Query Optimization

Chapter 13

What we want to cover today

- Overview of query optimization
- Generating equivalent expressions
- Cost estimation

Chapter 13 – Query Optimization

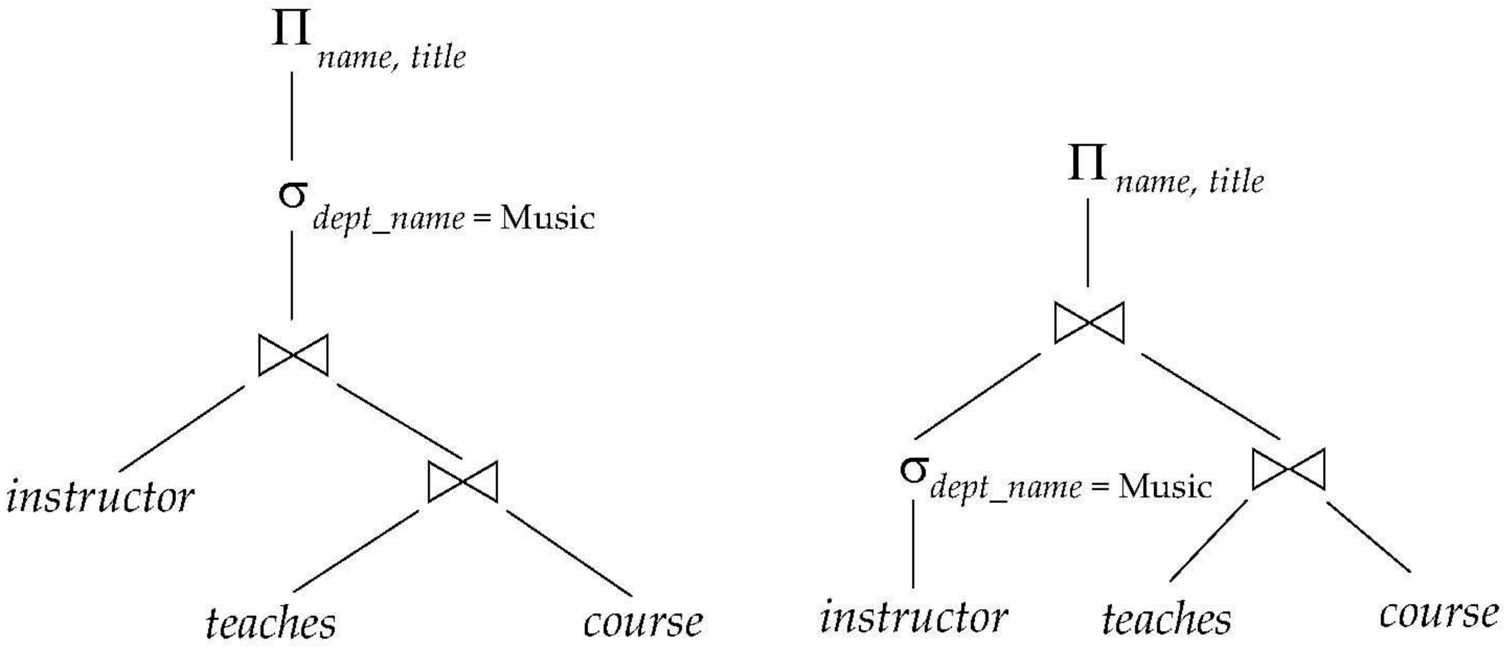
OVERVIEW

Query Optimization

- **Evaluation plan** is a combination of operations to execute a user query
- **Query optimization** is the process of selecting most efficient evaluation plan for a query
  - Generates alternative plans and picks the cheapest
  - There can be a large number of alternatives
  - Exhaustive search often not feasible

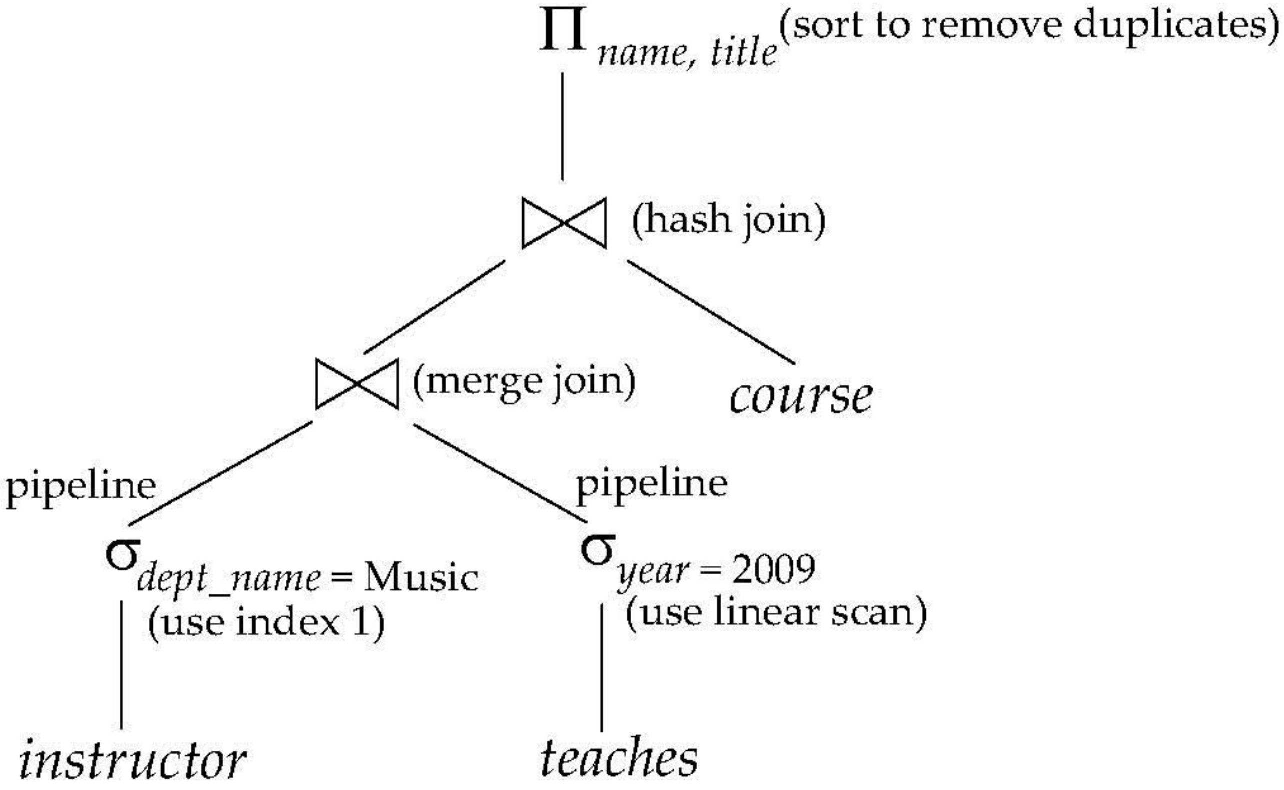
Steps in Query Opimization

Generate logically equivalent expressions



Steps in Query Optimization (Cont.)

Annotate expressions with methods to generate alternative evaluation plans



Steps in Query Optimization (Cont.)

Estimate costs of alternative evaluation plans and choose the cheapest

- Estimation of plan cost based on:
  - Statistical information about relations.
    - Eg. number of tuples, number of distinct values for an attribute
  - Statistics estimation for intermediate results
    - Used to compute cost of complex expressions
  - Cost formulae for algorithms,
    - Estimates computed using statistics

Chapter 13 – Query Optimization

GENERATING EQUIVALENT EXPRESSIONS

Equivalence Rules

- 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 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

This is very expensive in space and time!

Heuristics

- 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.

Heuristics (cont)

- Many optimizers consider 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

Example

instructor (ID, name, dept_name, salary)

teaches (ID, course_id, sec_id, semester, year)

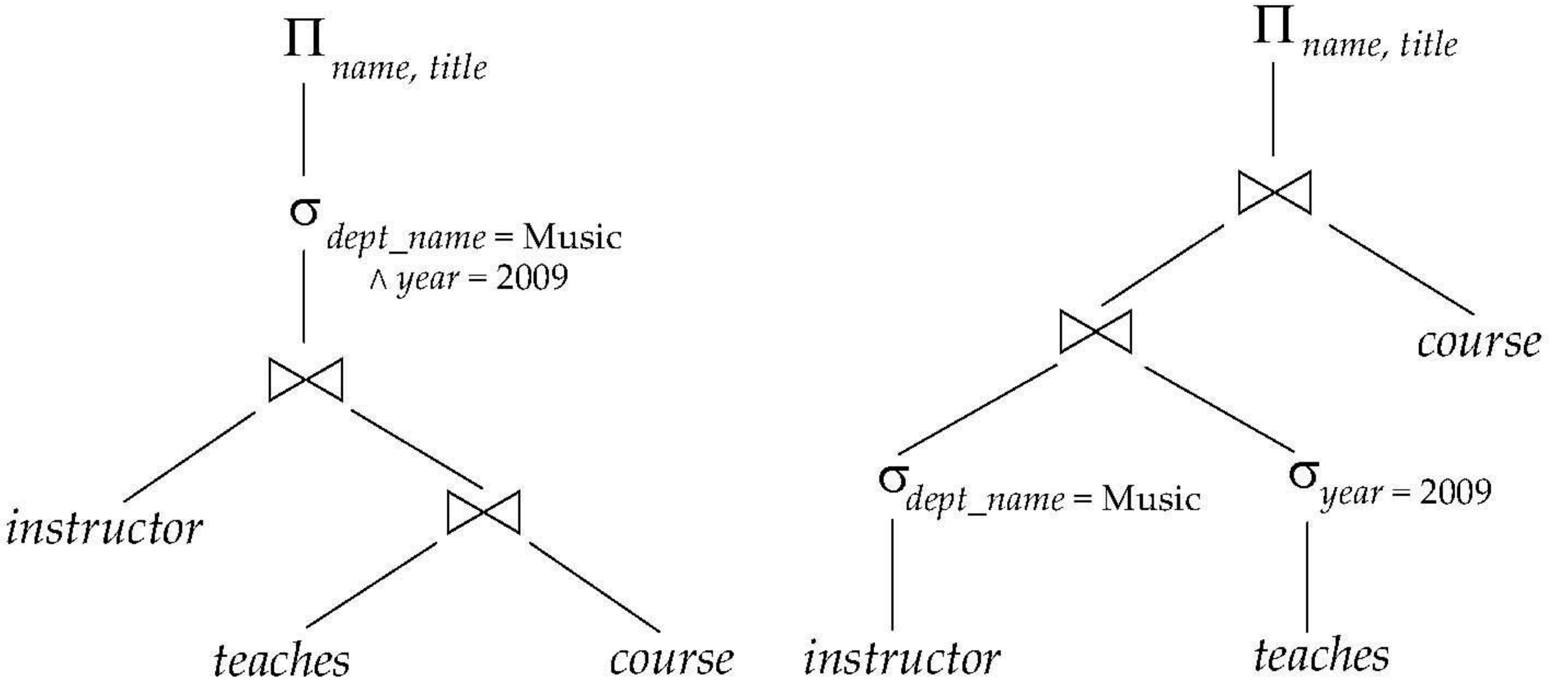
course (course_id, title, dept_name, credits)

Example (cont)

- Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught
  - $\Pi_{name,\ title}(\sigma_{dept\_name = \text{"Music"} \wedge year = 2009}$ $(instructor \bowtie (teaches \bowtie course)))$
- Natural joins are associative (smaller join first):
  - $\Pi_{name,\ title}(\sigma_{dept\_name = \text{"Music"} \wedge year = 2009}$ $((instructor \bowtie teaches) \bowtie course))$
- Perform selections early:

$$\sigma_{dept\_name = \text{"Music"}}(instructor) \bowtie \sigma_{year = 2009}(teaches)$$

Example (Cont.)



(a) Initial expression tree (b) Tree after multiple transformations

COST ESTIMATION

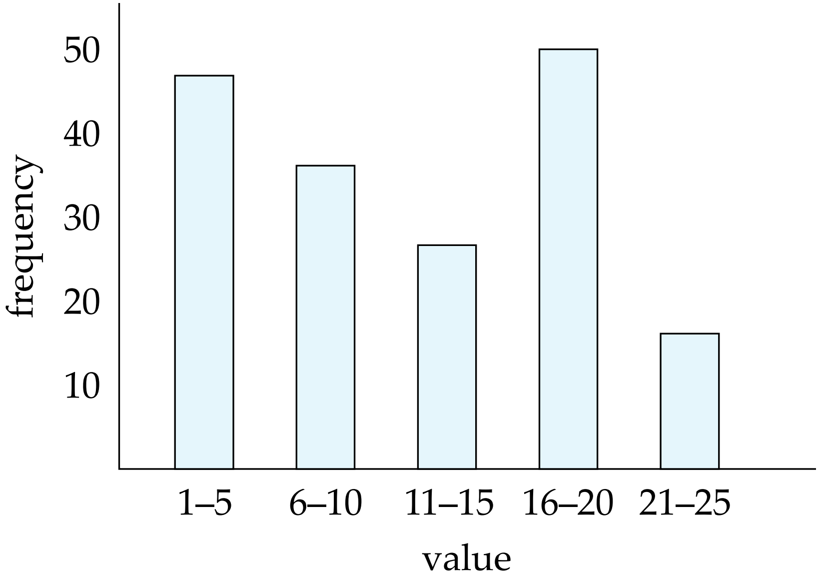
Catalog Information for Cost Estimation

- 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.
- 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 number of tuples in $\prod_A(r)$.
- If tuples of r are stored together physically in a file, then:

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

Catalog Information (Cont)

- Most DBMSs maintain a histogram of distribution of values for an attribute rather than just *V(A, r)*
- **Equi-width** histograms
- **Equi-depth** histograms
- Eg. histogram on attribute *age* of relation *person*



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 $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$.
- There are $(2(n - 1))!/(n - 1)!$ different join orders for above expression.
- 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.

Dynamic Programming in Optimization

- To find best join tree for a set *S* of *n* relations:
  - Consider all possible plans $S_1 \bowtie (S - S_1)$ 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 algorithm
  - 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 $\neq \infty$)
            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 $S1 \neq S$
            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]
```

* Modifications needed to allow indexed nested loops joins on relations that have selections (see book)

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.
  - Modify optimization algorithm:
    - 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 $O(2^n)$
- Cost-based optimization is expensive, but worthwhile for queries on large datasets (typical queries have small n, generally < 10)