



Chapter 16: Query Optimization

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



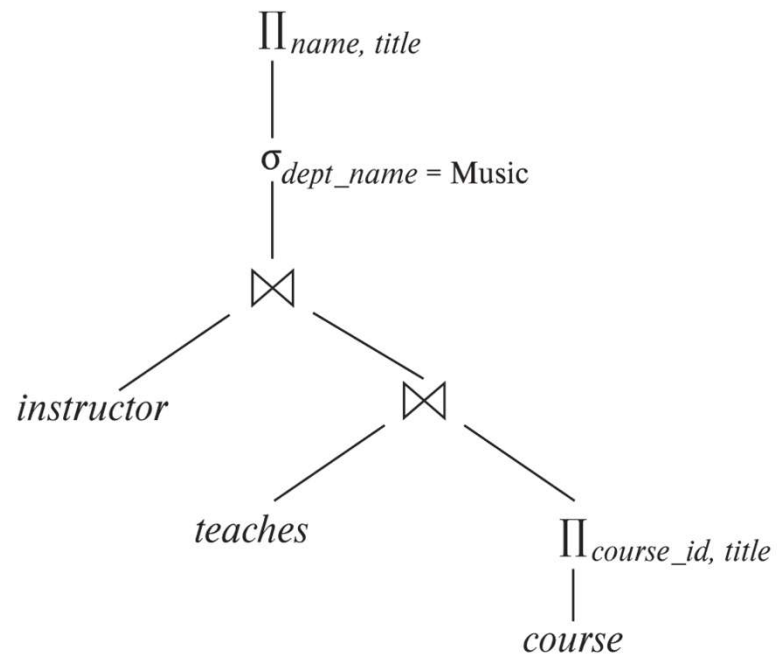
Outline

- 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

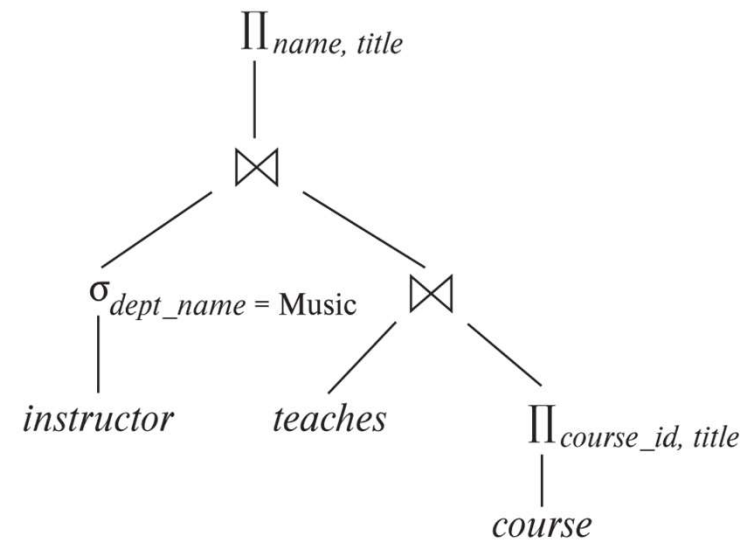


Introduction

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



(a) Initial expression tree

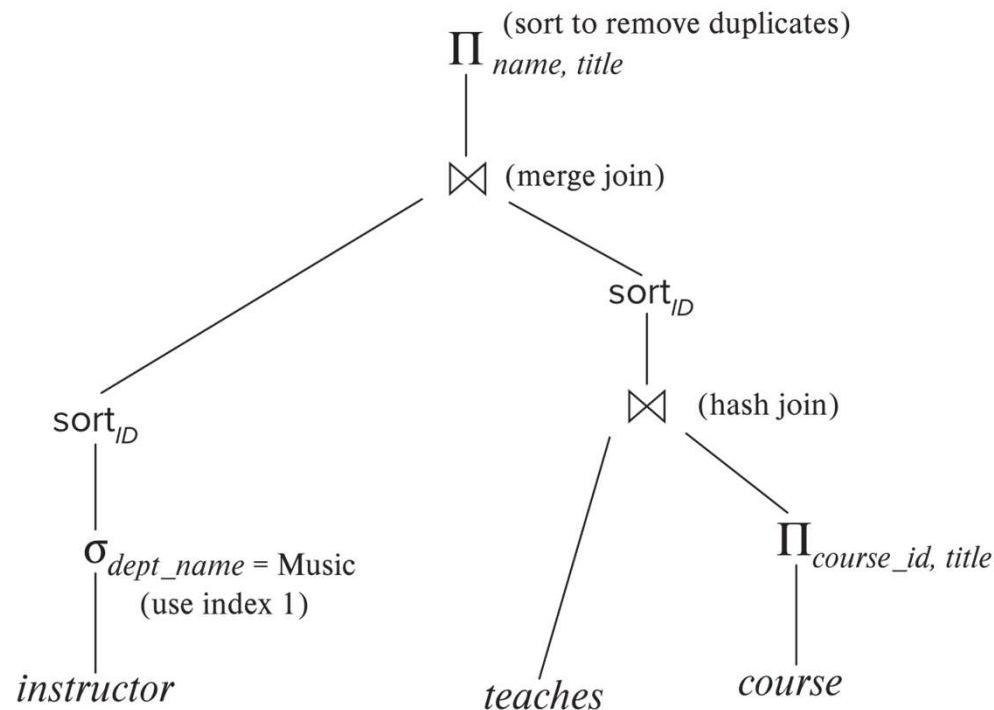


(b) Transformed expression tree



Introduction (Cont.)

- An **evaluation plan** 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 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:
 - 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



Viewing Query Evaluation Plans

- Most database support **explain** <query>
 - Displays plan chosen by query optimizer, along with cost estimates
 - Some syntax variations between databases
 - Oracle: **explain plan for** <query> followed by **select * from** table (*dbms_xplan.display*)
 - SQL Server: **set showplan_text on**
- Some databases (e.g. PostgreSQL) support **explain analyse** <query>
 - Shows actual runtime statistics found by running the query, in addition to showing the plan
- Some databases (e.g. PostgreSQL) show cost as *f./*
 - *f* is the cost of delivering first tuple and */* is cost of delivering all results



Generating Equivalent Expressions

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



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.

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

2. Selection operations are commutative.

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

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

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

where $L_1 \subseteq L_2 \dots \subseteq L_n$

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



Equivalence Rules (Cont.)

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

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

6. (a) Natural join operations are associative:

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

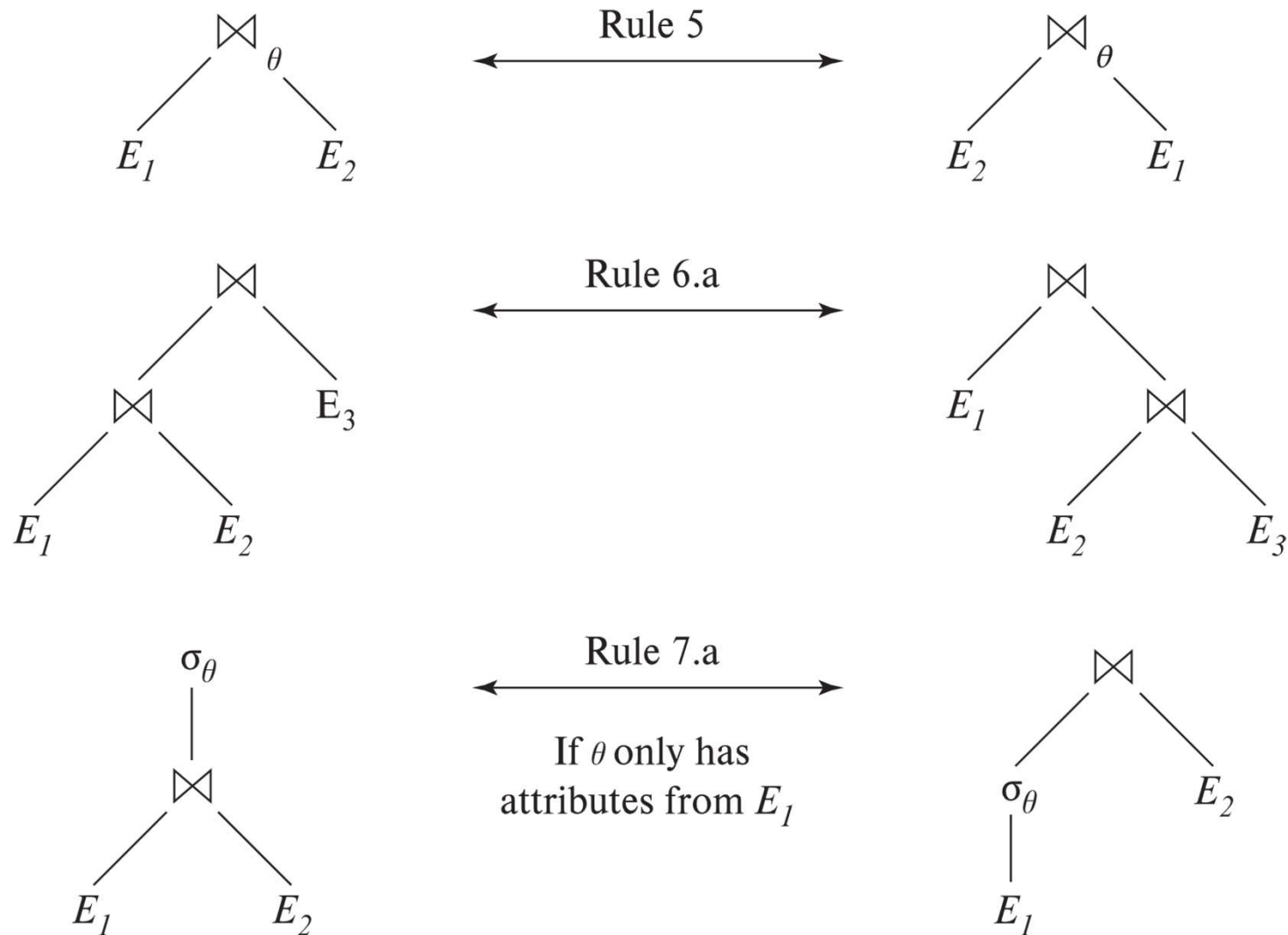
- (b) Theta joins are associative in the following manner:

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

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.

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) \equiv (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

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

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



Equivalence Rules (Cont.)

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

(a) if θ involves only attributes from $L_1 \cup L_2$:

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

(b) In general, 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 $L_1 \cup L_2$.

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

Similar equivalences hold for outerjoin operations: \bowtie , \ltimes , and \rtimes



Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 \equiv E_2 \cup E_1$$

$$E_1 \cap E_2 \equiv E_2 \cap E_1$$

(set difference is not commutative).

10. Set union and intersection are associative.

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

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

11. The selection operation distributes over \cup , \cap and $-$.

a. $\sigma_{\theta}(E_1 \cup E_2) \equiv \sigma_{\theta}(E_1) \cup \sigma_{\theta}(E_2)$

b. $\sigma_{\theta}(E_1 \cap E_2) \equiv \sigma_{\theta}(E_1) \cap \sigma_{\theta}(E_2)$

c. $\sigma_{\theta}(E_1 - E_2) \equiv \sigma_{\theta}(E_1) - \sigma_{\theta}(E_2)$

d. $\sigma_{\theta}(E_1 \cap E_2) \equiv \sigma_{\theta}(E_1) \cap E_2$

e. $\sigma_{\theta}(E_1 - E_2) \equiv \sigma_{\theta}(E_1) - E_2$

preceding equivalence does not hold for \cup

12. The projection operation distributes over union

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



Exercise

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



Equivalence Rules (Cont.)

13. Selection distributes over aggregation as below

$$\sigma_{\theta}(G\gamma_A(E)) \equiv G\gamma_A(\sigma_{\theta}(E))$$

provided θ only involves attributes in G

14. a. Full outerjoin is commutative:

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

- b. Left and right outerjoin are not commutative, but:

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

15. Selection distributes over left and right outerjoins as below, provided θ_1 only involves attributes of E_1

a. $\sigma_{\theta_1}(E_1 \bowtie_{\theta} E_2) \equiv (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} E_2$

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

16. Outerjoins can be replaced by inner joins under some conditions

a. $\sigma_{\theta_1}(E_1 \bowtie_{\theta} E_2) \equiv \sigma_{\theta_1}(E_1 \bowtie_{\theta} E_2)$

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

provided θ_1 is null rejecting on E_2



Equivalence Rules (Cont.)

- Example of 16.a

$$\sigma_{\text{year}=2017}(\text{instructor} \bowtie \text{teaches}) \equiv \sigma_{\text{year}=2017}(\text{instructor} \bowtie \text{teaches})$$

- Note that several equivalences that hold for joins do not hold for outerjoins

- $\sigma_{\text{year}=2017}(\text{instructor} \bowtie \text{teaches}) \not\equiv \text{instructor} \bowtie \sigma_{\text{year}=2017}(\text{teaches})$

- Outerjoins are not associative

$$(r \bowtie s) \bowtie t \not\equiv r \bowtie (s \bowtie t)$$

- e.g. with $r(A,B) = \{(1,1)\}$, $s(B,C) = \{(1,1)\}$, $t(A,C) = \{\}$

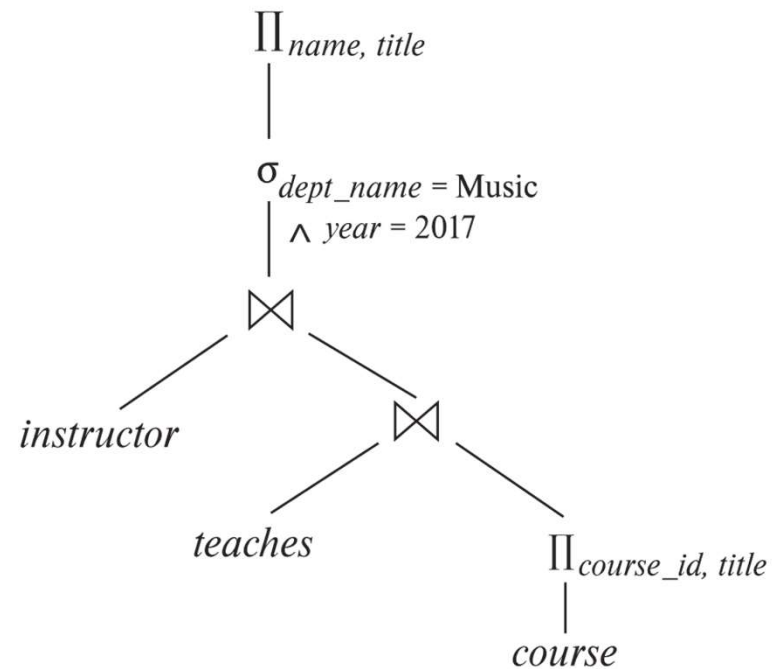


Transformation Example: Pushing Selections

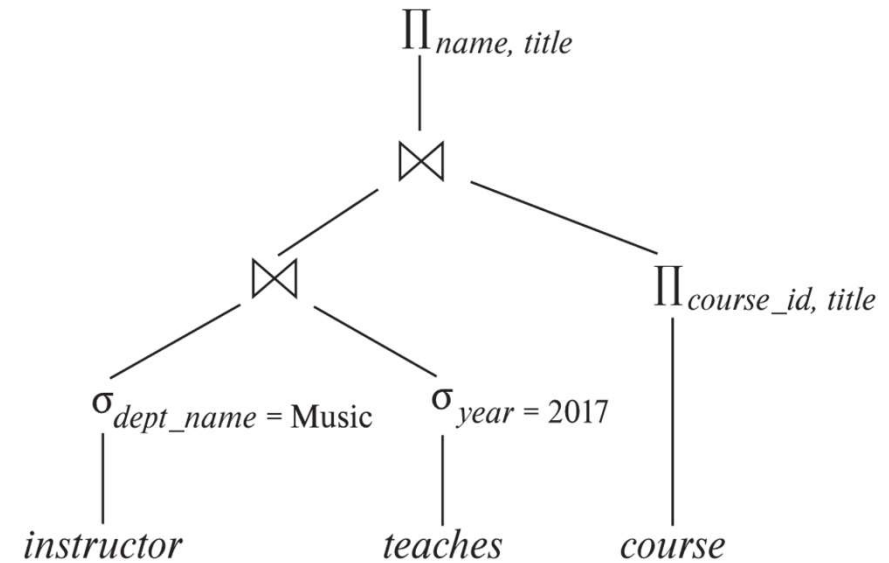
- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
 - $\Pi_{name, title}(\sigma_{dept_name = 'Music'}(instructor \bowtie (teaches \bowtie \Pi_{course_id, title}(course))))$
- Transformation using rule 7a.
 - $\Pi_{name, title}((\sigma_{dept_name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



Multiple Transformations (Cont.)



(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.



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) \bowtie

- 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.



Enumeration of Equivalent Expressions

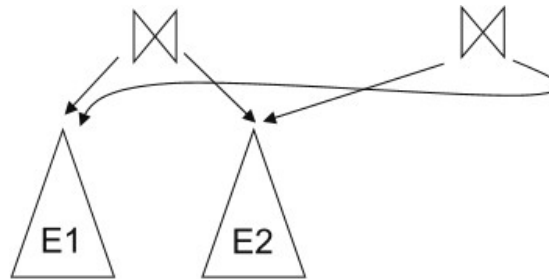
- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
 - Repeat
 - apply all applicable equivalence rules on every subexpression of every equivalent expression found so far
 - add newly 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
 - We will study only the special case of dynamic programming for join order optimization



Cost Estimation

- Cost of each operator computer as described in Chapter 15
 - 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
 - To do so, we require additional statistics
 - E.g., number of distinct values for an attribute
- More on cost estimation later



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.



Exercise

- 16.5
- No need to make a submission