Exercice 1 – Utilisation d'un index B-tree

1. **Étude initiale (sans index)**

Lancez :

```
EXPLAIN ANALYZE
SELECT *
  FROM title_basics
 WHERE primary_title LIKE 'The%';
```

2. **Mise en place de l'index**

   ```
   CREATE INDEX idx_title_basics ON title_basics(primary_title);
   ```

3. **Observation post-indexation**

   - Relancez la même requête `LIKE 'The%'` et comparez les résultats.
   - Vous constaterez qu'un B-tree n'accélère pas les recherches avec un motif préfixe (LIKE avec wildcard en fin).

4. **Expérimentations ciblées**
   Pour chacune des requêtes suivantes, exécutez `EXPLAIN ANALYZE` et mentionnez si l'index est employé :

Égalité stricte

```
SELECT *
  FROM title_basics
 WHERE primary_title = 'The Matrix';
```

   -

Recherche de préfixe

```
SELECT *
  FROM title_basics
 WHERE primary_title LIKE 'The%';
```

   -

Recherche de suffixe

```
SELECT *
  FROM title_basics
 WHERE primary_title LIKE '%The';
```

Recherche de sous-chaîne

```
SELECT *
  FROM title_basics
 WHERE primary_title LIKE '%The%';
```

Tri par titre

```
SELECT *
  FROM title_basics
 ORDER BY primary_title
 LIMIT 100;
```

5. **Synthèse**
    - Le B-tree est efficace pour :
        - les comparaisons d'égalité (=)
        - les bornes (<, >, `BETWEEN`)
        - le tri (`ORDER BY`)
    - Il n'est pas utilisé lorsque PostgreSQL estime qu'un scan séquentiel est plus rapide ou quand l'opération ne peut pas exploiter l'ordre (ex. LIKE '%…').
    - Ce type d'index convient surtout aux colonnes souvent filtrées par égalité ou comparaisons, ainsi qu'aux colonnes utilisées pour le tri (identifiants, dates, etc.).

Exercice 2 – Index de type Hash

Requête de base

```
EXPLAIN ANALYZE
SELECT *
  FROM title_basics
 WHERE tconst = 'tt0111161';
```

1. → Notez le temps d'exécution sans index Hash.

Création de l'index Hash

```
CREATE INDEX idx_tconst_hash
  ON title_basics
 USING hash (tconst);
```

2. **Comparaison avec un B-tree**
    - Créez aussi un `B-tree` sur `tconst`.
    - Mesurez les temps d'exécution pour la même requête avec chacun des deux index (Hash vs. B-tree).

Comparez ensuite la taille sur disque :

```
SELECT pg_indexes_size('idx_tconst_hash'),
       pg_indexes_size('idx_tconst_btree');
```

Recherche par plage

```
EXPLAIN ANALYZE
SELECT *
  FROM title_basics
 WHERE tconst BETWEEN 'tt0111160' AND 'tt0111170';
```

3. – Évaluez le plan d'exécution en présence de chaque index.

Exercice 3 – Index composés

Requête à deux critères

```
SELECT *
  FROM title_basics
 WHERE genres     = 'Drama'
   AND start_year = 1994;
```

1. → Observez les performances sans index.

2. **Index mono-colonne**

    - Créez un index sur `genres`, puis un autre sur `start_year`.

    - Comparez l'impact sur les temps de réponse.

3. **Index combiné**

    - Créez un index composite (`genres, start_year`) et mesurez les gains.

    - Recommencez avec l'ordre inversé (`start_year, genres`).

4. **Cas pratiques**
   Pour chaque index, testez :

    - Filtre par genre seul

    - Filtre par année seule

    - Filtre combiné

    - Tri par genre puis année

    - Tri par année puis genre

5. **Questions réflexives**

    - En quoi l'ordre des colonnes dans l'index composite influence-t-il son exploitation ?

    - Quels scénarios privilégient un index multi-colonne plutôt que plusieurs index mono-colonne ?

    - Selon quels critères choisit-on l'ordre optimal des colonnes dans un index composite ?