Indexation sur une sous-chaîne de caractères dans PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 mai 2010. Dernière mise à jour le 20 mai 2010

https://www.bortzmeyer.org/indexation-sous-chaine.html

Tous les SGBD disposent de mécanismes d'indexation pour accélerer l'accès aux données (commande SQL `CREATE INDEX` pour créer un tel index). Si on connait la chaîne de caractères **exacte**, on la trouve dans l'index et on a tout de suite ses données. Mais si on ne connait qu'une partie de la chaîne ?

Essayons avec PostgreSQL (tous ces essais ont été faits avec la version 8.3). D'abord, je crée une table associant des personnes (représentées par un simple numéro) à des numéros de téléphone :

```
CREATE TABLE PhonesPersons (person INTEGER NOT NULL, -- Not UNIQUE
                            -- since a person may have several phone numbers
                            phone TEXT NOT NULL -- Not UNIQUE since
                            -- a phone can be used by several persons
);
```

puis je peuple cette table pour des tests avec un million d'entrées (le script SQL complet est (en ligne sur `https://www.bortzmeyer.org/files/index-substring-create.sql`) et le programme de peuplement est un script Python, (en ligne sur `https://www.bortzmeyer.org/files/index-substring-popul` `py`) et, oui, tout ceci est une simplification, je sais bien que les numéros E164 peuvent avoir un code international de un ou trois chiffres). Testons d'abord sans index :

```
% time psql --pset pager essais -c \
        "SELECT person FROM PhonesPersons WHERE phone = '+33 8116243'"
 person
--------
 997313
(1 row)

psql [...] 0.06s user 0.01s system 14% cpu 0.506 total
```

En effet, sans index, PostgreSQL a dû parcourir toutes les données. Cherchons des explications <https://www.bortzmeyer.org/explain-postgresql.html> :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE phone = '+33 8116243';
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on phonespersons  (cost=0.00..18064.00 rows=1 width=4)
   Filter: (phone = '+33 8116243'::text)
(2 rows)
```

C'est confirmé, la requête a pris du temps car il a fallu un balayage séquentiel ("*Seq Scan*") des données. Maintenant, créeons un index avec `CREATE INDEX phone_idx ON PhonesPersons(phone)` et reessayons :

```
psql [...]  0.04s user 0.02s system 97% cpu 0.070 total
```

Le temps écoulé est beaucoup plus court (70 ms au lieu de 500). En effet, l'utilisation de l'index permet à PostgreSQL d'être plus efficace :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE phone = '+33 8116243';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using phone_idx on phonespersons  (cost=0.00..8.41 rows=1 width=4)
   Index Cond: (phone = '+33 8116243'::text)
(2 rows)
```

Bien, cela, c'était pour le cas où on cherchait un numéro de téléphone spécifique. Mais si on cherche une partie du numéro, par exemple uniquement les numéros français (commençant par +33) ? SQL dispose pour cela de la commande `LIKE` :

```
essais=> SELECT person FROM PhonesPersons WHERE phone LIKE '+33%';
 person
---------
  651229
  829130
...
```

mais elle n'utilise pas l'index et est donc très lente :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE phone LIKE '+33%';
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on phonespersons  (cost=0.00..18064.00 rows=163 width=4)
   Filter: (phone ~~ '+33%'::text)
(2 rows)
```

On peut convaincre `LIKE` d'utiliser un index grâce aux "*Operator Classes*" <http://www.postgresql.org/docs/current/interactive/indexes-opclass.html>. Par exemple, par défaut, `LIKE` utilise les "*locales*", ce qui peut sérieusement le ralentir, certaines "*locales*" étant complexes. Si on crée l'index avec `CREATE INDEX phone_idx ON PhonesPersons(phone varchar_pattern_ops)` pour utiliser l'"*operator class*" `varchar_pattern_ops` (insensible à la "*locale*"), l'index fonctionne :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE phone LIKE '+33%';
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on phonespersons  (cost=6.06..569.94 rows=163 width=4)
   Filter: (phone ~~ '+33%'::text)
   ->  Bitmap Index Scan on phone_idx  (cost=0.00..6.02 rows=163 width=0)
         Index Cond: ((phone ~>=~ '+33'::text) AND (phone ~<~ '+35'::text))
(4 rows)
```

Notez que cela ne marcherait pas si le joker (le caractère %) était au début de la chaîne (ce problème semble commun à tous les SGBD, à moins que quelqu'un aie un contre-exemple ?). Pour cela, il faut utiliser des méthodes plus complexes comme décrites dans les articles « Utiliser un index pour les recherches sur des motifs tels que « colonne LIKE '%chaîne' » <http://blog.postgresql.fr/index.php?post/drupal/396> » ou bien « Index inversé, en C <http://blog.postgresql.fr/index.php?post/drupal/393> ».

Mais, ici, je prends une autre méthode, reposant sur les fonctions.

L'idée est de créer un index pour une sous-chaîne. Pour simplifier, au début, on va supposer une sous-chaîne de longueur fixe, les trois premiers caractères, avec la fonction `substr` :

```
essais=> SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) = '+33';
 person
---------
  651229
  829130
...
```

Cette requête n'utilise pas l'index qui existe, car on ne donne au SGBD qu'une sous-chaîne :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) = '+33';
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on phonespersons  (cost=0.00..20564.00 rows=5000 width=4)
   Filter: (substr(phone, 1, 3) = '+33'::text)
(2 rows)
```

POur arranger cela, créeons un index pour la fonction `substr`. PostgreSQL permet d'indexer le résultat d'une fonction si celle-ci a été marquée comme `IMMUTABLE`, c'est-à-dire pure (ce qui est le cas de `substr`) : `CREATE INDEX countrycode_idx ON PhonesPersons (substr(phone, 1, 3));`. Désormais, l'index est utilisé :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) = '+33';
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on phonespersons  (cost=83.09..5808.11 rows=5000 width=4)
   Recheck Cond: (substr(phone, 1, 3) = '+33'::text)
   ->  Bitmap Index Scan on countrycode_idx  (cost=0.00..81.84 rows=5000 width=0)
         Index Cond: (substr(phone, 1, 3) = '+33'::text)
(4 rows)
```

Est-ce vraiment plus rapide ? Sans index :

```
% time psql --pset pager essais -c \
      "SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) = '+33'"
...
psql [...]  0.05s user 0.01s system 5% cpu 1.036 total
```

Avec index :

```
psql [...]  0.06s user 0.01s system 87% cpu 0.073 total
```

Donc, oui, l'index est réellement efficace, 73 milli-secondes d'attente au lieu d'une seconde entière.

Ici, le cas était simple car la sous-chaîne était de longueur fixe. Pour essayer avec des longueurs variables, abandonnons les numéros de téléphone pour les adresses de courrier électronique :

```
CREATE TABLE EmailsPersons (person INTEGER NOT NULL,
                            email TEXT NOT NULL
);
```

et créeons une fonction SQL pour extraire le TLD de l'adresse (notons qu'on l'a déclaré `IMMUTABLE`, ce qu'elle est réellement ; mais PostgreSQL ne vérifie pas cette déclaration, c'est au programmeur de faire attention) :

```
-- Extracts the TLD from a domain name (or an email address)
CREATE OR REPLACE FUNCTION tld(TEXT) RETURNS TEXT IMMUTABLE AS
   '
     DECLARE
       first_dot INTEGER;
       rest TEXT;
     BEGIN
       first_dot = strpos($1, ''.'');
       rest = substr($1, first_dot+1);
       IF strpos(rest, ''.'') = 0 THEN
          RETURN rest;
       ELSE
          RETURN last_label(rest);
       END IF;
     END;
   '
 LANGUAGE PLPGSQL;
```

On peut l'utiliser :

```
essais=> SELECT email, tld(email) FROM EmailsPersons WHERE person = 147676;
               email                | tld
------------------------------------+-----
 0jdm8k13r3ek4rxs2tqo7-t@khl4u6sf.de | de
 owqwtrs8o2o20s8ri3sb@g4eoas.fr      | fr
(2 rows)
```

Cette requête n'utilise pas d'index :

```
essais=> EXPLAIN SELECT email FROM EmailsPersons WHERE  tld(email) = 'de';
                             QUERY PLAN
---------------------------------------------------------------------
 Seq Scan on emailspersons  (cost=0.00..271292.00 rows=5000 width=38)
   Filter: (tld(email) = 'de'::text)
(2 rows)
```

Mais on peut créer un index sur cette fonction avec `CREATE INDEX tld_idx ON EmailsPersons (tld(email));` et il sera utilisé :

```
essais=> EXPLAIN SELECT email FROM EmailsPersons WHERE  tld(email) = 'de';
                                QUERY PLAN
--------------------------------------------------------------------------
 Bitmap Heap Scan on emailspersons  (cost=87.34..9201.36 rows=5000 width=38)
   Recheck Cond: (tld(email) = 'de'::text)
   ->  Bitmap Index Scan on tld_idx  (cost=0.00..86.09 rows=5000 width=0)
         Index Cond: (tld(email) = 'de'::text)
(4 rows)
```

Attention toutefois en testant l'utilisation de l'index avec `EXPLAIN`. Le SGBD peut parfaitement décider de ne **pas** utiliser un index, s'il estime que cela sera plus rapide sans (par exemple, si un critère de sélection est peu discriminant et qu'il faut extraire presque toutes les données, passer par l'index n'apporte rien). Le vrai test est donc le gain de performance, pas la réponse de `EXPLAIN`!

Pour les deux exemples que j'ai utilisé, où les données sont stockées sous forme de texte alors qu'elles sont en fait structurées, une solution meilleure serait peut-être de créer un type spécifique <http://www.postgresql.org/docs/current/interactive/xtypes.html>. C'est ce que fait par exemple "*Prefix Range*" <http://github.com/dimitri/prefix>.

À noter que les index de PostgreSQL ne marchent apparemment pas sur les requêtes de non-existence. Si on cherche les adresses dans le TLD `.de`, mais pas si on cherche les adresses n'étant **pas** dans ce TLD (opérateur <>). Dans le cas d'un TLD peu fréquent, cela pourrait s'expliquer pour la raison ci-dessus (s'il faut récupérer presque toutes les valeurs, l'index est inutile) mais j'observe le même problème pour les cas où il y a peu de données. (Voir la discussion sur StackOverflow <http://stackoverflow.com/questions/2864267/sql-indexes-for-not-equal-searches>.) C'est dommage, car cela pourrait certainement être utile.

Un contournement simple est de transformer le test « différent de CHAÎNE » en « strictement supérieur ou inférieur à CHAÎNE ». L'index est alors bien utilisé. Essayons avec le préfixe téléphonique espagnol, sur une base où ils sont nombreux, de façon à ce que la négation ne sélectionne pas trop de tuples (car, sinon, rappelez-vous, PostgreSQL peut décider de ne pas utiliser l'index) :

```
essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) <> '+34';
---------------------------------------------------------------------
 Seq Scan on phonespersons  (cost=0.00..20564.00 rows=9333 width=4)
   Filter: (substr(phone, 1, 3) <> '+34'::text)
(2 rows)

essais=> EXPLAIN SELECT person FROM PhonesPersons WHERE substr(phone, 1, 3) > '+34' OR substr(phone, 1, 3) < '+3
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Bitmap Heap Scan on phonespersons  (cost=163.34..6109.12 rows=9319 width=4)
   Recheck Cond: ((substr(phone, 1, 3) > '+34'::text) OR (substr(phone, 1, 3) < '+34'::text))
   ->  BitmapOr  (cost=163.34..163.34 rows=9333 width=0)
```

```
         ->  Bitmap Index Scan on areacode_idx  (cost=0.00..124.28 rows=7459 width=0)
               Index Cond: (substr(phone, 1, 3) > '+34'::text)
         ->  Bitmap Index Scan on areacode_idx  (cost=0.00..34.40 rows=1874 width=0)
               Index Cond: (substr(phone, 1, 3) < '+34'::text)
(7 rows)
```

Cela ne marche qu'avec certains types d'index <http://www.postgresql.org/docs/current/interactive/indexes-types.html>, ceux qui sont ordonnés (b-tree mais pas hash) et ont donc les opérateurs « supérieur à » et « inférieur à ».

Merci à Mark Byers, Guillaume Lelarge, Dimitri Fontaine, Mathieu Arnold, araqnid et Thomas Reiss pour leur aide érudite.