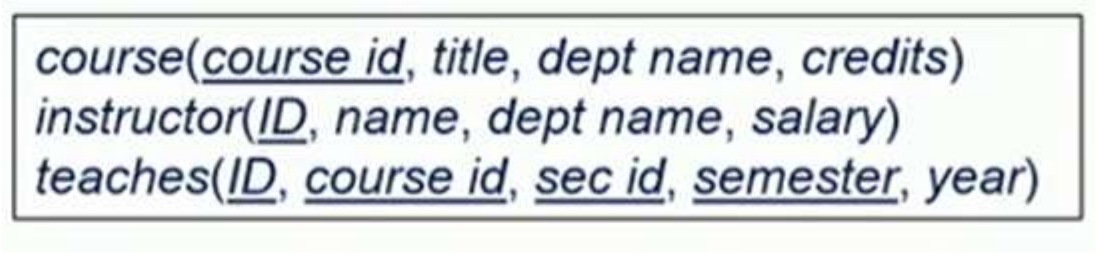
Query Optimization

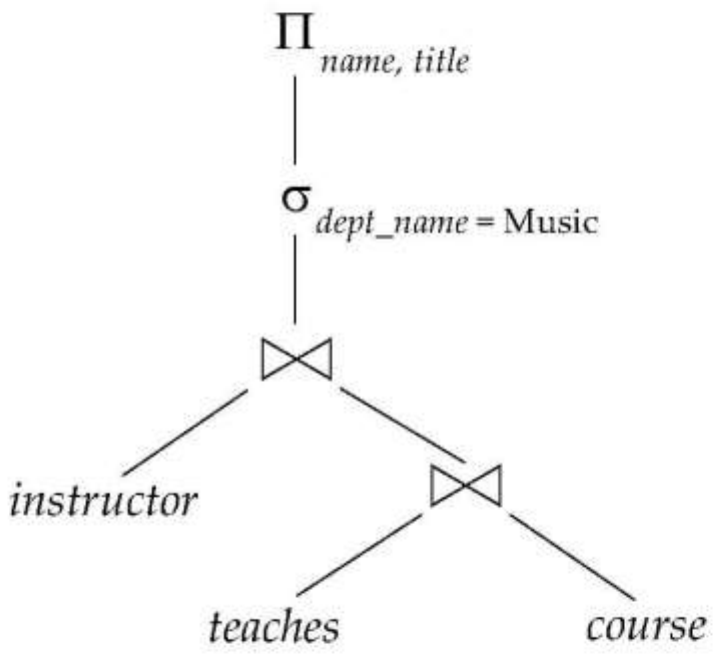
Query Optimization

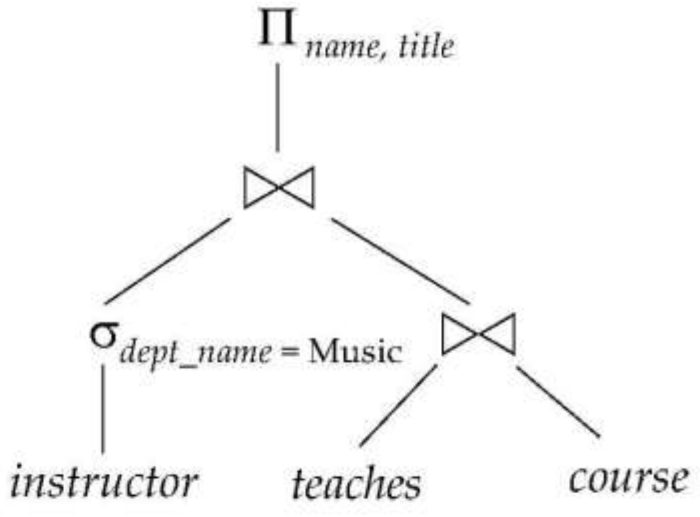
- Introduction
- Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- Statistical Information for Cost Estimation
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
- Materialized views



course(*course id*, *title*, *dept name*, *credits*)
instructor(*ID*, *name*, *dept name*, *salary*)
teaches(*ID*, *course id*, *sec id*, *semester*, *year*)

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

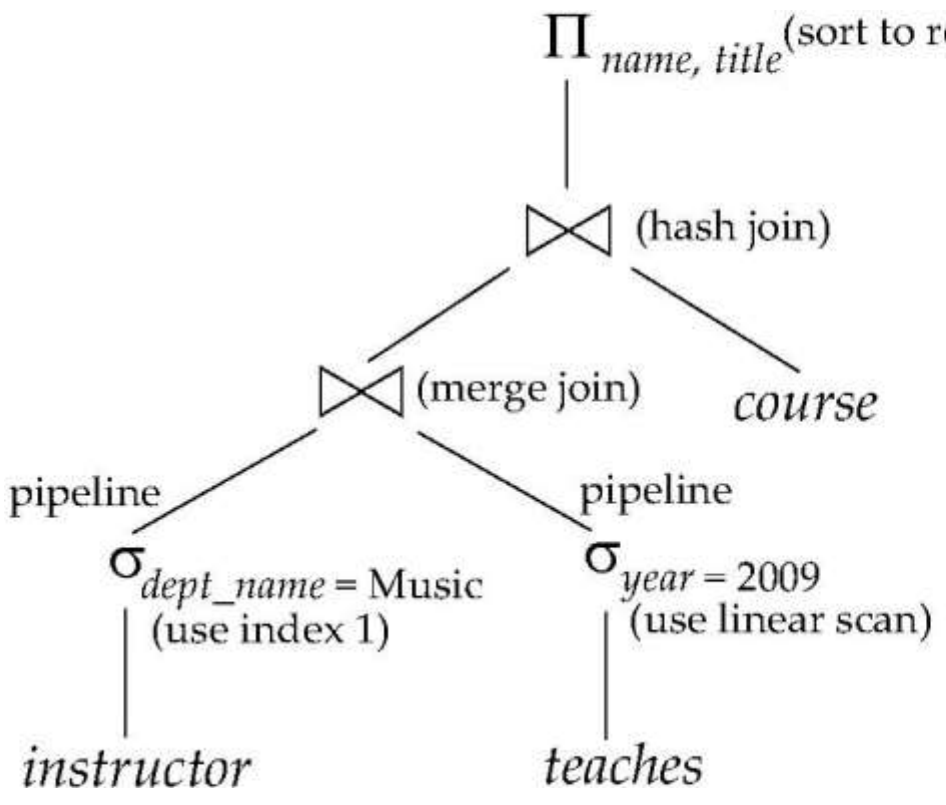




Introduction (Cont.)

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

$\Pi_{name,\ title}(\sigma_{dept\_name= \text{"Music"} \wedge year = 2009}(instructor \bowtie teaches \bowtie course))$


Π name, title (sort to remove duplicates)
⋈ (hash join)
⋈ (merge join)
course
pipeline
pipeline
σ dept_name = Music (use index 1)
σ year = 2009 (use linear scan)
instructor
teaches

course(<u>course id</u>, title, dept name, credits)
instructor(<u>ID</u>, name, dept name, salary)
teaches(<u>ID</u>, <u>course id</u>, <u>sec id</u>, <u>semester</u>, year)

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

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

Generating Equivalent Expressions

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
  - we don't care if they generate different results on databases that violate integrity constraints
- 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.

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

2. Selection operations are commutative.

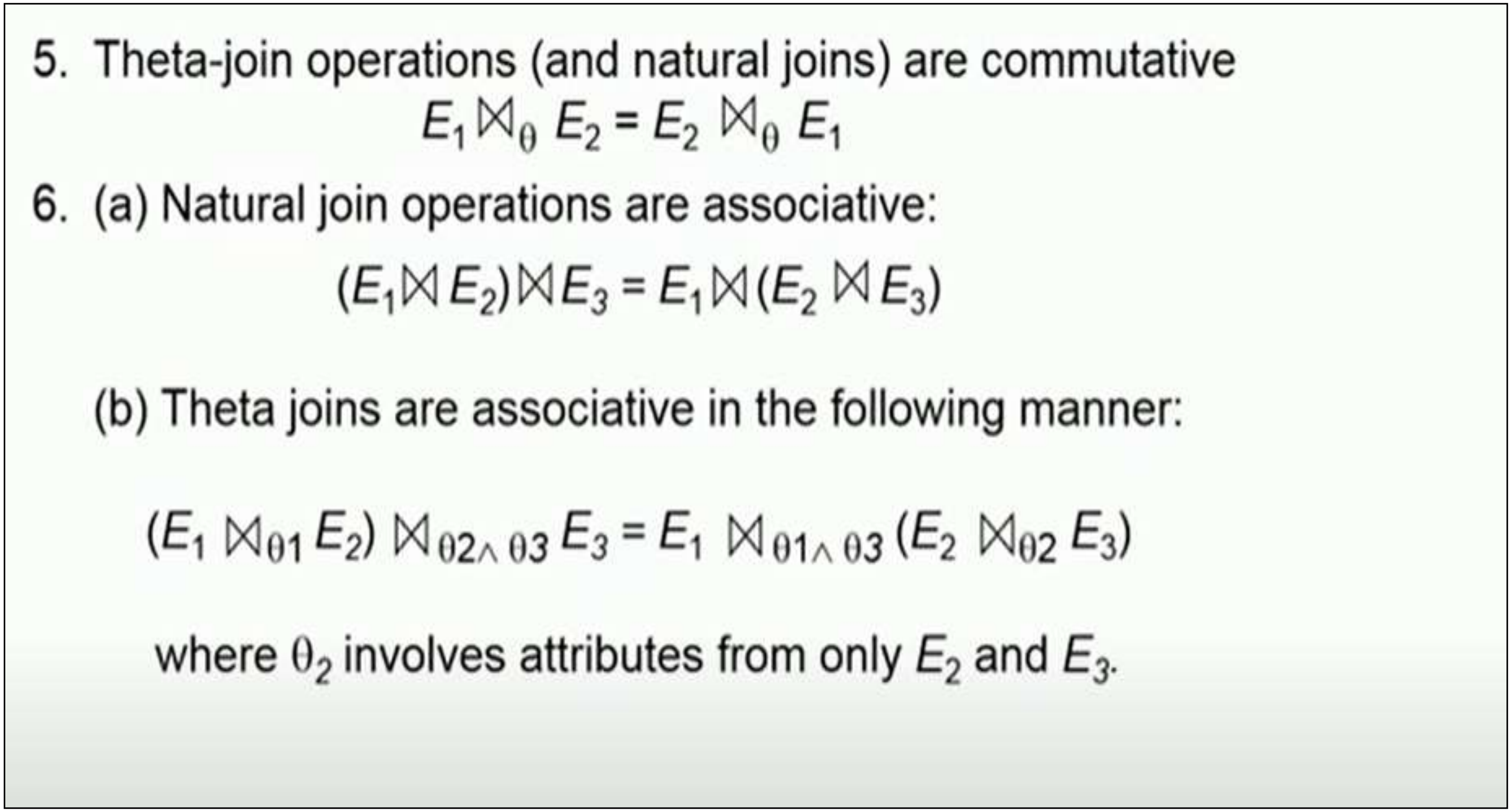
$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

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. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   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 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:

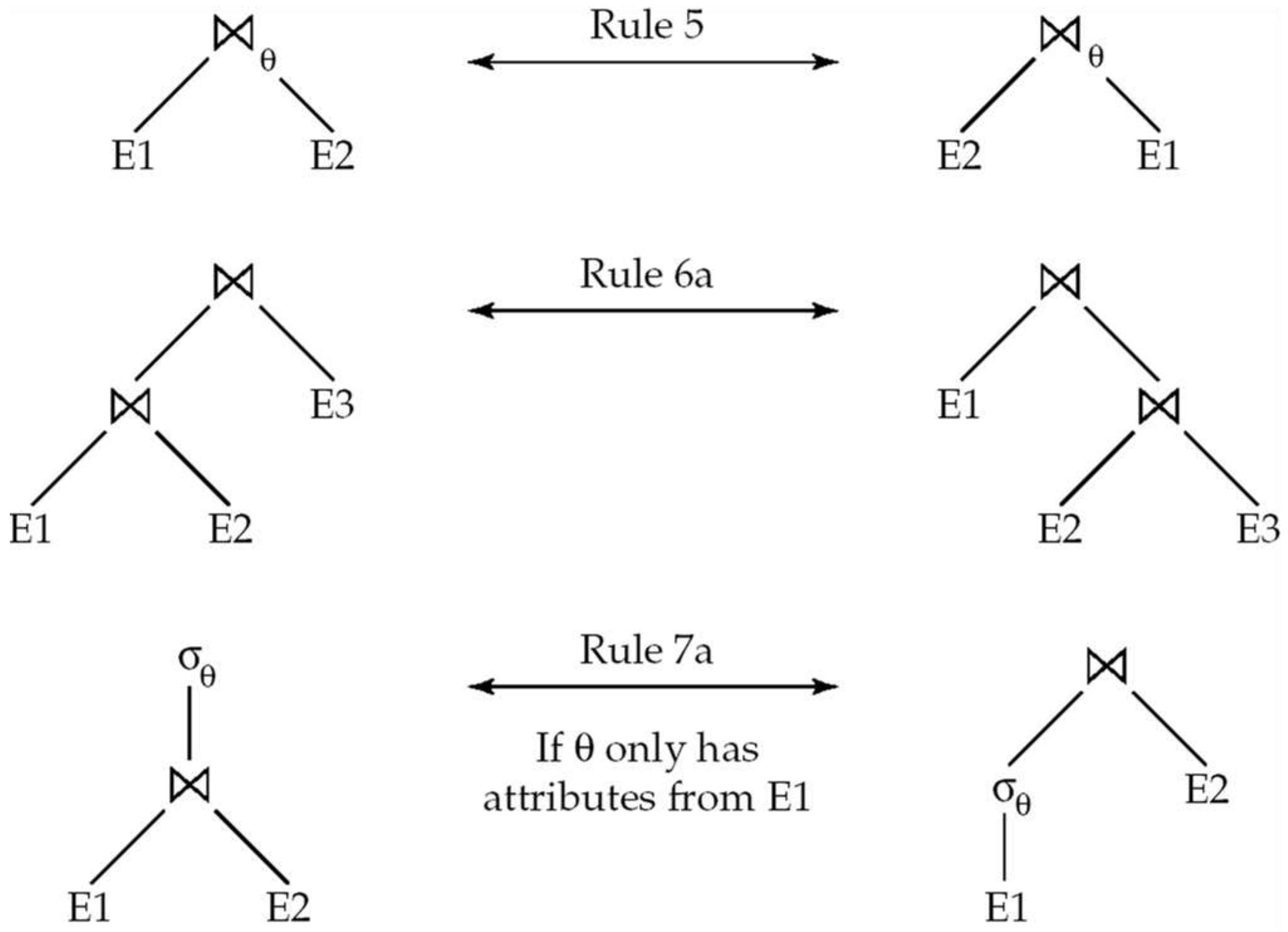
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(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 θ_2 involves attributes from only E_2 and 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 θ_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 θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

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

Equivalence Rules (Cont.)

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

   (a) if θ 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 θ, but are not in $L_1 \cup L_2$, and
   - let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$.

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

)

Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

   $E_1 \cup E_2 = E_2 \cup E_1$

   $E_1 \cap E_2 = E_2 \cap E_1$

   9. (set difference is not commutative).

10. Set union and intersection are associative.

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

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

9. 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

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

Exercise

- Create equivalence rules involving
  - The group by/aggregation operation
  - Left outer join operation

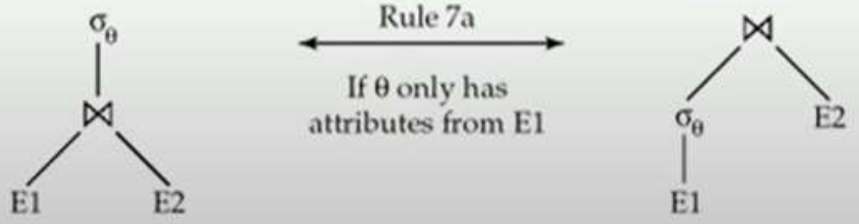
Transformation Example: Pushing Selections

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”}}$ $(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}(course))))$
- Transformation using rule 7a.
  - $\Pi_{name,\ title}((\sigma_{dept\_name=\text{“Music”}}(instructor)) \bowtie$ $(teaches \bowtie \Pi_{course\_id,\ title}(course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined.

Transformation Example: Pushing Projections

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{"Music"}}\ (instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course))))$
- Transformation using rule 7a
  - $\Pi_{name,\ title}((\sigma_{dept\_name=\ \text{"Music"}}(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined

course(<u>*course id*</u>, *title*, *dept name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept name*, *salary*)
teaches(<u>*ID*</u>, <u>*course id*</u>, <u>*sec id*</u>, <u>*semester*</u>, <u>*year*</u>)

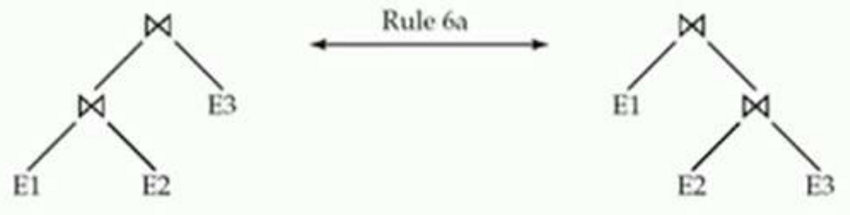


Example with Multiple Transformations

- 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
  - $\Pi_{name,\ title}(\sigma_{dept\_name= \text{“Music”} \wedge year = 2009}$ $(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}(course))))$
- Transformation using join associatively (Rule 6a):
  - $\Pi_{name,\ title}(\sigma_{dept\_name= \text{“Music”} \wedge gear = 2009}$ $((instructor \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course)))$
- Second form provides an opportunity to apply the “perform selections early” rule, resulting in the subexpression

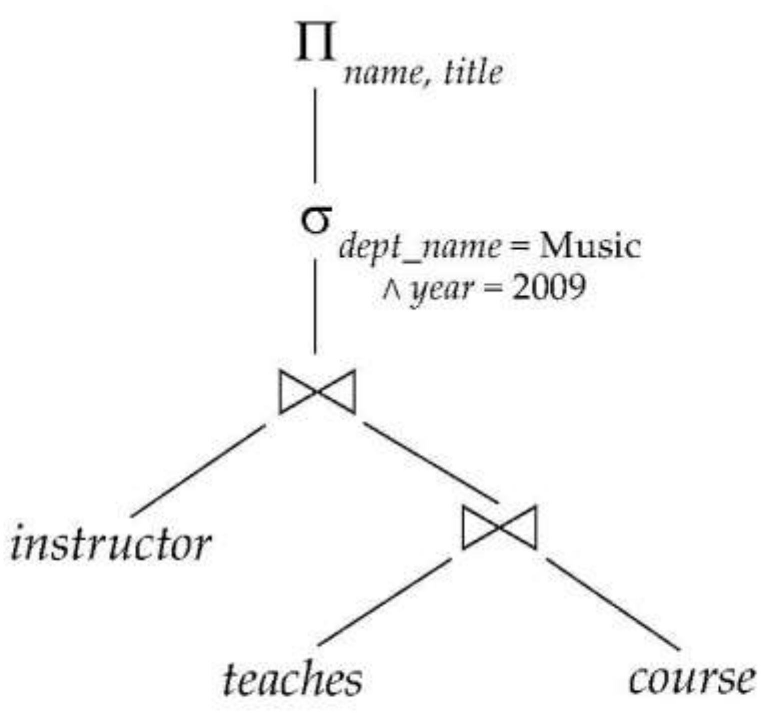
$$\sigma_{dept\_name = \text{“Music”}}(instructor) \bowtie \sigma_{year = 2009}(teaches)$$

course(*course id*, *title*, *dept name*, *credits*)
instructor(*ID*, *name*, *dept name*, *salary*)
teaches(*ID*, *course id*, *sec id*, *semester*, *year*)

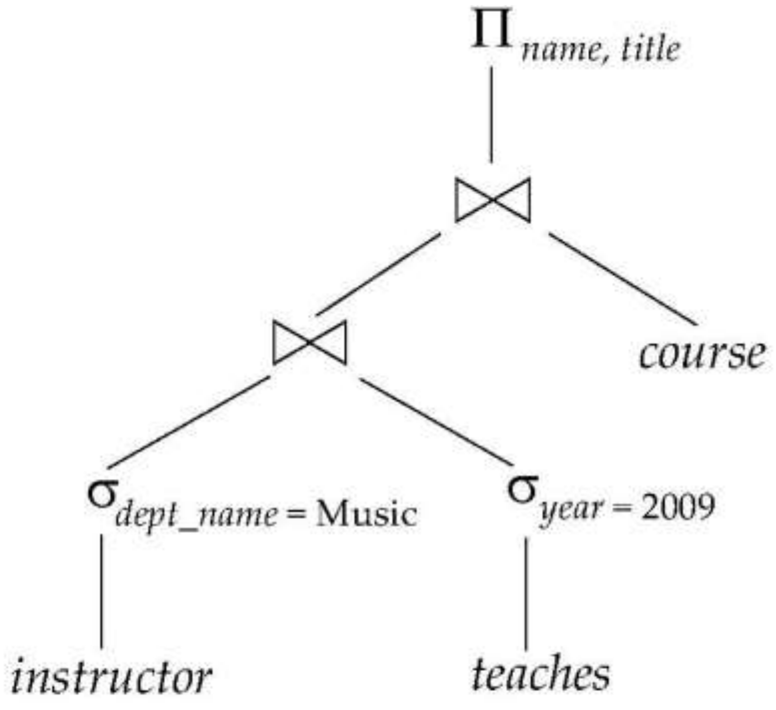


Multiple Transformations (Cont.)

course(*course id*, *title*, *dept name*, *credits*)
instructor(*ID*, *name*, *dept name*, *salary*)
teaches(*ID*, *course id*, *sec id*, *semester*, *year*)

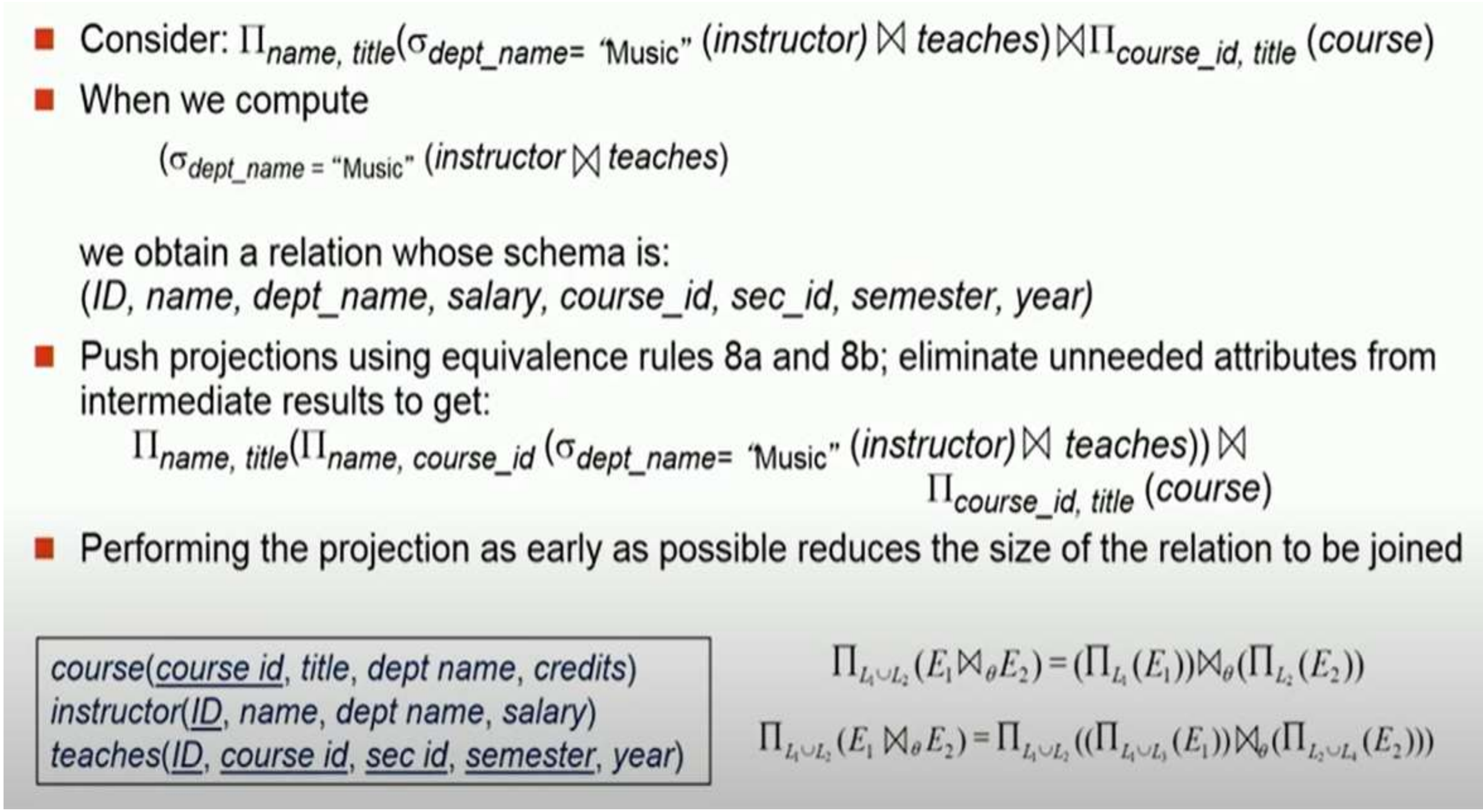


(a) Initial expression tree



(b) Tree after multiple transformations

Transformation Example: Pushing Projections



- Consider: $\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{"Music"}}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course)$
- When we compute

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

  we obtain a relation whose schema is:
  (ID, name, dept_name, salary, course_id, sec_id, semester, year)
- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:

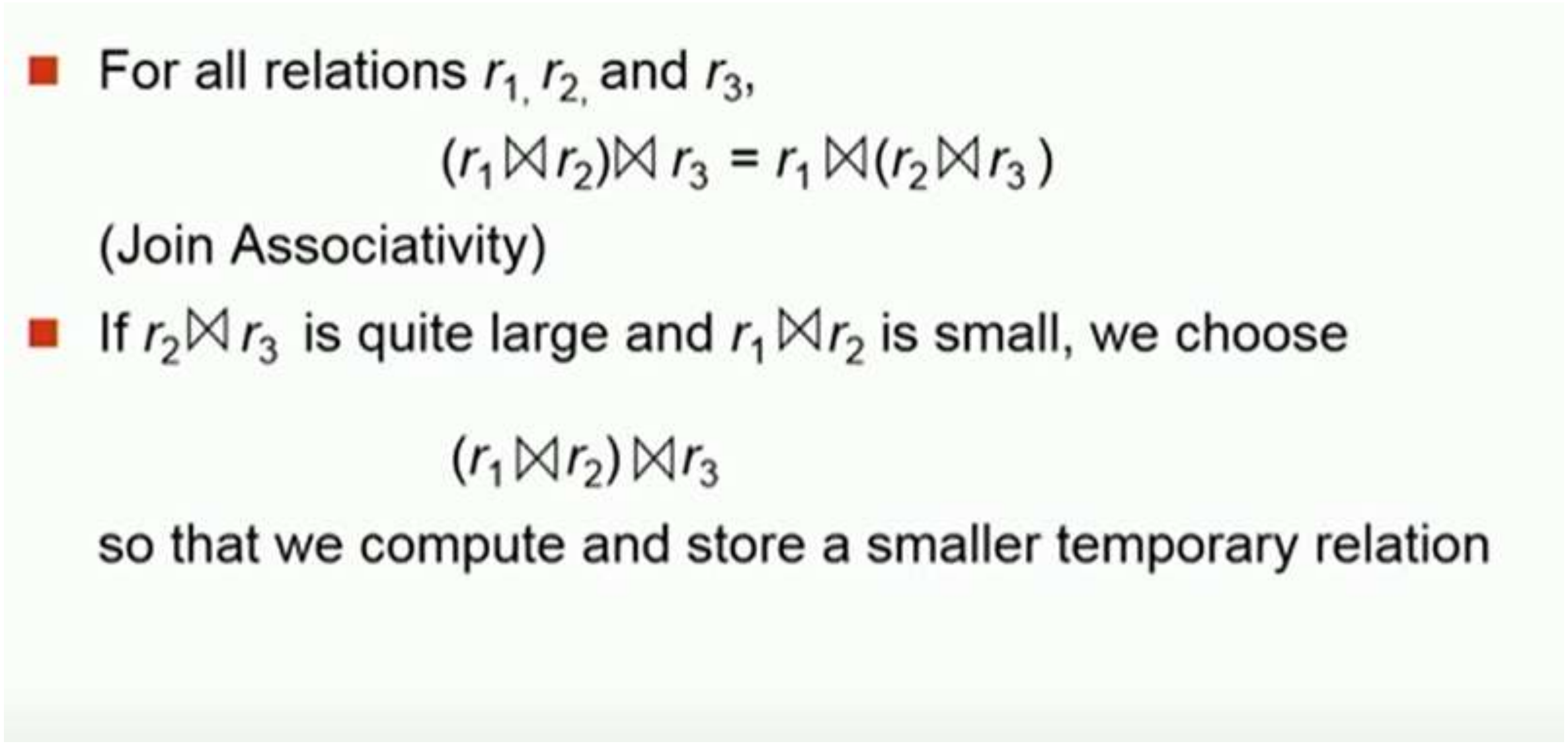
  $$\Pi_{name,\ title}(\Pi_{name,\ course\_id}(\sigma_{dept\_name=\ \text{"Music"}}(instructor) \bowtie teaches)) \bowtie \Pi_{course\_id,\ title}(course)$$
- Performing the projection as early as possible reduces the size of the relation to be joined

course(<u>*course id*</u>, *title*, *dept name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept name*, *salary*)
teaches(<u>*ID*</u>, <u>*course id*</u>, <u>*sec id*</u>, <u>*semester*</u>, *year*)

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

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

Join Ordering Example



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

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

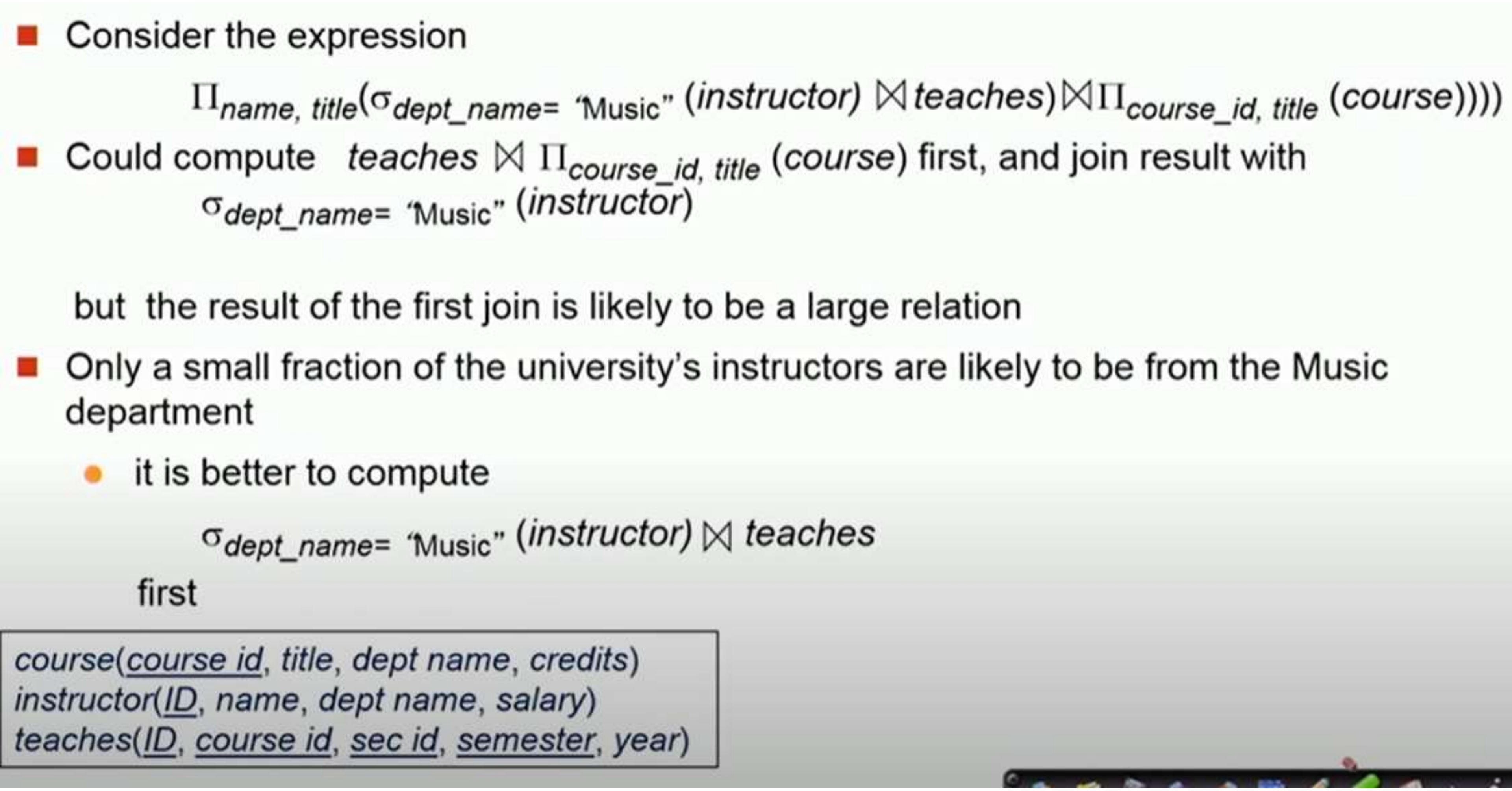
(Join Associativity)

- 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

Join Ordering Example (Cont.)



- Consider the expression

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

- Could compute $teaches \bowtie \Pi_{course\_id,\ title}(course)$ first, and join result with $\sigma_{dept\_name=\ \text{“Music”}}(instructor)$

  but the result of the first join is likely to be a large relation

- Only a small fraction of the university's instructors are likely to be from the Music department
  - it is better to compute

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

    first

course(<u>*course id*</u>, *title*, *dept name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept name*, *salary*)
teaches(<u>*ID*</u>, <u>*course id*</u>, <u>*sec id*</u>, <u>*semester*</u>, *year*)

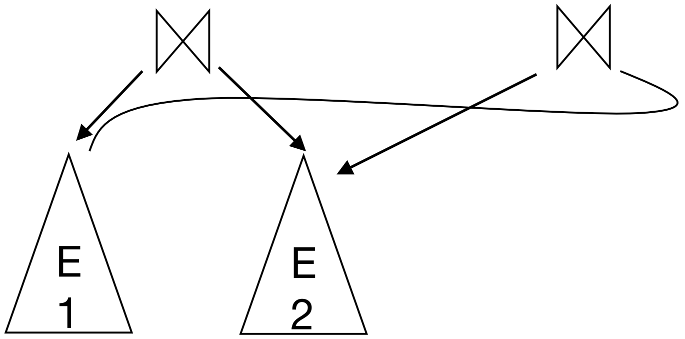
Enumeration of Equivalent Expressions

- 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 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
  - Two approaches
    - Optimized plan generation based on transformation rules
    - Special case approach for queries with only selections, projections and joins

Implementing Transformation Based Optimization

- Space requirements reduced by sharing common sub-expressions:
  - when E1 is generated from E2 by an equivalence rule, usually only the top level of the two are different, subtrees below are the same and can be shared using pointers
    - E.g. when applying join commutativity



  - Same sub-expression may get generated multiple times
    - Detect duplicate sub-expressions and share one copy
- Time requirements are reduced by not generating all expressions
  - Dynamic programming