Database Tutorial 4: Constraints, Triggers, and Views

30 November 2018

1. Consider an online bookshop which sometimes promotes books by displaying them on the front page of their website. Their web application uses a database created in PostgreSQL using the following statements:

```
CREATE TABLE Books (
    id INTEGER PRIMARY KEY,
    category TEXT,
    price FLOAT CHECK (price>0),
    promoted BOOLEAN DEFAULT True
) ;


INSERT INTO Books ( id , category , price ) VALUES( 1 , 'Dictionary ' , 100) ;
INSERT INTO Books ( id , category , price ) VALUES( 2 , 'Dictionary ' , 150) ;
INSERT INTO Books ( id , category , price ) VALUES( 3 , 'Science ' , 120);
INSERT INTO Books ( id , category , price ) VALUES( 4 , 'Science ' , 190);
INSERT INTO Books ( id , category , price ) VALUES( 5 , 'Science ' , 320);
```

   a. Create a new VIEW called “PromotionSummary” which outputs 3 columns named “category”, “minprice” and “maxprice” containing the category name, minimum price of all promoted books and maximum price of all promoted books. A promoted book has its “promoted” attribute set to *True*.
   b. Create a trigger so that, when a tuple from the “PromotionSummary” view is deleted, all Books from the corresponding category have their “promoted” attribute set to *False*. E.g. if the entry in “PromotionSummary” for category “Novel” is deleted, all entries in “Books” with category “Novel” have their “promoted” attribute set to *False*.

2. Database integrity can be improved by many techniques:
   - views: virtual tables that show useful information that would create redundancy if stored in the actual tables
   - constraints: conditions on attribute values and tuples
   - triggers: automated checks and actions performed on entire tables.

   As a general rule, these methods should be applied in the above order: if a view can do the job, constraints are not needed, and if constraints can do the job, triggers are not needed. The task in this question is to show how to guarantee integrity in an exam question database. The attributes that are needed are the following:

   - an Exam has a date and a course code, as well as a total number of points
   - a Question belongs to a certain exam, has its own question number, a number of points, and a text (the question itself)

   Your task is to give a database definition (tables, triggers, and views) that guarantees the following integrity conditions:

   a. A course can have maximally one exam on the same day.
   b. The number of points in an exam is the sum of the numbers of points in its questions.
   c. An exam question is uniquely determined by its number, together with the exam to which it belongs.

d. An exam may not use the same question text twice.
e. The number of questions in an exam may not exceed 10.
f. The number of questions in an exam must be at least 5. (Hint: can be tricky!)

Write your answer as SQL code (except for question f) for tables, views, and triggers, marking with letters ((a), (b), etc.) which part of the code guarantees which property. You can get 2 points for each of the properties. Unnecessarily complicated answers (e.g. using a trigger when a constraint would be enough) may reduce points.

3. In the year 2127, the first spaceship to colonize Mars carries 1337 colonists. When they arrive on the planet, they will build a city and live there. Following democratic principles, the spaceship captain, captain Picard, asks you to improve an SQL database to help with the voting process. The existing SQL database was created with the following statement:

```
CREATE TABLE Votes (
    cityname TEXT PRIMARY KEY ,
    votecount INT
) ;

-- To add a vote , you can use either INSERT or UPDATE , as shown below:
INSERT INTO Votes (cityname , votecount ) VALUES( 'Mars City One' , 34 ) ;
INSERT INTO Votes (cityname , votecount ) VALUES( 'New Gothenburg ' , 11);
INSERT INTO Votes (cityname , votecount ) VALUES( 'Picardia ' , 1);
UPDATE Votes SET votecount = votecount +3 WHERE cityname='New Gothenburg ';
```

a. Create a new VIEW called “VoteSummary” which outputs 2 columns named “cityname” and “percentage” containing the cityname and percentage of votes cast for that cityname. The output is sorted according to the votecount, highest votecount first. After the example votes above, there would be 34 votes for “Mars City One” out of a total of 49 votes, so the top row of the “VoteSummary” VIEW would be (’Mars City One’, 69.3878). There is no need to round of the percentage.
b. Create a trigger to update the “Votes” table, to keep track of how many colonists have not voted yet. This count will appear next to the special cityname “<not voted>”. In the example above, 49 votes have been cast out of 1337 possible votes. This means the trigger needs to create or update an entry with cityname “<not voted>” and votecount 1288 (= 1337 - 49). Keep in mind that you need to create this entry if it does not exist yet. There is no need for a trigger on DELETE.