Databases - Exercise 4:
Views, Constraints, Triggers

29 November 2019

1 Constraints and Triggers

Consider the following CREATE TABLE statements.

```
CREATE TABLE Artists(
  name TEXT PRIMARY KEY);

CREATE TABLE Songs(
  title TEXT PRIMARY KEY,
  length INT NOT NULL,
  artistName TEXT NOT NULL,
  FOREIGN KEY (artistName) REFERENCES Artists(name),
  CONSTRAINT SongTitleAndArtistNameIsUnique UNIQUE (title, artistName)
);

CREATE TABLE Playlists (
  id TEXT PRIMARY KEY,
  name TEXT,
  owner TEXT);

CREATE TABLE PlaylistSongs (
  playlist TEXT REFERENCES Playlists(id),
  song TEXT,
  artist TEXT,
  position INTEGER,
  FOREIGN KEY (song,artist) REFERENCES Songs(title,artistName),
  PRIMARY KEY (playlist,position)
);
```

1.1 Constraints

Which of the following properties are guaranteed by the given constraints?

1. Every song in the playlist exists.
2. All playlists of ~~one and~~ the same owner have different names.
3. All positions in a given playlist are unique.
4. The positions ~~can be~~ ARE listed in order 1, 2, 3, ... with no numbers missing.

Answer as follows:

- If a property is guaranteed, say by which constraints exactly.
- If a property is not guaranteed, give a counterexample.

1.2 Views

Create a view that, for a playlist with id M123, shows the contents of the playlist with the following layout:

```
 position | song               | artist       | length
----------+--------------------+--------------+--------
        1 | Dancing Quenn      | ABBA         | 232
        2 | Too late for love | John Lundvik | 187
```

1.3 Triggers

Create a trigger that enables directly building the playlist M123 via the view defined in the previous question. The behaviour should be as follows:

- Insertion of (position, song, artist, length) in M123 means an insert in PlaylistSongs such that
  - the song and artist are inserted as they are given by the user
  - the length is ignored (even if it contradicts the Songs table)
  - the position given by the user is respected, at the same time maintaining the sequential order 1, 2, 3, ... of the playlist

Maintaining the sequential order implies the following: assume that the positions in the old playlist are 1,2,3,4. Then

- if the user inserts in the next position, 5, then 5 is also used as the position of the row inserted to the table
- if it is larger, say 8, then the position of the row inserted to the table is still 5
- if it is smaller, say 3, then the position of the row inserted is 3, but the positions of the old rows 3 and 4 are changed to 4 and 5, respectively

Note: You may notice the problem that, while the trigger is running, the unicity of positions is temporarily violated. But you can ignore the complications with this: just make sure that, when the trigger has finished its work, all positions are unique.

2 Views, constraints, and triggers again

Database integrity can be improved by several techniques:

- Views: virtual tables that show useful information that would create redundancy if stored in the actual tables
- SQL Constraints: conditions on attribute values and tuples
- Triggers (and assertions): 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 implement a database for a cell phone company.

You are allowed to use any SQL features we have covered in the course. While the description below gives requirements for what should be in the database, you are allowed to divide it across as many tables and views as you need to. Points will be deducted if your solution uses a trigger where a constraint or view would suffice, or if your solution is drastically over-complicated.

For triggers, it is enough to specify which actions and tables it applies to, and PL/(pg)SQL pseudo-code of the function it executes.

Your task: The database contains Customers and Subscriptions. Each customer can have any number of subscriptions. Below are values that should be in the database:

- A Customer has a unique id number and a name, a monthly billing and a Boolean indicating if it is a private customer (true meaning it is a private customer).
- A Subscription belongs to a customer and has a unique phone number, a plan, a monthly fee and a balance.

Implement the following additional constraints in your design.
Put letters in the margin of your code indicating where each constraint is implemented (possibly the same letter in several places):

Check a Each plan must be one of 'prepaid', 'flatrate' or 'corporate'.

Check b The balance value must be 0 if the plan is not 'prepaid'.

Check (either OR) c Private customers cannot have 'corporate' plans (but non-private customers may still have any plans including but not limited to 'corporate').

View —> get billing from here + check value >= 0 d The monthly billing of a customer must be the sum of the fees of all that customers numbers, and all fees must be non-negative.

ON DELETE CASCADE e If a customer is deleted, its connected subscriptions should be deleted automatically.

Trigger f If the last subscription belonging to a customer is deleted, the customer should be deleted automatically.

3 Troubleshooting

Rotten Potatoes is a company with a website where users can review movies by rating and commenting on them. The company is requesting your services as a consultant since its database is not behaving as expected. Your task is to find why that is and propose how to fix it.

Domain description

- Users have a login which identifies them along with an unique email. Users can pay a fee to become premium, and their subscriptions have an expiration date.
- Movies have a title, year of release, and a genre that can be either action, sci-fi, comedy, or drama. Since movies can share tiles, they are disambiguated by their year of release.
- Users review movies by rating them with a point system from 1 to 5 stars; additionally, premium users can comment on their reviews.
- A movie score is determined by the average of its ratings.
- To keep users up to date, Rotten Potatoes provides a list of "certified crisp" movies, which are the movies whose score is above 4.

Questions

1. Is their implementation *addressing* all the constraints described in the domain? If so, highlight where, otherwise, mention what is missing.
2. After inserting a couple of reviews the company has noticed some movies listed as "certified crisp" whose average is less than 4. Why is this happening? what needs to be changed/added/removed?
3. To clean the database from fake accounts, the company have removed all reviews posted by shady users; however, "certified crisp" remain unchanged. What went wrong? how can it be fixed?
4. Is there redundancy in the information that is being stored? If so, where is it happening? and how can this be improved?

Implementation

```
1  -- Users' basic information
2  CREATE TABLE Users (
3      login TEXT PRIMARY KEY,
4      email TEXT NOT NULL,
5      UNIQUE (email));
6
7  -- Premium users are an specialization of users
8  CREATE TABLE PremiumUsers (
9      usr             TEXT NOT NULL REFERENCES Users,
10     expiration_date DATE NOT NULL,
11     PRIMARY KEY (usr));
12
13 -- Movies' basic information
14 CREATE TABLE Movies (
15     title        TEXT    NOT NULL,
16     year         INTEGER NOT NULL,
17     genre        TEXT    NOT NULL,
18     PRIMARY KEY (title, year),
19     CHECK (genre IN ('Action', 'Sci-fi', 'Comedy', 'Drama')));
20
21 -- All users can rate a movie
22 CREATE TABLE Ratings (
23     usr   TEXT    NOT NULL REFERENCES Users,
24     title TEXT    NOT NULL,
25     year  INTEGER NOT NULL,
26     rate  INTEGER NOT NULL,
27     PRIMARY KEY (usr, title, year),
28     FOREIGN KEY (title, year) REFERENCES Movies,
29     CHECK (rate IN (1,2,3,4,5)) -- Point system);
30
31 -- Only premium users are allowed to comment
32 CREATE TABLE Comments (
33     usr     CHAR(5) NOT NULL REFERENCES PremiumUsers,
34     title   TEXT    NOT NULL,
35     year    INTEGER NOT NULL,
36     comment TEXT    NOT NULL,
37     PRIMARY KEY (usr, title, year),
38     FOREIGN KEY (title, year) REFERENCES Movies);
39
40 -- Movies with score above 4.0
41 CREATE TABLE CertifiedCrisp (
42     title   TEXT    NOT NULL,
43     year    INTEGER NOT NULL,
44     PRIMARY KEY (title, year),
45     FOREIGN KEY (title, year) REFERENCES Movies);
46
```

```
47 CREATE VIEW MoviesScores AS
48     SELECT title AS movie, year, COALESCE (AVG(rate),0) AS score
49     FROM Movies NATURAL JOIN Ratings
50     GROUP BY (title , year);
51
52 CREATE VIEW Reviews AS
53     SELECT usr, title AS movie, year, rate, comment
54     FROM MOVIES
55     NATURAL LEFT JOIN Ratings
56     NATURAL LEFT JOIN Comments;
57
58 CREATE FUNCTION insert_review () RETURNS trigger AS $$
59 DECLARE newScore FLOAT;
60 BEGIN
61     INSERT INTO Ratings VALUES (NEW.usr, NEW.movie, NEW.year, NEW.rate);
62
63     IF (EXISTS ( SELECT usr FROM PremiumUsers WHERE usr = NEW.usr)
64         AND NEW.comment IS NOT NULL)
65     THEN INSERT INTO Comments
66                 VALUES (NEW.usr, NEW.movie, NEW.year, NEW.comment);
67     END IF;
68
69     newScore := (SELECT score FROM MoviesScores
70                  WHERE (movie,year) = (NEW.movie, NEW.year));
71
72     IF (NOT EXISTS (SELECT * FROM CertifiedCrisp
73                     WHERE (title,year) = (NEW.movie, NEW.year))
74         AND (newScore > 4))
75     THEN INSERT INTO CertifiedCrisp VALUES (NEW.movie, NEW.year);
76     END IF;
77     RETURN NEW;
78 END;
79 $$ LANGUAGE plpgsql;
80
81 CREATE FUNCTION delete_review () RETURNS trigger AS $$
82 BEGIN
83     DELETE FROM Ratings WHERE usr = OLD.usr AND title = OLD.movie
84            AND year = OLD.year;
85
86     IF (EXISTS (SELECT usr FROM PremiumUsers WHERE usr = OLD.usr))
87     THEN
88         DELETE FROM Comments WHERE usr = OLD.usr AND title = OLD.movie
89                AND year = OLD.year;
90     END IF;
91     RETURN OLD;
92 END;
93 $$ LANGUAGE plpgsql;
94
```

```
95  CREATE TRIGGER InsertReview
96    INSTEAD OF INSERT ON Reviews
97      FOR EACH ROW EXECUTE PROCEDURE insert_review();
98
99  CREATE TRIGGER DeleteReview
100   INSTEAD OF DELETE ON Reviews
101     FOR EACH ROW EXECUTE PROCEDURE delete_review();
```

The website only allows to insert and delete reviews through Reviews view so that users' premium subscriptions are checked, and "certified crisp" is updated accordingly.