TP #02 : Optimisation des requêtes avec les index

Prérequis

- PostgreSQL installé et accessible
- Base de données IMDb importée
- Les tables principales utilisées sont: title_basics, title_ratings

Exercice 1: Analyse d'une requête simple

1.1 Première analyse

Écrivez une requête qui utilise EXPLAIN ANALYZE pour afficher tous les titres de l'année 2020 dans la table title_basics.

1.2 Analyse du plan d'exécution

Observez les éléments suivants dans le résultat:

- Quelle stratégie PostgreSQL utilise-t-il? (Seq Scan, Index Scan, etc.)
- Combien de lignes sont retournées?
- Combien de lignes sont examinées puis rejetées?
- Quel est le temps d'exécution total?

Répondez aux questions:

1. Pourquoi PostgreSQL utilise-t-il un Parallel Sequential Scan?
2. La parallélisation est-elle justifiée ici? Pourquoi?
3. Que représente la valeur "Rows Removed by Filter"?

1.3 Création d'index

Créez un index simple sur la colonne start_Year de la table title_basics.

1.4 Analyse après indexation

Exécutez à nouveau la même requête que dans l'étape 1.1 et comparez les résultats.

1.5 Impact du nombre de colonnes

Modifiez votre requête pour sélectionner uniquement les colonnes tconst, primary_Title et start_Year (au lieu de toutes les colonnes avec *).
Exécutez cette nouvelle requête avec EXPLAIN ANALYZE et comparez les résultats avec ceux de l'étape 1.4.

1. Le temps d'exécution a-t-il changé? Pourquoi?
2. Le plan d'exécution est-il différent?
3. Pourquoi la sélection de moins de colonnes peut-elle améliorer les performances?

1.6 Analyse de l'impact global

Comparez avec le plan de l'étape 1.1:

1. Quelle nouvelle stratégie PostgreSQL utilise-t-il maintenant?
2. Le temps d'exécution s'est-il amélioré? De combien?
3. Que signifie "Bitmap Heap Scan" et "Bitmap Index Scan"?

4. Pourquoi l'amélioration n'est-elle pas plus importante?

Exercice 2: Requêtes avec filtres multiples

2.1 Requête avec conditions multiples

Écrivez une requête avec EXPLAIN ANALYZE qui recherche tous les films (title_Type = 'movie') sortis en 1950.

2.2 Analyse du plan d'exécution

Quelle stratégie est utilisée pour le filtre sur start_Year?

1. Comment est traité le filtre sur title_Type?
2. Combien de lignes passent le premier filtre, puis le second?
3. Quelles sont les limitations de notre index actuel?

2.3 Index composite

Créez un index composite qui combine les colonnes start_Year et title_Type.

2.4 Analyse après index composite

Exécutez à nouveau la même requête qu'à l'étape 2.1 et observez les différences.

2.5 Impact du nombre de colonnes

Modifiez votre requête pour sélectionner uniquement les colonnes tconst, primary_Title, start_Year et title_Type.
Comparez les performances avec la requête de l'étape 2.4:

1. Le temps d'exécution a-t-il changé?
2. Pourquoi cette optimisation est-elle plus ou moins efficace que dans l'exercice 1?
3. Dans quel cas un "covering index" (index qui contient toutes les colonnes de la requête) serait idéal?

2.6 Analyse de l'amélioration globale

1. Quelle est la différence de temps d'exécution par rapport à l'étape 2.1?
2. Comment l'index composite modifie-t-il la stratégie?
3. Pourquoi le nombre de blocs lus ("Heap Blocks") a-t-il diminué?
4. Dans quels cas un index composite est-il particulièrement efficace?

Exercice 3: Jointures et filtres

3.1 Jointure avec filtres

Écrivez une requête avec EXPLAIN ANALYZE qui joint les tables title_basics et title_ratings pour trouver le titre et la note des films sortis en 1994 ayant une note moyenne supérieure à 8.5.

3.2 Analyse du plan de jointure

1. Quel algorithme de jointure est utilisé?
2. Comment l'index sur start_Year est-il utilisé?
3. Comment est traitée la condition sur average_Rating?

4. Pourquoi PostgreSQL utilise-t-il le parallélisme?

3.3 Indexation de la seconde condition

Créez un index sur la colonne average_Rating de la table title_ratings.

3.4 Analyse après indexation

Exécutez à nouveau la requête de l'étape 3.1 et observez les différences.

3.5 Analyse de l'impact

1. L'algorithme de jointure a-t-il changé?
2. Comment l'index sur average_Rating est-il utilisé?
3. Le temps d'exécution s'est-il amélioré? Pourquoi?
4. Pourquoi PostgreSQL abandonne-t-il le parallélisme?

Exercice 4: Agrégation et tri

4.1 Requête complexe

Écrivez une requête avec EXPLAIN ANALYZE qui calcule, pour chaque année entre 1990 et 2000, le nombre de films et leur note moyenne, puis trie les résultats par note moyenne décroissante.

4.2 Analyse du plan complexe

1. Identifiez les différentes étapes du plan (scan, hash, agrégation, tri)
2. Pourquoi l'agrégation est-elle réalisée en deux phases ("Partial" puis "Finalize")?
3. Comment sont utilisés les index existants?
4. Le tri final est-il coûteux? Pourquoi?

4.3 Indexation des colonnes de jointure

Créez des index sur les colonnes tconst des deux tables (title_basics et title_ratings).

4.4 Analyse après indexation

Exécutez à nouveau la requête de l'étape 4.1 et observez les résultats.

4.5 Analyse des résultats

1. Les index de jointure sont-ils utilisés? Pourquoi?
2. Pourquoi le plan d'exécution reste-t-il pratiquement identique?
3. Dans quels cas les index de jointure seraient-ils plus efficaces?

Exercice 5: Recherche ponctuelle

5.1 Requête de recherche par identifiant

Écrivez une requête avec EXPLAIN ANALYZE qui recherche le titre et la note d'un film spécifique en utilisant son identifiant (tconst = 'tt0111161').

5.2 Analyse du plan

1. Quel algorithme de jointure est utilisé cette fois?
2. Comment les index sur tconst sont-ils utilisés?
3. Comparez le temps d'exécution avec les requêtes précédentes
4. Pourquoi cette requête est-elle si rapide?

Exercice 6: Synthèse et réflexion

Répondez aux questions suivantes:

1. Quand un index est-il le plus efficace?
    - Pour les recherches ponctuelles ou volumineuses?
    - Pour les colonnes avec forte ou faible cardinalité?
    - Pour les opérations de type égalité ou intervalle?

2. Quels algorithmes de jointure avez-vous observés?
    - Dans quels contextes chacun est-il préféré?
    - Comment le volume de données influence-t-il le choix?

3. Quand le parallélisme est-il activé?
    - Quels types d'opérations en bénéficient le plus?
    - Pourquoi n'est-il pas toujours utilisé?

4. Quels types d'index utiliser dans les cas suivants:
    - Recherche exacte sur une colonne
    - Filtrage sur plusieurs colonnes combinées
    - Tri fréquent sur une colonne
    - Jointures fréquentes entre tables