Query Optimization

CS460 Databases for Data Scientists

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Previously on Database Technology

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Last time**, we have seen different algorithms for query processing
  - plus their best/worst case cost estimates
- **Today**
  - how to come up with an *efficient* query execution plan

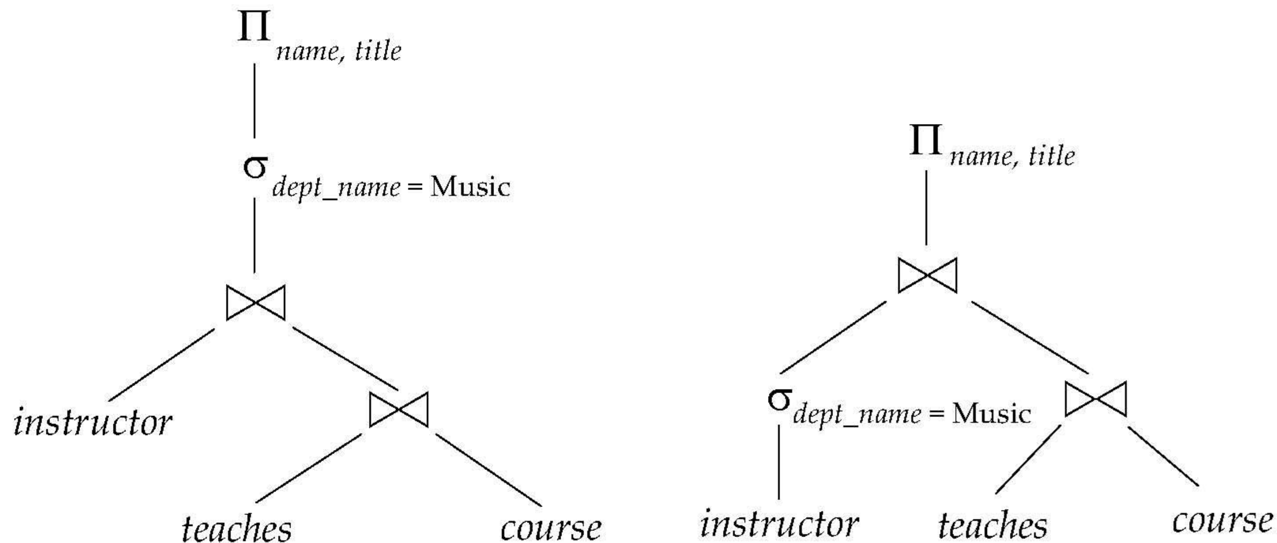
Agenda

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

Introduction

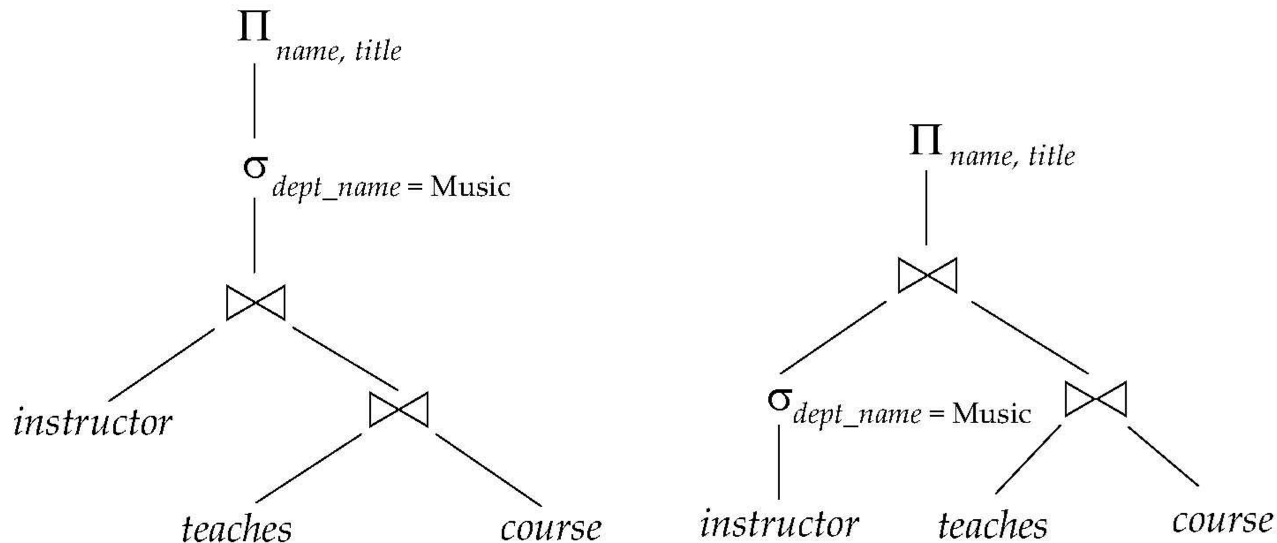
SELECT name, title
FROM instructor, teaches, course
WHERE instructor.inst_ID = teaches.inst_ID AND
course.dept_name = instructor.dept_name AND
dept_name="Music"



Introduction

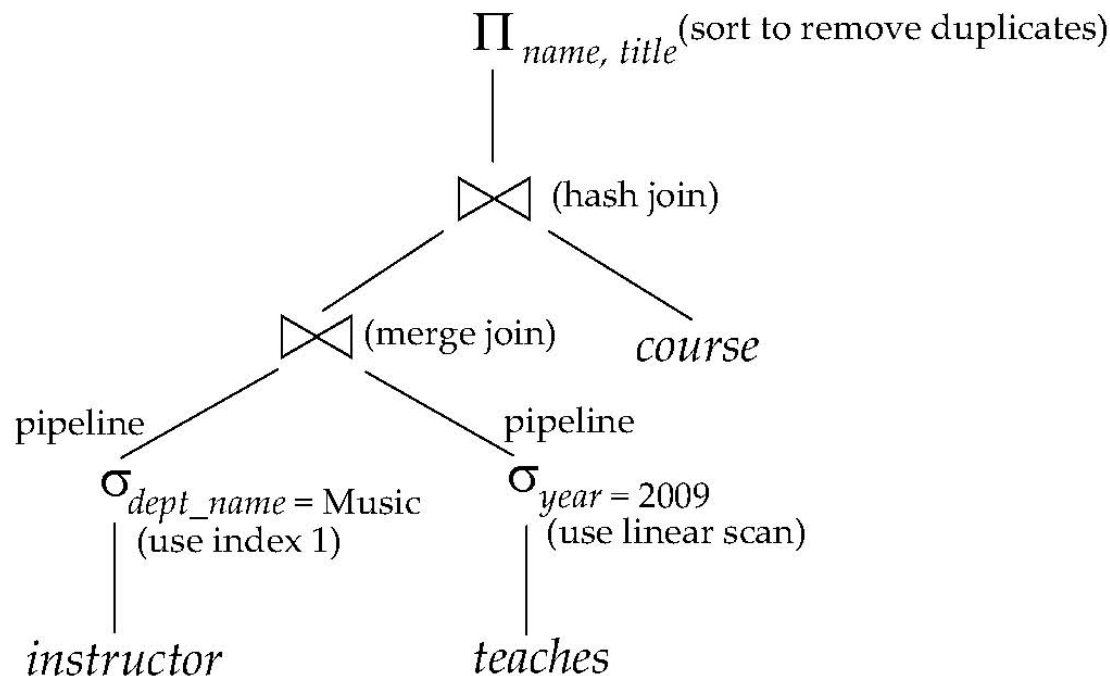
UNIVERSITY OF MANNHEIM
Data and Web Science Group

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



Evaluation Plans

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



Cost-based Query Optimization

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Cost difference between evaluation plans for a query can be enormous (seconds vs. days in some cases)
- Steps in cost-based query optimization
  1. Generate logically equivalent expressions using equivalence rules
  2. Annotate resulting 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, e.g.:
    - 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

Equivalence of Relational Algebra Expressions

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on *every legal* database instance
  - order of tuples is irrelevant
  - when they have the same number of duplicates (multiset version of the relational algebra)

- Equivalent results must not be a result of chance, e.g.
  - SELECT name FROM employee WHERE id=“12345”
    → “Smith”
  - SELECT name FROM employee WHERE birthday=“30.10.1974”
    → “Smith”
  - Those results could be different on a different database instance

Equivalence Rules

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

$$\sigma_{\theta 1 \wedge \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$$

```
SELECT name, title
FROM instructor
WHERE dept_name="Music" AND
          salary>50000
```

=

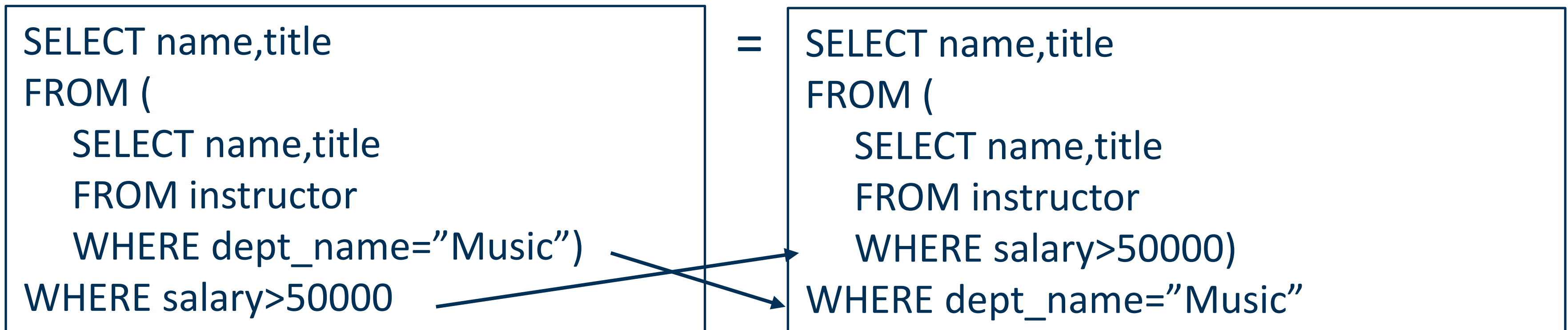
```
SELECT name,title
FROM (
      SELECT name,title
      FROM instructor
      WHERE dept_name="Music")
WHERE salary>50000
```

Note: SQL statements in these slides are solely illustrative!

Equivalence Rules

2. Selection operations are **commutative**.

$$\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$$



Equivalence Rules

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

$$\Pi_{L1}(\ \Pi_{L2}(\ ....\ (\Pi_{Ln}(E))...)) = \Pi_{L1}(E)$$

```
SELECT name, title
FROM (
    SELECT name,title,salary,dept_name
    FROM instructor
    WHERE dept_name=”Music” AND
                salary>50000
)
```

=

```
SELECT name,title
FROM instructor
WHERE dept_name=”Music” AND
            salary>50000
```

Equivalence Rules

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

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

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

```
SELECT name, building
FROM (
    SELECT name,building
    FROM instructor,department
    WHERE instructor_dept_name =
              department.dept_name)
WHERE SALARY>50000
```

=

```
SELECT name, building
FROM instructor, department
WHERE instructor.dept_name =
          department.dept_name AND
          salary>50000
```

Equivalence Rules

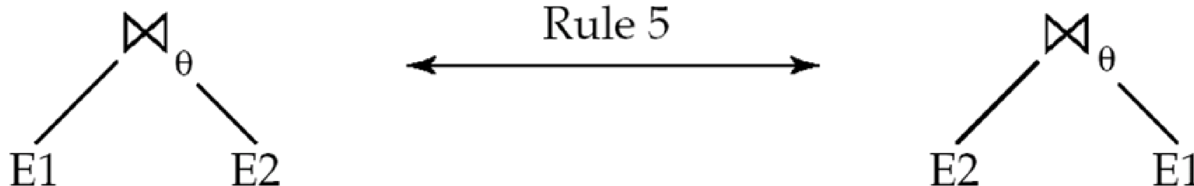
University of Mannheim, Data and Web Science Group

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

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

```
SELECT name, building
FROM instructor, department
WHERE instructor.dept_name =
        department.dept_name AND
        salary > 50000
```

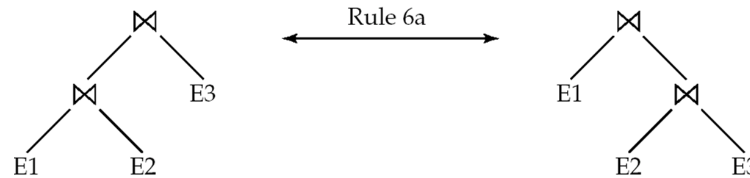
```
SELECT name, building
FROM department, instructor
WHERE department.dept_name =
        instructor.dept_name AND
        salary > 50000
```



Equivalence Rules (ctd.)

6. a) Natural join operations are **associative**

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




```
SELECT *
FROM instructor, (
    SELECT *
    FROM teaches, course
    WHERE teaches.course_ID =
              course.course_ID
) AS joined
WHERE instructor.inst_ID = joined.inst_ID
```

```
SELECT *
FROM course, (
    SELECT *
    FROM instructor,teaches
    WHERE instructor.inst_ID =
              teaches.inst_ID
) AS joined
WHERE course.course_ID = joined.course_ID
```

Equivalence Rules (ctd.)

6. a) Natural join operations are **associative**

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

- If $E_1 \bowtie E_2$ is quite large and $E_2 \bowtie E_3$ is small, we choose $E_1 \bowtie (E_2 \bowtie E_3)$ so that temporary relation is smaller
- Example:
  – (course $\bowtie$ teaches) $\bowtie$ $\sigma_{dept_name= \text{"Music"}}$ (*instructor*)
    - but the result of the first join is likely to be a large relation
  – course $\bowtie$ (*teaches* $\bowtie$ $\sigma_{dept_name= \text{"Music"}}$ (*instructor*))
    - Only a small fraction of the university's instructors are likely to be from the Music department

Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

6. 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 E2 and E3

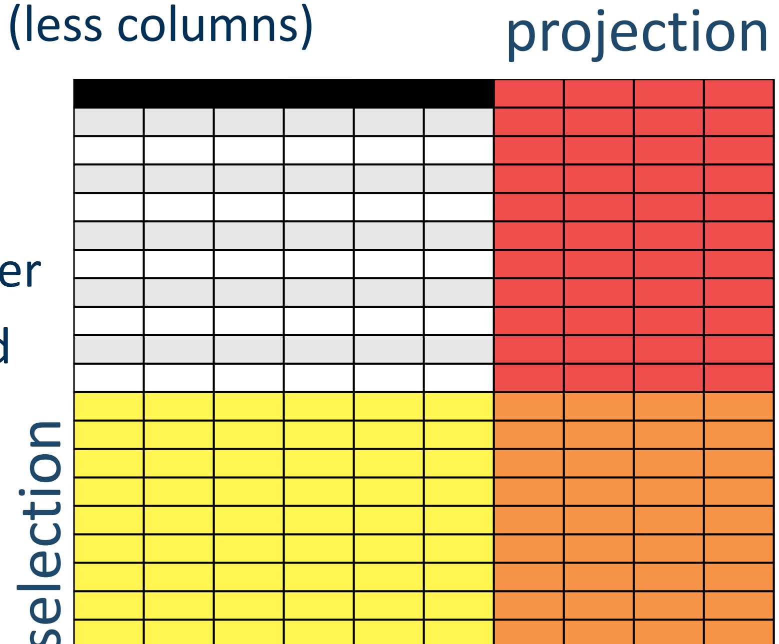
```
SELECT *
FROM instructor, (
    SELECT *
    FROM teaches, course
    WHERE teaches.course_ID =
               course.course_ID
) AS joined
WHERE instructor.inst_ID = joined.inst_ID
           AND salary > 50000
```

```
SELECT *
FROM course, (
    SELECT *
    FROM instructor,teaches
    WHERE instructor.inst_ID =
               teaches.inst_ID
) AS joined
WHERE course.course_ID = joined.course_ID
           AND salary > 50000
```

Pusing Selection and Projection

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Pushing selection to earlier steps
  - Leads to joining *shorter* tables (less rows)
- Pushing projection to earlier steps
  - Leads to joining *narrower* tables (less columns)
- In each case
  - Make intermediate results smaller
  - Reduce amount of cache needed
  - Make subsequent steps faster



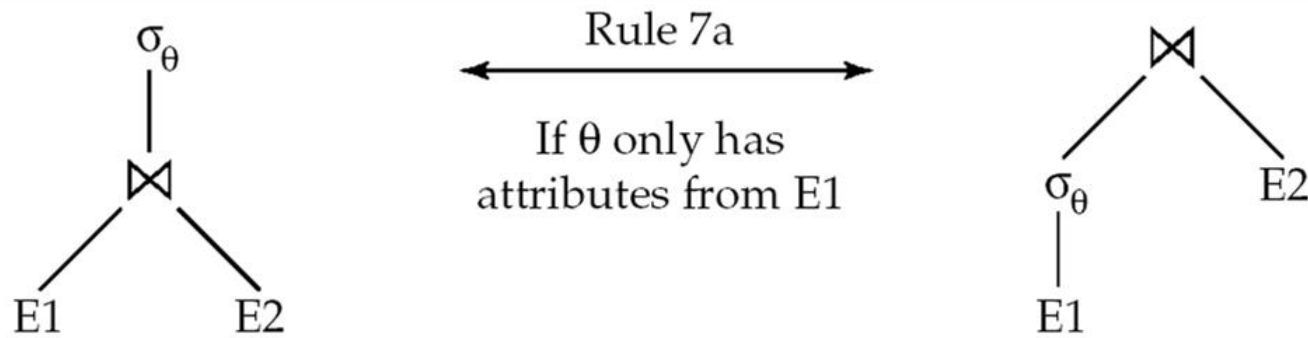
Equivalence Rules (ctd.)

7. The selection operation distributes over the theta join operation under the following two conditions:

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

Pushing **selections** down in the tree

$$\sigma_{\theta 0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_{\theta} E_2$$



Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

7. The selection operation distributes over the theta join operation under the following two conditions:

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

$\sigma_{salary>50000}($ instructor $\bowtie_{\theta}$ department$) = \sigma_{salary>50000}($instructor$) \bowtie_{\theta}$ department

```
SELECT *
FROM instructor, department
WHERE instructor.dept_name =
        department.dept_name AND
        salary>50000
```

```
SELECT *
FROM
  (SELECT * FROM instructor WHERE salary>50000)
  AS R1,
  (SELECT * FROM department) AS R2
WHERE R1.dept_name = R2.dept_name
```

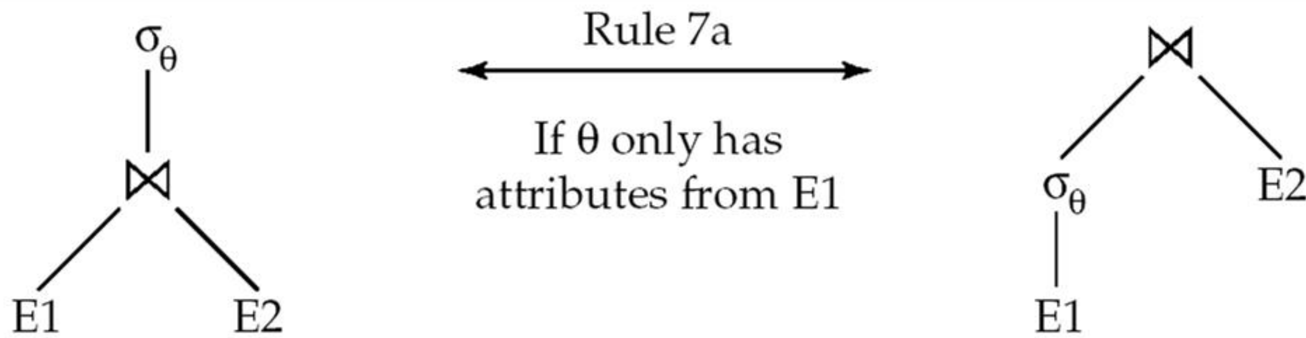
Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

7. The selection operation distributes over the theta join operation under the following two conditions:

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

$$\sigma_{\theta 0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_{\theta} E_2$$



Example: Pushing Selection

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- 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

Note the closing brackets

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

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

Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

7. The selection operation distributes over the theta join operation under the following two conditions:

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

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

Example with Multiple Expressions

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- 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

$$\Pi_{name,\ title}(\sigma_{dept_name= \text{"Music"} \land year = 2009}(instructor \bowtie (teaches \bowtie \Pi_{course_id,\ title}(course))))$$

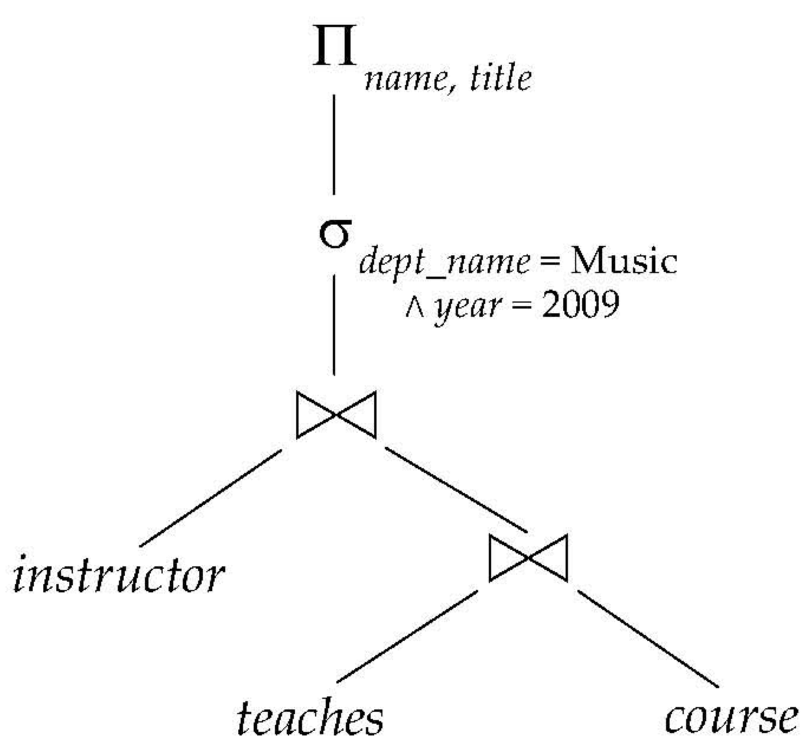
- Transformation using join associatively (Rule 6a):

$$\Pi_{name,\ title}(\sigma_{dept_name= \text{"Music"} \land year = 2009}((instructor \bowtie teaches) \bowtie \Pi_{course_id,\ title}(course)))$$

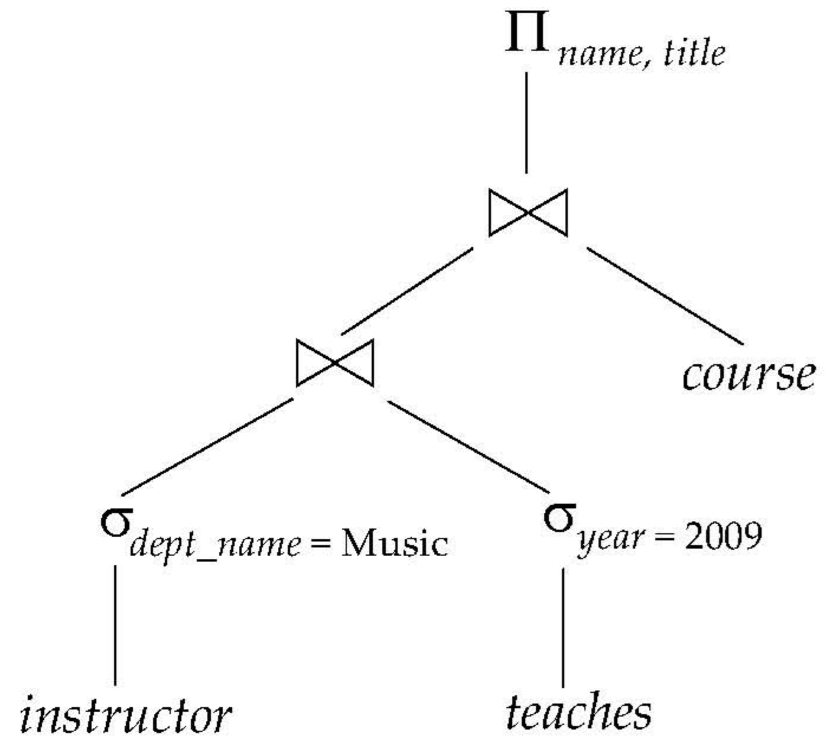
- Transformed expression provides an opportunity to apply the "perform selections early" rule (7b), resulting in

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

Example with Multiple Expressions



(a) Initial expression tree



(b) Tree after multiple transformations

Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

8. The projection operation distributes over the theta join operation as follows:
   (a) if θ involves only attributes from $L_1 \cup L_2$:

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

Pushing **projections** down in the tree

Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

   (b) Consider a join $E_1 \bowtie_\theta E_2$

   L_3 and L_4 are attributes needed for join but not afterwards

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

   – 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 $L_1 \cup L_2$

Equivalence Rules (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

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

   (b) Consider a join $E_1 \bowtie_\theta E_2$

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

L_1 contains name and salary
L_2 contains building

L_3 contains instructor.dept_name
L_4 contains department.dept_name

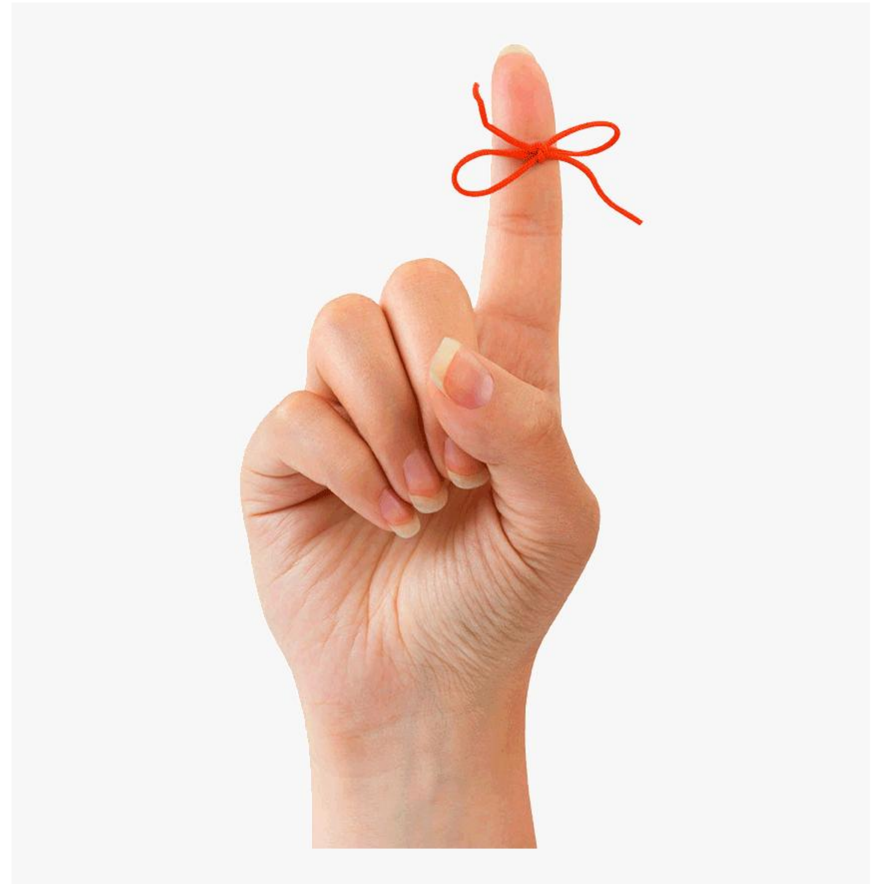
```
SELECT name,salary,building
FROM instructor, department
WHERE instructor.dept_name =
         department.dept_name
```

```
SELECT name,salary,building
FROM
 (SELECT name,salary,dept_name FROM instructor) AS R1,
 (SELECT building,dept_name FROM department) AS R2
WHERE R1.dept_name = R2.dept_name
```

From Equivalence Rules to Query Plans

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Equivalence rules
  - Find alternative query plans
- Query optimization
  - Pick an optimal query plan
  - (or at least a good one)



Enumeration of Equivalent Expressions

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- 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

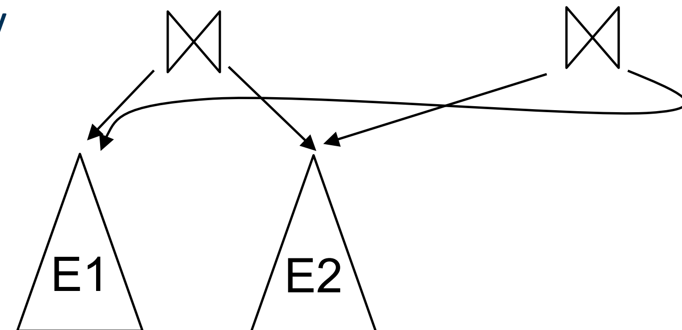
  add newly generated expressions to the set of equivalent expressions

  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

Transformation based Optimization

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- 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 for Execution Plans

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Cost of each operator computed as described in last lecture
  - 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

Cost-based Optimization

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Consider finding the best join ordering for $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$
- Join is commutative and associative. For n=3, we have
  - $(r_1,r_2),r_3$; $(r_2,r_1),r_3$; $r_3,(r_1,r_2)$; $r_3,(r_2,r_1)$;
    $(r_1,r_3),r_2$; $(r_3,r_1),r_2$; $r_2,(r_1,r_3)$; $r_2,(r_3,r_1)$;
    $(r_3,r_2),r_1$; $(r_2,r_3),r_1$; $r_1,(r_3,r_2)$; $r_1,(r_2,r_3)$.
- In general, the number is very large
  - Mathematically: $\frac{(2(n-1))!}{(n-1)!}$

Note: factorial complexity (O(n!)) is even worse than exponential!

$n = 5 \rightarrow 1{,}680$
$n = 10 \rightarrow$ >17 billion!

- No need to generate all the join orders
  - Dynamic programming:
    compute least-cost join order for any subset of $\{r_1, r_2, \ldots r_n\}$
  - reduces complexity to $O(3^n)$

Choosing a Good Execution Plan

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Naively: for each operation, pick the cheapest algorithm
  - given the statistics
  - caution: may not yield best overall algorithm!
- **Example 1**: merge-join may be costlier than hash-join
  - but may provide a sorted output
    which reduces the cost for an outer level aggregation
- **Example 2**: nested-loop join may be a costly variant
  - but provides opportunity for pipelining
- Practical query optimizers incorporate
  elements of the following two broad approaches
  - Search all the plans and choose the best plan in a cost-based fashion
  - Uses heuristics to choose a plan

Interesting Sort Orders

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Consider the expression $(r_1 \bowtie r_2) \bowtie r_3$ (with A as common attribute)
- An **interesting sort order** is a particular sort order of tuples that could be useful for a later operation
  - Using merge-join to compute $r_1 \bowtie r_2$ may be costlier than hash join
    - but generates result sorted on A
  - Which in turn may make merge-join with r_3 cheaper
    - which may reduce cost of join with r_3 and minimizing overall cost
  - Sort order may also be useful for result ordering and aggregation
- Not sufficient to find the best join order for each subset of the set of *n* given relations
  - must find the best join order for each subset, **for each interesting sort order**
    - extension of dynamic programming algorithms
  - Usually, number of interesting orders is quite small
    - does not affect time/space complexity significantly

Heuristic Optimization

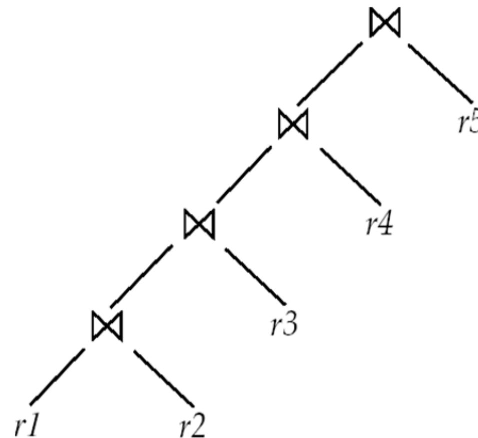
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Cost-based optimization is expensive, even with dynamic programming
- Alternative: use *heuristics* to reduce the number of choices that must be made in a cost-based fashion
  - may miss the *best* solution, but yields a *good* solution
- Heuristic optimization transforms the query tree by using a set of rules that typically 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

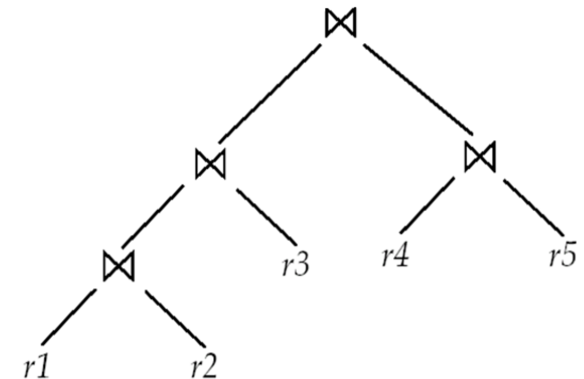
Practical Query Optimizers

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- 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
- Intricacies of SQL complicate query optimization
  - e.g. nested subqueries



(a) Left-deep join tree



(b) Non-left-deep join tree

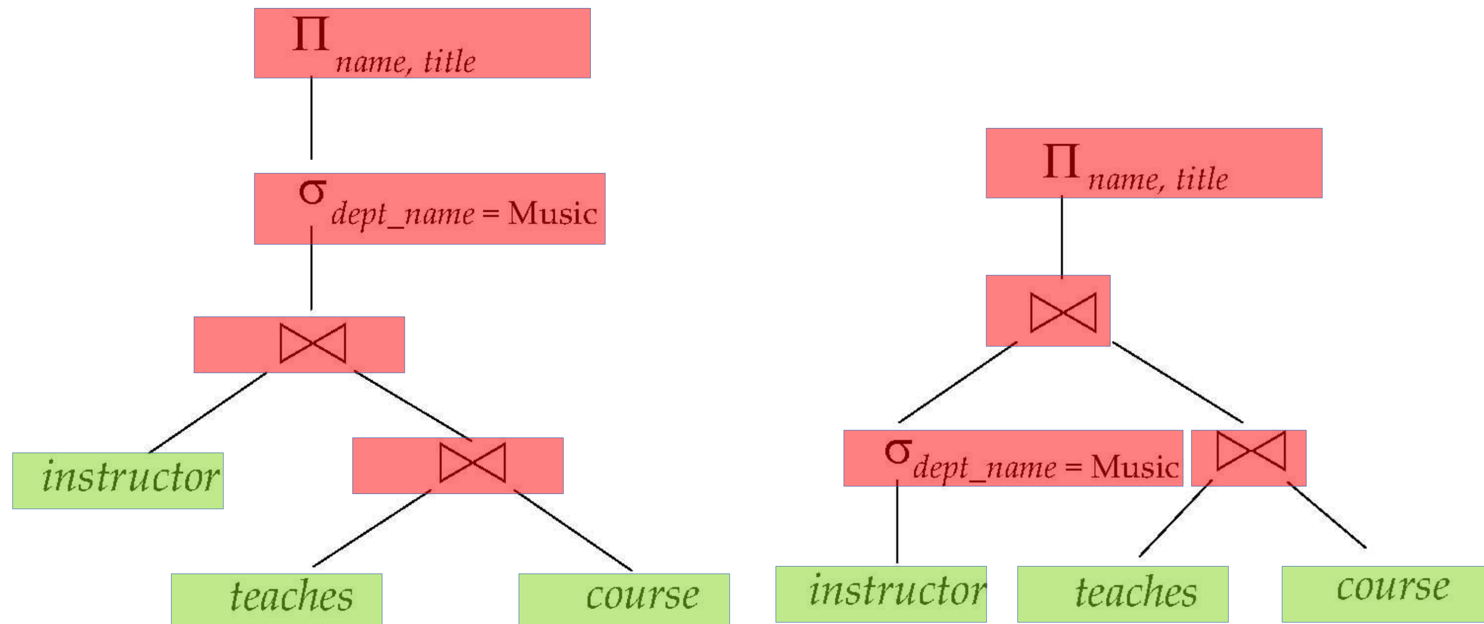
Practical Query Optimizers

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Savings vs. overhead
  - Large search space can lead to severe overhead
- Mixed approach: heuristics for cheap queries,
  exhaustive search for expensive query
- Strategies of practical optimizers (e.g., MS SQL Server) include
  - Optimization cost budget to stop optimization early
    - e.g.: found a plan with cost less than cost of optimization
  - Plan caching to reuse previously computed plan
    if query is resubmitted
    - Even with different constants in query

Relation Size Estimation

- We know the size of input tables
  - Not those of intermediate results
  - However, they might be interesting for the optimization
  - Hence, we have to *estimate* them



Statistical Information for Cost Estimation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Important numbers:
  - n_r: number of tuples in a relation r
  - ***V(A, r)***: number of **distinct values** that appear in relation r for attribute A; same as the size of $\prod_A(r)$
- Running example: student ⋈ takes
  - $n_{student}$ = 5,000
  - n_{takes} = 10,000
  - V(ID, takes) = 2500
    - on avg., each student who has taken a course has taken four courses
    - Attribute student_id in takes is a foreign key referencing student
  - V(ID, student) = 5000
    - Because ID in student relation is a primary key

Selection Size Estimates

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Given a selection criterion,
  c is the estimated number of matching tuples
- Exact selection: $\sigma_{A=v}(r)$
  - if A is a key attribute: $c = 1$
  - if A is a non-key attribute: $c = \frac{n_r}{V(A,r)}$

Selection Size Estimates

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Case of $\sigma_{A \geq V}(r)$ is symmetric

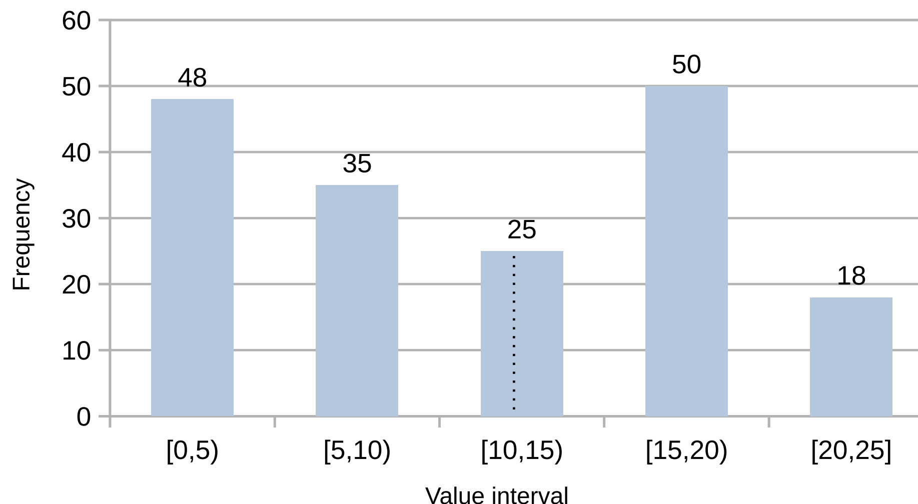
- Range selection: $\sigma_{A \leq V}(r)$
  - In absence of statistical information: $c = \frac{n_r}{2}$
  - If min(A,r) and max(A,r) are available in catalog
    - $c = 0$ if v < min(A,r)
    - $c = n_r * \frac{\text{v} - \min(\text{A,r})}{\max(\text{A,r}) - \min(\text{A,r})}$
  - With histogram: use all bins that completely satisfy condition and assume uniform distribution to calculate “partly” one bin

Example for Selection Size Estimation

- Range selection: $\sigma_{A \leq V}(r)$
  - Example: $\sigma_{\text{temperature} \leq 12}(r)$
    - without statistics (n_r = 176): $c = \frac{n_r}{2} = 88$
    - with min=0, max=25: $c = n_r * \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)} = 84.5$
    - using histogram: $c = 48 + 35 + 25*2/5 = 93$

assumes uniform distribution

assumes uniform distribution within histogram bars



Size Estimation for Complex Selections

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- s_i is the number of satisfying tuples of a condition θ_i

- **Conjunction:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$
  Assuming independence, estimate of tuples in the result is:

$$n_r * \frac{s_1}{n_r} * \frac{s_2}{n_r} * \cdots * \frac{s_n}{n_r} = n_r * \frac{s_1 * s_2 * \cdots * s_n}{n_r^n}$$

Probability is called selectivity

Number of conjunctions

- **Negation:** $\sigma_{\neg\theta}(r)$.
  Estimated number of tuples: $n_r - size(\sigma_\theta(r))$

Size Estimation for Complex Selections

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$.
- Let's use negation and disjunction:
  - $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r) = \sigma_{\neg(\neg \theta 1 \wedge \neg \theta 2 \wedge \ldots \wedge \neg \theta n)}(r)$

$\neg(A \vee B) = \neg A \wedge \neg B$

$$= n_r * \left(1 - \left(1 - \frac{s_1}{n_r}\right) * \left(1 - \frac{s_2}{n_r}\right) * \cdots * \left(1 - \frac{s_n}{n_r}\right)\right)$$

negation

Probability for $\neg\theta 2$

Estimating the Size of Joins

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Let r(R) and s(S) be relations (R and S are the attributes)
- The Cartesian product *r* x *s* contains $n_r * n_s$ tuples
  - each tuple occupies $s_r + s_s$ bytes.
- If $R \cap S = \emptyset$, then $r \bowtie s$ is the same as *r* x *s*
- 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*
- If $R \cap S$ is a foreign key in *s* referencing *r*, then the number of tuples in $r \bowtie s$ is exactly the same as the number of tuples in *s.*
  - The case for $R \cap S$ being a foreign key referencing *s* is symmetric.
  - In the example query *student* $\bowtie$ *takes,* *student_id* in *takes* is a foreign key referencing *student*
  - hence, the result has exactly n_{takes} tuples, which is 10,000

Estimating the Size of Joins

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- If $R \cap S = \{A\}$ is *not* a key for *r* or *s*
  If we assume that every tuple in r produces tuples in r ⋈ *s,*
  the number of tuples in *r* ⋈ *s* is estimated to be:
  $$\frac{n_r * n_s}{V(A,s)}$$
- If the reverse is true, the estimate obtained will be:
  $$\frac{n_r * n_s}{V(A,r)}$$
- The lower of these two estimates is probably the more accurate one, i.e., we ultimately use
  $$\min\left(\frac{n_r * n_s}{V(A,s)}, \frac{n_r * n_s}{V(A,r)}\right)$$
- Can improve on above if histograms are available
  – Use formulas similar to above, for each cell of histograms on the two relations

Estimating the Size of Joins

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- If $R \cap S = \{A\}$ is not a key for r or s
- *Example:*
  computing student ⋈ takes without join information

$$\frac{n_{student} * n_{takes}}{V(ID, takes)} = \frac{5{,}000 * 10{,}000}{2{,}500} = 20{,}000$$

$$\frac{n_{student} * n_{takes}}{V(ID, student)} = \frac{5{,}000 * 10{,}000}{5{,}000} = 10{,}000$$

→ The minimum of the two is 10,000

Estimating the Size of Joins

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Left/right outer join:
  - Estimated size of r ⟕ s = *size of* $r \bowtie s$ + *size of* r
  - Case of right outer join is symmetric
- Full outer join:
  - Estimated size of r ⟗ s = *size of* $r \bowtie s$ + size of r + size of s

- *Note:* These are pessimistic estimates
  - i.e. upper bounds

- In our example: not all students have to take courses
  - hence, computing student ⟕ takes make sense
- Estimated upper bound: 10,000 + 5,000 = 15,000

Size Estimation for Projection and Aggregation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Projection: estimated size of $\prod_A(r) = V(A,r)$
  - Example: find all students that have taken courses
  - $\prod_{student_id}(takes)$ = V(student_id, takes) =2,500
- Aggregation : estimated size of ${}_A g_F(r) = V(A,r)$
  - Example: compute average courses taken by students
  - For each student, an average is computed
    - Hence: ${}_{\text{student_id}}g_{\text{avg}}(\text{takes})$ = 2,500

A Note on Subqueries

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Consider:
  - **SELECT** *name*
    FROM *instructor*
    WHERE EXISTS (SELECT *
    FROM *teaches*
    WHERE *instructor.ID = teaches.ID* **AND** *teaches.year = 2007*)

  vs.

  - **SELECT** *name*
    FROM *instructor, teaches*
    WHERE *instructor.ID = teaches.ID* **AND** *teaches.year = 2007*
- For the RDBMS, joins are easier to optimize than subqueries
  - Rule of thumb:
    - if in doubt, use a join rather than a subquery
  - That they are equivalent does not mean that they have the same performance!

Summary

- Queries can be expressed in multiple forms
  - equivalent in terms of results
  - but different in terms of performance
- Query Optimization
  - pick best execution plan
  - estimate time/memory consumption for an execution plan
  - based on statistical information
- A widely researched area
  - e.g., exploiting advanced statistics about datasets
  - e.g., exploiting log files and histories
  - etc.

Questions?

UNIVERSITY OF MANNHEIM
Data and Web Science Group