- The goal of query optimization is to select an execution plan such that the total cost of executing the query is minimum.

  Total cost = I/O cost + CPU cost

Measures of Query Cost

- Cost difference between query evaluation plans can be enormous
  - E.g. seconds vs. days in some cases
  - It is worth spending time in finding “best” QEP
- Steps in **cost-based query optimization**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate in all possible ways resulting expressions to get alternative QEP
  3. Evaluate/estimate the cost (execution time) of each QEP
  4. Choose the cheapest QEP based on **estimated cost**
- Estimation of QEP cost based on:
  - Statistical information about relations (stored in the **Catalog**)
    - number of tuples, number of distinct values for an attribute
  - Statistics estimation for intermediate results
    - to compute cost of complex expressions
  - Cost formulae for algorithms, computed using statistics

Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - *disk accesses, CPU*, or even network *communication*
- Typically, disk access is the predominant cost and is also relatively easy to estimate.
- Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost

Choice of Evaluation Plans

- Must consider the interaction of evaluation techniques when choosing evaluation plans
  - choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. E.g.
    - merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for an outer level aggregation
    - nested-loop join may provide opportunity for *pipelining*
- Practical query optimizers incorporate elements of the following two broad approaches:
  1. Search all the plans and choose the best plan in a cost-based fashion
  2. Uses heuristics to choose a plan

Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
  - Note: order of tuples is irrelevant (and also order of attributes)
- In SQL, inputs and outputs are multisets of tuples
  - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance
  - We focus on relational algebra and treat relations as sets
- An **equivalence rule** states that expressions of two forms are equivalent
  - One can replace an expression of first form by one of the second form, or vice versa

Equivalence Rules

General Transformation Rules for Relational Algebra Operations

1. Cascade of s:

$$s_{\text{c1 AND c2 AND ... AND cn}}(R) = s_{c1}\,(s_{c2}\,(\ldots(s_{cn}(R))\ldots)\,)$$

2. Commutativity of s:

$$s_{c1}\,(s_{c2}(R)) = s_{c2}\,(s_{c1}(R))$$

3. Cascade of p:

$$p_{\text{List1}}\,(p_{\text{List2}}\,(\ldots(p_{\text{Listn}}(R))\ldots)\,) = p_{\text{List1}}(R)$$

4. Commuting s with p:

   If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:

$$p_{A1,\ A2,\ \ldots,\ An}\,(s_c\,(R)) = s_c\,(p_{A1,\ A2,\ \ldots,\ An}\,(R))$$

Equivalence Rules

General Transformation Rules for Relational Algebra Operations

5. Commutativity of ⋈(and x):

$$R \bowtie_c S = S \bowtie_c R; \qquad R \text{ X } S = S \text{ X } R$$

6. Commuting s 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:

$$s_c (R \bowtie S) = (s_c (R)) \bowtie S$$

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:

$$s_c (R \bowtie S) = (s_{c1} (R)) \bowtie (s_{c2} (S))$$

Equivalence Rules

General Transformation Rules for Relational Algebra Operations

7. Commuting p 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:

$$p_L (R \bowtie_C S) = (p_{A1, \ldots, An} (R)) \bowtie_C (p_{B1, \ldots, Bm} (S))$$

If the join condition *C* contains additional attributes not in L, these must be added to the projection list, and a final p operation is needed.

Equivalence Rules

General Transformation Rules for Relational Algebra Operations

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 q stands for any one of these four operations (throughout the expression), we have

$$(R \; q \; S) \; q \; T \;=\; R \; q \; (S \; q \; T)$$

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

$$s_c (R \; q \; S) \;=\; (s_c (R)) \; q \; (s_c (S))$$

Equivalence Rules

General Transformation Rules for Relational Algebra Operations

11. The p operation commutes with U.

p_L (R U S) = (p_L (R)) U (p_L (S))

12. Converting a (s, x) sequence into ⋈ :

If the condition c of a s that follows a x Corresponds to a join condition, convert the (s, x) sequence into a ⋈ as follows:

(s_C (R x S)) = (R $\bowtie_C$ S)

Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given one
- Can generate all equivalent expressions as follows:
  - Repeat (starting from the set containing only the given expression)
    - apply all applicable equivalence rules on every sub-expression of every equivalent expression found so far
    - add newly generated expressions to the set of equivalent expressions

    Until no new equivalent expressions are generated
- The above approach is very expensive in space and time
  - Space: efficient expression-representation techniques
    - 1 copy is stored for shared sub-expressions
  - Time: partial generation
    - Dynamic programming
    - Greedy techniques (select best choices at each step)
    - Heuristics, e.g., single-relation operations
      (selections, projections) are pushed inside (performed earlier)

Equivalence Rules

Summary of Heuristics for Algebraic Optimization

1. The main heuristic is to apply first the operations that *reduce the size* of intermediate results.

2. 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.)

3. 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.)

Cost-Based Optimization

- A big part of a cost-based optimizer (based on equivalence rules) is choosing the “best” order for join operations
- Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$.
- There are $(2(n-1))!/(n-1)!$ different join orders for above expression. With $n = 7$, the number is 665280, with $n = 10$, the number is greater than 17.6 billion!
- No need to generate all the join orders. Exploiting some monotonicity (optimal substructure property), the least-cost join order for any subset of $\{r_1, r_2, \ldots, r_n\}$ is computed only once.

Heuristic Optimization

- Cost-based optimization is expensive
- Systems may use *heuristics* to reduce the number of possibilities choices that must be considered
- Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:
  - Perform selection early (reduces the number of tuples)
  - Perform projection early (reduces the number of attributes)
  - Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations
  - Only consider left-deep join orders (particularly suited for pipelining as only one input has to be pipelined, the other is a relation)

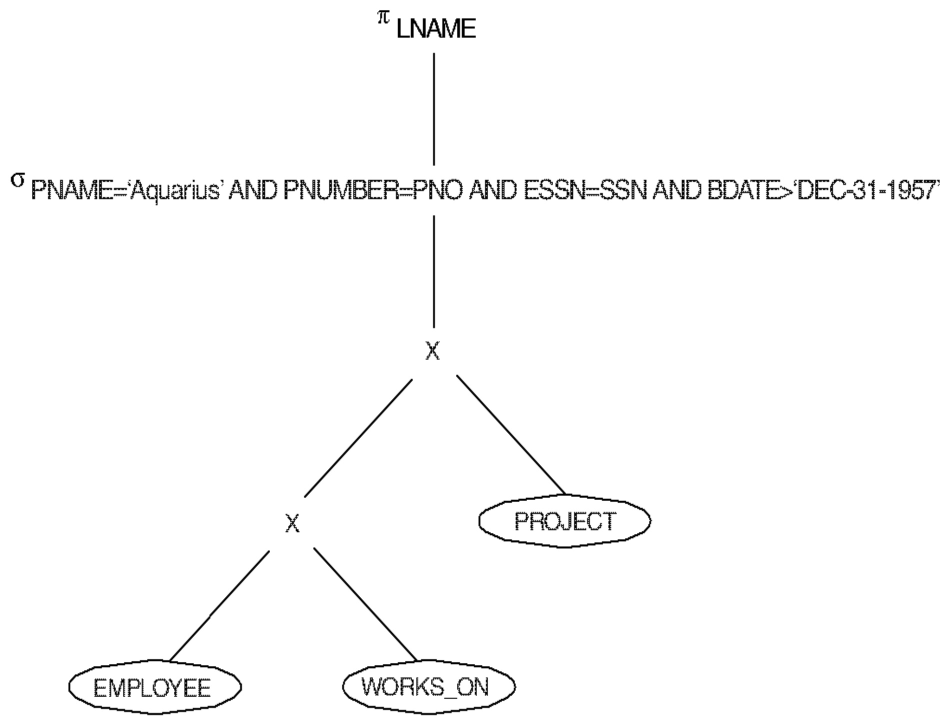
Using Heuristics in Query Optimization (9)

- **Example**:

```
Q: SELECT LNAME
     FROM    EMPLOYEE, WORKS_ON, PROJECT
     WHERE  PNAME = 'AQUARIUS' AND PNMUBER=PNO
                 AND ESSN=SSN AND BDATE > '1957-12-31';
```

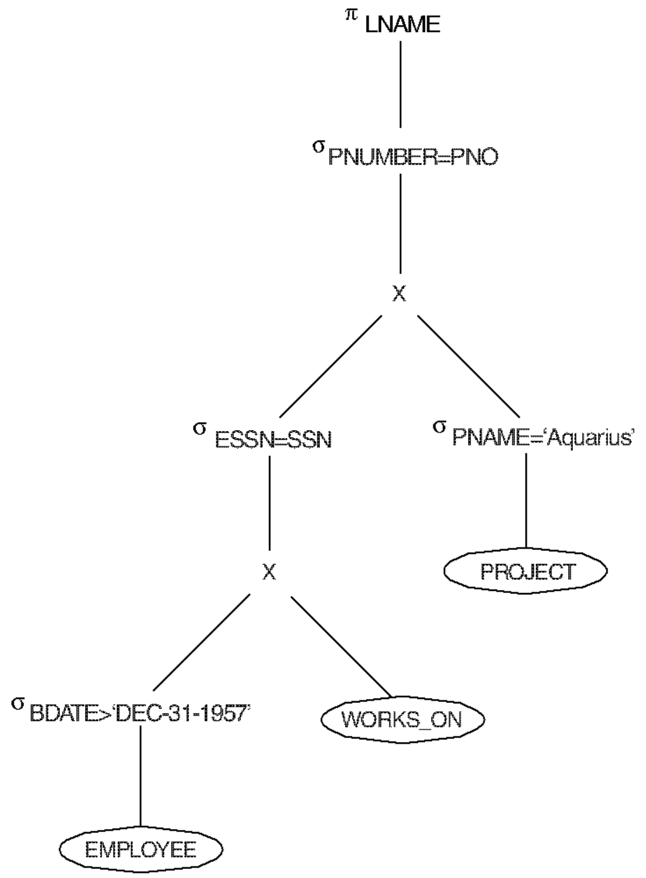
Using Heuristics in Query Optimization (10)

Steps in converting a query tree using heuristic optimization



π LNAME
σ PNAME='Aquarius' AND PNUMBER=PNO AND ESSN=SSN AND BDATE>'DEC-31-1957'
X
X
PROJECT
EMPLOYEE
WORKS_ON

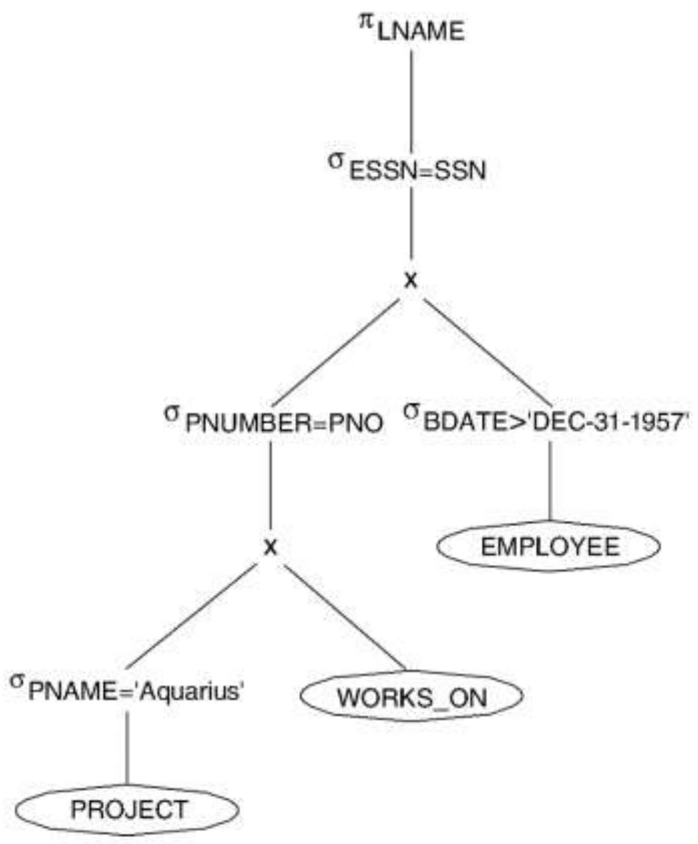
(a) Initial (canonical) query tree



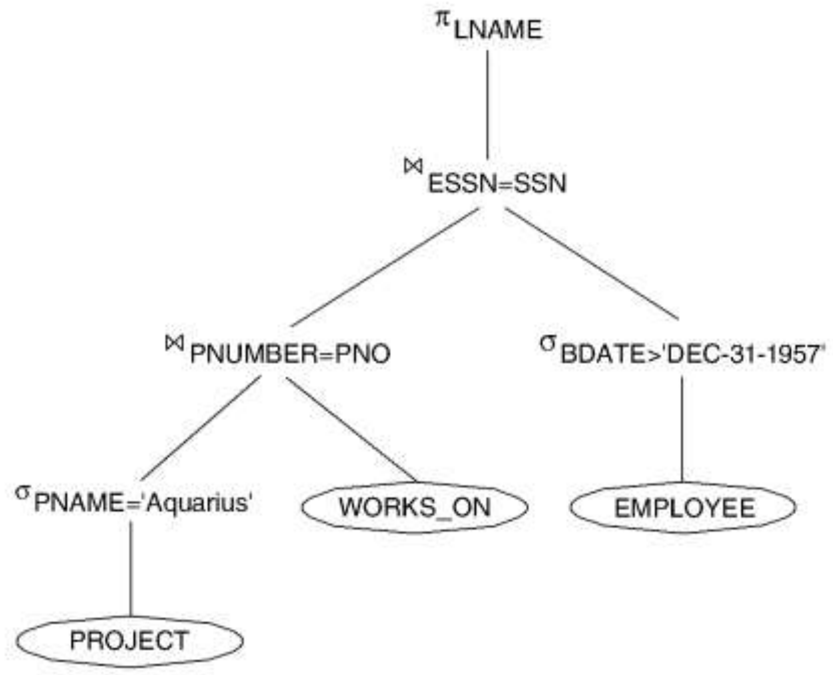
(b) Moving selection down the tree

Using Heuristics in Query Optimization (11)

Steps in converting a query tree using heuristic optimization



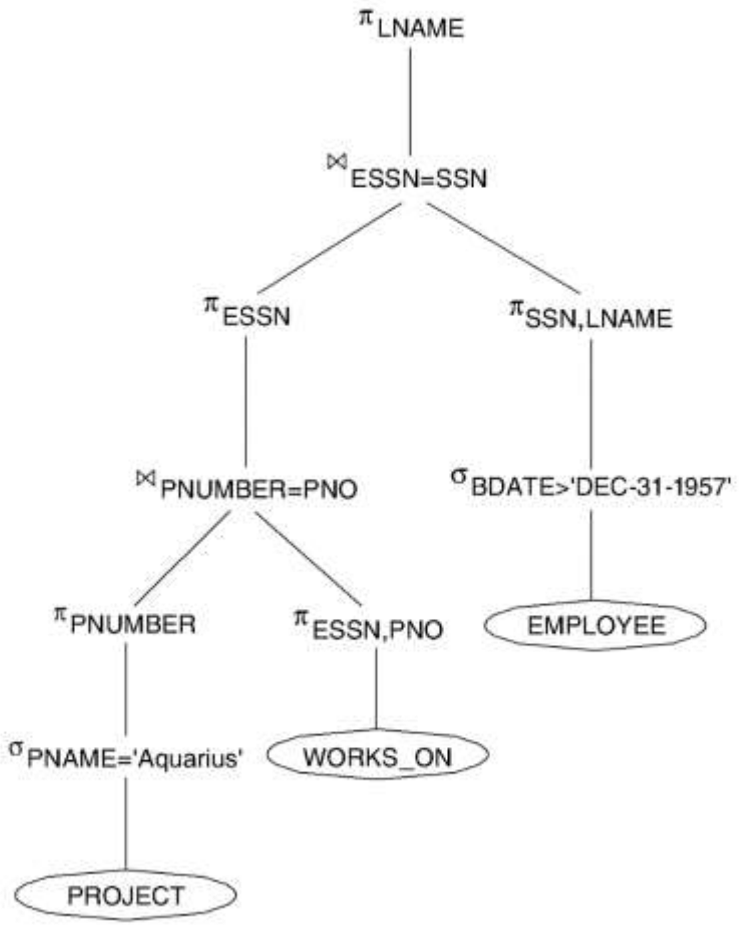
(c) Apply the more restrictive SELECT operations first.



(d) Replace CARTESIEN Product and SELECT with JOIN operations

Using Heuristics in Query Optimization (12)

Steps in converting a query tree using heuristic optimization



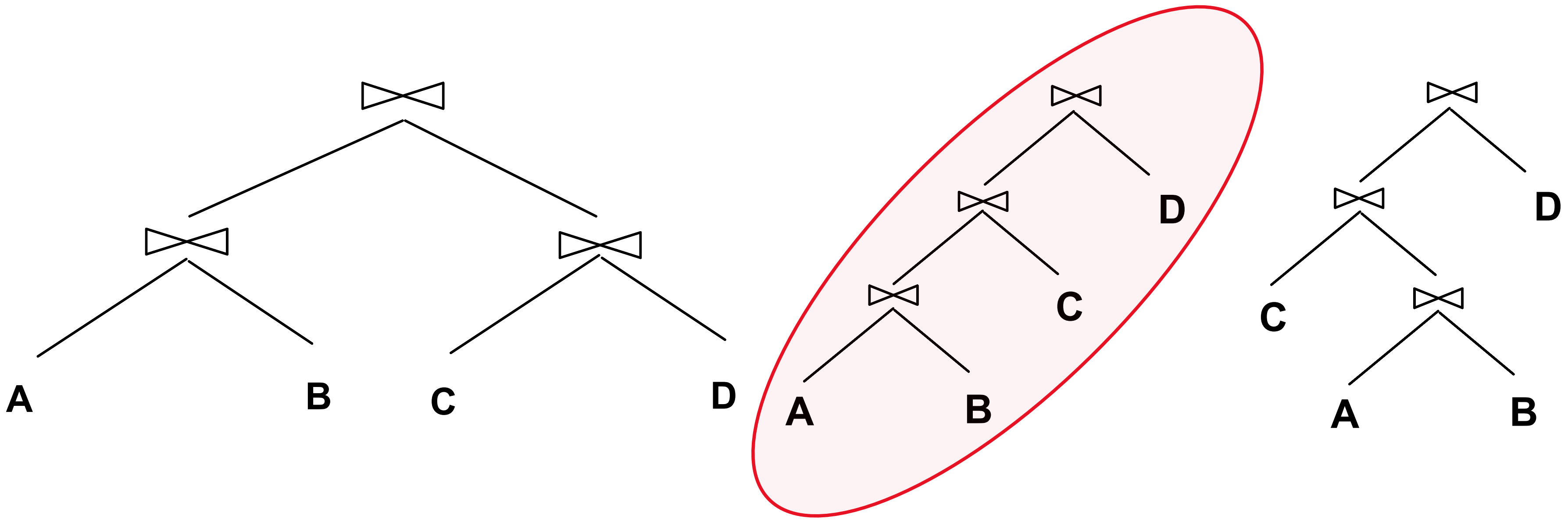
(e) Move PROJECT operations down the query tree

Structure of Query Optimizers

- Some systems use only heuristics, others combine heuristics with partial cost-based optimization.
- Many optimizers considers only left-deep join orders.
  - Plus heuristics to push selections and projections down the query tree
  - Reduces optimization complexity and generates plans amenable to pipelined evaluation.
- Heuristic optimization used in some versions of Oracle:
  - Repeatedly pick "best" relation to join next
    - it obtains and compares *n* plans (each starting with one relation. In each plan, pick the best next relation for the join

left-deep join orders

- Left-deep trees allow us to generate all *fully pipelined plans*. Intermediate results not written to temporary files.
- The *inner relation* of each join is a *base relation*


A
B
C
D
A
B
C
D
C
D
A
B