# Chapter 13: Query Optimization

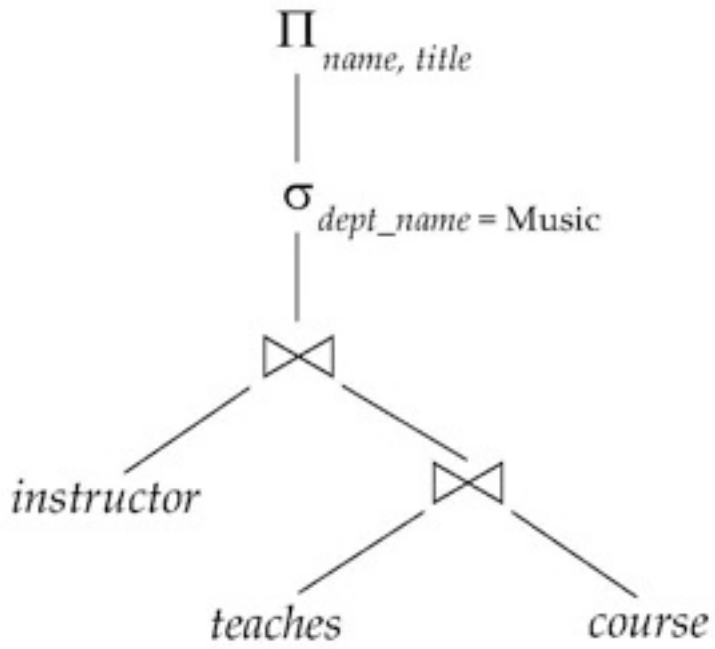
**Database System Concepts, 6th Ed.**

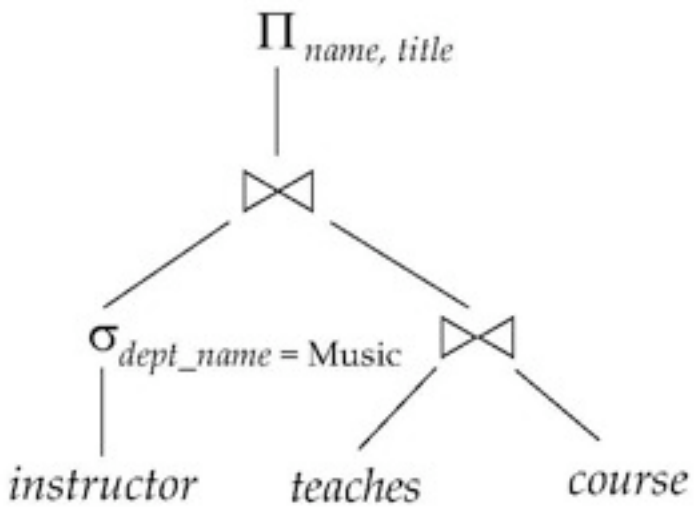
# Chapter 13: Query Optimization

- Introduction
  - Transformation of Relational Expressions
  - Cost-based optimization
  - Dynamic Programming for Choosing Evaluation Plans
  - Heuristic Optimization
  - Structure of Query Optimizers

# Introduction

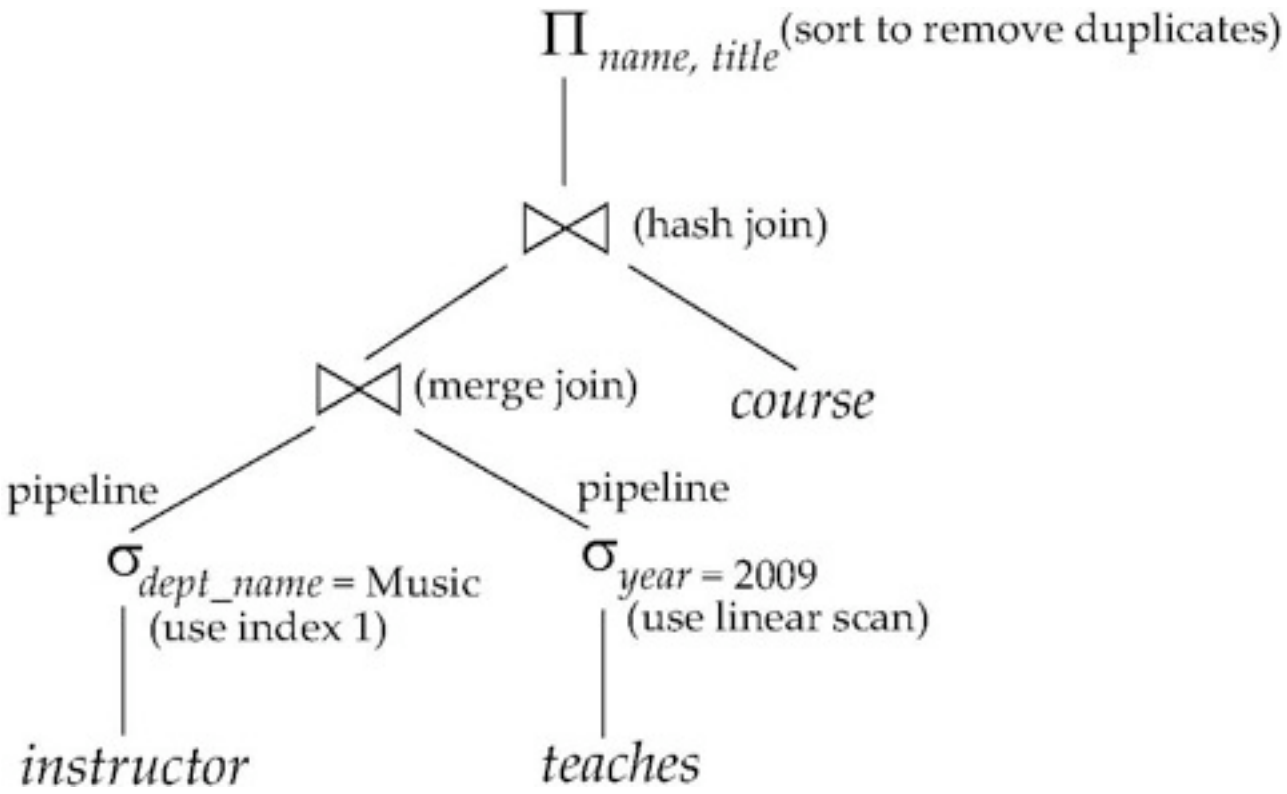
- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation



# Introduction (Cont.)

- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.



# Introduction (Cont.)

- Cost difference between evaluation plans for a query can be enormous
  - E.g. seconds vs. days in some cases
- Steps in **cost-based query optimization**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate resultant expressions to get alternative **query plans**
  3. Choose the cheapest plan based on **estimated cost**

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- Heuristic Optimization
- Structure of Query Optimizers

# 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
- An **equivalence rule** says that expressions of two forms are equivalent
  - Can replace expression of first form by second, or vice versa

# Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$$

4. Selections can be combined with Cartesian products and theta joins.

   a. $\sigma_{\theta}(E_1 \text{ X } E_2) = E_1 \bowtie_{\theta} E_2$

   b. $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

# Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. (a) Natural join operations are associative:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b) Theta joins are associative in the following manner:

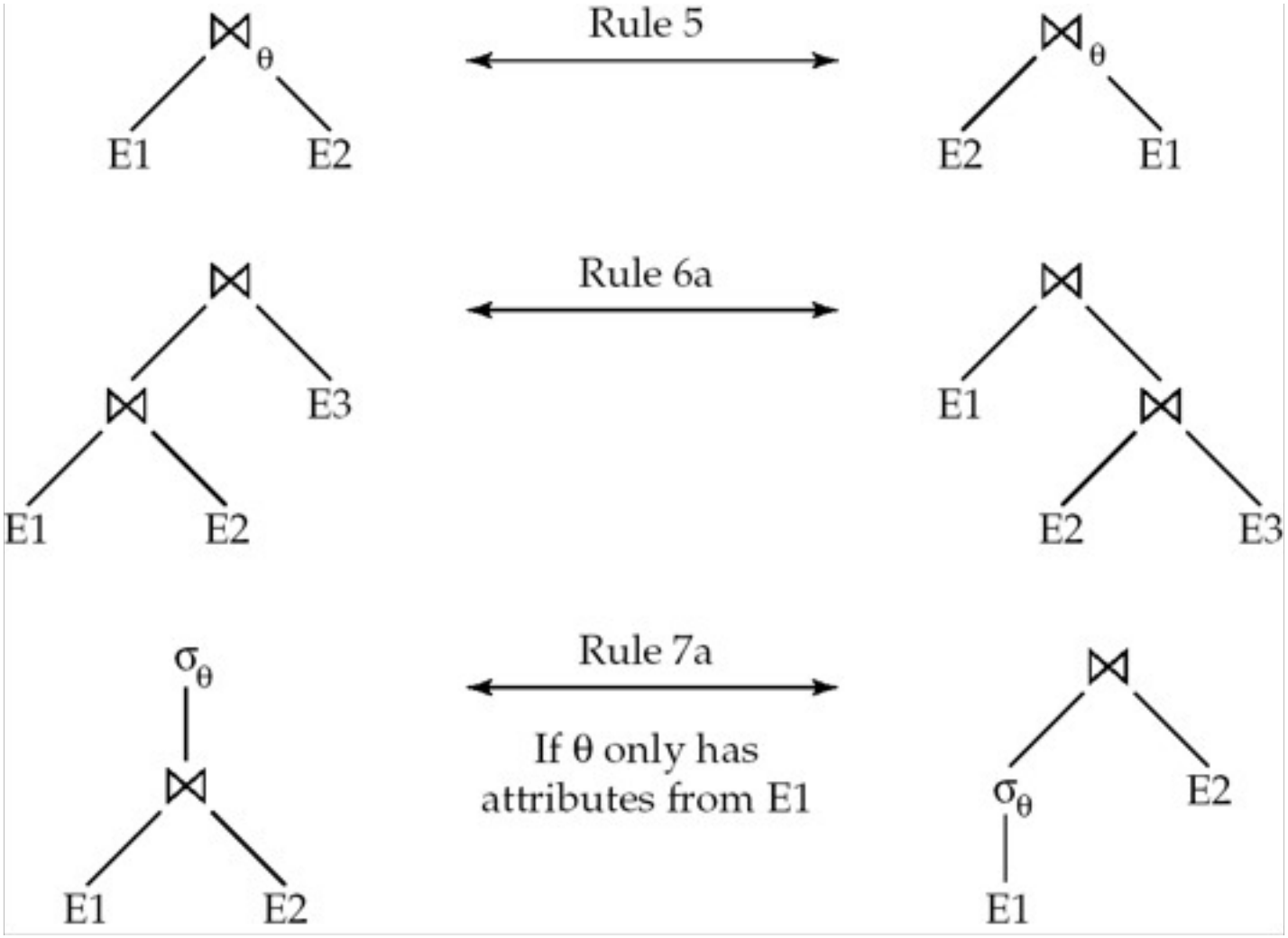
$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

where $\theta_2$ involves attributes from only $E_2$ and $E_3$.

$$E_1 = (A, B),\ E_2 = (C, D),\ E_3 = (F, G)$$

$$(E_1 \bowtie_{A = C} E_2) \bowtie_{B = F \wedge D = G} E_3 \quad = \quad E_1 \bowtie_{A = C \wedge B = F} (E_2 \bowtie_{D = G} E_3)$$

# Pictorial Depiction of Equivalence Rules



# Equivalence Rules (Cont.)

7. The selection operation distributes over the theta join operation under the following two conditions:
   (a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions ($E_1$) being joined.

$$\sigma_{\theta 0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_{\theta} E_2$$

   (b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

$$\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_{\theta} (\sigma_{\theta 2}(E_2))$$

$E_1$ (A, B), $E_2$ (C, D)

$$\sigma_{A<B}(E_1 \bowtie_{B=C} E_2) = \sigma_{A<B}(E_1) \bowtie_{B=C} E_2$$

$$\sigma_{A<D}(E_1 \bowtie_{B=C} E_2)?$$

# Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

   (a) if $\theta$ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   (b) Consider a join $E_1 \bowtie_\theta E_2$.

   - Let $L_1$ and $L_2$ be sets of attributes from $E_1$ and $E_2$, respectively.
   - Let $L_3$ be attributes of $E_1$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$, and
   - let $L_4$ be attributes of $E_2$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

$E_1$ (A, B, Z), $E_2$ (C, D)

$\Pi_A(E_1 \bowtie_{B=C} E_2) = \Pi_A(\Pi \quad (E_1) \bowtie_{B=C} \Pi \quad (E_2))$

# Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

   (a) if $\theta$ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   (b) Consider a join $E_1 \bowtie_\theta E_2$.

   - Let $L_1$ and $L_2$ be sets of attributes from $E_1$ and $E_2$, respectively.
   - Let $L_3$ be attributes of $E_1$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$, and
   - let $L_4$ be attributes of $E_2$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

$E_1$ (A, B, Z), $E_2$ (C, D)

$$\Pi_A(E_1 \bowtie_{B=C} E_2) = \Pi_A(\Pi_{A,B}(E_1) \bowtie_{B=C} \Pi \quad (E_2))$$

# Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

   (a) if $\theta$ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   (b) Consider a join $E_1 \bowtie_\theta E_2$.

   - Let $L_1$ and $L_2$ be sets of attributes from $E_1$ and $E_2$, respectively.
   - Let $L_3$ be attributes of $E_1$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$, and
   - let $L_4$ be attributes of $E_2$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

$E_1$ (A, B, Z), $E_2$ (C, D)

$$\Pi_A(E_1 \bowtie_{B=C} E_2) = \Pi_A(\Pi_{A,B}(E_1) \bowtie_{B=C} \Pi_C(E_2))$$

# Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

   - (set difference is not commutative).

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. The selection operation distributes over $\cup$, $\cap$ and $-$.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

and similarly for $\cup$ and $\cap$ in place of $-$

Also: $$\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

and similarly for $\cap$ in place of $-$, but not for $\cup$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

# Join Ordering Example

- For all relations $r_1$, $r_2$, and $r_3$,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

  (Join Associativity)

- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose

$$(r_1 \bowtie r_2) \bowtie r_3$$

  so that we compute and store a smaller temporary relation.

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- **Cost-based optimization**
- Dynamic Programming for Choosing Evaluation Plans
- Heuristic Optimization
- Structure of Query Optimizers

# Cost-Based Optimization

- Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie \ldots r_n$.
- 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 176 billion!
- No need to generate all the join orders. Using dynamic programming, the least-cost join order for any subset of $\{r_1, r_2, \ldots r_n\}$ is computed only once and stored for future use.

$$(((E_1 \bowtie E_2 \bowtie E_3) \bowtie E_4) \bowtie E_5) = (E_1 \bowtie E_2 \bowtie E_3) \bowtie (E_4 \bowtie E_5)$$

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- Heuristic Optimization
- Structure of Query Optimizers

# Dynamic Programming in Optimization

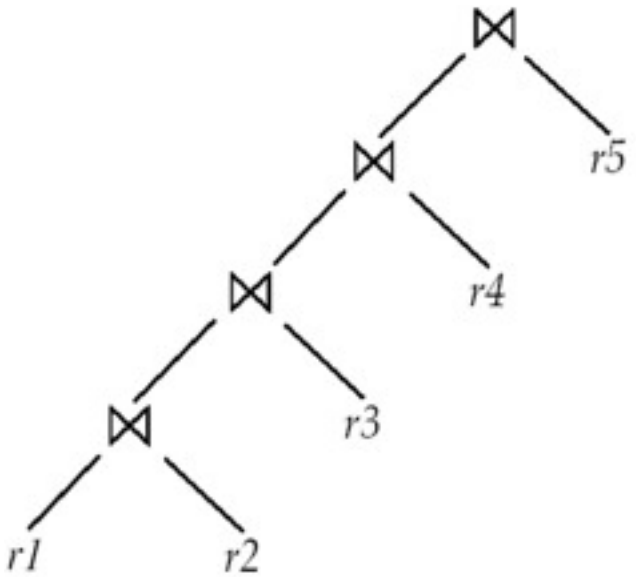
- To find best join tree for a set of *n* relations:
  - To find best plan for a set *S* of *n* relations, consider all possible plans of the form: $S_1 \bowtie (S - S_1)$ where $S_1$ is any non-empty subset of *S*.
  - Recursively compute costs for joining subsets of *S* to find the cost of each plan. Choose the cheapest.
  - When plan for any subset is computed, store it and reuse it when it is required again, instead of recomputing it
    - Dynamic programming

# Join Order Optimization Algorithm

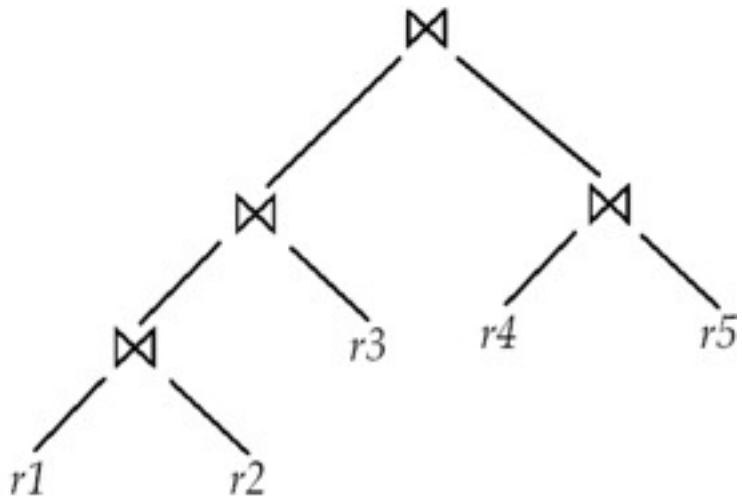
```
procedure findbestplan(S)
    if (bestplan[S].cost ≠ ∞)
        return bestplan[S]
    // else bestplan[S] has not been computed earlier, compute it now
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on the best way
        of accessing S  /* Using selections on S and indices on S */
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1= findbestplan(S1)
        P2= findbestplan(S - S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = "execute P1.plan; execute P2.plan;
                                join results of P1 and P2 using A"
    return bestplan[S]
```

# Left Deep Join Trees

- In **left-deep join trees,** the right-hand-side input for each join is a relation, not the result of an intermediate join.




(a) Left-deep join tree




(b) Non-left-deep join tree

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- **Heuristic Optimization**
- Structure of Query Optimizers

# Heuristic Optimization

- Cost-based optimization is expensive, even with dynamic programming.
- Systems may use *heuristics* to reduce the number of choices that must be made in a cost-based fashion.
- 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.
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- Heuristic Optimization
- Structure of Query Optimizers

# Structure of Query Optimizers

- 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
    - Starting from each of n starting points. Pick best among these
- Intricacies of SQL complicate query optimization
  - E.g. nested subqueries
- **Optimization cost budget** to stop optimization early (if cost of plan is less than cost of optimization)
- **Plan caching** to reuse previously computed plan if query is resubmitted

# Chapter 13: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- Heuristic Optimization
- Structure of Query Optimizers