Lab Exercise 05

Triggers & PLpgSQL functions

Aims

This exercise aims to give you practice in:

- implementing triggers via PLpgSQL functions

The following sections of the PostgreSQL Manual will be useful for this lab: the SQL `CREATE TRIGGER` statement; the PLpgSQL language description.

Background

For this Lab, we will make use of a similar beer ratings database to the one we used in lab04. Files containing a relational schema for this database and data to populate the tables are available in the files:

```
schema.sql
data.sql
test.sql
```

You can extract a copy of the data files into this directory. On the CSE workstations, you can do this via:

```
$ cp /home/cs9311/web/24T2/lab/05/*.sql ./
```

You should delete your old “beers” database, then re-create it afresh by loading the schema and data into the new database. The following commands will do this:

```
$ dropdb beers
$ createdb beers
$ psql beers -f ./schema.sql
... which will produce lots of NOTICE messages ...
$ psql beers -f ./data.sql
... which will produce messages about table ids ...
```

This will populate all of the tables except `Ratings` . We will populate this later.

Imagine that the database used in lab04 ends up as the back-end database for a web site `MyWorldOfBeer.com` with thousands of beers, tens of thousands of raters and millions of ratings.

One important function for such a site would be producing a list of the top-ten rated beers. It would be possible to produce such a list using the `BeerSummary` function from lab04. However, as the database grows, this operation would become slower and slower.

Let us imagine that with the size of database described above, it is now intolerably slow to produce a list of the top-ten rated beers. The database designers decide to add three extra columns to the `Beer` table to hold:

- `totRating` : the sum of all ratings for each beer
- `nRatings` : the number of ratings for each beer
- `rating` : the average rating for each beer (`totRating/nRatings`)

The values of these columns should always accurately reflect the state of the `Ratings` table.

We can express what is required as a series of semi-formal assertions:

```
for every Beer b (b.totRating = (sum(score) from Ratings where beer = b.id))
for every Beer b (b.nRatings = (count(score) from Ratings where beer = b.id))
for every Beer b (b.rating = b.totRating/b.nRatings, if b.nRatings > 0, null otherwise)
```

Of course, ensuring that the database always satisfies these constraints requires that the above columns in the `Beer` table be maintained This, in turn, requires that some work is done every time a rating is added, removed, or changed.

Exercise

Write triggers and their associated PLpgSQL functions to maintain these assertions on the `Beer` table in response to all possible changes to the `Ratings` table. Place your trigger and function definitions in a file called `lab05.sql` .

You can assume that the only kind of update operation is one that changes the rating by a given rater for a given beer. In other words, the only updates will be of the form:

```
update Ratings
set    score = X
where  rater = Y and beer = Z;
```

If you want to loosen that assumption (i.e. allow absolutely any kind of update), then that's fine too ... but it will require you write extra code.

While you're developing your triggers, you should test them by adding new rating records, updating existing rating records and deleting rating records, and then checking whether the above assertions are maintained.

For a final check, reset the database as above:

```
$ dropdb beers
$ createdb beers
$ psql beers -f ./schema.sql
... which will produce lots of NOTICE messages ...
$ psql beers -f ./data.sql
... which will produce messages about table ids ...
$ psql beers -f ./lab05.sql
... which will produce messages functions/triggers ...
```

and then run a sequence of modifications to the `Ratings` table via:

```
$ psql beers -f ./test.sql
```

If you then check the contents of the `Beer` table, you should observe:

```
beers=# select * from Beer order by id;
 id |          name          | style | brewer | totrating | nratings | rating
----+------------------------+-------+--------+-----------+----------+--------
  1 | Rasputin               |    10 |      9 |         8 |        3 |      2
  2 | 80/-                   |    13 |     11 |         4 |        1 |      4
  3 | Sierra Nevada Pale Ale |     3 |      6 |        20 |        5 |      4
  4 | Old Tire               |    11 |      7 |         5 |        1 |      5
  5 | Old                    |    12 |      3 |         7 |        2 |      3
  6 | New                    |     1 |      3 |         3 |        2 |      1
  7 | Fosters                |     1 |      1 |         3 |        1 |      3
  8 | James Squire Amber Ale |    12 |     12 |         3 |        1 |      3
  9 | James Squire Pilsener  |     2 |     12 |         7 |        2 |      3
 10 | Burragorang Bock       |     5 |      4 |         7 |        2 |      3
 11 | Scharer's Lager        |     1 |      4 |         3 |        1 |      3
 12 | Chimay Red             |     9 |     10 |         3 |        1 |      3
 13 | Chimay Blue            |     9 |     10 |         0 |        0 |
 14 | Victoria Bitter        |     1 |      1 |         3 |        3 |      1
 15 | Sterling               |     1 |      1 |         0 |        0 |
 16 | Empire                 |     1 |      1 |         6 |        2 |      3
 17 | Premium Light          |     1 |     14 |         0 |        0 |
 18 | Sparkling Ale          |    12 |     13 |         0 |        0 |
 19 | Sheaf Stout            |     3 |      3 |         0 |        0 |
 20 | Crown Lager            |     1 |      1 |         2 |        1 |      2
```

```
  21 | Bigfoot Barley Wine    |     4 |     6 |        3 |       1 |      3
  22 | James Squire Porter    |     7 |    12 |        0 |       0 |
  23 | Redback                |    14 |     5 |        9 |       2 |      4
  24 | XXXX                   |     1 |     2 |        5 |       1 |      5
  25 | Red                    |     1 |     3 |        0 |       0 |
(25 rows)
```

You will, of course, have observed that `test.sql` only performs insert operations. We assume that you have tested the triggers for all other operations yourself.

[Solution]