# L'information incomplète dans SQL Bases de données

Amélie Gheerbrant
IRIF, Université Paris Diderot
amelie@irif.fr

### La valeur Null

- Valeur de données Null: marqueur multi-fonction utilisé dès qu'il faut représenter qu'une information est manquante
  - source majeure de problèmes et d'incohérences

### L'information incomplète dans SQL

"... this topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible" "Those SQL features are ... fundamentally at odds with the way the world behaves"

C. Date & H. Darwin, 'A Guide to SQL Standard'

"If you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; all results become suspect. You can never trust the answers you get from a database with nulls"

C. Date, 'Database in Depth'

### Qu'est-ce que ça signifie Null?

- Dépend du contexte :
  - Valeur manquante : il y a une valeur, mais inconnue pour le moment
  - Non applicable : il n'y a pas de valeur (non définie)
- Comportement :
  - Constante : comme n'importe quelle autre valeur
  - Inconnu : une nouvelle valeur de vérité (true, false, unknown)
- « Méta » incomplétude :
  - la signification de NULL dépend de l'usage qui en est fait dans un certain contexte

# Valeur manquante versus non applicable

Personne				
<u>ID</u>	nom	instrument		
1	Lisa	Null		
2	Bart	Null		

## Valeur manquante versus non applicable

Personne			
<u>ID</u>	nom	instrument	
1	Lisa	Null	
2	Bart	Null	

- Est-ce que Lisa joue d'un instrument ?
- Est-ce que Bart joue d'un instrument ?

Personne			
<u>ID</u>	nom	musicien	instrument
1	Lisa	true	Null
2	Bart	false	Null

- Est-ce que Lisa joue d'un instrument ?
- Est-ce que Bart joue d'un instrument ?

Personne					
<u>ID</u>	nom	musicien	instru	ıment	
1	Lisa	true	Null	i.e. incor	nnu
2	Bart	false	Null	i.e. non app	licable

- Est-ce que Lisa joue d'un instrument ?
- Est-ce que Bart joue d'un instrument ?

Personne					
ID	nom	musicien	instru	ıment	
1	Lisa	true	Null	i.e. incor	าทเ
2	Bart	false	Null	i.e. non app	lica

- Et si on voulait exclure la valeur NULL de instrument mais seulement lorsque la valeur de musicien est true ?
  - spécifier instrument NOT NULL n'est pas adapté dans ce cas
  - il faut une contrainte plus fine

Personne					
<u>ID</u>	nom	musicien	instru	ıment	
1	Lisa	true	Null	i.e. incor	nnu
2	Bart	false	Null	i.e. non app	licable

- Pour exclure la valeur NULL de instrument lorsque la valeur de musicien est true, on pourrait utiliser :
  - CHECK (NOT (musicien='true' AND instrument IS NULL))

Personne					
ID	nom	musicien	instru	ıment	
1	Lisa	true	Null	i.e. incor	าทเ
2	Bart	false	Null	i.e. non app	lica

- Mais il serait raisonnable d'exclure aussi les cas dans lesquels musicien est false mais instrument n'est pas NULL:
  - CHECK (NOT (musicien='false' AND instrument IS NOT NULL))

Personne			
ID	nom	musicien	instrument
1	Lisa	NULL	saxophone
2	Bart	NULL	NULL

- D'après les valeurs de la seconde colonne, on ne sait pas si Bart et Lisa sont musiciens
  - NULL dans la colonne musicien représente une valeur manquante
  - mais alors, que représente NULL dans la colonne instrument ?!

Personne			
<u>ID</u>	nom	musicien	instrument
1	Lisa	NULL	saxophone
2	Bart	NULL	NULL

- D'après les valeurs de la seconde colonne, on ne sait pas si Bart et Lisa sont musiciens
  - NULL dans la colonne musicien représente une valeur manquante
  - mais alors, que représente NULL dans la colonne instrument ?!
- Et que représente la valeur saxophone dans instrument ?!
  - Exclure ce genre de cas avec CHECK
    CHECK (NOT (musicien='true' AND instrument IS NULL))
  - Déclarer musicien comme NOT NULL

Règle de bonne pratique : détacher les colonnes facultatives (élimination des valeurs non applicables)

Personne		
<u>ID</u>	nom	
1	Lisa	
2	Bart	

Musicien		
<u>ID</u>	instrument	
1	saxophone	

- + contraintes de clef étrangère et NOT NULL :
  - Musicien(ID) references Personne(ID)
  - instrument NOT NULL

- Les NULL ne sont pas autorisés dans les clefs primaires
  - CREATE TABLE R (A INT PRIMARY KEY);
  - INSERT INTO R VALUES (NULL);
  - ERROR: null value in column "a" violates not-null constraint
- En revanche, les NULL se comportent comme des valeurs distinctes (bien que manquantes...) avec UNIQUE
  - ▶ CREATE TABLE R (A INT UNIQUE, B INT);
  - ▶ INSERT INTO R VALUES (NULL, 1);

R			
A	В		
NULL	1		

- Les NULL ne sont pas autorisés dans les clefs primaires
  - CREATE TABLE R (A INT PRIMARY KEY);
  - ▶ INSERT INTO R VALUES (NULL);
  - ERROR: null value in column "a" violates not-null constraint
- En revanche, les NULL se comportent comme des valeurs distinctes (bien que manquantes...) avec UNIQUE
  - ▶ CREATE TABLE R (A INT UNIQUE, B INT);
  - ▶ INSERT INTO R VALUES (NULL, 1);
  - ▶ INSERT INTO R VALUES (NULL, 1);

R				
A	В			
NULL	1			
NULL	1			

 Rappel: la contrainte de clef primaire impose que le résultat de cette requête soit toujours vide

```
SELECT R1.A FROM R R1, R R2
WHERE R1.A=R2.A AND (R1.B<>R2.B)
UNION
SELECT A FROM R WHERE A IS NULL;
```

 Rappel: la contrainte UNIQUE impose que le résultat de cette requête soit toujours vide

```
SELECT R1.A FROM R R1, R R2
WHERE R1.A=R2.A AND (R1.B<>R2.B);
```

Ici R1.A=R2.A n'est ni true, ni false, mais unknown

R		
A	$\mathbf{B}$	
1	1	

S		
A	В	
1	NULL	

Contrainte de clef étrangère : S(A,B) REFERENCES R(A,B) la contrainte est satisfaite

• Rappel: la contrainte de clef étrangère impose que le résultat de cette requête soit toujours vide

```
SELECT A, B FROM S
WHERE A NOT IN (SELECT A FROM R)
OR
B NOT IN (SELECT B FROM R);
```

• Or ici NULL NOT IN (SELECT B FROM R) n'est ni true ni false, mais unknown

```
CREATE TABLE article (
    id integer primary key,
    nom varchar(25) not null,
    prix decimal(7,2),
    prix_solde decimal(7,2),
CHECK (prix_solde < prix)
);</pre>
```

Article					
<u>ID</u>	nom	prix	prix_solde		
8	pantalon	99.9	59.95		
5	pull	40	26		
3	veste	120	79.95		
9	chemise	20	NULL		

 <u>Rappel</u>: la contrainte CHECK impose que le résultat de cette requête soit toujours vide

```
SELECT * FROM Article
WHERE prix_solde >= prix;
```

- La dernière ligne est admise par la clause CHECK
- En revanche SELECT \* FROM Article WHERE prix\_solde < prix ; sélectionne toutes les lignes sauf la dernière

# Rappel: maintien de l'intégrité référentielle

### Prévoir les problèmes en amont :

```
CREATE TABLE Seance

(id_seance serial PRIMARY KEY)

titre VARCHAR(30), FOREIGN KEY REFERENCES Film(titre)

cinema VARCHAR(30)

ON DELETE SET NULL

ON UPDATE CASCADE

);
```

#### Cette déclaration force :

- à répercuter l'effacement d'un tuple dans Film en rendant NULL tous les champs titre de Séance qui avaient pour valeur celle du titre effacé
- à répercuter la mise à jour d'un tuple dans Film (pour ce qui concerne le champ titre) en opérant la même mise à jour dans les tuples de Seance concernés

# Rappel: maintien de l'intégrité référentielle

### Prévoir les problèmes en amont :

```
CREATE TABLE Seance

(id_seance serial PRIMARY KEY)

titre VARCHAR(30), FOREIGN KEY REFERENCES Film(titre)

cinema VARCHAR(30)

ON DELETE SET NULL

ON UPDATE CASCADE

);
```

#### Cette déclaration force :

• à répercuter l'effacement d'un tuple dans Film en rendant NULL tous les champs titre de Séance qui avaient pour valeur celle du titre effacé

pas de problème

à répercuter la mise à jour d'un tuple dans Film (pour ce qui concerne le champ titre) en opérant la même mise à jour dans les tuples de Seance concernés

### Valeur NULL et opérations arithmétiques

SELECT 1+NULL AS somme, 1-NULL AS diff, 1\*NULL AS mult, 1/NULL AS div;

somme	diff	mult	div
NULL	NULL	NULL	NULL

Toute opération arithmétique mettant en jeu un NULL a pour résultat NULL

### Valeur NULL et opérations arithmétiques

SELECT 1+NULL AS somme, 1-NULL AS diff, 1\*NULL AS mult, 1/NULL AS div, NULL/O AS div\_zero;

somme	diff	mult	div	div_zero
NULL	NULL	NULL	NULL	NULL

- En fait, même le résultat de la division de NULL par 0 retourne NULL!
  - pas d'erreur liée à la division par 0 ici
  - NULL est traité comme non applicable (indéfini)

### Valeur NULL et agrégation

R

A

0

NULL

1

NULL

Les opérateurs d'agrégation ignorent NULL

SELECT min(A) as min, max(A) as max, count(A) as count, sum(A) as somme, cast(avg(A) as numeric(2,1)) FROM R;

min	max	count	somme	moyenne
0	1	2	1	0.5

### Valeur NULL et agrégation

R

A

O

NULL

1

NULL

Les opérateurs d'agrégation ignorent NULL

une exception : COUNT(\*)

SELECT min(A) as min, max(A) as max, count(A) as count, sum(A) as somme, cast(avg(A) as numeric(2,1)), COUNT(\*) as count\_star FROM R;

min	max	count	somme	moyenne	count_star
0	1	2	1	0.5	4

R

## Valeur NULL et agrégation

A

0

NULL

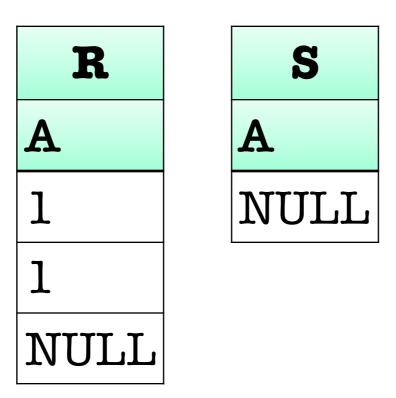
1

Le calcul d'un agrégat sur un multi-ensemble vide retourne NULL

SELECT min(A) as min, max(A) as max, count(A) as count, sum(A) as somme, avg(A) as moyenne FROM R WHERE A=2;

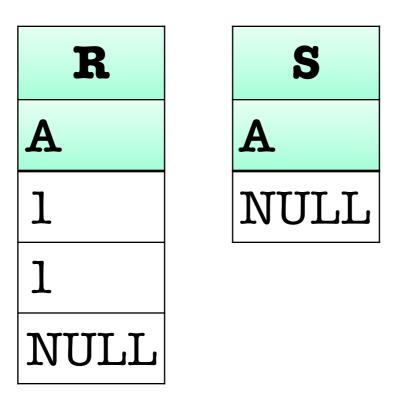
min	max	count	somme	moyenne
NULL	NULL	0	NULL	NULL

Sémantique de ces NULL : valeurs non applicables

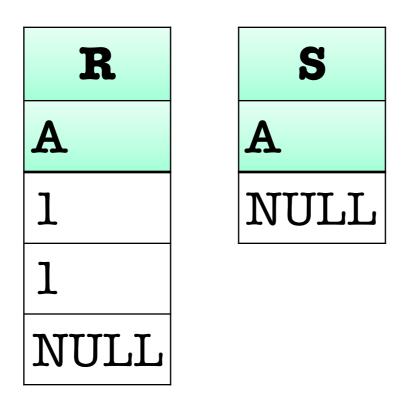


#### Que retournent :

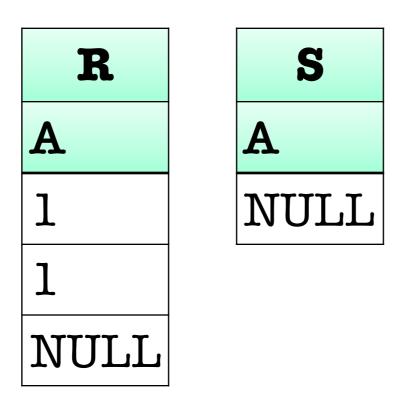
- Q1:SELECT \* FROM R UNION SELECT \* FROM S;
   Q2:SELECT \* FROM R INTERSECT SELECT \* FROM S;
- ▶ Q3: SELECT \* FROM R EXCEPT SELECT \* FROM S;



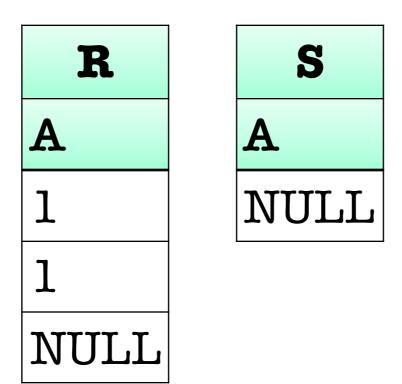
- Que retournent :
  - ▶ Q1:SELECT \* FROM R UNION SELECT \* FROM S;
  - ▶ Q2 : SELECT \* FROM R INTERSECT SELECT \* FROM S;
  - ▶ Q3 : SELECT \* FROM R EXCEPT SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL}



- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION SELECT \* FROM S;
  - ▶ Q2:SELECT \* FROM R INTERSECT SELECT \* FROM S;
  - ▶ Q3 : SELECT \* FROM R EXCEPT SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL}
  - Q2 retourne {NULL}

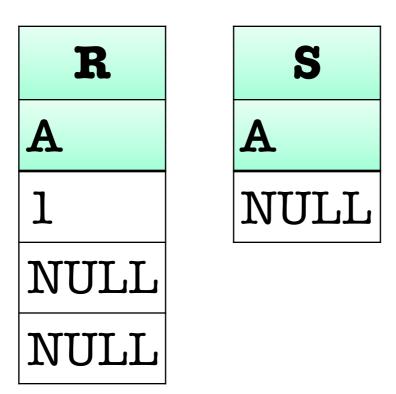


- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION SELECT \* FROM S;
  - ▶ Q2:SELECT \* FROM R INTERSECT SELECT \* FROM S;
  - ▶ Q3: SELECT \* FROM R EXCEPT SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL}
  - Q2 retourne {NULL}
  - Q3 retourne {1}



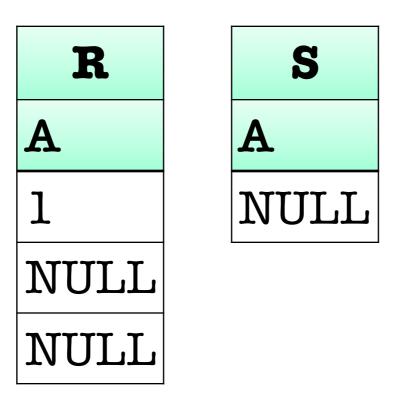
- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION SELECT \* FROM S;
  - ▶ Q2 : SELECT \* FROM R INTERSECT SELECT \* FROM S;
  - ▶ Q3: SELECT \* FROM R EXCEPT SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL}
  - Q2 retourne {NULL}
  - Q3 retourne {1}

Dans les opérations ensemblistes, les NULL sont donc traités comme n'importe quelle autre valeur

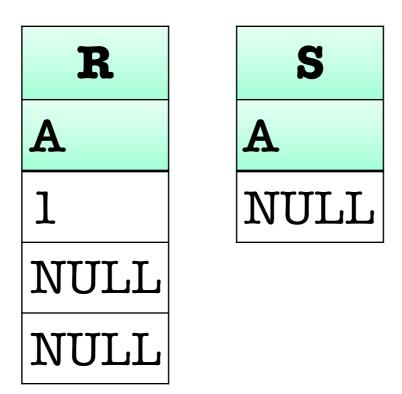


#### Que retournent :

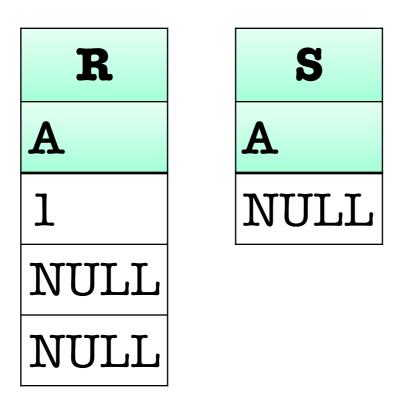
- Q1: SELECT \* FROM R UNION ALL SELECT \* FROM S;
   Q2: SELECT \* FROM R INTERSECT ALL SELECT \* FROM S;
- ▶ Q3: SELECT \* FROM R EXCEPT ALL SELECT \* FROM S;



- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION ALL SELECT \* FROM S;
  - ▶ Q2:SELECT \* FROM R INTERSECT ALL SELECT \* FROM S;
  - ▶ Q3:SELECT \* FROM R EXCEPT ALL SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL, NULL, NULL}



- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION ALL SELECT \* FROM S;
  - ▶ Q2:SELECT \* FROM R INTERSECT ALL SELECT \* FROM S;
  - ▶ Q3 : SELECT \* FROM R EXCEPT ALL SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL, NULL, NULL}
  - Q2 retourne {NULL}



- Que retournent :
  - ▶ Q1 : SELECT \* FROM R UNION ALL SELECT \* FROM S;
  - ▶ Q2:SELECT \* FROM R INTERSECT ALL SELECT \* FROM S;
  - ▶ Q3 : SELECT \* FROM R EXCEPT ALL SELECT \* FROM S;
- Réponse :
  - Q1 retourne {1, NULL, NULL, NULL}
  - Q2 retourne {NULL}
  - Q3 retourne {1, NULL}

### Valeur NULL dans les conditions de sélection

RSAA1NULLNULL

- Que retournent :
  - ▶ Q1 : SELECT \* FROM R, S WHERE R.A=S.A;
  - ▶ Q2 : SELECT \* FROM R, S WHERE R.A<>S.A;
  - ▶ Q3 : SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A;

R		S		R.A	S.A
A	×	A	=	1	NULL
1		NULL		NULL	NULL
NULL			36		

#### Valeur NULL dans les conditions de sélection

A A A NULL NULL

Le résultat de chacune de ces trois requêtes est vide

- Que retournent :
  - ▶ Q1 : SELECT \* FROM R, S WHERE R.A=S.A;
  - ▶ Q2 : SELECT \* FROM R, S WHERE R.A<>S.A;
  - ▶ Q3 : SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A;

R		S			R.A	S.A
A	×	A	=		1	NULL
1		NULL			NULL	NULL
NULL			I	37		

- Que retourne :
  - ▶ SELECT 1=NULL AS resultat;

- Que retourne :
  - ▶ SELECT 1=NULL AS resultat;

# résultat NULL

- Il ne s'agit pas d'un résultat indéterminé, mais d'une valeur de vérité :
  - UNKNOWN

• SELECT 1=NULL AS resultat;



- Il ne s'agit pas d'un résultat indéterminé, mais d'une valeur de vérité :
  - UNKNOWN
- SELECT 1=NULL OR TRUE AS resultat;

**résultat** true

• SELECT 1=NULL OR TRUE AS resultat;

**résultat** true

- En revanche SELECT NULL/1 OR TRUE AS resultat; renvoie:
  - ERROR: argument of OR must be type boolean, not type integer

#### Valeur NULL et évaluation des conditions de sélection

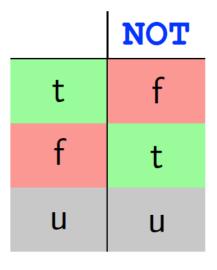
- SQL utilise trois valeurs de vérité : true (t), false (f), unknown (u)
  - toute comparaison (à l'exception de IS [NOT] NULL et EXISTS) dont l'un des arguments est NULL est évalué unknown

#### Valeur NULL et évaluation des conditions de sélection

- SQL utilise trois valeurs de vérité : true (t), false (f), unknown (u)
  - toute comparaison (à l'exception de IS [NOT] NULL et EXISTS) dont l'un des arguments est NULL est évalué unknown
  - les valeurs de vérité assignées à chaque comparaison se propagent via les tables de vérité suivantes :

AND	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

OR	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u



#### Valeur NULL et évaluation des conditions de sélection

- SQL utilise trois valeurs de vérité : true (t), false (f), unknown (u)
  - toute comparaison (à l'exception de IS [NOT] NULL et EXISTS) dont l'un des arguments est NULL est évalué unknown
  - les valeurs de vérité assignées à chaque comparaison se propagent via les tables de vérité suivantes :

AND	t	f	u	OR	t	f	u		NOT
t	t	f	u	t	t	t	t	t	f
f	f	f	f	f	t	f	u	f	t
u	u	f	u	u	t	u	u	u	u

les tuples retournés sont ceux pour lesquels la condition est évaluée true

R S A A I NULL NULL

- Que retournent :
  - ▶ Q1 : SELECT \* FROM R, S WHERE R.A=S.A;
  - ▶ Q2 : SELECT \* FROM R, S WHERE R.A<>S.A;
  - ▶ Q3 : SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A;

R		S		R.A	S.A
A	×	A	=	1	NULL
1		NULL		NULL	NULL
NULL			45	•	•

A

1

NULL

S A NULL

#### Que retournent :

Q1 : SELECT \* FROM R, S WHERE R.A=S.A;

▶ Q2:SELECT \* FROM R, S WHERE R.A<>S.A;

▶ Q3 : SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A ;

R.A	S.A
1	NULL
NULL	NULL

R.A=S.A
unknown
unknown

R

A

1

NULL

S

A

NULL

#### Que retournent :

▶ Q1 : SELECT \* FROM R, S WHERE R.A=S.A;

▶ Q2:SELECT \* FROM R, S WHERE R.A<>S.A;

Q3: SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A;

R.A	S.A
1	NULL
NULL	NULL

R.A=S.A

unknown

unknown

R.A<>S.A
unknown
unknown

R

A

1

NULL

S

A

NULL

• (,	ue	retol	urnent	•

▶ Q1 : SELECT \* FROM R, S WHERE R.A=S.A;

▶ Q2 : SELECT \* FROM R, S WHERE R.A<>S.A;

R.A	S.A
1	NULL
NULL	NULL

Q3: SELECT \* FROM R, S WHERE R.A=S.A OR R.A<>S.A;

R.A=S.A

unknown

unknown

R.A<>S.A

unknown

unknown

R.A=S.A OR R.A <> S.A

unknown

unknown

Sur une base de données sans NULL, Q1 et Q2 retournent le même résultat

```
Q1: SELECT R.A FROM R
INTERSECT
SELECT S.A FROM S;
Q2: SELECT DISTINCT R.A
FROM R, S
WHERE R.A=S.A;
```

Sur une base de données sans NULL, Q1 et Q2 retournent le même résultat

```
    Q1: SELECT R.A FROM R
        INTERSECT
        SELECT S.A FROM S;
    Q2: SELECT DISTINCT R.A
        FROM R, S
        WHERE R.A=S.A;
```

En revanche en présence de NULL, pas forcément...

Sur une base de données sans NULL, Q1 et Q2 retournent le même résultat

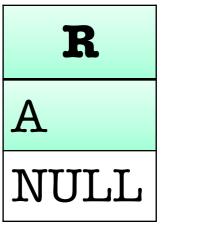
```
Q1: SELECT R.A FROM R
INTERSECT
SELECT S.A FROM S;
```

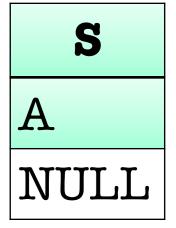
PROM R, S

WHERE R.A=S.A;

En revanche en présence de NULL, pas forcément...

- Q1 retourne {NULL}
- Q2 retourne la table vide





Sur une base de données sans NULL, Q1 et Q2 retournent le même résultat

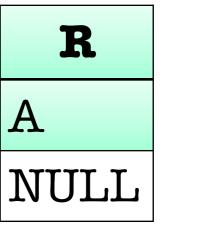
```
Q1: SELECT R.A FROM R
INTERSECT
SELECT S.A FROM S;
```

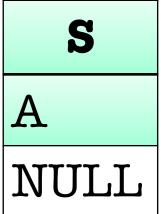
PROM R, S

WHERE R.A=S.A;

En revanche en présence de NULL, pas forcément...

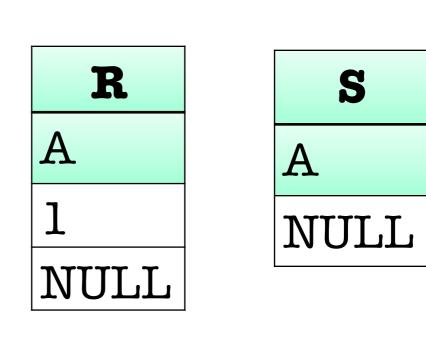
- Q1 retourne {NULL}
- Q2 retourne la table vide





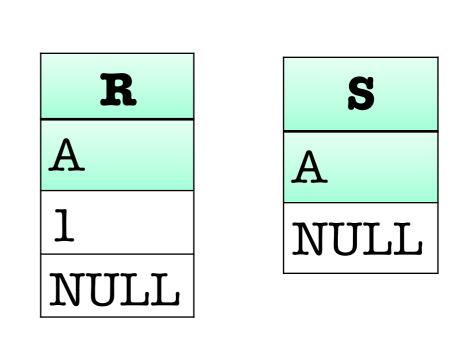
Nous avons vu trois manières (a priori équivalentes) d'exprimer la négation

```
▶ Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE NOT EXISTS (
     SELECT *
     FROM S
     WHERE S.A=R.A);
 Q3: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A
     FROM S);
```



Nous avons vu trois manières (a priori équivalentes) d'exprimer la négation

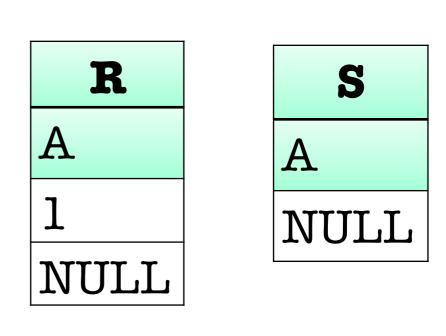
```
▶ Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE NOT EXISTS (
     SELECT *
     FROM S
     WHERE S.A=R.A);
 Q3: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A.
     FROM S);
```



QI retourne {I}

Nous avons vu trois manières (a priori équivalentes) d'exprimer la négation

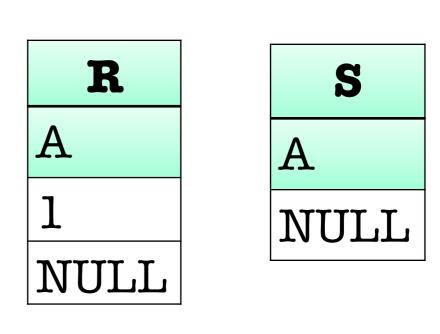
```
▶ Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE NOT EXISTS (
     SELECT *
     FROM S
     WHERE S.A=R.A);
 Q3: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A.
     FROM S);
```



- QI retourne {I}
- Q2 retourne {I, NULL}

Nous avons vu trois manières (a priori équivalentes) d'exprimer la négation

```
▶ Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A
     FROM R
     WHERE NOT EXISTS (
     SELECT *
     FROM S
     WHERE S.A=R.A);
 Q3: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A.
     FROM S);
```



- QI retourne {I}
- Q2 retourne {I, NULL}
- Q3 retourne {}

Films						
titre	réalisateur	durée				
Les Créatures	Varda	92				
Cléo de 5 à 7	Varda	90				
Sans toit ni loi	Varda	NULL				

```
P Q1 : SELECT F1.titre FROM Films F1
WHERE NOT EXISTS (
SELECT * FROM Films F2
WHERE F2.duree > F1.duree);
FROM Films);
P Q2 : SELECT titre FROM Films
WHERE duree = (
P Q1 retourne {Les Créatures, Sans toit ni loi}
SELECT MAX(duree)
FROM Films);
```

Films						
titre	réalisateur	durée				
Les Créatures	Varda	92				
Cléo de 5 à 7	Varda	90				
Sans toit ni loi	Varda	NULL				

Films			
titre	réalisateur	durée	
Les Créatures	Varda	92	
Cléo de 5 à 7	Varda	90	
Sans toit ni loi	Varda	NULL	

Films			
titre	réalisateur	durée	
Les Créatures	Varda	92	
Cléo de 5 à 7	Varda	90	
Sans toit ni loi	Varda	NULL	

Q1: SELECT Fl.titre FROM Films Fl WHERE NOT EXISTS ( SELECT \* FROM Films F2 WHERE F2.duree > Fl.duree);

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

Films			
titre	réalisateur	durée	
Les Créatures	Varda	92	
Cléo de 5 à 7	Varda	90	
Sans toit ni loi	Varda	NULL	

P Q3 : SELECT titre FROM Films
WHERE duree >= ALL (
SELECT duree
FROM Films);

Aucun film ne sera retourné. La valeur de durée étant NULL, la condition durée > NULL vaut UNKNOWN quelque soit la valeur de duree et la condition >=ALL est évalué à false quel que soit le tuple considéré.

Films			
titre	réalisateur	durée	
Les Créatures	Varda	92	
Cléo de 5 à 7	Varda	90	
Sans toit ni loi	Varda	NULL	

- En présence de NULL, la somme ne semble pas toujours distributive...
  - ► 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

- En présence de NULL, la somme ne semble pas toujours distributive...
  - > 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

▶ Q1 : SELECT TITULAIRE, SUM(H\_COURS)+SUM(H\_TP) AS CHARGE

FROM ACTIVITE	TITULAIRE	CHARGE
GROUP BY TITULAIRE;	JLH	60
	NHA	
	PHE	90

VEN

- En présence de NULL, la somme ne semble pas toujours distributive...
  - > 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

▶ Q1 : SELECT TITULAIRE, SUM(H\_COURS+H\_TP) AS CHARGE

FROM ACTIVITE
GROUP BY TITULAIRE;

CHARGE
30
45

### Valeur NULL et résultats de requêtes problématiques

- Imaginons que la BD du système de défense antimissile de l'OTAN recense des missiles ciblant des villes à protéger, ainsi que le lancement de missiles anti balistiques envoyés pour les intercepter
  - Requête : y a-t-il un missile qui cible une ville à protéger mais n'a pas encore été intercepté ?

Missiles		
id_m	cible	
Ml	Paris	
M2	Londres	
M3	Berlin	

```
SELECT M.id_m, M.cible
FROM Missiles M
WHERE M.id_m NOT IN

(SELECT I.missile FROM Intercepte I
WHERE statut = 'actif');
```

Intercepte			
id_i missile statu			
Il	Ml	actif	
12	NULL	actif	

# Valeur NULL et résultats de requêtes problématiques

- Imaginons que la BD du système de défense antimissile de l'OTAN recense des missiles ciblant des villes à protéger, ainsi que le lancement de missiles anti balistiques envoyés pour les intercepter
  - Requête : y a-t-il un missile qui cible une ville à protéger mais n'a pas encore été intercepté ?

Missiles		
id_m	cible	
Ml	Paris	
M2	Londres	
M3	Berlin	

SELECT M.id\_m, M.cible

FROM Missiles M

WHERE M.id\_m NOT IN

(SELECT I.missile FROM Intercepte I

WHERE statut = 'actif');

Un missile anti balistique a été envoyé, mais sa cible n'a pas été entrée dans la BD

Intercepte		
id_i	missile	statut
Il	Ml	actif
12	NULL	actif

### Valeur NULL et résultats de requêtes problématiques

- La requête retourne l'ensemble vide car l'évaluation de M2 NOT IN {M1, NULL} et M3 NOT IN {M1, NULL} donne UNKNOWN
- Mais ni M2 ni M3 n'ont été interceptés...

Requête : y a-t-il un missile qui cible une ville à protéger mais n'a pas encore

SELECT M.id\_m, M.cible

été intercepté?

Missiles		
id_m cible		
Ml	Paris	
M2	Londres	
M3	Berlin	

Un missile anti balistique a été envoyé, mais sa cible n'a pas été entrée dans la BD

FROM Missiles M
WHERE M.id\_m NOT IN

(SELECT I.missile FROM Intercepte I
WHERE statut = 'actif');

Intercepte		
id_i	missile	statut
Il	Ml	actif
12	NULL	actif

Les choses semblent vraiment absurdes...

Même le résultat de la requête suivante (apparemment tautologique) est suspect :

SELECT missile

FROM Intercepte

WHERE missile='M1' OR missile <>'M1';

Intercepte		
id_i	missile	statut
I1 (	M1)	actif
12	NULL	actif

En fait, SELECT missile n'est pas l'union de FROM Intercepte;

SELECT missile SELECT missile

FROM Intercepte et FROM Intercepte

WHERE missile='M1'; WHERE missile<>'M1';

Intercepte		
id_i	missile	statut
II (	M1)	actif
12	NULL	actif

70

En présence de NULL il est crucial de toujours bien distinguer dans les requêtes les cas avec et sans NULL :

SELECT missile

FROM Intercepte

WHERE missile='M1' OR missile <>'M1' OR missile IS NULL;

Intercepte		
id_i	missile	statut
Il /	Ml	actif
12	NULL	actif

En présence de NULL il est crucial de toujours bien distinguer dans les requêtes les cas avec et sans NULL :

SELECT missile

FROM Intercepte

WHERE missile='M1' OR missile <>'M1' OR missile IS NULL;

Intercepte		
id_i	missile	statut
Il /	M1	actif
12	NULL	actif

En présence de NULL il est crucial de toujours bien distinguer dans les requêtes les cas avec et sans NULL :

SELECT missile

FROM Intercepte

WHERE missile='M1' OR missile <>'M1' OR missile IS NULL;

Intercepte			
id_i missile statut			
Il /	M1	actif	
12	NULL	actif	

<u>ATTENTION</u>: ne jamais utiliser = NULL à la place de IS NULL (la comparaison n'évalue jamais à true)

- Imaginons que la BD du système de défense antimissile de l'OTAN recense des missiles ciblant des villes à protéger, ainsi que le lancement de missiles anti balistiques envoyés pour les intercepter
  - Requête : y a-t-il un missile qui cible une ville à protéger mais n'a pas encore été intercepté ?

Missiles		
id_m	cible	
M1	Paris	
M2	Londres	
M3	Berlin	

```
SELECT M.id_m, M.cible
FROM Missiles M
WHERE M.id_m NOT IN

(SELECT I.missile FROM Intercepte I
WHERE statut = 'actif'
AND I.missile IS NOT NULL);
```

Intercepte			
id_i missile statu			
Il	Ml	actif	
12 NULL actif			

Trois manières équivalentes de retourner les films les plus longs

```
    Q1: SELECT F1.titre FROM Films F1
        WHERE F1.duree IS NOT NULL
        AND NOT EXISTS (
        SELECT * FROM Films F2
        WHERE F2.duree > F1.duree );
    Q2: SELECT titre FROM Films
        WHERE duree = (
        SELECT MAX(duree)
```

FROM Films);

Q3:SELECT titre FROM Films

WHERE duree >=ALL(

SELECT duree

FROM Films

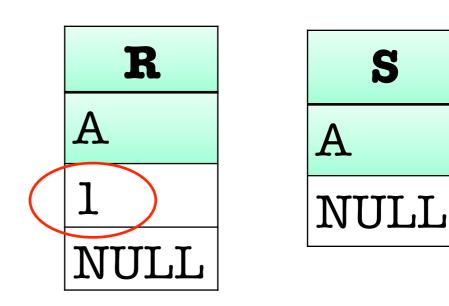
WHERE duree IS NOT NULL);

	Films				
	titre	réalisateur	durée		
(	Les Créatures	Varda	92		
	Cléo de 5 à 7	Varda	90		
	Sans toit ni loi	Varda	NULL		

Trois manières équivalentes d'exprimer la négation

```
Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A IS NOT NULL
     AND NOT EXISTS (
     SELECT *
     FROM S WHERE S.A=R.A);
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A FROM S
```

WHERE S.A IS NOT NULL);

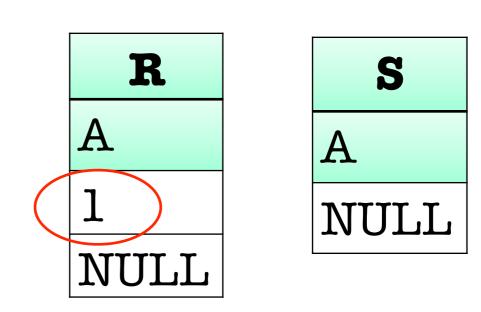


C.f.: « Making SQL queries Correct on Incomplete Databases : a Feasibility Study » PODS'16 (Guagliardo, Libkin)

Trois manières équivalentes d'exprimer la négation

```
Q1 : SELECT R.A FROM R
     EXCEPT
     SELECT S.A FROM S;
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A IS NOT NULL
     AND NOT EXISTS (
     SELECT *
     FROM S WHERE S.A=R.A);
 Q2: SELECT DISTINCT R.A.
     FROM R
     WHERE R.A NOT IN (
     SELECT S.A FROM S
```

WHERE S.A IS NOT NULL);



C.f.: « Making SQL queries Correct on Incomplete Databases : a Feasibility Study » PODS'16 (Guagliardo, Libkin)

• Cas non traité dans PODS'16:

SELECT TITULAIRE, SUM(CHARGE) AS CHARGE

FROM (SELECT TITULAIRE, SUM(H\_COURS) AS CHARGE

FROM ACTIVITE	TITULAIRE	CHARGE
GROUP BY TITULAIRE	JLH	60
GIOUF DI III CLAIIUL	NHA	75
UNION ALL	PHE	90
SELECT TITULAIRE, SUM(H TP) AS CHARGE	VEN	105

FROM ACTIVITE

GROUP BY TITULAIRE) AS HEURES GROUP BY TITULAIRE;

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

# Les expressions de calcul

- Une expression de type numérique, chaîne de caractères ou encore temporel est évaluée à NULL si l'un de ses arguments est NULL
  - A+B+C vaut NULL si A ou B ou C vaut 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 vaut NULL
  - Mr | nom vaut NULL si nom vaut NULL
  - budget budget vaut NULL si budget vaut NULL

# La fonction PostgreSQL coalesce

• La fonction coaelesce accepte un nombre arbitraire d'arguments et retourne le premier de ses arguments non NULL

```
CREATE TABLE article (
    id integer primary key,
    nom varchar(25) not null,
    prix decimal(7,2),
    reduction decimal(7,2),
CHECK (reduction < prix)
);</pre>
```

Article			
<u>ID</u>	nom	prix	reduction
8	pantalon	99.9	59.95
5	pull	40	26
3	veste	120	79.95
9	chemise	20	NULL

```
SELECT nom, (prix-reduction) as prix_net FROM Article;
```

retourne {(pantalon, 39.95), (pull, 14), (veste, 40.05)}

# La fonction PostgreSQL coalesce

 La fonction coaelesce accepte un nombre arbitraire d'arguments et retourne le premier de ses arguments non NULL

```
CREATE TABLE article (
    id integer primary key,
    nom varchar(25) not null,
    prix decimal(7,2),
    reduction decimal(7,2),
CHECK (reduction < prix)
);</pre>
```

Article			
<u>ID</u>	nom	prix	reduction
8	pantalon	99.9	59.95
5	pull	40	26
3	veste	120	79.95
9	chemise	20	NULL

```
SELECT nom, (prix-COALESCE(reduction,0)) as prix_net FROM Article;
```

retourne {(pantalon, 39.95), (pull, 14), (veste, 40.05), (chemise, 20)}

## Les expressions avec case

 L'expression case accepte un nombre arbitraire d'arguments et retourne le premier de ses arguments non NULL

```
CREATE TABLE article (
    id integer primary key,
    nom varchar(25) not null,
    prix decimal(7,2),
    reduction decimal(7,2),

CHECK (reduction < prix)
);
```

Article			
<u>ID</u>	nom	prix	reduction
8	pantalon	99.9	59.95
5	pull	40	26
3	veste	120	79.95
9	chemise	20	NULL

```
SELECT nom, (prix - CASE WHEN reduction IS NULL THEN O
ELSE reduction
END) AS prix_net
```

FROM article; retourne {(pantalon, 39.95), (pull, 14), (veste, 40.05), (chemise, 20)}

#### Coalesce versus case

• En termes de performance case et coalesce sont équivalents, mais coalesce rend l'écriture de la requête plus concise

```
SELECT nom, (prix - CASE WHEN reduction IS NULL THEN 0

ELSE reduction

END) AS prix_net

FROM article;

versus

SELECT nom, (prix- COALESCE(reduction,0)) as prix_net

FROM Article;
```

Studio		
nom	titre	
United	Licence to kill	
United Rain Man		
Dreamworks Gladiator		

Films		
titre	bénéfice	
Licence to kill	156	
Rain Man	412	
Fargo	25	

· Requête : pour chaque studio, trouver le bénéfice global généré par ses films

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL JOIN FILMS
GROUP BY STUDIO.NOM;

- On fait d'abord la jointure
  - mais Dreamworks est perdu: Gladiator ne correspond à rien dans Films

nom	titre	bénéfice
United	Licence to kill	156
United	Rain Man	412

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

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL JOIN FILMS
GROUP BY STUDIO.NOM;

- Résultat : {(United, 568)}
  - mais Dreamworks est perdu: Gladiator ne correspond à rien dans Films

nom	titre	bénéfice
United	Licence to	156
United	Rain Man	412
Dreamwork	Gladiator	NULL

 Requête : pour chaque studio, trouver le bénéfice global généré par ses films (en listant tous les studios)

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL LEFT OUTER JOIN FILMS
GROUP BY STUDIO.NOM;

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

nom	titre	bénéfice
United	Licence to	156
United	Rain Man	412
Dreamwork	Gladiator	NULL

 Requête : pour chaque studio, trouver le bénéfice global généré par ses films (en listant tous les studios)

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL LEFT OUTER JOIN FILMS
GROUP BY STUDIO.NOM;

• Résultat : {(United, 568), (Dreamworks, NULL)}

nom	titre	bénéfice
United	Licence to	156
United	Rain Man	412
NULL	Fargo	25

 Requête : pour chaque studio, trouver le bénéfice global généré par ses films (en considérant tous les films, mais pas forcément tous les studios)

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS
GROUP BY STUDIO.NOM;

• Résultat : {(United, 568), (NULL, 25)}

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

 Requête : pour chaque studio, trouver le bénéfice global généré par ses films (en considérant tous les films et tous les studios)

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL FULL OUTER JOIN FILMS
GROUP BY STUDIO.NOM;

• Résultat: {(United, 568), (Dreamworks, NULL), (NULL, 25)}

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

#### Remarque:

A LEFT OUTER JOIN B ON A.id=B.id
A RIGHT OUTER JOIN B ON A.id=B.id
A FULL OUTER JOIN B ON A.id=B.id
sont également des constructions
valides!

 Requête : pour chaque studio, trouver le bénéfice global généré par ses films (en considérant tous les films et tous les studios)

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL FULL OUTER JOIN FILMS
GROUP BY STUDIO.NOM;

• Résultat: {(United, 568), (Dreamworks, NULL), (NULL, 25)}

Attention : modifions un peu les données

Studio		
nom	titre	
United	Licence to kill	
United	Sun Man	
Dreamworks	Gladiator	

Films		
titre	bénéfice	
Licence to kill	156	
Rain Man	412	
Fargo	25	

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS

GROUP BY STUDIO.NOM;

#### Valeur de données NULL et GROUP BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

Comportement de GROUP BY :

```
SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS
GROUP BY STUDIO.NOM;
```

- Résultat : {(United, 156), (NULL, 437)}
  - le GROUP BY a groupé ensemble tous les tuples dont la valeur de STUDIO.NOM était NULL !

#### Valeur de données NULL et GROUP BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

• Comportement de GROUP BY:

SELECT STUDIO.NOM, SUM(Films.benefice)

FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS

GROUP BY STUDIO.NOM;

GROUP BY traite NULL

comme n'importe quelle

Résultat : {(United, 156), (NULL, 437)}

autre valeur

le GROUP BY a groupé ensemble tous les tuples dont la valeur de STUDIO.NOM était NULL !

#### Valeur de données NULL et GROUP BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

Comportement de GROUP BY :

SELECT COALESCE(STUDIO.NOM, 'Inconnu'), SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS

GROUP BY STUDIO.NOM;

On peut renommer les colonnes

NULL grâce à COALESCE

- Résultat : {(United, 156), (Inconnu, 437)}
  - le GROUP BY a groupé ensemble tous les tuples dont la valeur de STUDIO.NOM était NULL !

#### Valeur de données NULL et ORDER BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

Comportement de ORDER BY :

```
SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS
GROUP BY STUDIO.NOM
ORDER BY STUDIO.NOM;
```

- Résultat : {(United, 156), (NULL, 437)}
  - ORDER BY liste les valeurs NULL de STUDIO.NOM en dernier (spécifique à PostgreSQL)

#### Valeur de données NULL et ORDER BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

Comportement de ORDER BY :

```
SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS
GROUP BY STUDIO.NOM
ORDER BY STUDIO.NOM NULLS LAST;
```

- Résultat : {(United, 156), (NULL,437)}
  - l'ordre dans lequel les NULL sont listés peut être choisi

#### Valeur de données NULL et ORDER BY

nom	titre	bénéfice
United	Licence to	156
NULL	Rain Man	412
NULL	Fargo	25

Comportement de ORDER BY :

SELECT STUDIO.NOM, SUM(Films.benefice)
FROM STUDIO NATURAL RIGHT OUTER JOIN FILMS
GROUP BY STUDIO.NOM
ORDER BY STUDIO.NOM NULLS FIRST;

- Résultat : {(NULL, 437), (United, 156)}
  - l'ordre dans lequel les NULL sont listés peut être choisi