Intro to DB

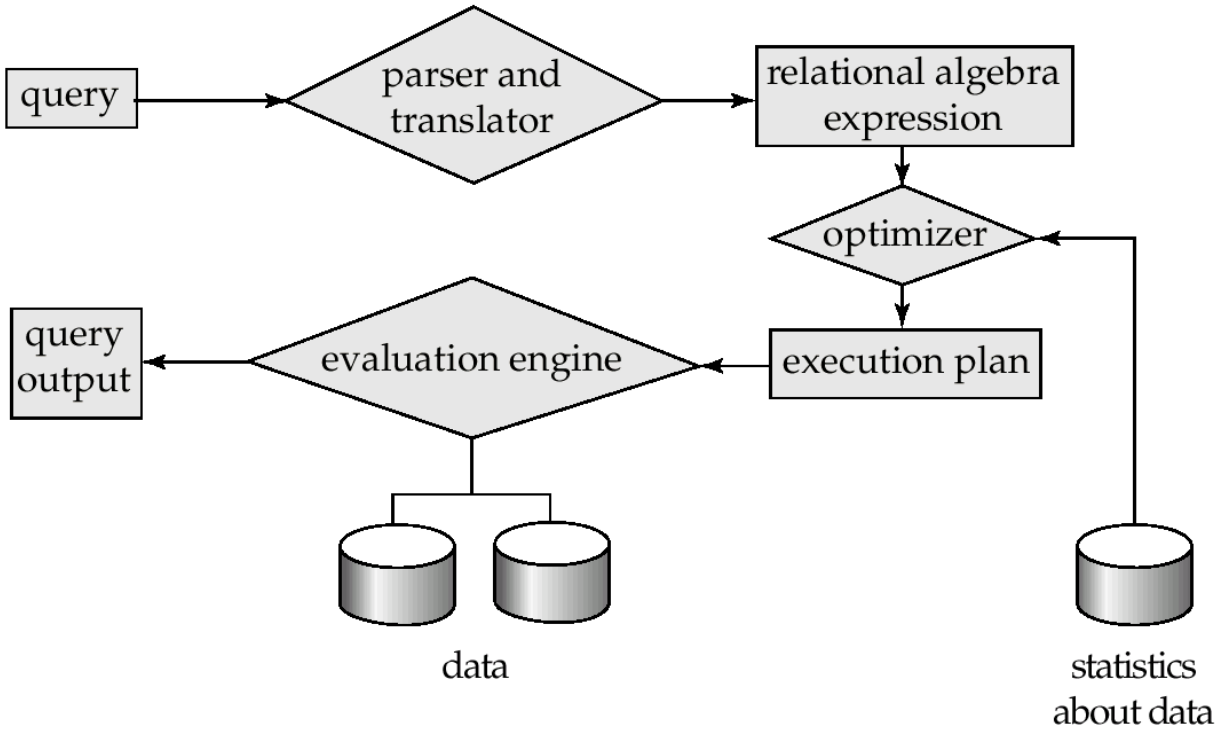
CHAPTER 13
QUERY OPTIMIZATION

Chapter 13: Query Optimization

- Overview
- Transformation of Relational Expressions
- Estimating Statistics of Expression Results
- Choice of Evaluation Plans
- Materialized Views
- Advanced Topics

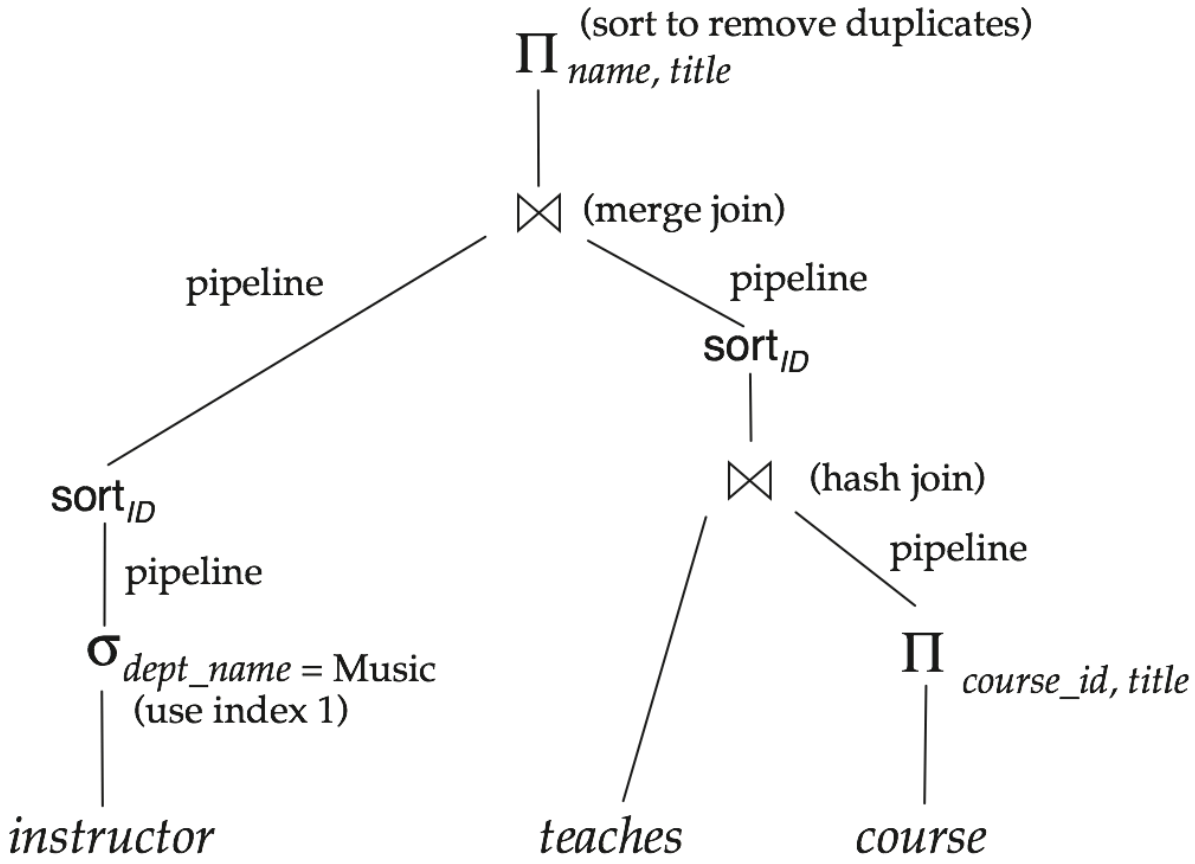
Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Evaluation Plan

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



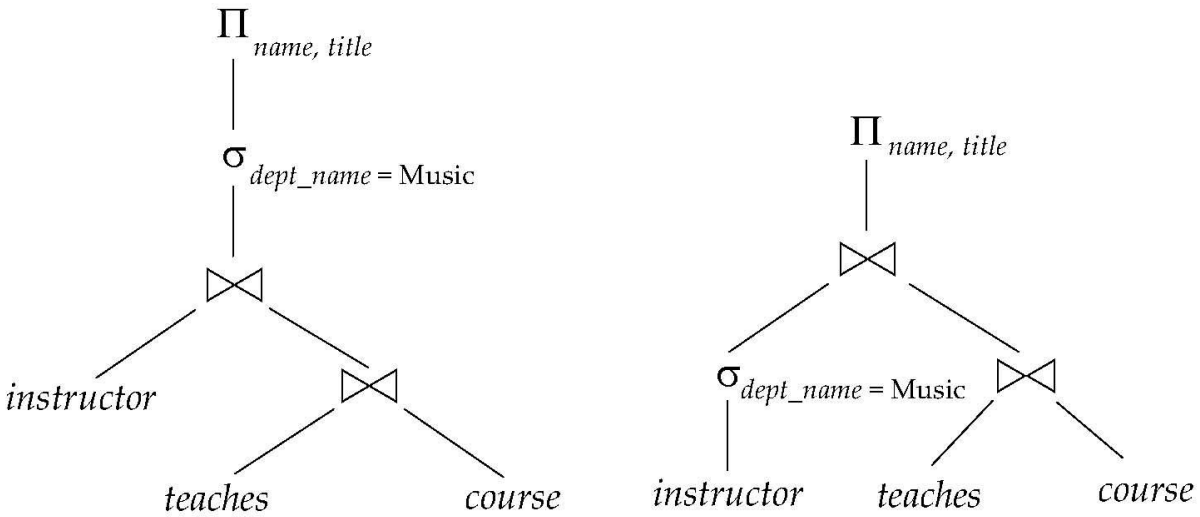
Query Optimization

- Equivalence of Expressions

  Given a DB schema *S*, a query *Q* on *S* is *equivalent* to another query *Q'* on *S*, if the answer sets of *Q* and *Q'* are the same in *any* instances of the DB.

$\Pi_{name,\ title}(\sigma_{dept=\text{“Music”}}(\ instructor \bowtie (teaches \bowtie course\ )))$ vs

$\Pi_{name,\ title}(\ (\sigma_{dept=\text{“Music”}}(instructor)) \bowtie teaches \bowtie course\ )$



Generation of Evaluation Plan

- Generation of query-evaluation plans for an expression involves several steps:
  1. Generating logically equivalent expressions
     - Use *equivalence rules* to transform an expression into an equivalent one.
  2. Annotating resulting expressions to get alternative query plans
  3. Choosing the cheapest plan based on *estimated cost*
-

- Query optimization is the process of *selecting* the *most efficient query evaluation plan* for a given query
  - A plan with the smallest estimated cost of execution
  - In a centralized system, disk access is the predominant cost

Equivalence Rules

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

2. Selection operations are commutative.

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

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

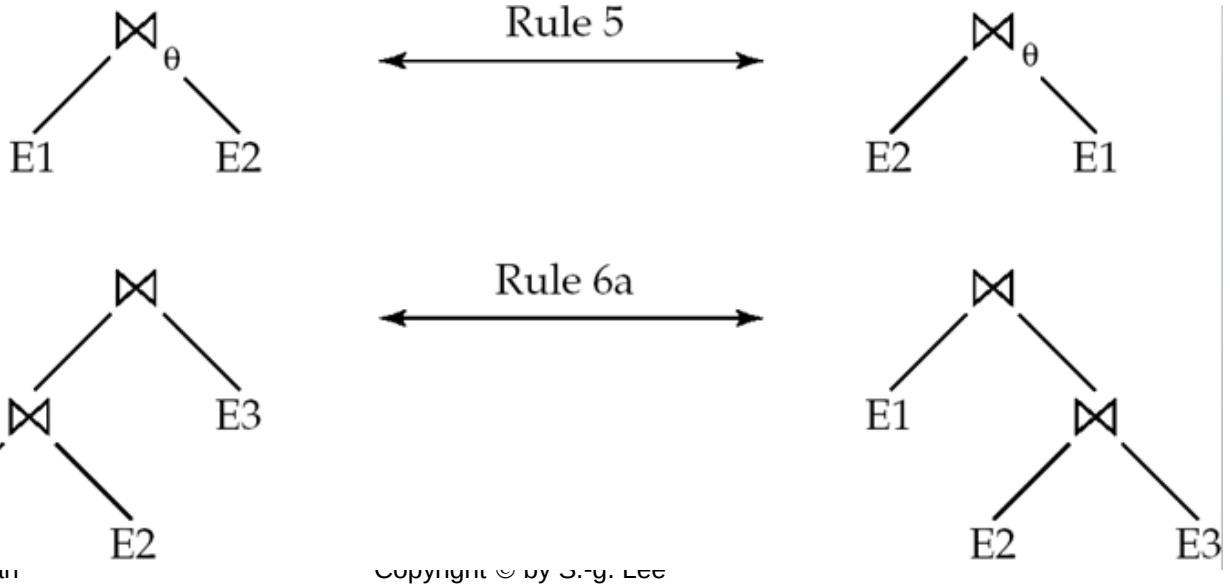
Equivalence Rules (cont.)

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

6. (a) Natural join operations are associative:

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

where θ_2 involves attributes from only E_2 and E_3.

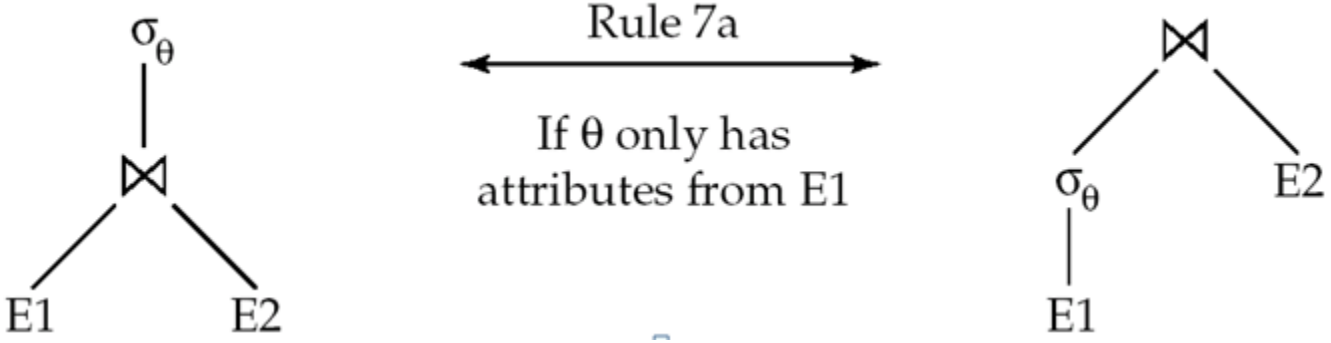


Equivalence Rules (cont.)

7. Selection operation distributes over theta join

   (a) when all the attributes in θ_0 involve only the attributes of one of the expressions (E_1) being joined.

   (b) when θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.



Transformation – Example 1

- Query:

  Find the names of all instructors in the Music department, along with the titles of the courses that they teach.

$$\Pi_{name,\ title}(\sigma_{dept=\text{“Music”}}\ (instructor \bowtie (teaches \bowtie course)))$$

- Transformation using rule 7a.

- Performing the selection as early as possible reduces the size of the relation to be joined.

Transformation – Example 2

- 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.

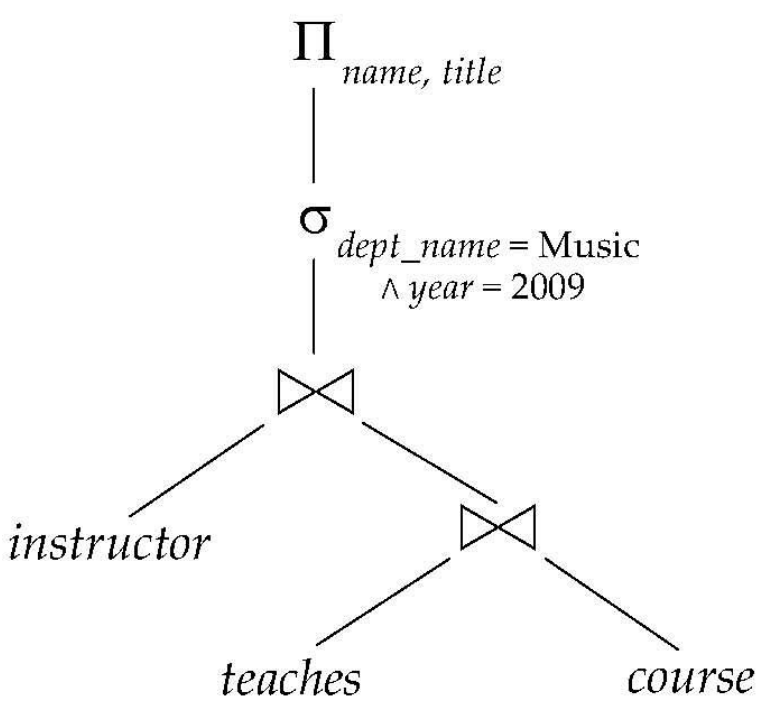
$$\Pi_{name,\ title}(\sigma_{dept=\text{“Music”} \wedge year\ =\ 2009} (instructor \bowtie (teaches \bowtie \Pi_{c\_id,\ title}\ (course))))$$

- Using join associativity (Rule 6a):

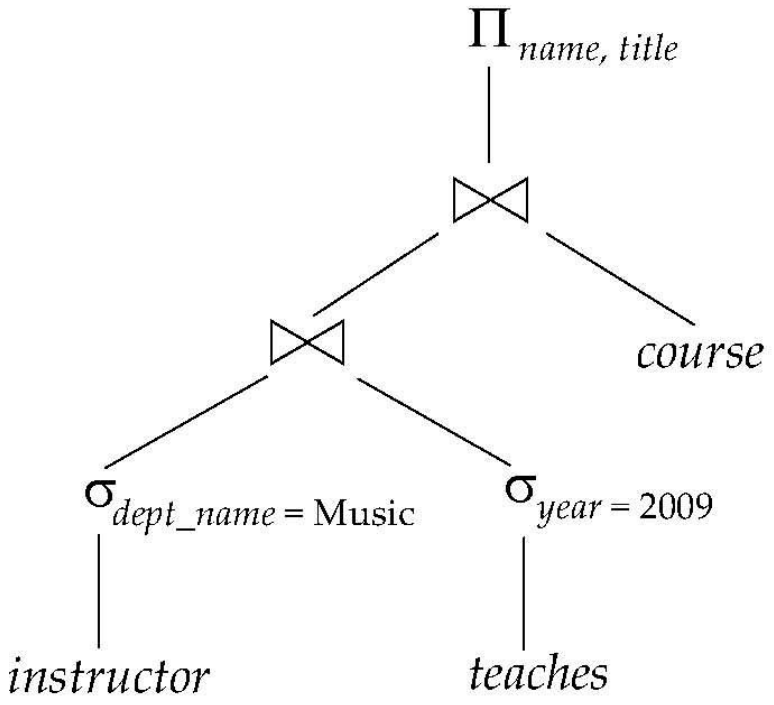
$$\Pi_{name,\ title}(\sigma_{dept=\text{“Music”} \wedge year\ =\ 2009} ((instructor \bowtie teaches) \bowtie \Pi_{c\_id,\ title}\ (course))))$$

- Push selection in (Rules 7a & 7b):

Transformation – Example 2 (cont.)



(a) Initial expression tree



(b) Tree after multiple transformations

Join Ordering

- $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$ (Rule 6a)
-
- Example

$$\Pi_{name,\ title}((\sigma_{dept=\text{“Music”}}(instructor) \bowtie teaches) \bowtie \Pi_{c\_id,\ title}(course))$$

  - $\sigma_{dept=\text{“Music”}}(instructor) \bowtie teaches$

    vs
  - $teaches \bowtie \Pi_{c\_id,\ title}(course)$

Cost Estimation

- Cost of computing each operator is 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
  -
  - To do so, we require additional statistics
    - E.g. number of distinct values for an attribute
- Recall that
  - Typically disk access is the predominant cost.

- We do not include cost of writing the final output to disk
- We refer to the cost estimate of algorithm *A* as E_A

Statistics for Cost Estimation

- n_r: number of tuples in a relation r
- b_r: number of blocks containing tuples of r

  $b_r = \lceil n_r / f_r \rceil$, if tuples of r are stored together physically in a file
- 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 size of $\prod_A(r)$
- $SC(A, r)$: selection cardinality of attribute A of relation r; average number of records that satisfy equality on A

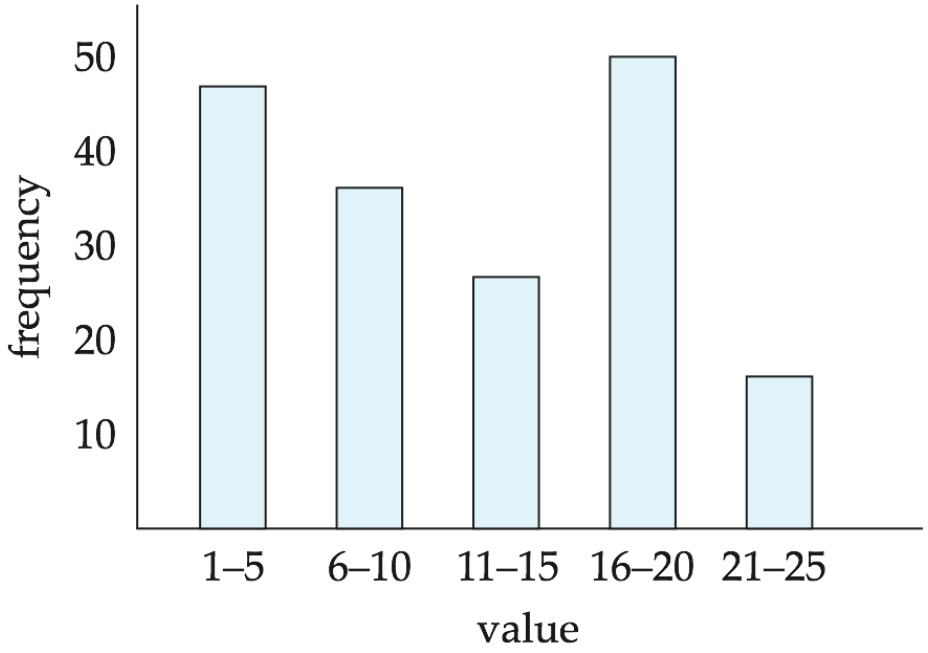
Catalog Information about Indices

- f_i: average fan-out of internal nodes of index i,

  for tree-structured indices such as B+-trees
- HT_i: number of levels in index i — i.e., the height of i
  - For a balanced tree index (such as B+-tree) on attribute A of relation r, $HT_i = \lceil \log_{fi} (V(A,r)) \rceil$.
  - For a hash index, HT_i is 1.
  - LB_i: number of lowest-level index blocks in i — i.e, the number of blocks at the leaf level of the index.

Histograms

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



- **Equi-width** histograms
- **Equi-depth** histograms

Selection Size Estimation

- Equality selection $\sigma_{A=v}(r)$
  - $SC(A, r)$: number of records that will satisfy the selection

    $= n_r \ / \ V(A,r)$ general case

    $= 1$ (if A is key)

  - $\lceil SC(A, r)/f_r \rceil$ — number of blocks that these records will occupy

  - E.g. Binary search cost estimate becomes

$$E_A = \lceil \log_2(b_r) \rceil * (t_T + t_S) \ + \ (\lceil SC(A, r) / f_r \rceil - 1) * t_T$$

Join Operation: Running Example

Running example:

$student \bowtie takes$

Catalog information for join examples:

- $n_{student}$ = 5,000.
- $f_{student}$ = 50, which implies that $b_{student}$ =5000/50 = 100.
- n_{takes} = 10000.
- f_{takes} = 25, which implies that b_{takes} = 10000/25 = 400.
- *V(ID, takes)* = 2500, which implies that on average, each student who has taken a course has taken 4 courses.
  - Attribute *ID* in *takes* is a foreign key referencing *student.*
- *V*(*ID, student*) = 5000 (*primary key!*)

Join Size Estimation

- $r \bowtie s = r \times s$ if $R \cap S = \emptyset$
  - $r \times s$ contains $n_r * n_s$ tuples
  -
- If $R \cap S$ is a key for R
  - then a tuple of s will join with at most one tuple from r
  - therefore, the number of tuples in $r \bowtie s$ is no greater than the number of tuples in s.
  - In the example query *student* $\bowtie$ *takes*
    - *ID* in *takes* is a foreign key of *student*
    - hence, the result has (exactly) n_{takes} = 10,000 tuples

Join Size Estimation (cont.)

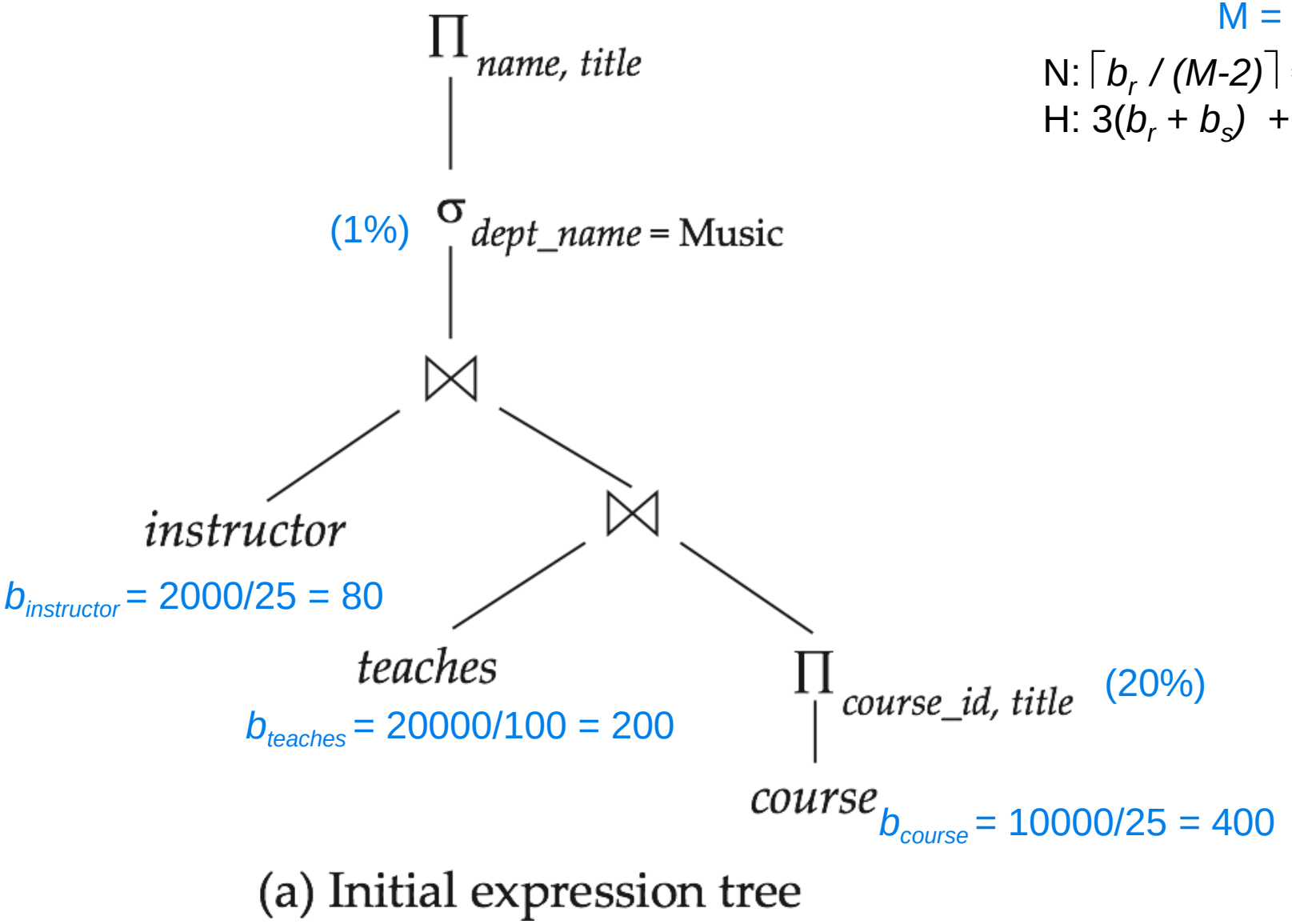
- If $R \cap S = \{A\}$ is not a key for R nor S
  - If every tuple t in r produces tuples in $r \bowtie s$:
  - If the reverse is true:
  - The lower of these two estimates is probably the more accurate one.
  - Compute the size estimates for *student* ⋈ *takes* without using information about foreign keys:
    - *V(ID, takes)* = 2,500
      V(ID, student) = 5,000
    - The two estimates are 5,000 * 10,000/2,500 = 20,000 and
      5,000 * 10,000/5,000 = 10,000

Cost Estimation of Expressions

M = 22

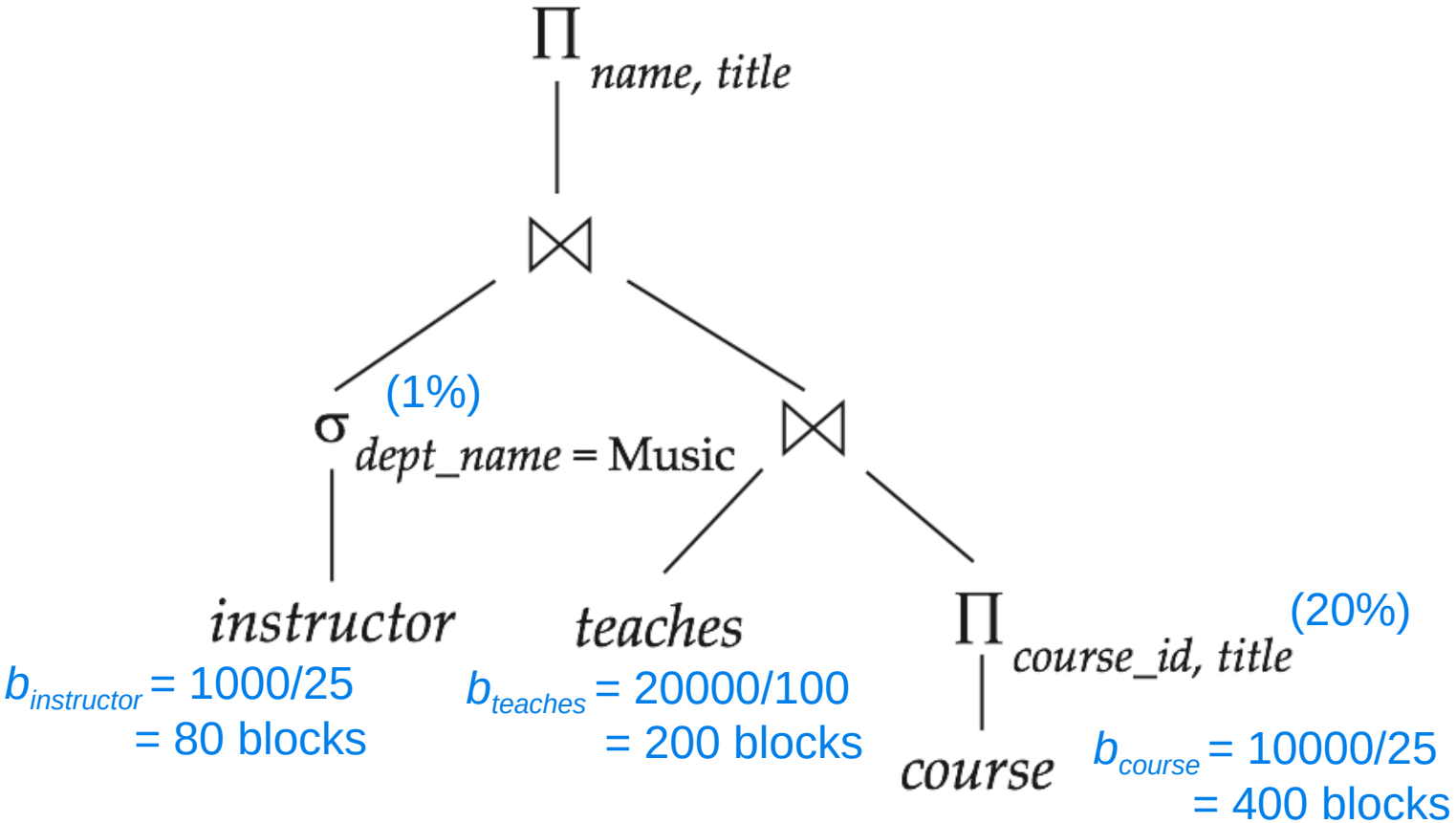
N: $\lceil b_r / (M-2) \rceil * b_s + b_r$

H: $3(b_r + b_s) + 4*n_h$



(a) Initial expression tree

Cost Estimation of Expressions



(b) Transformed expression tree

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.
    - 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

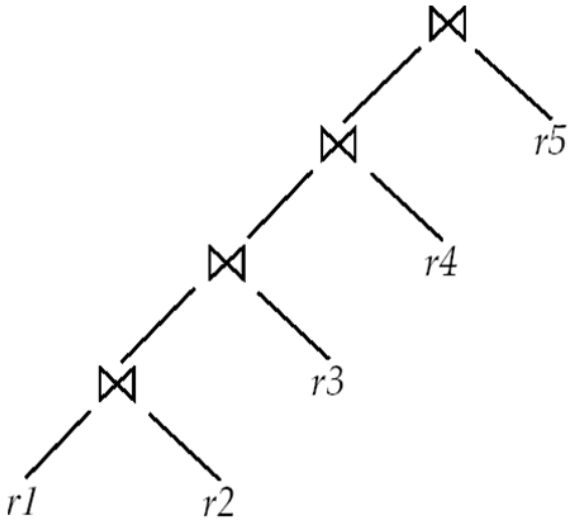
  for above expression.
  - with n = 7, the number is 665280
  - with n = 10, the number is > 176 billion!
- Can reduce search space using *dynamic programming*
  - Using dynamic programming, the least-cost join order for any subset of $\{r_1, r_2, \ldots r_n\}$ is computed only once and stored for future use.
  - Time complexity of optimization with bushy trees is $O(3^n)$
    - Space complexity: $O(2^n)$
  - With n = 10, this number is 59,000 (instead of 176 billion!)

Heuristic Optimization

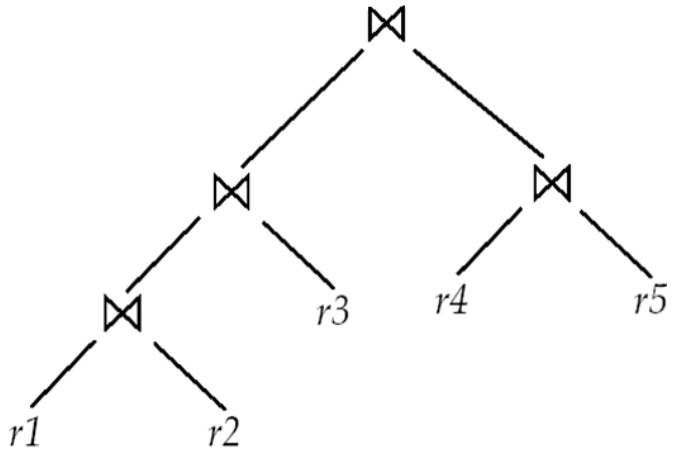
- Cost-based optimization is expensive, even with dynamic programming
  - Search space grows exponentially!
- Heuristic optimization
  - make transformations based on a set of rules that typically (but not in all cases) improve execution performance:
    - Perform selection early (reduces the number of tuples)
    - Perform projection early (reduces the number of attributes)
    - Perform most restrictive selection and join operations before other similar operations
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

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.
- If only left-deep trees are considered, time complexity of finding best join order is $O(n\ 2^n)$
  - n=10 => 10,000 (cf. 59,000 or 176 billion)
  - Space complexity remains at $O(2^n)$



(a) Left-deep join tree



(b) Non-left-deep join tree

END OF CHAPTER 13