Fakultas Informatika
School of Computing
Telkom University

CSH2D3 - Database System

05 | Query Processing

Goals of the Meeting

01

Students knows the basic process of query processing

02

Students understand how to translate SQL Queries into Relational Algebra Expression (RAE)

Outline

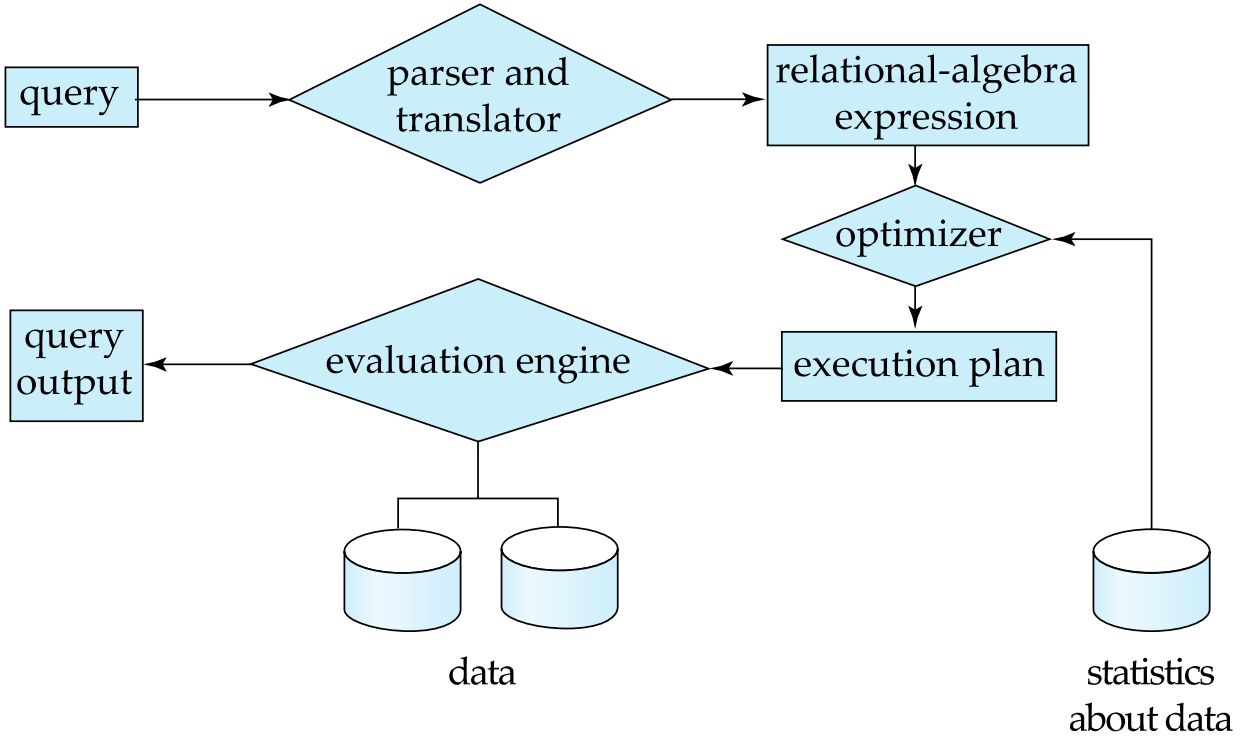
Steps of Query Processing

Relational Algebra Expression

Steps 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

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:

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

- 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:

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

  - 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:

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

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 (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:

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

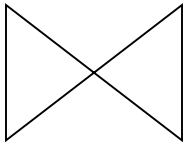
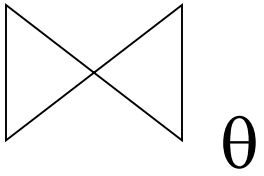
- Thus

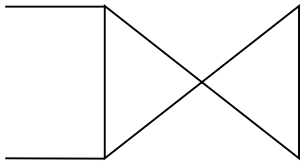
$$\sigma_{instructor.id = teaches.id} (instructor \times teaches))$$

- Can equivalently be written as
  - **SELECT * FROM instructor JOIN teaches ON instructor.id = teaches.id**

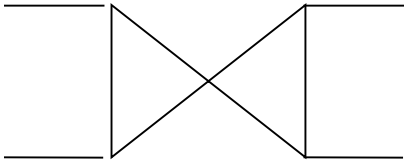
$$instructor \bowtie_{Instructor.id = teaches.id} teaches$$

Join Operation (Cont.)

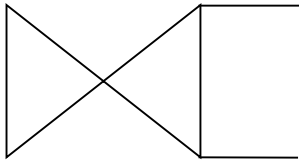
- Natural join : 
- Inner join : 
- Outer join :



Left Outer Join



Full Outer Join



Right Outer Join

Union Operation

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. 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*)
- Example: to find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both
  - Query: **SELECT course_id FROM section WHERE semester='Fall' AND year=2017 UNION SELECT course_id FROM section WHERE semester='Spring' AND year=2018**
  - RAE:

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

Union Operation (Cont.)

- Result of:

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

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:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.
  - Query: **SELECT course_id FROM section WHERE semester='Fall' AND year=2017 INTERSECT SELECT course_id FROM section WHERE semester='Spring' AND year=2018**
  - RAE:

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

  - Result

course_id
CS-101

Set-Intersection Operation Example 2

Relation r

country_name	region_id
US	America
Indonesia	Asia
Canada	America
Spain	Europe
England	Europe

Relation *s*

country_name	region_id
Indonesia	Asia
Spain	Europe
England	Europe
Italy	Europe
Thailand	Asia
South Africa	Africa

r ∩ *s*

country_name	region_id
Indonesia	Asia
Spain	Europe
England	Europe

Set-Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$
- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester
  - Query: **SELECT course_id FROM section WHERE semester='Fall' AND year=2017 MINUS SELECT course_id FROM section WHERE semester='Spring' AND year=2018**
  - RAE:

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

course_id
CS-347
PHY-101

Set-Difference Operation Example 2

Relation r

country_name	region_id
US	America
Indonesia	Asia
Canada	America
Spain	Europe
England	Europe

Relation *s*

country_name	region_id
Indonesia	Asia
Spain	Europe
England	Europe
Italy	Europe
Thailand	Asia
South Africa	Africa

r - *s*

country_name	region_id
US	America
Canada	America

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.

$$Physics \leftarrow \sigma_{dept\_name="Physics"}(instructor)$$
$$Music \leftarrow \sigma_{dept\_name="Music"}(instructor)$$
$$Physics \cup Music$$

- 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.

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 $A_1, A_2, \ldots, A_n$.

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:

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

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

- 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
- Query 1

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

- Query 2

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

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


Any
Questions

Exercises

Given the employee database as follow:

employee **(empID, *person_name*, *street*, *city*)**

works **(emp*ID*, *compID*, *salary*)**

company **(*compID, company_name*, *city*)**

Give an expression in the relational algebra to express each of the following queries:

1. Find the name and city of each employee who does not lives in “Miami”
2. Find the name of each employee whose salary is greater than equal to $100000.
3. Find the name and salary of each employee whose salary is between $50000 and $100000.
4. Find the name of each employee who lives in “Miami” or whose salary is lower than $100000.
5. Find the company name and name of each employee who does not work for “BigBank”.
6. Find the company name, city, and name of each employee who lives in the same city as the company for which she or he works.

References

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