Bases de Données

Amélie Gheerbrant

Université Paris Diderot
UFR Informatique
Laboratoire d'Informatique Algorithmique : Fondements et Applications

amelie@liafa.univ-paris-diderot.fr

17 novembre 2014

L'Information incomplète : les valeurs de données nulles

Comment s'en accommode-t-on en pratique ?

Probablement l'aspect de SQL le plus critiqué :

> *.. [this] topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible.*

> *... those SQL features are not fully consistent ; indeed, in some ways they are fundamentally at odds with the way the world behaves.*

> *A recommendation : avoid nulls.*

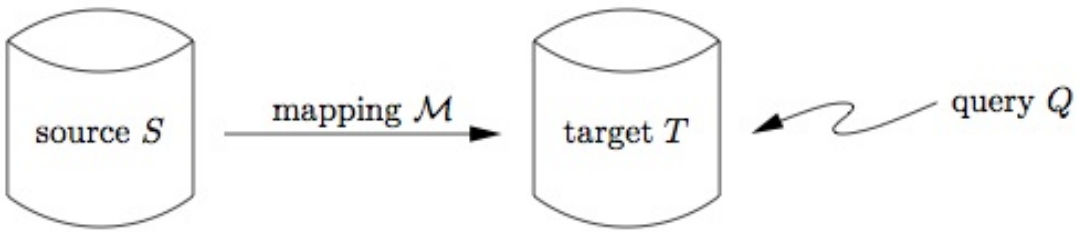
> *Use [nulls] properly and they work for you, but abuse them, and they can ruin everything*

Et pourtant, caractéristique des données sur le Web ("big data") :

> *This data is huge, structured but highly heterogeneous, and includes substantial parts of uncertain or incomplete nature.*

L'information incomplète dans les bases de données Web

"Big data" + interopérabilité = incomplétude massive



Vol :

Départ	Arrivée
Édimbourg	Paris
Paris	Santiago

Voyage :

Départ	Arrivée	Hôtel
Édimbourg	Paris	NULL
Paris	Santiago	NULL

Interopérabilité des données : problème consistant à intégrer des données provenant de sources structurées différemment.

Théorie versus pratique

La théorie de l'information incomplète

- C'est quoi, l'information incomplète ?
- Quelles opérations relationnelles peuvent être évaluées correctement en présence d'information incomplète ?
- Ça veut dire quoi "évalué correctement" ?

L'information incomplète dans SQL

- Les choses sont beaucoup trop simplifiées.
- Mène à des réponses incohérentes.
- Mais parfois les nulls peuvent être très utiles.

C'est quoi une valeur de donnée nulle ?

Parfois il nous manque certaines informations.

Film	titre	réalisateur	acteur
	Dr. Strangelove	Kubrick	Sellers
	Dr. Strangelove	Kubrick	Scott
	null	Polanski	Nicholson
	null	Polanski	Huston
	Star Wars	Lucas	Ford
	Frantic	null	Ford

"null", qu'est-ce que ça pourrait bien vouloir dire ?...
Il y a trois possibilités :

- Information pertinente mais absente : la valeur existe, mais on ne la connait pas pour l'instant.
- Information non pertinente : la valeur n'existe pas (exemple : un film sans acteurs).
- Information inconnue : il n'y a pas d'information.

Représenter des relations avec des nulls : les tables de Codd

- Dans les tables de Codd, on met des variables distinctes pour les valeurs nulles :

Film	titre	réalisateur	acteur
	Dr. Strangelove	Kubrick	Sellers
	Dr. Strangelove	Kubrick	Scott
	x	Polanski	Nicholson
	y	Polanski	Huston
	Star Wars	Lucas	Ford
	Frantic	z	Ford

- La sémantique de la table de Codd appelée Film est l'ensemble POSS(Film) de toutes les tables sans nulls qu'elle représente.
- i.e., on substitue des valeurs aux variables.

Tables dans POSS(Film)

Les tables suivantes sont toutes dans POSS(Film) (qui est l'ensemble des relations qu'elle représente) :

titre	réalisateur	acteur
Dr. Strangelove	Kubrick	Sellers
Dr. Strangelove	Kubrick	Scott
Star Wars	Polanski	Nicholson
Titanic	Polanski	Huston
Star Wars	Lucas	Ford
Frantic	Cameron	Ford

titre	réalisateur	acteur
Dr. Strangelove	Kubrick	Sellers
Dr. Strangelove	Kubrick	Scott
Titanic	Polanski	Nicholson
Titanic	Polanski	Huston
Star Wars	Lucas	Ford
Frantic	Lucas	Ford

titre	réalisateur	acteur
Dr. Strangelove	Kubrick	Sellers
Dr. Strangelove	Kubrick	Scott
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Huston
Star Wars	Lucas	Ford
Frantic	Polanski	Ford

titre	réalisateur	acteur
Dr. Strangelove	Kubrick	Sellers
Dr. Strangelove	Kubrick	Scott
Solaris	Polanski	Nicholson
Stalker	Polanski	Huston
Star Wars	Lucas	Ford
Frantic	Tarkovski	Ford

etc...

Poser des requêtes sur des tables de Codd

- Soit Q une requête de l'algèbre universelle, ou une requête SQL, et T une table de Codd. Comment définir la réponse $Q(T)$ à la requête ?
- On sait seulement appliquer Q à des relations sans nulls. Si on fait ça on trouve :

$$Q_{POSS}(T) = \{Q(R) \mid R \in POSS(T)\}$$

- S'il y avait une table de Codd T' telle que $POSS(T') = Q_{POSS}(T)$, alors on pourrait dire que T' est $Q(T)$, i.e. :

$$POSS(Q(T)) = \{Q(R) \mid R \in POSS(T)\}$$

- Question : est-ce qu'on peut toujours trouver une table T' satisfaisant cette propriété ?

Poser des requêtes sur des tables de Codd

- Soit la requête Q :

```
SELECT * FROM Film
WHERE titre='Frantic';
```

- Soit la table de Codd $Film'$:

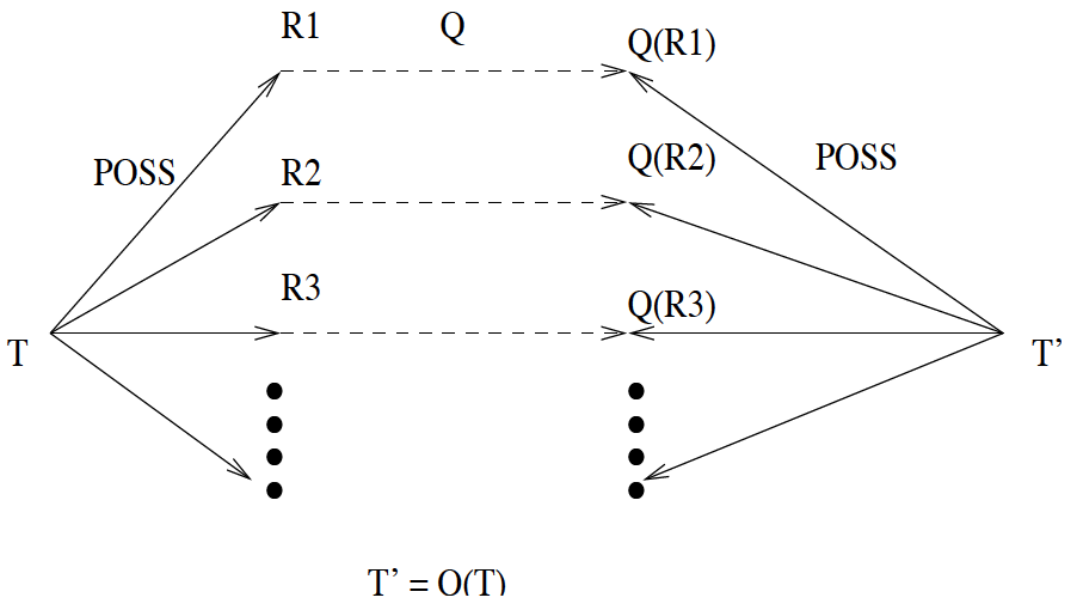
titre	realisateur	acteur
Frantic	z	Ford

- T' est bien telle que $POSS(Film') = Q_{POSS}(Film)$, et donc $Film'$ est $Q(Film)$, i.e. :

$$POSS(Q(Film)) = \{Q(R) \mid R \in POSS(Film)\}$$

Systèmes de représentation

- Façon correcte de répondre aux requêtes sur les tables :



- Question : peut-on toujours trouver $Q(T)$ pour toute table T et requête Q dans un language de requête ?
- **Mauvaise nouvelle** : même pour des requêtes très simples de l'algèbre relationnelle, ce n'est pas possible.

La sélection et les tables de Codd

Soient :

T	A	B
	0	1
	x	2

$Q : \sigma_{A=3}(T)$

Existe-t-il T' telle que $POSS(T') = \{Q(R) \mid R \in POSS(T)\}$?

Soient :

R_1	A	B
	0	1
	2	2

R_2	A	B
	0	1
	3	2

$Q(R1) = \emptyset, Q(R2) = \{(3, 2)\}$, et donc T' ne peut pas exister, car

$$\emptyset \in POSS(T') \text{ si et seulement si } T' = \emptyset$$

Que peut-on faire ?

- Idée 1 : considérer les réponses sûres.
- Un tuple t est dans la réponse sûre à Q sur $T_1, \ldots, T_n$ si $t \in Q(R_1, \ldots, R_n)$ pour tout $R_1 \in POSS(T_1), \ldots, R_n \in POSS(T_n)$
- Idée 2 : étendre les tables de Codd en ajoutant des contraintes sur les valeurs de données nulles (e.g., certaines doivent être égales, certaines doivent ne pas être égales, etc).
- Combiner ces idées permet d'évaluer les requêtes lorsque l'information contenue dans les tables est incomplète.
- Malheureusement les algorithmes d'évaluation, et - pire - les résultats des requêtes, sont souvent très compliqués.

L'information incomplète dans SQL

- L'approche SQL : il y a un seul `NULL` "multi fonction" pour tous les cas d'information manquante/inapplicable
- Les nulls apparaissent comme des entrées dans les tables ; parfois ils sont affichés comme `NULL`, parfois comme "_".
- Leur présence provoque des soucis dès que l'on essaie de faire des comparaisons
- L'union de `SELECT * FROM R WHERE R.A=1;` et `SELECT * FROM R WHERE R.A<>1;` devrait donner la même chose que `SELECT * FROM R;`
- Mais en fait, non.
- Parce que, si `R.A` est `NULL`, alors ni `R.A=1` ni `R.A<>1` n'est vrai.

L'information incomplète dans SQL

- `R.A` a trois valeurs : 1, null, et 2.
- `SELECT * FROM R WHERE R.A=1;` retourne $\frac{\text{A}}{1}$
- `SELECT * FROM R WHERE R.A<>1;` retourne $\frac{\text{A}}{2}$
- Comment tester = `NULL` ? Nouvelle comparaison : `IS NULL` (`IS NOT NULL` est disponible aussi)
- `SELECT * FROM R WHERE R.A IS NULL;` retourne $\frac{\text{A}}{\texttt{NULL}}$
- `SELECT * FROM R;` est l'union de

  ```
  SELECT * FROM R WHERE R.A=1;
  SELECT * FROM R WHERE R.A<>1;
  SELECT * FROM R WHERE R.A IS NULL;
  ```

Les nulls et les autres opérations

- Ca donne quoi, `1+NULL` ? Quelle est la valeur de vérité de `"3=NULL"` ?
- Les nulls ne peuvent pas être utilisés explicitement dans des opérations et des sélections comme `WHERE R.A=NULL` et `SELECT 5-NULL`.
- Pour toute opération arithmétique, sur les chaînes de caractères, etc., si un argument est null, alors le résultat est nul.
- Pour `R.A`={1,`NULL`}, `S.B`={2},

```
SELECT R.A + S.B
FROM R, S;
```

  retourne {3, `NULL`}.
- Quelle est la valeur de `R.A=S.B` ? Si `R.A=1`, `S.B=2`, alors c'est faux. Si `R.A=NULL`, `S.B=2`, la valeur est inconnue (`UNKNOWN`).

La logique des nulls

- Comment la valeur "UNKNOWN" intéragit-elle avec les connecteurs Booléens ? Qu'est ce que "NOT UNKNOWN" ? Qu'est-ce que "UNKNOWN OR TRUE" ?

x	NOT x
vrai	faux
faux	vrai
inconnu	inconnu

ET	vrai	faux	inconnu
vrai	vrai	faux	inconnu
faux	faux	faux	faux
inconnu	inconnu	faux	inconnu

OU	vrai	faux	inconnu
vrai	vrai	vrai	vrai
faux	vrai	faux	inconnu
inconnu	vrai	inconnu	inconnu

- Problème avec les nulls : les gens pensent rarement en termes de logique tri-valuée !

Les nulls et l'agrégation

Supposons que `SELECT * FROM R` retourne

```
A
-----------
          1
          -
```

Alors `SELECT COUNT(*) FROM R` retourne 2.

Mais `SELECT COUNT(R.A) FROM R` retourne 1...

Les nulls et l'agrégation

- On s'attendrait à ce que les nulls soient propagés à travers les opérations arithmétiques
- `SELECT SUM(R.A) FROM R` est la somme

$$a_1 + a_2 + \ldots +_a n$$

de toutes les valeurs dans la colonne A ; si l'une est inconnue, alors le résultat devrait être inconnu aussi.

- Mais `SELECT SUM(R.A) FROM R` retourne 1 si $R.A = \{1, null\}$
- Règle la plus commune pour les fonctions d'agrégation :
  d'abord, ignorer tous les nulls,
  ensuite, calculer la valeur.
- Seule exception : `COUNT(*)`.

Sommes et valeurs null

En présence de valeurs null, la somme n'est pas distributive, i.e., `SUM(A + B)` n'est pas nécessairement égal à `sum(A) + sum(B)`.

ACTIVITE

CODE_ACTIV	INTITULE	TITULAIRE	(H_COURS)	(H_TP)
INFO 1232	Java	PHE	30	15
INFO 1241	Labo prog.	PHE		45
INFO 2101	BD	JLH	30	
INFO 2111	Projet qualité	NHA		30
INFO 2120	Modélisation	JLH	20	10
INFO 2213	Conception	VEN	45	
INFO 2214	Mise en œuvre	VEN	60	
INFO 2231	Labo gestion	NHA		45

```
select TITULAIRE, sum(H_COURS) + sum(H_TP) as CHARGE
from ACTIVITE
group by TITULAIRE;
```

donne

TITULAIRE	CHARGE
JLH	60
NHA	
PHE	90
VEN	

Sommes et valeurs null

En présence de valeurs null, la somme n'est pas distributive, i.e., `SUM(A + B)` n'est pas nécessairement égal à `sum(A) + sum(B)`.

ACTIVITE

CODE_ACTIV	INTITULE	TITULAIRE	(H_COURS)	(H_TP)
INFO 1232	Java	PHE	30	15
INFO 1241	Labo prog.	PHE		45
INFO 2101	BD	JLH	30	
INFO 2111	Projet qualité	NHA		30
INFO 2120	Modélisation	JLH	20	10
INFO 2213	Conception	VEN	45	
INFO 2214	Mise en œuvre	VEN	60	
INFO 2231	Labo gestion	NHA		45

```
select TITULAIRE, sum(H_COURS + H_TP) as CHARGE
from ACTIVITE
group by TITULAIRE;
```

donne

TITULAIRE	CHARGE
JLH	30
NHA	
PHE	45
VEN	

Les nulls dans les sous requêtes

- Supposons :

$$R1.A = \{1, 2\} \qquad R2.A = \{1, 2, 3, 4\}$$

- ```
  SELECT R2.A
  FROM R2
  WHERE R2.A NOT IN (SELECT R1.A
                     FROM R1);
  ```
- Résultat : $\{3, 4\}$
- Maintenant, insérons un null dans $R1$:

$$R1.A = \{1, 2, null\}$$

  et lançons la même requête.
- Le résultat est maintenant $\emptyset$!
  ```

Les nulls dans les sous requêtes

- Bien que ce résultat puisse sembler contrintuitif, il est correct.
- Quelle est la valeur de `3 NOT IN (SELECT R1.A FROM R1)` ?

```
3 NOT IN {1,2,null}
= NOT (3 IN {1,2,null})
= NOT((3 = 1) OR (3=2) OR (3=null))
= NOT(false OR false OR unknown)
= NOT (unknown)
= unknown
```

- De même, l'évaluation de `4 NOT IN {1,2,null}` est unknown, et celle de `1 NOT IN {1,2,null}` et de `2 NOT IN {1,2,null}` est false.
- La requête retourne donc $\emptyset$.

Les nulls dans les sous requêtes

- "En théorie", ça donnerait quoi un résultat d'évaluation correct pour cette requête en présence d'information incomplète ?
- Le résultat de

```
SELECT R2.A
FROM R2
WHERE R2.A NOT IN (SELECT R1.A
                   FROM R1);
```

sur

R1	A
	1
	2
	x

et

R2	A
	1
	2
	3
	4

donnerait quelque chose comme

A	condition
3	$x = 0$
4	$x \neq 0$
3	$y = 0$
4	$y = 0$

- pas implémenté directement en pratique (trop compliqué)

Les nulls peuvent être dangereux !

- Prenons le Système de défense antimissile de l'OTAN et sa base de données recensant des missiles ciblant des villes, ainsi que les missiles lancés pour les intercepter.
- Requête : est-ce qu'il y a un missile ciblant une ville et n'ayant pas encore été intercepté ?

```
SELECT M.numero, M.cible
FROM Missiles M
WHERE M.cible IN (SELECT Nom
                  FROM Ville) AND
      M.numero NOT IN (SELECT I.Missile
                       FROM Intercepte I
                       WHERE I.Statut = 'actif');
```

- Supposons qu'un missile ait été lancé et qu'il faille l'intercepter, mais que sa cible n'ait pas été entrée dans la base de données.

Les nulls peuvent être dangereux !

Missile	numero	cible
	M1	Paris
	M2	Londres
	M3	Berlin

Intercepte	num_int	num_missile	statut
	I1	M1	actif
	I2	NULL	actif

```
SELECT M.numero, M.cible
FROM Missiles M
WHERE M.cible IN (SELECT Nom
                  FROM Ville) AND
      M.numero NOT IN (SELECT I.Missile
                       FROM Intercepte I
                       WHERE I.Statut = 'actif');
```

- La requête retourne l'ensemble vide, car l'évaluation de : `M2 NOT IN {M1, null} AND M3 NOT IN {M1, null}` donne `UNKNOWN`.
- bien que ni M2 ni M3 n'aient été interceptés !
- Situation très improbable ? Oui, oui, probablement... (Heureusement !)

Les nulls dans les sous requêtes : un dernier petit exemple

Supposons que l'on recherche les films pour lesquels il n'y a pas de film plus long.

```
SELECT titre
FROM Film F1
WHERE NOT EXISTS (SELECT *
                  FROM Film F2
                  WHERE F2.duree > F1.duree);
```

En plus des réponses attendues, on obtiendra tous les films dont la durée est inconnue. La valeur de `duree` étant `NULL`, la condition `F2.duree > F1.duree` vaut `UNKNOWN` et l'ensemble entre parenthèses est vide. Le prédicat `NOT EXISTS` est évalué à `TRUE`, et la ligne est retenue !

La propagation des nulls : résumé

Les expressions de calcul

Une expression dont l'évaluation renvoie une valeur numérique, caractères ou temporelle est évaluée à `NULL` si l'un de ses arguments est `NULL` :

- A + B + C vaut `NULL` si A ou B ou C est `NULL` ;
- recette - (SELECT budget from .. WHERE ..) vaut `NULL` si la sous-requête from-where correspond à l'ensemble vide ;
- cast(attribut as varchar(10)) vaut null si attribut = `NULL` ;
- Mr. || nom vaut `NULL` si nom = `NULL` ;
- budget - budget vaut `NULL` si budget = `NULL`.

La propagation des nulls : résumé

Les fonctions agrégatives

Ces fonctions travaillent sur des ensembles dont les éventuelles valeurs null sont ignorées. Si l'ensemble est vide, les fonctions `SUM`, `AVG`, `MIN`, et `MAX` renvoient `NULL`, tandis que `COUNT` renvoie 0.

On notera que cette règle ne suit pas celle des expressions de calcul. En effet, la fonction `SUM` devrait, comme toute somme, renvoyer `NULL` dès que l'un au moins de ses éléments est null, ce qui n'est pas le cas. L'intuition l'emporte ici sur la rigueur.

SQL et les nulls : deux recommendations

Le détachement des colonnes facultatives

Problème : genre souvent non renseigné dans
`Film(titre, annee, realisateur, genre)`

Solution : Décomposer la relation de manière à extraire tout attribut (ou groupe d'attributs) facultatif sous la forme d'une table autonome

```
CREATE TABLE Film (titre .. NOT NULL PRIMARY KEY,
                   annee .. NOT NULL,
                   realisateur .. NOT NULL);
CREATE TABLE GenreFilm(titre .. NOT NULL PRIMARY KEY,
                       genre .. not null,
                       foreign key (titre) reference Film);
```

Ces deux tables ne comportent plus que des colonnes `NOT NULL`.

SQL et les nulls : deux recommendations

Utilisation de valeurs par défaut

La colonne facultative est déclarée `NOT NULL`, mais est accompagnée d'une valeur par défaut. La prise en compte de cette valeur est à charge de l'utilisateur.

```
CREATE TABLE Film (titre .. NOT NULL PRIMARY KEY,
                   annee .. NOT NULL,
                   realisateur .. NOT NULL;
                   genre ..  DEFAULT 'inconnu' NOT NULL);
```

Une dernière règle de prudence : en présence de valeurs null, bien distinguer dans les requêtes SQL les cas sans et avec valeurs null.

Lorsque les nulls sont utiles : les jointures externes

- Exemple :

Studio	nom	titre
	'United'	'Licence to kill'
	'United'	'Rain man'
	'Dreamworks'	'Gladiator'

Film	titre	bénéfice
	'Licence to kill'	156
	'Rain man'	412
	'Fargo'	25

- Requête : pour chaque studio, trouver le bénéfice total généré par ses films

```
SELECT Studio.Nom, SUM(Film.benefice)
FROM Studio NATURAL JOIN Film
GROUP BY Studio.Nom;
```

- Réponse : ('United', 568)
- Mais souvent on voudrait ('Dreamworks', ...) aussi !
- 'Dreamworks' est perdu parce que 'Gladiator' ne correspond à rien dans Film.

Les jointures externes

- Et si on ne veut pas perdre 'Dreamworks' ?
- Alors on utilise des jointures externes :

```
SELECT Studio.nom, SUM(Film.bénéfice)
FROM Studio NATURAL LEFT OUTER JOIN Film
GROUP BY Studio.Nom;
```

- Résultat : ('United', 568), ('Dreamworks', null)
- `Studio NATURAL LEFT OUTER JOIN Film` correspond à :

nom	titre	bénéfice
'United'	'Licence to kill'	156
'United'	'Rain Man'	412
'Dreamworks'	'Gladiator'	null

Les autres jointures externes

- `Studio NATURAL RIGHT OUTER JOIN Film` correspond à :

nom	titre	bénéfice
'United'	'Licence to kill'	156
'United'	'Rain Man'	412
null	'Fargo'	25

- `Studio NATURAL FULL OUTER JOIN Film` correspond à :

nom	titre	bénéfice
'United'	'Licence to kill'	156
'United'	'Rain Man'	412
'Dreamworks'	'Gladiator'	null
null	'Fargo'	25

- Idée : on fait une jointure, mais on garde aussi certains des tuples qui n'ont pas de correspondant, en les complétant avec des nulls.

Les jointures externes

- Avec `R NATURAL LEFT OUTER JOIN S`, on conserve les tuples de R sans correspondant (i.e., de la relation de gauche).
- Avec `R NATURAL RIGHT OUTER JOIN S`, on conserve les tuples de S sans correspondant (i.e., de la relation de droite).
- Avec `R NATURAL FULL OUTER JOIN S`, on conserve les tuples sans correspondant de R et de S (i.e., des deux relations).

Les jointures externes et l'agrégation

- Attention : si on utilise des jointures externes dans les requêtes d'agrégation, on obtient des nulls comme résultat pour tous les agrégats sauf `COUNT`

```
CREATE VIEW V (B1, B2) AS
SELECT Studio.nom, Film.bénéfice
FROM Studio NATURAL LEFT OUTER JOIN Film;

SELECT * FROM V;
```

  Retourne

B1	B2
'Dreamworks'	null
'United'	156
'United'	412

- `SELECT B1, SUM(B2) FROM V GROUP BY B1;` retourne ('Dreamworks', null), ('United', 568).
- `SELECT B1, COUNT(B2) FROM V GROUP BY B1;` retourne ('Dreamworks', 0), ('United', 2).

Les jointures externes

- Il y a des systèmes qui n'aiment pas le mot `NATURAL` et qui ne vous laisseront faire que

```
R NATURAL LEFT/RIGHT/FULL OUTER JOIN S
        ON condition
```

- Exemple :

```
SELECT *
FROM Studio LEFT OUTER JOIN Film ON
Studio.titre=Film.titre;
```

- Résultat :

nom	titre	titre	bénéfice
'United'	'Licence to kill'	'Licence to kill'	156
'United'	'Rain man'	'Rain man'	412
'Dreamworks'	'Gladiator'	null	null