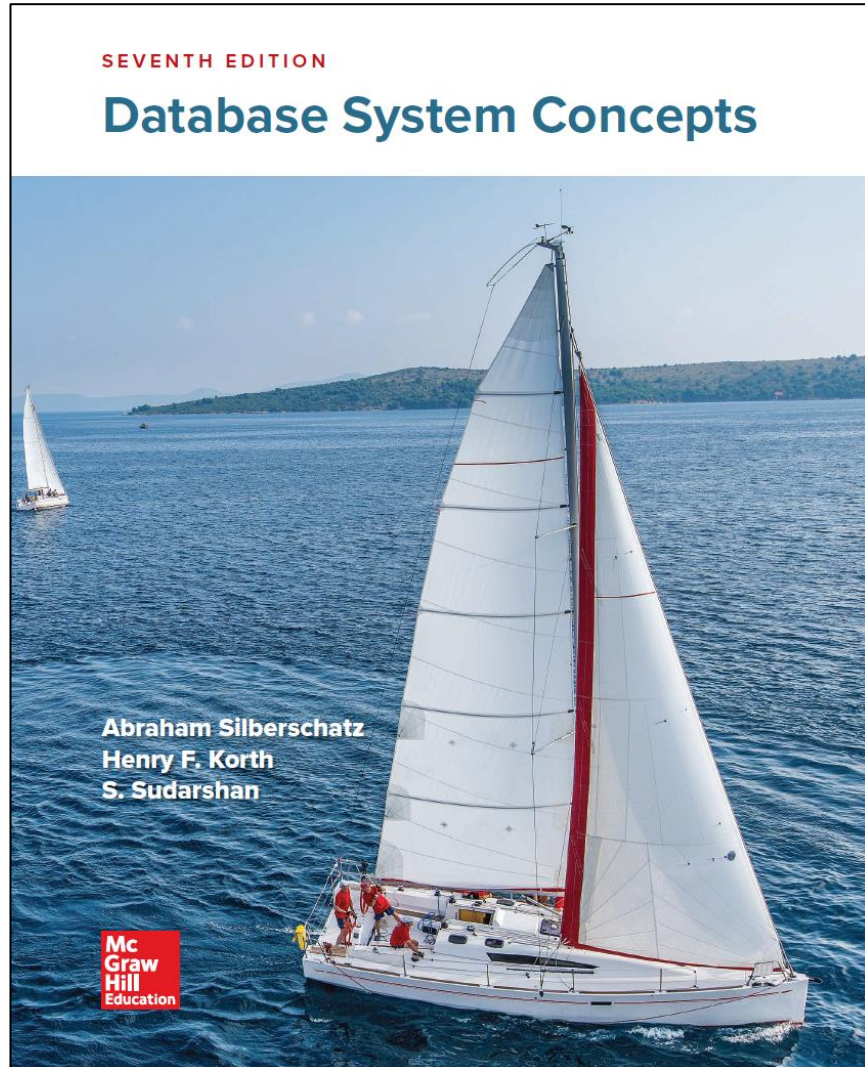


# Data Administration in Information Systems

---

Query optimization

# Query optimization



Contents xi

## Chapter 16 Query Optimization

16.1 Overview	743	16.5 Materialized Views	778
16.2 Transformation of Relational Expressions	747	16.6 Advanced Topics in Query Optimization	783
16.3 Estimating Statistics of Expression Results	757	16.7 Summary	787
16.4 Choice of Evaluation Plans	766	Exercises	789
		Further Reading	794

## PART SEVEN ■ TRANSACTION MANAGEMENT

### Chapter 17 Transactions

17.1 Transaction Concept	799	17.8 Transaction Isolation Levels	821
17.2 A Simple Transaction Model	801	17.9 Implementation of Isolation Levels	823
17.3 Storage Structure	804	17.10 Transactions as SQL Statements	826
17.4 Transaction Atomicity and Durability	805	17.11 Summary	828
17.5 Transaction Isolation	807	Exercises	831
17.6 Serializability	812	Further Reading	834
17.7 Transaction Isolation and Atomicity	819		

### Chapter 18 Concurrency Control

18.1 Lock-Based Protocols	835	18.8 Snapshot Isolation	872
18.2 Deadlock Handling	849	18.9 Weak Levels of Consistency in Practice	880
18.3 Multiple Granularity	853	18.10 Advanced Topics in Concurrency Control	883
18.4 Insert Operations, Delete Operations, and Predicate Reads	857	18.11 Summary	894
18.5 Timestamp-Based Protocols	861	Exercises	899
18.6 Validation-Based Protocols	866	Further Reading	904
18.7 Multiversion Schemes	869		

### Chapter 19 Recovery System

19.1 Failure Classification	907	19.8 Early Lock Release and Logical Undo Operations	935
19.2 Storage	908	19.9 ARIES	941
19.3 Recovery and Atomicity	912	19.10 Recovery in Main-Memory Databases	947
19.4 Recovery Algorithm	922	19.11 Summary	948
19.5 Buffer Management	926	Exercises	952
19.6 Failure with Loss of Non-Volatile Storage	930	Further Reading	956
19.7 High Availability Using Remote Backup Systems	931		

# Example

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	
32343	El Said	History	
33456	Gold	Physics	
45565	Katz	Comp. Sc	
58583	Califieri	History	
76543	Singh	Finance	
76766	Crick	Biology	
83821	Brandt	Comp. Sc	
98345	Kim	Elec. Eng	

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	
45565	CS-101	1	Spring	
45565	CS-319	1	Spring	
76766	BIO-101	1	Summer	
76766	BIO-301	1	Summer	
83821	CS-190	1	Spring	
83821	CS-190	2	Spring	
83821	CS-319	2	Spring	
98345	EE-181	1	Spring	

Figure 2.1 The *instructor* relation.

Figure 2.7 The *teaches* relation.

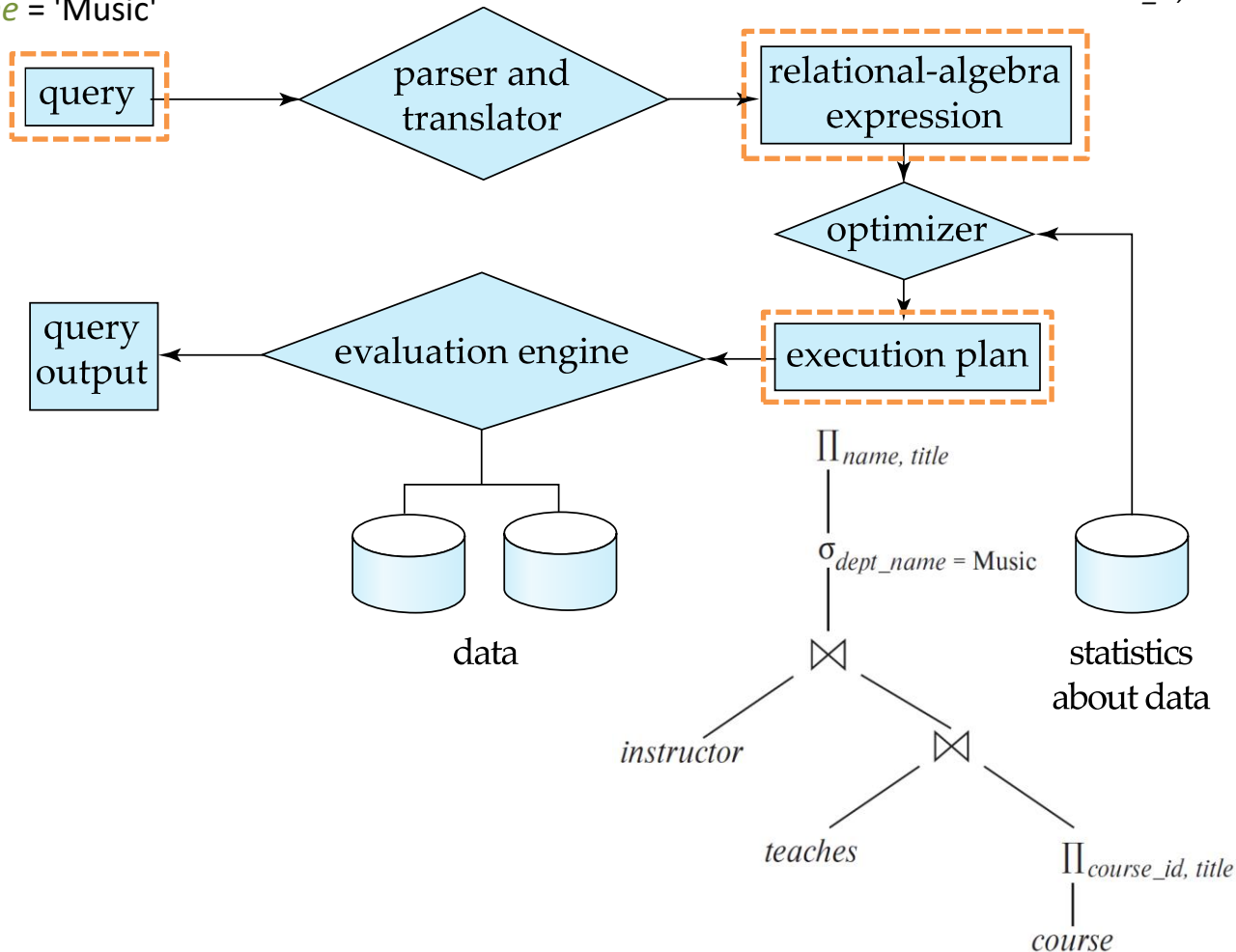
course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation.

```
select name, title
from instructor
  natural join teaches
  natural join course
where dept_name = 'Music'
```

# Query processing

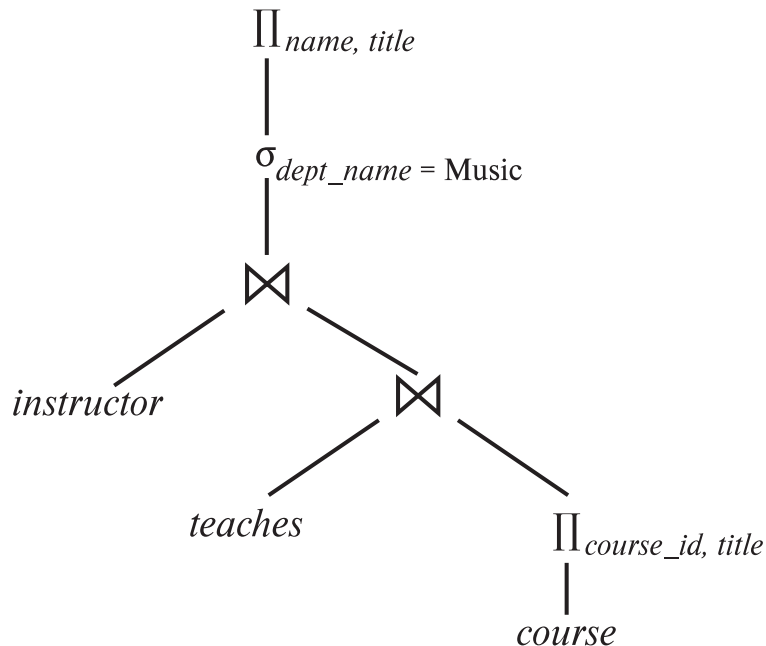
```
select name, title
from instructor
      natural join teaches
      natural join course
where dept_name = 'Music'
```

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


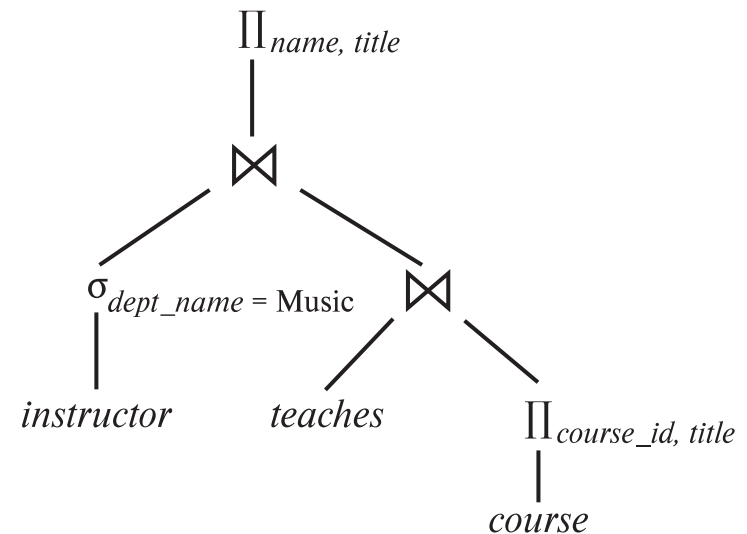
# Introduction

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

```
select name, title
from instructor
      natural join teaches
      natural join course
where dept_name = 'Music'
```



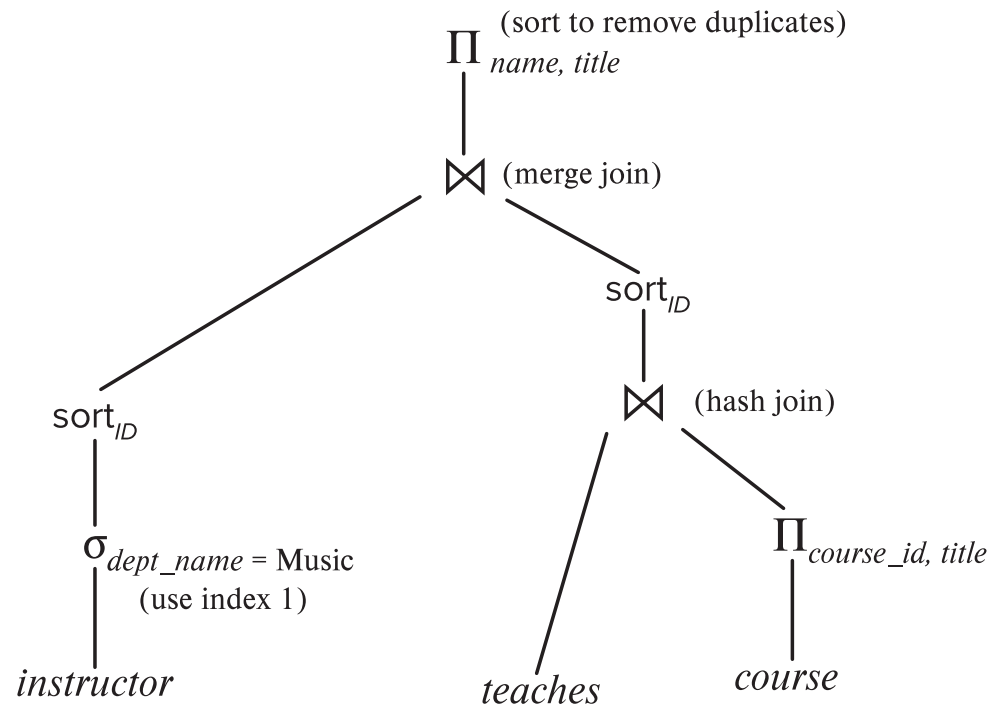
(a) Initial expression tree



(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 database system

# 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

# 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 database instance
  - Note: order of tuples is irrelevant
- 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 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:

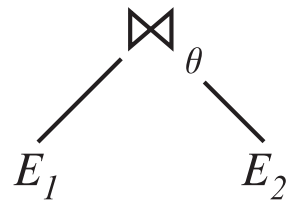
$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

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

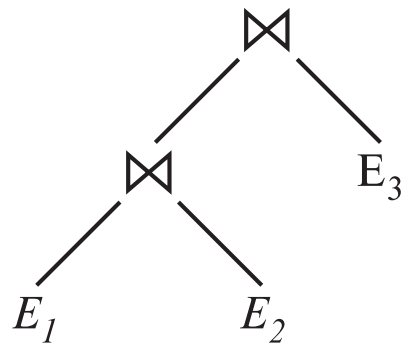
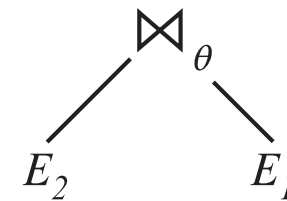
$$(E_1 \bowtie_{\theta_{12}} E_2) \bowtie_{\theta_{13} \wedge \theta_{23}} E_3 \equiv E_1 \bowtie_{\theta_{12} \wedge \theta_{13}} (E_2 \bowtie_{\theta_{23}} E_3)$$

where  $\theta_{ij}$  involves attributes from  $E_i$  and  $E_j$  only.

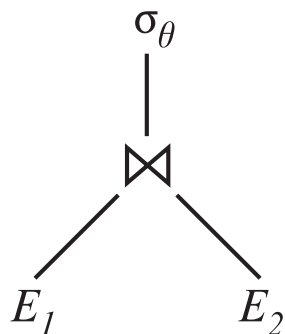
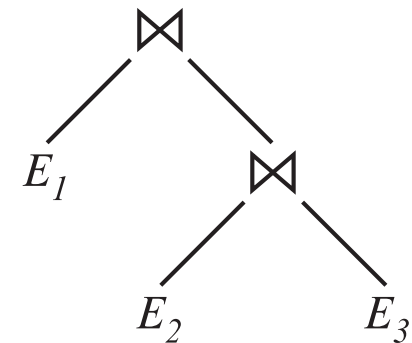
# Pictorial Depiction of Equivalence Rules



Rule 5

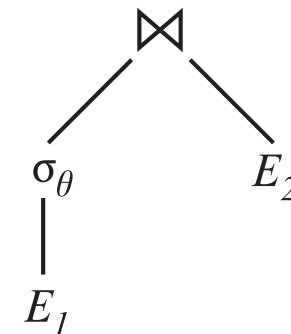


Rule 6.a



Rule 7.a

If  $\theta$  only has  
attributes from  $E_1$



## Equivalence Rules (Cont.)

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

(a) When all the attributes in  $\theta_1$  involve only the attributes of one of the expressions ( $E_1$ ) being joined.

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

(b) When  $\theta_1$  involves only the attributes of  $E_1$  and  $\theta_2$  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  $\theta$  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  $L_{1*}$  be attributes of  $E_1$  that are in join condition  $\theta$ , but are not in  $L_1 \cup L_2$
- Let  $L_{2*}$  be attributes of  $E_2$  that are in join condition  $\theta$ , but are not in  $L_1 \cup L_2$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) \equiv \Pi_{L_1 \cup L_2}(\Pi_{L_1 \cup L_{1*}}(E_1) \bowtie_{\theta} \Pi_{L_2 \cup L_{2*}}(E_2))$$

# Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

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

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

(but 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  $-$ .

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

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

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

12. The projection operation distributes over union

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

# 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.

# Example with Multiple Transformations

- Query: Find the names of all instructors in the Music department who have taught a course in 2017, along with the titles of the courses that they taught

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

- Transformation using join associatively (Rule 6a):

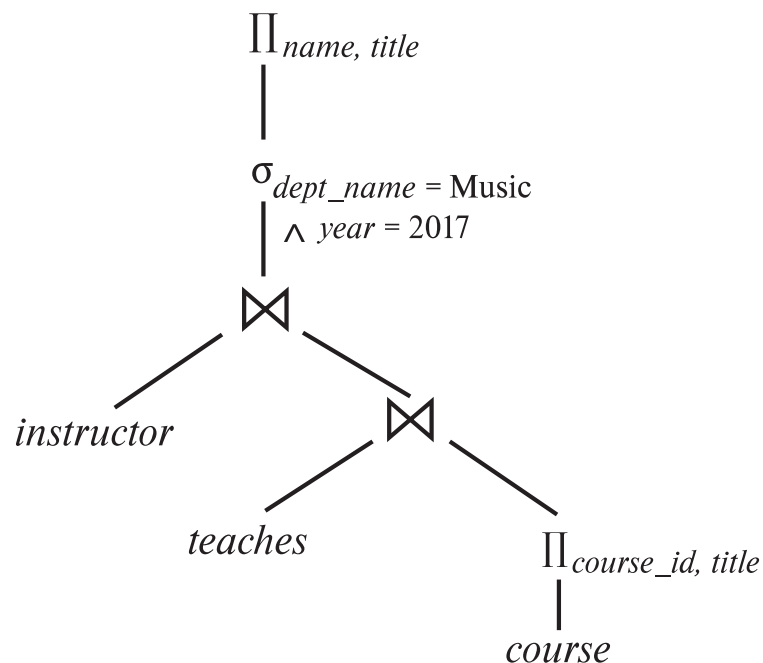
$$\Pi_{name, title}(\sigma_{dept\_name = 'Music' \wedge year = 2017} ((instructor \bowtie teaches) \bowtie \Pi_{course\_id, title} (course)))$$

- Now apply the "perform selections early" rule, resulting in the subexpression

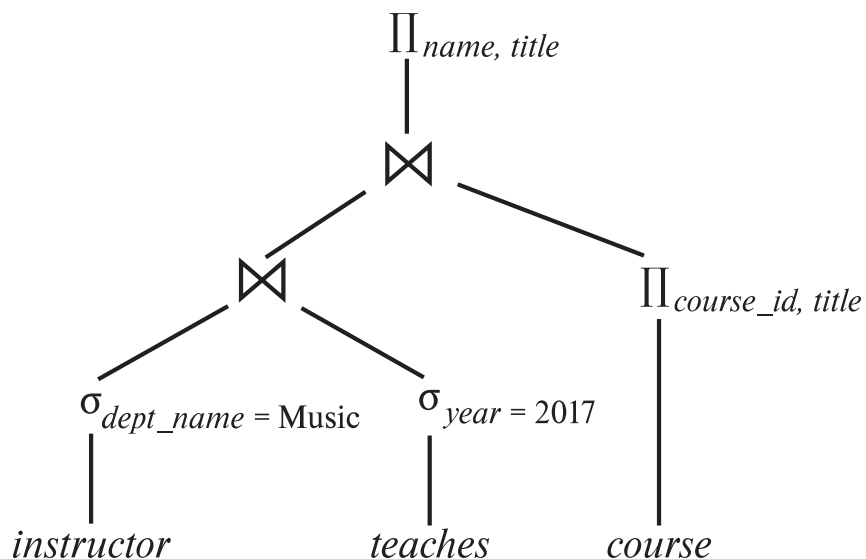
$$\sigma_{dept\_name = 'Music'} (instructor) \bowtie \sigma_{year = 2017} (teaches)$$



# Multiple Transformations (Cont.)



(a) Initial expression tree



(b) Tree after multiple transformations

# Transformation Example: Pushing Projections

- Consider:  $\Pi_{name, title}(\sigma_{dept\_name = 'Music'}(instructor) \bowtie teaches \bowtie \Pi_{course\_id, title}(course))$
- When we compute  $\sigma_{dept\_name = 'Music'}(instructor) \bowtie teaches$
- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:  
$$\Pi_{name, title}(\Pi_{name, course\_id}(\sigma_{dept\_name = 'Music'}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id, title}(course))$$
- 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

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

- Could compute  $(teaches \bowtie \Pi_{course\_id, title}(course))$  first, and join result with

$$\sigma_{dept\_name = 'Music'}(instructor)$$

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

- Only a small fraction of instructors are likely to be from the *Music* department

- it is better to compute

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

first.

# Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Could 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
  - It is not necessary to generate every possible expression
  - Take cost estimates into account; avoid examining many expressions

# Cost Estimation

- Cost of different operations described in previous 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

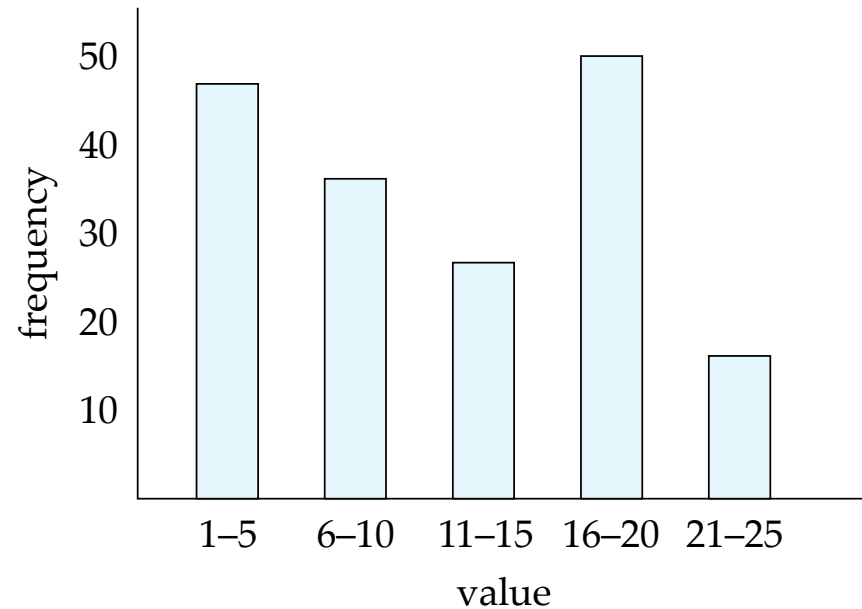
# Statistical Information for Cost Estimation

- $n_r$ : number of tuples in a relation  $r$ .
- $b_r$ : number of blocks containing tuples 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  $\Pi_A(r)$ .
- If tuples of  $r$  are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

# Histograms

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



- **Equi-width** histograms
- **Equi-depth** histograms break up range such that each range has (approximately) the same number of tuples
  - e.g. (4, 8, 14, 19)
- Some databases also store  $n$  **most-frequent values** and their counts
  - histogram is built on remaining values only



# Histograms (Cont.)

- Histograms and other statistics usually computed based on a **random sample**
- Statistics may be out of date
  - Some databases have a command to be executed to update statistics
  - Others automatically recompute statistics
    - e.g., when number of tuples in a relation changes by some percentage

# Selection Size Estimation

- $\sigma_{A=v}(r)$ 
  - Number of records that will satisfy the selection:  $n_r / V(A,r)$
  - Equality condition on a key attribute: *size estimate* = 1
- $\sigma_{A \leq v}(r)$  (case of  $\sigma_{A \geq v}(r)$  is symmetric)
  - Let  $c$  denote the estimated number of tuples satisfying the condition.
  - If  $\min(A,r)$  and  $\max(A,r)$  are available in catalog
    - $c = 0$  if  $v < \min(A,r)$
    - $c = n_r$  if  $v \geq \max(A,r)$
    - $c = n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
  - If histograms available, can refine above estimate
  - In absence of statistical information,  $c$  is assumed to be  $n_r / 2$ .

# Size Estimation of Complex Selections

- The **selectivity** of a condition  $\theta_i$  is the probability that a tuple in the relation  $r$  satisfies  $\theta_i$ .
  - If  $s_i$  is the number of satisfying tuples in  $r$ , the selectivity of  $\theta_i$  is:  $s_i/n_r$
- **Conjunction:**  $\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r)$ . Assuming independence, estimate of tuples in the result is:

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

- **Disjunction:**  $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$ . Estimated number of tuples:

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

- **Negation:**  $\sigma_{\neg \theta_1}(r)$ . Estimated number of tuples:

$$n_r * \left( 1 - \frac{s_1}{n_r} \right)$$

# Join Operation: Running Example

- Running example: *student* ⋈ *takes*
- Catalog information for join examples:
  - $n_{student} = 5000$
  - $f_{student} = 50$ , 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!)

# Estimation of the Size of Joins

- The Cartesian product  $r \times s$  contains  $n_r \cdot n_s$  tuples
- If  $R \cap S = \emptyset$ , then  $r \bowtie s$  is the same as  $r \times s$ .
- 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 in  $R$  referencing  $S$  is symmetric.
- In the example query  $student \bowtie takes$ ,  $ID$  in  $takes$  is a foreign key referencing  $student$ 
  - hence, the result has exactly  $n_{takes}$  tuples, which is 10000

# Estimation of the Size of Joins (Cont.)

- If  $R \cap S = \{A\}$  is not a key for  $R$  or  $S$ .

If we assume that every tuple  $t$  in  $R$  produces tuples in  $R \bowtie S$ , the number of tuples in  $R \bowtie S$  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.

- Can improve on above if histograms are available

# Estimation of the Size of Joins (Cont.)

- Compute the size estimates for  $student \bowtie takes$  without using information about foreign keys:
  - $n_{student} = 5000$ ,  $n_{takes} = 10000$ ,  $V(ID, takes) = 2500$ , and  $V(ID, student) = 5000$
  - The two estimates are:
    - $n_{student} * n_{takes} / V(ID, takes) = 5000 * 10000 / 2500 = 20000$
    - $n_{student} * n_{takes} / V(ID, student) = 5000 * 10000 / 5000 = 10000$
  - We choose the lower estimate, which is the correct one, because not every student takes courses (only half of them do)

# Size Estimation for Other Operations

- Projection: estimated size of  $\Pi_A(r) = V(A,r)$
- Set operations
  - For unions/intersections of selections on the same relation: rewrite and use size estimate for selections
    - e.g.,  $\sigma_{\theta_1}(r) \cup \sigma_{\theta_2}(r)$  can be rewritten as  $\sigma_{\theta_1 \vee \theta_2}(r)$
  - For operations on different relations:
    - estimated size of  $r \cup s$  = size of  $r$  + size of  $s$ .
    - estimated size of  $r \cap s$  = minimum size of  $r$  and size of  $s$ .
    - estimated size of  $r - s$  =  $r$ .
    - All the three estimates may be quite inaccurate, but provide upper bounds on the sizes.



# Estimation of Number of Distinct Values

Selections:  $\sigma_{\theta}(r)$

- If  $\theta$  forces  $A$  to take a specified value:  $V(A, \sigma_{\theta}(r)) = 1$ .
  - e.g.,  $A = 3$
- If  $\theta$  forces  $A$  to take on one of a specified set of values:  
 $V(A, \sigma_{\theta}(r)) = \text{number of specified values}$ .
  - e.g.,  $(A = 1 \vee A = 3 \vee A = 4)$
- If the selection condition  $\theta$  is of the form  $A \text{ op } v$   
estimated  $V(A, \sigma_{\theta}(r)) = V(A, r) * s/n_r$ 
  - where  $s/n_r$  is the selectivity of the selection.
- In all the other cases: use approximate estimate of  
 $\min\{V(A, r), n_{\sigma_{\theta}(r)}\}$

# Estimation of Distinct Values (Cont.)

Joins:  $r \bowtie s$

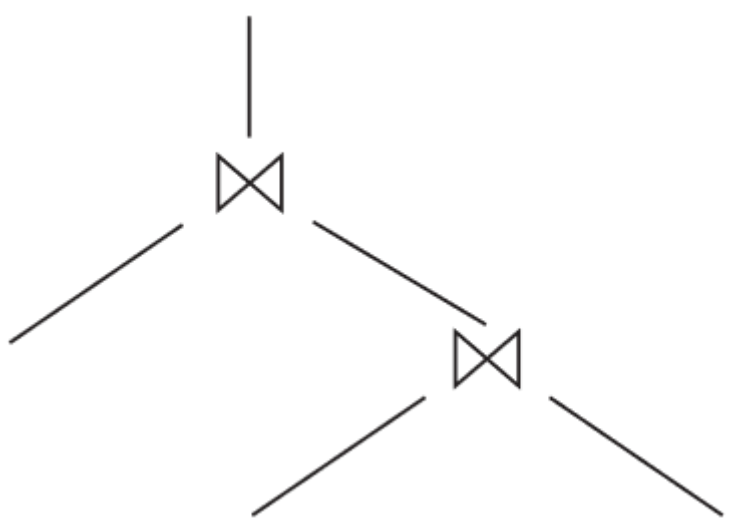
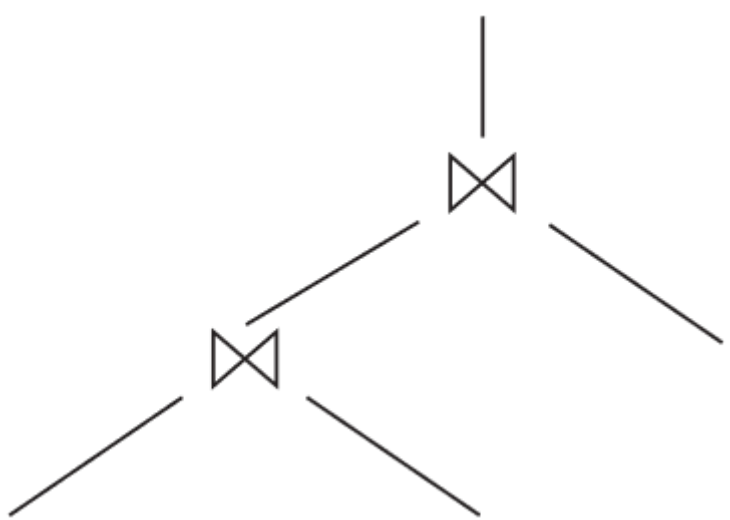
- If all attributes in  $A$  are from  $r$ 
  - estimated  $V(A, r \bowtie s) = \min\{V(A, r), n_{r \bowtie s}\}$
- If  $A$  contains attributes  $A1$  from  $r$  and  $A2$  from  $s$ 
  - estimated  $V(A, r \bowtie s) = \min\{V(A1, r) * V(A2 - A1, s),$   
 $V(A1 - A2, r) * V(A2, s),$   
 $n_{r \bowtie s}\}$

# 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.

# Cost-Based Optimization

- Consider finding the best join-order for  $r_1 \bowtie r_2 \bowtie r_3$

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

# Cost-Based Optimization (Cont.)

- Now consider finding the best join-order for:

$$(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$$

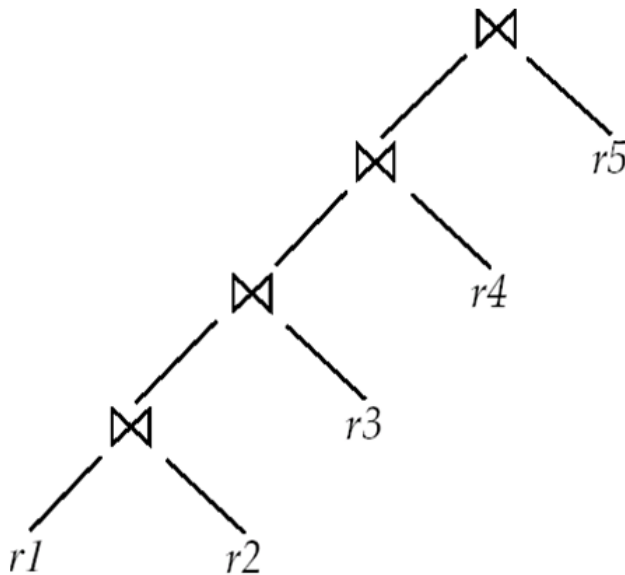
- There are 12 different join orders for  $r_1 \bowtie r_2 \bowtie r_3$  and another 12 orders for  $(...) \bowtie r_4 \bowtie r_5$
- Should we consider  $12 \times 12$  joins orders?
- No. Only  $12+12$ . We choose the best order for  $r_1 \bowtie r_2 \bowtie r_3$  and the best order for  $(...) \bowtie r_4 \bowtie r_5$  independently.
- When an optimization problem can be solved by optimizing sub-problems independently, we can use **dynamic programming**.

# Cost-Based Optimization (Cont.)

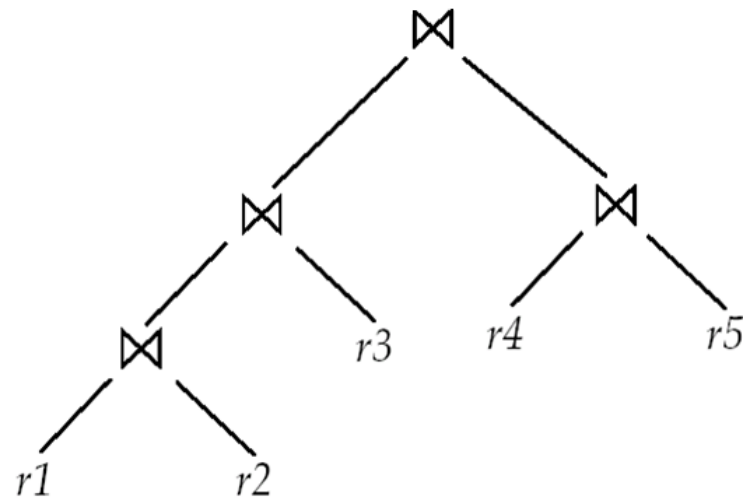
- Consider finding the best join-order for  $r_1 \bowtie r_2 \bowtie \dots \bowtie r_n$ .
- There are  $(2(n-1))!/(n-1)!$  different join orders for above expression. With  $n = 7$ , the number is 665280, with  $n = 10$ , the number is greater than 176 billion!
- No need to generate all the join orders. Using dynamic programming, the least-cost join order for any subset of  $\{r_1, r_2, \dots, r_n\}$  is computed only once and stored for future use.
- Find the best join order for every subset, by finding the best order for each subset of every subset, etc.
  - Do this recursively and reusing previously found solutions for each subset.

# Heuristics in Optimization

- Alternatively, use heuristics
  - E.g. in **left-deep join trees**, the right-hand-side input for each join is always a relation, not the result of an intermediate join.
  - Fewer join orders to consider.



(a) Left-deep join tree



(b) Non-left-deep join tree

# Heuristics in Optimization (Cont.)

- Cost-based optimization is expensive, even with dynamic programming.
- Systems may use *heuristics* to reduce the number of choices that must be made in a cost-based fashion.
- Many optimizers considers 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.



# Heuristics in Optimization (Cont.)

- Heuristic optimization transforms the query-tree by using 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 (i.e., with smallest result size) before other similar operations.
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

# Memoization and Plan Cache

- Besides the order of operations, there are multiple algorithms to choose from (e.g. hash join, nested-loop join, merge join)
  - These algorithms are **physically equivalent** (produce the same results)
  - Choice of best plan includes optimizing the query-tree (**equivalence rules**) and choosing the best algorithms (**physical equivalence rules**)
- Concept of **memoization**
  - Store the best plan for a subexpression the first time it is optimized, and reuse it on repeated optimization calls on same subexpression
- Implemented as **plan caching**
  - Reuse previously computed plan if query is resubmitted
  - Even with different constants in query

# Materialized Views

- A **materialized view** is a view whose contents are computed and stored.
- Consider the view:

```
create view my_students(ID, name) as  
select student.ID, student.name  
from student, takes  
where student.ID = takes.ID  
      and takes.course_id = 'CS-347';
```

- Materializing the above view would be very useful if the list of students is required frequently

# Materialized View Maintenance

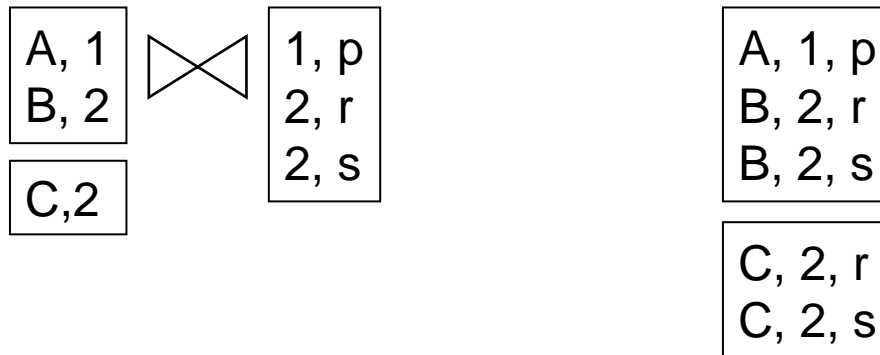
- The task of keeping a materialized view up-to-date with the underlying data is known as **materialized view maintenance**
- Materialized views can be maintained by recomputation on every update
- A better option is to use **incremental view maintenance**
  - **Changes to database relations are used to compute changes to the materialized view, which is then updated**
- View maintenance can be done by
  - Manually defining triggers on insert, delete, and update of each relation in the view definition
  - Manually written code to update the view whenever database relations are updated
  - Periodic recomputation (e.g. nightly)
  - Incremental maintenance supported by many database systems
    - Avoids manual effort/correctness issues

# Incremental View Maintenance

- The changes (inserts and deletes) to a relation or expressions are referred to as its **differential**
  - Set of tuples inserted to and deleted from  $r$  are denoted  $i_r$  and  $d_r$
- To simplify our description, we only consider inserts and deletes
  - We replace updates to a tuple by deletion of the tuple followed by insertion of the update tuple
- We describe how to compute the change to the result of each relational operation, given changes to its inputs
- We then outline how to handle relational algebra expressions

# Join Operation

- Consider the materialized view  $v = r \bowtie s$  and an update to  $r$
- Let  $r^{old}$  and  $r^{new}$  denote the old and new states of relation  $r$
- Consider the case of an insert to  $r$ :
  - We can write  $r^{new} \bowtie s$  as  $(r^{old} \cup i_r) \bowtie s$
  - And rewrite the above to  $(r^{old} \bowtie s) \cup (i_r \bowtie s)$
  - But  $(r^{old} \bowtie s)$  is simply the old value of the materialized view, so the incremental change to the view is just  $i_r \bowtie s$
- Thus, for inserts:  $v^{new} = v^{old} \cup (i_r \bowtie s)$
- Similarly for deletes:  $v^{new} = v^{old} - (d_r \bowtie s)$



# Selection Operation

- Selection: Consider a view  $v = \sigma_{\theta}(r)$ .
- We modify  $r$  by inserting a set of tuples  $i_r$  or deleting  $d_r$
- Then:
  - $v^{new} = v^{old} \cup \sigma_{\theta}(i_r)$
  - $v^{new} = v^{old} - \sigma_{\theta}(d_r)$

# Projection Operation

- Projection is a more difficult operation
  - $R = (A,B)$ , and  $r(R) = \{ (a,2), (a,3) \}$
  - $\Pi_A(r)$  has a single tuple  $(a)$ .
  - If we delete  $(a,2)$  from  $r$ , we should not delete the tuple  $(a)$  from  $\Pi_A(r)$ 
    - but if we then delete  $(a,3)$  as well, we should delete the tuple!
- For each tuple in a projection  $\Pi_A(r)$ , we will keep a count of how many times it was derived
  - On insert of a tuple to  $r$ , if the resultant tuple is already in  $\Pi_A(r)$  we increment its count, else we add a new tuple with count = 1
  - On delete of a tuple from  $r$ , we decrement the count of the corresponding tuple in  $\Pi_A(r)$ 
    - if the count becomes 0, we delete the tuple from  $\Pi_A(r)$



# Other Operations

- Set intersection:  $v = r \cap s$ 
  - when a tuple is inserted in  $r$  we check if it is present in  $s$ , and if so we add it to  $v$ .
  - If the tuple is deleted from  $r$ , we delete it from the intersection if it is present.
  - Updates to  $s$  are symmetric
  - The other set operations, *union* and *set difference* are handled in a similar fashion.

# Query Optimization and Materialized Views

- Rewriting queries to use materialized views:
  - A materialized view  $v = r \bowtie s$  is available
  - A user submits a query  $r \bowtie s \bowtie t$
  - We can rewrite the query as  $v \bowtie t$ 
    - Whether to do so depends on cost estimates for the two options
- Replacing a use of a materialized view:
  - A materialized view  $v = r \bowtie s$  is available
  - User submits a query  $\sigma_{A=10}(v)$  but the view has no index on  $A$
  - Suppose  $r$  has an index on  $A$ , and  $s$  has an index on the common attribute
  - Then the best plan may be to replace  $v$  by  $r \bowtie s$ , which can lead to the query plan  $\sigma_{A=10}(r) \bowtie s$
- Query optimizer should consider all above options and choose the best overall plan

# Materialized View Creation

- **Materialized view creation:** "What is the best set of views to materialize?"
- **Index creation:** "What is the best set of indices to create?"
  - closely related, but simpler
- Materialized view creation and index creation based on typical system **workload** (queries and updates)
  - Typical goal: minimize time to execute workload , subject to constraints on space and time taken for some critical queries/updates
  - One of the steps in database tuning (more on tuning in next lectures)
- Commercial database systems provide tools (called "tuning assistants" or "wizards") to help the database administrator choose what indices and materialized views to create.