Base de données

Séance 9

Requêtes avec SELECT (suite)

Plan

- Jointure externe
  - à gauche
  - à droite
  - complète
- Requêtes complexes

Jointure à gauche

- Soit une table décrivant des commerciaux, et une autre décrivant des affaires.
- On veut établir le comptage du chiffre d'affaire de tous les commerciaux, y compris ceux qui n'ont rien fait.

Commercial	
id	**nom**
1	**John**
2	**Henri**
3	**Chuck**

Affaire		
id_affaire	**ca**	**id_commercial**
1	**100**	**2**
2	**350**	**1**
3	**50**	**1**
4	**200**	**2**

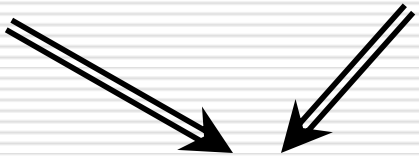
L'echec de la jointure classique

```
SELECT c.id, SUM(ca)
FROM commercial c, affaire a
WHERE id_commercial = c.id
GROUP BY c.id;
```

- Ne donne pas le résultat escompté
- Il manque les commerciaux feignants

Commercial	
id	nom
1	John
2	Henri
3	Chuck

Affaire		
id_affaire	ca	id_commercial
1	100	2
2	350	1
3	50	1
4	200	2



R = Commercial $\bowtie_{id = id\_commercial}$ Affaire

id	nom	id_affaire	ca	id_commercial
1	John	2	350	1
1	John	3	50	1
2	Henri	1	100	2
2	Henri	4	200	2

$_{id}F_{id,SUM(ca)}(R)$

id	SUM(ca)
1	400
2	300

Solution : la jointure gauche

```
SELECT c.id, SUM(ca)
FROM commercial c LEFT JOIN affaire a
ON id_commercial = c.id
GROUP BY c.id;
```

- Ajoute à la jointure interne les commerciaux qui n'ont pas d'affaire

Commercial	
id	nom
1	John
2	Henri
3	Chuck

Affaire		
id_affaire	ca	id_commercial
1	100	2
2	350	1
3	50	1
4	200	2

R = Commercial $\bowtie_{id = id\_commercial}$ Affaire

id	nom	id_affaire	ca	id_commercial
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

$_{id}F_{id,SUM(ca)}(R)$

id	SUM(ca)
1	400
2	300
3	0

La jointure externe : syntaxe SQL

```
SELECT … FROM
    <table gauche>
        LEFT |RIGHT | FULL [OUTER] JOIN
    <table droite>
ON
   <condition de jointure>
WHERE … ;
```

Les jointures externes : LEFT

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

Les jointures externes : LEFT avec pivot

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

Les jointures externes : RIGHT

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

Les jointures externes : RIGHT avec pivot

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

Les jointures externes : FULL

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

Les jointures externes : exemple

Comptez pour chaque producteur le nombre de vins qu'il a récolté, y compris ceux qui n'ont rien produit :

```
SELECT p.id,COUNT(r.id_vin)
FROM producteur p LEFT OUTER JOIN recolte r
ON r.id_producteur = p.id
GROUP BY p.id ;
```

Les jointures externes : exemple

```
SELECT
   p.id,COUNT(r.id_vin)
FROM
   producteur p, recolte r
WHERE
   p.id = r.id_producteur (+)
GROUP BY
   p.id ;
```

Les jointures externes : clauses WHERE

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

N'est pas identique à

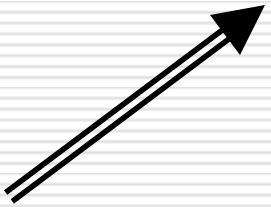
```
SELECT * FROM tab1
LEFT OUTER JOIN tab2
ON tab1.col11 = tab2.col21
WHERE tab1.col11 <> 'a';
```

Les jointures externes : clauses WHERE

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

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

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



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

Les jointures externes : clauses WHERE

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' ;
```

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

Requêtes complexes

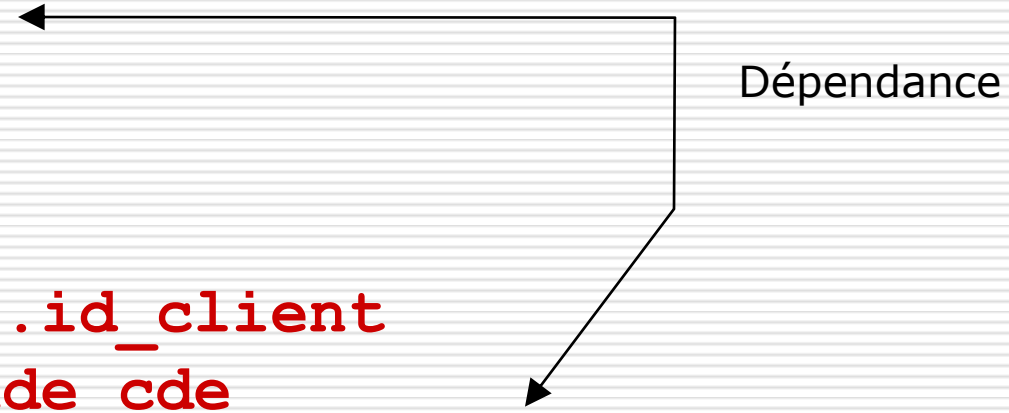
- Décomposer le problème en sous-problèmes simples
- Relier les sous-résultats par :
  - opérateur ensembliste
  - jointure
  - sous-requête
- Projeter les informations utiles

Sous-requêtes dépendantes

- Une sous-requête dépendante utilise une donnée de la requête principale.

WHERE EXISTS et WHERE NOT EXISTS

```
SELECT
   c.id_client
FROM
   client c
WHERE EXISTS
   (SELECT cde.id_client
   FROM commande cde
   WHERE cde.id_client = c.id_client)
```



- Cette requête utilise une **sous-requête dépendante.**
- Le prédicat est vrai quand le résultat de la sous-requête est non vide (contient au moins une ligne).

Division : AR vers SQL

- Quels sont les acteurs qui ont joué dans tous les films de Lars von Trier ?
- En algèbre relationnelle :

  Individu(num_ind, nom, prenom)

  Jouer(#*num_ind, #num_film*, role)

  Film(num_film, #*num_ind*, titre, genre, annee)

Idée :

Résultat = Jouer ÷ (films de Lars von Trier)

Les films de Lars von Trier :

$R1 = \sigma_{nom='von\ Trier'\ and\ prenom='Lars'}(Film \bowtie Individu)$

num_film	num_ind	titre	genre	année	nom	prenom
05	13	Dogville	Drame	2002	von Trier	Lars
04	13	Breaking …	Drame	1996	von Trier	Lars

$R2 = \Pi_{num\_film}(R1)$

$\Pi_{num\_ind,\ num\_film}(Jouer)$

num_ind	num_film
01	05
02	05
03	04
04	04
05	03
06	03
07	03
08	02
09	01
10	01
11	01
04	05
16	07

÷

R2

num_film
05
04

num_ind
01
02
04

∩

num_ind
03
04

R3

num_ind
04

$\Pi_{num\_ind,\ num\_film}(Jouer)$

num_ind	num_film
01	05
02	05
03	04
04	04
05	03
06	03
07	03
08	02
09	01
10	01
11	01
04	05
16	07

÷

R2

num_film
05
04

num_ind
01
02
04

∩

num_ind
03
04

R3

num_ind
04

⋈ Individu

$\Pi_{prenom,nom}$

R4

prenom	nom
Stellan	Skarsgard

Reformulation en SQL

- Quels sont les acteurs qui vérifient : le nombre de films réalisés par Lars von Trier dans lesquels l'acteur a joué est égal au nombre de films réalisés par Lars von Trier.

Traduction SQL

```
SELECT i.nom, i.prenom
FROM   individu i, jouer j, film f
WHERE i.num_ind = j.num_ind
AND    j.num_film = f.num_film
AND    f.num_film IN ( SELECT num_film
                       FROM   film f, individu i
                       WHERE f.num_ind = i.num_ind
                       AND    i.nom = 'von Trier'
                       AND    i.prenom = 'Lars')
GROUP BY i.num_ind,i.nom, i.prenom
HAVING COUNT (DISTINCT f.num_film) = (SELECT COUNT(*)
                  FROM film NATURAL JOIN individu
                  WHERE nom = 'von Trier' AND prenom = 'Lars');
```