

Exercice Scientifique et technique

Comment accélérer les requêtes SQL sous PostgreSQL ?

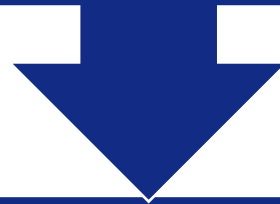
Aoucher Anaïs

Sommaire

- Indexation
 - BTREE
 - BRIN
 - HASH
 - GIN
- Vue matérialisée
- Partitionnement

Indexation

L'indexation est la création de structures sur des colonnes pour permettre à la base de retrouver plus rapidement les lignes qui correspondent à une condition



Plusieurs types d'index, mais ici on va en étudier 4 principaux:

BTREE

BRIN

GIN

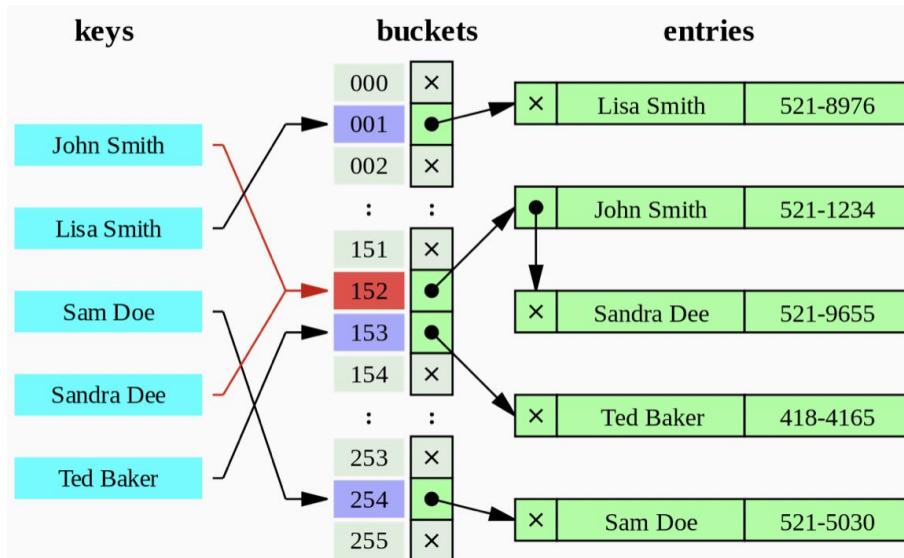
HASH

HASH et BTREE

HASH

Structure d'indexation qui applique une fonction de hachage à la valeur indexée pour la ranger dans un seau de hash

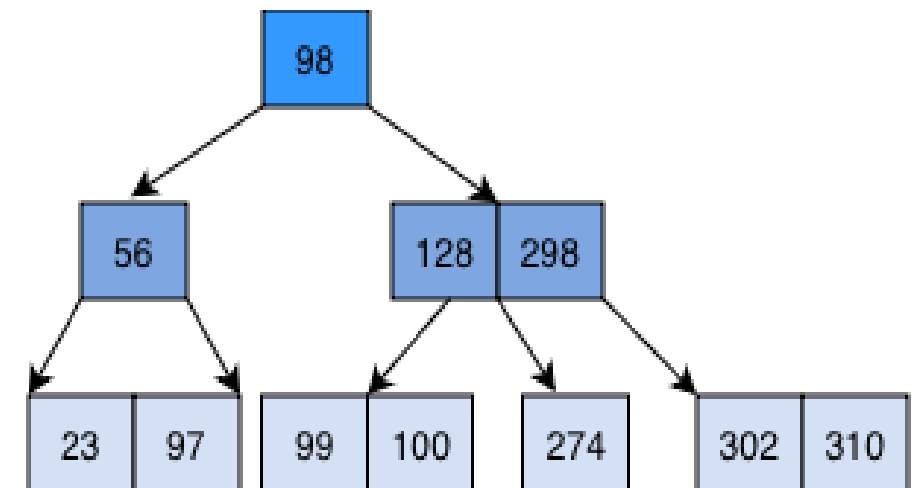
- Avantages :
 - Très efficace sur les égalités
- Inconvénient :
 - Pas performant sur les autres cas
 - Index peut possiblement être lourd



BTREE

Structure arborescente équilibrée et ordonnée

- Avantages : efficace sur des égalités, tris et plage de valeurs
- Inconvénient : inefficace sur des très grandes tables pas triées

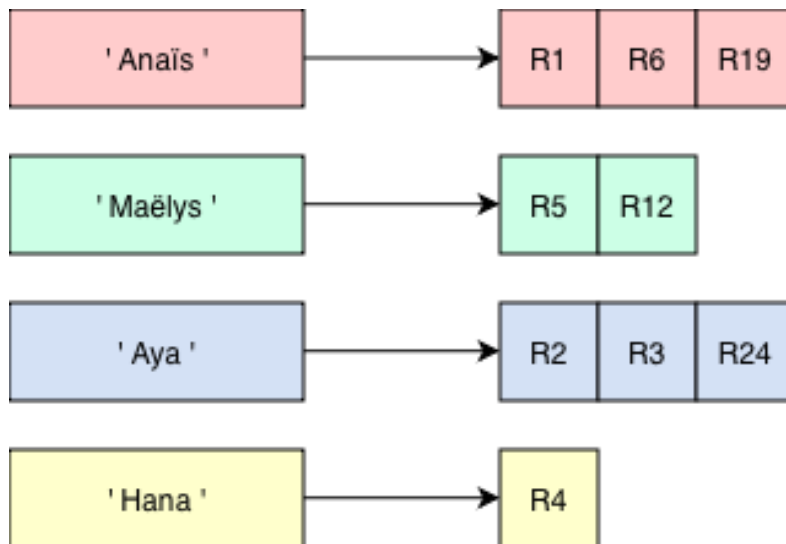


GIN et BRIN

GIN

Un index inversé, pour chaque élément contenu dans une valeur il mémorise la liste des lignes où cet élément apparaît

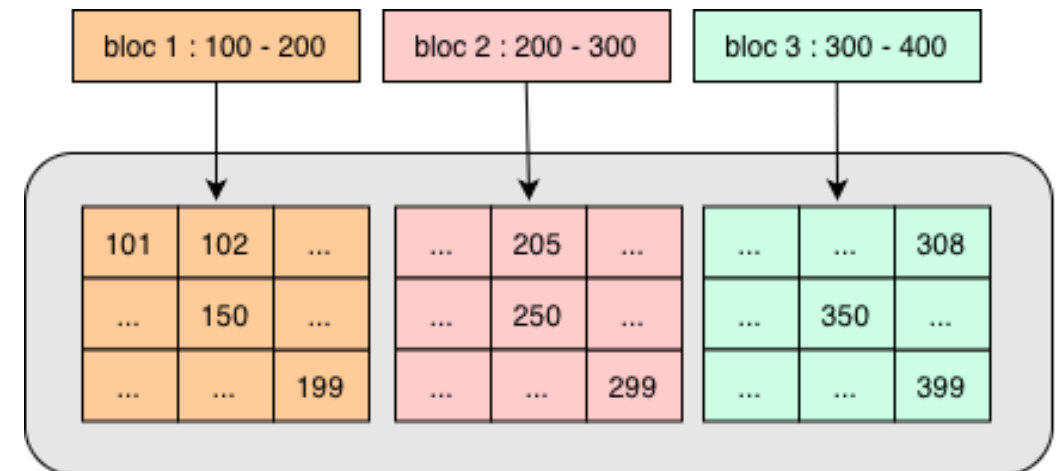
- Avantages :
 - Performant pour la recherche textuelle
- Inconvénient :
 - Index lourd
 - Peu performant sur les égalités ou tris



BRIN

Structure organisée en intervalle (min, max), où chaque intervalle est associé à un bloc de données

- Avantages :
 - Index léger
 - Performant sur de grands volumes de données triés
- Inconvénient :
 - Si table non triée alors peut être pas performant
 - Pas adapté à tous les cas



Expérience - Base de données – schéma

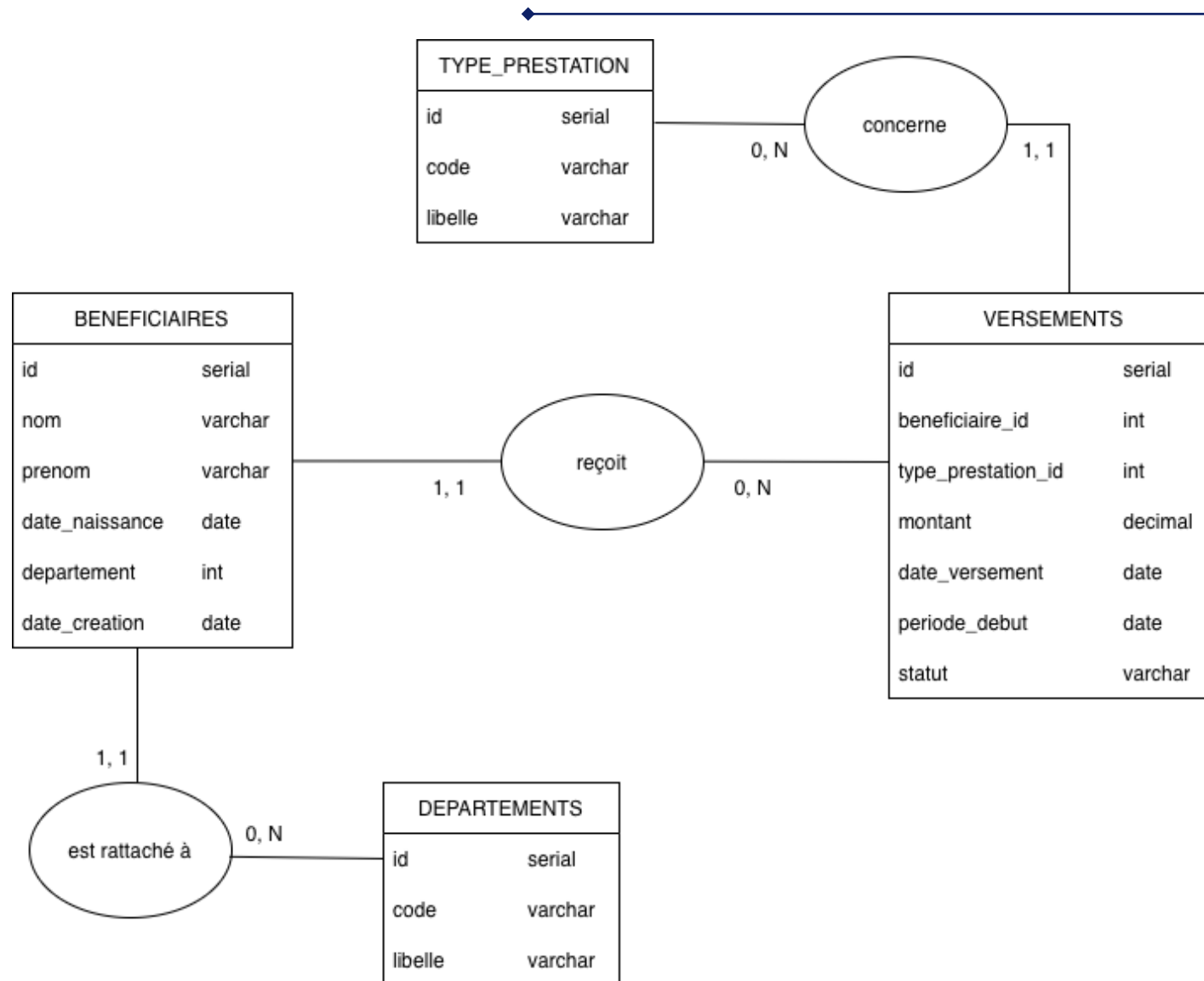


table	count
departements	20
types_prestation	7
beneficiaires	100000
versements	3000000

(4 rows)

Temps d'exécution des requêtes (en ms)

1^{ère} requête

```
SELECT COUNT(*), SUM(v.montant)
FROM versements v
WHERE v.beneficiaire_id = (
    SELECT beneficiaire_id
    FROM versements
    LIMIT 1
)
```

- Egalité
- table versements lourde

	requête 1
sans index	334,844
btree	9,471
brin	120,215
gin	337,198
hash	10,199

- Btree et hash ~ 35x plus rapide que sans index
- Brin ~ 3 fois plus rapide que sans index
- Gin pas de différences

2^{ème} requête

```
SELECT COUNT(*), SUM(v.montant)
FROM versements v
WHERE v.date_versement >= '2023-06-01'
      AND v.date_versement < '2023-07-01'
```

- Filtre par intervalle

	requête 2
sans index	101,370
btree	134,070
brin	102,241
gin	114,646
hash	418,996

- Sans index ~ brin et gin
- Btree pas plus performant mais pas inefficace non plus
- Hash beaucoup plus lent

Temps d'exécution des requêtes (en ms)

3ème requête

```
SELECT id, date_versement, montant
FROM versements
ORDER BY date_versement ASC
LIMIT 10
```

- Tri
- table versements lourde

	requête 3
sans index	185,667
btree	0,957
brin	197,272
gin	193,610
hash	185,219

- Btree beaucoup plus efficace que sans index sur un tri
- Sans index ~ brin ~ gin ~ hash

4ème requête

```
SELECT COUNT(*)
FROM beneficiaires b
WHERE b.nom LIKE '%martin%'
```

- Recherche textuelle

	requête 4
sans index	88,557
btree	70,852
brin	89,021
gin	10,688
hash	71,752

- Gin ~ 8x plus rapide que sans index
- Autres index : pas de différence flagrante

Pour quelle volumétrie un index est-il utile ?

pour plusieurs tailles de table (30, 300, 3k, 30k, 300k, 3M lignes), on exécute la même requête d'égalité
-- « WHERE id = ... d'abord sans index puis avec un index, et on compare les temps mesurés.

```
SELECT COUNT(*)  
FROM versements_test  
WHERE beneficiaire_id = (SELECT beneficiaire_id FROM versements_test LIMIT 1);
```

volume de données	sans index	avec index
30	1,416	0,15
300	0,152	0,134
3000	0,265	0,137
30000	1,59	0,208
300000	14,473	0,31
3000000	2161,988	0,432

un index dans ce cas devient utile à partir
de 300 000 lignes

Poids des indexes

table	colonne	btree	brin	gin	hash
versements	date_versement	20 MB	24 kB	-	136 MB
versements	periode_debut	20 MB	24 kB	-	137 MB
versements	beneficiaire_id	22 MB	24 kB	46 MB	96 MB
versements	statut	25 MB	32 kB	74 MB	138 MB
beneficiaires	departement_id	704 kB	24 kB	416 kB	6024 kB
beneficiaires	nom	704 kB	24 kB	1456 kB	5960 kB
versements	statut + beneficiaire	25 MB	32 kB	74 MB	-

- Brin beaucoup plus léger que tous les autres indexes
- Btree est assez raisonnable
- Gin peut être lourd sur des indexes composite ou sur des grosses tables
- Hash assez lourd

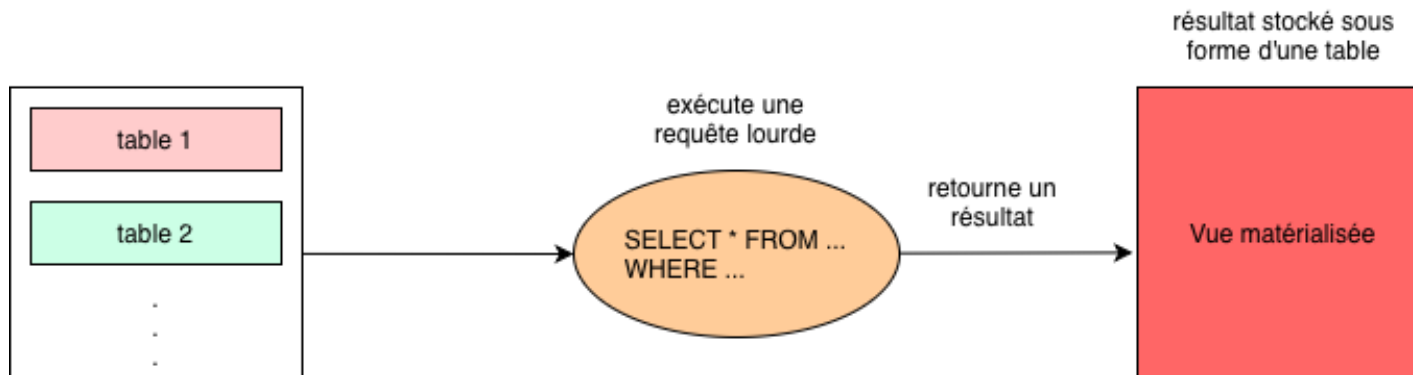
Conclusion

- Btree plus adapté aux égalités et aux tris (index par défaut dans postgres)
- Brin efficace sur des grosses tables et des colonnes ordonnées
- Hash adapté aux égalités mais pas plus efficace que btree
- Gin adapté à la recherche textuelle

Vues matérialisées

Principe

Précalculer le résultat d'une requête lourde, le stocker comme une table, puis relire ce résultat déjà prêt au lieu de recalculer à chaque fois



Avantages

- Lecture très rapide pour des requêtes lourdes qui sont souvent exécutées
- Peut être indexée comme une table normale

Inconvénient

- Pas à jour en temps réel

Expérience

```
SELECT
  date_trunc('month', v.date_versement) AS mois,
  d.code                               AS departement,
  v.statut,
  COUNT(*)                            AS nb_versements,
  SUM(v.montant)                       AS total_montant
FROM versements v
JOIN beneficiaires b ON b.id = v.beneficiaire_id
JOIN departements d ON d.id = b.departement_id
GROUP BY mois, d.code, v.statut
ORDER BY mois, departement, statut;
```

Requête 1

Temps d'exécution (en ms)

Sans VM	Avec VM	Avec VM + index
2367,769	14,294	6,788

- Sans VM : doit relire la table en entier et refait chaque calcul à chaque exécution
- Avec VM : lecture d'un résultat précalculé => 160x plus rapide que sans VM
- Avec VM + index : index accélère encore plus la requête => 350x plus rapide que sans VM

```
SELECT
  mois,
  departement,
  statut,
  nb_versements,
  total_montant
FROM stats_mensuelles
ORDER BY mois, departement, statut;
```

Requête 2

Poids des tables

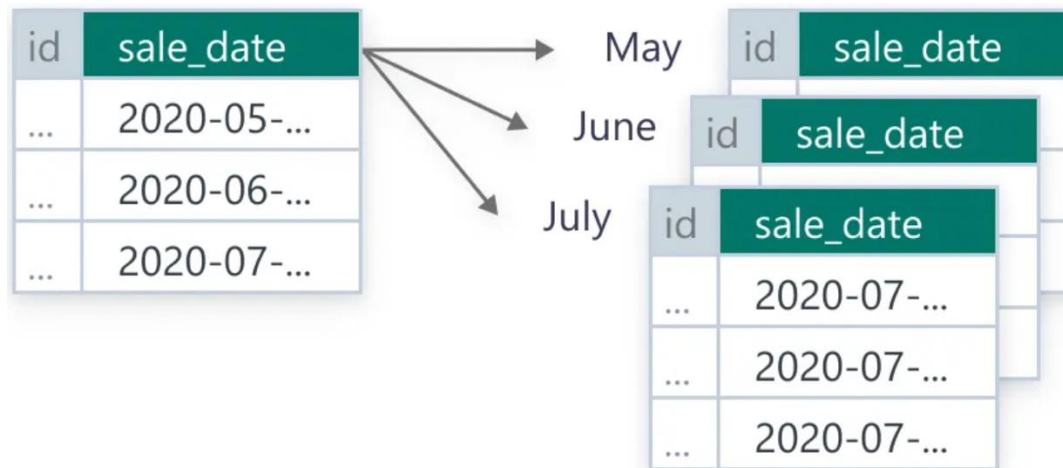
versements	stats_mensuelles
283 MB	584 kB

- Table stats_mensuelles beaucoup plus légère que versement car moins de ligne

Partitionnement

Principe

Découpage d'une grande table en plusieurs sous-tables (partitions), chacune contenant un morceau des lignes selon un critère



Avantages

- Moins de données lues
- Meilleure gestion des très grosses tables

Inconvénients

- Bénéfice limité pour petites/moyennes tables

Source : <https://engineering.workable.com/postgres-live-partitioning-of-existing-tables-15a99c16b291>

Expérience

1

```
SELECT COUNT(*), SUM(v.montant)
FROM versements v
WHERE v.date_versement >= '2023-06-01'
AND v.date_versement < '2023-07-01';
```

```
SELECT COUNT(*), SUM(v.montant)
FROM versements_partitionnes v
WHERE v.date_versement >= '2023-06-01'
AND v.date_versement < '2023-07-01';
```

2

```
SELECT date_trunc('month', v.date_versement) AS mois,
       COUNT(*) AS nb,
       SUM(v.montant) AS total
FROM versements v
WHERE v.date_versement >= '2020-01-01'
AND v.date_versement < '2024-01-01'
GROUP BY date_trunc('month', v.date_versement)
ORDER BY mois;
```

```
SELECT date_trunc('month', v.date_versement) AS mois,
       COUNT(*) AS nb,
       SUM(v.montant) AS total
FROM versements_partitionnes v
WHERE v.date_versement >= '2020-01-01'
AND v.date_versement < '2024-01-01'
GROUP BY date_trunc('month', v.date_versement)
ORDER BY mois;
```

Temps d'exécution (en ms)

	requête 1	requête 2
sans partition	459,982	809,403
avec partition	90,834	537,024

- 1ere requête (type filtre) : 5x plus rapide avec partition
- 2^{ème} requête (plus large): 1,5x plus rapide avec partition

Poids des partitions

versements	versements_a_p artir_2022	versements_ava nt_2022
283 MB	132 MB	88 MB

- Le poids partitions est du même ordre de grandeur que la table d'origine

Conclusion

Indexation

Taille des données

Partitionnement

Qualité des requêtes

Vue matérialisée

Puissance de matériel

Bibliographie

Indexation

- <https://docs.postgresql.fr/9.6/indexes-types.html>
- <https://postgrespro.com/blog/pgsql/4161516>
- <https://www.crunchydata.com/blog/postgres-indexing-when-does-brin-win>

Vue matérialisée

- <https://www.navicat.fr/company/aboutus/blog/2732-introduction-aux-vues-matérialisées-postgresql>
- <https://docs.postgresql.fr/9.6/sql-creatematerializedview.html>

Partitionnement

- <https://docs.postgresql.fr/12/ddl-partitioning.html>
- <https://engineering.workable.com/postgres-live-partitioning-of-existing-tables-15a99c16b291>

Général

- <https://www.youtube.com/watch?v=TAJKNBPv4Wc>