**TP #04 : Techniques de requêtage optimisées**

**Exercice 1.1: Analyse des films populaires et bien notés**

**1. Requête initiale**

SELECT

b.primary\_title,

r.averagerating,

b.genres

FROM

title\_basics b

JOIN

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

WHERE

b.title\_type = 'movie'

AND b.start\_year >= 2000

AND r.numvotes > 10000

ORDER BY

r.averagerating DESC

LIMIT 20;

**2. Exécutez avec EXPLAIN ANALYZE et notez les types de jointures utilisés**

avec EXPLAIN ANALYZE on voit que les je vois que PostgreSQL utilise une jointure en boucle imbriquée (Nested Loop) entre les tables “title\_ratings” et “title\_basics”. La table “title\_ratings” est lue avec un scan séquentiel parallèle.

**3. Ajoutez des index pertinents et mesurez l'amélioration**

Nous avons ajouté les index suivant :

CREATE INDEX idx\_ratings\_numvotes ON title\_ratings(numvotes);

CREATE INDEX idx\_basics\_start\_year ON title\_basics(start\_year);

CREATE INDEX idx\_basics\_title\_type ON title\_basics(title\_type);

On constate une nette amélioration des performances. Le temps d'exécution est passé d’environ 327 ms à 203 ms. Le plan d’exécution montre également que PostgreSQL utilise désormais un Bitmap Index Scan sur numvotes, ce qui permet de filtrer plus efficacement les films populaires.

**Exercice 1.2: Acteurs et réalisateurs à succès**

**1. La requête**

SELECT

nb.primaryname,

tb.primary\_title,

tr.averagerating

FROM

title\_principals tp

JOIN

name\_basics nb ON tp.nconst = nb.nconst

JOIN

title\_basics tb ON tp.tconst = tb.tconst

JOIN

title\_ratings tr ON tb.tconst = tr.tconst

WHERE

tp.category IN ('actor', 'actress')

AND tr.averagerating > 8.0

GROUP BY

nb.primaryname, tb.primary\_title, tr.averagerating, tp.nconst

HAVING

COUNT(tp.tconst) >= 3

**2. Requête optimisée**

SELECT

nb.primaryname,

tb.primary\_title,

tr.averagerating

FROM

title\_ratings tr

JOIN

title\_basics tb ON tr.tconst = tb.tconst

JOIN

title\_principals tp ON tb.tconst = tp.tconst

JOIN

name\_basics nb ON tp.nconst = nb.nconst

WHERE

tp.category IN ('actor', 'actress')

AND tr.averagerating > 8.0

GROUP BY

nb.primaryname, tb.primary\_title, tr.averagerating, tp.nconst

HAVING

COUNT(tp.tconst) >= 3

**3. Comparaison des plans d'exécution avant/après optimisation**

Avant optimisation, la requête commence par la table la plus volumineuse, ce qui génère un traitement de 47 secondes. Après optimisation, en démarrant par title\_ratings filtrée sur les notes > 8.0, le volume de données est réduit, ce qui accélère l’exécution à environ 30 secondes. Le changement d’ordre des jointures améliore donc nettement les performances.

**Partie 2: Sous-requêtes et CTE efficaces**

**Exercice 2.1: Transformation de sous-requêtes**

**1. La requête :**

SELECT tb.primary\_title, tb.start\_year, tr.averagerating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

WHERE tr.averagerating > (

SELECT AVG(r.averagerating)

FROM title\_basics b

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

WHERE b.genres LIKE '%' || tb.genres || '%'

)

ORDER BY tr.averagerating DESC

LIMIT 100;

La requête utilise une sous-requête corrélée dans la clause WHERE. Pour chaque ligne de title\_basics (alias tb), la sous-requête calcule la moyenne des notes (AVG(r.averagerating)) des films appartenant aux mêmes genres. La sous-requête est exécutée une fois par ligne de la table extérieure (tb), ce qui peut entraîner un nombre très élevé d’exécutions, surtout si la table est grande. Cela cause une consommation importante de ressources CPU et mémoire, ralentissant considérablement la requête.

**2. Requête optimisée**

WITH genre\_avg AS (

SELECT

unnest(string\_to\_array(genres, ',')) AS genre,

AVG(tr.averagerating) AS avg\_rating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

GROUP BY genre

)

SELECT

tb.primary\_title,

tb.start\_year,

tr.averagerating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

JOIN LATERAL unnest(string\_to\_array(tb.genres, ',')) AS g(genre) ON true

JOIN genre\_avg ga ON g.genre = ga.genre

WHERE tr.averagerating > ga.avg\_rating

ORDER BY tr.averagerating DESC

LIMIT 100;

**3. Comparaison des plan d'exécution**

La première requête est lente parce qu’elle calcule la moyenne des notes pour chaque film dans une sous-requête corrélée, ce qui fait beaucoup de calculs répétitifs. Du coup, elle prend énormément de temps.

La version optimisée calcule la moyenne par genre une seule fois grâce à une CTE, puis fait une jointure avec les films. Ça évite de recalculer sans arrêt, ce qui rend la requête beaucoup plus rapide (environ 7 secondes au lieu de plus de 20 minutes).

**Exercice 2.2: Utilisation de CTE**

WITH genre\_avg AS (

SELECT

unnest(string\_to\_array(tb.genres, ',')) AS genre,

AVG(tr.averagerating) AS avg\_rating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

GROUP BY genre

)

SELECT

tb.primary\_title,

tb.start\_year,

tr.averagerating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

JOIN LATERAL unnest(string\_to\_array(tb.genres, ',')) AS g(genre) ON true

JOIN genre\_avg ga ON g.genre = ga.genre

WHERE tr.averagerating > ga.avg\_rating

ORDER BY tr.averagerating DESC

LIMIT 100;

**2. Film exceptionnels**

WITH genre\_avg AS (

SELECT

unnest(string\_to\_array(tb.genres, ',')) AS genre,

AVG(tr.averagerating) AS avg\_rating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

GROUP BY genre

)

SELECT

tb.primary\_title,

tb.start\_year,

tr.averagerating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

JOIN LATERAL unnest(string\_to\_array(tb.genres, ',')) AS g(genre) ON true

JOIN genre\_avg ga ON g.genre = ga.genre

WHERE tr.averagerating >= ga.avg\_rating + 2

ORDER BY tr.averagerating DESC

LIMIT 100;

**3. Films exceptionnels avec leurs acteurs principaux**

WITH genre\_avg AS (

SELECT

unnest(string\_to\_array(tb.genres, ',')) AS genre,

AVG(tr.averagerating) AS avg\_rating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

GROUP BY genre

),

exceptional\_films AS (

SELECT DISTINCT

tb.tconst,

tb.primary\_title,

tb.start\_year,

tr.averagerating

FROM title\_basics tb

JOIN title\_ratings tr ON tb.tconst = tr.tconst

JOIN LATERAL unnest(string\_to\_array(tb.genres, ',')) AS g(genre) ON true

JOIN genre\_avg ga ON g.genre = ga.genre

WHERE tr.averagerating >= ga.avg\_rating + 2

)

SELECT

ef.primary\_title,

ef.start\_year,

ef.averagerating,

nb.primaryname AS actor\_name,

tp.category

FROM exceptional\_films ef

JOIN title\_principals tp ON ef.tconst = tp.tconst

JOIN name\_basics nb ON tp.nconst = nb.nconst

WHERE tp.category IN ('actor', 'actress')

ORDER BY ef.averagerating DESC, ef.primary\_title, actor\_name

LIMIT 100;

**Partie 3: Window Functions et agrégations avancées**

**Exercice 3.1: Analyse de carrière avec Window Functions**

**1. Films par ordre chronologique**

SELECT

tb.primary\_title,

tb.start\_year,

tr.averagerating,

ROW\_NUMBER() OVER (ORDER BY tb.start\_year) AS film\_order

FROM title\_principals tp

JOIN name\_basics nb ON tp.nconst = nb.nconst

JOIN title\_basics tb ON tp.tconst = tb.tconst

LEFT JOIN title\_ratings tr ON tb.tconst = tr.tconst

WHERE nb.primaryname = 'Tom Hanks'

AND tp.category IN ('actor', 'actress')

ORDER BY tb.start\_year;