Homework 6: Advanced SQL Features [100 points]

Due Date: Tuesday, Feb 28 (5:00 PM)

Submission

All HW assignments should contain both your student ID and your name and must be submitted online, formatted in SQL script form per the instructions provided on Piazza, through the associated dropbox on EEE. See the table below for the HW submission opportunities. Note that after 5 PM on Thursday no further HW submissions will be accepted. (We will be releasing the solution at that time.) Please strive to get all your work in on time! If possible, try to save the one dropped assignment for the end of the term when you are most likely to want/need it.

<table>
<thead>
<tr>
<th>Date / Time</th>
<th>Grade Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday, Feb 28 (5:00 PM)</td>
<td>Full credit will be available</td>
</tr>
<tr>
<td>Wednesday, Mar 1 (5:00PM)</td>
<td>20 points will be deducted</td>
</tr>
<tr>
<td>Thursday, Mar 2 (5:00 PM)</td>
<td>40 points will be deducted</td>
</tr>
</tbody>
</table>

Advanced SQL Features [100 pts]

Congratulations! TopicalBirds is happy with the SQL query work that you just finished doing for them. Now it’s time to get even more real – it’s time to use some of the advanced SQL goodies offered by MySQL to further improve their application (and its underlying data integrity).

Schema, Data, and Tools

TopicalBirds is still satisfied with the schema from HW’s #2 and #5. You can refer to the provided scripts to remind yourself of their schemas. You can also look at the relations’ schemas in MySQL Workbench when you are using it for this assignment. An updated data set for HW#6 will be provided for you to use in testing your queries. Reminders about how to load that schema and its associated sample data – and how to run queries – can be found in the instructions linked from HW #5’s entry on the course wiki page. You are to continue using MySQL for all of the tasks in this assignment and turn in the results per the provided instructions. (When creating stored procedures, you can use MySQL’s “Create a new ...” wizard, accessible as shown below.)
Okay, it’s time for you to get to work, as the Peacock known as “realDonaldTrump” is anxious to keep on chirping…!

1. Stored Procedures [40pts]

As a number of you have already observed, all is not right with the set of Chirps in the database. In fact, they look almost randomly generated…! (😊) The CTO of TopicalBirds.com is worried that Russian hackers have been targeting the company, and she wants to tighten up the security and integrity of the database to prevent further intrusions. As a step in that direction, you have been asked to write a pair of stored procedures to support the site’s web pages for chirping and for parroting chirps. Once these stored procedures are in place, the CTO will revoke everyone’s INSERT privileges on the Chirp table and make the stored procedures the only way to add new chirps by granting execution access (only) for the procedures instead.

a) [20 pts] Create and exercise a SQL stored procedure called **NewChirp(...)** that the application can use to add a newly created (non-parroted) chirp to the database. The stored procedure should automatically use the current date and time to set those fields of the new chirp, and it should automatically generate the new chirp’s number by adding one to the chirper’s previous highest chirp number.

i) [15pts] Using the following skeletal stored procedure code as inspiration, use MySQL’s stored procedure creation wizard by clicking the circled button as shown on the previous page to create the NewChirp(...) stored procedure.

```sql
CREATE PROCEDURE NewChirp(
    new_btag VARCHAR(30),
    loc_lat DECIMAL(10,6),
    loc_long DECIMAL(10,6),
    sentiment DECIMAL(2,1),
    content VARCHAR(255))
BEGIN
    DECLARE new_cno INT(11);
    SET new_cno = (SELECT ...);
    INSERT INTO Chirp (...)
    VALUES ( ...);
END;
```

ii) [5pts] Verify that your new stored procedure works properly by calling it as follows to add a fresh chirp and then running a query to show its after-effects:

```
CALL NewChirp ('realDonaldTrump', 0.0, 0.0, 1.0, 'Russia is our new best friend!');
SELECT * FROM Chirp WHERE btag LIKE 'real%';
```

b) Create and exercise a second SQL stored procedure called **ParrotChirp(...)** that the application can use to insert a parroting chirp into the database. This new stored procedure
should once again use the current date and time to set those fields of the new chirp, and it should again automatically generate the new chirp’s number by adding one to the chirper’s previous highest chirp number. However, since this is a parroted chirp, instead of taking as its arguments the new chirp’s sentiment and textual content, this procedure should take the key fields of the chirp to be parroted and copy the source chirp’s sentiment and text values into the corresponding fields of the newly created parrot chirp.

i) [15pts] Using the following skeletal stored procedure API as inspiration, use MySQL’s stored procedure creation wizard to create the ParrotChirp(...) stored procedure. (Hint: You will need to declare and use several variables in the body of this stored procedure in order to first determine the appropriate chirp number, sentiment, and text values for the newly created chirp.)

```
CREATE PROCEDURE ParrotChirp(
    new_btag VARCHAR(30),
    loc_lat DECIMAL(10,6),
    loc_long DECIMAL(10,6),
    orig_btag VARCHAR(30),
    orig_cno INT(11))
BEGIN
    ...
END;
```

ii) [5pts] Demonstrate that your stored procedure works properly by calling it to have the bird tagged as “swolf” parrot the 10th chirp from “realDonaldTrump” and examining the after-effects:

```
CALL ParrotChirp ('swolf', 0.0, 0.0, 'realDonaldTrump', 10);
SELECT * FROM Chirp WHERE btag = 'swolf';
```

2. Alter Table [20pts]

The CTO is wishing that the original design had made Ad a weak entity dependent on Watcher so that a Watcher’s Ads would also go away when a Watcher cancels their business agreement with TopicalBirds.com and the Watcher’s record is deleted. Note that the current design is set up to prevent a Watcher from being deleted if it has any Ads.

a) [5 pts] Write a DELETE statement to remove one of the Watchers that currently has Ads from the database. Execute the statement and show that the system is currently working as designed, thereby preventing the deletion.

b) [5 pts] Write and execute an ALTER TABLE statement to drop the foreign key constraint that’s causing the current behavior.

c) [10 pts] Write and execute an ALTER TABLE statement to add a new foreign key constraint that will cause a Watcher’s Ads to be deleted when the Watcher is deleted and then re-run the DELETE statement from (a) to show that the system now works in the desired new way.

3. Triggers [20pts]

The CTO has observed that some of the lesser-paid application developers seem incapable of following instructions. Instead of deleting birds at the User level in the type hierarchy, i.e., by targeting the User table, some of the developers continue to delete birds at the Bird level. As
a result, the CTO has asked you to safeguard the database by creating a row-level trigger on the Bird table that, if a Bird is deleted, cascades the deletion to the User table.

a) [15 pts] Use the CREATE TRIGGER statement in MySQL to define the desired trigger.

b) [5 pts] Write and execute a DELETE statement to test the proper operation of your trigger.

4. Views [20pts]
The fledgling data science team at TopicalBirds.com is setting out to study the chirping behavior of the site’s birds. Since the CTO trusts your SQL query-writing skills more than she does those of this new team, she has asked you to create a view called ParrotStatistics that they can write SELECT queries against to conduct their study. The fields of this view should be as follows:

1. Bird tag
2. Bird e-mail address
3. Bird age
4. Number of chirps
5-7. Min, max, and average chirp sentiments
8-9. Starting and ending date of the period when this bird was chirping

Note that while not all birds may have chirped, all birds should nonetheless appear in this view (including any birds with a chirp count of zero).

a) [15 pts] Create the desired view by writing an appropriate CREATE VIEW statement.

b) [5 pts] Show the usefulness of your view by writing a SELECT query against the view that lists the 3 nastiest birds in descending order of nastiness (where bird nastiness is measured based on average chirp sentiment).

5. Extra Credit [10pts]
The CTO has decided that the application should allow parroted chirps to either copy a source chirp verbatim or to provide their own content and sentiment and simply refer to the source chirp (e.g., as a way of issuing “commentary chirps” through parroting). Write and test a generalized stored procedure called AddChirp(...) that handles all cases: If the original chirp’s btag and cno are missing (i.e., are NULL), then the chirp being added should be inserted as a fresh chirp; otherwise this is a parroted chirp. For adding parroted chirps, if the chirp being added comes with its own sentiment and textual content, those values should be stored in the new chirp in addition to recording its parroting connection; if not, the new chirp’s sentiment and text should be copied from the specified original chirp. The desired API is:

CREATE PROCEDURE AddChirp(
    new_btag VARCHAR(30),
    loc_lat DECIMAL(10,6),
    loc_long DECIMAL(10,6),
    sentiment DECIMAL(2,1),
    content VARCHAR(255),
    orig_btag VARCHAR(30),
    orig_cno INT(11))
BEGIN
    ...

i) Create the specified stored procedure.

ii) Demonstrate that the new procedure indeed works properly for all three cases (new chirp, copy chirp, and commentary chirp). Note that a typical procedure call example for the last case might be:

```
CALL AddChirp ('HillaryClinton', 0.0, 0.0, -1.0, 'I told you so...', 'realDonaldTrump', 10);
```