Fakultas Informatika
School of Computing
Telkom University

Database System

05 | Query Processing and Optimization

Tahun Ajar Ganjil 2024/2025

Oleh:
Tim Dosen

Goals of the Meeting

01

Students knows the basic process of query processing and understand how to translate SQL Queries into Relational Algebra Expression (RAE)

02

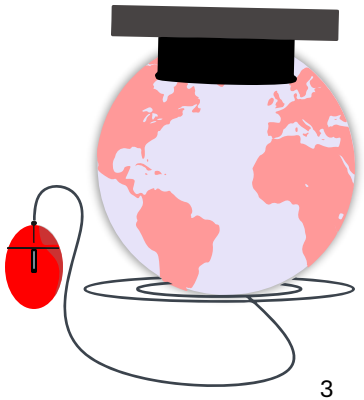
Students know various algorithms for selection and join operations and how to measures query costs

03

Students know various ways to optimize query processing, generate equivalent expressions, and execute an SQL statement to view query evaluation plans in DBMS

OUTLINES

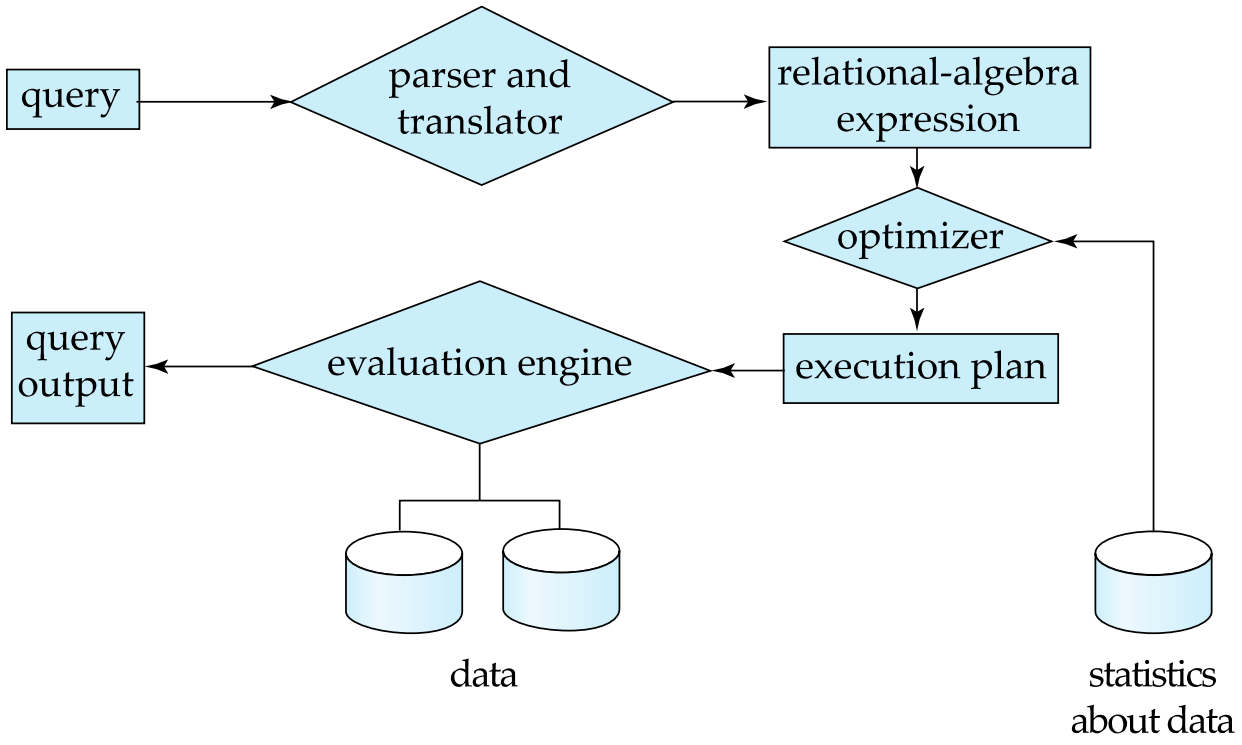
- Step of Query Processing
- Relational Algebra Expression (RAE)
- Query Cost
- Equivalen Rules Query Evaluation Plans on DBMS

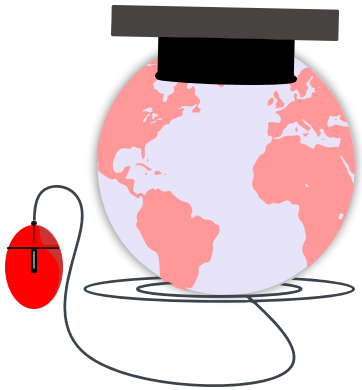


STEP OF QUERY PROCESSING

BASIC STEPS IN QUERY PROCESSING

1. Parsing and translation
2. Optimization
3. Evaluation





BASIC STEPS IN QUERY PROCESSING (CONT.)

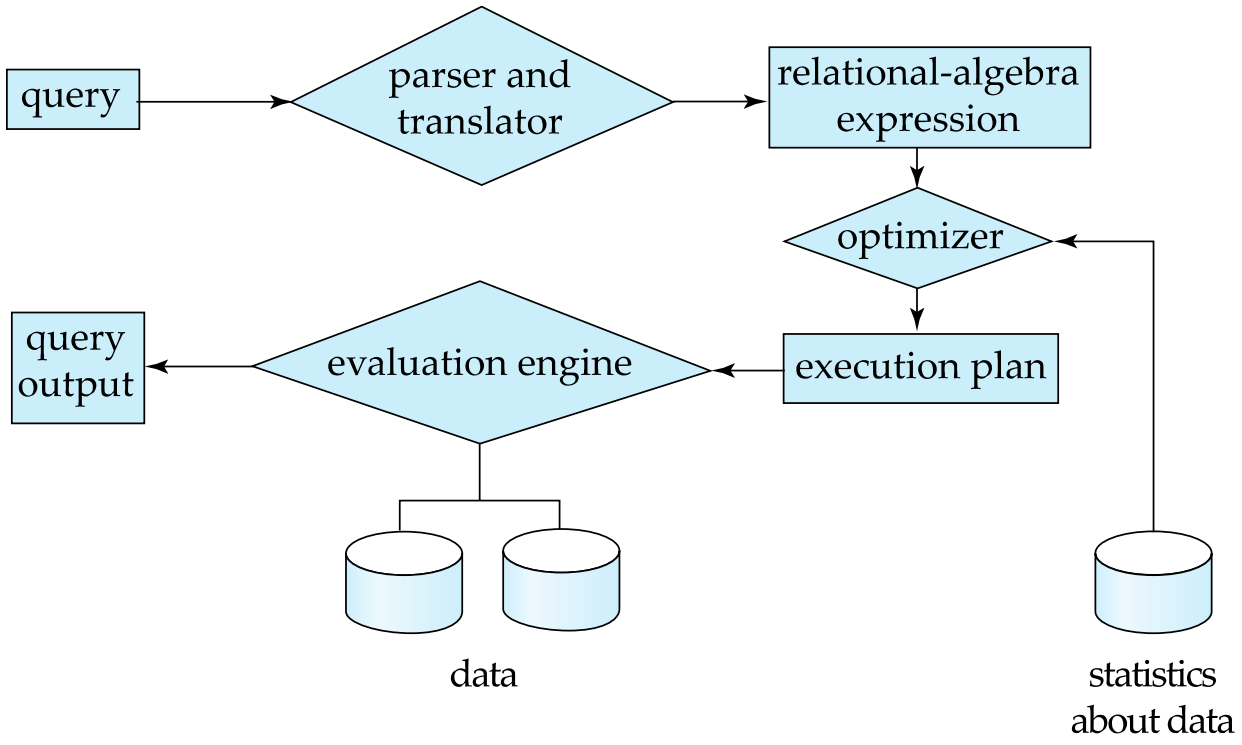
- Parsing and translation
  - translate the query into its internal form. This is then translated into **relational algebra**.
  - Parser checks syntax, verifies relations
- Optimization
  - Each relational algebra operation can be evaluated using one of several different algorithms
  - Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**
  - Amongst all equivalent evaluation plans choose the one with lowest cost.
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

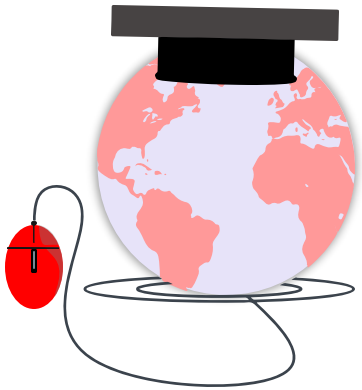
RELATIONAL ALGEBRA EXPRESSION

- PARSING & TRANSLATION QUERY TO RAE

BASIC STEPS IN QUERY PROCESSING

1. **Parsing and translation**
2. Optimization
3. Evaluation





RELATIONAL ALGEBRA

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Operators
  - select: σ
  - project: $\prod$
  - cartesian product: x
  - join: $\bowtie$
  - union: $\cup$
  - set-intersection: $\cap$
  - set-difference: –
  - assignment: $\leftarrow$
  - rename: ρ

SELECT OPERATION

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.
  - Query

    SELECT * FROM instructor WHERE dept_name = 'Physics'
  - Relational Algebra (RA)

    $\sigma_{dept\_name= "Physics"}(instructor)$
  - Result

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

SELECT OPERATION (CONT.)

- We allow comparisons using

  $=, \neq, >, \geq. <. \leq$

  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:

  $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
- Example: Find the instructors in Physics with a salary greater than $90,000, we write:
  - Query: **SELECT * FROM instructor WHERE dept_name = 'Physics' AND salary > 90000**
  - RA:

    $$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90,000}(instructor)$$

- The select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - Query: **SELECT * FROM department WHERE dept_name = building**
  - RA:

    $$\sigma_{dept\_name = building}(department)$$

PROJECT OPERATION

- A unary operation that returns its argument relation, with certain attributes left out.
- Notation:

$$\Pi_{A_1,A_2,A_3 \ldots A_k}(r)$$

  where A_1, A_2, ..., A_k are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

PROJECT OPERATION EXAMPLE

- Example: eliminate the *dept_name* attribute of *instructor*
- Query: **SELECT id, name, salary FROM instructor**
- RA:

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

- Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

COMPOSITION OF RELATIONAL OPERATIONS

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query -- Find the names of all instructors in the Physics department.
  - Query: **SELECT name FROM instructor WHERE dept_name = 'Physics'**
  - RAE:

$$\Pi_{name}(\sigma_{dept\_name = \text{“Physics”}} (instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

CARTESIAN-PRODUCT OPERATION

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:
  - Query: **SELECT * FROM instructor CROSS JOIN teaches**
    or **SELECT * FROM instructor, teaches**
  - RA:

    instructor* X *teaches
- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation (see next slide)
- Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*

THE *INSTRUCTOR* x *TEACHES TABLE*

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

JOIN OPERATION

- The Cartesian-Product

  instructor* X *teaches

  associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who did NOT teach a particular course.
- To get only those tuples of "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:
  - Query: **SELECT * FROM instructor, teaches WHERE instructor.id = teaches.id**
  - RAE:

    $\sigma_{instructor.id = teaches.id}$ **(*instructor* x *teaches*))**

  - We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

JOIN OPERATION (CONT.)

- The table corresponding to:

$\sigma_{instructor.id = teaches.id}$ (*instructor* x *teaches*))

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

JOIN OPERATION (CONT.)

- The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.
- Consider relations *r* (*R*) and *s* (*S*)
- Let "theta" be a predicate on attributes in the schema R "union" S. The join operation r $\bowtie_\theta$ s is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

Thus

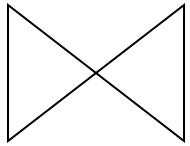
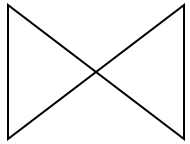
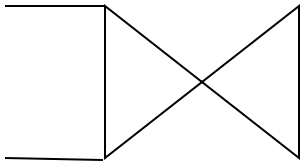
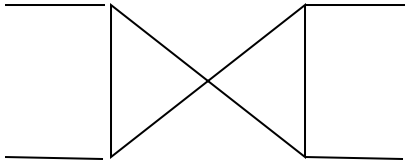
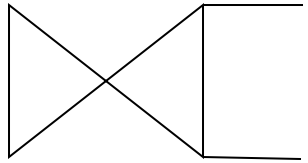
$$\sigma_{instructor.id = teaches.id} (instructor \times teaches))$$

Can equivalently be written as

- **SELECT * FROM instructor JOIN teaches ON instructor.id = teaches.id**

$$instructor \bowtie_{Instructor.id = teaches.id} teaches$$

JOIN OPERATION (CONT.)

- Natural join : 
- Inner join :  θ
- Outer join :   

Left Outer Join Full Outer Join Right Outer Join

RENAME OPERATION

- The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, ρ, is provided for that purpose
- The expression:

$$\rho_x (E)$$

returns the result of expression *E* under the name *x*

- Another form of the rename operation:

$$\rho_{x(A1,A2,..An)} (E)$$

returns the result of expression *E* under the name *x*, and with the attributes renamed to $A_1, A_2, ..., A_n$.

RENAME OPERATION EXAMPLE

- SELECT * FROM countries nation:

$$\rho_{nation}\ (countries)$$

- SELECT country_id AS id, country_name AS name, region_id FROM countries nation:

$$\rho_{nation\ (id,\ name,\ region\_id)}\ (countries)$$

- SELECT country_id AS id, country_name AS name FROM countries nation:

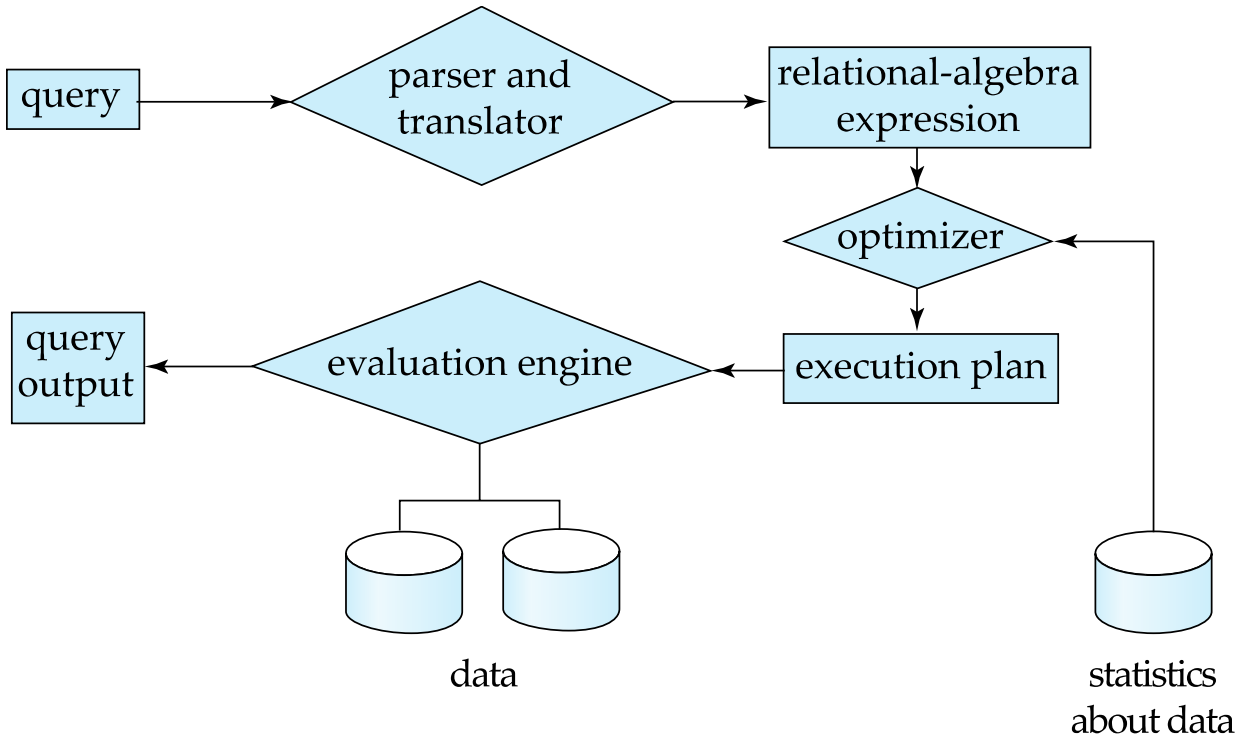
$$\Pi_{id,\ name}\ (\rho_{nation\ (id,\ name,\ region\_id)}\ (countries)\ )$$

QUERY OPTIMIZATION

- MEASURES OF QUERY COSTS
- SELECTION ALGORITHM

BASIC STEPS IN QUERY PROCESSING

1. Parsing and translation
2. **Optimization**
3. Evaluation



BASIC STEPS IN QUERY PROCESSING: OPTIMIZATION

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\prod_{salary}(instructor))$ is equivalent to $\prod_{salary}(\sigma_{salary<75000}(instructor))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**. E.g.,:
  - Use an index on *salary* to find instructors with salary < 75000,
  - Or perform complete relation scan and discard instructors with salary ≥ 75000

BASIC STEPS: OPTIMIZATION (CONT.)

- **Query Optimization**: Amongst all equivalent evaluation plans choose the one with lowest cost.
  - Cost is estimated using statistical information from the database catalog
    - e.g.. number of tuples in each relation, size of tuples, etc.
- In this chapter we study
  - **How to measure query costs**
  - **Algorithms for evaluating relational algebra operations**
  - **How to combine algorithms for individual operations in order to evaluate a complete expression**

MEASURES OF QUERY COST

- Many factors contribute to time cost
  - *disk access, CPU*, and network *communication*
- Cost can be measured based on
  - **response time**, i.e. total elapsed time for answering query, or
  - total **resource consumption**
- We use total **resource consumption** as cost metric
  - Response time harder to estimate, and minimizing resource consumption is a good idea in a shared database
- We **ignore CPU costs** for simplicity
  - Real systems do take CPU cost into account
  - Network costs must be considered for parallel systems
- We describe how estimate the cost of each operation
  - We do **not include cost to writing output to disk**

MEASURES OF QUERY COST

- Disk cost can be estimated as:
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
- For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - t_T – time to transfer one block
    - Assuming for simplicity that write cost is same as read cost
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks
    $b * t_T + S * t_S$
- t_S and t_T depend on where data is stored; with 4 KB blocks:
  - High end magnetic disk: t_S = 4 msec and t_T =0.1 msec
  - SSD: t_S = 20-90 microsec and t_T = 2-10 microsec for 4KB

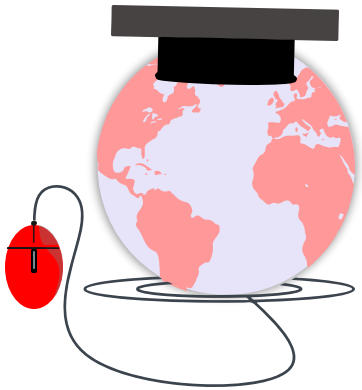
MEASURES OF QUERY COST (CONT.)

- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation
- Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
- Worst case estimates assume that no data is initially in buffer and only the minimum amount of memory needed for the operation is available
  - But more optimistic estimates are used in practice

SELECTIONS INVOLVING EQUALITY

COST ESTIMATES FOR SELECTION ALGORITHMS

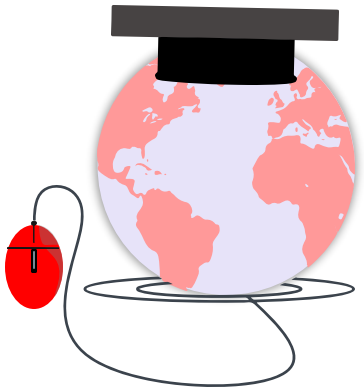
	Algorithm	Cost	Reason
A1	Linear Search	$t_S + b_r * t_T$	One initial seek plus b_r block transfers, where b_r denotes the number of blocks in the file.
A1	Linear Search, Equality on Key	Average case $t_S + (b_r/2) * t_T$	Since at most one record satisfies condition, scan can be terminated as soon as the required record is found. In the worst case, b_r blocks transfers are still required.
A2	Primary B^+-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	(Where h_i denotes the height of the index.) Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer.



SELECTIONS INVOLVING EQUALITY(2)

COST ESTIMATES FOR SELECTION ALGORITHMS

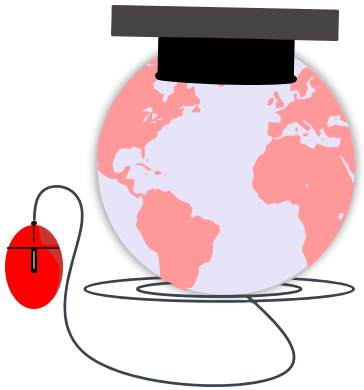
A3	Primary B$^+$-tree Index, Equality on Nonkey	$h_i * (t_T + t_S) + b * t_T$	One seek for each level of the tree, one seek for the first block. Here b is the number of blocks containing records with the specified search key, all of which are read. These blocks are leaf blocks assumed to be stored sequentially (since it is a primary index) and don't require additional seeks.
A4	Secondary B$^+$-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	This case is similar to primary index.
A4	Secondary B$^+$-tree Index, Equality on Nonkey	$(h_i + n) * (t_T + t_S)$	(Where n is the number of records fetched.) Here, cost of index traversal is the same as for A3, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if n is large.



SELECTIONS INVOLVING COMPARISONS

COST ESTIMATES FOR SELECTION ALGORITHMS

A5	Primary B$^+$-tree Index, Comparison	$h_i * (t_T + t_S) + b * t_T$	Identical to the case of A3, equality on nonkey.
A6	Secondary B$^+$-tree Index, Comparison	$(h_i + n) * (t_T + t_S)$	Identical to the case of A4, equality on nonkey.



JOIN OPERATION

- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
- Choice based on cost estimate
- Examples use the following information
  - Number of records of *student*: 5,000 *takes*: 10,000
  - Number of blocks of *student*: 100 *takes*: 400

NESTED-LOOP JOIN

- To compute the theta join $r \bowtie_\theta s$

  for each tuple t_r **in** r **do begin**
    for each tuple t_s **in** s **do begin**
      test pair (t_r, t_s) to see if they satisfy the join condition θ
      if they do, add $t_r \bullet t_s$ to the result.
    end
  end

- r is called the **outer relation** and s the **inner relation** of the join.
- Requires no indices and can be used with any kind of join condition.
- Expensive since it examines every pair of tuples in the two relations.

NESTED-LOOP JOIN (CONT.)

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is
  $n_r * b_s + b_r$ block transfers, plus $n_r + b_r$ seeks
- If the smaller relation fits entirely in memory, use that as the inner relation.
  - Reduces cost to $b_r + b_s$ block transfers and 2 seeks
- Assuming worst case memory availability cost estimate is
  - with *student* as outer relation:
    - 5000 * 400 + 100 = 2,000,100 block transfers,
    - 5000 + 100 = 5100 seeks
  - with *takes* as the outer relation
    - 10000 * 100 + 400 = 1,000,400 block transfers and 10,400 seeks
- If smaller relation (*student*) fits entirely in memory, the cost estimate will be 500 block transfers.
- Block nested-loops algorithm (next slide) is preferable.

BLOCK NESTED-LOOP JOIN

- Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.

for each block B_r **of** r **do begin**
  for each block B_s **of** s **do begin**
    for each tuple t_r **in** B_r **do begin**
      for each tuple t_s **in** B_s **do begin**
        Check if (t_r, t_s) satisfy the join condition
        if they do, add $t_r \bullet t_s$ to the result.
      end
    end
  end
end

BLOCK NESTED-LOOP JOIN (CONT.)

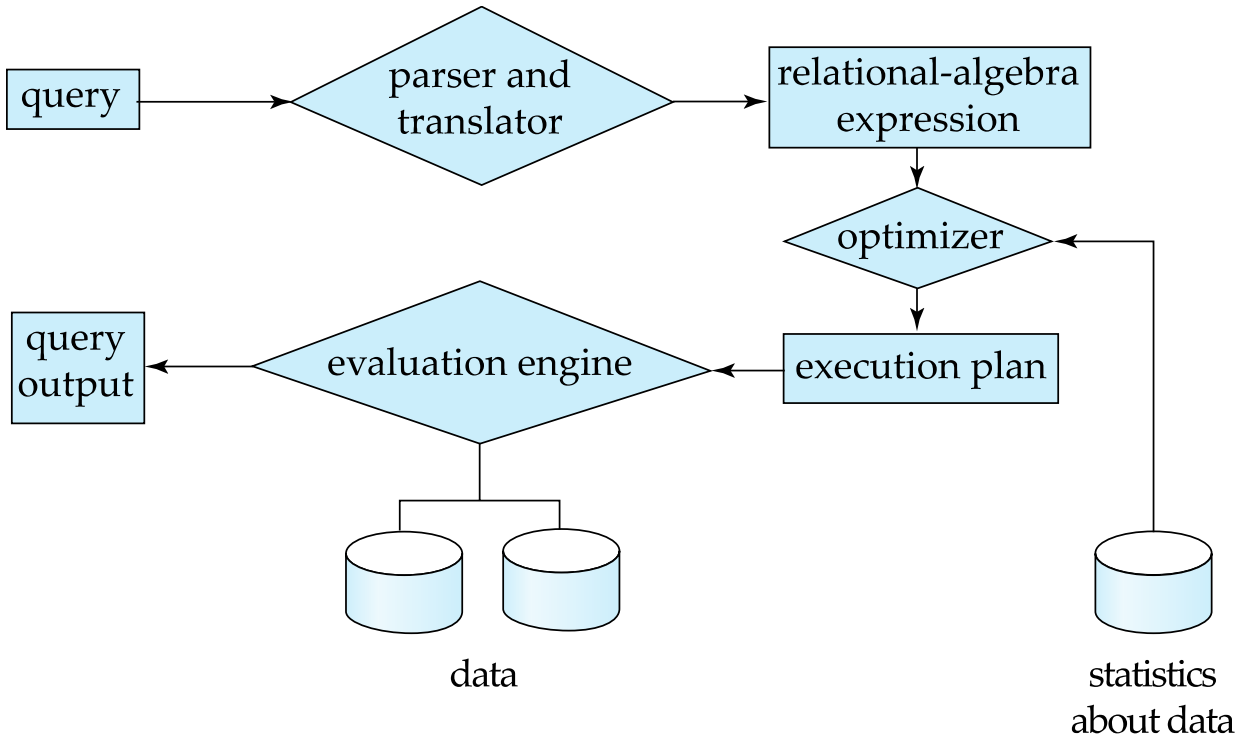
- Worst case estimate: $b_r * b_s + b_r$ block transfers + 2 * b_r seeks
  - Each block in the inner relation *s* is read once for each *block* in the outer relation
- Best case: $b_r + b_s$ block transfers + 2 seeks.
- Improvements to nested loop and block nested loop algorithms:
  - In block nested-loop, use *M* — 2 disk blocks as blocking unit for outer relations, where *M* = memory size in blocks; use remaining two blocks to buffer inner relation and output
    - Cost = $\lceil b_r / (M-2) \rceil * b_s + b_r$ block transfers + $2\lceil b_r / (M-2) \rceil$ seeks
  - If equi-join attribute forms a key or inner relation, stop inner loop on first match
  - Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
  - Use index on inner relation if available

QUERY EVALUATION

- EQUIVALENCE RULES

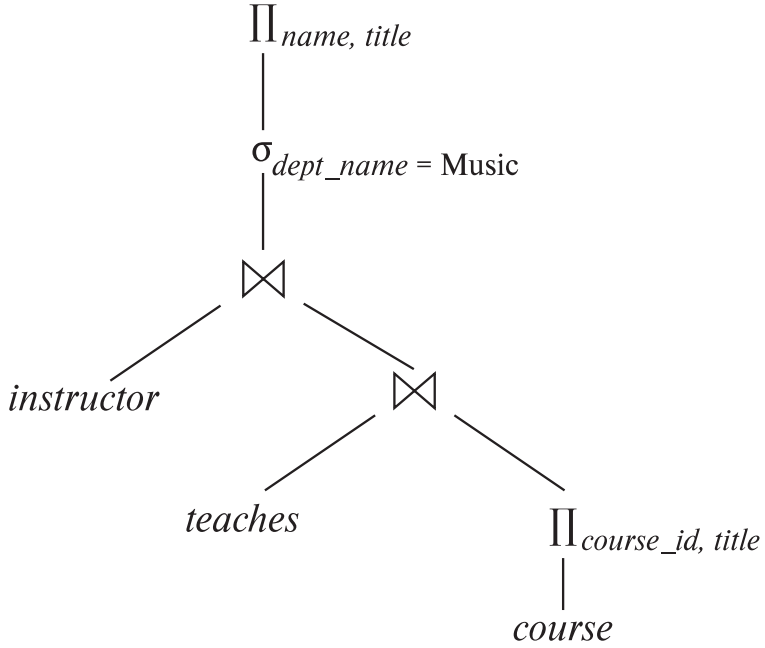
BASIC STEPS IN QUERY PROCESSING

1. Parsing and translation
2. Optimization
3. **Evaluation**

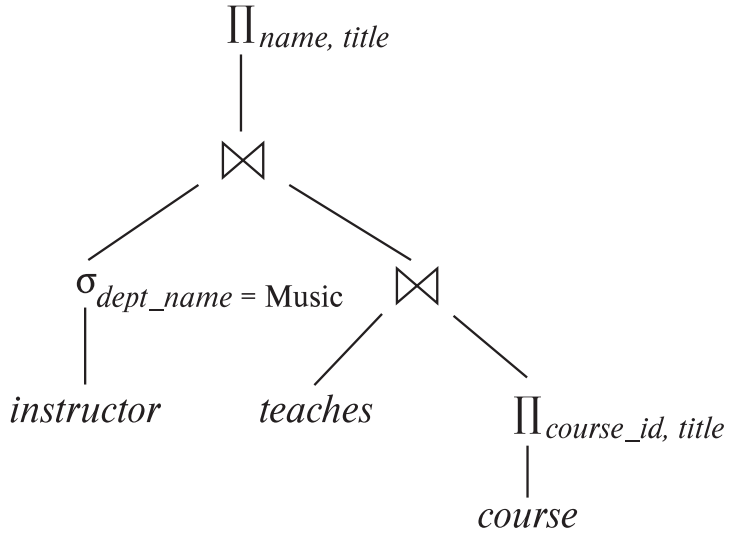


INTRODUCTION

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



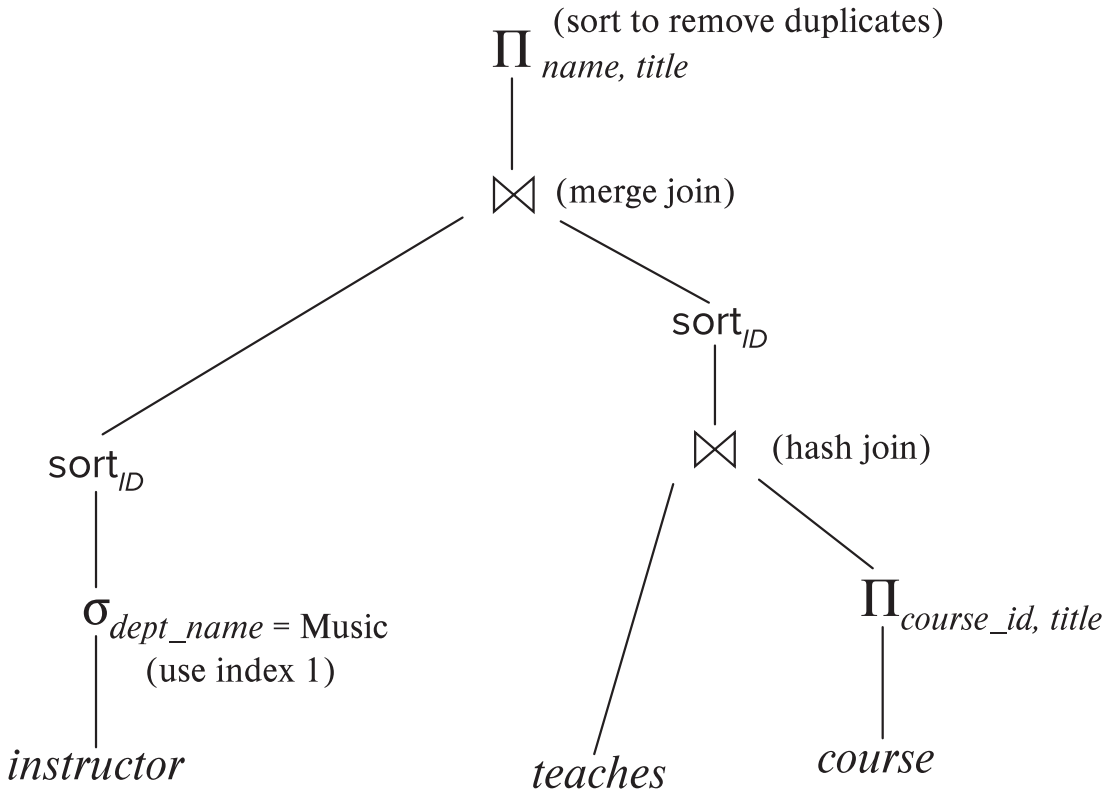
(a) Initial expression tree



(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.



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 *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.

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

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

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:

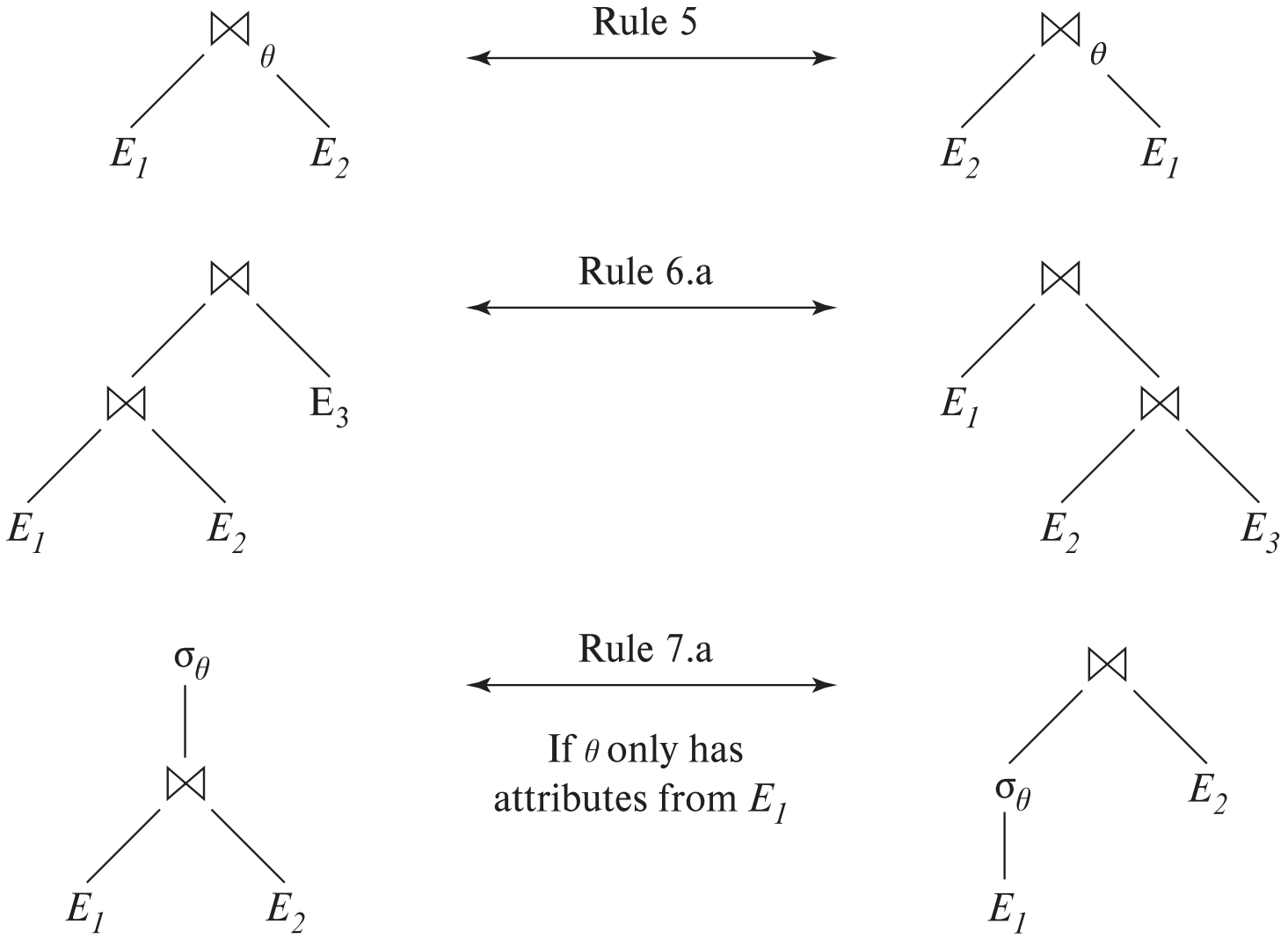
$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

(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.

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

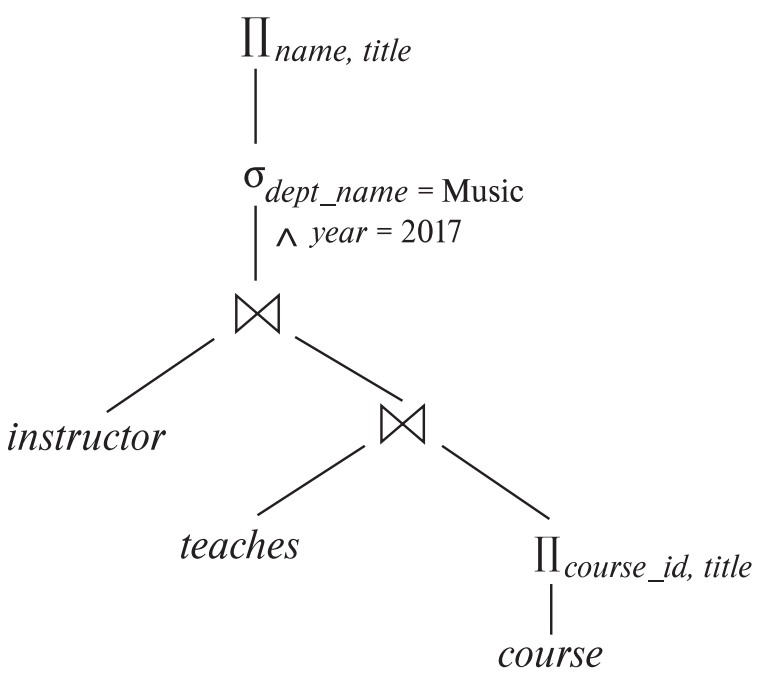
   (b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

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

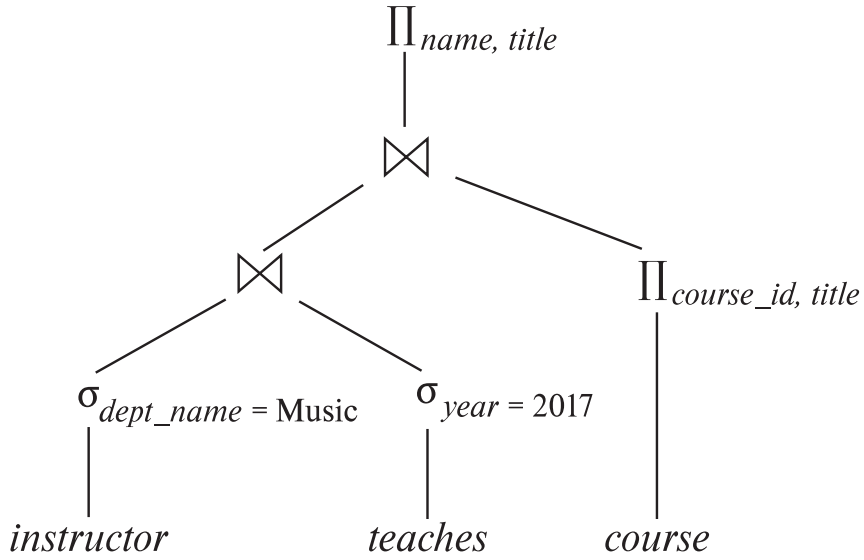
TRANSFORMATION EXAMPLE: PUSHING SELECTIONS

- 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.

MULTIPLE TRANSFORMATIONS



(a) Initial expression tree



(b) Tree after multiple transformations

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.

COST ESTIMATION

- Cost of each operator computer
  - 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

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.

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..l*
  - *f* is the cost of delivering first tuple and *l* is cost of delivering all results

REFERENCE

Silberschatz, Korth, and Sudarshan. *Database System Concepts* – 7th Edition. McGraw-Hill. 2019.

Slides adapted from Database System Concepts Slide.

Source: https://www.db-book.com/db7/slides-dir/index.html

ANY QUESTIONS?