Introduction to Databases

Lecture 7

Data Manipulation Language - SELECT

Nga Nguyen

Planning

- NATURAL JOIN
- OUTER JOIN
- Complex queries

NATURAL JOIN

- The join is on the columns of the same name (with the same types)
- The columns of the result table are the union of columns from two tables

```
SELECT * FROM T1 NATURAL JOIN T2
```

Natural join vs equi-join

tab1		tab2	
col1	col12	col1	col22
a	x	a	1
b	y	b	2
c	z	d	3

```
SELECT *
FROM tab1
NATURAL JOIN tab2;
```

a x 1
b y 2

```
SELECT * FROM tab1 t1
JOIN tab2 t2
ON t1.col1 = t2.col1;
```

a x a 1
b y b 2

NATURAL JOIN : be careful !

```
SELECT DISTINCT  name
FROM     cinema  NATURAL JOIN projection
                 NATURAL JOIN film
                 NATURAL JOIN player
                 NATURAL JOIN individual
WHERE            name = 'Travolta';
```

Problem : confusion on columns who have the same name => USING clause but to AVOID !

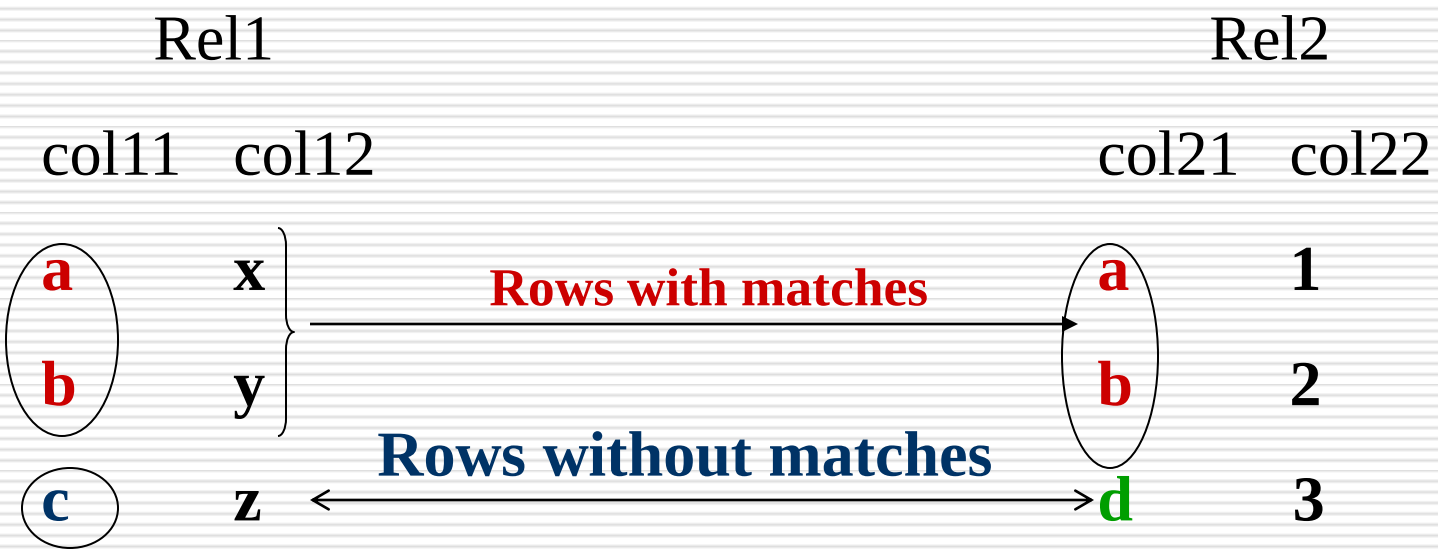
OUTTER JOIN

- The outer join retrieves rows corresponding to join criteria, but **also those for which there are no matches**.
- Different outer joins:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

SQL Syntax

```
SELECT … FROM
    <left table>
LEFT |RIGHT | FULL [OUTER] JOIN
    <right table>
ON
   <join condition>
WHERE … ;
```

OUTER JOIN


Rel1
Rel2
col11
col12
col21
col22
a
b
c
x
y
z
a
b
d
1
2
3
Rows with matches
Rows without matches

LEFT OUTER JOIN

tab1		tab2	
col11	col12	col21	col22
a	x	a	1
b	y	b	2
c	z	d	3

```
SELECT *
FROM tab1 LEFT OUTER JOIN tab2
ON tab1.col11 = tab2.col21;
```

a	x	a	1
b	y	b	2
c	z	NULL	NULL

Implicit LEFT OUTER JOIN

tab1		tab2	
col11	**col12**	**col21**	**col22**
a	x	a	1
b	y	b	2
c	z	d	3

```
SELECT *
FROM tab1,tab2
WHERE tab1.col1 = tab2.col1(+);
```

a	x	**a**	**1**
b	y	**b**	**2**
c	z	NULL	NULL

RIGHT OUTER JOIN

tab1

col11	col12
a	x
b	y
c	z

tab2

col21	col22
a	1
b	2
d	3

```
SELECT *
FROM tab1 RIGHT OUTER JOIN tab2
ON tab1.col11 = tab2.col21;
```

⟹

a	x	a	1
b	y	b	2
NULL	NULL	d	3

Implicit RIGHT OUTER JOIN

tab1		tab2	
col11	col12	col21	col22
a	x	a	1
b	y	b	2
c	z	d	3

```
SELECT *
FROM tab1,tab2
WHERE tab1.col11(+) = tab2.col21;
```

a	x	a	1
b	y	b	2
NULL	NULL	d	3

FULL OUTER JOIN

tab1		tab2	
col11	col12	col21	col22
a	x	a	1
b	y	b	2
c	z	d	3

```
SELECT *
FROM tab1
FULL OUTER JOIN tab2
ON tab1.col11 = tab2.col21;
```

a	x	a	1
b	y	b	2
c	z	NULL	NULL
NULL	NULL	d	3

Example

- A relation describing the businessman and a relation describing their business
- We want to establish the sum of the turnover of all businessman, including those who did nothing.

Businessman	
id	name
1	John
2	Henri
3	Chuck

⋈

Business		
id	turnover	id_businessman
1	100	2
2	350	1
3	50	1
4	200	2

Bm.id	name	B. id	turnover	id_businessman
1	John	2	350	1
1	John	3	50	1
2	Henri	1	100	2
2	Henri	4	200	2
3	Chuck	NULL	NULL	NULL

Bm.id	SUM(ca)
1	400
2	300
3	0

Normal join

```
SELECT bm.id, SUM(turnover)
FROM businessman bm, business b
WHERE bm.id = id_businessman
GROUP BY bm.id;
```

- Do not give the expected result (no lazy businessman)

Solution : OUTER JOIN

```
SELECT bm.id, SUM(turnover)
FROM businessman bm LEFT JOIN business b
ON bm.id = id_businessman
GROUP BY bm.id;
```

OUTER JOIN : WHERE clause

```
SELECT * FROM tab1
LEFT OUTER JOIN tab2
ON tab1.col11 = tab2.col21
AND tab1.col11 <> 'a';
```

is not equivalent to

```
SELECT * FROM tab1
LEFT OUTER JOIN tab2
ON tab1.col11 = tab2.col21
WHERE tab1.col11 <> 'a';
```

OUTER JOIN : WHERE clause

tab1

col11	col12
a	x
b	y
c	z

tab2

col21	col22
a	1
b	2
d	3

```
SELECT * FROM tab1
LEFT OUTER JOIN tab2
ON col11 = col21
AND col11 <> 'a';
```

a	**x**	NULL	NULL
b	**y**	b	2
c	**z**	NULL	NULL

$tab1 \bowtie_{col11=col21 \text{ and } col11 \neq 'a'} tab2$

OUTER JOIN : WHERE clause

tab1		tab2	
col11	**col12**	**col21**	**col22**
a	x	a	1
b	y	b	2
c	z	d	3

b	**y**	b	2
c	**z**	NULL	NULL

```
SELECT * FROM tab1
LEFT OUTER JOIN tab2
ON tab1.col11 = tab2.col21
WHERE tab1.col11 <> 'a' ;
```

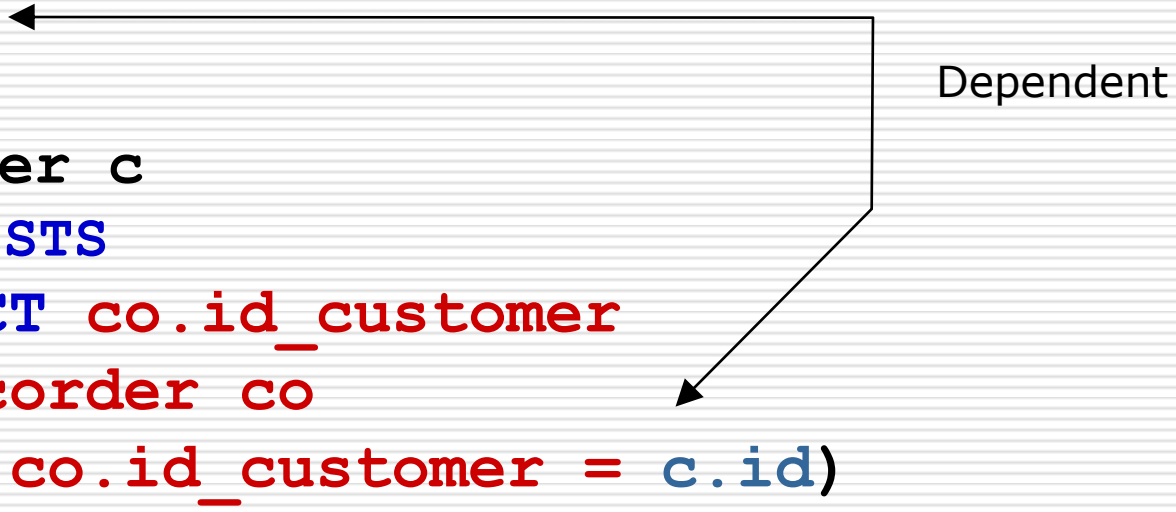
$\sigma_{col11 \neq 'a'}(tab1 \bowtie_{col11=col21} tab2)$

Complex queries

- Divide-and-conquer !
- Connect the sub-results by :
  - Set operations
  - Join
  - Sub-queries

WHERE EXISTS and WHERE NOT EXISTS

```
SELECT
   c.id
FROM
   customer c
WHERE EXISTS
   (SELECT co.id_customer
   FROM corder co
   WHERE co.id_customer = c.id)
```



- The query uses a dependent sub-query
- The predicate is true when the result of the sub-query is not empty

Query reformulation

Individual (num_ind, name, surname)

Film (num_film, num_ind, title, kind, fyear)

Play (num_ind, num_film, role)

- Which actors have played in all the films of Lars von Trier?
- Which actors verify : the number of films made by Lars von Trier in which the actor played is equal to the total number of films made by Lars von Trier.

SQL

```
SELECT i.name, i.surname
FROM   individual i, play p, film f
WHERE i.num_ind = p.num_ind
AND    p.num_film = f.num_film
AND    f.num_film IN ( SELECT num_film
                       FROM   film f, individual i
                       WHERE f.num_ind = i.num_ind
                       AND    i.name = 'von Trier'
                       AND    i.surname = 'Lars')
GROUP BY i.num_ind, i.name, i.surname
HAVING COUNT (DISTINCT f.num_film) = (SELECT COUNT(*)
                  FROM film NATURAL JOIN individual
                  WHERE name = 'von Trier' AND surname = 'Lars');
```