Principles of Database Systems (CS307)

Lecture 9: Relational Algebra & Query Processing/Optimization

Ran Cheng

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Dr Yuxin Ma, Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Principles of Database Systems (CS307)

Lecture 9: Relational Algebra & Query Processing/Optimization

Ran Cheng

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Dr Yuxin Ma, Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Relational Algebra

Relational Algebra

- A procedural language consisting of a set of operations that take one or more relations as input and produce a new relation as their result.

- 6 Basic Operators:
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: $-$
  - Cartesian product: x
  - rename: ρ

Select Operation

- The select operation selects tuples that satisfy a given **predicate** (谓词：用来代替或者展示其客体性质、特征或者客体之间关系的词项)
  - Notation: $\sigma_p(r)$
  - p is called the selection predicate
  - r is a relation

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

- Example
  - Select those tuples of the *instructor* relation where the instructor is in the "Physics" department

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

select operation　selection predicate　relation

Select Operation

- 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 $90,000, we write:

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

- The select predicate may include comparisons between **two attributes**.
  - Example, find all departments whose name is the same as their building name:

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

Project Operation

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

Notation: $\prod_{A_1,A_2,A_3 \ldots A_k} (r)$

where A_1, A_2, ..., A_k are attribute names and r is a relation name.

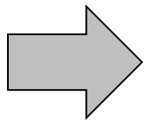
  - The result is defined as the relation of k columns obtained by erasing the columns that are not listed
  - **Duplicate rows removed** from result, since relations are sets

Project Operation

- Example: eliminate the *dept_name* attribute of instructor
  - Query: ⟹ not write its attributes' name at this place

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

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000



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, 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

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

  - Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation
  - How about in SQL expression?

Cartesian-Product Operation

$$A \times B = \{(x, y) \mid x \in A \wedge y \in B\}$$

∨ : or
¬ : not
∧ : and

- The Cartesian-product operation (denoted by ×) allows us to combine information from any two relations.
  - Example: the Cartesian product of the relations instructor and teaches is written as:

$$\text{instructor} \times \text{teaches}$$

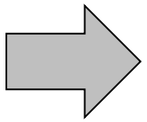
- We construct a tuple of the result out of each possible pair of tuples
  - … one from the instructor relation and one from the teaches relation (see next slide)
  - Since the instructor ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the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

The "instructor × teaches" table

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

×

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



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

- Problem: The Cartesian-Product "instructor × 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

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
...	...	...	...	...	...	...	...	...

Join Operation

- To get only those tuples of "instructor × teaches" that pertain to instructors and the courses that they taught, we write:

be appropriate, related, or applicable:
*matters **pertaining to** the organization of government.*

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

  - We get only those tuples of "instructor × teaches" that pertain (关于) to instructors and the courses that they taught
    - i.e., those tuples where instructor.id = teaches.id

$\sigma_p(r)$ p: predicate

Join Operation

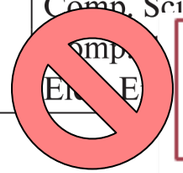
- The table corresponding to $\sigma_{instructor.id = teaches.id}(instructor \times teaches)$:

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

- The table corresponding to $\sigma_{instructor.id = teaches.id}(instructor \times teaches)$:

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.						
98345	Kim	El	E					



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
...	...	...	...	...	...	...	...	...

... will NOT include such tuples (rows) with different IDs

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
  - Example: The solution for the same question in the previous slide

- A little bit history:
  - `join` was introduced in SQL-1999 (later than this original way)

```sql
select m.title, c.credited_as,
       p.first_name, p.surname
from movies m,
     credits c,
     people p
where c.movieid = m.movieid
  and p.peopleid = c.peopleid
  and m.country = 'cn'
```

divided by logical operators. NOT equal sign

The SQL syntax was derived from the form of cartesian products in **relational algreba**

$$\sigma_{movies.id = credits.id \wedge people.peopleid = credits.peopleid \wedge movies.country = \text{“cn”}} (Movies \times Credits \times People)$$

and and

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
  - Example: The solution for the same question in the previous slide

- A little bit history:
  - `join` was introduced in SQL-1999 (later than this original way)

```
select m.title, c.credited_as,
       p.first_name, p.surname
from movies m,
     credits c,
     people p
where c.movieid = m.movieid
  and p.peopleid = c.peopleid
  and m.country = 'cn'
```

Use commas as the "multiplication signs"

The "select operation" is written as the "where" clause here

The SQL syntax was derived from the form of cartesian products in **relational algreba**

$$\sigma_{movies.id = credits.id \wedge \text{people.peopleid} = \text{credits.peopleid} \wedge \text{movies.country} = \text{“cn”}} (Movies \times Credits \times People)$$

Join Operation

- 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 (θ)" (参数化) 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:

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

Union Operation

- The union operation allows us to combine two relations
  - Notation: $r \cup s$
- For $r \cup s$ to be valid:
  - r, s must have the same arity (same number of attributes)
  - The attribute domains must be compatible
    - Example: 2nd column of r deals with the same type of values as does the 2nd column of s

Union Operation

- Example: To find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

section

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations
  - Notation: $r \cap s$
- Assume (same as Union):
  - r, s have the same arity
  - Attributes of r and s are compatible

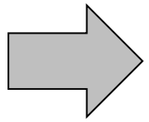
Set-Intersection Operation

- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters

$$\Pi_{course\_id}\,(\sigma_{semester=\text{"Fall"}\,\wedge\,year=2017}\,(section)) \cap \Pi_{course\_id}\,(\sigma_{semester=\text{"Spring"}\,\wedge\,year=2018}\,(section))$$

section

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A



course_id
CS-101

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another
  - Notation: $r - s$
- Assume (same as Union and Set Intersection):
  - r, s have the same arity
  - Attributes of r and s are compatible

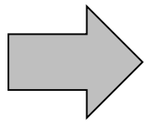
Set Difference Operation

- Example: Find all courses taught in the Fall 2017 semester, but NOT in the Spring 2018 semester

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) - \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

section

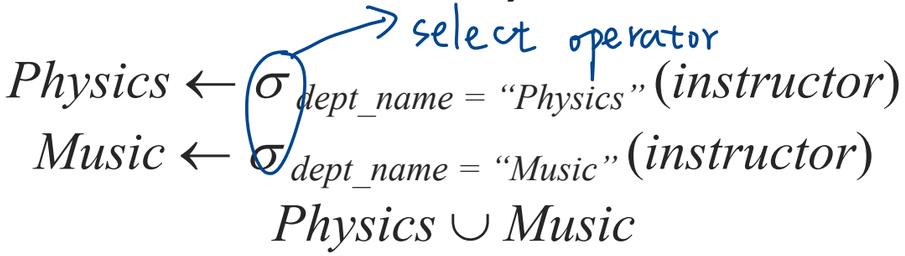
course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A



course_id
CS-347
PHY-101

The Assignment Operation

- It is convenient at times to write a relational-algebra **expression** by assigning parts of it to temporary relation variables
  - The assignment operation is denoted by ← and works like assignment in a programming language
- Example: Find all instructor in the "Physics" and Music department



  - With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

The 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 which also renames the columns:
    - $\rho_{x(A1,A2,..An)}(E)$

Equivalent Queries

- There is more than one way to write a query in relational algebra
  - Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000
    - Query 1

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

    - Query 2

$$\sigma_{dept\_name = \text{“Physics”}}(\sigma_{salary > 90.000}(instructor))$$

    - The two queries are not identical
      - they are, however, equivalent -- they give the same result on any database

Equivalent Queries

- Example: Find information about courses taught by instructors in the Physics department
  - Query 1

$$\sigma_{dept\_name = \text{“Physics”}} (instructor \bowtie_{instructor.ID = teaches.ID} teaches)$$

    - (Join first, then select)

  - Query 2

$$(\sigma_{dept\_name = \text{“Physics”}} (instructor)) \bowtie_{instructor.ID = teaches.ID} teaches$$

    - (Select first, then join)

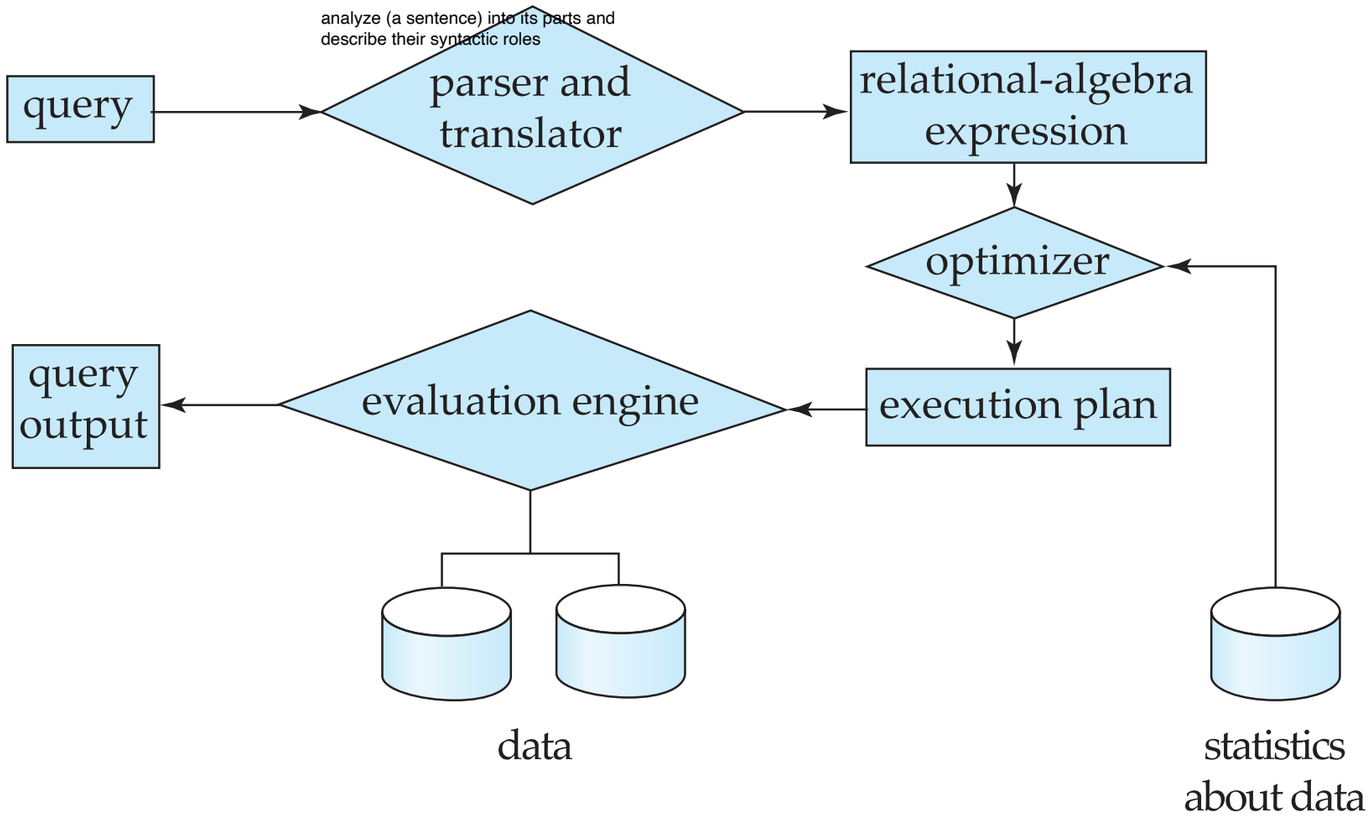
Equivalent Queries

- Application of Relational Algebra: Query Optimization
  - Transform queries into equivalent ones with less computational cost

Query Processing

Basic Steps in Query Processing

- Parsing and translation: translate the query into its internal form, and then into relational algebra.
- Evaluation: the query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.



Basic Steps in Query Processing

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\Pi_{salary}(instructor))$ is equivalent to $\Pi_{salary}(\sigma_{salary<75000}(instructor))$

- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.

CT

add notes to (a text or diagram) giving explanation or comment

- 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

Query Optimization

a·mong | əˈməNG | (*mainly British* also amongst

- **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.
- We will 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

Measure of Query Cost

- Many factors contribute to time cost
  - *disk access*, *CPU*, and network *communication*
- Cost can be measured based on
  - **response time (latency)**, *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
  - Main cost is from the **hard-disk**

Measure 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

- Let's take ***selection*** operation as an example

Measure of Query Cost

- For simplicity we just use the **number of block transfers（read/write）** *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

- Let's take ***selection*** operation as an example

Example: Selection Operation

- **File scan**
- Algorithm: **Linear Search**. Scan each file block and test all records to see whether they satisfy the selection condition.
  - Cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation *r*
  - Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices
- Note: binary search generally does NOT make sense since data is not stored consecutively
  - except when there is an index available and binary search requires more seeks than index search

Example: Selection Operation

- Faster ways?
- **Index scan** – search algorithms that use an index
  - Selection condition must be on search-key of index

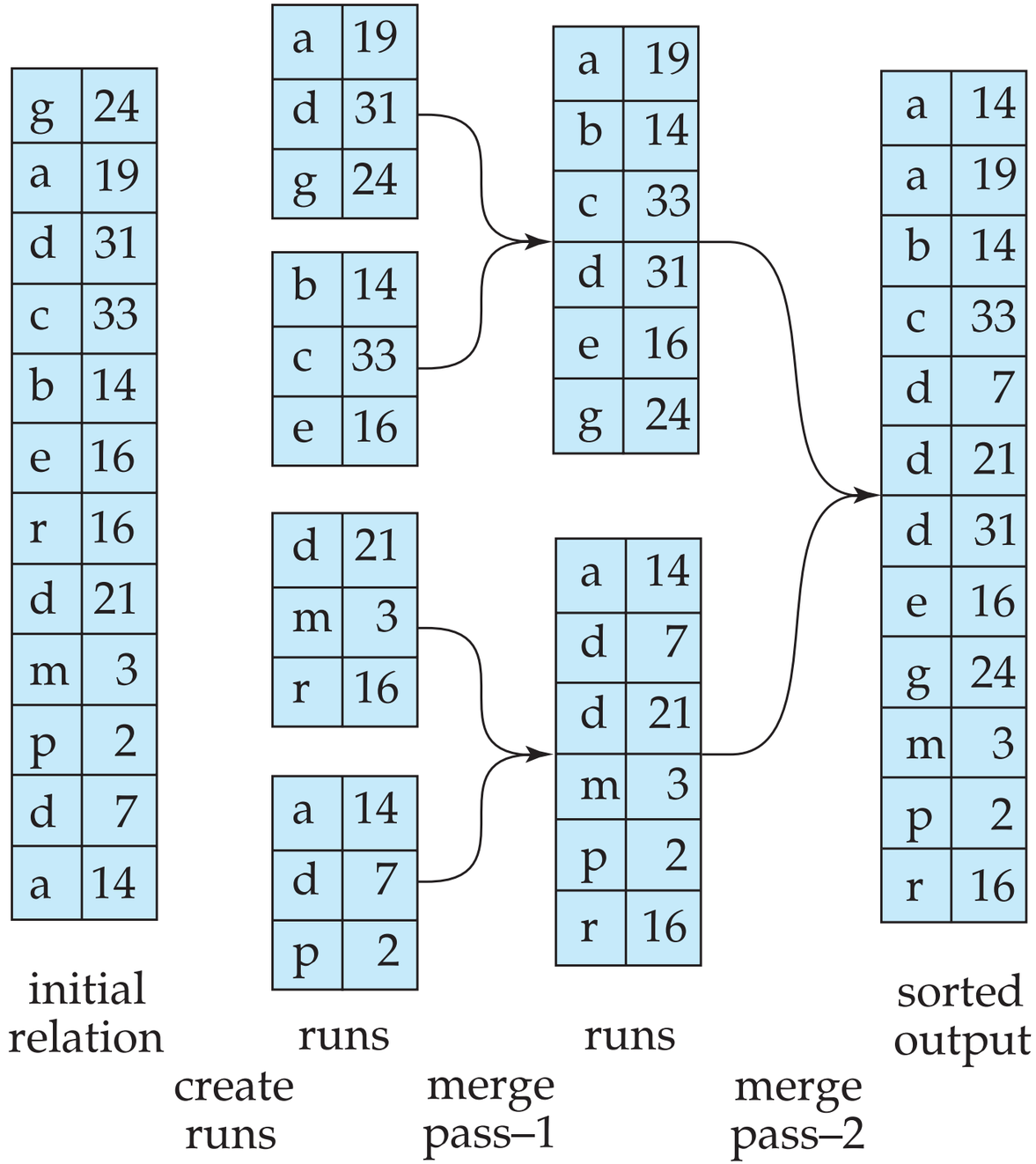
- We may build an index on the relation, and then use the index to read the relation in ***sorted*** order.
  - For relations that fit in memory, techniques like **quicksort** can be used.
  - For relations that do NOT fit in memory, **external merge-sort** is a good choice. Why?

Example: Selection Operation Merge-Sort

- All intermediate results can be stored in the buffer space
- Disk is cheap for the additional buffer space
- Slower, but scalable – **batch**

able to be measured or graded according to a scale.

a quantity or consignment of goods produced at one time: ***a batch of*** *cookies*



Example: Selection Operation
Merge-Sort

Let *M* denote **batch size** (in blocks).

1. **Create sorted runs**. Let *i* be 0 initially.
   Repeatedly do the following till the end of the relation:
   (a) Read *M* blocks of relation into memory
   (b) Sort the in-memory blocks
   (c) Write sorted data to run R_i; increment *i*.

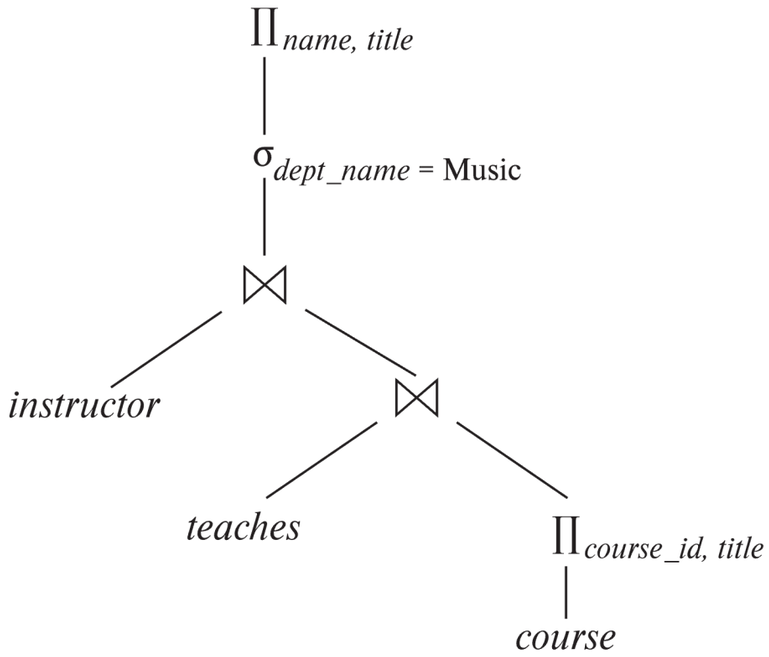
Let the final value of *i* be *N*

2. ***Merge*** *the runs – How?*

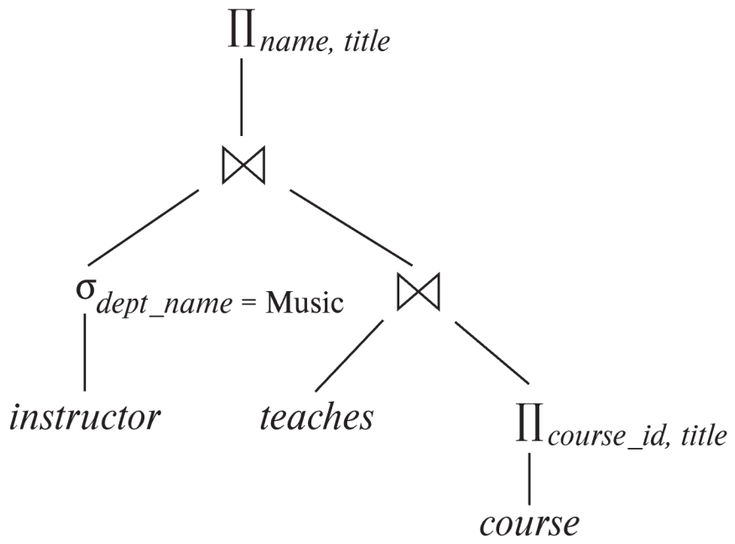
Query Optimization

Evaluating a given query

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
  - One expression -> multiple ways to evaluate



(a) Initial expression tree

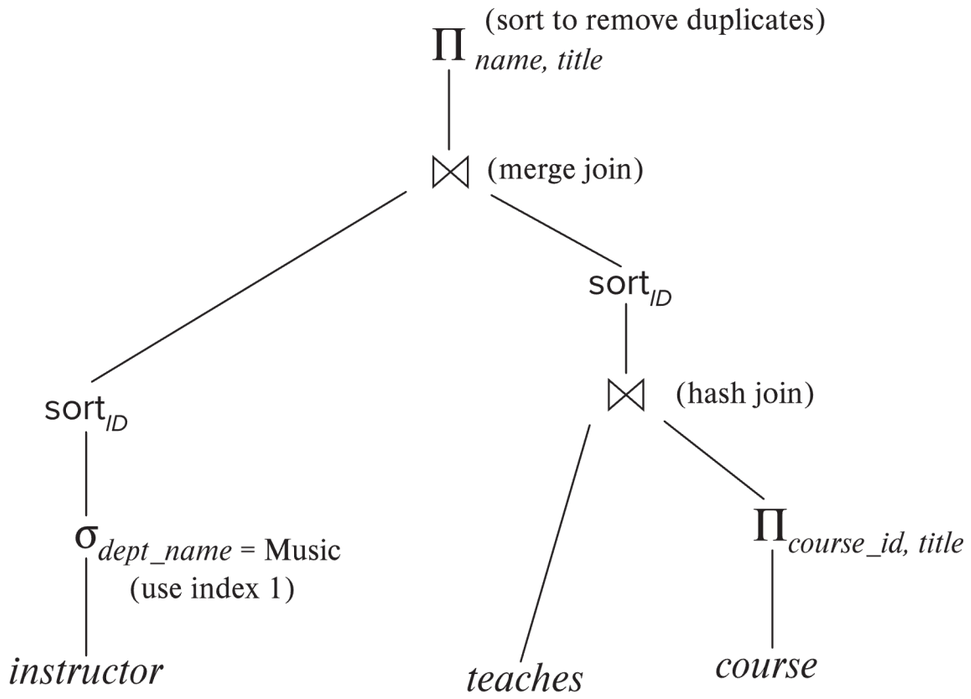


(b) Transformed expression tree

Evaluation plan

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

bring the different elements of (a complex activity or organization) into a relationship that will ensure efficiency or harmony



Query optimization via evaluation plans

- 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 evaluation 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
  - …
- Result of optimization – **Query plan**:
  - Postgre -- EXPLAIN

More to read

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

Part 5: Querying		
15. Query Processing	pptx, pdf	Feb 5, 2019
16. Query Optimization	pptx, pdf	Mar 8, 2019