Indexer les données non-scalaires

Un bref aperçu… :)

Marc Cousin

Scalaire ??

Algèbre linéaire (vecteurs…maths…)

- «Qu'on peut mettre sur une échelle»
- Par opposition aux vecteurs et autres créatures à plusieurs dimensions, machins pas triables, etc

==> C'est des données à plusieurs dimensions qu'on va parler !

Plusieurs dimensions

- Ranges (une dimension, deux valeurs)
- Tableaux (n dimensions, p valeurs)
- Données géométriques (2, 3, 4 dimensions la plupart du temps). Indexation par la Bounding Box
- Hiérarchies (position dans l'arbre)
- Etc…

Indexam (Access Method)

- Btree: que des scalaires:
  - [1,2,8,3] > [2,3,1,5] ? C'est mort !
- GIN: Generalized Inverted Index
- GiST: Generalized Search Tree
- SP-Gist: Space-Partitioned Gist
- Bloom: Bloom filter (9.6!)
- Vodka, Etc…

Dalibo embauche… :)

Index

- Btree
  - [1,2,8,3] > [2,3,1,5] ? C'est mort !
- GIN: Generalized Inverted Index
- GiST: Generalized Search Tree
- SP-Gist: Space-Partitioned Gist
- Bloom: Bloom filter (9.6!)
- Vodka, Etc…

Opérateurs

- Pour utiliser un index GIN, opérateurs spécifiques (contenance @> la plupart du temps)

GIN

- Index inversé. Pour chaque valeur rencontrée, on crée la liste des endroits où on peut le trouver
- Un dessin !

Directement tiré de la doc de
la concurrence
(Elastic Search)

1: Winter is coming.
2: Ours is the fury.
3: The choice is yours.

term	freq	documents
choice	1	3
coming	1	1
fury	1	2
is	3	1, 2, 3
ours	1	2
the	2	2, 3
winter	1	1
yours	1	3

Dictionary

Postings

Types supportés

- À peu près tout:
  - Tableaux
  - Ranges
  - Full Text Search
  - HStore
  - JSONB
  - varbit
  - …
  - `select * from pg_opclass where opcmethod in (select oid from pg_am where amname='gin');`

JSONB ?

- Tout le monde ne parle que de ça
- C'est super cool
- Il y en a partout sur le net…
- Voyons autre chose

Indexer un tableau

- On peut indexer simplement tous les tableaux de types communs
- Stocker des listes dans un attribut (tant qu'on se moque des formes normales)
- On n'a pas toujours le choix du schéma
- Il y a bien plus sale qu'un tableau (on y vient :))

Tableau simple (liste d'attributs)

530	{1,12,127,3}
531	{2,9,1}

- 100 millions d'enregistrements. Entre 1 et 10 enregistrements dans la liste (aléatoire), valeurs entre 1 et 100. 24Go

```
CREATE INDEX idx_gin
ON compteurs USING GIN (attributes);
```

- Atteeeeeendre… l'index fait 28Go.

- Cette présentation a permis de trouver une fuite mémoire dans la construction de l'index en 9.5!

Opérateurs

- Pour les index GIN, opérateurs spécialisés suivant le type de données (contenance par exemple)

```
tp=# explain analyze select * from compteurs where attributes @> '{1,21}';
                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on compteurs  (cost=79.37..9899.55 rows=2500 width=224) (actual time=3.746..3.746 rows=0 loops=1)
   Recheck Cond: (attributes @> '{1,21}'::integer[])
   ->  Bitmap Index Scan on idx_gin  (cost=0.00..78.75 rows=2500 width=0) (actual time=3.743..3.743 rows=0 loops=1)
         Index Cond: (attributes @> '{1,21}'::integer[])
 Planning time: 0.138 ms
 Execution time: 3.800 ms
```

Indexer du moche

immat	modele	caracteristiques
RX-048-YN	kangoo	climatisation,jantes aluminium,toit ouvrant,regulateur de vitesse
MN-987-RP	megane	abs,boite automatique,toit ouvrant

C'est laid, mais on en rencontre…

Ce qui est fait habituellement

```
SELECT * FROM voitures
WHERE caracteristiques LIKE '%climatisation%'
  AND caracteristiques LIKE '%toit ouvrant%'
```

Full Scan, ou index sur trigramme. Inefficace (200ms) pour 100 000 enregistrements

On peut aussi l'écrire avec des expressions régulières, pour être plus rigoureux…

GIN to the rescue!

```
CREATE INDEX idx_split on voitures USING GIN
(regexp_split_to_array(caracteristiques,','));
```

```
SELECT * FROM voitures
WHERE regexp_split_to_array(caracteristiques,',')
@> '{"climatisation","toit ouvrant"}';
```

3ms contre 200ms.

Un peu de sucre

```
CREATE FUNCTION split_caracteristiques_match (text,text)
RETURNS BOOLEAN LANGUAGE SQL
AS $$
SELECT regexp_split_to_array($1,',') @>
regexp_split_to_array($2,',') $$;


CREATE OPERATOR @-@ (PROCEDURE=split_caracteristiques_match,
 LEFTARG='text',
 RIGHTARG='text');


SELECT * FROM voitures where caracteristiques @-@
'climatisation,toit ouvrant';
```

Et l'index fonctionne toujours…

Indexer du texte brut

- Charger tout le projet Gutenberg FR (22M lignes)

```
CREATE EXTENSION pg_trgm;
```

select show_trgm('texte');

show_trgm

{" t"," te",ext,"te ",tex,xte}

- C'est un TABLEAU !

```
CREATE INDEX idx_trgm ON textes USING gin (contenu
gin_trgm_ops);
```

```
select count(*) from textes where contenu ilike '%valjean%';
 count
-------
  1214
(1 row)

Time: 17,795 ms
```

```
select count(*) from textes where contenu ~* '(valjean.*fantine)|(fantine.*valjean)';
 count
-------
     8
(1 row)

Time: 225,943 ms
```

Et c'est pas tout…

- La structure de GIN est compressée:
  - si peu de valeurs distinctes, compression très élevée
  - ==> compacité comparable aux index bitmap
- On n'a pas parlé de GiST, SP-Gist, BRIN & co
- Recherche k-NN
- Filtres de bloom en «démo» en 9.6
  - Requête sur une liste arbitraire de colonnes, que pour égalité
  - Support de int32 et varchar seul pour le moment