CS 336 -- Principles of Information and Data Management

Fall 2021

Requirements Specification for the Database Programming Project

Introduction

In this project, you are asked to build on Homework 3 and the election data from Pennsylvania (table Penna). In addition to queries, you will support updates, constraints, assertions, and triggers. You will use stored procedures to build API calls for each of the functions specified further in the description. The core (full credit) part of this project does not require any UI, **it is purely a set of API calls**. For extra credit you can build a simple UI (instructions will follow).

It is an **individual project.**

You will have to install your own web server that will host your web application as well as a MySQL server locally on your computer.

Election Results Database project

We will rely on data from the Penna table used in HW3.

Part 1 (10%)

1. Specify functional dependencies in Penna (no trivial FDs please).
2. Is Penna in BCNF? If not, decompose Penna into a BCNF scheme.

Part 2 (50%)

Create a set of stored procedures, which return answers to queries, either in tabular form or with plots (to better illustrate the results). These procedures will serve as API calls and will use the following three types of input: precinct, candidate, and timestamp. For each problem, the name and input of the procedure is listed first followed by what your procedure should show.

1) The Precinct

Given a precinct as input, build the following stored procedures:

a) *Winner(precinct)* Show who won this precinct, Trump or Biden. Show what percentage of total votes went to the winner. Show what the final number of total votes was in this precinct.

b) *RankALL(precinct*) Show the numerical rank of this precinct in terms of the number of total votes it received (at the last timestamp) among all precincts in the database

c) *RankCounty(precinct)* Show the numerical rank of this precinct in terms of the number of total votes it received (at the last timestamp) among all precincts in the *county* this precinct belongs to

d) *PlotPrecinct(precinct)* Show a timeseries graph plotting the three attributes *totalvotes*, *Trump*, *Biden* as the Timestamps progress (X-axis for Timestamp, Y-axis for votes) for the given precinct.

Additionally, build the following stored procedure:

e) *EarliestPrecinct(vote_count)* Show the first precinct to reach *vote_count* (i.e., the input) total votes as well as the timestamp when it occurred. If multiple precincts have reached the input value at the timestamp, return the precinct from among those with the most total votes.

2) **The Candidates**

Given a candidate as input, build the following stored procedures:

a) *PrecinctsWon(candidate)* List precincts the candidate won displaying both the vote difference between the two candidates and the total votes the candidate received. Order the list by the vote difference.

b) *PrecinctsWonCount(candidate)* Show the count of how many precincts the candidate won.

c) *PrecinctsFullLead(candidate)* List precincts which the candidate held a lead for at every timestamp

d) *PlotCandidate(candidate)* Show a timeseries plot for the candidate plotting the number of votes that candidate received at each timestamp

Additionally, build the following stored procedures without an input:

e) *PrecinctsWonCategory()* Create a stored procedure based on a defined precinct category which is a subset of all precincts (i.e. Townships, Wards, etc), it

is up to you how many of these procedures you want to define. For example, you might create a stored procedure **PrecinctsWonTownships()** which will use all the township precincts. For each procedure, return the name of the candidate who won that category of precincts as well as vote difference and the total votes of each candidate in that category.

3) **The Timestamp**

Given a timestamp as input, build the following stored procedures:

a) ***TotalVotes(timestamp, category)*** This stored procedure will take a category as input in the form of either ALL, Trump or Biden. The procedure should show an ordered list of precincts by either totalvote, Trump, or Biden (based on the input category) at that timestamp.

b) ***GainDelta(timestamp)*** Using the timestamp preceding the input timestamp, return DELTA representing the amount of time passed since that preceding timestamp as well as GAIN, the number of additional votes gained since that preceding timestamp. Also return the ratio GAIN/DELTA,

c) RankTimestamp() Rank all timestamps by the above GAIN/DELTA ratio in descending order

Additionally, build the following stored procedure:

d) ***VotesPerDay(day)*** Show votes for Biden, Trump, and total votes that occurred on just *day* (i.e., *day* should be an input between 03 and 11 corresponding to the day of the timestamp)

4) **Suspicious or Interesting Data**

Is there anything suspicious about the data, some form of “look what I have found” type information? Justify why it might be suspicious - do not just submit a random query. Show some interest in data and imagine that you are an election fraud investigator. Give a query, result - and explanation why it is suspicious.

Part 3 (10%)

In addition, you are asked to write SQL queries 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) Neither totalvotes, Trump's votes nor Biden's votes for any precinct and at any timestamp after 2020-11-05 00:00:00 will be smaller than the same attribute at the timestamp 2020-11-05 00:00:00 for that precinct.

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. **Notice that you will have just written SQL queries here**.

Part 4 (30%)

You should enforce integrity constraints in your database, by specifying the primary keys, as well as the foreign key (it will only be one).

Lastly, you should create a "modification" stored procedure where end users should be able to modify (insert/update/delete) to any table in your database. If an update is not accepted (e.g. because of a foreign key violation), you should provide the feedback message "Violates foreign key". If an update is valid, you can then show a success message.

For example, the procedure ***INSERT(tuple, table, constraint)*** which will return two values: "insertion accepted", "insertion rejected due to constraint violation". The constraints can be the primary key, foreign key, or any of the three assertions defined in the previous section.

You are free to update any table in this project anyway you want. The data is yours :-) You do not have to ask anyone for permission. Just do not delete all your data (it is easy to do 'Delete from TABLE').

4.1 Triggers and Update driven Stored Procedures

We wish to ensure that our database remains consistent even after modifications to our tables. Thus, write the following triggers for your tables to occur during modifications:

a) For each table in your database scheme you should create three log tables and three triggers. These tables will be called *Updated Tuples, Inserted Tuples and Deleted Tuples.* All three tables should have the same schema as the original table 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.

4.2 Stored Procedures (Notice we have changed the trigger to stored procedure – which is easier)

MoveVotes(*Precinct, Timest, CoreCandidate, Number_of_Moved_Votes*)

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

b) *Timest* must be existing timestamp. If *Timest* 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 *CoreCandidate* to another candidate and it cannot be larger than number of votes that the *CoreCandidate* 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".

After you are done with exceptions, you 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 full code, fds, schemes)
2) A demo video where you show updates (insertions/deletions) which succeed and which fail and demonstrate all the messaging. The demo should also show how the triggers 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: Tuesday, November 30 at 11:59pm

Good luck!