TP 2

# 1 :

## 1.1

EXPLAIN ANALYZE

SELECT \*

FROM title\_basics

WHERE startYear = 2020;

## 1.2

PostgreSQL utilise un Parallel Seq Scan

438 666 lignes

3 737 720 lignes rejetées

222,452 ms

La requête parcourt une grande table sans index.

PostgreSQL estime qu’un scan séquentiel en parallèle sera plus efficace qu’un index scan. C’est justifier ça c’est une grande table sans index

Le nombre de lignes examinées mais rejetées par « WHERE »

## 1.3

CREATE INDEX idx\_title\_basics\_startyear

ON title\_basics(startYear);

## 1.4

Il y a un changement de stratégie avec un Bitmap Index Scan suivis d’un Bitmap Heap Scan. On a un temps d’execution plus long qui est de 339ms

## 1.5

EXPLAIN ANALYZE

SELECT tconst, primaryTitle, startYear

FROM title\_basics

WHERE startYear = 2020;

Oui le temps s’est amélioré il est passé a 164ms.

La stratégie est la même.

Il y a moins de données a lire sur le disque

## 1.6

La même stratégie expliquée dans 1.4.

Oui le temps s’est bien amélioré on est passé de 225ms a 164ms.

Bitmap Index Scan : Utilisation de l’index pour trouver les lignes qui vont satisfaire la condition.

Bitmap Heap Scan : Lecture des données dans la table.

Pour moi l’amélioration semble assez importante.

# 2 :

## 2.1

EXPLAIN ANALYZE

SELECT \*

FROM title\_basics

WHERE titleType = 'movie'

AND startYear = 1950;

## 2.2

Potgres utilise Bitmap Index Scan sur l’index idx\_title\_basics\_startyear.

Il est traité en second.

Primier filtre : 8276

Second filtre : 2011

L’index idx\_title\_basics\_startyear ne couvre que startYear.

## 2.3

CREATE INDEX idx\_title\_basics\_startyear\_titletype

ON title\_basics(startYear, titleType);

## 2.4

Il y a un enorme gains de vitesse on passe de 329ms a 0.75ms, Il a besoin de lire moins de ligne grâce au nouvel index, il en lit 2011 au lieu de 8276 avant

## 2.5

En terme de vitesse il y a a peut pres le même temps d’execution 0.74ms.

Le requête l’air d’etre exactement la même.

## 2.6

On passe de 329ms a 0.74ms.

Il n’y a pas de filtrage a faire suplémentaire tout est gérer grâce a l’index.

Le nombres de lignes a réduit car il n’y a pas un premier passage dur l’année puis sur le titletype car les deux sont géré grâce a l’index.

Dans les requetes avec plusieurs filtres sur des grosses tables.

# 3 :

## 3.1 :

EXPLAIN ANALYZE

SELECT b.primaryTitle, r.averageRating

FROM title\_basics b

JOIN title\_ratings r ON b.tconst = r.tconst

WHERE b.titleType = 'movie'

AND b.startYear = 1994

AND r.averageRating > 8.5;

## 3.2 :

Nested Loop.

Via un index composite sur (startYear, titleType).

Elle est appliquée après avoir récupéré les lignes de title\_ratings.

Car il a la possibilité de répartir le coût du Nested Loop sur plusieurs workers.

## 3.3 :

CREATE INDEX idx\_title\_ratings\_averagerating

ON title\_ratings(averageRating);

## 3.4 :

La vitesse a été amélioré on passe a 8.9ms.

## 3.5 :

Non, l’algorithme de jointure reste un Nested Loop.

Il n’est pas utilisé directement.

Oui il s’est amélioré.

Il y a toujours un scan parallèle.

# 4 :

## 4.1 :

EXPLAIN ANALYZE

SELECT b.startYear,

COUNT(\*) AS nb\_films,

AVG(r.averageRating) AS moyenne\_notes

FROM title\_basics b

JOIN title\_ratings r ON b.tconst = r.tconst

WHERE b.titleType = 'movie'

AND b.startYear BETWEEN 1990 AND 2000

GROUP BY b.startYear

ORDER BY moyenne\_notes DESC;

## 4.2 :

Parallel Seq Scan sur title\_ratings puis Parallel Bitmap Heap Scan sur title\_basics > Parallel Hash Join > Partial GroupAggregate > Gather Merge > Finalize GroupAggregate > sorting

Pour exploiter le parallélisme.

Index utilisé sur title\_basics.

Non car il y n’y a pas beaucoup de lignes a trier.

## 4.3 :

CREATE INDEX idx\_title\_basics\_tconst

ON title\_basics(tconst);

### 4.4 :

Reduction du temps d’exécution a 111ms.

## 4.5 :

Non, les index tconst ne sont pas utilisés dans cette requête.

Le plan est identique car le contexte d'exécution n’a pas changé

C’est utile sur les requetés très sélectives.

# 5 :

## 5.1 :

EXPLAIN ANALYZE

SELECT b.primaryTitle, r.averageRating

FROM title\_basics b

JOIN title\_ratings r ON b.tconst = r.tconst

WHERE b.tconst = 'tt0111161';

## 5.2 :

Nested Loop.

Index Scan using title\_basics\_pkey on title\_basics b et Index Scan using title\_ratings\_pkey on title\_ratings r.

La requete est vraiment tres rapide elle n’a duré que 0.42ms.

Car il n’y apas de tru et de filtre a faire, on utilise un index de la clé primaire.

# 6 :

## 6.1 :

Les index sont les plus efficaces pour les recherches ponctuelles.

Index utiles pour les colonnes à forte cardinalité.

Les index sont très efficaces pour les opérations d’égalité.

## 6.2 :

Nested Loop pour les petits volumes.

Hash Join pour les grands volumes.