TP3 :

# 1

## :

EXPLAIN ANALYZE

SELECT \*

FROM title\_basics

WHERE primaryTitle LIKE 'The%';

## 1.2:

CREATE INDEX idx\_title\_basics\_primarytitle

ON title\_basics(primaryTitle);

## 1.3:

C’est la meme requete qui est faite il n’y a as de différence. Postgres décide de ne pas utiliser l’index car il y a trop de correspondance.

## 1.4 :

Oui

Non

Non

Non

Oui

## 1.5 :

les recherches ordonnées et les opérations d’égalité.

Les wildcards

Pour des recherches précises ou des sort.

## :

EXPLAIN ANALYZE

SELECT \*

FROM title\_basics

WHERE tconst = 'tt0111161';

## :

CREATE INDEX idx\_title\_basics\_tconst\_hash

ON title\_basics USING hash (tconst);

## :

Temps avec B-tree : 0.519s

Temps avec Hash: 0.020s

B-tree : 351mb

Hash : 321mb

Par égalité :

"Index Scan using idx\_title\_basics\_tconst\_hash on title\_basics (cost=0.00..8.02 rows=1 width=86) (actual time=0.011..0.011 rows=1 loops=1)"

" Index Cond: ((tconst)::text = 'tt0111161'::text)"

"Planning Time: 0.045 s"

"Execution Time: 0.019 s"

Par plage:

"Index Scan using title\_basics\_pkey on title\_basics (cost=0.43..8.46 rows=1 width=86) (actual time=0.022..0.030 rows=98 loops=1)"

" Index Cond: (((tconst)::text >= 'tt0111100'::text) AND ((tconst)::text <= 'tt0111199'::text))"

"Planning Time: 0.099 s"

"Execution Time: 0.038 s"

## :

Le Hash est plus rapide que le B-tree pour les recherches par égalité stricte.

Le Hash ne gère pas l’ordre, donc il ne peut pas traiter les plages.

On privilégie le Hash pour des égalités fréquentes sur une colonne à forte cardinalité.

## :

SELECT \*

FROM title\_basics

WHERE startYear = 1994

AND genres LIKE '%Drama%';

## 3.2:

J’ai obtenus une response en 0.137s pour 20949 lignes.

## 3.3 :

J’obtiens mes lignes en 0.094s

## 3.4 :

Maintenant j’obtiens une reponse en 0.75s

## 3.5 :

0.822s

0.058s

0.035s

6.838s

6.307s

## 3.6 :

L’ordre des colonnes détermine si l’index peut être utilisé. Il faut filtrer d’abord sur la première colonne pour qu’il soit exploité.

Un index composite est meilleur si les colonnes sont souvent filtrées ou triées ensemble. Il évite d’avoir à combiner plusieurs index simples.

On place en premier la colonne la plus filtrante ou la plus utilisée dans les requêtes pour maximiser l’efficacité.

# 4

## 4.1 :

2010 168278

2020 105890

2000 78269

1990 45958

1980 44191

1970 40485

1960 31862

1950 23858

1920 22093

1930 20569

1940 14720

1910 12993

1900 184

1890 19

2030 3

1880 1

## 4.2 :

SELECT \*

FROM title\_basics

WHERE titleType = 'movie'

AND startYear BETWEEN 2020 AND 2029;

## 4.4 :

On obtient une réponse en 0.0733s

On obtient une réponse en 0.0272s

Pour la taille :

idx\_title\_basics\_2020s = 736 kB

idx\_title\_basics\_startyear\_full = 4904 kB

## 4.5 :

Un index partiel est plus léger et plus rapide, mais ne sert que pour un sous-ensemble précis.

Il est utile quand les requêtes ciblent toujours les mêmes conditions, comme une période ou un statut.

Il est adapté si la condition est fréquente, stable et réduit clairement le volume de données concerné.

# 5

## 5.1 :

SELECT \*

FROM title\_basics

WHERE primaryTitle ILIKE 'star wars';

## 5.2 :

On a un Parallel Seq Scan qui dure 0.804s avec 3883917 de lignes analysé et 75 retournées.

## 5.3 :

CREATE INDEX idx\_title\_basics\_lower\_primarytitle

ON title\_basics (LOWER(primaryTitle));

## 5.4 :

On passe a 0.7ms de temps d’execution

Ca utilise un Bitmap Heap Scan + Index Scan

## 5.5 :

CREATE INDEX idx\_title\_basics\_startyear\_text

ON title\_basics ((startYear::text));

On obtiens la reéponse en 331ms

## 5.6 :

L'expression doit être identique pour que PostgreSQL reconnaisse et utilise l’index.

Les index d'expressions ralentissent légèrement les insertions et mises à jour, car l'expression doit être recalculée.

Les transformations fréquentes sont LOWER(), COALESCE(), EXTRACT(), LEFT(), et des calculs arithmétiques.

# 6

## 6.1 :

SELECT primaryTitle, startYear

FROM title\_basics

WHERE genres = 'Drama';

## 6.2 :

CREATE INDEX idx\_title\_basics\_genres

ON title\_basics(genres);

PostgreSQL a utilisé un Bitmap Index Scan sur genres, suivi d’un Bitmap Heap Scan, car les colonnes primaryTitle et startYear ne sont pas dans l’index.

## 6.3 :

CREATE INDEX idx\_title\_basics\_genres\_covering

ON title\_basics(genres)

INCLUDE (primaryTitle, startYear);