Lecture 6

Relational Algebra

COMP3278A

Introduction to Database Management Systems

Dr. Ping Luo
Email : pluo@cs.hku.hk

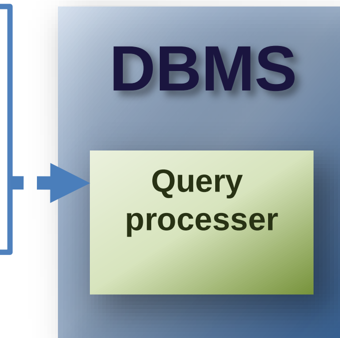
Department of Computer Science, The University of Hong Kong

Acknowledgement: Dr Chui Chun Kit

Motivation

- Find the dept. names where employees named Smith work.

```
SELECT D.name
FROM Employees E, Works_in W, Departments D
WHERE E.name = 'Smith'  AND
        E.employee_id = W.employee_id AND
        W.department_id = D.department_id;
```



How does the DBMS execute this SQL query?
e.g. Join which two tables first?
Which **constraint** is applied first?

Employees

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id	since
1	1	2001-1-1
2	1	2002-4-1
2	2	2005-2-2
3	3	2003-1-1
4	3	2005-1-1

Departments

department_id	name	budget
1	Toys	122000
2	Tools	239000
3	Food	100000

Relational Algebra

- **Relational Algebra** is similar to normal algebra (as in 2+3*x-y), except it uses **relations (tables)** as **operand**, and a new set of **operators**.
- The inner, lower-level operations of a relational DBMS are (or are similar to) relational algebra operations.
- **We need to know about relational algebra to understand query execution and optimization in a relational DBMS.**

Section 1

Basic operators

Basic operators

- Select (σ)
- Project (π)
- Union (∪)
- Set difference (-)
- Cartesian product (×)
- Rename (ρ)

That is to say, there should be programs (or functions) implemented in the DBMS for each of these relational operators

5

Selection

- $\sigma_p(R) = \{ t \mid t \in R \wedge p(t) \}$
- Example
  - Consider the relation **Author** (<u>authorID</u>, name, date of birth)
  - Select all authors called “May”.

```
SELECT *
FROM Author
WHERE name = "May";
```

SQL

→ Query processer →

$\sigma_{name=\text{“May”}}$(**Author**)

Relational algebra

Author

authorID	name	date of birth
101	May	Nov 16
102	Bonnie	Jan 15
103	May	Jul 11
104	Raymond	Apr 30
105	Tiffany	Oct 10

$\sigma_{name=\text{“May”}}$(**Author**)

authorID	name	date of birth
101	**May**	Nov 16
103	**May**	Jul 11

Projection

- $\pi_{A1, A2, \ldots, Ak}(R)$

  The result = relation over the k attributes $A1, A2, \ldots, Ak$ obtained from R by erasing the columns that are not listed and eliminating duplicate rows

  - A copy of R with only listed attributes A1 to Ak.
- Example
  - Consider the relation **Book** (<u>bookID</u>, title, publisher)
  - Report only the **bookID** and **title** of all the books.

```
SELECT bookID, title
FROM Book;
```

SQL

→ Query processer →

$\pi_{bookID,\ title}(\textbf{Book})$

Relational algebra

Book

bookID	title	publisher
115	Stuffy doll	ABC
116	Angel's feather	MTG
117	Little girl	MGH
118	Myr	ABC

$\pi_{bookID,\ title}(\textbf{Book})$

bookID	title
115	Stuffy doll
116	Angel's feather
117	Little girl
118	Myr

Union

- $R \cup S = \{ t \mid t \in R \vee t \in S \}$
  - R and S must have the **same number of attributes** and **attribute data types are compatible**.
- Example
  - Find the name of all products in Audio_CD and DVD tables.

```
SELECT name
FROM Audio_CD
UNION
SELECT name
FROM DVD;
```

SQL

→ Query processer →

$\pi_{name}(\textbf{Audio\_CD}) \cup \pi_{name}(\textbf{DVD})$

Relational algebra

Audio_CD

name	#tracks
One Heart	14
Miracle	14

DVD

name	length	subtitle
Prince of Persia	110	English, Chinese
Villon’s Wife	90	Japanese
Legend is born: Ip Man	90	Chinese

Union

π_{name} (Audio_CD) ∪ π_{name} (DVD)

name
One Heart
Miracle
Prince of Persia
Villon’s Wife
Legend is born: Ip Man

Note that the left and right part of the Union operator has to be **comparable**. (i.e. same **type**, same number of attributes)

π_{name} (Audio_CD)

name
One Heart
Miracle

π_{name} (DVD)

name
Prince of Persia
Villon’s Wife
Legend is born: Ip Man

Audio_CD

name	#tracks
One Heart	14
Miracle	14

DVD

name	length	subtitle
Prince of Persia	110	English, Chinese
Villon’s Wife	90	Japanese
Legend is born: Ip Man	90	Chinese

Set difference

- R-S = { t | t ∈ R ∧ t ∉ S }
  - R and S must have the **same number of attributes** and **attribute data types are compatible**.
- Example
  - Find the ID of the students who haven't submitted the assignment.

```
SELECT student_id
FROM Student
EXCEPT
SELECT student_id
FROM Submit;
```

SQL

→ **Query processer** →

$$\pi_{student\_id}(\textbf{Student}) - \pi_{student\_id}(\textbf{Submit})$$

Relational algebra

Student

student_id	name	gender	major
123	Kit	M	CS
456	Yvonne	F	CS
789	Paul	M	CS

Submit

student_id	assignment_id	date
456	1	28/9
789	1	25/9

Set difference

$\pi_{student\_id}(\textbf{Student}) - \pi_{student\_id}(\textbf{Submit})$

student_id
123

Note that the left and right part of the Set difference operator has to be **comparable**. (i.e. same **type**, same number of attributes)

$\pi_{student\_id}(\textbf{Student})$

student_id
123
456
789

$\pi_{student\_id}(\textbf{Submit})$

student_id
456
789

Student

student_id	name	gender	major
123	Kit	M	CS
456	Yvonne	F	CS
789	Paul	M	CS

Submit

student_id	assignment_id	date
456	1	28/9
789	1	25/9

Cartesian product

- $R \times S = \{ t\,q \mid t \in R \wedge q \in S \}$
  - No attributes with a common name in R and S.
- Example
  - Display the date of the tutorials of the course "Introduction to Database Management Systems".

```
SELECT Tutorial.date
FROM Course, Tutorial
WHERE
Coruse.name="Introduction to
Database Management Systems" AND
Course.course_id = Tutorial.course_id;
```

SQL

Query processer

$$\pi_{\text{Tutorial.date}}\Big(\sigma_{\text{Course.name="Introduction to Database Management Systems"}}\big(\sigma_{\text{Course.course\_id=Tutorial.course\_id"}}(\textbf{Course} \times \textbf{Tutorial})\big)\Big)$$

Relational algebra

Course

course_id	name
c1119	Data Structures and Algorithms
c0278a	Introduction to Database Management Systems

Tutorial

tutorial_id	course_id	date
1	c1119	5/9
1	c0278a	7/9
2	c0278a	15/9

Cartesian product

Course × Tutorial

Course.course_id	Course.name	Tutorial .tutorial_id	Tutorial .course_id	Tutorial .date
c1119	Data Structures and Algorithms	1	c1119	5/9
c1119	Data Structures and Algorithms	1	c0278a	7/9
c1119	Data Structures and Algorithms	2	c0278a	15/9
c0278a	Introduction to Database Management Systems	1	c1119	5/9
c0278a	Introduction to Database Management Systems	1	c0278a	7/9
c0278a	Introduction to Database Management Systems	2	c0278a	15/9

↑

Course

course_id	name
c1119	Data Structures and Algorithms
c0278a	Introduction to Database Management Systems

Tutorial

tutorial_id	course_id	date
1	c1119	5/9
1	c0278a	7/9
2	c0278a	15/9

Cartesian product

Course × Tutorial

Course.course_id	Course.name	Tutorial .tutorial_id	Tutorial .course_id	Tutorial .date
c1119	Data Structures and Algorithms	1	c1119	5/9
c1119	Data Structures and Algorithms	1	c0278a	7/9
c1119	Data Structures and Algorithms	2	c0278a	15/9
c0278a	Introduction to Database Management Systems	1	c1119	5/9
c0278a	Introduction to Database Management Systems	1	c0278a	7/9
c0278a	Introduction to Database Management Systems	2	c0278a	15/9

$\sigma_{\text{Course.course\_id=Tutorial.course\_id}}$**(Course × Tutorial)**

Course.course_id	Course.name	Tutorial .tutorial_id	Tutorial .course_id	Tutorial .date
c1119	Data Structures and Algorithms	1	c1119	5/9
c0278a	Introduction to Database Management Systems	1	c0278a	7/9
c0278a	Introduction to Database Management Systems	2	c0278a	15/9

Cartesian product

$\pi_{Tutorial.date}($
$\sigma_{Course.name=\text{"Introduction to Database Management Systems"}}($
$\sigma_{Course.course\_id=Tutorial.course\_id"}$ **(Course $\times$ Tutorial)**
$)$
$)$

Tutorial.date
7/9
15/9

$\sigma_{Course.name=\text{"Introduction to Database Management Systems"}}(\sigma_{Course.course\_id=Tutorial.course\_id"}$ **(Course $\times$ Tutorial)**$)$

Course.course_id	Course.name	Tutorial .tutorial_id	Tutorial .course_id	Tutorial .date
c0278a	Introduction to Database Management Systems	1	c0278a	7/9
c0278a	Introduction to Database Management Systems	2	c0278a	15/9

$\sigma_{Course.course\_id=Tutorial.course\_id}$ **(Course $\times$ Tutorial)**

Course.course_id	Course.name	Tutorial .tutorial_id	Tutorial .course_id	Tutorial .date
c1119	Data Structures and Algorithms	1	c1119	5/9
c0278a	Introduction to Database Management Systems	1	c0278a	7/9
c0278a	Introduction to Database Management Systems	2	c0278a	15/9

Rename

- Notation: $\rho_X(E)$
  - Rename operator allows us to name and refer to the results of relational-algebra expressions
- $\rho_X(E)$ returns the expression E under the name X

```
SELECT Tutorial.date
FROM Course, Tutorial
WHERE
Coruse.name="Introduction to Database
Management Systems" AND
Course.course_id = Tutorial.course_id;
```

SQL

$$\pi_{Tutorial.date}(\sigma_{Course.name=\text{"Introduction to Database Management Systems"}}(\sigma_{Course.course\_id=Tutorial.course\_id\text{"}}(\textbf{Course} \times \textbf{Tutorial})))$$

Relational algebra (Without rename)

```
SELECT T.date
FROM Course C, Tutorial T
WHERE
C.name="Introduction to Database
Management Systems" AND
C.course_id = T.course_id;
```

SQL

$$\pi_{\textbf{T}.date}(\sigma_{\textbf{C}.name=\text{"Introduction to Database Management Systems"}}(\sigma_{\textbf{C}.course\_id=\textbf{T}.course\_id\text{"}}(\rho_{\textbf{C}}(\textbf{Course}) \times \rho_{\textbf{T}}(\textbf{Tutorial}))))$$

Relational algebra (With Course rename to C, Tutorial rename to T)

Section 2

Exercises

Question 1

- Given the following relational schema:
  - **Student** (UID, name, age).
  - **Course** (CID, title).
  - **Enroll** (UID, CID) with UID referencing **Student** and CID referencing **Course**.

*UID and CID, age are interger; name and title are varchar.

- Which of the following is (are) valid Relational Algebra expression(s)?
  - π_{UID}(**Student**)-π_{CID}(**Course**) ✔
  - **Course**-π_{UID}(**Enroll**) ✗

The left and right parts of Set difference have to be **comparable** (same number of attributes).

Question 1

- Given the following relational schema:
  - **Student** (UID, **name**, **age**).
  - **Course** (CID, **title**).
  - **Enroll** (UID, CID) with **UID** referencing **Student** and **CID** referencing **Course**.

*UID and CID, age are interger; name and title are varchar.

- Which of the following is (are) valid Relational Algebra expression(s)?
  - $\sigma_{age<18}(\textbf{Student} \cup \textbf{Course})$ ✗ Student and Course have different attributes.
  - $\sigma_{age<18}(\pi_{UID,\ name}(\textbf{Student}))$ ✗ No attribute “age” for selection.

Question 2

```
SELECT employee_id,name
FROM Employee
WHERE employee_id < 100;
```

SQL

- Find the employeeIDs and names of employees whose employeeID < 100.

Employee

employeeID	name	start_date
97	May	Nov 16, 1997
98	Felix	Jun 30, 2003
99	May	Sep 18, 2007
100	George	Jan 1, 2008

Query processer → Option 1

Query processer → Option 2

Option 1

$\sigma_{employeeID<100}$(**Employee**)

employeeID	name	start_date
97	May	Nov 16, 1997
98	Felix	Jun 30, 2003
99	May	Sep 18, 2007

$\pi_{name,\ employeeID}(\sigma_{employeeID<100}($**Employee**$))$

employeeID	name
97	May
98	Felix
99	May

Option 2

$\pi_{name,\ employeeID}$(**Employee**)

employeeID	name
97	May
98	Felix
99	May
100	George

$\sigma_{employeeID<100}(\pi_{name,\ employeeID}($**Employee**$))$

employeeID	name
97	May
98	Felix
99	May

Which one is better?

Question 3

- Find the dept. id(s) where employees named Smith work.

Option 1

$$\pi_{W.department\_id}(\sigma_{E.name=\text{“Smith”}}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(\textbf{Employees}) \times \rho_W(\textbf{Works\_in}))))$$

Now we compute the Cartesian product between **Employees** and **Works_in** first, which creates an **intermediate relation** with 10,000*5 = 50,000 tuples! Can we reduce the size of intermediate relation?

Employees

employee_id	name	salary
1	Jones	26000
2	Smith	28000
…	…	…
10000	…	…

Employees (10000 tuples)

Works_in

employee_id	department_id	since
1	1	2001-1-1
2	1	2002-4-1
2	2	2005-2-2
3	3	2003-1-1
4	3	2005-1-1

Works_in (5 tuples)

```
SELECT W.department_id
FROM Employees E, Works_in W
WHERE E.name = 'Smith' AND
      E.employee_id = W.employee_id;
```

Question 3

- Find the dept. id(s) where employees named Smith work.

Option 1

$$\pi_{W.department\_id}(\sigma_{E.name=\text{“Smith”}}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(\textbf{Employees}) \times \rho_W(\textbf{Works\_in}))))$$

Option 2

$$\pi_{W.department\_id}(\sigma_{E.employee\_id=W.employee\_id}(\sigma_{E.name=\text{“Smith”}}(\rho_E(\textbf{Employees})) \times \rho_W(\textbf{Works\_in})))$$

We apply selection on Employees before the Cartesian product. If there is only two employee named Smith, the **intermediate relation** can be reduced to 2*5 = 10 tuples!

Employees

employee_id	name	salary
1	Jones	26000
2	Smith	28000
…	…	…
10000	…	…

Employees (10000 tuples)

Works_in

employee_id	department_id	since
1	1	2001-1-1
2	1	2002-4-1
2	2	2005-2-2
3	3	2003-1-1
4	3	2005-1-1

Works_in (5 tuples)

```sql
SELECT W.department_id
FROM Employees E, Works_in W
WHERE E.name = 'Smith' AND
        E.employee_id = W.employee_id;
```

Banking example

Foreign key

Foreign key

Branch

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Loan

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Borrower

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Foreign key

Foreign key

Foreign key

Account

branch_id	account_id	balance
B1	A1	500
B2	A2	400
B2	A3	900
B1	A4	700

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Foreign key

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Question 4

- Find the names of all customers who have a loan, an account, or both in a bank.

```
SELECT customer_id
FROM Borrower
UNION
SELECT customer_id
FROM Depositor
```

$\pi_{customer\_id}$ (Borrower) $\cup$ $\pi_{customer\_id}$ (Depositor)

- Find the names of all customers who have both a loan and an account in a bank.

```
SELECT customer_id
FROM Borrower
INTERSECT
SELECT customer_id
FROM Depositor
```

$\pi_{customer\_id}$ (Borrower) $\cap$ $\pi_{customer\_id}$ (Depositor)

Wait! Do we have set intersection in relational algebra ☺?

$\pi_{customer\_id}$ (Borrower) – ($\pi_{customer\_id}$ (Borrower) – $\pi_{customer\_id}$ (Depositor))

Additional operators

- These operations do not add any power to relational algebra, but simplify common queries.
  - Set intersection ($\cap$)
  - Natural join ($\bowtie$)
  - Assignment ($\leftarrow$)
  - Left outer join (⟕), Right outer join (⟖)
  - Division ($\div$)

Section 3

Additional operators

Motivation

- The fundamental operators of the relational algebra introduced are sufficient to express any relational-algebra query.
- However, if we restrict ourselves to just the fundamental operators, certain common queries are tedious to express.
  - Therefore, we define additional operators that do not add any power to the algebra, but simplify common queries

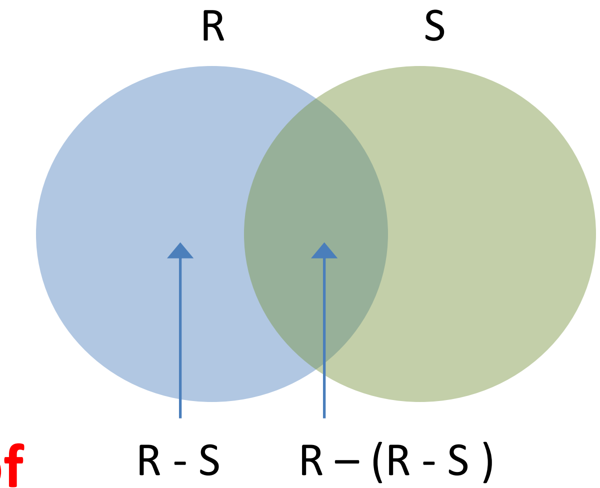
That is to say, for each additional operator, we can give an equivalent expression that use only the fundamental operators.

Additional operators

- Set intersection ($\cap$)
- Natural join ($\bowtie$)
- Assignment ($\leftarrow$)
- Left outer join (⟕), Right outer join (⟖)
- Division ($\div$)

28

Set intersection



- $R \cap S = R - (R - S)$
  - R and S must have the **same number of attributes** and **attribute data types are compatible**.
- Example
  - **Query:** Find the employee_id of employees who work in department 1 **and** department 3.

```
SELECT employee_id
FROM Works_in
WHERE department_id=1

INTERSECT

SELECT employee_id
FROM Works_in
WHERE department_id=3
```

SQL

→ Query processer →

$$(\pi_{employee\_id}(\sigma_{department\_id=1}(\textbf{Works\_in}))) \cap (\pi_{employee\_id}(\sigma_{department\_id=3}(\textbf{Works\_in})))$$

Relational algebra with set intersection

$$(\pi_{employee\_id}(\sigma_{department\_id=1}(\textbf{Works\_in}))) - ((\pi_{employee\_id}(\sigma_{department\_id=1}(\textbf{Works\_in}))) - (\pi_{employee\_id}(\sigma_{department\_id=3}(\textbf{Works\_in}))))$$

Equivalence relational algebra with only fundamental operators

Set intersection

For your reference, there is another way to answer the same query by joining the **Works_in** table with itself.

```sql
SELECT DISTINCT employee_id
FROM Works_in W1, Works_in W2
WHERE
W1.employee_id = W2.employee_id AND
W1.department_id=1 AND
W2. department_id = 3
```

SQL

→ Query processer →

Works_in

employee_id	department_id
1	1
2	1
1	3

$$\pi_{employee\_id}(\sigma_{W1.department\_id=1 \wedge W2.department\_id=3}(\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\mathbf{Works\_in}) \times \rho_{W2}(\mathbf{Works\_in}))))$$

Set intersection

For your reference, there is another way to answer the same query by joining the **Works_in** table with itself.

ρ_{W1} (**Works_in**) × ρ_{W2} (**Works_in**)

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	1
1	1	2	1
1	1	1	3
2	1	1	1
2	1	2	1
2	1	1	3
1	3	1	1
1	3	2	1
1	3	1	3

Works_in

employee_id	department_id
1	1
2	1
1	3

```
SELECT DISTINCT employee_id
FROM Works_in W1, Works_in W2
WHERE
W1.employee_id = W2.employee_id AND
W1.department_id=1 AND
W2. department_id = 3
```

SQL

Query processer

$$\pi_{employee\_id} (\sigma_{W1.department\_id=1 \wedge W2.department\_id=3} (\sigma_{W1.employee\_id=W2.employee\_id} (\rho_{W1} (\textbf{Works\_in}) \times \rho_{W2} (\textbf{Works\_in}))))$$

Set intersection

For your reference, there is another way to answer the same query by joining the **Works_in** table with itself.

$\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in}))$

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	1
1	1	1	3
2	1	2	1
1	3	1	1
1	3	1	3

$\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in})$

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	1
1	1	2	1
1	1	1	3
2	1	1	1
2	1	2	1
2	1	1	3
1	3	1	1
1	3	2	1
1	3	1	3

Works_in

employee_id	department_id
1	1
2	1
1	3

```sql
SELECT DISTINCT employee_id
FROM Works_in W1, Works_in W2
WHERE
W1.employee_id = W2.employee_id AND
W1.department_id=1 AND
W2. department_id = 3
```

SQL

Query processer

$\pi_{employee\_id}(\sigma_{W1.department\_id=1 \land W2.department\_id=3}(\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in}))))$

Set intersection

$\sigma_{W1.department\_id=1 \wedge W2.department\_id=3}(\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in})))$

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	3

$\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in}))$

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	1
1	1	1	3
2	1	2	1
1	3	1	1
1	3	1	3

$\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in})$

W1.employee_id	W1.department_id	W2.employee_id	W2.department_id
1	1	1	1
1	1	2	1
1	1	1	3
2	1	1	1
2	1	2	1
2	1	1	3
1	3	1	1
1	3	2	1
1	3	1	3

Works_in

employee_id	department_id
1	1
2	1
1	3

For your reference, there is another way to answer the same query by joining the **Works_in** table with itself.

```
SELECT DISTINCT employee_id
FROM Works_in W1, Works_in W2
WHERE
W1.employee_id = W2.employee_id AND
W1.department_id=1 AND
W2. department_id = 3
```

SQL

Query processer

$\pi_{employee\_id}(\sigma_{W1.department\_id=1 \wedge W2.department\_id=3}(\sigma_{W1.employee\_id=W2.employee\_id}(\rho_{W1}(\textbf{Works\_in}) \times \rho_{W2}(\textbf{Works\_in}))))$

Natural join

- Usually, a query that involves a Cartesian product includes a selection operation on the result of the Cartesian product.
  - The selection operation most often requires that all **attributes that are common** to the relations that are involved in the Cartesian product **be equated**.
- $r \bowtie s = \pi_{R \cup S}(\ \sigma_{r.A1 = s.A1 \land r.A2 = s.A2 \land \ldots \land r.An = s.An}\ (\ r \times s\ )\ )$

  Requires the common attributes to be equated

  - where $R \cap S = \{A1, A2, \ldots, An\}$

Commutative: $r \bowtie s = s \bowtie r$

Natural join

- The schema of R $\bowtie$ S is R-schema $\cup$ S-schema (repeated attributes are removed)
  - For each pair of tuples t_r from R and t_s from S,
  - If t_r and t_s share the same value over each of the common attributes in R and S,
  - Tuple t will be added to the result of R $\bowtie$ S.

Natural join

R

A	B
1	1
1	2
2	3

S

A	C
1	2
2	1

- Common attributes: $R \cap S = \{A\}$
- Attributes of the resulting relation: $R \cup S = \{A,B,C\}$

$R \times S$

R.A	R.B	S.A	S.C
1	1	1	2
1	1	2	1
1	2	1	2
1	2	2	1
2	3	1	2
2	3	2	1

→

$\sigma_{R.A=S.A}(\mathbf{R} \times \mathbf{S})$

R.A	R.B	S.A	S.C
1	1	1	2
1	2	1	2
2	3	2	1

→

$\pi_{A,B,C}(\sigma_{R.A=S.A}(\mathbf{R} \times \mathbf{S}))$

A	B	C
1	1	2
1	2	2
2	3	1

equivalent to:

$R \bowtie S$

A	B	C
1	1	2
1	2	2
2	3	1

Assignment

- It is convenient to write a relational-algebra expression by assigning parts of it to **temporary relation variables**.
- The assignment operator, denoted by "$\leftarrow$", works like assignment operator "=" in programming language.
  - temp1 $\leftarrow \pi_a(R)$
  - temp2 $\leftarrow \pi_a(S)$
  - result $\leftarrow$ temp1 – temp2

With the assignment operator, a query can be written as a sequential program.

temp1, temp2, result are called "**relation variable**".

Outer join

- The outer-join operator is an extension of the join operation to deal with missing information.

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

Natural join (e.g., joining Customer and Depositor) result in a table **without the information of customers (e.g., C3 in this case) who has no account**.

- Outer join
  - Natural join result, plus
  - The **tuples that do not match any tuples from the other side**.

Left outer join

$$R ⟕ S = (R \bowtie S) \cup (R - \pi_R(R \bowtie S)) \times \{(\textit{null}, \ldots, \textit{null})\}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

Let's illustrate why the left outer join is defined as $(R \bowtie S) \cup (R - \pi_R(R \bowtie S)) \times \{(\textit{null}, \ldots, \textit{null})\}$ through a step-by-step illustration.

Left outer join

$$R ⟕ S = (R \bowtie S) \cup (R - \pi_R(R \bowtie S)) \times \{(\textit{null}, \ldots, \textit{null})\}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

Customer - $\pi_{\text{Customer's attributes}}$ **(Customer ⋈ Depositor)**

customer_id	name	address
C3	Jolly	CB311

Finding missed tuples in the natural join

$R - \pi_R(R \bowtie S)$ is to generate the tuples in R (i.e., Customer) that are missed in the natural join.

i.e., C3 Jolly in Customer doesn't have any matched records in Depositor, we use this part to recover Jolly's record.

Left outer join

$$R ⟕ S = (R \bowtie S) \cup (R - \pi_R(R \bowtie S)) \times \{ (\textit{null}, \ldots, \textit{null}) \}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Customer $\bowtie$ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

Customer - $\pi_{\text{Customer's attributes}}$ **(Customer $\bowtie$ Depositor)**

customer_id	name	address
C3	Jolly	CB311

Customer - $\pi_{\text{Customer's attributes}}$ **(Customer $\bowtie$ Depositor)** $\times \{ (\textit{null}, \ldots, \textit{null}) \}$

customer_id	name	address	account_id
C3	Jolly	CB311	*null*

Constructing the missed tuple by adding null value to extra attributes
The "$\times \{ (\textit{null}, \ldots, \textit{null}) \}$" part simply add back the remaining column values as ***null*** because there is no matched records in S (i.e., Depositor).

Left outer join

$$R \;⟕\; S = (R \bowtie S) \cup (R - \pi_R(R \bowtie S)) \times \{(\textbf{\textit{null}}, \ldots, \textbf{\textit{null}})\}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C4

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

Customer - $\pi_{\text{Customer's attributes}}$ **(Customer ⋈ Depositor)**

customer_id	name	address
C3	Jolly	CB311

Customer - $\pi_{\text{Customer's attributes}}$ **(Customer ⋈ Depositor**) × { (***null***, … , ***null***) }

customer_id	name	address	account_id
C3	Jolly	CB311	***null***

(Customer ⋈ Depositor) ∪ **(Customer** - $\pi_{\text{Customer's attributes}}$ **(Customer ⋈ Depositor**)) × { (***null***, … , ***null***) }

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C3	Jolly	CB311	***null***
C4	Yvonne	CB415	A3
C4	Yvonne	CB415	A4

equivalent to: **Customer ⟕ Depositor**

Right outer join

$$R ⟖ S = (R \bowtie S) \cup (S - \pi_S(R \bowtie S)) \times \{ (\textbf{\textit{null}}, \ldots , \textbf{\textit{null}}) \}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C5

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3

Let's illustrate why the right outer join is defined as

$(R \bowtie S) \cup (S - \pi_S(R \bowtie S)) \times \{ (\textbf{\textit{null}}, \ldots , \textbf{\textit{null}}) \}$

through a step-by-step illustration.

Right outer join

$$R ⟖ S = (R \bowtie S) \cup (S - \pi_S(R \bowtie S)) \times \{(\textbf{\textit{null}}, \ldots, \textbf{\textit{null}})\}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C5

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3

Depositor - $\pi_{\text{Depositor's attributes}}$ **(Customer ⋈ Depositor**)

customer_id	account_id
C5	A4

Finding missed tuples in the natural join

$S - \pi_S(R \bowtie S)$ is to generate the tuples in S (i.e., Depositor) that are missed in the natural join.

i.e., C5 in Depositor doesn't have any matched records in Customer, we use this part to recover C5's record.

Right outer join

$R ⟖ S = (R \bowtie S) \cup (S - \pi_S(R \bowtie S)) \times \{ (\textbf{\textit{null}}, \dots , \textbf{\textit{null}}) \}$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C5

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3

Depositor - $\pi_{\text{Depositor's attributes}}$ (**Customer ⋈ Depositor**)

customer_id	account_id
C5	A4

Depositor - $\pi_{\text{Depositor's attributes}}$ (**Customer ⋈ Depositor**) $\times \{ (\textbf{\textit{null}}, \dots , \textbf{\textit{null}}) \}$

customer_id	name	address	account_id
C5	***null***	***null***	A4

Constructing the missed tuple by adding *null* value to extra attributes

The "$\times \{ (\textbf{\textit{null}}, \dots , \textbf{\textit{null}}) \}$" part simply add back the remaining column values as ***null*** because there is no matched records in R (i.e., Customer).

Right outer join

$$R ⟖ S = (R \bowtie S) \cup (S - \pi_S(R \bowtie S)) \times \{ (\textit{null}, \ldots, \textit{null}) \}$$

Customer

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Depositor

account_id	customer_id
A1	C1
A1	C2
A2	C2
A3	C4
A4	C5

Customer ⋈ Depositor

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3

Depositor - $\pi_{\text{Depositor's attributes}}$ **(Customer ⋈ Depositor)**

customer_id	account_id
C5	A4

Depositor - $\pi_{\text{Depositor's attributes}}$ **(Customer ⋈ Depositor)** $\times \{ (\textit{null}, \ldots, \textit{null}) \}$

customer_id	name	address	account_id
C5	***null***	***null***	A4

(Customer ⋈ Depositor) $\cup$ **(Depositor** - $\pi_{\text{Depositor's attributes}}$ **(Customer ⋈ Depositor))** $\times \{ (\textit{null}, \ldots, \textit{null}) \}$

customer_id	name	address	account_id
C1	Kit	CB320	A1
C2	Ben	CB326	A1
C2	Ben	CB326	A2
C4	Yvonne	CB415	A3
C5	***null***	***null***	A4

equivalent to: **Customer ⟖ Depositor**

Division

- Notation: $R \div S$
- Definition
  - Let $S \subseteq R$

    The attributes in relation S is a subset of the attributes in relation R.
  - $R \div S = \{ t \mid t \in \pi_{R-S}(R) \wedge (\forall s \in S, ((t \cup s) \in R)) \}$

R

A	B
1	1
2	1
2	2
3	3
4	1
4	2
4	3

S

B
1
2

R ÷ S

A
2
4

Condition 1. A resulting tuple t has to be in the relation $\pi_{R-S}(R)$

$\pi_{R-S}(R)$

A
1
2
3
4

Condition 2. And if we combine t with each tuple $s \in S$, all the combined tuples have to be included in R.

① For each tuple $t \in \pi_{R-S}(R)$		② **"$\forall$ s $\in$ S, t $\cup$ s" part:** Generate tuples by union t with all tuples $s \in S$		③ **"$\forall$ s $\in$ S, ((t $\cup$ s) $\in$ R)" part:** Then we check if both tuples generated are in R.
$t_1 \in \pi_{R-S}(R)$	A: 1	$\forall s \in S, (t_1 \cup s)$	(A, B): (1, 1), (1, 2)	In this case, not both tuples are in R, so t_1 is **NOT** in result of $R \div S$
$t_2 \in \pi_{R-S}(R)$	A: 2	$\forall s \in S, (t_2 \cup s)$	(A, B): (2, 1), (2, 2)	In this case, both tuples are in R, so t_2 is in result of $R \div S$

Division

R

A	B
1	1
2	1
2	2
3	3
4	1
4	2
4	3

S

B
1
2

R ÷ S

A
2
4

Observation

Let's focus on the result of **R ÷ S** (say, the tuple A=2), it means that

- For the tuples with values in attribute A equals to 2 in relation R,
- Those tuples's values in attribute B covers **ALL** values in attribute B of S.

- Division is used to express queries with "**all**"
  - Find the IDs of all students who have taken **all** CS courses (dpt_id = 1).

Student

student_id	name	dpt_id
1	Peter	1
2	Sharon	1
3	David	2
4	Joe	3

Takes

student_id	course_id	Grade
1	1	A
1	2	B
1	3	A+
2	3	B-
3	3	B
4	1	C
4	2	A-

Course

course_id	title	dpt_id	credit
1	Intro to DB	1	6
2	Programming I	1	6
3	Accounting	2	6

Division

- **Step 1. All part (the relation S):** What is the course ID of **all** CS courses (dpt_id = 1)?

$\pi_{course\_id}(\sigma_{dpt\_id=1}(Course))$

course_id
1
2

$\sigma_{dpt\_id=1}(Course)$

course_id	title	dpt_id	credit
1	Intro to DB	1	6
2	Programming I	1	6

Student

student_id	name	dpt_id
1	Peter	1
2	Sharon	1
3	David	2
4	Joe	3

Takes

student_id	course_id	Grade
1	1	A
1	2	B
1	3	A+
2	3	B-
3	3	B
4	1	C
4	2	A-

Course

course_id	title	dpt_id	credit
1	Intro to DB	1	6
2	Programming I	1	6
3	Accounting	2	6

Division

- **Step 2. Dividend part (the relation R):** What is the list of Takes tuples (i.e., all < student_id, course_id > pairs)?

$\pi_{student\_id,course\_id}$ (Takes)

student_id	course_id
1	1
1	2
1	3
2	3
3	3
4	1
4	2

π_{course_id} ($\sigma_{dpt\_id=1}$(Course))

course_id
1
2

Student

student_id	name	dpt_id
1	Peter	1
2	Sharon	1
3	David	2
4	Joe	3

Takes

student_id	course_id	Grade
1	1	A
1	2	B
1	3	A+
2	3	B-
3	3	B
4	1	C
4	2	A-

Course

course_id	title	dpt_id	credit
1	Intro to DB	1	6
2	Programming I	1	6
3	Accounting	2	6

Division

- **Step 3. Division:** Which student in $\pi_{\text{student\_id,course\_id}}$ (Takes) takes **all** CS courses $\pi_{\text{course\_id}}(\sigma_{\text{dpt\_id=1}}(\text{course}))$?.

$\pi_{\text{student\_id,course\_id}}$ (Takes)

student_id	course_id
1	1
1	2
1	3
2	3
3	3
4	1
4	2

$\pi_{\text{course\_id}}(\sigma_{\text{dpt\_id=1}}(\text{Course}))$

course_id
1
2

$\pi_{\text{student\_id,course\_id}}(\text{Takes}) \div \pi_{\text{course\_id}}(\sigma_{\text{dpt\_id=1}}(\text{Course}))$

student_id
1
4

Explanation: Let's focus on student_id = 1 in the result

- It means that, for the tuples in **Takes** with student_id=1,
- Those tuple's value in course_id attribute covers **all** 1 and 2 (i.e., the CS courses).
- That is to say, student with student_id 1 takes **ALL** CS courses.
- The same argument apples to student_id 4.

Division

- Division has a property: $R \times S \div S = R$

R

A	B
1	1
1	2
2	3

S

C	D
1	2
2	2

$R \times S$

A	B	C	D
1	1	1	2
1	1	2	2
1	2	1	2
1	2	2	2
2	3	1	2
2	3	2	2

$(R \times S) \div S$

A	B
1	1
1	2
2	3

= R

An intuitive property of the division operator of the relational algebra is simply that it is the inverse of the cartesian product. For example, if you have two relations R and S, then, if U is a relation defined as the cartesian product of them: `U = R x S`, the division is the operator such that `U ÷ R = S` and `U ÷ S = R`

Section 4

Extended operators

Aggregation

- Aggregation function takes a collection of values and returns a single-valued result.
  - e.g., avg, min, max, sum, count, count-distinct
- Aggregate operation in relational algebra:
  - **Grouping** - Divides the tuples into groups.
  - **Aggregation** - Computes an aggregation function in each group to create a result tuple.

Aggregation

$$_{G_1, G_2, \ldots, G_n}\,g\,_{F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}\ (E)$$

- **E** – a relation (can be a result of relational algebra expression).
- **$G_1, \ldots, G_n$** – attributes used to form groups.
  - Tuples with the same values in **G_1** to **G_n** are put into the same group.
  - **G** can be empty, which means that the whole relation is one group.
- **$F_i(A_i)$** – an aggregate function applied on an attribute.

Aggregation

Account

branch_id	account_id	balance
B1	A1	500
B2	A2	400
B2	A3	900
B1	A4	700

$$\rho_{Result(branch\_id,\ sum\_of\_balance)} \left({}_{branch\_id}\, g_{\,sum(balance)} (Account) \right)$$

Result

branch_id	sum_of_balance
B1	1200
B2	1300

- **Step 1.** Let's group the tuples in Account according to their branch_id.
- **Step 2.** Then aggregate the tuples in each group by summing their values in the balance attribute.
- **Step 3.** Since the resulting relation has no name after aggregation, we use renaming operator to give name to the relation and attributes.

Aggregation

- Note that grouping **can be done on multiple attributes**.
- E.g., in this case, we group tuples in Student with the same values in both dpt_id and gender attributes.
- i.e., We are finding the number of male / female students in each department.

Student

student_id	name	dpt_id	gender
1	Peter	1	M
2	Sharon	1	F
3	David	2	M
4	Joe	2	M
5	Betty	1	F

$$\rho_{Result(dpt\_id,\ gender,\ count)} \left({}_{dpt\_id,\ gender}\, g_{\ count()} (Student) \right)$$

Result

dpt_id	gender	count
1	M	1
1	F	2
2	M	2

Section 5

Algebraic properties

Transformation of expression

- A query can be expressed in several different ways, with different cost of evaluation.
- Two relational-algebra expressions are said to be equivalent if, on every legal database instance, the two expressions generate the same set of tuples.

In the following discussions, we treat a relation as a set of tuples, the order of the tuples is irrelevant.

Equivalence rules

- **Rule 1.** Only the final operations in a sequence of projection operations are needed.

$$\pi_{L1}(\pi_{L2}(\ldots(\pi_{Ln}(E))\ldots)) = \pi_{L1}(E)$$

$\pi_{employee\_id}(\pi_{employee\_id,salary}$**(Employee))**

employee_id
1
2
3

$\pi_{employee\_id,salary}$**(Employee)**

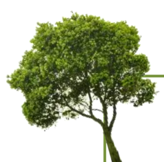
employee_id	name
1	Jones
2	Smith
3	Smith

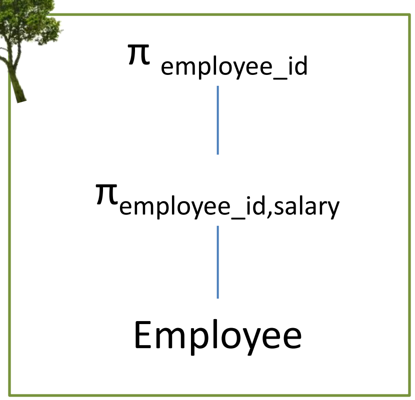
Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Smith	24000

$\pi_{employee\_id}($
$\quad \pi_{employee\_id,salary}($ Employee $)$
$)$

Regular expression





Expression tree

Expression tree:

- Tells which operator is executed ahead of another.
- It allow transformation of the execution order by applying the equivalence rules. (Alter the tree)

Equivalence rules

- **Rule 1.** Only the final operations in a sequence of projection operations are needed.

$$\pi_{L1}(\pi_{L2}(\ldots(\pi_{Ln}(E))\ldots)) = \pi_{L1}(E)$$

$\pi_{employee\_id}($
$\quad \pi_{employee\_id,salary}($ Employee $)$
$)$

Regular expression

equivalent to

$\pi_{employee\_id}($ Employee $)$

Regular expression

$\pi_{employee\_id}$(**Employee**)

employee_id
1
2
3

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Smith	24000

$\pi_{employee\_id}$
|
$\pi_{employee\_id,salary}$
|
Employee

Expression tree

Remove it

equivalent to

$\pi_{employee\_id}$
|
Employee

Transformed expression tree

Equivalence rules

- **Rule 2.** Conjunctive selection operations can be deconstructed into a sequence of individual selections.

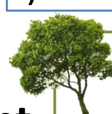
$$\sigma_{p1 \wedge p2}(E) = \sigma_{p1}(\sigma_{p2}(E))$$

$\sigma_{name="Smith" \wedge salary>24000}(\text{Employee})$

$\sigma_{name="Smith" \wedge salary>24000}$
|
Employee

equivalent to

$\sigma_{name="Smith"}(\sigma_{salary>24000}(\text{Employee}))$



$\sigma_{name="Smith"}$
|
$\sigma_{salary>24000}$
|
Employee

$\sigma_{name="Smith"}(\sigma_{salary>24000}(\textbf{Employee}))$

employee_id	name	salary
2	Smith	28000

$\sigma_{salary>24000}(\textbf{Employee})$

employee_id	name	salary
1	Jones	26000
2	Smith	28000

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Smith	24000

You may wonder why breaking down the conjunctive selections is useful.
We will show that it is useful **to reduce temporary result**.
We can try to push each selection predicates down the tree (to perform selection as early as possible).

Equivalence rules

- **Rule 3.** Selection operations are commutative.

$$\sigma_{p1}(\sigma_{p2}(E)) = \sigma_{p2}(\sigma_{p1}(E))$$

$\sigma_{name="Smith"}(\sigma_{salary>24000}($ **Employee** $))$

employee_id	name	salary
2	Smith	28000

$\sigma_{salary>24000}$(**Employee**)

employee_id	name	salary
1	Jones	26000
2	Smith	28000
4	David	25000

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Smith	24000
4	David	25000

$\sigma_{name="Smith"}($
$\sigma_{salary>24000}$(Employee)
$)$

$\sigma_{salary>24000}($
$\sigma_{name="Smith"}$(Employee)
$)$

$\sigma_{name="Smith"}$
|
$\sigma_{salary>24000}$
|
Employee

$\sigma_{salary>24000}$
|
$\sigma_{name="Smith"}$
|
Employee

$\sigma_{salary>24000}(\sigma_{name="Smith"}($ **Employee** $))$

employee_id	name	salary
2	Smith	28000

$\sigma_{name="Smith"}$(**Employee**)

employee_id	name	salary
2	Smith	28000
3	Smith	24000

Employes

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Smith	24000
4	David	25000

Note that the two executions have different costs. In particular, the size of their temporary relations are different.

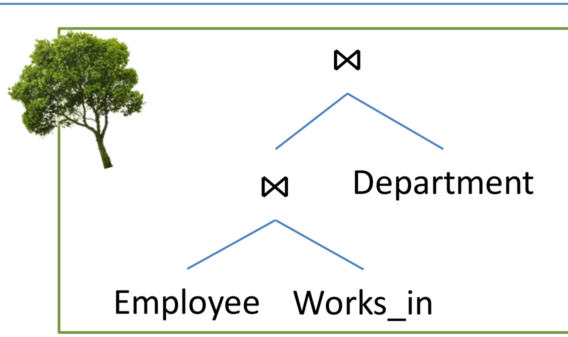
Equivalence rules

$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An}(r \times s))$$

- **Rule 4.** Natural join operations are associative.

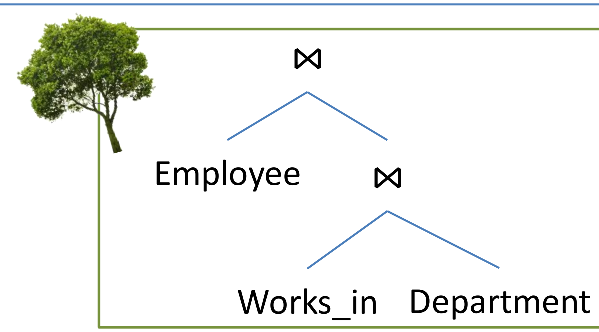
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(Employee ⋈ Works_in) ⋈ Department



equivalent to

Employee ⋈ (Works_in ⋈ Department)

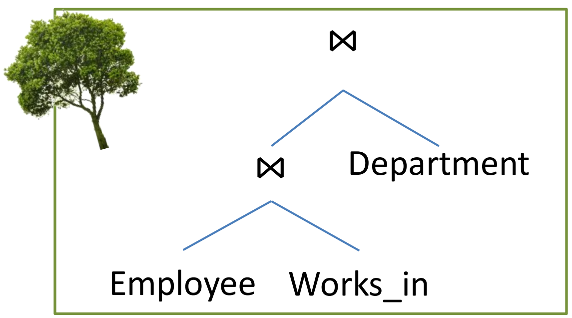


Although both expression trees return the **same resulting relation**, these two expression trees have **different costs** because the size of their temporary relations are different

Equivalence rules

- **Expression tree A.**

(Employee ⋈ Works_in) ⋈ Department



(Employee ⋈ Works_in) ⋈ Department

employee_id	Employee. name	salary	department_id	Department. name
1	Jones	26000	1	Toys
2	Smith	28000	1	Toys
2	Smith	28000	2	Tools
3	Parker	35000	3	Food
4	Smith	24000	3	Food

Employee ⋈ Works_in

employee_id	name	salary	department_id
1	Jones	26000	1
2	Smith	28000	1
2	Smith	28000	2
3	Parker	35000	3
4	Smith	24000	3

Natural join evaluates 4*5 = 20 combinations
result temporary relation consists of 5 tuples and 4 columns.

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

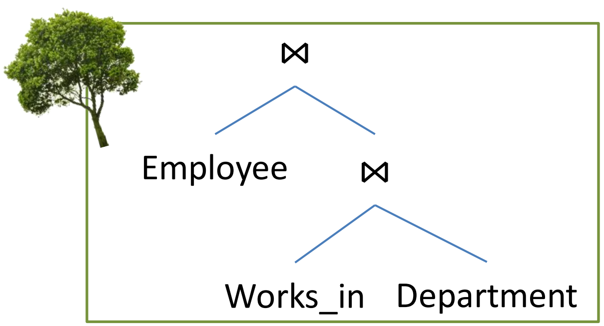
Department

department_id	name
1	Toys
2	Tools
3	Food

Equivalence rules

- **Expression tree B.**

Employee ⋈ (Works_in ⋈ Department)



Employee ⋈ (Works_in ⋈ Department)

employee_id	Employee. name	salary	department_id	Department. name
1	Jones	26000	1	Toys
2	Smith	28000	1	Toys
2	Smith	28000	2	Tools
3	Parker	35000	3	Food
4	Smith	24000	3	Food

Works_in ⋈ Department

employee_id	department_id	name
1	1	Toys
2	1	Toys
2	2	Tools
3	3	Food
4	3	Food

Natural join evaluates 5*3 = 15 combinations
result temporary relation consists of 5 tuples and 3 columns.

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

Department

department_id	name
1	Toys
2	Tools
3	Food

Equivalence rules

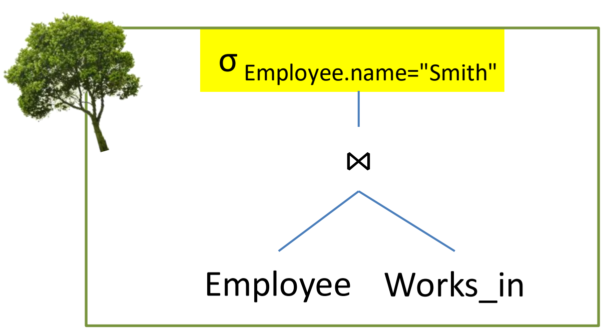
$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An}(r \times s))$$

- **Rule 5.** The selection operation distributes over the natural join operation under the following two conditions
  - **Rule 5a.** It distributes when all the attributes in selection condition **involve only the attributes of one of the expressions** (say, E_1) being joined.

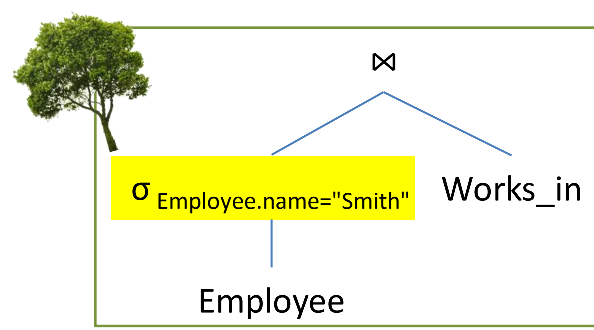
$$\sigma_p(E_1 \bowtie E_2) = (\sigma_p(E_1) \bowtie E_2)$$

$\sigma_{Employee.name="Smith"}($
Employee ⋈ Works_in
$)$

$(\sigma_{Employee.name="Smith"}(Employee) \bowtie Works\_in)$



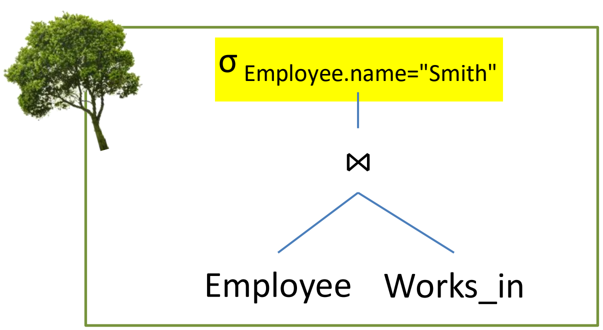
equivalent to



Equivalence rules

- **Expression tree A.**

$\sigma_{Employee.name="Smith"}($
$Employee \bowtie Works\_in$
$)$



$\sigma_{Employee.name="Smith"}$(Employee ⋈ Works_in)

employee_id	name	salary	department_id
2	Smith	28000	1
2	Smith	28000	2
4	Smith	24000	3

Employee ⋈ Works_in

employee_id	name	salary	department_id
1	Jones	26000	1
2	Smith	28000	1
2	Smith	28000	2
3	Parker	35000	3
4	Smith	24000	3

Natural join evaluates 4*5 = 20 combinations
result temporary relation consists of 5 tuples and 4 columns.

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

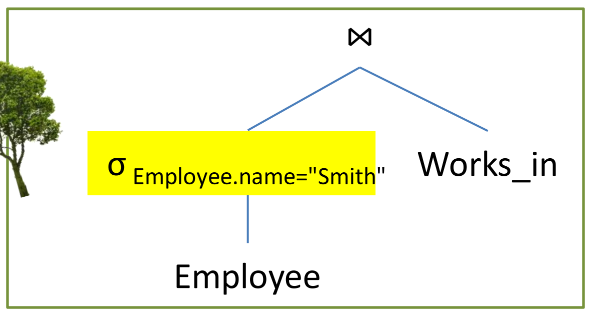
Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

Equivalence rules

- **Expression tree B.**

σ Employee.name="Smith" (Employee) ⋈ Works_in



σ Employee.name="Smith" **(Employee)**

employee_id	name	salary
2	Smith	28000
4	Smith	24000

Natural join evaluates 2*5 = 10 combinations
result temporary relation consists of 3 tuples and 4 columns.

σ Employee.name="Smith" **(Employee) ⋈ Works_in**

employee_id	name	salary	department_id
2	Smith	28000	1
2	Smith	28000	2
4	Smith	24000	3

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

When comparing with the equivalence expression tree 1, we can see that if we push the selection predicates down the natural join (perform selection earlier than joining), then we will probably have a **smaller temporary relation**.

Equivalence rules

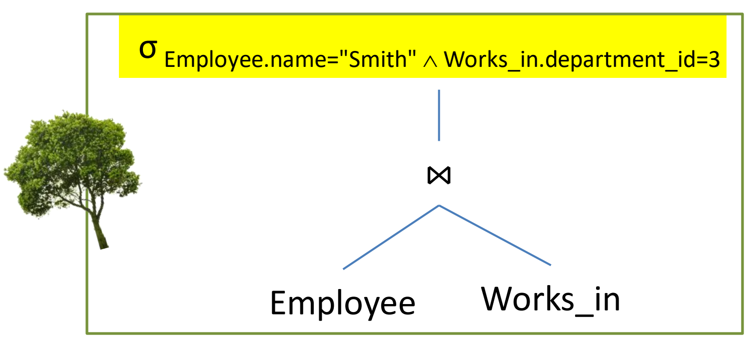
$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A1 = s.A1 \land r.A2 = s.A2 \land \ldots \land r.An = s.An}(r \times s))$$

- **Rule 5b.** The selection distributes when selection condition p1 involves only the attributes of E_1 and p2 involves only the attributes of E_2.

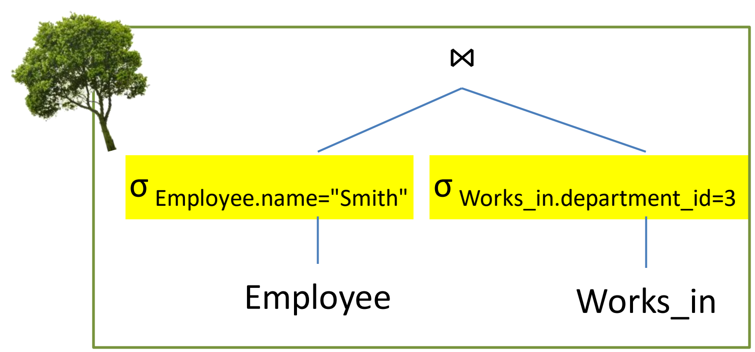
$$\sigma_{p1 \land p2}(E_1 \bowtie E_2) = (\sigma_{p1}(E_1) \bowtie \sigma_{p2}(E_2))$$

$\sigma_{\text{Employee.name="Smith"} \land \text{Works\_in.department\_id=3}}($ **Employee** $\bowtie$ **Works_in** $)$

$($ $\sigma_{\text{Employee.name="Smith"}}$ **(Employee)** $\bowtie$ $\sigma_{\text{Works\_in.department\_id=3}}$ (**Works_in**))



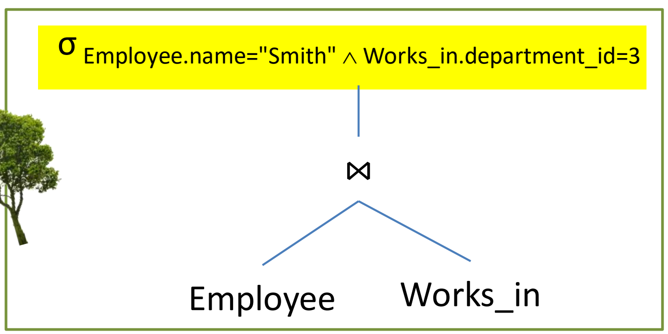
equivalent to



Equivalence rules

- **Expression tree A.**

$\sigma_{\text{Employee.name="Smith"} \wedge \text{Works\_in.department\_id=3}}($
Employee ⋈ Works_in
$)$



$\sigma_{\text{Employee.name="Smith"} \wedge \text{Works\_in.department\_id=3}}$(**Employee ⋈ Works_in**)

employee_id	name	salary	department_id
4	Smith	24000	3

Employee ⋈ Works_in

employee_id	name	salary	department_id
1	Jones	26000	1
2	Smith	28000	1
2	Smith	28000	2
3	Parker	35000	3
4	Smith	24000	3

Natural join evaluates 4*5 = 20 combinations

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

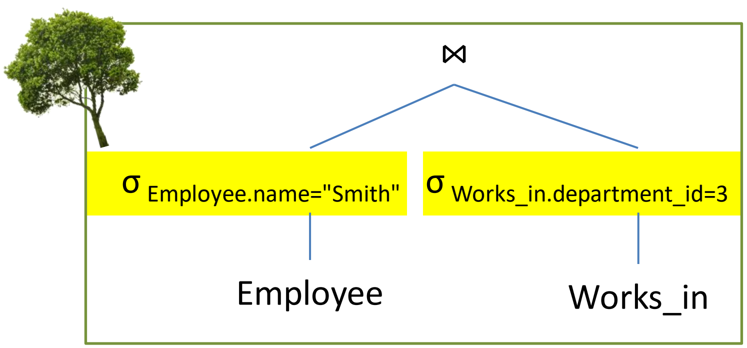
Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

Equivalence rules

- **Expression tree B.**

(σ Employee.name="Smith" (**Employee**)
⋈
σ Works_in.department_id=3 (**Works_in**))



(σ Employee.name="Smith" (**Employee**) ⋈ σ Works_in.department_id=3(**Works_in**))

employee_id	name	salary	department_id
4	Smith	24000	3

Natural join evaluates 4 combinations

σ Employee.name="Smith"(**Employee**)

employee_id	name	salary
2	Smith	28000
4	Smith	24000

σ Works_in.department_id=3(**Works_in**)

employee_id	department_id
3	3
4	3

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

When comparing with the equivalence expression 1, we can see that if we push the selection predicates down the natural join (perform selection earlier than joining), the **natural join would consider fewer combinations**.

Equivalence rules

$$r \bowtie s = \pi_{R \cup S}(\ \sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An}(\ r \times s\ )\ )$$

- **Rule 6.** The projection operation can distribute over the natural join operation.

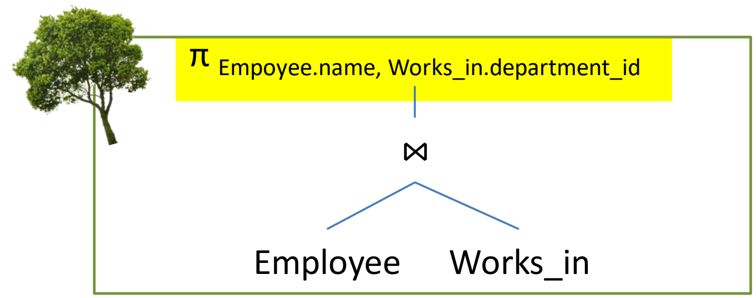
$$\pi_{L1 \cup L2}(\ E_1 \bowtie E_2) = \pi_{L1 \cup L2}(\ \ (\pi_{L1 \cup L3}(E_1)) \bowtie (\pi_{L2 \cup L3}(E_2))\ )$$

- Let L1 and L2 be some attributes from E1 and E2,respectively.
- Let L3 be attributes that are involved in join condition, but are not in $L1 \cup L2$.

Equivalence rules

$$\pi_{L1 \cup L2}(E_1 \bowtie E_2) = \pi_{L1 \cup L2}((\pi_{L1 \cup L3}(E_1)) \bowtie (\pi_{L2 \cup L3}(E_2)))$$

$\pi_{\text{Empoyee.name, Works\_in.department\_id}}$ (**Employee ⋈ Works_in**)

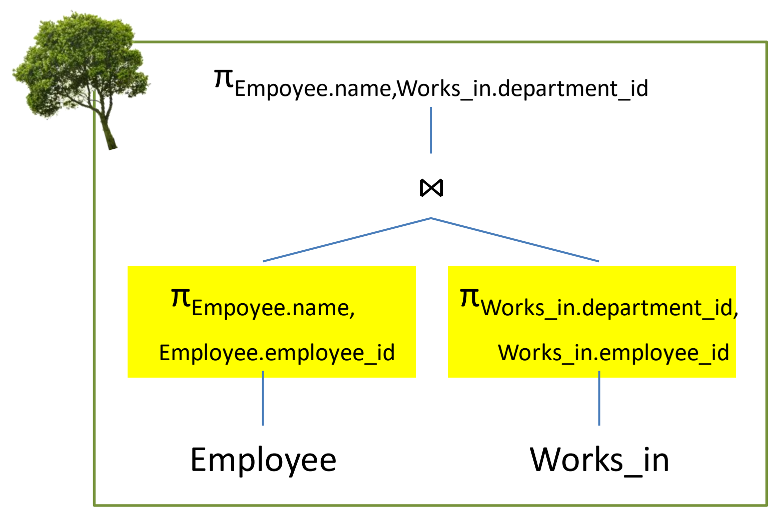


- L1 = Empoyee.name
- L2 = Works_in.department_id
- L3 = employee_id

The attribute used in natural join

equivalent to

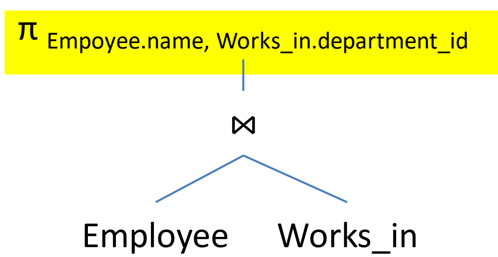
$\pi_{\text{Empoyee.name, Works\_in.department\_id}}$ (
$\pi_{\text{Empoyee.name, Employee.employee\_id}}$ (Employee)
⋈
$\pi_{\text{Works\_in.department\_id, Works\_in.employee\_id}}$ (Works_in)
)



Equivalence rules

- **Expression tree A.**

$\pi_{\text{Empoyee.name, Works\_in.department\_id}}$ **(Employee ⋈ Works_in)**



$\pi_{\text{Empoyee.name, Works\_in.department\_id}}$ **(Employee ⋈ Works_in)**

name	department_id
Jones	1
Smith	1
Smith	2
Parker	3
Smith	3

Employee ⋈ Works_in

employee_id	name	salary	department_id	since
1	Jones	26000	1	2012/1/1
2	Smith	28000	1	2011/3/2
2	Smith	28000	2	2014/2/1
3	Parker	35000	3	2013/2/2
4	Smith	24000	3	2013/2/8

Natural join evaluates 4*5 = 20 combinations, result temporary relation consists of 5 columns.

Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

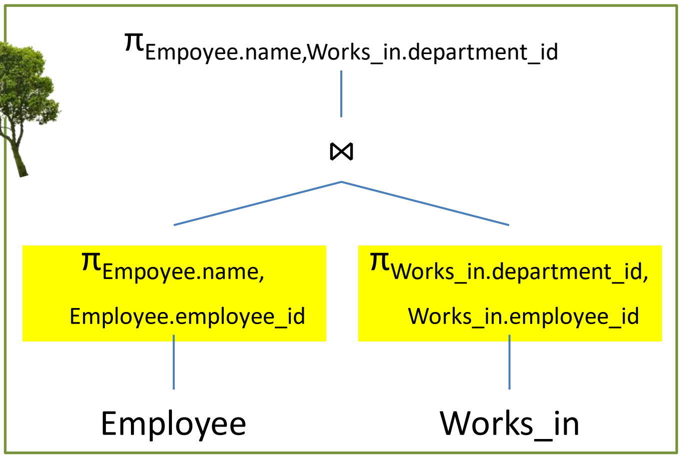
Works_in

employee_id	department_id	since
1	1	2012/1/1
2	1	2011/3/2
2	2	2014/2/1
3	3	2013/2/2
4	3	2013/2/8

Equivalence rules

Expression tree B.

$\pi_{\text{Empoyee.name, Works\_in.department\_id}}($
$\quad\pi_{\text{Empoyee.name, Employee.employee\_id}}(\text{Employee})$
$\quad\bowtie$
$\quad\pi_{\text{Works\_in.department\_id, Works\_in.employee\_id}}(\text{Works\_in})$
$)$



Employee

employee_id	name	salary
1	Jones	26000
2	Smith	28000
3	Parker	35000
4	Smith	24000

Works_in

employee_id	department_id	since
1	1	2012/1/1
2	1	2011/3/2
2	2	2014/2/1
3	3	2013/2/2
4	3	2013/2/8

$\pi_{\text{Empoyee.name, Employee.employee\_id}}(\textbf{ Employee })$

employee_id	name
1	Jones
2	Smith
3	Parker
4	Smith

$\pi_{\text{Works\_in.department\_id, Works\_in.employee\_id}}(\textbf{Works\_in})$

employee_id	department_id
1	1
2	1
2	2
3	3
4	3

Natural join evaluates 4*5 = 20 combinations, result temporary relation consists of 3 columns.

$\pi_{\text{Empoyee.name, Employee.employee\_id}}(\textbf{ Employee }) \bowtie \pi_{\text{Works\_in.department\_id, Works\_in.employee\_id}}(\textbf{Works\_in})$

employee_id	name	department_id
1	Jones	1
2	Smith	1
2	Smith	2
3	Parker	3
4	Smith	3

$\pi_{\text{Empoyee.name, Works\_in.department\_id}}(\pi_{\text{Empoyee.name, Employee.employee\_id}}(\textbf{ Employee }) \bowtie \pi_{\text{Works\_in.department\_id, Works\_in.employee\_id}}(\textbf{Works\_in}))$

name	department_id
Jones	1
Smith	1
Smith	2
Parker	3
Smith	3

Equivalence rules

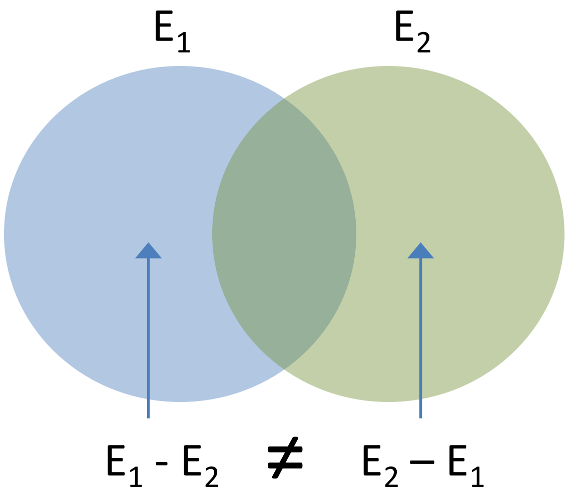
- **Rule 7.** The set operations union and intersections are commutative.

$$E_1 \cup E_2 = E_2 \cup E_1$$

$$E_1 \cap E_2 = E_2 \cap E_1$$

- The set different operation is NOT commutative

$$E_1 - E_2 \neq E_2 - E_1$$



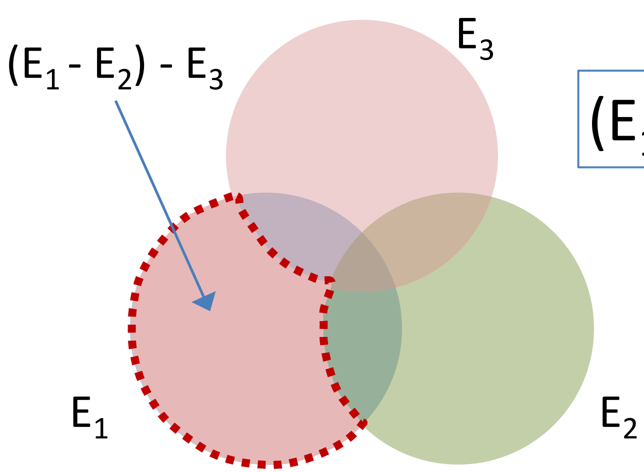
Equivalence rules

- **Rule 8.** The set operations union and intersections are associative.

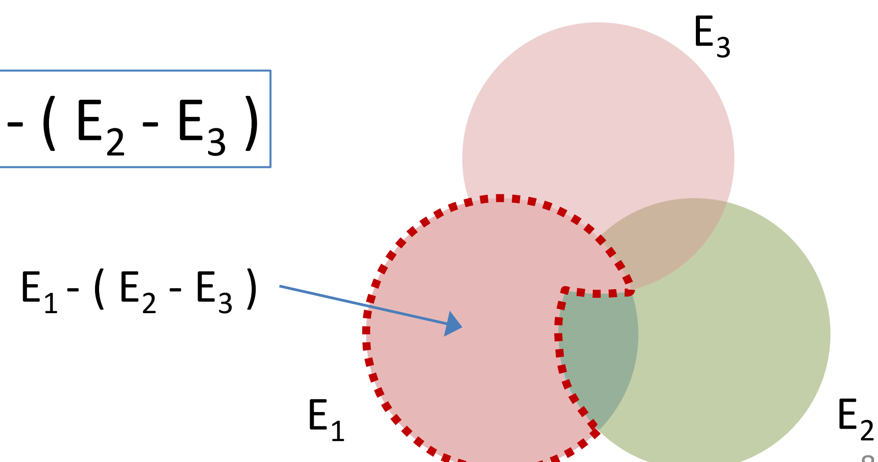
$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$

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

- The set different operation is NOT associative.



$$(E_1 - E_2) - E_3 \neq E_1 - (E_2 - E_3)$$



Equivalence rules

- **Rule 9.** The selection operation distributes over the union, intersection and set difference operations

$$\sigma_p (E_1 \cup E_2) = \sigma_p (E_1) \cup \sigma_p (E_2)$$

$$\sigma_p (E_1 \cap E_2) = \sigma_p (E_1) \cap \sigma_p (E_2)$$

$$\sigma_p (E_1 - E_2) = \sigma_p (E_1) - \sigma_p (E_2)$$

Audio_CD

ID	name	provider_id	stock	#tracks
CD1	One Heart	P1	55	14
CD2	Miracle	P2	4	14

DVD

ID	name	provider_id	stock	length
DVD1	Prince of Persia	P2	3	110
DVD2	Iron man 3	P3	60	90
DVD3	Legend is born: Ip Man	P3	17	90

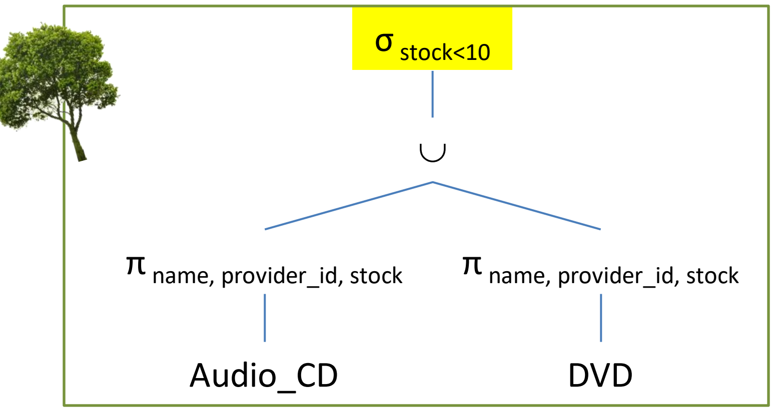
$$\sigma_{stock<10} (\pi_{name,\ provider\_id,\ stock} (Audio\_CD) \cup \pi_{name,\ provider\_id,\ stock} (DVD))$$

equivalent to

$$\sigma_{stock<10} (\pi_{name,\ provider\_id,\ stock} (Audio\_CD)) \cup \sigma_{stock<10} (\pi_{name,\ provider\_id,\ stock} (DVD))$$

Equivalence rules

$\sigma_{stock<10}($
$\quad \pi_{name,\ provider\_id,\ stock}(\ Audio\_CD\ ) \cup$
$\quad \pi_{name,\ provider\_id,\ stock}(\ DVD\ )$
$)$



$\sigma_{stock<10}(\pi_{name,\ provider\_id,\ stock}(\ \textbf{Audio\_CD}\ ) \cup \pi_{name,\ provider\_id,\ stock}(\ \textbf{DVD}\ )\ )$

name	provider_id	stock
Miracle	P2	4
Prince of Persia	P2	3

$\pi_{name,\ provider\_id,\ stock}(\ \textbf{Audio\_CD}\ ) \cup \pi_{name,\ provider\_id,\ stock}(\ \textbf{DVD}\ )$

name	provider_id	stock
One Heart	P1	55
Miracle	P2	4
Prince of Persia	P2	3
Iron man 3	P3	60
Legend is born: Ip Man	P3	17

Audio_CD

ID	name	provider_id	stock	#tracks
CD1	One Heart	P1	55	14
CD2	Miracle	P2	4	14

$\pi_{name,\ provider\_id,\ stock}(\textbf{Audio\_CD})$

name	provider_id	stock
One Heart	P1	55
Miracle	P2	4

DVD

ID	name	provider_id	stock	length
DVD1	Prince of Persia	P2	3	110
DVD2	Iron man 3	P3	60	90
DVD3	Legend is born: Ip Man	P3	17	90

$\pi_{name,\ provider\_id,\ stock}(\textbf{DVD})$

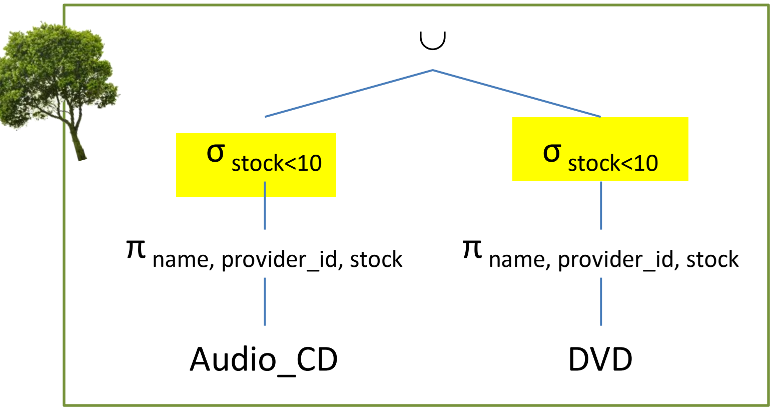
name	provider_id	stock
Prince of Persia	P2	3
Iron man 3	P3	60
Legend is born: Ip Man	P3	17

Equivalence rules

$\sigma_{stock<10}(\pi_{name, provider\_id, stock}(Audio\_CD)) \cup \sigma_{stock<10}(\pi_{name, provider\_id, stock}(DVD))$

name	provider_id	stock
Miracle	P2	4
Prince of Persia	P2	3

$\sigma_{stock<10}(\pi_{name, provider\_id, stock}(Audio\_CD)) \cup$
$\sigma_{stock<10}(\pi_{name, provider\_id, stock}(DVD))$



$\sigma_{stock<10}(\pi_{name, provider\_id, stock}(Audio\_CD))$

name	provider_id	stock
Miracle	P2	4

$\sigma_{stock<10}(\pi_{name, provider\_id, stock}(DVD))$

name	provider_id	stock
Prince of Persia	P2	3

Audio_CD

ID	name	provider_id	stock	#tracks
CD1	One Heart	P1	55	14
CD2	Miracle	P2	4	14

$\pi_{name, provider\_id, stock}(Audio\_CD)$

name	provider_id	stock
One Heart	P1	55
Miracle	P2	4

DVD

ID	name	provider_id	stock	length
DVD1	Prince of Persia	P2	3	110
DVD2	Iron man 3	P3	60	90
DVD3	Legend is born: Ip Man	P3	17	90

$\pi_{name, provider\_id, stock}(DVD)$

name	provider_id	stock
Prince of Persia	P2	3
Iron man 3	P3	60
Legend is born: Ip Man	P3	17

Equivalence rules

- **Rule 10.** The projection operation distributes over the union operation

$$\pi_L (E_1 \cup E_2) = \pi_L (E_1) \cup \pi_L (E_2)$$

- Projection does not distribute over intersection and set difference.

$$\pi_L (E_1 \cap E_2) \neq \pi_L (E_1) \cap \pi_L (E_2)$$

$$\pi_L (E_1 - E_2) \neq \pi_L (E_1) - \pi_L (E_2)$$

Equivalence rules

Why projection does not distribute over intersection and set difference?

To show that projection does not distribute over intersection, you only need to provide a counter example.

R

A	B
1	1

S

A	B
1	2

$\pi_A(R)$

A
1

$\pi_A(S)$

A
1

$R \cap S$

A	B

Empty set

$\pi_A(R \cap S)$

A

Empty set

$\pi_A(R) \cap \pi_A(S)$

A
1

This counter example shows that

$$\pi_A(R \cap S) \neq \pi_A(R) \cap \pi_A(S)$$

Now, can you try to construct a counter example to show that projection does not distribute over **set difference**?

Section 6

Example of query optimization

Transformation

- Find the names of all instructors in the CS department (dpt_id = 1) who have taught a course in 2nd semester, together with the course title of all the courses that the instructors teach.

```
SELECT I.name, C.title
FROM Instructor I, Teaches T, Course C
WHERE I.dpt_id = 1  AND
        T.sem=2 AND
        I.instructor_id = T.instructor_id AND
        T.course_id = C.course_id ;
```

SQL

$$\pi_{I.name,C.title} \Big(\sigma_{I.dpt\_id=1 \wedge T.sem=2} \big(\rho_I(\text{Instructor}) \bowtie (\rho_T(\text{Teaches}) \bowtie \rho_C(\text{Course})) \big) \Big)$$

Relational algebra

Instructor

instructor_id	name	dpt_id
1	Kit	1
2	Ben	1
3	Michael	2
4	William	3

Teaches

instructor_id	course_id	sem
1	1	1
1	2	2
2	4	1
3	3	2

Course

course_id	title	credit
1	Intro to DB	6
2	Programming I	6
3	Accounting	6
4	Algorithms	6

Transformation

$\pi_{I.name,C.title}($
$\sigma_{I.dpt\_id=1 \wedge T.sem=2}($
$\rho_I($ **Instructor**$) \bowtie (\rho_T($ **Teaches**$) \bowtie \rho_C($ **Course**$))$
$)$
$)$

equivalent to

$\pi_{I.name,C.title}($
$\sigma_{I.dpt\_id=1 \wedge T.sem=2}($
$\rho_I($ **Instructor**$) \bowtie$
$(\rho_T($ **Teaches**$) \bowtie \pi_{C.course\_id, C.title}(\rho_C($ **Course**$)))$
$)$
$)$

Rule 6

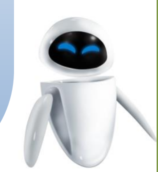
$\pi_{L1 \cup L2}(E_1 \bowtie E_2) =$
$\pi_{L1 \cup L2}((\pi_{L1 \cup L3}(E_1)) \bowtie (\pi_{L2 \cup L3}(E_2)))$

BEFORE

$\pi_{I.name,C.title}$
|
$\sigma_{I.dpt\_id=1 \wedge T.sem=2}$
|
$\bowtie$ (I, $\bowtie$ (T, C))

Let's try to push the projection $\pi_{C.title}$ downward and apply it ahead of the natural joins.

Since this natural join requires C.course_id = T.course_id, therefore, **we have to add the joining attribute C.course_id to make the projection $\pi_{C.course\_id,C.title}$**



AFTER

$\pi_{I.name,C.title}$
|
$\sigma_{I.dpt\_id=1 \wedge T.sem=2}$
|
$\bowtie$ (I, $\bowtie$ (T, $\pi_{C.course\_id,C.title}$ (C)))

Transformation

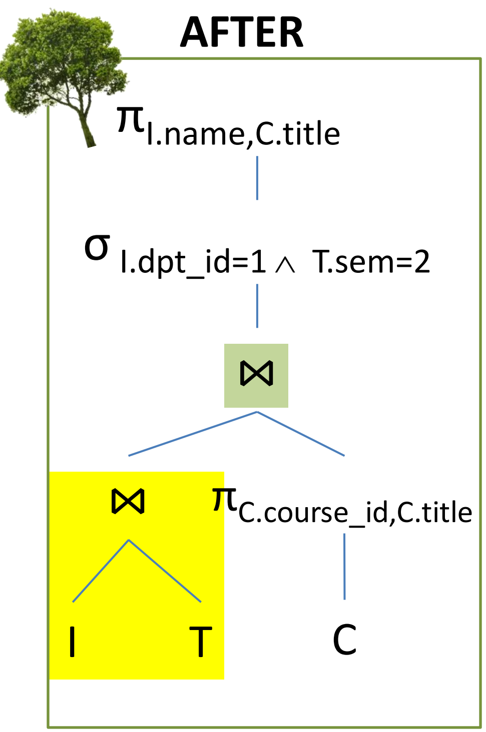
$\pi_{I.name,C.title}($
$\quad\sigma_{I.dpt\_id=1 \wedge T.sem=2}($
$\quad\quad(\rho_I(\textbf{Instructor}) \bowtie \rho_T(\textbf{Teaches}))$
$\quad\quad\bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))$
$\quad)$
$)$

equivalent to

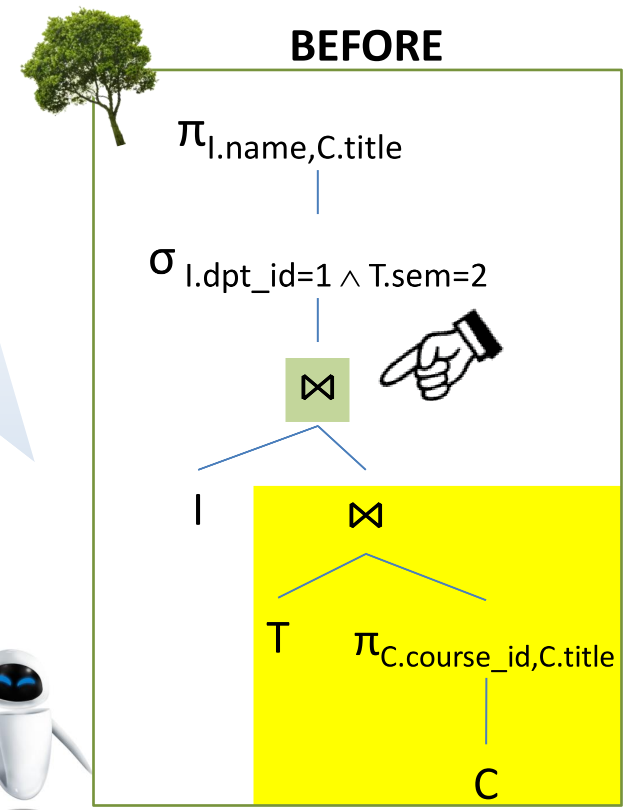
$\pi_{I.name,C.title}($
$\quad\sigma_{I.dpt\_id=1 \wedge T.sem=2}($
$\quad\quad\rho_I(\textbf{Instructor}) \bowtie$
$\quad\quad(\rho_T(\textbf{Teaches}) \bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course})))$
$\quad)$
$)$

Rule 4

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



- Now we would like to push the selection down to reduce the size of the temporary result of the natural join.
- As the selection involves relations I and T only, we would like to rearrange the natural joins to make I and T under one natural join.
- Since natural joins are associative, we can make such rearrangement.



Transformation

$$\pi_{I.name,C.title}(\ \sigma_{I.dpt\_id=1\ \wedge\ T.sem=2}(\ (\rho_I(\textbf{Instructor}) \bowtie \rho_T(\textbf{Teaches})) \bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))\ )\ )$$

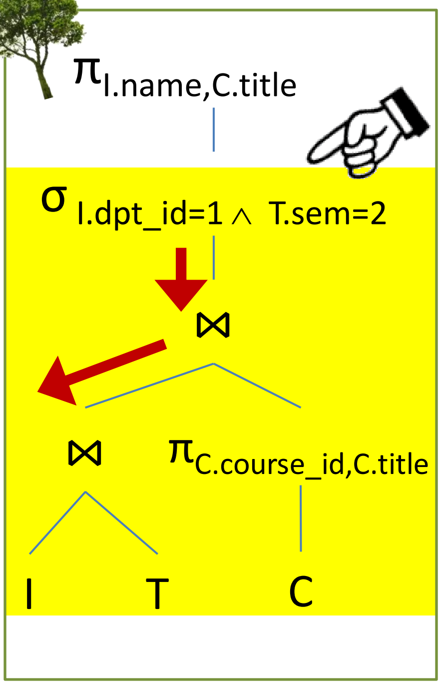
equivalent to

$$\pi_{I.name,C.title}(\ (\ \sigma_{I.dpt\_id=1\ \wedge\ T.sem=2}(\ \rho_I(\textbf{Instructor}) \bowtie \rho_T(\textbf{Teaches})\ )\ ) \bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))\ )$$

Rule 5a

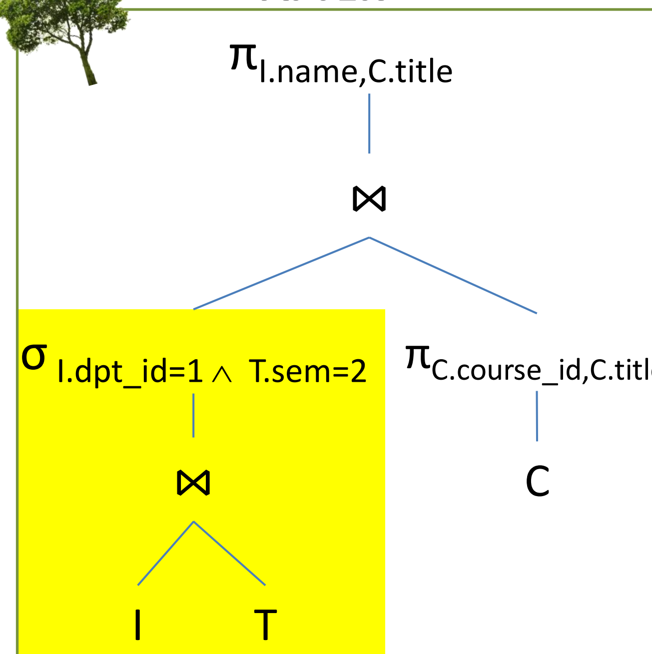
$$\sigma_{p1}(E_1 \bowtie E_2) = (\sigma_{p1}(E_1) \bowtie E_2)$$

BEFORE



- Now we can push the selection down one level.
- According to Rule 5a, we can distribute both selection predicates to the L.H.S. of the selection as the R.H.S. does not contain any attribute in the selection predicate.

AFTER



Transformation

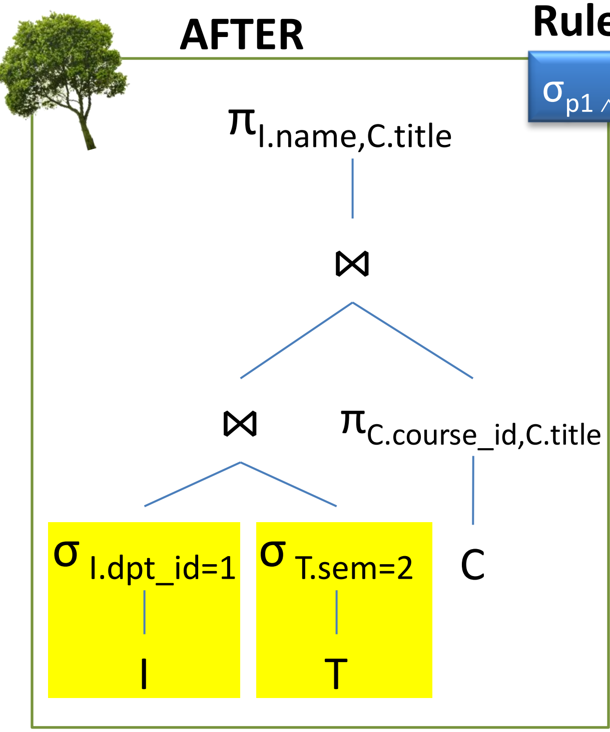
$$\pi_{I.name,C.title}\Big(\big(\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor})) \bowtie \sigma_{T.sem=2}(\rho_T(\textbf{Teaches})) \big) \bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course})) \Big)$$

equivalent to

$$\pi_{I.name,C.title}\Big(\big(\sigma_{I.dpt\_id=1\ \wedge\ T.sem=2}(\rho_I(\textbf{Instructor}) \bowtie \rho_T(\textbf{Teaches})) \big) \bowtie \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course})) \Big)$$

Rule 5b

$$\sigma_{p1\ \wedge\ p2}(E_1 \bowtie E_2) = (\sigma_{p1}(E_1) \bowtie \sigma_{p1}(E_2))$$



- Now we can further push the selection one more level down by applying rule 5b.
- According to Rule 5b, we can distribute
  - $\sigma_{I.dpt\_id=1}$ to I
  - $\sigma_{T.sem=2}$ to T

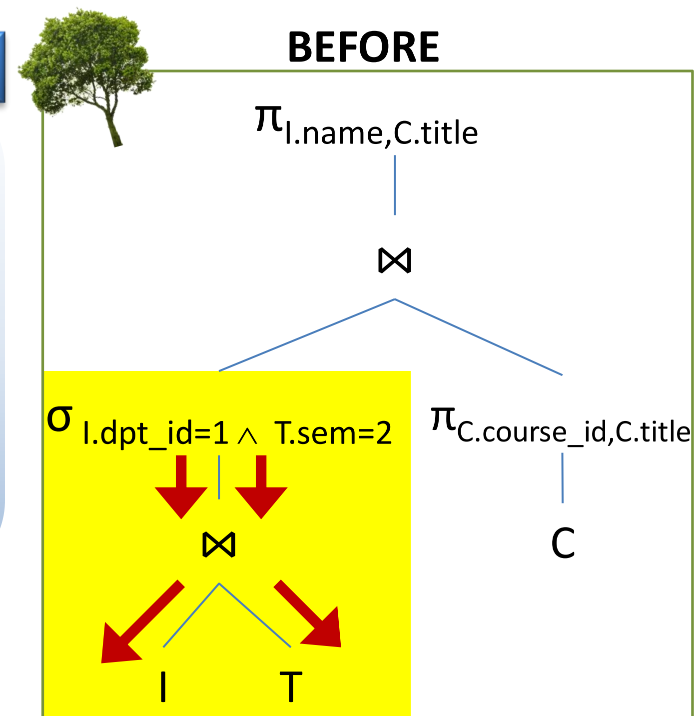


Illustration (original tree)

$\pi_{I.name,C.title}($

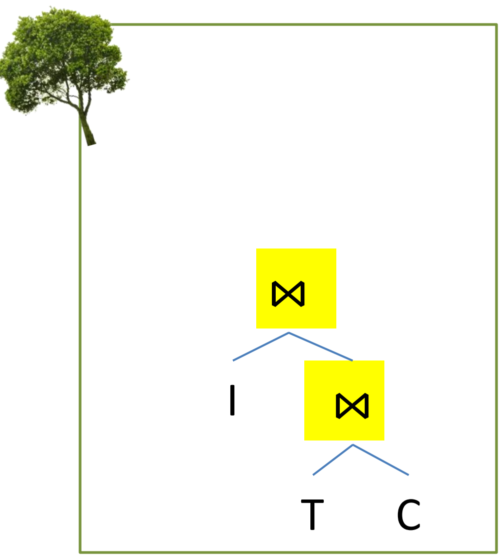
$\quad\sigma_{I.dpt\_id=1 \wedge T.sem=2}($

$\quad\quad\rho_I(\textbf{Instructor}) \bowtie$

$\quad\quad(\rho_T(\textbf{Teaches}) \bowtie \rho_C(\textbf{Course}))$

$\quad)$

$)$



$\rho_I(\textbf{Instructor}) \bowtie (\rho_T(\textbf{Teaches}) \bowtie \rho_C(\textbf{Course}))$

instructor_id	name	dpt_id	course_id	sem	title	credit
1	Kit	1	1	1	Intro to DB	6
1	Kit	1	2	2	Programming I	6
2	Ben	1	4	1	Algorithms	6
3	Michael	2	3	2	Accounting	6

Natural join evaluates 4*4 = 16 combinations.

$\rho_T(\textbf{Teaches}) \bowtie \rho_C(\textbf{Course})$

instructor_id	name	dpt_id	course_id	sem
1	Kit	1	1	1
1	Kit	1	2	2
2	Ben	1	4	1
3	Michael	2	3	2

Natural join evaluates 4*4 = 16 combinations.

Instructor

instructor_id	name	dpt_id
1	Kit	1
2	Ben	1
3	Michael	2
4	William	3

Teaches

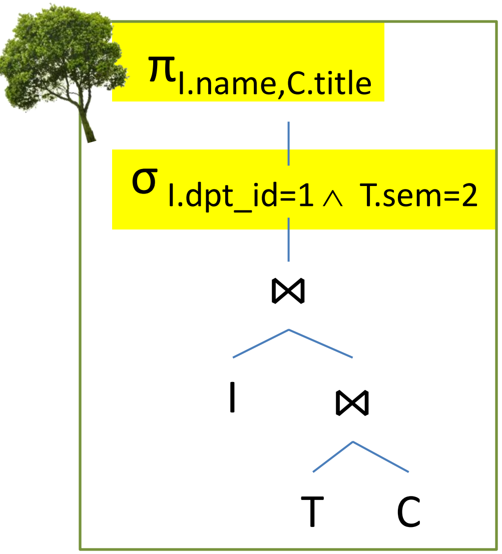
instructor_id	course_id	sem
1	1	1
1	2	2
2	4	1
3	3	2

Course

course_id	title	credit
1	Intro to DB	6
2	Programming I	6
3	Accounting	6
4	Algorithms	6

Illustration (original tree)

$\pi_{I.name,C.title}$ (
  $\sigma_{I.dpt\_id = 1 \wedge T.sem=2}$(
    ρ_I(**Instructor**) ⋈
    (ρ_T(**Teaches**) ⋈ ρ_C(**Course**))
)
)



ρ_I(**Instructor**) ⋈ (ρ_T(**Teaches**) ⋈ ρ_C(**Course**))

instructor_id	name	dpt_id	course_id	sem	title	credit
1	Kit	1	1	1	Intro to DB	6
1	Kit	1	2	2	Programming I	6
2	Ben	1	4	1	Algorithms	6
3	Michael	2	3	2	Accounting	6

$\sigma_{I.dpt\_id = 1 \wedge T.sem=2}$(ρ_I(**Instructor**) ⋈ (ρ_T(**Teaches**) ⋈ ρ_C(**Course**)))

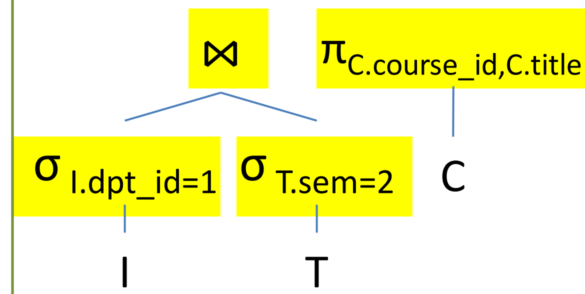
instructor_id	name	dpt_id	course_id	sem	title	credit
1	Kit	1	2	2	Programming I	6

$\pi_{I.name,C.title}$ ($\sigma_{I.dpt\_id = 1 \wedge T.sem=2}$(ρ_I(**Instructor**) ⋈ (ρ_T(**Teaches**) ⋈ ρ_C(**Course**))))

name	title
Kit	Programming I

Illustration (transformed tree)

$$\pi_{I.name,C.title}\Big(\big(\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor})) \bowtie \sigma_{T.sem=2}(\rho_T(\textbf{Teaches})) \big) \bowtie \pi_{C.course\_id,\,C.title}(\rho_C(\textbf{Course})) \Big)$$



$\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor})) \bowtie \sigma_{T.sem=2}(\rho_T(\textbf{Teaches}))$

instructor_id	name	dpt_id	course_id	sem
1	Kit	1	2	2

Natural join evaluates 2*2 = 4 combinations.

$\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor}))$

instructor_id	name	dpt_id
1	Kit	1
2	Ben	1

$\sigma_{T.sem=2}(\rho_T(\textbf{Teaches}))$

instructor_id	course_id	sem
1	2	2
3	3	2

$\pi_{C.course\_id,\,C.title}(\rho_C(\textbf{Course}))$

course_id	title
1	Intro to DB
2	Programming I
3	Accounting
4	Algorithms

Instructor

instructor_id	name	dpt_id
1	Kit	1
2	Ben	1
3	Michael	2
4	William	3

Teaches

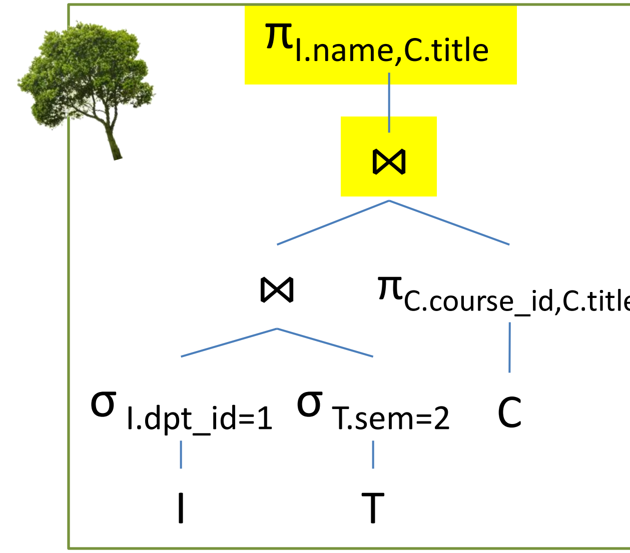
instructor_id	course_id	sem
1	1	1
1	2	2
2	4	1
3	3	2

Course

course_id	title	credit
1	Intro to DB	6
2	Programming I	6
3	Accounting	6
4	Algorithms	6

Illustration (transformed tree)

$\pi_{I.name,C.title}($
$\quad(\ \sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor}))$
$\qquad\bowtie$
$\quad\ \sigma_{T.sem=2}\ (\rho_T(\textbf{Teaches}))$
$\quad)$
$\quad\bowtie\ \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))$
$)$



$\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor}))\bowtie\ \sigma_{T.sem=2}(\rho_T(\textbf{Teaches}))$

instructor_id	name	dpt_id	course_id	sem
1	Kit	1	2	2

Natural join evaluates 1*4 = 4 combinations.

$\pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))$

course_id	title
1	Intro to DB
2	Programming I
3	Accounting
4	Algorithms

$\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor}))\bowtie\ \sigma_{T.sem=2}(\rho_T(\textbf{Teaches}))\bowtie\pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course}))$

instructor_id	name	dpt_id	course_id	sem	title
1	Kit	1	2	2	Programming I

$\pi_{I.name,C.title}(\sigma_{I.dpt\_id=1}(\rho_I(\textbf{Instructor}))\bowtie\ \sigma_{T.sem=2}(\rho_T(\textbf{Teaches}))\bowtie\ \pi_{C.course\_id,\ C.title}(\rho_C(\textbf{Course})))$

name	title
Kit	Programming I

Summary

- **Relational algebra (RA) defines a set of algebraic operations on tables, and output tables as result.**
  - 6 fundamental operations
  - Additional operations does not extend the power of the fundamental operators, but they simplify the expression.
  - Extended operations add expressive power.
- **Relational algebra (RA) is the basics of query optimization.**

Lecture 6

END

COMP3278

Introduction to Database Management Systems

Dr. Ping Luo
Email : pluo@cs.hku.hk

Department of Computer Science, The University of Hong Kong