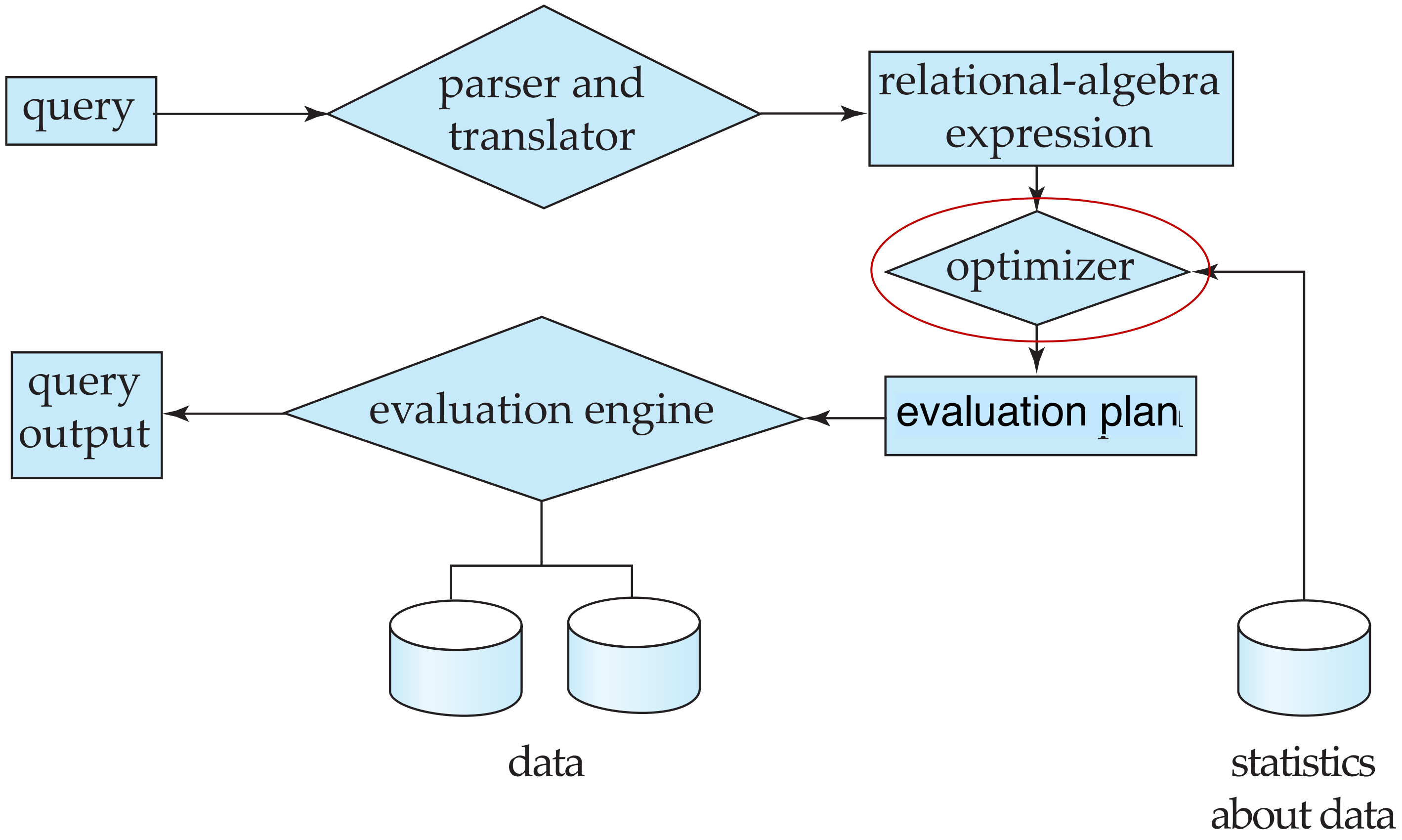
Query Processing

Steps of Query Processing

1. Parsing and translation
   - Parser to check syntax, verify relation names, etc.
   - Translate query into internal form (~ relational algebra)
2. Optimization
   1. Equivalent expression trees
   2. Best evaluation plan
3. Evaluation



Outline

- Query Optimization
- Generation Equivalent Expressions
  - Equivalence Rules
- Cost Estimation of Expressions

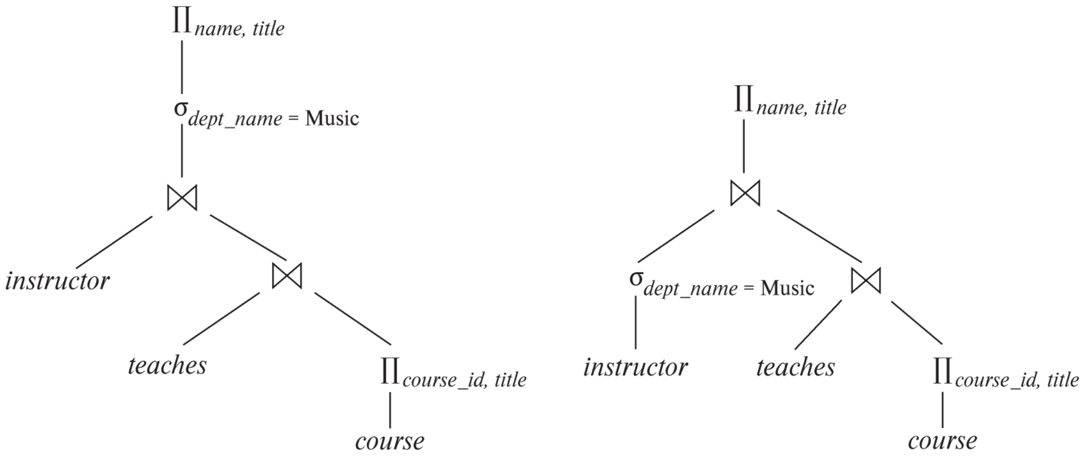
Query optimization

- Single relational algebra expression ~ many equivalent expressions
  - Equivalent expressions trees might involve different computational cost

```
SELECT salary FROM instructor WHERE salary < 75000
```

$$\sigma_{salary<75000}(\Pi_{salary}(instructor))$$

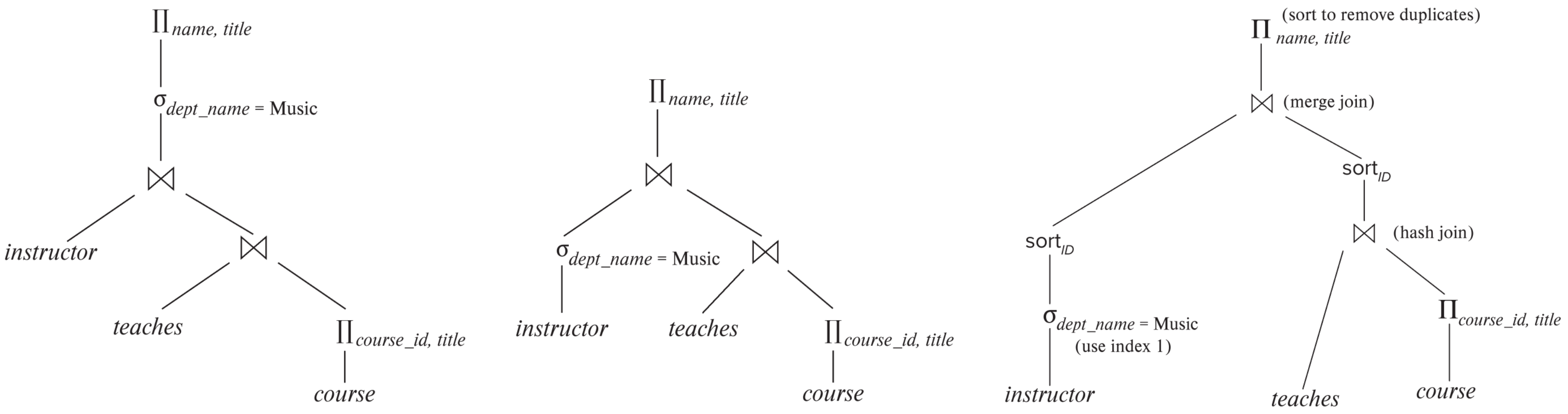
$$\Pi_{salary}(\sigma_{salary<75000}(instructor))$$



(a) Initial expression tree (b) Transformed expression tree

Query optimization

- Different algorithms can be used to evaluate the same relational algebra operation
  - Evaluation primitive: A relational algebra operation annotated with instructions on how to evaluate it
  - **Evaluation plan**: Sequence of primitives to evaluate a query
    - Feed to the *query-evaluation engine*



(a) Initial expression tree (b) Transformed expression tree (c) Query evaluation plan

Query optimization

- **Query optimization**: To select the most efficient **evaluation plan** among all the possible ones to process a query
  - User is not expected to write the most efficient query!

- Worth taking some time to select a good plan!
  - Cost difference may be huge… From seconds to days!

- Steps for query optimization:
  1. Generate **equivalent expression trees** using **equivalence rules**
     - Expected to find (better) expressions at relational-algebra level
  2. Annotate resultant expressions to get alternative evaluation plans
     - Detailed strategy for processing the query (algorithms, indices, etc.)
  3. Choose the cheapest **evaluation plan** based on **estimated costs**

Query optimization

- Do you want to see the actual query evaluation plan?
  - Use **`explain`** *<query>* to display plan chosen by query optimizer, along with cost estimates
    Syntax in your DBMS may change!
  - Use **`explain analyse`** *<query>* to see actual runtime statistics found by running the query

Generating Equivalent Expressions

- Two relational algebra expressions are **equivalent** if both generate the same set of tuples on every *legal* DB instance
  - The order is irrelevant
  - SQL considers *multisets* of tuples
    - Two expressions are *equivalent* if both generate the same *multiset* of tuples on every *legal* DB instance.

- The *optimizer* uses equivalence rules to obtain equivalent expressions
  - An **equivalence rule** states that expressions of two forms are equivalent

Equivalence Rules: some examples

1. Conjunctive selection is similar to *sequence of selections*:

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

2. Selection operation is *commutative*:

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

3. In a sequence of projections, only the last one matters:

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

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

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

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

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

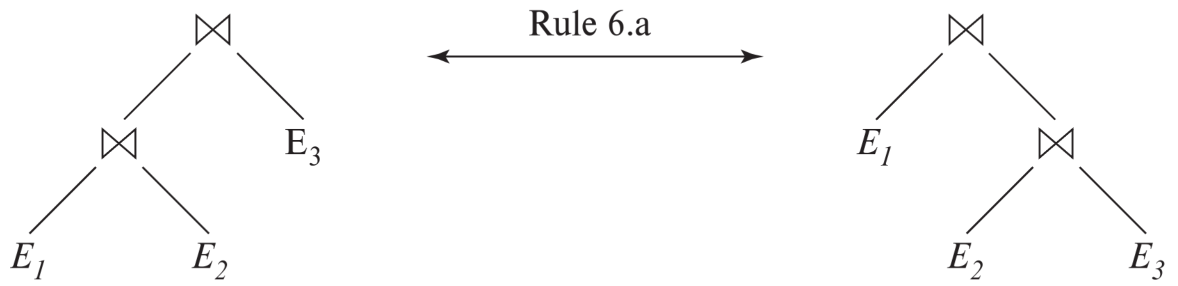
Equivalence Rules: some examples

5. Theta (and natural) join is *commutative*:

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

6. a) Natural join is *associative*:

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



b) Theta join is *associative* only when:

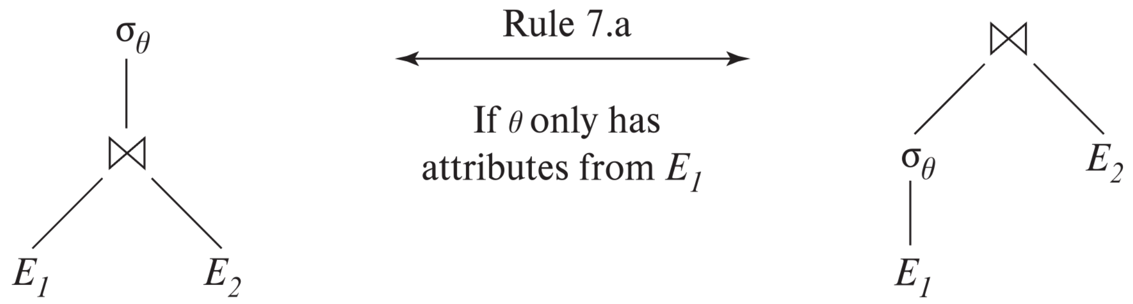
$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 \equiv E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

where θ_2 involves attributes from only E_2 and E_3.

Equivalence Rules: some examples

7. The selection operation *distributes* over the theta join when all the attributes in θ_s come from a single join expression:

$$\sigma_{\theta_s}(E_1 \bowtie_{\theta_j} E_2) \equiv (\sigma_{\theta_s}(E_1)) \bowtie_{\theta_j} E_2$$



8. The projection operation *distributes* over the theta join if the join condition θ involves only attributes from $L_1 \cup L_2$:

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

where L_1 and L_2 are attributes from E_1 and E_2, respectively.

And many more…

Generating Equivalent Expressions

- How is all this useful?
- Consider join associativity:

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

  - We can decide to compute **first** joins that lead to **smaller intermediate** relations

E.g.,

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

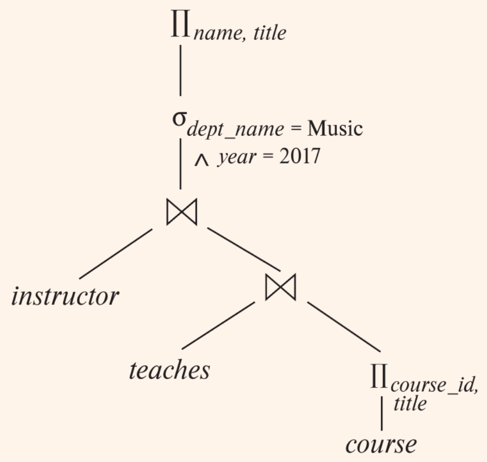
Only a few instructors belong to *Music* dept., so start with

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

Generating Equivalent Expressions

E.g., *"Find all instructors' names of Music dept. who taught a course in 2017, and the title of these courses"*

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"} \wedge year = 2017}(instructor \bowtie (teaches \bowtie \Pi_{course\_id,title}(course))))$$

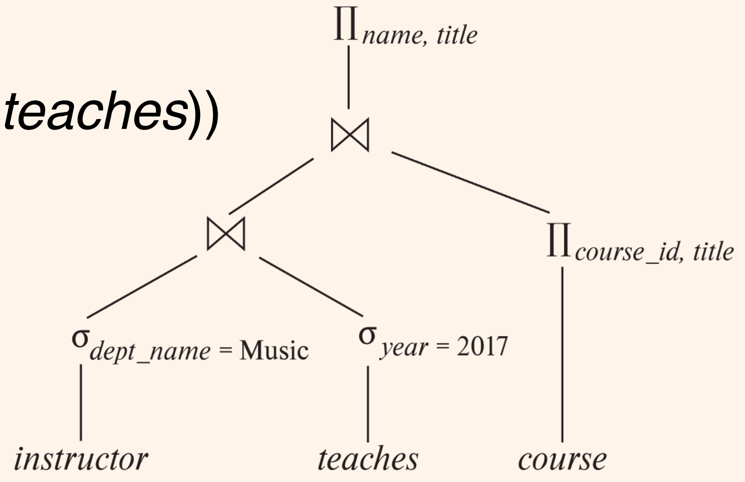


- By join associativity (ER 6a):

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"} \wedge year = 2017}(instructor \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course))$$

- By "conjunctive selection to sequence of selections" (ER 1) and "selection distribution on join" (ER 7):

$$\Pi_{name,\ title}((\sigma_{dept\_name=\text{"Music"}}(instructor) \bowtie \sigma_{year = 2017}(teaches)) \bowtie \Pi_{course\_id,\ title}(course))$$



Generating Equivalent Expressions

- Query optimizers perform systematic **enumeration of equivalent expressions**
  - Simplest approach (expensive!!):

```
genAllEquivExpressions(E):
1. T ⟵ {E}
2. while T grows then
   1. Check equivalence rules Rj in expressions Ei in T
   2. if any subexpression ei of Ei matches one side of Rj then
      1. E' = Ei - Rj^1 + Rj^2          //Ei but substituting Rj sides
      2. if E' not in T then    T ⟵ T ∪{E'}
3. return T
```

- Two improvements:
  - Use shared subexpressions to reduce space requirements
  - Use **cost estimates** to avoid certain expressions

Now what? Or which?

- Now we know how to **generate** tons of equivalent **expressions**…

What's next?

- Can we tell **which expression is better**?

Let's estimate their cost

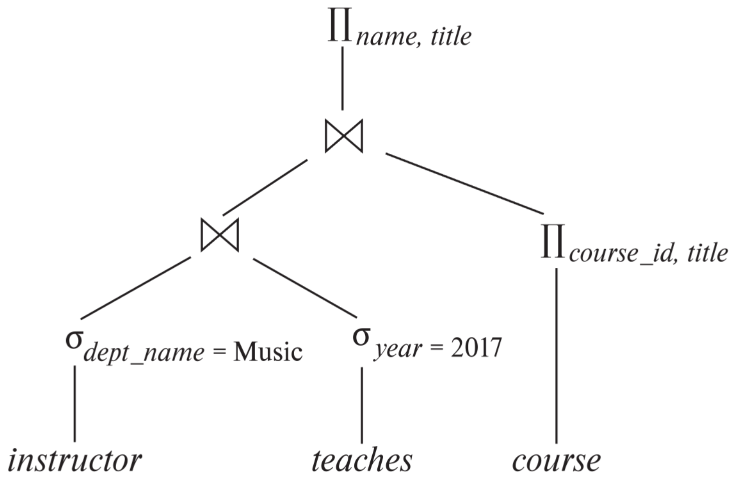
Less cost = more efficient query processing

Cost Estimation of Expressions

- **Cost** of an expression depends on the size of the input relation(s) and other statistics.
  - We use **approximated** values, and so is the cost
    - Recomputed after each DB modification (expensive!!)
    - Recomputed during idle time on specific events or under demand (outdated values used!!)
- In an expression tree, estimation of relevant statistics starts **from the leaves up** to the root.

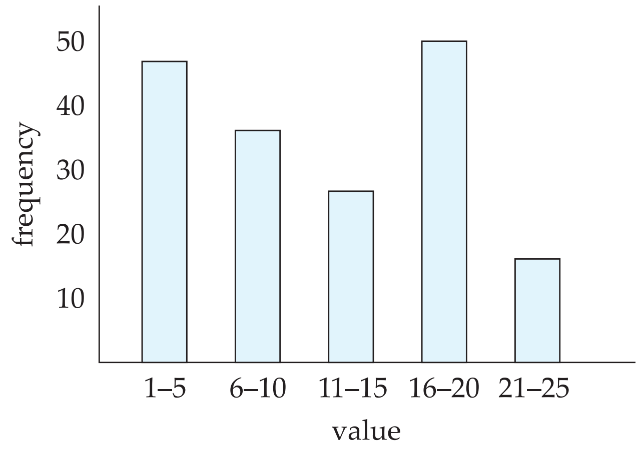
Statistics:

- n_r: no. tuples in relation r
- l_r: size of tuples of relation r
- $v_{r(A)}$: no. distinct values of the (set of) attribute(s) A in relation r
- b_r: no. blocks that contain tuples of relation r
- f_r: blocking factor of relation r (no. tuples of r that fit into a block)
  - If tuples are physically stored together: $b_r = \lceil n_r / f_r \rceil$
- Info about indices (e.g., tree depth, no. leaves)



Cost Estimation of Expressions

- **Histograms** are usually stored too on numeric *attributes*
  - Equal-width vs Equal-depth
  - Frequency
  - No. distinct values
  - Without no. distinct values, cost estimates assume uniform distribution



- Many DB systems store the ***N* most-frequent values** with their counts
  - Histogram is built only on remaining values

Size Estimation of *Selection*

- A **single *equality* comparison**: $\sigma_{A=a}(r)$
  - $c = n_r / v_{r(A)}$: no. records that would satisfy the selection
    No information → uniformity
  - $c = 1$: if A is a key attribute
  - $c = f_{h_a}/v_{h_a}$: Use histogram when available

- A **single comparison**: $\sigma_{A \le a}(r)$ [$\sigma_{A \ge a}(r)$ is symmetric]
  - If $min_{r(A)}$ and $max_{r(A)}$ are stored in our statistic's catalog:
    - $c = 0$, if $a < min_{r(A)}$
    - $c = n_r$, if $a \ge max_{r(A)}$
    - $c = n_r \cdot (a - min_{r(A)})/(max_{r(A)} - min_{r(A)})$, otherwise
  - Use histograms when available
  - $c = n_r / 2$: if value a is unknown or in absence of statistical information

Size Estimation of *Join*

- **Cartesian product** ($r \times s$):
  - $n_x = n_r \cdot n_s$ tuples, where each tuple takes $l_x = l_r + l_s$ bytes

- **Natural join** ($r \bowtie s$): Let us assume r(R) and s(S)
  - If $R \cap S = \varnothing$, then $(r \bowtie s) \equiv (r \times s)$
  - If $R \cap S$ is a foreign key in S referencing R, then
    no. tuples in $(r \bowtie s)$ = no. tuples in s
  - If $R \cap S = \{A\}$ is not a key in any case, assuming *value uniformity:*
    $n_{\bowtie} = \min(n_r \cdot n_s / v_{r(A)} \, ; \, n_r \cdot n_s / v_{s(A)})$

With histograms, these estimates can be improved

- **Theta join** ($r \bowtie_\theta s$): calculate it from $(r \bowtie_\theta s) \equiv \sigma_\theta(r \times s)$

Number of Distinct Values Estimation

- **Selection**: $\sigma_\theta(r)$
  - If θ forces A to take a specific value (e.g., $\theta \equiv (A = 3)$):
    $$V_{\sigma_\theta(r)(A)} = 1$$
  - If θ forces A to take a value on a set (e.g., $\theta \equiv (A = 1) \vee (A = 3)$):
    $$V_{\sigma_\theta(r)(A)} = \text{size of the given value set}$$
  - If $\theta \equiv (A\ op\ r)$: *** op = comparison operator*
    $$V_{\sigma_\theta(r)(A)} = V_{r(A)} \cdot s$$
  - In general: $$V_{\sigma_\theta(r)(A)} = \min(V_{r(A)},\ n_{\sigma_\theta(r)})$$
- **Join**: $r \bowtie s$
  - If all attributes in A are from r:
    $$V_{(r \bowtie s)(A)} = \min(V_{r(A)}\ ;\ n_{(r \bowtie s)})$$
  - If A contains attributes A_r and A_s from r and s:
    $$V_{(r \bowtie s)(A)} = \min(V_{r(A_r)} \cdot V_{s(A_s - A_r)}\ ;\ V_{r(A_r - A_s)} \cdot V_{s(A_s)}\ ;\ n_{r \bowtie s})$$

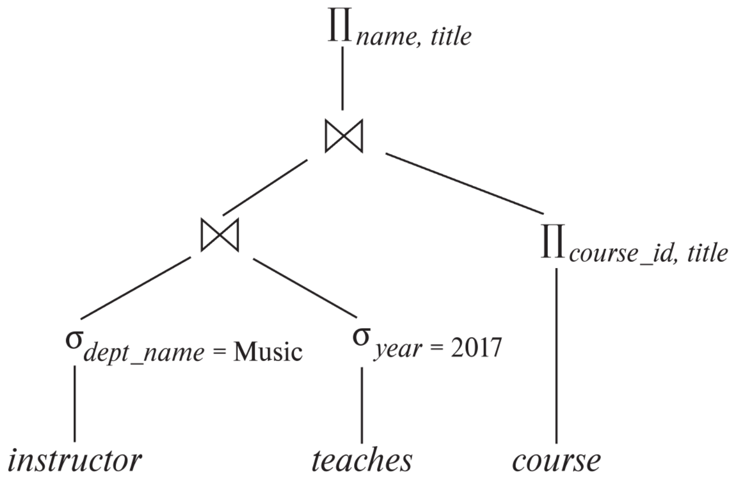
$A \equiv$ attribute(s) obtained after the join

Cost Estimation of Expressions

- **Cost** of an expression depends on the size of the input relation(s) and other statistics.
  - We use **approximated** values, and so is the cost
    - Recomputed after each DB modification (expensive!!)
    - Recomputed during idle time on specific events or under demand (outdated values used!!)
- In an expression tree, estimation of relevant statistics starts **from the leaves up** to the root.

Statistics:

- n_r: no. tuples in relation r
- l_r: size of tuples of relation r
- $v_{r(A)}$: no. distinct values of the (set of) attribute(s) A in relation r
- b_r: no. blocks that contain tuples of relation r
- f_r: blocking factor of relation r (no. tuples of r that fit into a block)
  - If tuples are physically stored together: $b_r = \lceil n_r / f_r \rceil$
- Info about indices (e.g., tree depth, no. leaves)



Query Processing