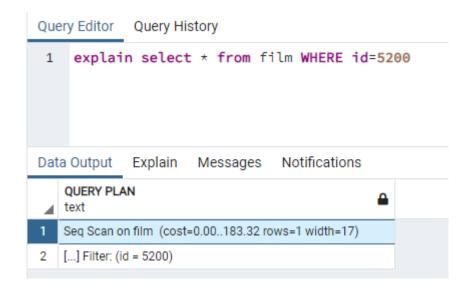
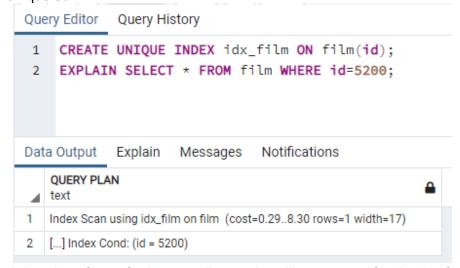
1. SELECT * FROM film WHERE id=5200

Sans index:

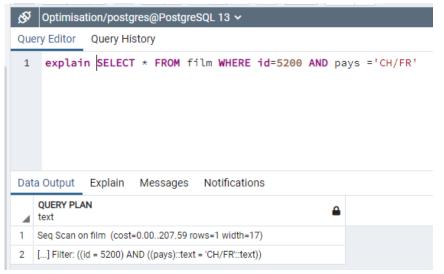


Avec index unique sur ID:



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

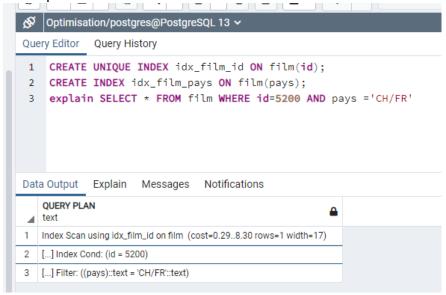
2. SELECT * FROM film WHERE id=5200 AND pays ='CH/FR' Sans index :



Avec index unique sur ID:



Avec index non unique sur PAYS:



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 | |
|---------------------|----------------------|----------|------------------------------|----------|
| 4 | relation text | <u> </u> | size (data) text | <u> </u> |
| 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'

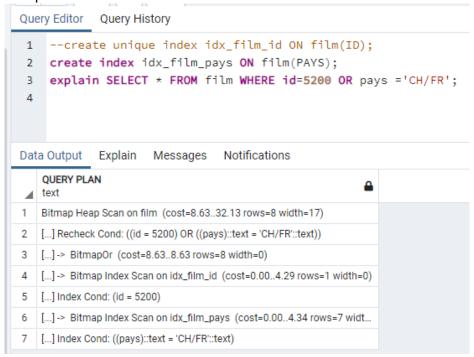
Sans index:



Avec index unique sur ID:

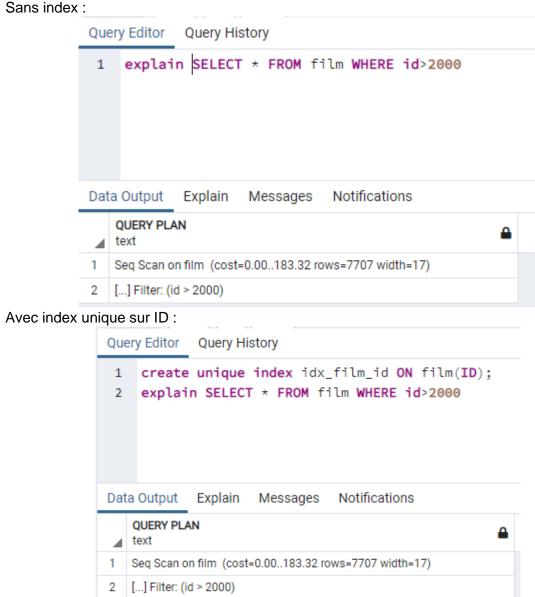


Avec index unique sur PAYS:



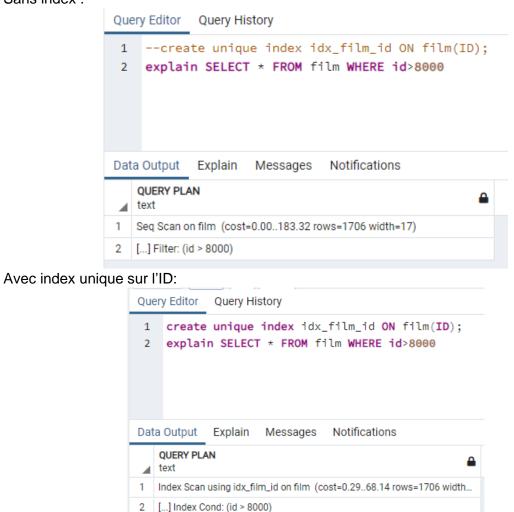
Dans le cas d'un **OR**, l'utilisation d'un index non unique sur **PAYS** permet un moindre cout d'utilisation que la même requête contenant un **AND**.

4. SELECT * FROM film WHERE id>2000



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 :

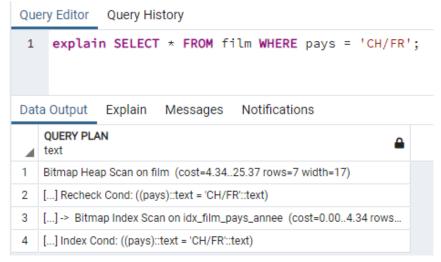


Contrairement à la question précédente, une recherche id>8000 concerne une plus faible partie de la table, et l'utilisation d'un index permet de diminuer le cout.

6. Index multicolonne

Avec index film(pays, année):

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



SELECT * FROM film WHERE annee = 1991;



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

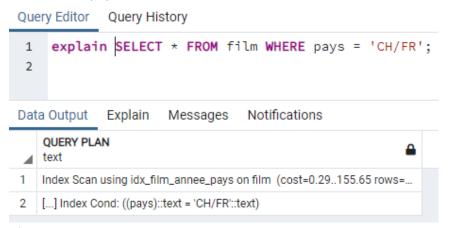


SELECT * FROM film WHERE annee = 1991 OR pays = 'CH/FR';

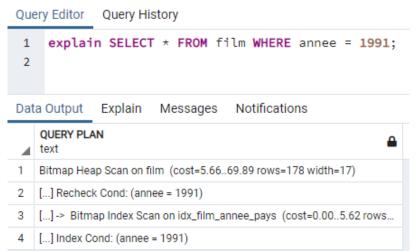


Avec index film(annee, pays):

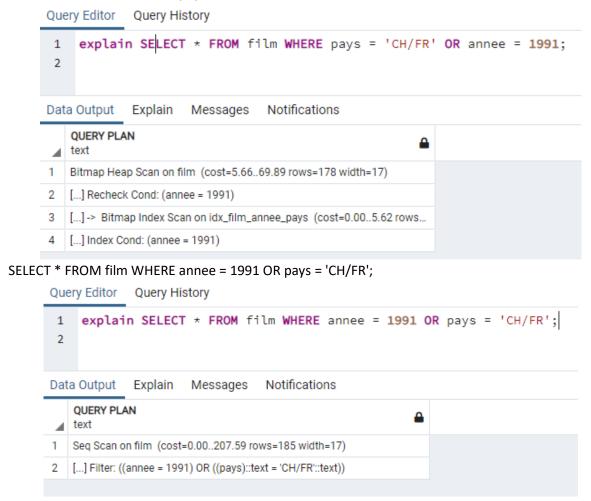
SELECT * FROM film WHERE pays = 'CH/FR';



SELECT * FROM film WHERE annee = 1991;



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



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

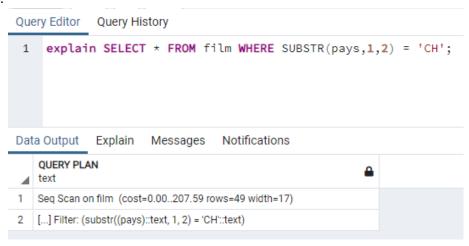
Malgré le plus faible volume des index multicolonne que les index monocolonne, ils ne sont que rarement utilisés sous PostgreSQL, contrairement aux simple index, plus pertinents et fréquemment utilisés par ce SGBD

7. Suite index multicolonne

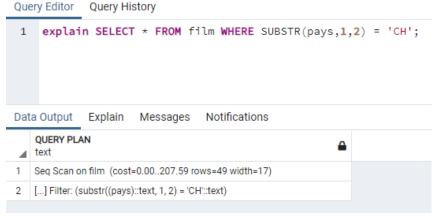
On préconisera d'indexer à part les id_real et id_film, puisqu'elles sont toutes deux clés de leur table.

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

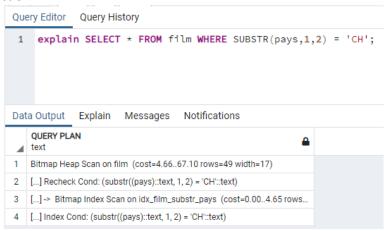
Sans index:



Avec index non unique sur nom:



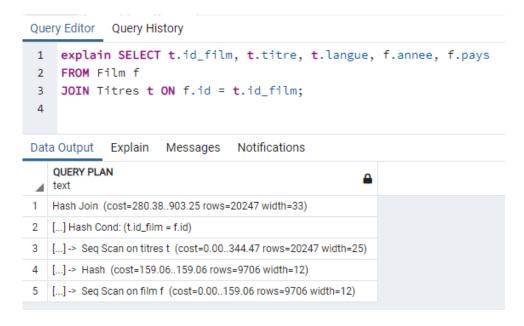
Avec index sur fonction:



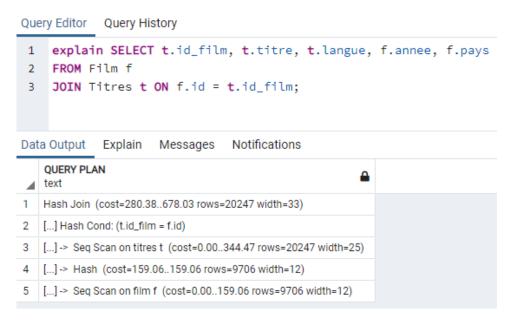
Dans ce cas, l'utilisation de l'index sur fonction est privilégiée car sa rapidité de recherche est plus importante que la rapidité d'insertion et de mise à jour.

9.

Sans index:

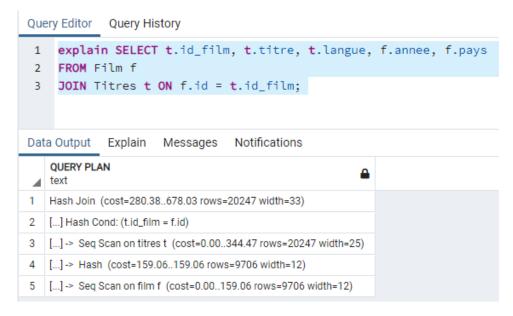


Avec index unique sur id de film:



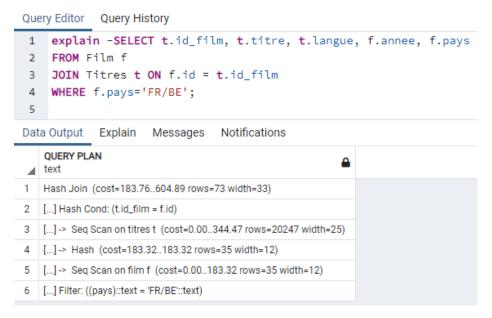
Il n'y a pas eu de modification au niveau du plan d'exécution, malgré la diminution du cout de la requête.

Avec index non unique sur titres(id_film):



Pas de modification sur le plan d'éxécution

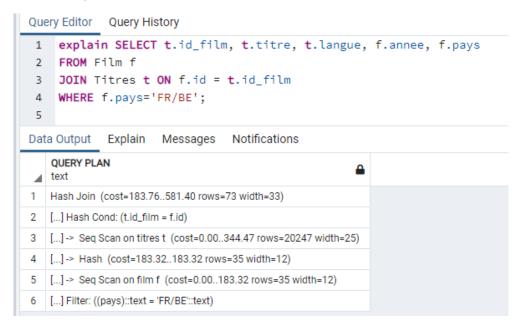
10.



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

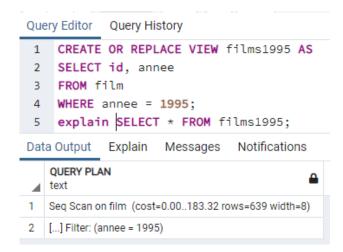
Le fait d'ajouter un nouveau filtre diminue la quantité d'éléments à afficher et dans lesquels chercher

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



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

11.

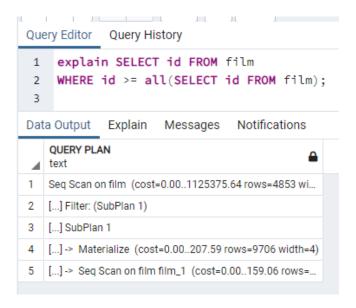


On peut préconiser d'optimiser la VIEW (par exemple en ajoutant un index).

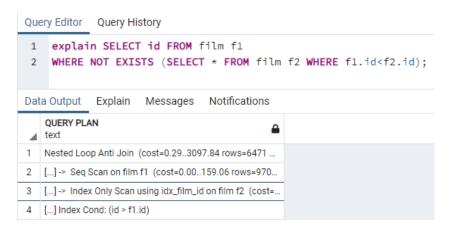
Optimisation de requêtes :

12. Requetes similaires en résultat mais pas en cout

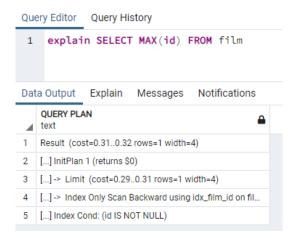
1ère requête:



2eme requête:



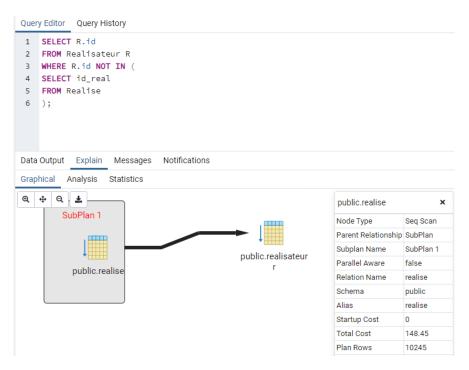
3eme requête:



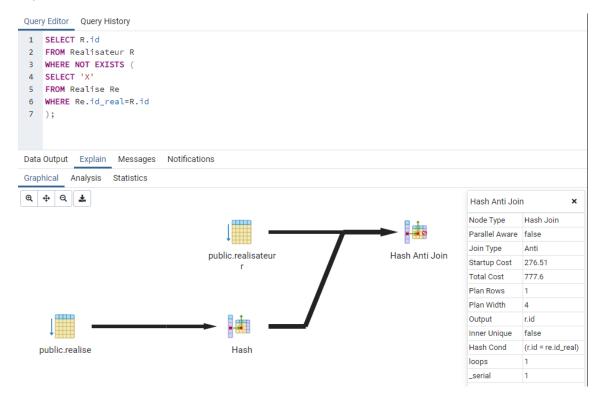
On remarque que la requête 3 est la moins couteuse suivis de loin par la 2 suivis par la 1 extrêmement couteuse.

13. Gf

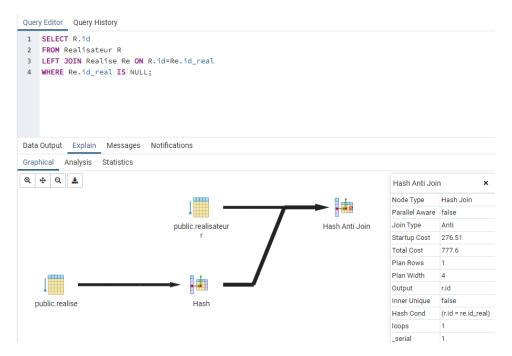
Requête A:



Requête B:



Requête C:



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

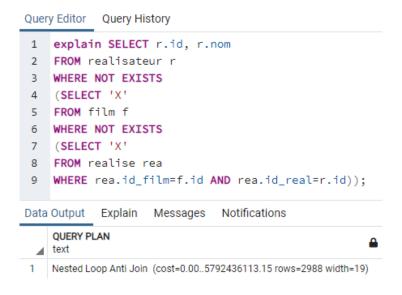
SELECT * FROM film WHERE id IN (2000, 2001); → 0.877 ms

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:



Requête B:

