CS 336 -- Principles of Information and Data Management

Fall 2022

Requirements Specification for the Database Programming Project

Introduction

You will use stored procedures to build API calls for each of the functions specified further in the description. This project does not require any UI, **it is purely a set of API calls and triggers**

It is an **individual project.**

You will have to install MySQL server locally on your computer. No need to build any web application. Just like in assignments 1 and 2, simply submit all procedures and triggers along with screen capture video showing the results of procedures and triggers in response to insertion, deletion or update, whenever appropriate.

Common questions: How do I get my Penna table back after I run procedures which change the table instance? Answer: Make a copy of Penna and call it OldPenna.

Election Results Database project

Part 1 (30%) – Powering simple interface to Penna table.

Write the following stored procedures

1. **API1(candidate, timestamp, precinct)** - Given a candidate C, timestamp T and precinct P, return how many votes did the candidate C have at T or largest timestamp T’ smaller than T, in case T does not appear in Penna. In case T < minimum timestamp in Penna, return 0 as number of votes for C. When T > maximum timestamp in Penna, return the last vote in this precinct for this candidate.

2. **API2(date)** - Given a date, return the candidate who had the most votes at the last timestamp for this date as well as how many votes he got. For example the last timestamp for 2020-11-06 will be 2020-11-06 23:51:43.
3. **API3(candidate)** - Given a candidate return top 10 precincts that this candidate won. Order precincts by the attribute: totalvotes and list TOP 10 in descending order of totalvotes.
4. **API4(precinct)** - Given a precinct**,** Show who won this precinct (Trump or Biden) as well as what percentage of total votes went to the winner.
5. **API5(string)** - Given a string s of characters, create a stored procedure which determines who won more votes in all precincts whose names contain this string s and how many votes did they get in total. For example, for s= 'Township', the procedure will return the name (Trump or Biden) who won more votes in union of precincts which have "Township" in their name as well as sum of votes for the winner in such precincts. **Hint**: Use Locate() function from mysql to find if value of a variable is substring of a column name.

Make sure you handle errors correctly – that is you have exception handling for wrong candidate name, wrong precinct or timestamp. In other words if someone makes a mistake submitting a wrong name of precinct, candidate, or wrong format of timestamp (say '05-11-2020'), your procedure should return "incorrect candidate', 'incorrect precinct' or incorrect timestamp. Incorrect timestamp means incorrect format, since as indicated in API1(), if format is correct, but timestamp does not exist in Penna, you can use approximation rules from instructions for AP1().

Part 2 (30%)

1) **newPenna()**: This stored procedure will create a table ***newPenna,*** showing for each precinct how many votes were added to totalvotes, Trump, Biden between timestamp T and the last timestamp directly preceding T. In other words, create a table like Penna but replace totalvotes with newvotes, Trump with new_Trump and Biden with new_Biden. Stored procedure with cursor is recommended.

For example the tuple in the new table newPenna:

newPenna('Hanover', '2020-11-06 19:10:53', 36, 27,9) states that 36 additional votes were added at timestamp 2020-11-06 19:10:53' since the last timestamp preceding it (which is 2020-11-06 16:26:51), 27 were added for Biden and 9 were added for Trump in Hanover precinct..

2) Switch(): This stored procedure will return list of precincts, which have switched their winner from one candidate in last 24 hours of vote collection (i.e 24 hours before the last Timestamp data was collected) and that candidate was the ultimate winner of this precinct.   The format of the table should be:

Switch(precinct, timestamp, fromCandidate, toCandidate) where fromCandidate is the candidate who was leading at timestamp in precinct, but he lost the lead to the toCandidate (who maintained that lead till the end)

For example

Switch('Hanover', '2020-11-07 16:41:11', Trump', 'Biden')

will mean that Biden took the lead from Trump on '2020-11-07 16:41:11' in Hanover Precinct and led all the way till the end of count in Hanover precinct.

Part 3 (10%)

Write SQL queries or stored procedures to check if the following patterns are enforced in the database:

a) The sum of votes for Trump and Biden cannot be larger than totalvotes
b) There cannot be any tuples with timestamps later than Nov 11 and earlier than Nov3
c) Totalvotes for any precinct and at any timestamp T > 2020-11-05 00:00:00,   will be larger or equal to totalvotes at T'<T where T'>2020-11-05 00:00:00 for that precinct. In other words the total votes of any precinct should not decrease with increasing timestamps within the day of 2020-11-05.

You should write SQL queries to verify the constraints and return TRUE or FALSE (in case constraint is not satisfied). Queries that don't return a boolean value won't be accepted.

Part 4 (30%)

4.1 Triggers and Update driven Stored Procedures

Create three tables *Updated Tuples, Inserted Tuples and Deleted Tuples.* All three tables should have the same schema as Penna and should store any tuples which were updated (store them as they were before the update), any tuples which were inserted, and any tuples which were deleted in their corresponding tables. The triggers should populate these tables upon each update/insertion/deletion. There will be one trigger for the update operation, one trigger for the insert operation and one trigger for the delete operation. Initially theses tables should be empty. In your video you should show how updates, insertions and deletions result in adding tuples to the *Updated Tuples, Inserted Tuples and Deleted Tuples*. Show multiple examples of deletions, insertions and updates, and display these tables after each of them.

4.2 MoveVotes()

MoveVotes(*Precinct, Timestamp, Candidate, Number_of_Moved_Votes*)

a) *Precinct* **–** *one of the existing precincts*

b) *Timestamp* must be existing timestamp. If *Timestamp* does not appear in Penna than *MoveVotes* should display a message "*Unknown Timestamp"*.

c) The *Number_of_Moved_Votes* parameter (always positive integer) shows the number of votes to be moved from the *Candidate* to another candidate and it cannot be larger than number of votes that the *Candidate* has at the Timestamp. If this is the case *MoveVotes* () should display a message "Not enough votes".

d) Of course if *CoreCandidate* is neither Trump nor Biden, *MoveVotes()* should say "Wrong Candidate".

Just as stated before each exception should lead to a message "wrong candidate name", "wrong precinct name' or not existing timestamp (this time we will not approximate like in API1()). In your video demonstrate some exceptions as well as the result of your procedure by running a query following call of MoveVotes().

MoveVotes() should move the Number_of_Moved_Votes from *CoreCandidate* to another candidate (there are only two) and do it not just for this Timestamp (the first parameter) but also for all T>Timestamp, that is all future timestamps in the given precinct.

For example MoveVotes(Red Hill, 2020-11-06 15:38:36,'Trump',100) will remove 100 votes from Trump and move it to Biden at 2020-11-06 15:38:36 and all future timestamps after that in the Red Hill precinct.

Submission Files

1) Submit all your work (queries, procedures, triggers)
2) A demo video to show how Part4 stored procedures work.

3) README.txt: a .txt file mentioning anything you want us to know about your application. You can omit this file in case you have nothing to mention.

DEADLINE: Monday, November 14 at 11:59pm

Good luck!