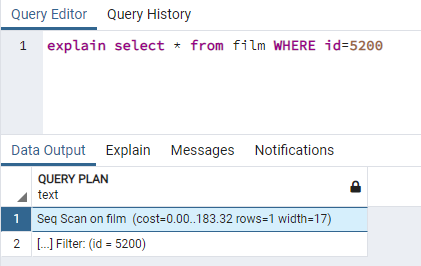
1. SELECT \* FROM film WHERE id=5200

Sans index :



Avec index unique sur ID:

Une image contenant texte

Description générée automatiquement

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

1. SELECT \* FROM film WHERE id=5200 AND pays =’CH/FR’

Sans index :

Une image contenant texte

Description générée automatiquement

Avec index unique sur ID :

Une image contenant texte

Description générée automatiquement

Avec index non unique sur PAYS :

Une image contenant texte

Description générée automatiquement

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

Nos index ont les volumes suivants :

Une image contenant table

Description générée automatiquement

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.

1. SELECT \* FROM film WHERE id=5200 OR pays =’CH/FR’

Sans index :

Une image contenant texte

Description générée automatiquement

Avec index unique sur ID :

Une image contenant texte

Description générée automatiquement

Avec index unique sur PAYS :

Une image contenant texte

Description générée automatiquement

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**.

1. SELECT \* FROM film WHERE id>2000

Sans index :

Une image contenant texte

Description générée automatiquement

Avec index unique sur ID :

Une image contenant texte

Description générée automatiquement

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.

1. SELECT \* FROM film WHERE id>8000

Sans index :

Une image contenant texte

Description générée automatiquement

Avec index unique sur l’ID:

Une image contenant texte

Description générée automatiquement

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.

1. Index multicolonne

Avec index film(pays, année) :

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

Une image contenant texte

Description générée automatiquement

SELECT \* FROM film WHERE annee = 1991;

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

Avec index film(annee, pays) :

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

Une image contenant texte

Description générée automatiquement

SELECT \* FROM film WHERE annee = 1991;

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

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**

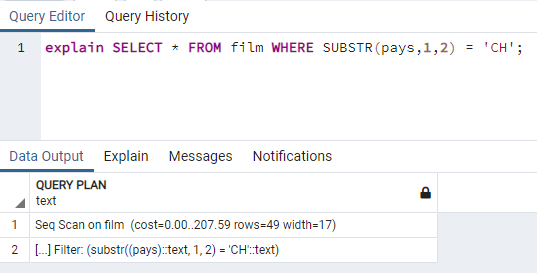
Malgré le plus faible volume des index multicolonne que les index monocolonne, ils ne sont que rarement utilisés sous PostgreSQL, contrairement aux simple index, plus pertinents et fréquemment utilisés par ce SGBD

1. Suite index multicolonne

On préconisera d’indexer à part les id\_real et id\_film, puisqu’elles sont toutes deux clés de leur table.

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

Sans index :



Avec index non unique sur nom :

Une image contenant texte

Description générée automatiquement

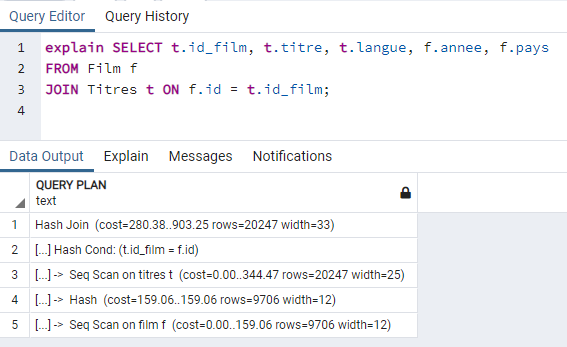
Avec index sur fonction :

Une image contenant texte

Description générée automatiquement

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.

Sans index :



Avec index unique sur id de film:

Une image contenant texte

Description générée automatiquement

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) :

Une image contenant texte

Description générée automatiquement

Pas de modification sur le plan d’éxécution

Une image contenant texte

Description générée automatiquement

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 :

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

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

Optimisation de requêtes :

1. Requetes similaires en résultat mais pas en cout

1ère requête :

Une image contenant texte

Description générée automatiquement

2eme requête :

Une image contenant texte

Description générée automatiquement

3eme requête :

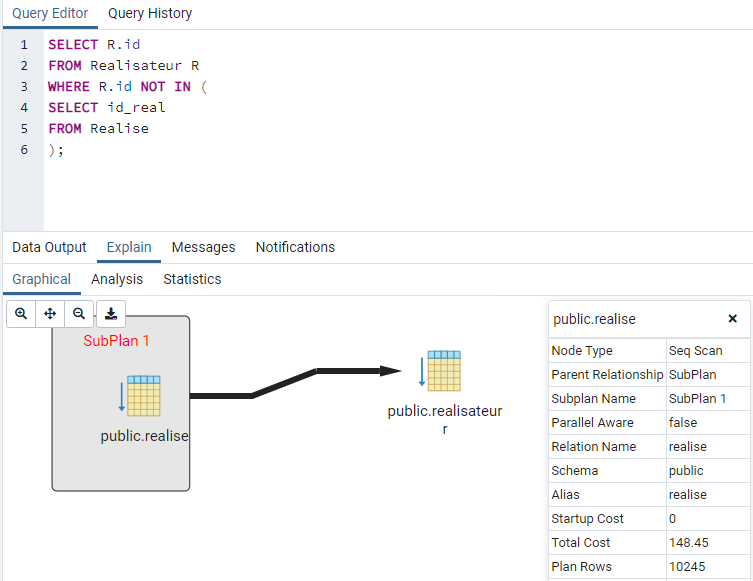
Une image contenant texte

Description générée automatiquement

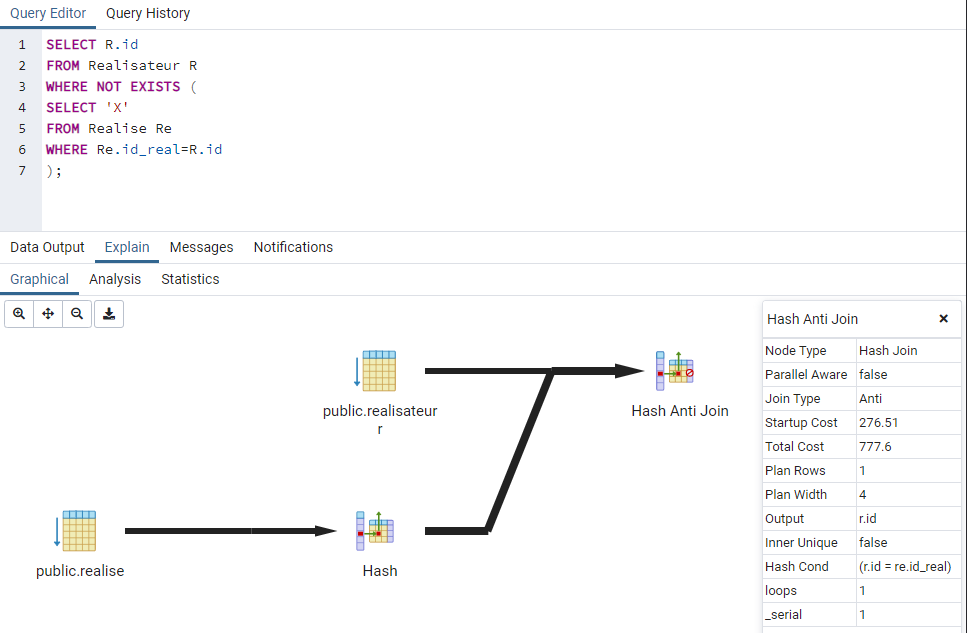
On remarque que la requête 3 est la moins couteuse suivis de loin par la 2 suivis par la 1 extrêmement couteuse.

1. Gf

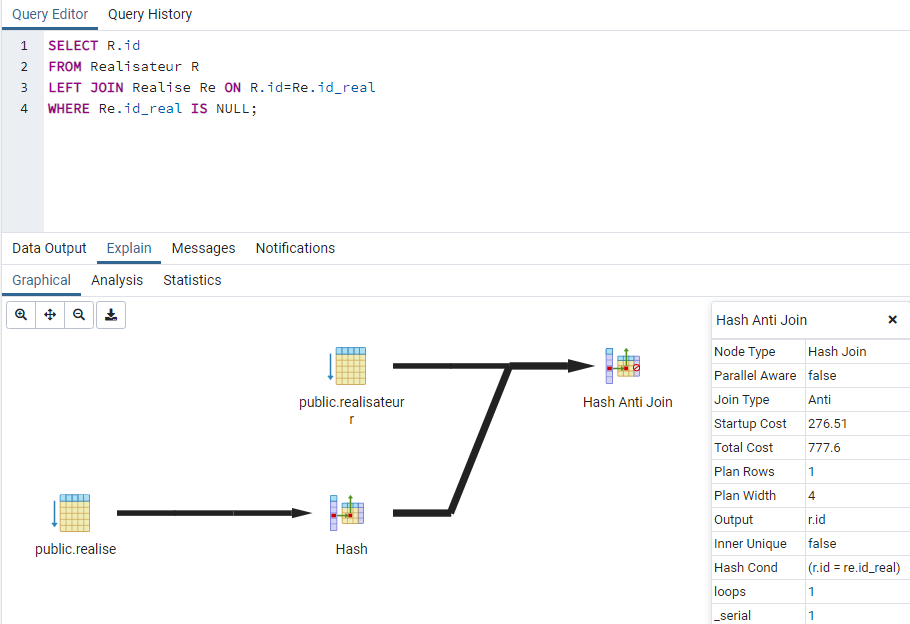
Requête A :



Requête B :



Requête C :



La requête la moins couteuse est donc la requête A

1. 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.

1. Opérateur relationnel de division

Requête A :

Une image contenant texte

Description générée automatiquement

Requête B :

Une image contenant texte

Description générée automatiquement