PostgreSQL et les quantiles, via les « window functions »

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 novembre 2012

https://www.bortzmeyer.org/postgresql-quantiles.html

Le SGBD PostgreSQL dispose de tas de fonctions utiles. Une relativement peu connue est le système des « "*window functions*" » qui agissent sur un découpage d'un ensemble de tuples, s'appliquant à chaque part découpée. Cela permet notamment de travailler sur les quantiles.

Bon, avant de détailler, voyons un cas réel et concret. On a une base de données du personnel et une table stocke les salaires :

```
CREATE TABLE Salaries
  (id SERIAL UNIQUE NOT NULL,
   name TEXT UNIQUE NOT NULL,
   salary INTEGER NOT NULL);

INSERT INTO Salaries (name, salary) VALUES ('John', 3000);
INSERT INTO Salaries (name, salary) VALUES ('Sandy', 2000);
...
```

On charge cette table (le fichier complet est disponible (en ligne sur https://www.bortzmeyer.org/files/postgresql-quantiles-create.sql)). Il est alors facile de trouver des choses comme le salaire moyen :

```
essais=> SELECT round(avg(salary)) FROM Salaries;
 round
-------
  3417
(1 row)
```

ou de trier selon le salaire, ici du plus payé vers le moins payé :

```
essais=> SELECT name,salary FROM Salaries ORDER BY salary DESC;
  name   | salary
---------+--------
 Patt    |   9600
 Robert  |   7000
 Steve   |   4000
...
```

Mais on cherche des choses plus subtiles. Par exemple, le salaire moyen plait aux dirigeants pour la communication car il est relativement élevé, tiré vers le haut par quelques gros salaires. En cas de forte inégalité, ce qui est le cas dans l'entreprise de notre exemple, il est nettement moins pertinent que le le salaire médian <https://www.bortzmeyer.org/mediane-et-moyenne.html>. Il existe plusieurs méthodes pour calculer ce dernier. Par exemple, la médiane étant par définition le salaire tel que la moitié des employés gagnent davantage, un calcul approximatif (qui ignore les arrondis) est :

```
essais=> SELECT salary FROM salaries ORDER BY salary DESC LIMIT
                           (SELECT count(id) FROM salaries) / 2;
...
   2900
```

Un moyen plus sophistiqué est de travailler sur les quantiles. Les quantiles sont les découpages des données triées en N parties égales. Si N = 100, on parle de centiles, si N = 10 de déciles, etc. Pour la médiane, on va prendre N = 2. Mais, avant cela, un petit détour par les « "*window functions*" <http://www.postgresql.org/docs/current/interactive/functions-window.html> » de PostgreSQL.

Ces fonctions opèrent sur une partie des tuples retournés par une requête. Le découpage en parties est fait par la clause `OVER`. Prenons l'exemple d'un découpage en autant de parties qu'il y a de tuples triés, et de la « "*window function*" » `rank()` qui indique la place d'un tuple par rapport aux autres :

```
essais=> SELECT name,salary,rank() OVER (ORDER BY salary DESC) FROM Salaries
                                ORDER BY name;
  name   | salary | rank
---------+--------+------
 Alfonso |   2300 |   14
 Bill    |   2900 |    8
 Bob     |   2800 |   10
 Chris   |   2900 |    8
 Chuck   |   3800 |    4
...
```

Le `(ORDER BY salary DESC)` après `OVER` indique que les tuples vont être classés selon le salaire décroissant. `rank()` va alors s'appliquer à ce classement : Alfonso a le quatorzième salaire (en partant du haut), Bill le huitième, etc. Notez que les ex aequo sont possible (ici, Bill et Chris). `row_number()` ferait la même chose que `rank()` mais sans les ex aequo.

Revenons maintenant aux quantiles. La fonction `ntile()` permet de découper un ensemble de tuples en quantiles (le nombre de quantiles étant l'argument de `ntile()`, ici on utilise des terciles) :

```
essais=> SELECT  name,salary,ntile(3) OVER (ORDER BY salary) FROM Salaries;
  name   | salary | ntile
---------+--------+-------
 Sandy   |   2000 |     1
...
 Lou     |   2600 |     2
...
 Peter   |   3400 |     3
```

Ici, Sandy est dans le tercile le plus bas (`ntile()` renvoie 1) et Peter dans celui des meilleurs salaires.

On peut utiliser les quantiles de tas de façons. Par exemple :

```
essais=> SELECT n,min(salary),max(salary) FROM
             (SELECT name,salary,ntile(3) OVER (ORDER BY salary) as n FROM Salaries) AS q
             GROUP BY n ORDER BY n;
 n | min  | max
---+------+------
 1 | 2000 | 2500
 2 | 2600 | 3000
 3 | 3400 | 9600
```

Ici, on a la plage des salaires pour chaque tercile. Et la médiane que j'avais promise ?

```
essais=> SELECT n,min(salary),max(salary) FROM
             (SELECT name,salary,ntile(2) OVER (ORDER BY salary) as n FROM Salaries) AS q
             GROUP BY n ORDER BY n;
 n | min  | max
---+------+------
 1 | 2000 | 2800
 2 | 2900 | 9600
```

Elle est de 2850.

Une autre façon de représenter l'inégalité est de voir quelle part des salaires représente chaque quantile. Passons à des quintiles (N = 5) :

```
essais=> SELECT n, sum(salary)*100/(SELECT sum(salary) FROM Salaries) AS percentage
            FROM (SELECT name,salary,ntile(5) OVER (ORDER BY salary) as n FROM Salaries) AS q
             GROUP BY n ORDER BY n;
 n | percentage
---+------------
 1 |         13
 2 |         16
 3 |         18
 4 |         17
 5 |         33
```

Le quintile supérieur représente le tiers du total des salaires. (On note que c'est le seul des cinq quantiles à représenter plus que sa part, qui serait de 20 % en cas de totale égalité.)

Tout cela ne sert évidemment pas qu'aux salaires. Si on prend des requêtes DNS analysées par DNSmezzo <`http://www.dnsmezzo.net/`>, et qu'on étudie la répartition du trafic par résolveur, on observe que certains résolveurs n'envoient que très peu de requêtes (c'était peut-être un étudiant faisant un ou deux dig), alors que d'autres sont bien plus bavards. Ici, on étudie 447 658 requêtes reçues par un serveur faisant autorité, en IPv6. On observe 3 281 préfixes IPv6 de longueur /48. Quelle est la répartition des requêtes ?

```
dnsmezzo=> SELECT n,min(requests),max(requests),
       to_char(sum(requests)*100/(SELECT count(id) FROM DNS_packets WHERE query AND family(src_address)=6),
        (SELECT prefix,requests,ntile(10) OVER (ORDER BY requests) AS n FROM
             (SELECT set_masklen(src_address::cidr, 48) AS prefix, count(id) AS requests FROM
                 DNS_packets WHERE query AND  family(src_address)=6
                GROUP BY prefix ORDER by requests DESC) AS qinternal) AS qexternal
   GROUP BY n ORDER BY n;

 n  | min |  max  | percentage
----+-----+-------+------------
  1 |   1 |     1 |    .1
  2 |   1 |     1 |    .1
  3 |   1 |     2 |    .1
  4 |   2 |     3 |    .2
  5 |   3 |     4 |    .2
  6 |   4 |     7 |    .4
  7 |   7 |    14 |    .8
  8 |  14 |    32 |   1.6
  9 |  32 |    89 |   3.9
 10 |  89 | 81718 |  92.7
```

On voit que les cinq premiers déciles ont un trafic négligeable, et que le dixième décile, les clients les plus bavards, fait plus de 90 % des requêtes à lui seul. (En utilisant des centiles et non plus des déciles, on confirme cette grande inégalité : le centile supérieur fait 72 % des requêtes.)

Un bon tutoriel sur les « "*windows functions*" » sur PostgreSQL (et qui couvre la clause `PARTITION` dont je n'ai pas parlé ici), est en ligne <http://www.postgresql.org/docs/current/interactive/tutorial-window.html>. Pour jouer avec les quantiles, il existe aussi une extension de PostgreSQL que je n'ai pas testée, quantile <http://pgxn.org/dist/quantile>. Elle est notamment censée être plus rapide.

Ces fonctions ne sont pas spécifiques à PostgreSQL, elles sont apparues dans la norme SQL en 2003 sous le nom de « fonctions analytiques ». On les trouve dans plusieurs autres SGBD, par exemple Firebird <http://www.firebirdsql.org/file/community/ppts/fbcon11/fb3windowing.pdf>. On les trouve dans beaucoup de questions sur StackOverflow <http://stackoverflow.com/questions/tagged/window-functions>. Merci à Malek Shabou et Pierre Yager pour leur relecture.