

1.

```
select * from realisateur where id=2800
```

Sans index :

Panneau sortie	
Sortie de données	Expliquer (Explain) Messages Historique
	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..112.70 rows=1 width=19)
2	Filter: (id = 2800)

On remarque que la requete fait un balayage sur toute la table.

Avec un index unique :

	QUERY PLAN
	text
1	Index Scan using idx rea id on realisateur (cost=0.28..8.30 rows=1 width=19)
2	Index Cond: (id = 2800)

On remarque que la requete ne parcourt plus que l'index et a un plus faible coût.

2.

```
select * from realisateur where id=2800 and nom='spielberg'
```

Plan d'exécution :

Sortie de données	Expliquer (Explain) Messages Historique
	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..127.64 rows=1 width=19)

on rajoute l'index unique :

Sortie de données		Expliquer (Explain)	Messages	Historique	▼
	QUERY PLAN				
	text				
1	Index Scan using indx on realisateur (cost=0.28..8.30 rows=1 width=19)				
2	Index Cond: (id = 2800)				
3	Filter: ((nom)::text = 'spielberg'::text)				

On remarque que la requete ne parcourt plus que l'index, et il fait ensuite un filtre sur le nom

On rajoute un index sur le nom du réalisateur :

	QUERY PLAN				
	text				
1	Index Scan using indx real on realisateur (cost=0.28..8.30 rows=1 width=19)				
2	Index Cond: ((nom)::text = 'spielberg'::text)				
3	Filter: (id = 2800)				

On remarque qu'il utilise l'index réalisateur car c'est le dernier index mis à jour même si il est moins pertinent dans ce cas car il est plus lourd (sur un même nombre de ligne le char est plus volumineux que le int).

3.

```
select * from realisateur where id=2800 or nom='spielberg'
```

Plan d'exécution :

	QUERY PLAN				
	text				
1	Seq Scan on realisateur (cost=0.00..127.64 rows=2 width=19)				
2	Filter: ((id = 2800) OR ((nom)::text = 'spielberg'::text))				

Sans surprise sans index la requete balaye toute la table. Le plan est le même que pour la question précédente sans index à l'exception du filtre qui change du and au or.

Avec un seul des deux index on reste sur un seq scan, cependant lors de la présence des deux simultanément on passe sur ce plan :

Panneau sortie	
Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Bitmap Heap Scan on realisateur (cost=8.58..15.23 rows=2 width=19)
2	Recheck Cond: ((id = 2800) OR ((nom)::text = 'spielberg'::text))
3	-> BitmapOr (cost=8.58..8.58 rows=2 width=0)
4	-> Bitmap Index Scan on idx real (cost=0.00..4.29 rows=1 width=0)
5	Index Cond: (id = 2800)
6	-> Bitmap Index Scan on idx real nom (cost=0.00..4.29 rows=1 width=0)
7	Index Cond: ((nom)::text = 'spielberg'::text)

On utilise bien les deux index contrairement à la requête précédente qui ne parcourt que l'index le plus récent et le coup est bien diminué.

4.

```
select * from realisateur where id>1000
```

le plan d'exécution :

Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..112.70 rows=4976 width=19)
2	Filter: (id > 1000)

avec un index unique :

	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..112.70 rows=4976 width=19)
2	Filter: (id > 1000)

On voit qu'il n'utilise pas l'index car il estime avec les stats que l'utilisation de l'index est trop coûteux cependant si on lui demande un filtre plus précis

5.

<pre>select * from realisateur where id>4000</pre>	
anneau sortie	
Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Index Scan using idx real on realisateur (cost=0.28..78.86 rows=1976 width=)
2	Index Cond: (id > 4000)

On voit avec un filtre plus précis que la requete passe bien par l'index.

6.

<pre>explain select * from titres where titre='Char'</pre>	
anneau sortie	
Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Seq Scan on titres (cost=0.00..395.09 rows=1 width=25)
2	Filter: ((titre)::text = 'Char'::text)

On ajoute un index unique sur le couple (id_film,titre,langue) :

Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Seq Scan on titres (cost=0.00..395.09 rows=1 width=25)
2	Filter: ((titre)::text = 'Char'::text)

on remarque que postgres ne sait pas gérer les index composite car si on crée un index unique (titre,langue,id_film) il l'utilise :

Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Index Only Scan using idx uk titre on titres (cost=0.29..8.30 rows=1 width=)
2	Index Cond: (titre = 'Char'::text)

7.

```
explain
select * from titres where titre='Char' and id_film=1000
```

anneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Seq Scan on titres (cost=0.00..445.70 rows=1 width=25)
2	Filter: (((titre)::text = 'Char'::text) AND (id film = 1000))

avec un index unique composite (id_film,titre,langue) :

anneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Index Only Scan using idx uk titre on titres (cost=0.29..8.31 rows=1 width=)
2	Index Cond: ((id film = 1000) AND (titre = 'Char'::text))

l'index est bien utilisé.

On remplace le and par or :

```
explain
select * from titres where titre='Char' or id film=1000

create unique index idx uk titre on Titres(id film titre langue)
```

anneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Seq Scan on titres (cost=0.00..445.70 rows=3 width=25)
2	Filter: (((titre)::text = 'Char'::text) OR (id film = 1000))

on voit que la requete ne passe par l'index car il faut un index différent pour chaque partie du or est l'index composite n'est pas interpréter comme tel sur postgres.

8.

```
select * from realisateur where substr(nom,1,2) = 'Sp'
```

anneau sortie	
Sortie de données	Expliquer (Explain) Messages Historique
	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..127.64 rows=30 width=19)
2	Filter: (substr((nom)::text, 1, 2) = 'Sp'::text)

on voit que la requete parcourt toute la table avant de faire son filtre,

avec un index sur le nom du réalisateur :

```
select * from realisateur where substr(nom,1,2) = 'Sp'
```

anneau sortie	
Sortie de données	Expliquer (Explain) Messages Historique
	QUERY PLAN
	text
1	Seq Scan on realisateur (cost=0.00..127.64 rows=30 width=19)
2	Filter: (substr((nom)::text, 1, 2) = 'Sp'::text)

On voit que la requete n'utilise pas l'index car on passe par une fonction de transformation. Si on crée l'index sur cette dite fonction on a le plan d'exécution suivant :

Sortie de données		Expliquer (Explain) Messages Historique
	QUERY PLAN	
	text	
1	Bitmap Heap Scan on realisateur (cost=4.51..42.75 rows=30 width=19)	
2	Recheck Cond: (substr((nom)::text, 1, 2) = 'Sp'::text)	
3	-> Bitmap Index Scan on idx (cost=0.00..4.51 rows=30 width=0)	
4	Index Cond: (substr((nom)::text, 1, 2) = 'Sp'::text)	

9. Sans index

```
select t.*
from film f join titres t on f.id=t.id_film
```

anneau sortie	
Sortie de données	Expliquer (Explain) Messages Historique
	QUERY PLAN
	text
1	Hash Join (cost=280.38..903.25 rows=20247 width=25)
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=4)
5	-> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=4)

Sortie de données	Expliquer (Explain)	Messages	Historique
QUERY PLAN text			
1	Hash Join (cost=280.38..678.03 rows=20247 width=25)		
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=4)		
5	-> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=4)		

lorsque que l'on pose un index sur la PK , on voit que le coup de la requete a diminuer mais le plan n'a pas changer.

Sortie de données	Expliquer (Explain)	Messages	Historique
QUERY PLAN text			
1	Hash Join (cost=280.38..678.03 rows=20247 width=25)		
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=4)		
5	-> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=4)		

Lorsque l'on pose un index sur la Fk le cout/plan ne change pas car les statistiques determine que les index ne sont pas rentable.

10.

Sans index

<pre>explain select t.* from film f join titres t on f.id = t.id_film where f.id = 2800</pre>			
Panneau sortie			
Sortie de données	Expliquer (Explain)	Messages	Historique
QUERY PLAN text			
1	Nested Loop (cost=0.00..578.43 rows=2 width=25)		
2	-> Seq Scan on film f (cost=0.00..183.32 rows=1 width=4)		
3	Filter: (id = 2800)		
4	-> Seq Scan on titres t (cost=0.00..395.09 rows=2 width=25)		
5	Filter: (id film = 2800)		

On rajoute un index unique sur l'id de film :

QUERY PLAN text			
1	Nested Loop (cost=0.29..403.41 rows=2 width=25)		
2	-> Index Only Scan using idx on film f (cost=0.29..8.30 rows=1 width=4)		
3	Index Cond: (id = 2800)		
4	-> Seq Scan on titres t (cost=0.00..395.09 rows=2 width=25)		
5	Filter: (id film = 2800)		

on voit que l'index est bien utilisé lors du filtre sur la table film avant la jointure.

On rajoute un index sur le cahmp id_film :

Sortie de données		Expliquer (Explain)	Messages	Historique	▼
	QUERY PLAN text				
1	Nested Loop (cost=0.57..16.65 rows=2 width=25)				
2	-> Index Only Scan using idx on film f (cost=0.29..8.30 rows=1 width=4)				
3	Index Cond: (id = 2800)				
4	-> Index Scan using idx titre on titres t (cost=0.29..8.32 rows=2 width=)				
5	Index Cond: (id film = 2800)				

On voit que les deux index sont utilisé , on a diminuer le coup de la jointure par 10.

11.

```
explain
select * from film1995
```

anneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN
	text
1	Hash Join (cost=363.77..559.15 rows=674 width=23)
2	Hash Cond: (a.id real = r.id)
3	-> Hash Join (cost=191.31..384.92 rows=674 width=12)
4	Hash Cond: (a.id film = f.id)
5	-> Seq Scan on realise a (cost=0.00..148.45 rows=10245 width=8)
6	-> Hash (cost=183.32..183.32 rows=639 width=8)
7	-> Seq Scan on film f (cost=0.00..183.32 rows=639 width=8)
8	Filter: (annee = 1995)
9	-> Hash (cost=97.76..97.76 rows=5976 width=19)
10	-> Seq Scan on realisateur r (cost=0.00..97.76 rows=5976 width=)

On constate que la vue a un plan d'exécution et donc elle peut etre automatisé.

12.

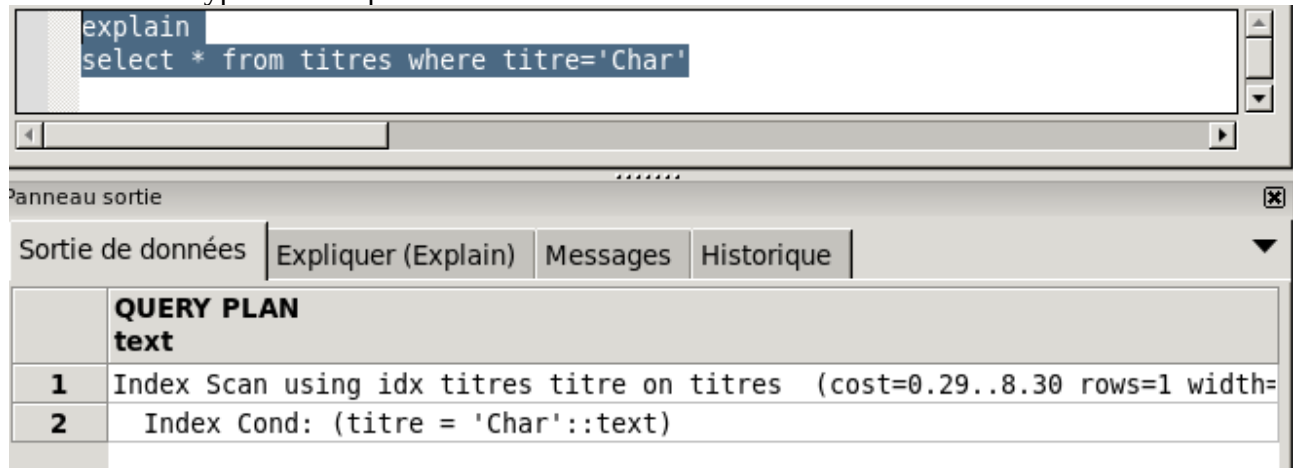
```
select * from pg_indexes where tablename='titres'
```

Panneau sortie

Sortie de données

Expliquer (Explain)	Messages	Historique		
schemaname name	tablename name	indexname name	tablespace name	indexdef text

Comme ça on est sûr de ne pas avoir d'index sur la table titres.
On modifie le type du champ titre en TEXT et on exécute :



The screenshot shows a database query tool interface. The top panel contains the SQL query: `explain select * from titres where titre='Char'`. Below the query is a horizontal scrollbar. The bottom panel, titled "Panneau sortie", has tabs for "Sortie de données", "Expliquer (Explain)", "Messages", and "Historique". The "Expliquer (Explain)" tab is selected, displaying the query plan. The plan is titled "QUERY PLAN" and "text". It consists of two steps: 1. "Index Scan using idx titres titre on titres (cost=0.29..8.30 rows=1 width=)" and 2. "Index Cond: (titre = 'Char'::text)".

```
explain
select * from titres where titre='Char'
```

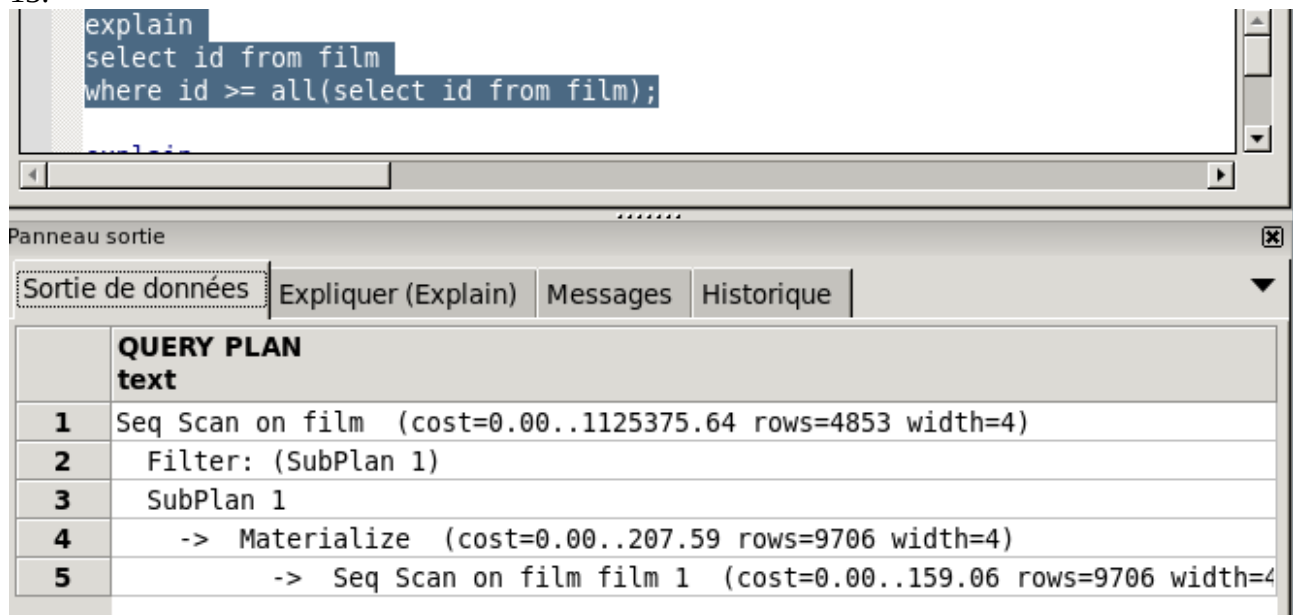
Panneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Index Scan using idx titres titre on titres (cost=0.29..8.30 rows=1 width=)
2	Index Cond: (titre = 'Char'::text)

On constate que l'index est tout aussi performant que le varchar , les deux type sont donc identique sur le sgbd.

13.



The screenshot shows a database query tool interface. The top panel contains the SQL query: `explain select id from film where id >= all(select id from film);`. Below the query is a horizontal scrollbar. The bottom panel, titled "Panneau sortie", has tabs for "Sortie de données", "Expliquer (Explain)", "Messages", and "Historique". The "Expliquer (Explain)" tab is selected, displaying the query plan. The plan is titled "QUERY PLAN" and "text". It consists of five steps: 1. "Seq Scan on film (cost=0.00..1125375.64 rows=4853 width=4)", 2. "Filter: (SubPlan 1)", 3. "SubPlan 1", 4. "-> Materialize (cost=0.00..207.59 rows=9706 width=4)", and 5. "-> Seq Scan on film film 1 (cost=0.00..159.06 rows=9706 width=4)".

```
explain
select id from film
where id >= all(select id from film);
```

Panneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Seq Scan on film (cost=0.00..1125375.64 rows=4853 width=4)
2	Filter: (SubPlan 1)
3	SubPlan 1
4	-> Materialize (cost=0.00..207.59 rows=9706 width=4)
5	-> Seq Scan on film film 1 (cost=0.00..159.06 rows=9706 width=4)

On remarque ici que l'on parcourt la table plusieurs fois,

<pre>explain select id from film f1 where not exists (select * from film f2 where f1.id<f2.id);</pre>	
anneau sortie	
<div>Sortie de données</div> <div> <div>Expliquer (Explain)</div> <div>Messages</div> <div>Historique</div> </div>	
	QUERY PLAN text
1	Nested Loop Anti Join (cost=0.00..942552.32 rows=6471 width=4)
2	Join Filter: (f1.id < f2.id)
3	-> Seq Scan on film f1 (cost=0.00..159.06 rows=9706 width=4)
4	-> Materialize (cost=0.00..207.59 rows=9706 width=4)
5	-> Seq Scan on film f2 (cost=0.00..159.06 rows=9706 width=4)

Meme remarque que la requete precedente , on a ici une requete synchronisé donc on parcours la table autant de fois que de lignes.

<pre>explain select max(id) from film;</pre>	
anneau sortie	
<div>Sortie de données</div> <div> <div>Expliquer (Explain)</div> <div>Messages</div> <div>Historique</div> </div>	
	QUERY PLAN text
1	Aggregate (cost=183.32..183.33 rows=1 width=4)
2	-> Seq Scan on film (cost=0.00..159.06 rows=9706 width=4)

On voit ici que l'on ne parcour la table qu'une seul fois , c'est donc la requete la plus performante et de loin.

14.

```

explain
select r.id
from realisateur r
where r.id not in (
    select id_real
    from realise
);

```

panneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Seq Scan on realisateur r (cost=174.06..286.76 rows=2988 width=4)
2	Filter: (NOT (hashed SubPlan 1))
3	SubPlan 1
4	-> Seq Scan on realise (cost=0.00..148.45 rows=10245 width=4)

```

explain
select r.id
from realisateur r
where not exists (
    select *
    from realise re
    where re.id real=r.id
);

```

panneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Hash Anti Join (cost=276.51..777.60 rows=1 width=4)
2	Hash Cond: (r.id = re.id real)
3	-> Seq Scan on realisateur r (cost=0.00..97.76 rows=5976 width=4)
4	-> Hash (cost=148.45..148.45 rows=10245 width=4)
5	-> Seq Scan on realise re (cost=0.00..148.45 rows=10245 width=4)

```

explain
select r.id
from realisateur r
left join realise re on r.id=re.id real
where re.id real is null;

```

anneau sortie

Sortie de données

Expliquer (Explain)

Messages

Historique

QUERY PLAN text

- | | |
|---|---|
| 1 | Hash Anti Join (cost=276.51..777.60 rows=1 width=4) |
| 2 | Hash Cond: (r.id = re.id real) |
| 3 | -> Seq Scan on realisateur r (cost=0.00..97.76 rows=5976 width=4) |
| 4 | -> Hash (cost=148.45..148.45 rows=10245 width=4) |
| 5 | -> Seq Scan on realise re (cost=0.00..148.45 rows=10245 width=4) |

```

explain
select r.id
from realisateur R
left join realise re on r.id=re.id real
group by r.id
having count(re.id real)=0;

```

anneau sortie

Sortie de données

Expliquer (Explain)

Messages

Historique

QUERY PLAN text

- | | |
|---|--|
| 1 | HashAggregate (cost=399.05..458.81 rows=5976 width=4) |
| 2 | Group Key: r.id |
| 3 | Filter: (count(re.id real) = 0) |
| 4 | -> Hash Right Join (cost=172.46..347.83 rows=10245 width=8) |
| 5 | Hash Cond: (re.id real = r.id) |
| 6 | -> Seq Scan on realise re (cost=0.00..148.45 rows=10245 width=4) |
| 7 | -> Hash (cost=97.76..97.76 rows=5976 width=4) |
| 8 | -> Seq Scan on realisateur r (cost=0.00..97.76 rows=5976 wi |

a. sans plan d'exécution la requete not in est plus performante que les autres,

- b. On peut deduire de ces différences que postgres est nettement moins performant que Oracle pour optimiser les requetes
- c. les requetes sont optimisables avec des indexes sur les clauses des where et les criteres de jointures

15.

explain

select * from film where id between 2000 and 2001;

Panneau sortie

Sortie de données

Expliquer (Explain)

Messages

Historique

QUERY PLAN

text

1	Seq Scan on film (cost=0.00..207.59 rows=1 width=17)
2	Filter: ((id >= 2000) AND (id <= 2001))

Requêtes précédentes

▼

Supprimer

explain

select * from film where id=2000 or id=2001;

Panneau sortie

Sortie de données

Expliquer (Explain)

Messages

Historique

QUERY PLAN

text

1	Seq Scan on film (cost=0.00..207.59 rows=2 width=17)
2	Filter: ((id = 2000) OR (id = 2001))

<pre>explain select * from film where id in (2000,2001);</pre>	
Panneau sortie	
Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Seq Scan on film (cost=0.00..183.32 rows=2 width=17)
2	Filter: (id = ANY ('{2000,2001}'::integer[]))

<pre>explain select * from film where id=2000 union select * from film where id=2001;</pre>	
Panneau sortie	
Sortie de données Expliquer (Explain) Messages Historique	
	QUERY PLAN
	text
1	Unique (cost=366.68..366.70 rows=2 width=118)
2	-> Sort (cost=366.68..366.68 rows=2 width=118)
3	Sort Key: film.id, film.pays, film.annee, film.eur
4	-> Append (cost=0.00..366.67 rows=2 width=118)
5	-> Seq Scan on film (cost=0.00..183.32 rows=1 width=17)
6	Filter: (id = 2000)
7	-> Seq Scan on film film 1 (cost=0.00..183.32 rows=1 width=17)
8	Filter: (id = 2001)

la requete la plus performante est la requete avec le IN , on note qu'il ne faut surtout pas faire de requete union car on parcourt la table deux fois.

16.

```

explain
SELECT r.id, r.nom
FROM realisateur r WHERE NOT EXISTS
  (SELECT 'X' FROM film f
   WHERE
    NOT EXISTS(
      SELECT 'X'
      FROM realise rea
      WHERE rea.id_film=f.id AND rea.id_real=r.id));

```

Panneau sortie

Sortie de données | Expliquer (Explain) | Messages | Historique

	QUERY PLAN text
1	Nested Loop Anti Join (cost=0.00..5792436113.15 rows=2988 width=19)
2	Join Filter: (NOT (alternatives: SubPlan 1 or hashed SubPlan 2))
3	-> Seq Scan on realisateur r (cost=0.00..97.76 rows=5976 width=19)
4	-> Materialize (cost=0.00..207.59 rows=9706 width=4)
5	-> Seq Scan on film f (cost=0.00..159.06 rows=9706 width=4)
6	SubPlan 1
7	-> Seq Scan on realise rea (cost=0.00..199.67 rows=1 width=0)
8	Filter: ((id film = f.id) AND (id real = r.id))
9	SubPlan 2
10	-> Seq Scan on realise rea 1 (cost=0.00..148.45 rows=10245 width=8)

```

explain
SELECT r.id, r.nom
FROM realisateur r
JOIN realise rea ON r.id=rea.id_real
GROUP BY r.id, r.nom
HAVING COUNT(DISTINCT rea.id_film) = (SELECT COUNT(*) FROM film);

```

anneau sortie

Sortie de données

Expliquer (Explain)

Messages

Historique

QUERY PLAN

text

1	GroupAggregate (cost=1213.61..1375.82 rows=5976 width=19)
2	Group Key: r.id, r.nom
3	Filter: (count(DISTINCT rea.id film) = \$0)
4	InitPlan 1 (returns \$0)
5	-> Aggregate (cost=183.32..183.33 rows=1 width=8)
6	-> Seq Scan on film (cost=0.00..159.06 rows=9706 width=0)
7	-> Sort (cost=1030.28..1055.89 rows=10245 width=23)
8	Sort Key: r.id, r.nom
9	-> Hash Join (cost=172.46..347.83 rows=10245 width=23)
10	Hash Cond: (rea.id real = r.id)
11	-> Seq Scan on realise rea (cost=0.00..148.45 rows=10245)
12	-> Hash (cost=97.76..97.76 rows=5976 width=19)
13	-> Seq Scan on realisateur r (cost=0.00..97.76 row:

on remarque que l'algorithme de division basé sur le count est beaucoup plus performant