

1. SELECT * FROM film WHERE id=5200

Sans index :

Query Editor

Query History

1

explain select * from film WHERE id=5200

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Seq Scan on film (cost=0.00..183.32 rows=1 width=17)

2

[...] Filter: (id = 5200)

Avec index unique sur ID:

Query Editor		Query History		
1	CREATE UNIQUE INDEX idx_film ON film(id);			
2	EXPLAIN SELECT * FROM film WHERE id=5200;			
Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Index Scan using idx_film on film (cost=0.29..8.30 rows=1 width=17)			
2	[...] Index Cond: (id = 5200)			

On obtient le même résultat que l'exemple utilisant une clé primaire à la place de l'index.

2. `SELECT * FROM film WHERE id=5200 AND pays = 'CH/FR'`
Sans index :

Optimisation/postgres@PostgreSQL 13	
Query Editor	Query History
<pre>1 explain SELECT * FROM film WHERE id=5200 AND pays = 'CH/FR'</pre>	
Data Output Explain Messages Notifications	
QUERY PLAN	
text	
1	Seq Scan on film (cost=0.00..207.59 rows=1 width=17)
2	[...] Filter: ((id = 5200) AND ((pays)::text = 'CH/FR'::text))

Avec index unique sur ID :

Optimisation/postgres@PostgreSQL 13	
Query Editor	Query History
<pre>1 CREATE UNIQUE INDEX idx_film_id ON film(id); 2 explain SELECT * FROM film WHERE id=5200 AND pays = 'CH/FR'</pre>	
Data Output Explain Messages Notifications	
QUERY PLAN	
text	
1	Index Scan using idx_film_id on film (cost=0.29..8.30 rows=1 width=17)
2	[...] Index Cond: (id = 5200)
3	[...] Filter: ((pays)::text = 'CH/FR'::text)

Avec index non unique sur PAYS :

Optimisation/postgres@PostgreSQL 13	
Query Editor	Query History
<pre>1 CREATE UNIQUE INDEX idx_film_id ON film(id); 2 CREATE INDEX idx_film_pays ON film(pays); 3 explain SELECT * FROM film WHERE id=5200 AND pays = 'CH/FR'</pre>	
Data Output Explain Messages Notifications	
QUERY PLAN	
text	
1	Index Scan using idx_film_id on film (cost=0.29..8.30 rows=1 width=17)
2	[...] Index Cond: (id = 5200)
3	[...] Filter: ((pays)::text = 'CH/FR'::text)

On remarque que l'on obtient le même cout entre les deux manières de faire

Nos index ont les volumes suivants :

	Data Output	Explain	Messages	Notifications
	relation text		size (data) text	
1	public.titres		1168 kB (1136 kB data)	
2	public.film		880 kB (496 kB data)	
3	public.realise		400 kB (368 kB data)	
4	public.realisateur		336 kB (304 kB data)	
5	public.idx_film_id		232 kB (index)	
6	public.idx_film_pays		112 kB (index)	
7	public.seq_film		8192 bytes (8192 bytes data)	
8	public.seq_real		8192 bytes (8192 bytes data)	

On voit que l'index **idx_film_id** est le plus volumineux. On en déduit que le volume des index n'a pas d'influence sur le cout des requêtes.

3. SELECT * FROM film WHERE id=5200 OR pays ='CH/FR'

Sans index :

Query Editor	Query History
1 explain SELECT * FROM film WHERE id=5200 OR pays ='CH/FR';	
2	

Data Output	Explain	Messages	Notifications
QUERY PLAN text			
1 Seq Scan on film (cost=0.00..207.59 rows=8 width=17)			
2 [...] Filter: ((id = 5200) OR ((pays)::text = 'CH/FR':text))			

Avec index unique sur ID :

Query Editor	Query History
1 create unique index idx_film_id ON film(ID);	
2 explain SELECT * FROM film WHERE id=5200 OR pays ='CH/FR';	
3	

Data Output	Explain	Messages	Notifications
QUERY PLAN text			
1 Seq Scan on film (cost=0.00..207.59 rows=8 width=17)			
2 [...] Filter: ((id = 5200) OR ((pays)::text = 'CH/FR':text))			



Avec index unique sur PAYS :

Query Editor		Query History
<pre>1 --create unique index idx_film_id ON film(ID); 2 create index idx_film_pays ON film(PAYS); 3 explain SELECT * FROM film WHERE id=5200 OR pays = 'CH/FR'; 4</pre>		
Data Output		Explain Messages Notifications
<div>QUERY PLAN</div> <div>text</div>		
1	Bitmap Heap Scan on film (cost=8.63..32.13 rows=8 width=17)	
2	[...] Recheck Cond: ((id = 5200) OR ((pays)::text = 'CH/FR'::text))	
3	[...] -> BitmapOr (cost=8.63..8.63 rows=8 width=0)	
4	[...] -> Bitmap Index Scan on idx_film_id (cost=0.00..4.29 rows=1 width=0)	
5	[...] Index Cond: (id = 5200)	
6	[...] -> Bitmap Index Scan on idx_film_pays (cost=0.00..4.34 rows=7 width=0)	
7	[...] Index Cond: ((pays)::text = 'CH/FR'::text)	



Dans le cas d'un **OR**, l'utilisation d'un index non unique sur **PAYS** permet un moindre cout d'utilisation que la même requête contenant un **AND**.

4. SELECT * FROM film WHERE id>2000

Sans index :

Query Editor		Query History		
1	explain SELECT * FROM film WHERE id>2000			
Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Seq Scan on film (cost=0.00..183.32 rows=7707 width=17)			
2	[...] Filter: (id > 2000)			


Avec index unique sur ID :

Query Editor		Query History		
1	create unique index idx_film_id ON film(ID);			
2	explain SELECT * FROM film WHERE id>2000			
Data Output		Explain	Messages	Notifications
		QUERY PLAN		
		text		
1	Seq Scan on film (cost=0.00..183.32 rows=7707 width=17)			
2	[...] Filter: (id > 2000)			

Dans le cas d'une recherche id>2000, une partie trop significative de la table est concernée, et l'utilisation d'un index devient contre-performante.

5. SELECT * FROM film WHERE id>8000

Sans index :

Query Editor		Query History		
1	<code>--create unique index idx_film_id ON film(ID);</code>			
2	<code>explain SELECT * FROM film WHERE id>8000</code>			
Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Seq Scan on film (cost=0.00..183.32 rows=1706 width=17)			
2	[...] Filter: (id > 8000)			

Avec index unique sur l'ID:

Query Editor		Query History	
1	create unique index idx_film_id ON film(ID);		
2	explain SELECT * FROM film WHERE id>8000		
Data Output			
Explain			
Messages			
Notifications			
QUERY PLAN			
text			
1	Index Scan using idx_film_id on film (cost=0.29..68.14 rows=1706 width...		
2	[...] Index Cond: (id > 8000)		

Contrairement à la question précédente, une recherche id>8000 concerne une plus faible partie de la table, et l'utilisation d'un index permet de diminuer le cout.

6. Index multicolonne

Avec index film(pays, année) :

SELECT * FROM film WHERE pays = 'CH/FR';

Query Editor		Query History
1	explain SELECT * FROM film WHERE pays = 'CH/FR';	
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Bitmap Heap Scan on film (cost=4.34..25.37 rows=7 width=17)	
2	[...] Recheck Cond: ((pays)::text = 'CH/FR':text)	
3	[...] -> Bitmap Index Scan on idx_film_pays_annee (cost=0.00..4.34 rows=7)	
4	[...] Index Cond: ((pays)::text = 'CH/FR':text)	

SELECT * FROM film WHERE annee = 1991;

Query Editor		Query History
1	explain SELECT * FROM film WHERE annee = 1991;	
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Seq Scan on film (cost=0.00..183.32 rows=178 width=17)	
2	[...] Filter: (annee = 1991)	

SELECT * FROM film WHERE pays = 'CH/FR' OR annee = 1991;

Query Editor		Query History
1	explain SELECT * FROM film WHERE pays = 'CH/FR' OR annee = 1991;	
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Seq Scan on film (cost=0.00..207.59 rows=185 width=17)	
2	[...] Filter: (((pays)::text = 'CH/FR':text) OR (annee = 1991))	

SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR';

Query Editor		Query History
1	explain	SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR';
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Seq Scan on film (cost=0.00..207.59 rows=185 width=17)	
2	[...] Filter: ((annee = 1991) OR ((pays)::text = 'CH/FR'::text))	

Avec index film(annee, pays) :

SELECT * FROM film WHERE pays = 'CH/FR';

Query Editor		Query History
1	explain	SELECT * FROM film WHERE pays = 'CH/FR';
2		
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Index Scan using idx_film_annee_pays on film (cost=0.29..155.65 rows=...	
2	[...] Index Cond: ((pays)::text = 'CH/FR'::text)	

SELECT * FROM film WHERE annee = 1991;

Query Editor		Query History
1	explain	SELECT * FROM film WHERE annee = 1991;
2		
Data Output		
Explain		
Messages		
Notifications		
	QUERY PLAN	
	text	
1	Bitmap Heap Scan on film (cost=5.66..69.89 rows=178 width=17)	
2	[...] Recheck Cond: (annee = 1991)	
3	[...] -> Bitmap Index Scan on idx_film_annee_pays (cost=0.00..5.62 rows=...	
4	[...] Index Cond: (annee = 1991)	

SELECT * FROM film WHERE pays = 'CH/FR' OR annee = 1991;

Query Editor		Query History		
1	explain SELECT * FROM film WHERE pays = 'CH/FR' OR annee = 1991;			
2				
Data Output		Explain	Messages	Notifications
QUERY PLAN				
text				
1	Bitmap Heap Scan on film (cost=5.66..69.89 rows=178 width=17)			
2	[...] Recheck Cond: (annee = 1991)			
3	[...] -> Bitmap Index Scan on idx_film_annee_pays (cost=0.00..5.62 rows=...			
4	[...] Index Cond: (annee = 1991)			

SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR';

Query Editor		Query History		
1	explain SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR';			
2				
Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Seq Scan on film (cost=0.00..207.59 rows=185 width=17)			
2	[...] Filter: ((annee = 1991) OR ((pays)::text = 'CH/FR'::text))			

Index monocolonne :

CREATE INDEX idx_film_pays ON film(pays);

CREATE INDEX idx_film_annee ON film(annee);

SELECT * FROM film WHERE pays = 'CH/FR'; **Cout : 25.37**

SELECT * FROM film WHERE annee = 1991; **Cout : 12.40**

SELECT * FROM film WHERE pays = 'CH/FR' OR annee = 1991; **Cout : 74.83**

SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR'; **Cout : 74.83**

Malgré le plus faible volume des index multicolonne que les index monocolonne, ils ne sont que rarement utilisés sous PostgreSQL, contrairement aux simple index, plus pertinents et fréquemment utilisés par ce SGBD

7. Suite index multicolonne

On préconisera d'indexer à part les id_real et id_film, puisqu'elles sont toutes deux clés de leur table.

8. `SELECT * FROM film WHERE SUBSTR(pays,1,2) = 'CH';`

Sans index :

Query Editor	Query History
<pre>1 explain SELECT * FROM film WHERE SUBSTR(pays,1,2) = 'CH';</pre>	
Data Output	Explain Messages Notifications
QUERY PLAN text	
1	Seq Scan on film (cost=0.00..207.59 rows=49 width=17)
2	[...] Filter: (substr((pays)::text, 1, 2) = 'CH'::text)

Avec index non unique sur nom :

Query Editor	Query History
<pre>1 explain SELECT * FROM film WHERE SUBSTR(pays,1,2) = 'CH';</pre>	
Data Output	Explain Messages Notifications
QUERY PLAN text	
1	Seq Scan on film (cost=0.00..207.59 rows=49 width=17)
2	[...] Filter: (substr((pays)::text, 1, 2) = 'CH'::text)

Avec index sur fonction :

Query Editor	Query History
<pre>1 explain SELECT * FROM film WHERE SUBSTR(pays,1,2) = 'CH';</pre>	
Data Output	Explain Messages Notifications
QUERY PLAN text	
1	Bitmap Heap Scan on film (cost=4.66..67.10 rows=49 width=17)
2	[...] Recheck Cond: (substr((pays)::text, 1, 2) = 'CH'::text)
3	[...] -> Bitmap Index Scan on idx_film_substr_pays (cost=0.00..4.65 rows=49 width=17)
4	[...] Index Cond: (substr((pays)::text, 1, 2) = 'CH'::text)

Dans ce cas, l'utilisation de l'index sur fonction est privilégiée car sa rapidité de recherche est plus importante que la rapidité d'insertion et de mise à jour.

9.

Sans index :

Query Editor		Query History
1	<code>explain SELECT t.id_film, t.titre, t.langue, f.annee, f.pays</code>	
2	<code>FROM Film f</code>	
3	<code>JOIN Titres t ON f.id = t.id_film;</code>	
4		

Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Hash Join (cost=280.38..903.25 rows=20247 width=33)			
2	[...] Hash Cond: (t.id_film = f.id)			
3	[...] -> Seq Scan on titres t (cost=0.00..344.47 rows=20247 width=25)			
4	[...] -> Hash (cost=159.06..159.06 rows=9706 width=12)			
5	[...] -> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=12)			

Avec index unique sur id de film:

Query Editor		Query History
1	<code>explain SELECT t.id_film, t.titre, t.langue, f.annee, f.pays</code>	
2	<code>FROM Film f</code>	
3	<code>JOIN Titres t ON f.id = t.id_film;</code>	

Data Output		Explain	Messages	Notifications
	QUERY PLAN			
	text			
1	Hash Join (cost=280.38..678.03 rows=20247 width=33)			
2	[...] Hash Cond: (t.id_film = f.id)			
3	[...] -> Seq Scan on titres t (cost=0.00..344.47 rows=20247 width=25)			
4	[...] -> Hash (cost=159.06..159.06 rows=9706 width=12)			
5	[...] -> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=12)			

Il n'y a pas eu de modification au niveau du plan d'exécution, malgré la diminution du cout de la requête.

Avec index non unique sur titres(id_film) :

Query Editor

Query History

1

explain SELECT t.id_film, t.titre, t.langue, f.annee, f.pays

2

FROM Film f

3

JOIN Titres t ON f.id = t.id_film;

Data Output

Explain

Messages

Notifications

▲

QUERY PLAN

text

🔒

1

Hash Join (cost=280.38..678.03 rows=20247 width=33)

2

[...] Hash Cond: (t.id_film = f.id)

3

[...] -> Seq Scan on titres t (cost=0.00..344.47 rows=20247 width=25)

4

[...] -> Hash (cost=159.06..159.06 rows=9706 width=12)

5

[...] -> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=12)

Pas de modification sur le plan d'exécution

10.

Query Editor

Query History

1

2

3

4

5

explain -SELECT t.id_film, t.titre, t.langue, f.annee, f.pays

FROM Film f

JOIN Titres t ON f.id = t.id_film

WHERE f.pays='FR/BE';

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

2

3

4

5

6

Hash Join (cost=183.76..604.89 rows=73 width=33)

[...] Hash Cond: (t.id_film = f.id)

[...] -> Seq Scan on titres t (cost=0.00..344.47 rows=20247 width=25)

[...] -> Hash (cost=183.32..183.32 rows=35 width=12)


[...] -> Seq Scan on film f (cost=0.00..183.32 rows=35 width=12)

[...] Filter: ((pays)::text = 'FR/BE'::text)

On a un filtre supplémentaire par rapport à la question précédente : **WHERE f.pays='FR/BE'**, et le cout est plus faible.

Le fait d'ajouter un nouveau filtre diminue la quantité d'éléments à afficher et dans lesquels chercher

Ajout d'un index unique sur l'id de film :

Query Editor		Query History
<pre>1 explain SELECT t.id_film, t.titre, t.langue, f.annee, f.pays 2 FROM Film f 3 JOIN Titres t ON f.id = t.id_film 4 WHERE f.pays='FR/BE'; 5</pre>		
Data Output		Explain Messages Notifications
QUERY PLAN text		
1	Hash Join (cost=183.76..581.40 rows=73 width=33)	
2	[...] Hash Cond: (t.id_film = f.id)	
3	[...] -> Seq Scan on titres t (cost=0.00..344.47 rows=20247 width=25)	
4	[...] -> Hash (cost=183.32..183.32 rows=35 width=12)	
5	[...] -> Seq Scan on film f (cost=0.00..183.32 rows=35 width=12)	
6	[...] Filter: ((pays)::text = 'FR/BE'::text)	

On observe une baisse du cout lors de l'ajout de l'index unique.

11.

Query Editor	Query History
1	CREATE OR REPLACE VIEW films1995 AS
2	SELECT id, annee
3	FROM film
4	WHERE annee = 1995;
5	explain SELECT * FROM films1995;
Data Output	Explain Messages Notifications
	<div>QUERY PLAN</div> <div>text</div> <div>1</div> <div>Seq Scan on film (cost=0.00..183.32 rows=639 width=8)</div> <div>2</div> <div>[...] Filter: (annee = 1995)</div>

On peut préconiser d'optimiser la **VIEW** (par exemple en ajoutant un index).

Optimisation de requêtes :

12. Requetes similaires en résultat mais pas en cout

1^{ère} requête :

Query Editor	Query History
1	explain SELECT id FROM film
2	WHERE id >= all(SELECT id FROM film);
3	
Data Output	Explain Messages Notifications
	<div>QUERY PLAN</div> <div>text</div> <div>1</div> <div>Seq Scan on film (cost=0.00..1125375.64 rows=4853 wi...</div> <div>2</div> <div>[...] Filter: (SubPlan 1)</div> <div>3</div> <div>[...] SubPlan 1</div> <div>4</div> <div>[...] -> Materialize (cost=0.00..207.59 rows=9706 width=4)</div> <div>5</div> <div>[...] -> Seq Scan on film film_1 (cost=0.00..159.06 rows=...</div>

2eme requête :

Query Editor	Query History
1	explain SELECT id FROM film f1
2	WHERE NOT EXISTS (SELECT * FROM film f2 WHERE f1.id < f2.id);
Data Output	Explain Messages Notifications
	<div>QUERY PLAN</div> <div>text</div> <div>1</div> <div>Nested Loop Anti Join (cost=0.29..3097.84 rows=6471 ...</div> <div>2</div> <div>[...] -> Seq Scan on film f1 (cost=0.00..159.06 rows=970...</div> <div>3</div> <div>[...] -> Index Only Scan using idx_film_id on film f2 (cost=...</div> <div>4</div> <div>[...] Index Cond: (id > f1.id)</div>

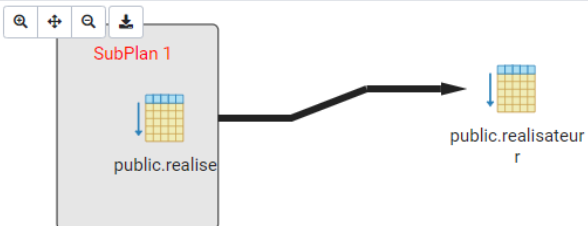
3eme requête :

Query Editor	Query History
1 explain SELECT MAX (id) FROM film	
Data Output	Explain
QUERY PLAN	
text	
1	Result (cost=0.31..0.32 rows=1 width=4)
2	[...] InitPlan 1 (returns \$0)
3	[...] -> Limit (cost=0.29..0.31 rows=1 width=4)
4	[...] -> Index Only Scan Backward using idx_film_id on fil...
5	[...] Index Cond: (id IS NOT NULL)

On remarque que la requête 3 est la moins couteuse suivis de loin par la 2 suivis par la 1 extrêmement couteuse.

13. Gf

Requête A :

Query Editor	Query History																						
1 SELECT R.id 2 FROM Realisateur R 3 WHERE R.id NOT IN (4 SELECT id_real 5 FROM Realise 6);																							
Data Output	Explain																						
Graphical Analysis Statistics																							
																							
<table><thead><tr><th>public.realise</th><th>x</th></tr></thead><tbody><tr><td>Node Type</td><td>Seq Scan</td></tr><tr><td>Parent Relationship</td><td>SubPlan</td></tr><tr><td>Subplan Name</td><td>SubPlan 1</td></tr><tr><td>Parallel Aware</td><td>false</td></tr><tr><td>Relation Name</td><td>realise</td></tr><tr><td>Schema</td><td>public</td></tr><tr><td>Alias</td><td>realise</td></tr><tr><td>Startup Cost</td><td>0</td></tr><tr><td>Total Cost</td><td>148.45</td></tr><tr><td>Plan Rows</td><td>10245</td></tr></tbody></table>		public.realise	x	Node Type	Seq Scan	Parent Relationship	SubPlan	Subplan Name	SubPlan 1	Parallel Aware	false	Relation Name	realise	Schema	public	Alias	realise	Startup Cost	0	Total Cost	148.45	Plan Rows	10245
public.realise	x																						
Node Type	Seq Scan																						
Parent Relationship	SubPlan																						
Subplan Name	SubPlan 1																						
Parallel Aware	false																						
Relation Name	realise																						
Schema	public																						
Alias	realise																						
Startup Cost	0																						
Total Cost	148.45																						
Plan Rows	10245																						

Requête B :

Query Editor Query History

```

1 SELECT R.id
2 FROM Realisateur R
3 WHERE NOT EXISTS (
4 SELECT 'X'
5 FROM Realise Re
6 WHERE Re.id_real=R.id
7 );

```

Data Output **Explain** Messages Notifications

Graphical Analysis Statistics

🔍 ⛶ 🔍 📄

public.realisateur
r

public.realise

Hash

Hash Anti Join

Hash Anti Join	
Node Type	Hash Join
Parallel Aware	false
Join Type	Anti
Startup Cost	276.51
Total Cost	777.6
Plan Rows	1
Plan Width	4
Output	r.id
Inner Unique	false
Hash Cond	(r.id = re.id_real)
loops	1
_serial	1

Requête C :

Query Editor Query History

```

1 SELECT R.id
2 FROM Realisateur R
3 LEFT JOIN Realise Re ON R.id=Re.id_real
4 WHERE Re.id_real IS NULL;

```

Data Output **Explain** Messages Notifications

Graphical Analysis Statistics

🔍 ⛶ 🔍 📄

public.realisateur
r

public.realise

Hash

Hash Anti Join

Hash Anti Join	
Node Type	Hash Join
Parallel Aware	false
Join Type	Anti
Startup Cost	276.51
Total Cost	777.6
Plan Rows	1
Plan Width	4
Output	r.id
Inner Unique	false
Hash Cond	(r.id = re.id_real)
loops	1
_serial	1

La requête la moins couteuse est donc la requête A

14. Sur mon PC :

SELECT * FROM film WHERE id BETWEEN 2000 and 2001; → 1.6ms

SELECT * FROM film WHERE id = 2000 OR id= 2001; → 1.975 ms

SELECT * FROM film WHERE id IN (2000, 2001); → 0.877 ms

SELECT * FROM film WHERE id = 2000 UNION SELECT * FROM film WHERE id = 2001; → 2.55 ms

La troisième requête est donc la plus performante. On en déduit que l'utilisation du **IN** est plus optimisée quand à **BETWEEN / OR / UNION** dans ce cas précis.

15. Opérateur relationnel de division

Requête A :

Query Editor Query History

```
1 explain SELECT r.id, r.nom
2 FROM realisateur r
3 WHERE NOT EXISTS
4 (SELECT 'X'
5 FROM film f
6 WHERE NOT EXISTS
7 (SELECT 'X'
8 FROM realise rea
9 WHERE rea.id_film=f.id AND rea.id_real=r.id));
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 Nested Loop Anti Join (cost=0.00..5792436113.15 rows=2988 width=19)

Requête B :

Query Editor Query History

```
1 explain SELECT r.id, r.nom
2 FROM realisateur r
3 JOIN realise rea ON r.id=rea.id_real
4 GROUP BY r.id, r.nom
5 HAVING COUNT(DISTINCT rea.id_film) = (
6 SELECT COUNT(*) FROM film
7 );
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 GroupAggregate (cost=1327.57..1504.72 rows=30 width=19)

