Databases
TDA357/DIT621

Exercise 4

Constraints, Views, Functions and Triggers

Database integrity can be improved by several techniques:

- SQL Constraints: conditions on attribute values and tuples;
- Views: virtual tables that show useful information that would create redundancy if stored in the actual tables;
- 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 constraint or a view can do the job, then triggers are not needed.

1 Music

Consider the following `CREATE TABLE` statements:

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

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

CREATE TABLE Playlists (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  owner TEXT NOT NULL );
```

```
CREATE TABLE PlaylistSongs (
  playlist TEXT REFERENCES Playlists(id),
  song TEXT,
  artist TEXT,
  position INT NOT NULL,
  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?

a) Every song in the playlist exists.

b) All playlists of one and the same owner have different names.

c) All positions in a given playlist are unique.

d) The positions can be 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.

Solution:

a) Yes: the foreign key reference in `PlaylistSongs` to `Songs (title, artistName)` makes sure that the song can be added to the playlist only if it exists in the table Songs.

b) No, there is nothing that prohibits this. The same owner can have more than one playlist with the same name since only the id is required to be unique.
Try for example to add the values `(PL1, jazz, Joe)` and `(PL2, jazz, Joe)` to `Playlist` and see what happens.

c) Yes: The `PRIMARY KEY (playlist, position)` in `PlaylistSongs` makes sure of this.

d) No, there is nothing that prohibits this. The following would be accepted: 2, 5, 9, ...

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 Queen	ABBA	232
2	Too late for love	John Lundvik	187

Solution:

See view `SongsInM123` in corresponding sql file.

1.3 Triggers

Create a trigger that enables directly building the playlist `M123` via the view defined in the previous question.

The insertion of `(position, song, artist, length)` in `M123` should be 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.

Solution:

See function `insertInM123` and trigger `insertPlayList` in the corresponding sql file.

2 Cell Phone Company

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.

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 (it is possibly that the same letter needs to be put in several places):

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

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

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

d) The monthly billing of a customer should contain both the id and the name of the customer, and it must be the sum of the fees of all the customers numbers. Make sure fees are non-negative!

e) If a customer is deleted, its connected subscriptions should be deleted automatically.

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

Solution:

See corresponding sql file.

3 Veterinary clinic admin system

(Previous exam question.)

Recall the vet clinic problem from exercises 1, extended with the table that was added for the bookings.

Clients (<u>name</u>, email, phonenr)
Pets (<u>chipnr</u>, name, owner, type, born)
owner → Client.name
type ∈ {dog, cat, rabbit}
Vets (<u>id</u>, name, phonenr, specialisation)
specialisation ∈ {dog, cat, rabbit}
Bookings (<u>emp</u>, chipnr, <u>bdate</u>, <u>time</u>, length)
emp → Vets.id
chipnr → Pets.chipnr
time ∈ {9,10,11,13,14,15}
length ∈ {30,60}
UNIQUE (chipnr, bdate, time)

Recall that chipnr and id are numbers that identify the pets and the employees respectively, and born is the year on which the pet was born. Assume name is enough to identify a client. The clinic only takes care of dogs, cats or rabbits. Consultations can only be booked at 9, 10, 11, 13, 14 or 15 hours. Length of consultations can be 30 or 60 minutes.

After running the clinic for a few years, the owners would like to make some improvements in their database. Propose a solution (meaning, the corresponding full SQL code) to each of the changes the company wants to make.

Recall that as a general rule, contraints, views and triggers should be applied in the appropriate order: if a constraint is enough just add it, and if a constraint or a view can adequately do the job, do not use a trigger!

a) Keep track of the name and phone number of all clients with at least 2 pets in the clinic.

b) If a pet is older than 5 years, one can only book 60 minutes consultation with a veterinarian that is specialised in that kind of animal.

c) Keep track of the name and specialisation of all veterinarians who had more than 1000 hours consultation in the clinic.

   Note than only consultations that already took place (meaning those with date today or earlier) should be counted, not the consultations that will take place in the future.

Solution:

See corresponding sql file.

4 Troubleshooting for Rotten Potatoes

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 a 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 titles, 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.

The website only allows to insert and delete reviews through `Reviews` view (see implementation code down below) so that users' premium subscriptions are checked, and "certified crisp" is updated accordingly.

Questions

a) Is their implementation *addressing* all the constraints described in the domain? If so, highlight where, otherwise, mention what is missing.

Solution:

YES. Line numbers below refer to the numbers in the exercise sheet, not in the sql file with the code.

line 3: login identifies users,
line 5: users' emails are unique,
line 18: movies are uniquely identified by their title and year of release,
line 19: movies genre is limited to the specified categories,
line 29: movies rates go from 1 to 5 stars,
line 33: only premium users can add comments,
lines 47-50: movies' score is computed as the average of its rates,
lines 58-79: only movies with score over 4 are added to certified crisp

b) 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?

Solution:

Function `insertReview` checks if the `newScore` is over 4 to insert the movie en `CertifiedCrisp`, however, it never checks if `newScore` is less or equal than 4 and the movie already has been classified as crisp, if that is the case, it should be deleted from `CertifiedCrisp`.

To solve this issue we add an `ELSE IF` statement between lines 75-76 in the exercise sheet; see added code for part b) in the corresponding sql file.

c) 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?

Solution:

Function `deleteReview` never updates `CertifiedCrisp` accordingly, to do so, we need to add a variable for the new score and perform the same checks that where done in `insertReview`—including what we added in the previous question. See added code for part c) in the corresponding sql file.

d) Is there redundancy in the information that is being stored? If so, where is it happening? and how can this be improved?

Solution:

YES. There is redundancy since table `CertifiedCrisp` can be easily modeled as a view, which indicates that we are storing information that is already in the database.

Once the table is replaced by a view (see view `NewCertifiedCrip` in the corresponding sql file), functions `insertReview` and `deleteReview` can be simplified by removing all the parts that took care of updating table `CertifiedCrisp`.

Implementation

```
-- Users' basic information
CREATE TABLE Users (
    login TEXT PRIMARY KEY,
    email TEXT NOT NULL,
    UNIQUE (email));

-- Premium users are an specialization of users
CREATE TABLE PremiumUsers (
    usr             TEXT NOT NULL REFERENCES Users,
    expiration_date DATE NOT NULL,
    PRIMARY KEY (usr));

-- Movies' basic information
CREATE TABLE Movies (
    title        TEXT    NOT NULL,
    year         INTEGER NOT NULL,
    genre        TEXT    NOT NULL,
    PRIMARY KEY (title, year),
    CHECK (genre IN ('Action', 'Sci-fi', 'Comedy', 'Drama')));

-- All users can rate a movie
CREATE TABLE Ratings (
    usr   TEXT    NOT NULL REFERENCES Users,
    title TEXT    NOT NULL,
    year  INTEGER NOT NULL,
    rate  INTEGER NOT NULL,
    PRIMARY KEY (usr, title, year),
    FOREIGN KEY (title, year) REFERENCES Movies,
    CHECK (rate IN (1,2,3,4,5)) -- Point system);

-- Only premium users are allowed to comment
CREATE TABLE Comments (
    usr     CHAR(5) NOT NULL REFERENCES PremiumUsers,
    title   TEXT    NOT NULL,
    year    INTEGER NOT NULL,
    comment TEXT    NOT NULL,
    PRIMARY KEY (usr, title, year),
    FOREIGN KEY (title, year) REFERENCES Movies);

-- Movies with score above 4.0
CREATE TABLE CertifiedCrisp (
    title   TEXT    NOT NULL,
    year    INTEGER NOT NULL,
    PRIMARY KEY (title, year),
    FOREIGN KEY (title, year) REFERENCES Movies);

CREATE VIEW MoviesScores AS
    SELECT title AS movie, year, COALESCE (AVG(rate),0) AS score
    FROM Movies NATURAL JOIN Ratings
    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 insertReview () 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 deleteReview () 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 InsertReviewTrigger
96    INSTEAD OF INSERT ON Reviews
97      FOR EACH ROW EXECUTE PROCEDURE insertReview();
98
99  CREATE TRIGGER DeleteReviewTrigger
100   INSTEAD OF DELETE ON Reviews
101     FOR EACH ROW EXECUTE PROCEDURE deleteReview();
```