Chapter 4

Relational Algebra

COMP3278C
Introduction to Database Management Systems

Dr. CHEN, Yi
Email: chenyi1@hku.hk

School of Computing & Data Science, The University of Hong Kong

Acknowledgement: Dr. Chui Chun Kit, Dr. Reynold Cheng, Dr. Ping Luo

Why do we learn Relational Algebra?

Query: Find the department names where employees named Smith work.

Employee

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

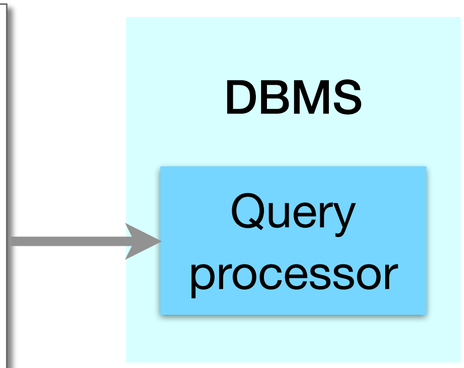
Works_in

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

Department

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

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



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

Relational Algebra (RA)

- The low-level operations of a relational DBMS are based on **Relation Algebra (RA)** operations.
- **Relation Algebra (RA)** is analogous to algebra in maths. e.g., 2+3*x-y; it uses **relations** (**tables**) as operand, and a set of **operators.**

Basic RA operators

- Select (σ)
- Project (π)
- Union ($\cup$)
- Set difference (-)
- Cartesian product ($\times$)
- Rename (ρ)

In DBMS, each of these operators are implemented as programs/functions.

Select (σ)

- $\sigma_p(R) = \{t \mid t \in R \wedge p(t)\}$

- Example
  - Select all authors named “May”.

Author

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

```
SELECT *
FROM Author
WHERE name = “May”;
```

→ Query processor → $\sigma_{name='May'}(Author)$

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

Project (π)

- $\pi_{A_1,A_2,...,A_k}(R)$
  - A copy of R with only listed attributes A_1 to A_k.
- Example
  - Report only the bookID and title of all books

Book

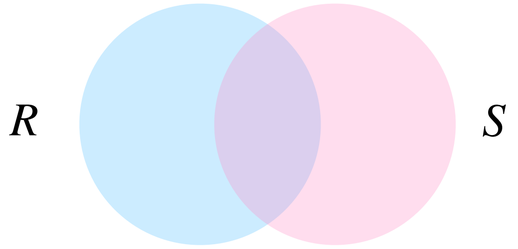
bookID	title	publisher
115	Stuffy doll	ABC
116	Angel's feather	MTG
117	Little girl	MGH
118	Myr	ABC

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

→ Query processor → $\pi_{bookID,title}(Book)$

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 domains compatible.
- Example
  - Find the name of all products in Audio_CD and DVD tables.

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

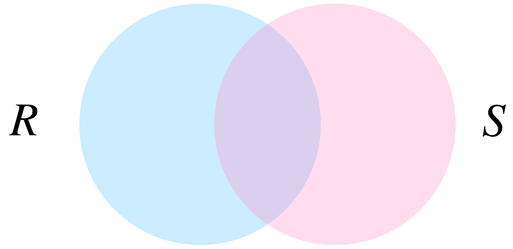
```
SELECT name
FROM Audio_CD
UNION
SELECT name
FROM DVD;
```

→ Query processor

↓

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

Union (∪)



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

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

$$\pi_{name}(Audio\_CD)$$

name
One Heart
Miracle

$$\pi_{name}(DVD)$$

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

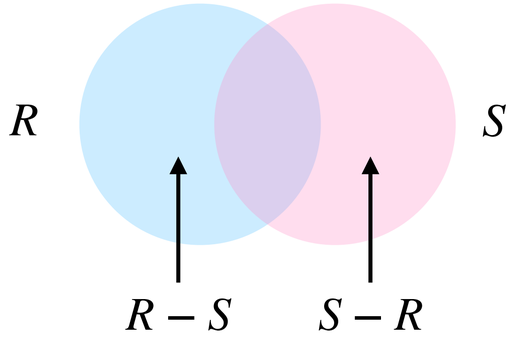
Audio_CD

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 (-)



- $R - S = \{t \mid t \in R \wedge t \notin S\}$
  - R and S must have the same number of attributes and attribute domains compatible.
- Example
  - Find the ID of the students who haven't submitted the assignment.

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

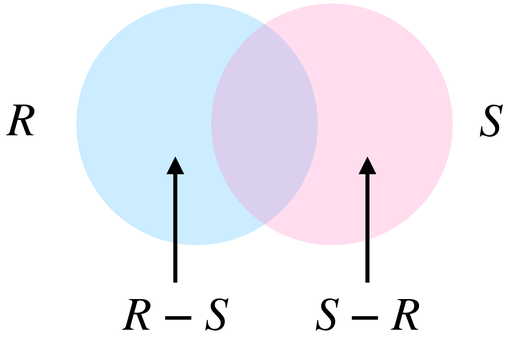
```
SELECT student_id
FROM Student
EXCEPT
SELECT student_id
FROM Submit;
```

→ Query processor

↓

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

Set difference (-)



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

student_id
123

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

student_id
123
456
789

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

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 (×)

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

Tutorial

tutorial_id	course_id	date
1	c1119	5/9
1	c3278c	7/9
2	c3278c	15/9

Course

course_id	name
c1119	DataStructures and Algorithms
c3278c	Introduction of Database Management Systems

```
SELECT Tutorial.date
FROM Course, Tutorial
WHERE
Course.name = “Introduction to Database
Management Systems” AND
Course.course_id = Tutorial.course_id
```

→ Query processor

↓

$$\pi_{Tutorial.date}(\sigma_{Course.name='IntroductiontoDatabaseManagementSystem'}(\sigma_{Course.course\_id=Tutorial.course\_id}(Course \times Tutorial)))$$

$$\pi_{Tutorial.date}(\sigma_{Course.name='IntroductiontoDatabaseManagementSystems'}(\sigma_{Course.course\_id=Tutorial.course\_id}(Course \times Tutorial)))$$

Tutorial.date
7/9
15/9

$$\sigma_{Course.name='IntroductiontoDatabaseManagementSystems'}(\sigma_{Course.course\_id=Tutorial.course\_id}(Course \times Tutorial))$$

Course.course_id	Course.name	Tutorial.tutorial_id	Tutorial.course_id	Tutorial.date
c3278c	Introduction of Database Management Systems	1	c3278c	7/9
c3278c	Introduction of Database Management Systems	2	c3278c	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	DataStructures and Algorithms	1	c1119	5/9
c3278c	Introduction of Database Management Systems	1	c3278c	7/9
c3278c	Introduction of Database Management Systems	2	c3278c	15/9

$$Course \times Tutorial$$

Course.course_id	Course.name	Tutorial.tutorial_id	Tutorial.course_id	Tutorial.date
c1119	DataStructures and Algorithms	1	c1119	5/9
c1119	DataStructures and Algorithms	1	c3278c	7/9
c1119	DataStructures and Algorithms	2	c3278c	15/9
c3278c	Introduction of Database Management Systems	1	c1119	5/9
c3278c	Introduction of Database Management Systems	1	c3278c	7/9
c3278c	Introduction of Database Management Systems	2	c3278c	15/9

Course

course_id	name
c1119	DataStructures and Algorithms
c3278c	Introduction of Database Management Systems

Tutorial

tutorial_id	course_id	date
1	c1119	5/9
1	c3278c	7/9
2	c3278c	15/9

Rename (ρ)

- $\rho_X(E)$
  - 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
Course.name = “Introduction to Database
Management Systems” AND
Course.course_id = Tutorial.course_id
```

$$\pi_{Tutorial.date}(\sigma_{Course.name='IntroductiontoDatabaseManagementSystem'}(\sigma_{Course.course\_id=Tutorial.course\_id}(Course \times Tutorial)))$$

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

$$\pi_{T.date}(\sigma_{C.name='IntroductiontoDatabaseManagementSystem'}(\sigma_{C.course\_id=T.course\_id}(\rho_C(Course) \times \rho_T(Tutorial))))$$

Basic RA operators

- Select (σ)
- Project (π)
- Union ($\cup$)
- Set difference (-)
- Cartesian product ($\times$)
- Rename (ρ)

Exercise time

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,CID, age: integer; name, title: varchar*
- Q: Which of the following is (are) valid Relational Algebra expression(s)?
  - $\pi_{UID}(Student) - \pi_{CID}(Course)$
  - $Course - \pi_{UID}(Enroll)$
  - $\sigma_{age<18}(Student \cup Course)$
  - $\sigma_{age<18}(\pi_{UID,name}(Student))$

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,CID, age: integer; name, title: varchar*
- Q: Which of the following is (are) valid Relational Algebra expression(s)?
  - ✅ $\pi_{UID}(Student) - \pi_{CID}(Course)$
  - ❌ $Course - \pi_{UID}(Enroll)$
  - ❌ $\sigma_{age<18}(Student \cup Course)$
  - $\sigma_{age<18}(\pi_{UID,name}(Student))$

- *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,CID, age: integer; name, title: varchar*
- Q: Which of the following is (are) valid Relational Algebra expression(s)?

- $\pi_{UID}(Student) - \pi_{CID}(Course)$

- $Course - \pi_{UID}(Enroll)$

- $\sigma_{age<18}(Student \cup Course)$

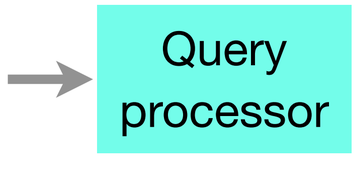
- $\sigma_{age<18}(\pi_{UID,name}(Student))$

- *The left and right parts of Set difference have to be comparable (same number of attributes).*
- *No attribute "age" for selection*

Question 2

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

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



Question 2

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

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

→ Query processor →

1. $\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$
2. $\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

Question 2

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

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

→ Query processor →

1. $\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$
2. $\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

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

1 ←

$\sigma_{employee\_id<100}(Employee)$

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

↓

$\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$

employeeID	name
97	May
98	Felix
99	May

Question 2

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

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

→ Query processor →

1. $\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$
2. $\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

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

1 ←

$\sigma_{employee\_id<100}(Employee)$

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

↓

$\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$

employeeID	name
97	May
98	Felix
99	May

2 →

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

employeeID	name
97	May
98	Felix
99	May
100	George

↓

$\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

employeeID	name
97	May
98	Felix
99	May

Question 2

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

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

→ Query processor →

1. $\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$
2. $\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

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

1 ←

$\sigma_{employee\_id<100}(Employee)$

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

↓

$\pi_{name,employee\_id}(\sigma_{employee\_id<100}(Employee))$

employeeID	name
97	May
98	Felix
99	May

2 →

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

employeeID	name
97	May
98	Felix
99	May
100	George

↓

$\sigma_{employee\_id<100}(\pi_{name,employee\_id}(Employee))$

employeeID	name
97	May
98	Felix
99	May

Which one is better?

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Q: How many tuples of this intermediate relation created?

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Q: How many tuples of this intermediate relation created?

*A: 10,000 * 5 = 50,000 tuples*

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Q: How many tuples of this intermediate relation created?

*A: 10,000 * 5 = 50,000 tuples*

Q: Can we reduce the size of intermediate relation?

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Option 2: $\pi_{W.department\_id}(\sigma_{E.employee\_id=W.employee\_id}(\sigma_{E.name='Smith}(\rho_E(Employee)) \times \rho_W(Works\_in)))$

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Option 2: $\pi_{W.department\_id}(\sigma_{E.employee\_id=W.employee\_id}(\sigma_{E.name='Smith}(\rho_E(Employee)) \times \rho_W(Works\_in)))$

Q: How many tuples of this intermediate relation created?

Question 3

- Find the department ID where employees named Smith work

Employee

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

(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

```
SELECT W.department_id
FROM Employee E, Works_in W AND
        E.employee_id = W.employee_id;
```

Option 1: $\pi_{W.department\_id}(\sigma_{E.name='Smith'}(\sigma_{E.employee\_id=W.employee\_id}(\rho_E(Employee) \times \rho_W(Works\_in))))$

Option 2: $\pi_{W.department\_id}(\sigma_{E.employee\_id=W.employee\_id}(\sigma_{E.name='Smith}(\rho_E(Employee)) \times \rho_W(Works\_in)))$

Q: How many tuples of this intermediate relation created?

*A: 2 * 5 = 10 tuples*

Question 4

Borrower

customer_id	load_id
C1	L3
C4	L2
C2	L1

Owner

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

- 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 Owner;
```

- 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 Owner;
```

Question 4

Borrower

customer_id	load_id
C1	L3
C4	L2
C2	L1

Owner

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

- 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 Owner;
```

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

- 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 Owner;
```

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

Question 4

Borrower

customer_id	load_id
C1	L3
C4	L2
C2	L1

Owner

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

- 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 Owner;
```

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

- 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 Owner;
```

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

Wait! Do we have set intersection in relational algebra?

Question 4

Borrower

customer_id	load_id
C1	L3
C4	L2
C2	L1

Owner

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

- 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 Owner;
```

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

- 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 Owner;
```

✗ $\pi_{customer\_id}(Borrower) \cap \pi_{customer\_id}(Owner)$

Wait! Do we have set intersection in relational algebra?

✓ $\pi_{customer\_id}(Borrower) - (\pi_{customer\_id}(Borrower) - \pi_{customer\_id}(Owner))$

Certain common queries are lengthy to express with basic operators.

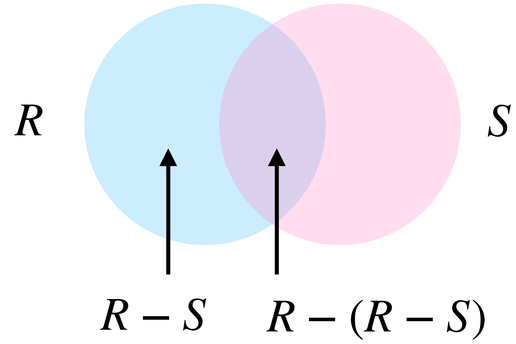
Additional Operators

- The basic operators are sufficient to express a query.
- Additional operators do not add any power to algebra, but simplify common queries.
  - For each additional operator, there is an equivalent expression that use only the basic operators.

Additional operators

- Set intersection (∩)
- Natural join (⋈)
- Assignment (←)
- Left outer join (⟕), Right outer join (⟖)
- Division (÷)
- …

Set intersection (∩)



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

```
SELECT employee_id
FROM Works_in
WHERE department_id = 1
INTERSECT
SELECT employee_id
FROM Works_in
WHERE department_id = 3
```

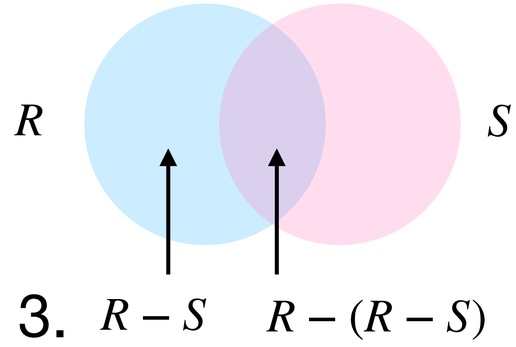
→ Query processor →

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

=

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

Set intersection (∩)



- Find the employee_id of employees who work in department 1 and 3.

```
SELECT employee_id
FROM Works_in
WHERE department_id = 1
INTERSECT
SELECT employee_id
FROM Works_in
WHERE department_id = 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
```

Query processor

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

Natural join (⋈)

Course

course_id	name
c1119	DataStructures and Algorithms
c3278c	Introduction of Database Management Systems

Tutorial

tutorial_id	course_id	date
1	c1119	5/9
1	c3278c	7/9
2	c3278c	15/9

Course × *Tutorial*

Course.course_id	Course.name	Tutorial.tutorial_id	Tutorial.course_id	Tutorial.date
c1119	DataStructures and Algorithms	1	c1119	5/9
c1119	DataStructures and Algorithms	1	c3278c	7/9
c1119	DataStructures and Algorithms	2	c3278c	15/9
c3278c	Introduction of Database Management Systems	1	c1119	5/9
c3278c	Introduction of Database Management Systems	1	c3278c	7/9
c3278c	Introduction of Database Management Systems	2	c3278c	15/9

Course ⋈ *Tutorial*

Course.course_id	Course.name	Tutorial.tutorial_id	Tutorial.course_id	Tutorial.date
c1119	DataStructures and Algorithms	1	c1119	5/9
c3278c	Introduction of Database Management Systems	1	c3278c	7/9
c3278c	Introduction of Database Management Systems	2	c3278c	15/9

Natural join ($\bowtie$)

- Usually, a query that involves a Cartesian product includes a selection operation on the result of the Cartesian product.
  - The selection operation often requires that all attributes that are common to the relations that are involved in the Cartesian product be equated.
- The schema of $R \bowtie S$ is R-schema $\cup$ S-schema (repeated attributes are removed)
  - For each pair of tuple 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 $t(t_r \cup t_s)$ will be added to the result $R \bowtie S$.
- $R \bowtie S = \pi_{R \cup S}(\sigma_{R.A_1=S.A_1 \wedge R.A_2=S.A_2 \wedge \ldots \wedge R.A_n=S.A_n}(R \times S))$

  where $R \cap S = \{A_1, A_2, \ldots, A_n\}$

Natural join ($\bowtie$)

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: $R \cup S = \{A, B, C\}$
  - $R \bowtie S = \pi_{R \cup S}(\sigma_{R.A_1 = S.A_1 \wedge R.A_2 = S.A_2 \wedge \ldots \wedge R.A_n = S.A_n}(R \times S))$
  - $R \bowtie S = \pi_{A,B,C}(\sigma_{R.A = S.A}(R \times S))$

$R \times S$

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

→

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

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

→

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

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

Assignment (←)

- When a RA expression is very long, it is convenient to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The assignment operator, denoted by ←, works like assignment operator = in programming language.
- Example
  - $temp_1 \leftarrow \pi_a(R)$
  - $temp_2 \leftarrow \pi_a(S)$
  - $result \leftarrow temp_1 - temp_2$
    - $temp_1, temp_2, result$ are called “relation variable”
    - With the assignment operator, a query can be written as a sequential program.

Outer Join

- Outer join extends join to handling missing information

Customer

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

Owner

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

$Customer \bowtie Owner$

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 result in a table without the info of customers (e.g., C3) 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 $\ltimes$

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

Left outer join ⟕

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

Customer

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

Owner

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

$$Customer \leftouterjoin Owner = (Customer \bowtie Owner) \cup (Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$$

Left outer join ⟕

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

Customer

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

Owner

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

$Customer \bowtie Owner$

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

$$Customer \text{ ⟕ } Owner = (Customer \bowtie Owner) \cup (Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$$

Left outer join ⟕

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

Customer

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

Owner

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

$Customer \bowtie Owner$

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

$$Customer \leftouterjoin Owner = (Customer \bowtie Owner) \cup (Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$$

$$Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)$$

customer_id	name	address
C3	Jolly	CB311

Finding missed tuples in the natural join

e.g., C3 in Customer doesn't have any records in Owner, so we use this expression to "recover" Jolly's records.

Left outer join ⟕

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

Customer

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

Owner

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

$Customer \bowtie Owner$

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

$Customer \text{ ⟕ } Owner = (Customer \bowtie Owner) \cup (Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$

$Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)$

customer_id	name	address
C3	Jolly	CB311

$Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$

customer_id	name	address	account_id
C3	Jolly	CB311	*null*

Constructing the missing tuple by adding null value to extra attributes

The $\times \{(null, \ldots, null)\}$ part adds the remaining column values as null because there are no matching records in S (i.e., Depositor).

Left outer join ⟕

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

Customer

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

Owner

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

$Customer \bowtie Owner$

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

$$Customer \leftouterjoin Owner = (Customer \bowtie Owner) \cup (Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$$

$Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)$

customer_id	name	address
C3	Jolly	CB311

$Customer - \pi_{customer\_id,name,address}(Customer \bowtie Owner)) \times \{(null)\}$

customer_id	name	address	account_id
C3	Jolly	CB311	*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

Right outer join $\bowtie\!\!\!\!\subset$

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

Right outer join ⟖

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

Customer

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

Owner

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

$$Customer \rtimes Owner = (Customer \bowtie Owner) \cup (Owner - \pi_{account\_id, customer\_id}(Customer \bowtie Owner)) \times \{(null, null)\}$$

Right outer join ⟖

- $R ⟖ S = (R ⋈ S) ∪ (S - \pi_S(R ⋈ S)) × \{(null, \ldots, null)\}$

Customer

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

Owner

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

$Customer ⋈ Owner$

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

$$Customer ⟕ Owner = (Customer ⋈ Owner) ∪ (Owner - \pi_{account\_id, customer\_id}(Customer ⋈ Owner)) × \{(null, null)\}$$

Right outer join ⟖

- $R ⟖ S = (R ⋈ S) ∪ (S - \pi_S(R ⋈ S)) × \{(null, \ldots, null)\}$

Customer

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

Owner

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

$Customer ⋈ Owner$

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

$Customer ⟕ Owner = (Customer ⋈ Owner) ∪ (Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)) × \{(null, null)\}$

$Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)$

account_id	customer_id
C5	A4

Finding missed tuples in the natural join

Right outer join ⟖

- $R ⟖ S = (R ⋈ S) ∪ (S - \pi_S(R ⋈ S)) \times \{(null, \ldots, null)\}$

Customer

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

Owner

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

$Customer ⋈ Owner$

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

$$Customer ⟖ Owner = (Customer ⋈ Owner) ∪ (Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)) \times \{(null, null)\}$$

$$Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)$$

account_id	customer_id
C5	A4

$$Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)) \times \{(null, null)\}$$

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

Constructing the missing tuple by adding null value to extra attributes

Right outer join ⟖

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

Customer

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

Owner

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

$Customer ⋈ Owner$

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

$$Customer ⟖ Owner = (Customer ⋈ Owner) \cup (Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)) \times \{(null, null)\}$$

$Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)$

account_id	customer_id
C5	A4

$Owner - \pi_{account\_id,customer\_id}(Customer ⋈ Owner)) \times \{(null, null)\}$

account_id	customer_id	name	address
C5	A4	*null*	*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

Chapter 4

END

COMP3278C
Introduction to Database Management Systems

Dr. CHEN, Yi
Email: chenyi1@hku.hk

School of Computing & Data Science, The University of Hong Kong

Acknowledgement: Dr. Chui Chun Kit, Dr. Reynold Cheng, Dr. Ping Luo

Division ÷

- $R \div S$

R

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

S

A	C
1	2
2	1

$R \div S$

A
1
2

- Definition
  - Let $S \subseteq R$
  - $R \div S = \{t \mid t \in \pi_{R-S}(R) \wedge (\forall s \in S, ((t \cup s) \in R))\}$