Chapter 13: Query Optimization

Data Management for Big Data

2018-2019 (spring semester)

Dario Della Monica

These slides are a modified version of the slides provided with the book

Database System Concepts, 6th Ed.

The original version of the slides is available at: https://www.db-book.com/

Chapter 13: Query Optimization

- Introduction
- Generating Equivalent Expressions
- Statistical Information for Cost Estimation (the Catalog)
- Choice of Evaluation Plans
  - Dynamic Programming for Choosing Evaluation Plans

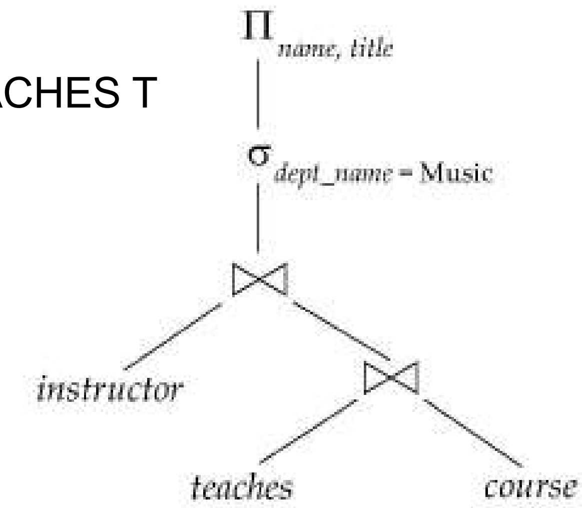
Introduction

- **Query optimization** is the process the best **query execution plan** (**QEP**) among the many possible ones
- Alternative ways to execute a given query
  - Equivalent relational algebra expressions
  - Different implementation choices for each relational algebra operation

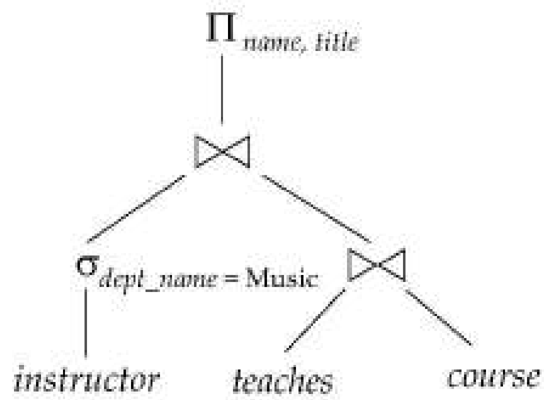
INSTR(i_id, name, dept_name, ...)
COURSE(c_id, title, ...)
TEACHES(i_id, c_id, ...)

The name of all instructors in the department of Music together with the titles of all courses they teach

```
SELECT  I.name, C.title
FROM INSTR I, COURSE C, TEACHES T
WHERE I.i_id = T.i_id
 AND T.c_id = C.c_id
AND dept_name="Music"
```



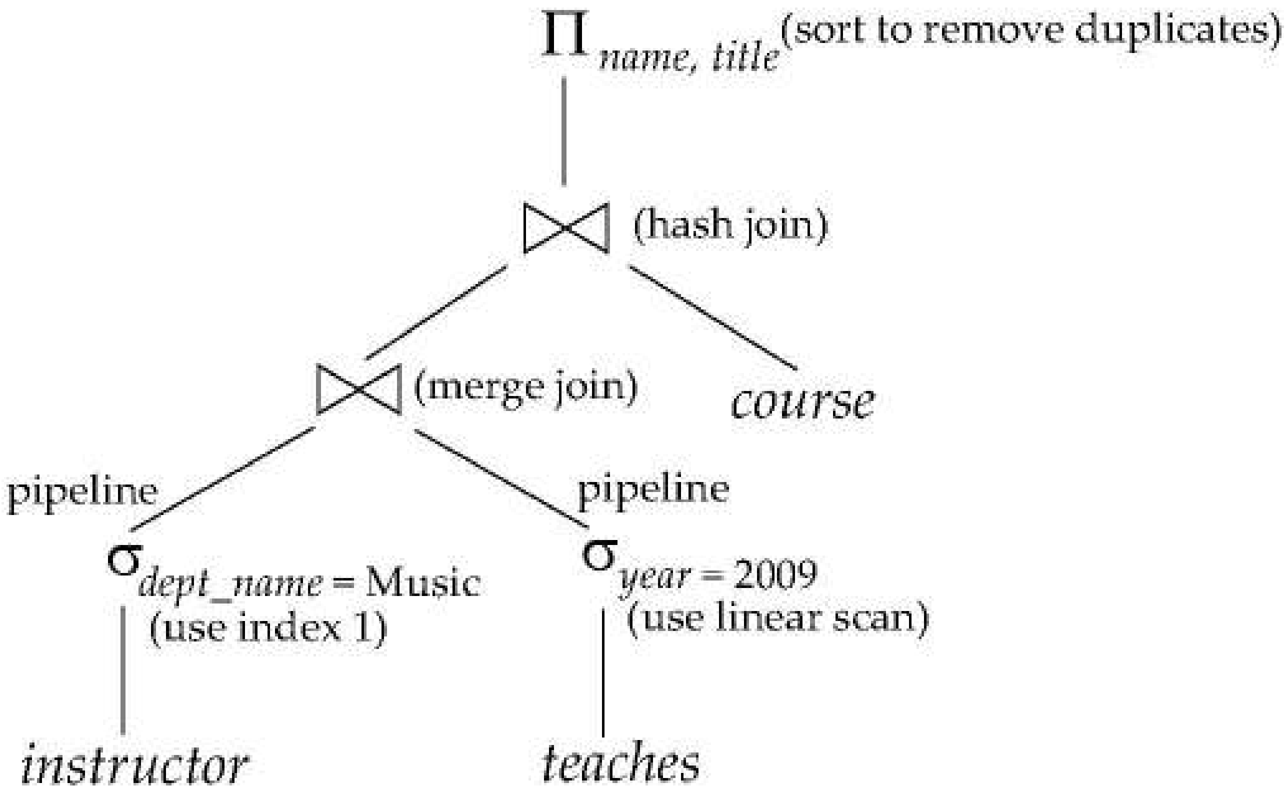
$\prod(\sigma(INSTR \bowtie (TEACHES \bowtie COURSE)))$



$\prod(\sigma(INSTR) \bowtie (TEACHES \bowtie COURSE))$

Introduction (Cont.)

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



- Find out how to view query execution plans on your favorite database

Introduction (Cont.)

- Cost difference between query evaluation plans 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 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

Generating Equivalent Expressions

Database System Concepts, 6th Ed.

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)
  - we don’t care if they generate different results on databases that violate integrity constraints (e.g., uniqueness of keys)
- 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

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

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

   where $L_1 \subseteq L_2 \subseteq \ldots \subseteq L_n$

4. Selections can be combined with Cartesian products and theta joins.
   a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   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 (and thus natural joins) operations are commutative.

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

   (but the order is important for efficiency)

6. (a) Natural join operations are associative:

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

   (again, the order is important for efficiency)

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

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

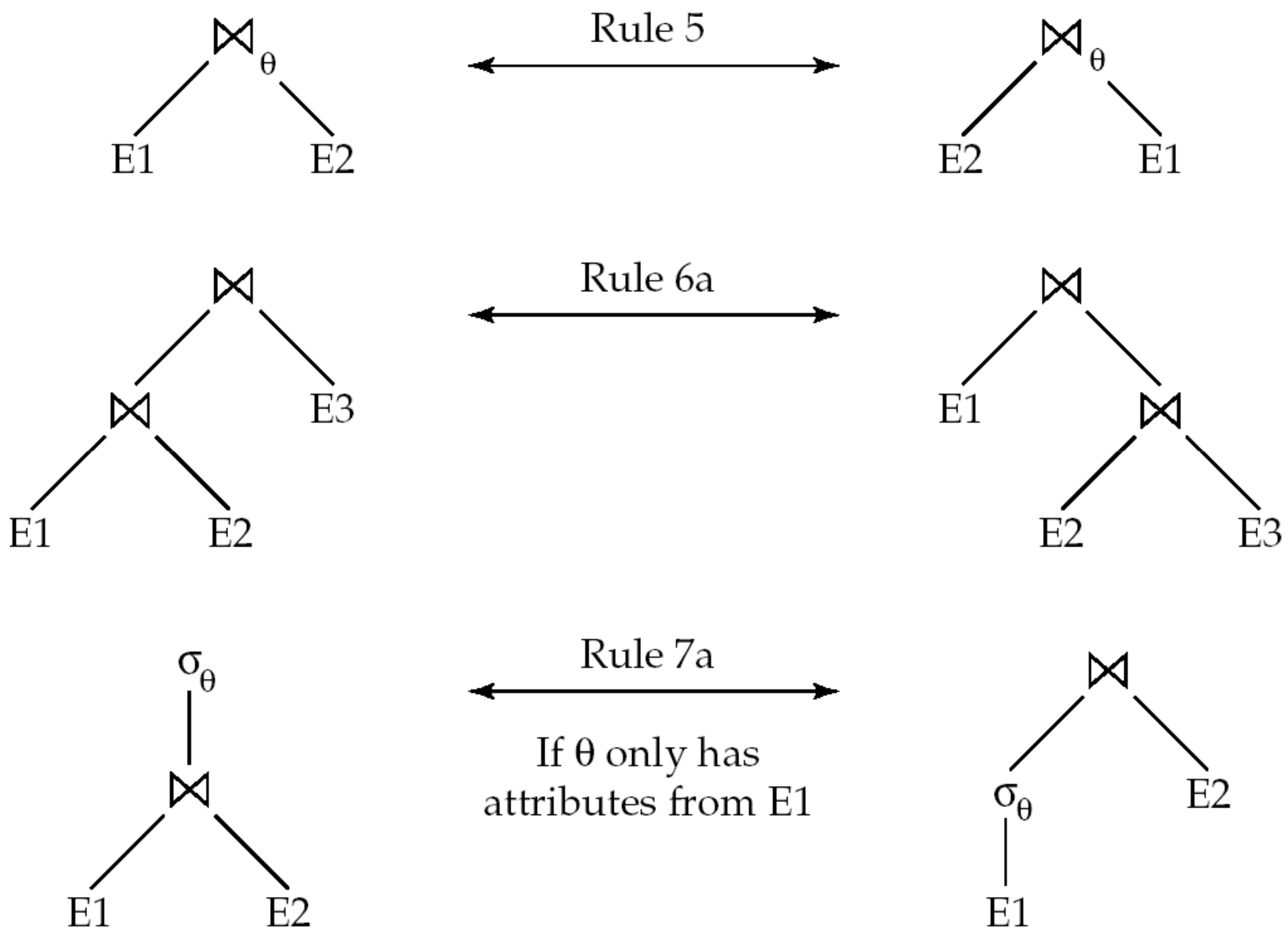
   where θ_1 involves attributes from only E_1 and E_2
   and θ_2 involves attributes from only E_2 and E_3

More equivalences at Ch. 13.2 of the book *

---

* Silberschatz, Korth, and Sudarshan, *Database System Concepts*, 6° ed.

Pictorial Depiction of Equivalence Rules



Exercise

- Create equivalence rules to push selection inside a left outer join Ex. 13.1(c) *
- Disprove the equivalence Ex. 13.1(d) *

$$(R ⟕ S) ⟕ T \;=\; R ⟕ (S ⟕ T)$$

Definition (left outer join): the result of a left outer join T = R ⟕ S is a super-set of the result of the join T' = R ⋈ S in that all tuples in T' appear in T. In addition, T preserve those tuples that are lost in the join, by creating tuples in T that are filled with *null* values

STUD

stud_id	name	surname
1	gino	bianchi
2	filippo	neri
3	mario	rossi

TAKES

stud_id	course	grade
1	Math	30
1	Algebra	26
2	Progr.	22
2	Math	28
2	Logic	30

STUD ⟕ TAKES

stud_id	name	surname	course	grade
1	gino	bianchi	Math	30
1	gino	bianchi	Algebra	26
2	filippo	neri	Progr.	22
2	filippo	neri	Math	28
2	filippo	neri	Logic	30
3	mario	rossi	***null***	***null***

* **Silberschatz, Korth, and Sudarshan, *Database System Concepts*, 6° ed.**

Solutions

- Create equivalence rules involving left outer join and selection

$$\sigma_\theta (R ⟕ S) = \sigma_\theta (R) ⟕ S$$

where θ uses only attributes of R

- Disprove the equivalence $(R ⟕ S) ⟕ T = R ⟕ (S ⟕ T)$

R

A_R	A_{RS}	A_{RT}
1	1	1

S

A_S	A_{RS}	A_{ST}
1	1	1

T

A_T	A_{RT}	A_{ST}
1	2	1

$R ⟕ S$

A_R	A_{RS}	A_{RT}	A_S	A_{ST}
1	1	1	1	1

$S ⟕ T$

A_S	A_{RS}	A_{ST}	A_T	A_{RT}
1	1	1	1	2

$(R ⟕ S) ⟕ T$

A_R	A_{RS}	A_{RT}	A_S	A_{ST}	A_T
1	1	1	1	1	null

$R ⟕ (S ⟕ T)$

A_R	A_{RS}	A_{RT}	A_S	A_{ST}	A_T
1	1	1	null	null	null

Solutions (cont'd)

- Disprove the equivalence $(R ⟕ S) ⟕ T = R ⟕ (S ⟕ T)$

  Another counter-example (to fix for solution given on the webpage of the book)

R

A	A_R
1	1

S

A	A_S
2	1

T

A	A_T
1	1

$R ⟕ S$

A	A_R	A_S
1	1	null

$S ⟕ T$

A	A_S	A_T
2	1	null

$(R ⟕ S) ⟕ T$

A	A_R	A_S	A_T
1	1	null	1

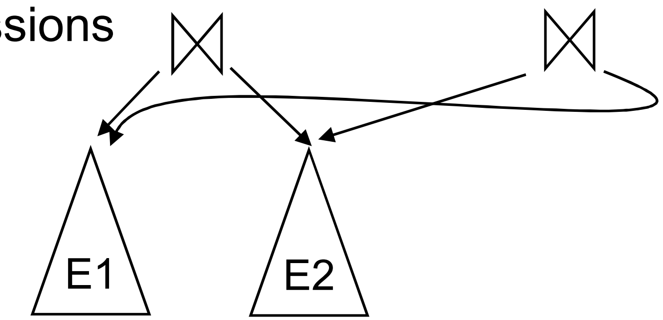
$R ⟕ (S ⟕ T)$

A	A_R	A_S	A_T
1	1	null	null

Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
  - Repeat
    - 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 above
- The above approach is very expensive in space and time
  - Space: sharing (re-using) common sub-expressions
    (detect duplicate sub-expressions and share one copy)
  - Time:
    - Dynamic programming
    - Greedy techniques (select best choices at each step)
    - Heuristics, e.g., single-relation operations
      (selections, projections) are pushed inside (performed earlier)



Statistical Information for Cost Estimation (the Catalog)

Database System Concepts, 6th Ed.

Cost Estimation

- Cost of each operator computed as described in Chapter 12 [*]
  - Need statistics of input relations
    - E.g. number of tuples, sizes of tuples
- Inputs can be results of sub-expressions
  - Need to estimate statistics of expression results
  - E.g., selectivity rate based on number of distinct values for an attribute
- Statistics are collected in the **Catalog**

---

[*] **Silberschatz, Korth, and Sudarshan, *Database System Concepts*, 6° ed.**

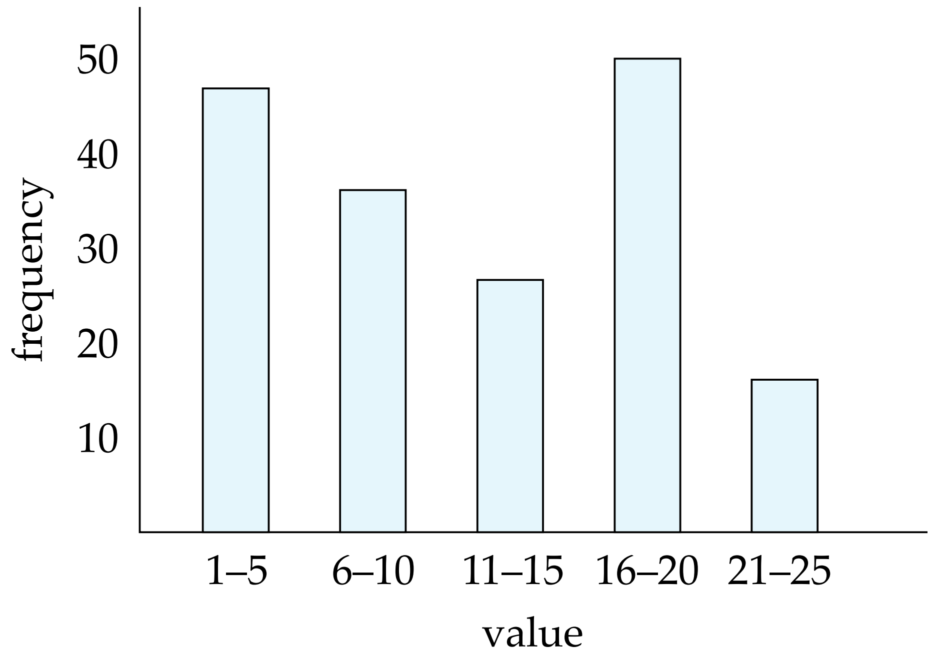
Statistical Information for Cost Estimation

- Statistics information for cost estimation is maintained in the **Catalog**
- The catalog is itself stored in the database
- It contains:
  - n_r: number of tuples in a relation *r*
  - b_r: number of blocks containing tuples of *r*
  - l_r: size of a tuple of *r* (in bytes)
  - f_r: blocking factor of *r* — i.e., the number of tuples of *r* that fit into one block
  - *V(A, r):* number of distinct values that appear in *r* for attribute *A;* same as the size of $\prod_A(r)$
  - *min(A,r)*: smallest value appearing in relation *r* for attribute *A*;
  - *max(A,r)*: largest value appearing in relation *r* for attribute *A*;
  - If tuples of *r* are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Histograms

- Histogram on attribute *age* of relation *person*



- For each range
  - Number of records (tuples) with value in the range
  - Also, number of distinct values in the range
- Without histogram information, uniform distribution is assumed

Selection Size Estimation

- $\sigma_{A=v}(r)$
  - $n_r / V(A,r)$: number of records that will satisfy the selection (uniform distribution)
  - Equality condition on a key attribute: *size estimate* = 1
- $\sigma_{A \leq v}(r)$ (case of $\sigma_{A \geq v}(r)$ is symmetric)
  - ***n***: estimated number of tuples satisfying the condition is computed assuming that min(A,r) and max(A,r) are available in catalog
    - n = 0 if v < min(A,r)
    - $n = n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$ otherwise (uniform distribution)
  - In absence of statistical information or when *v* is unknown at time of cost estimation (e.g., *v* is computed at run-time by the application using the DB) *n* is assumed to be $n_r / 2$
- If histograms are available, we can refine above estimate by using values for restricted ranges instead of values referring to the entire domain (n_r, *V(A,r), min(A, r), max(A, r)*)

Choice of Evaluation Plans

Database System Concepts, 6th Ed.

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

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 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.

Cost-Based Optimization: An example

- Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 \bowtie r_5$
- Number of possible different join orderings: $\frac{(2(n-1))!}{(n-1)!} = \frac{8!}{4!} = 1680$
- The least-cost join order for any subset of $\{ r_1, r_2, r_3, r_4, r_5 \}$ is computed only once
- Assume we want to compute $N_{123/45}$: number of possible different join orderings where r_1, r_2, r_3 sare grouped together, e.g.,

$(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$ $\quad$ $(r_2 \bowtie r_3 \bowtie r_1) \bowtie (r_5 \bowtie r_4)$ $\quad$ $r_4 \bowtie (r_5 \bowtie (r_1 \bowtie (r_2 \bowtie r_3)))$

- The naïve approach
  - $N_{123/45} = N_{123} * N_{45}$
  - $N_{123} = \frac{4!}{2!} = 12$ $\quad$ (N_{123} : # ways of arranging r_1, r_2, and r_3)
  - $N_{45} = N_{123} = 12$ $\quad$ (N_{45} : # ways of arranging r_4 and r_5 wrt. block of r_1, r_2, and r_3)
  - $N_{123/45} = 12 * 12 = 144$
- Exploiting optimal substructure property:
  - compute only once best ordering for $r_1 \bowtie r_2 \bowtie r_3$: 12 possibilities (N_{123})
  - compute best ordering for $R_{123} \bowtie r_4 \bowtie r_5$: 12 possibilities (N_{45})
  - Therefore, $\quad$ $N_{123/45} = 12 + 12 = 24$

Dynamic Programming in Optimization

- To find best join tree (equivalently, best join order) for a set of *n* relations:
  - To find best plan for a set *S* of *n* relations, consider all possible plans of the form:

    $$S' \bowtie (S \setminus S')$$

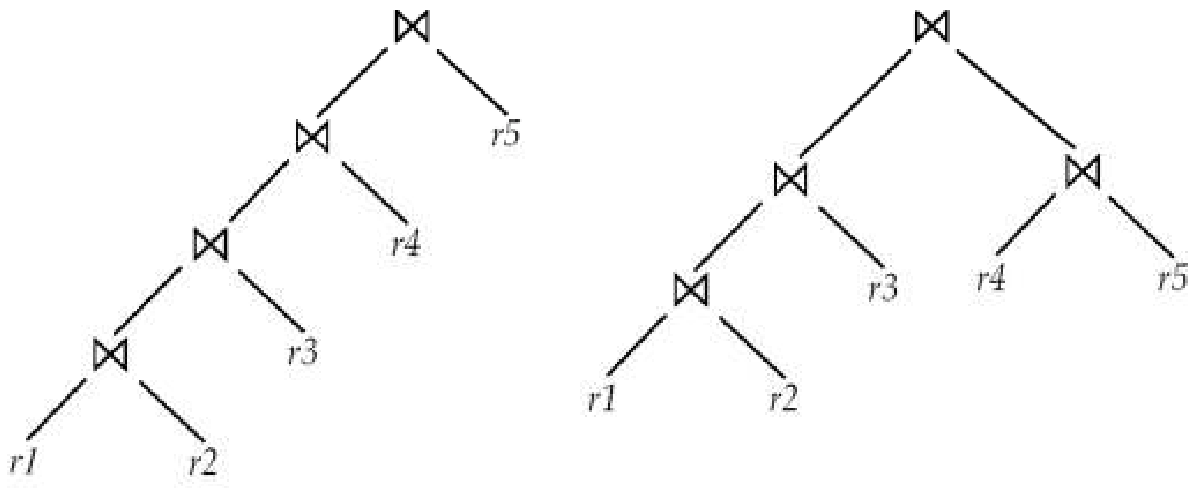
    for every non-empty subset *S'* of *S*
  - Recursively compute costs of best join orders for subsets *S'* and *S* \ *S'* to find the cost of each plan. Choose the cheapest of the $2^n - 2$ alternatives
  - Base case for recursion: single relation access plan
    - Apply all selections on R_i using best choice of indices on R_i
  - When a plan for a subset is computed, store it and reuse it when it is required again, instead of re-computing 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 *S*1 of *S* such that *S*1 ≠ *S*
        P1= findbestplan(*S*1)
        P2= findbestplan(*S* - *S*1)
        A = best algorithm for joining results of *P*1 and *P*2
        cost = *P*1.*cost* + *P*2.*cost* + cost of *A*
        if *cost* < *bestplan*[*S*].*cost*
            bestplan[*S*].*cost* = cost
            bestplan[*S*].*plan* = "execute *P*1.*plan*; execute *P*2.*plan*;
                join results of *P*1 and *P*2 using *A*"
    return *bestplan*[*S*]

Cost of Optimization

- With dynamic programming time complexity of optimization is $O(3^n)$.
  - With n = 10, this number is 59000 instead of 17.6 billion!
- Space complexity is $O(2^n)$
- Better time performance when considering only left-deep tree $O(n\ 2^n)$ Space complexity remains at $O(2^n)$ (heuristic approach)



(a) Left-deep join tree (b) Non-left-deep join tree

- Cost-based optimization is expensive, but worthwhile for queries on large datasets (typical queries have small n, generally < 10)

Cost Based Optimization with Equivalence Rules

- **Physical equivalence rules** equates logical operations (e.g., join) to physical ones (i.e., implementations – e.g., nested-loop join, merge join)
  - Relational algebra expression are converted into QEP with implementation details
- Efficient optimizer based on equivalence rules depends on
  - A space efficient representation of expressions which avoids making multiple copies of sub-expressions
  - Efficient techniques for detecting duplicate derivations of expressions
  - A form of dynamic programming, which stores the best plan for a sub-expression the first time it is optimized, and reuses in on repeated optimization calls on same sub-expression
  - Cost-based pruning techniques that avoid generating all plans (greedy, heuristics, dynamic programming/optimal substructure property)

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
  - Only consider left-deep join orders (particularly suited for pipelining as only one input has to be pipelined, the other is a relation)

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
    - Starting from each of n starting points. Pick best among these

End of Chapter