COL 362 & COL 632

Query Optimization

31 Mar 2023

Cost Based Query Optimization

Generate **logically equivalent expressions** using equivalence rules → Annotate resultant expressions to get **alternative query plans** → Choose the cheapest plan based **on estimated cost**

- Estimation of plan cost based on
  - Statistical information about relations
  - Statistics estimation for intermediate results
  - Cost formulae for algorithms, computed using statistics

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
- 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.
- 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.

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

2. Selection operations are commutative.

   $$\sigma_{\theta_1}\left(\sigma_{\theta_2}(E)\right) \equiv \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

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

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

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

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

   a. $\sigma_\theta(E_1 \times E_2) \equiv E_1 \bowtie_\theta E_2$

   b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) \equiv E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

Many more …. Refer to the textbook

$r_1 \bowtie r_3 \bowtie r_2$

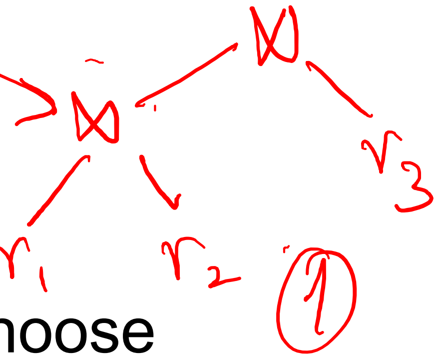
$(r_1 \bowtie (r_2) \bowtie r_3)$

Join Ordering

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

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

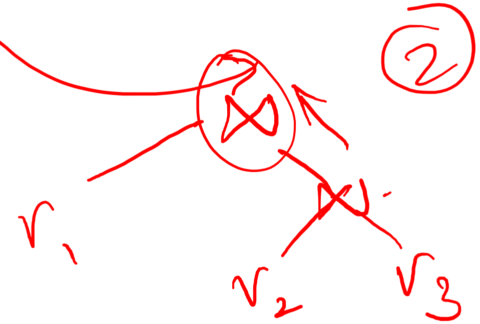
(Join Associativity)



- 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.



Join Ordering

- Consider the following two *equivalent expressions*

$$\Pi_{name,title}\left(\left(\sigma_{deptname="Music"}(instructor) \bowtie teaches\right) \bowtie \Pi_{courseid,title}(course)\right)$$

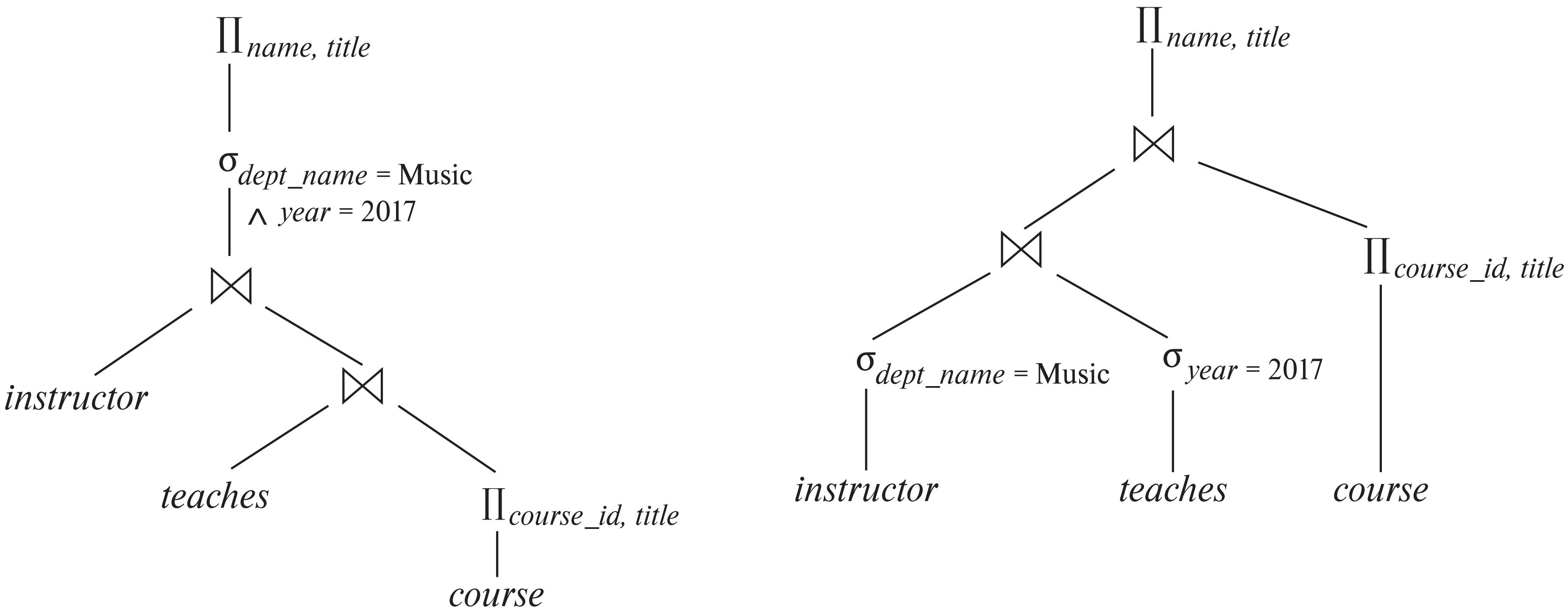
5

$$\Pi_{name,title}\left(\left(teaches \bowtie \Pi_{courseid,title}(course)\right) \bowtie \sigma_{deptname="Music"}(instructor)\right)$$

0

- Which is better?

Join Ordering



(a) Initial expression tree

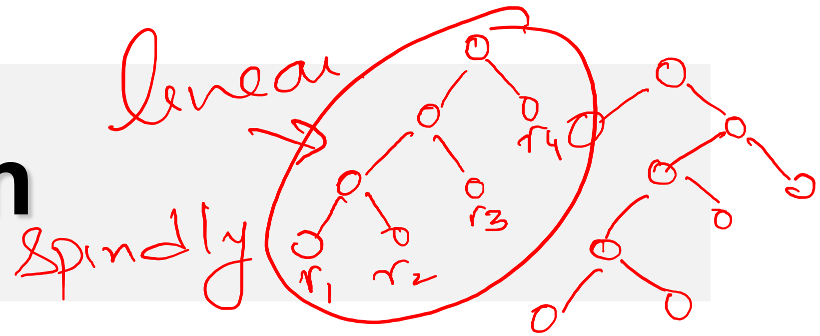
(b) Tree after multiple transformations

Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions
  Repeat
  - apply **all applicable** equivalence rules on every subexpression 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
  1. Optimized plan generation based on transformation rules
  2. Special case approach for queries with only selections, projections and joins

Cost-Based Optimization



- Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$
- The total number of possibilities is given by n-1-th Catalan number (# of binary trees with n leaves)

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!}$$

- No. of join orders of bushy trees (specific kind) : $\boxed{\frac{(2(n-1))!}{(n-1)!}}$

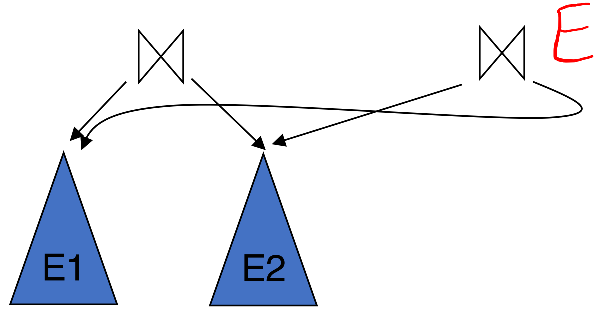
$$\frac{(2n-2)!}{(n-1)!}$$

If n = 7 => 665,280 join orders,

If n = 10 => 176 billion join orders

Transformation Based Optimization

- Space requirements reduced by sharing common sub-expressions:



E_2 E_1

- Time requirements are reduced by not generating all expressions

Dynamic Programming in Optimization

- To find best join tree 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 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 plan for any subset is computed, store it and reuse it when it is required again, instead of recomputing it

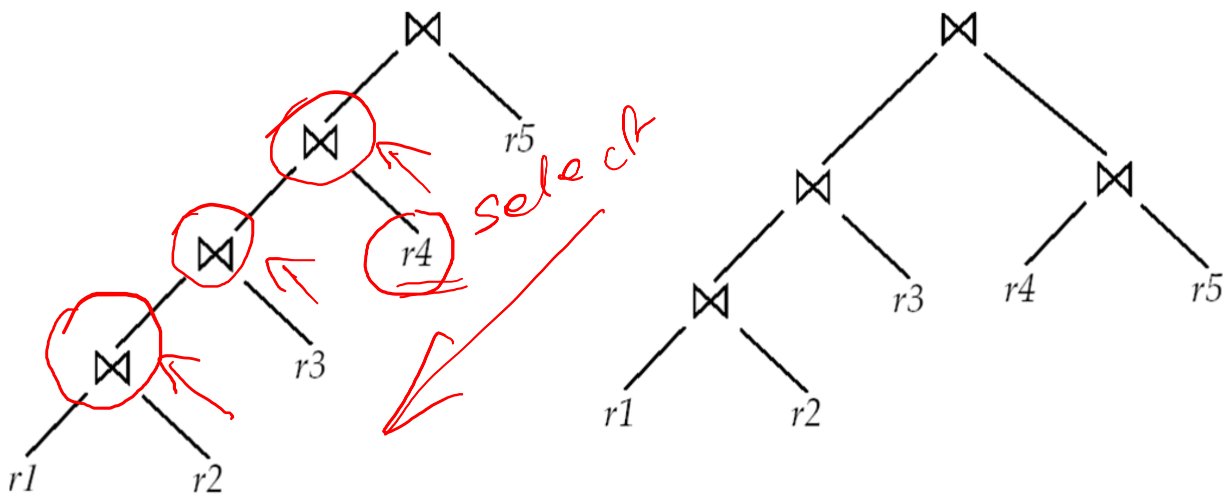
Join Order Optimization Algorithm

```
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]
```

Sum is not always true

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

Cost of Optimization

- With dynamic programming time complexity of optimization with bushy trees is $O(3^n)$.
  - With *n* = 10, this number is 59000 instead of 176 billion!
- Space complexity is $O(2^n)$
- To find best left-deep join tree for a set of *n* relations:
  - Consider *n* alternatives with one relation as right-hand side input and the other relations as left-hand side input.
  - Replace "**for each** non−empty subset $S1$ of S such that $S1 \neq S$"
  - By: **for each** relation r in S let S1 = S − r .
- If only left-deep trees are considered, time complexity of finding best join order is $O(n\ 2^n)$
  - Space complexity remains at $O(2^n)$

Interesting Sort Orders

- Consider the expression $(r_1 \bowtie r_2) \bowtie r_3$ (with *A* as common attribute)
- An **interesting sort order** is a particular sort order of tuples that could be useful for a later operation
  - Using **merge-join** to compute $r_1 \bowtie r_2$ may be costlier than **hash join** but the *result is sorted on A*
  - Which in turn may make **merge-join with r_3** cheaper, which may reduce cost of join with r_3 and minimizing overall cost
  - Sort order may also be useful for order by and for grouping
- Not sufficient to find the best join order for each subset of the set of *n* given relations
  - must find the best join order for each subset, **for each interesting sort order**
- Simple extension of earlier dynamic programming algorithms
  - Usually, number of interesting orders is small and doesn't affect time/space complexity significantly

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.