CSS222 DATABASE SYSTEMS

LAB 3: SET OPERATION AND JOINING

Dr. Wittawin Susutti
wittawin.sus@kmutt.ac.th

Set operations

Find courses that ran in Fall 2017 or in Spring 2018

(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
UNION
(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

Find courses that ran in Fall 2017 and in Spring 2018

(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
INTERSECT
(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

Find courses that ran in Fall 2017 but not in Spring 2018

(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
EXCEPT
(**SELECT** *course_id* **FROM** *section* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

Set operations

- Set operations **UNION, INTERSECT,** and **EXCEPT**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates, use the corresponding multiset versions **UNION ALL, INTERSECT ALL** and **EXCEPT ALL.**
- Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - *m* + *n* times in *r* **UNION ALL** *s*
  - min(*m,n)* times in *r* **INTERSECT ALL** *s*
  - max(0, *m* – *n)* times in *r* **EXCEPT ALL** *s*

Duplicates

Multiset versions of some of the relational algebra operators – given multiset relations r_1 and r_2:

- $\sigma_\theta(r_1)$**:** If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$.
- $\Pi_A(r)$**:** For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.
- $r_1 \times r_2$**:** If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are $c_1 \times c_2$ copies of the tuple t_1. t_2 in $r_1 \times r_2$

Duplicates

- Example: Suppose multiset relations r_1 (*A, B*) and r_2 (*C*) are as follows:

  r_1 = {(1, *a*) (2,*a*)} r_2 = {(2), (3), (3)}

- Then $\Pi_B(r_1)$ would be {*a*, *a*}, while $\Pi_B(r_1)$ x r_2 would be

  {(*a*,2), (*a*,2), (*a*,3), (*a*,3), (*a*,3), (*a*,3)}

- SQL duplicate semantics:

  SELECT $A_1, A_2, \ldots, A_n$

  FROM $r_1, r_2, \ldots, r_m$

  is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Joined Relations

- **Join operations** take two relations and return as a result another relation.

- A join operation is a **Cartesian product** which requires that tuples in the two relations match.

SELECT *name*, *course_id*
FROM *student* **JOIN** *takes*

SELECT *name*, *course_id*
FROM *student, takes*

Inner Join in SQL

- If a value exists for the ID column in one table but not the other, then the join fails for the rows containing that value and those rows are excluded from the result set.
- This type of join is known as an **inner join**, and it is the most commonly used type of join.
- If you do not specify the type of join, an inner join appied by default.

SELECT *name*, *course_id*
FROM *student* **INNER JOIN** *takes*
ON *student.ID = takes.ID*;

SELECT *name*, *course_id*
FROM *student* **INNER JOIN** *takes*
USING *(ID)*;

Join Condition

- The **ON** condition allows a general predicate over the relations being joined

- This predicate is written like a **WHERE** clause predicate except for the use of the keyword **ON**

```
SELECT *
FROM  student INNER JOIN takes ON student.ID  = takes.ID;
```

  - The **ON** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.

- Equivalent to:

```
SELECT *
FROM  student , takes
WHERE  student.ID  = takes.ID;
```

Joining Three or More Tables

- Use Sakila database, find name and city of all customer.

```
SELECT c.first_name, c.last_name, ct.city
FROM customer c
INNER JOIN address a
ON c.address_id = a.address_id
INNER JOIN city ct
ON a.city_id = ct.city_id;
```

```
SELECT c.first_name, c.last_name, ct.city
FROM city ct
INNER JOIN address a
ON a.city_id = ct.city_id
INNER JOIN customer c
ON c.address_id = a.address_id;
```

```
SELECT c.first_name, c.last_name, ct.city
FROM address a
INNER JOIN city ct
ON a.city_id = ct.city_id
INNER JOIN customer c
ON c.address_id = a.address_id;
```

Outer Join

- An extension of the join operation that **avoids loss of information**.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses ***null*** values.
- Three forms of outer join: left outer join, right outer join, and full outer join

Outer Join Examples

Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that
  course information is missing for CS-437
  prereq information is missing for CS-315

Left Outer Join

course **LEFT OUTER JOIN** *prereq*

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

In relational algebra: *course* ⟕ *prereq*

Left Outer Join

```
SELECT f.film_id, f.title, i.inventory_id
FROM film f
INNER JOIN inventory i
ON f.film_id = i.film_id
WHERE f.film_id BETWEEN 13 AND 15;
```

```
SELECT f.film_id, f.title, i.inventory_id
FROM film f
LEFT OUTER JOIN inventory i
ON f.film_id = i.film_id
WHERE f.film_id BETWEEN 13 AND 15;
```

Right Outer Join

course **RIGHT OUTER JOIN** *prereq*

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

In relational algebra: *course* ⟖ *prereq*

Right Outer Join

```
SELECT f.film_id, f.title, i.inventory_id
FROM inventory i
RIGHT OUTER JOIN film f
ON f.film_id = i.film_id
WHERE f.film_id BETWEEN 13 AND 15;
```

```
SELECT f.film_id, f.title, i.inventory_id
FROM film f
LEFT OUTER JOIN inventory i
ON f.film_id = i.film_id
WHERE f.film_id BETWEEN 13 AND 15;
```

Full Outer Join

course **FULL OUTER JOIN** *prereq*

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

In relational algebra: *course* ⟗ *prereq*

Cross Joins

```
SELECT c.name category_name, l.name language_name
FROM category c
CROSS JOIN language l;
```

Natural Join in SQL

- Natural join matches tuples with the same values for all common attributes and retains only one copy of each common column.
- List the names of instructors along with the course ID of the courses that they taught

  SELECT *name*, *course_id*
  FROM *student, takes*
  WHERE *student.ID = takes.ID*;

- Same query in SQL with “natural join” construct

  SELECT *name*, *course_id*
  FROM *student* **NATURAL JOIN** *takes*;

- The **FROM** clause in can have multiple relations combined using natural join:

  SELECT $A_1, A_2, \ldots A_n$
  FROM r_1 **NATURAL JOIN** r_2 **NATURAL JOIN .. NATURAL JOIN** r_n
  WHERE P ;

Dangerous in Natural Join

- Beware of unrelated attributes with same name which get equated incorrectly

- Example: List the names of students' instructors along with the titles of courses that they have taken
  - Correct version

    SELECT *name*, *title*
    FROM *student* **NATURAL JOIN** *takes*, *course*
    WHERE *takes.course_id* = *course.course_id*;
  - Incorrect version

    SELECT *name*, *title*
    FROM *student* **NATURAL JOIN** *takes* **NATURAL JOIN** *course*;
    - This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.
    - The correct version (above), correctly outputs such pairs.

Natural Join with Using Clause

- To avoid the danger of equating attributes erroneously, we can use the "**USING**" construct that allows us to specify exactly which columns should be equated.

  Query example

  SELECT *name*, *title*
  FROM (*student* **NATURAL JOIN** *takes*) **JOIN** *course* **USING** (*course_id*)

Joined Types and Conditions

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition –** defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type –** defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $(A_1, A_2, \ldots, A_n)$

Joined Relations – Examples

course **RIGHT OUTER JOIN** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

course **FULL OUTER JOIN** *prereq* **USING** (*course_id*)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Joined Relations – Examples

course **INNER JOIN** *prereq* **ON** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

course **LEFT OUTER JOIN** *prereq* **ON** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*

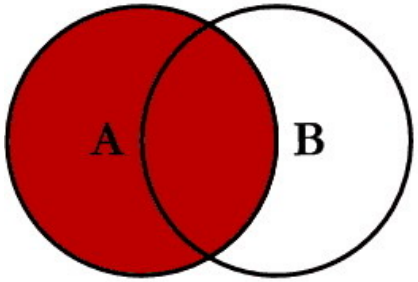
Natural join

- For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught
  - **SELECT** name, course_id **FROM** instructor, teaches **WHERE** instructor.ID= teaches.ID;
  - **SELECT** name, course_id **FROM** instructor **NATURAL JOIN** teaches;

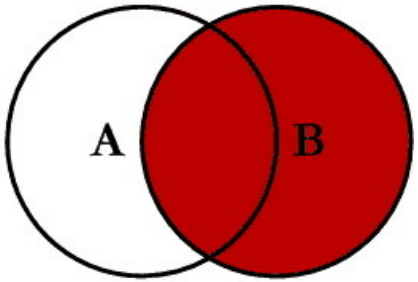
Natural join

- List the names of instructors along with the the titles of courses that they teach.”
  - **SELECT** name, title **FROM** instructor **NATURAL JOIN** teaches, course **WHERE** teaches.course_id= course.course_id;
  - **SELECT** name, title **FROM** instructor **NATURAL JOIN** teaches **NATURAL JOIN** course;
  - **SELECT** name, title **FROM** (instructor **NATURAL JOIN** teaches) **JOIN** course **USING** (course_id);

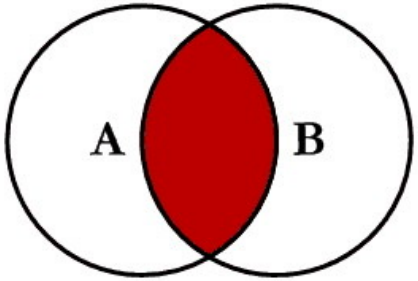
SQL JOINS



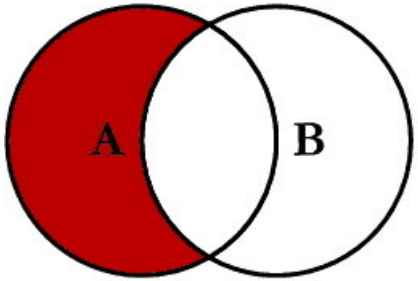

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



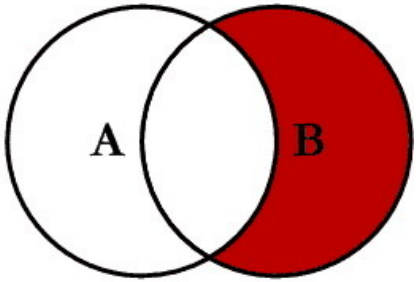

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



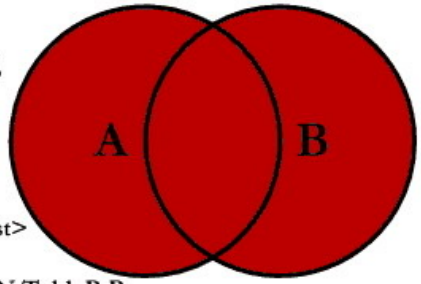

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



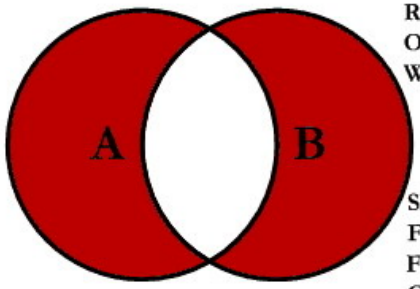

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Homework

- Rewrite these statements without outer join
  - **SELECT** * **FROM** student **NATURAL LEFT OUTER JOIN** takes
  - **SELECT** * **FROM** student **NATURAL RIGHT OUTER JOIN** takes

Homework

- Display a list of all instructors, showing their ID, name, and the number of sections that they have taught.

- Display the list of all course sections offered in Spring 2010, along with the names of the instructors teaching the section. If a section has more than one instructor, it should appear as many times in the result as it has instructors.

- Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries. Make sure to correctly handle departments with no instructors.