CS 336 -- Principles of Information and Data Management

Fall 2022

Requirements Specification for the Database Programming Project

Name: Nivesh Nayee

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

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.

```
USE `testDB`;
DROP procedure IF EXISTS `API1`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`API1`;
```

```
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `API1`(IN C varchar(50) ,
IN T varchar(50), IN P varchar(50))
BEGIN
        IF (select count(timestamp) from penna where timestamp = T) = 0
        THEN
        select C, if(C='Trump',Trump, Biden) as Votes from Penna
        where
        timestamp = (select max(timestamp) from penna where timestamp < T)
        &&
        precinct = P;
        ELSE
        select C, sum(if(C='Trump',Trump, Biden)) as Votes from Penna
   where
   timestamp = T
   and
   precinct = P;
   END IF;
END$$

DELIMITER ;
;
```

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.

```
USE `testDB`;
DROP procedure IF EXISTS `API2`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`API2`;
;
```

```
DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `API2`(date varchar(20))
BEGIN
select if(a.T > a.B, 'Trump', 'Biden') as Candidate, if(a.T > a.B, a.T, a.B ) as Votes
from
(
select sum(Trump) as T, sum(Biden) B from Penna
where
timestamp = (select max(timestamp) from Penna where timestamp like concat(date,'%') )
) as a;
END$$

DELIMITER ;
;
```

3. **API3(candidate)** - Given a candidate return top 10 precincts that this candidate win. Order precincts by total votes and list TOP 10 in descending order of totalvotes.

```
USE `testDB`;
DROP procedure IF EXISTS `API3`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`API3`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `API3`(Candidate
varchar(50))
BEGIN
select precinct,
if(candidate = 'Trump', (case when sum(Trump) > sum(Biden) then sum(Trump)
end) ,
(case when sum(Biden) > sum(Trump) then sum(Biden) end) )  as Votes from
Penna
```

```
group by precinct
order by Votes desc
limit 10;
END$$

DELIMITER ;
;
```

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.

```
USE `testDB`;
DROP procedure IF EXISTS `API4`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`API4`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `API4`(p varchar(100))
BEGIN
select sum(Trump) as 'Trump Votes', sum(Biden) as 'Biden Votes',
 if(sum(Trump) > sum(Biden), 'Trump', 'Biden') as Won,
if(
sum(Trump) > sum(Biden),
concat( ( (sum(Trump)/sum(totalvotes))*100 ),'%'),
concat( ( (sum(Biden)/sum(totalvotes))*100 ),'%')
) as Percentage
from Penna
where precinct like concat(p,'%')
group by precinct;
END$$

DELIMITER ;
;
```

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.

```
USE `testDB`;
DROP procedure IF EXISTS `API5`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`API5`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `API5`(s varchar(50))
BEGIN
select precinct, if(sum(Trump) > sum(Biden),'Trump', 'Biden') as Won,
if(sum(Trump) > sum(Biden),sum(Trump), sum(Biden)) as 'Total Votes'
from Penna
where
precinct like concat('%', s, '%')
group by precinct;
END$$

DELIMITER ;
;
```

Make sure you handle errors correctly – that is you have exception handling for wrong candidate name or wrong precinct or wrong date.

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

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..

- **NEWPENNA():**

```
        USE `testDB`;
DROP procedure IF EXISTS `newPenna`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`newPenna`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `newPenna`()
BEGIN
        DECLARE var_count int DEFAULT 0;
        DECLARE var_end_count int DEFAULT 0;

        declare nbiden int default 0;
   declare ntrump int default 0;
   declare ntotal int default 0;
   declare pre varchar(100) default 'hi';

   declare id_ int default 0;
   declare timestamp_ varchar(100) default Null;
   declare sate_ varchar(100) default Null;
   declare locality_ varchar(100) default Null;
   declare precinct_ varchar(100) default Null;
   declare geo_ varchar(100) default Null;
   declare totalvotes_ int default 0;
   declare biden_ int default 0;
   declare trump_ int default 0;
   declare filestamp_ varchar(100) default Null;

   DECLARE cur CURSOR for
   select * from Penna order by precinct, timestamp;

   DROP TABLE IF EXISTS newPenna;
   CREATE TABLE `newPenna` (
   `ID` INT NOT NULL,
   `Timestamp` DATETIME NULL,
   `state` VARCHAR(100) NULL,
   `locality` VARCHAR(100) NULL,
   `precinct` VARCHAR(100) NULL,
   `geo` VARCHAR(100) NULL,
   `newtotalvotes` INT NULL,
   `newBiden` INT NULL,
   `newTrump` INT NULL,
```

```
  `filestamp` VARCHAR(100) NULL
  );

  SET var_count = 0;
      select count(*) into var_end_count from Penna order by precinct, timestamp;

  open cur;
  Fetch next from cur into id_, timestamp_, sate_, locality_,
  precinct_, geo_, totalvotes_, biden_, trump_, filestamp_;

  while var_count < var_end_count
  do
              if pre != precinct_
              then
                      set pre = precinct_;
                      set nbiden = biden_;
                      set ntrump = trump_;
                      set ntotal = totalvotes_;
              else

                      if nbiden != biden_ || ntrump != trump_ || ntotal != totalvotes_
                      then
                      insert into newPenna
                      values
                      ( id_, timestamp_, sate_, locality_, precinct_, geo_,
                        totalvotes_-ntotal, biden_ - nbiden, trump_ - ntrump, filestamp_);
                      set nbiden = biden_;
                      set ntrump = trump_;
                      set ntotal = totalvotes_;
                      end if;
              end if;
     Fetch next from cur into id_, timestamp_, sate_, locality_,
     precinct_, geo_, totalvotes_, biden_, trump_, filestamp_;
     set var_count = var_count + 1;
  end while;
  close cur;
END$$
DELIMITER ;
;
```

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.

- **SWITCH():**

```
USE `testDB`;
DROP procedure IF EXISTS `Switch`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`Switch`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Switch`()
BEGIN
	DECLARE var_count int DEFAULT 0;
	DECLARE var_end_count int DEFAULT 0;

	declare winner varchar(50) default null;
   declare prevwinner varchar(50) default null;


   declare prevtime varchar(100) default null;
   declare prevprecinct varchar(100) default null;

   declare timestamp_ varchar(100) default Null;
   declare precinct_ varchar(100) default Null;
   declare biden_ int default 0;
   declare trump_ int default 0;

   DECLARE cur CURSOR for
   select timestamp, precinct, biden, trump from Penna order by precinct,
timestamp desc;
```

```
    DROP TABLE IF EXISTS switch;
    CREATE TABLE `switch` (
    `Timestamp` DATETIME NULL,
    `precinct` VARCHAR(100) NULL,
    `FromCandidate` varchar(100) Null,
    `ToCandidate` varchar(100) null
    );

    SET var_count = 0;
        select count(*) into var_end_count from Penna order by precinct,
timestamp desc;

    open cur;
    Fetch next from cur into timestamp_, precinct_, biden_, trump_;

    while var_count < var_end_count
    do
                set winner = if(biden_ > trump_, 'biden', 'trump');
                if prevprecinct = precinct_ && prevtime = timestamp_ && winner !=
prevwinner
                then
                        insert into switch
                        values
                        ( prevtime, prevprecinct, if(winner != 'trump', 'trump',
'biden'), if(winner = 'trump', 'trump', 'biden') );
                end if;
                set prevtime = timestamp_;
                set prevprecinct = precinct_;
                set prevwinner = winner;
                Fetch next from cur into timestamp_, precinct_, biden_, trump_;
                set var_count = var_count + 1;
        end while;
    close cur;
    select * from switch;
END$$

DELIMITER ;
;
```

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

- select if((sum(Trump) + sum(Biden)) <= sum(totalvotes), 'True' , 'False') AS Result, sum(Trump), sum(Biden), sum(totalvotes) from Penna;

b) There cannot be any tuples with timestamps later than Nov 11 and earlier than Nov3

- ```
  USE `testDB`;
  DROP procedure IF EXISTS `API6`;
  USE `testDB`;
  DROP procedure IF EXISTS `testDB`.`API6`;
  ;

  DELIMITER $$
  USE `testDB`$$
  CREATE DEFINER=`root`@`localhost` PROCEDURE `API6`()
  BEGIN
  if(exists(select timestamp from Penna
  where
  timestamp > '2020-11-03%'
  &&
  timestamp < '2020-11-11%') )
  then
  select 'False';
  else
  select 'True';
  end if;
  END$$

  DELIMITER ;
  ;
  ```

c) Totalvotes for any precinct and at any timestamp T > 2020-11-05 00:00:00, will be smaller than totalvotes at T'<T but T'>2020-11-05 00:00:00 for that precinct.

- ```
  select if(sum(p1.totalvotes) > sum(p.totalvotes), 'true', 'false') as t from penna p ,
  penna p1
  where
  p.precinct = p1.precinct
  &&
  p.timestamp > '2020-11-05 00:00:00'
  &&
  p1.timestamp > p.timestamp;
  ```

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.

AFTER INSERT:

```
DROP TRIGGER IF EXISTS `testDB`.`penna_AFTER_INSERT`;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` TRIGGER `penna_AFTER_INSERT`
AFTER INSERT ON `penna` FOR EACH ROW BEGIN
insert into insertedTuples
values(
new.ID, new.Timestamp, new.state, new.locality, new.precinct, new.geo, new.totalvotes,
new.Biden, new.Trump, new.filestamp
);
END$$
DELIMITER ;
```

BEFORE UPDATE:

```
DROP TRIGGER IF EXISTS `testDB`.`penna_BEFORE_UPDATE`;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` TRIGGER `penna_BEFORE_UPDATE`
BEFORE UPDATE ON `penna` FOR EACH ROW BEGIN
insert into updatedTuples
values
(
old.ID, old.Timestamp, old.state, old.locality, old.precinct, old.geo, old.totalvotes,
```

```
old.Biden, old.Trump, old.filestamp
);
END$$
DELIMITER ;
```

BEFORE DELETE:

```
DROP TRIGGER IF EXISTS `testDB`.`penna_BEFORE_DELETE`;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` TRIGGER `penna_BEFORE_DELETE`
 BEFORE DELETE ON `penna` FOR EACH ROW BEGIN
insert into deletedTuples
values
(
old.ID, old.Timestamp, old.state, old.locality, old.precinct, old.geo, old.totalvotes,
old.Biden, old.Trump, old.filestamp
);
END$$
DELIMITER ;
```

4.2 Stored Procedure simulating Trigger

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".

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.

MOVEVOTES():

```
USE `testDB`;
DROP procedure IF EXISTS `MoveVotes`;

USE `testDB`;
DROP procedure IF EXISTS `testDB`.`MoveVotes`;
;

DELIMITER $$
USE `testDB`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `MoveVotes`(in Precinct_ varchar(100), in timestamp_ varchar(100), in Candidate varchar(50), in numMovedVotes int )
BEGIN

        declare moveBiden int default if(Candidate = "Biden", numMovedVotes, 0);
        declare moveTrump int default if(Candidate = "Biden", 0, numMovedVotes);

    IF (select count(precinct) from penna where precinct = Precinct_) = 0
    THEN
    select concat('The ', Precinct_, ' not valid');
    ELSEIF (select count(timestamp) from penna where timestamp = timestamp_) = 0
    THEN
    select concat('The ', timestamp_ , ' is not Valid');
    ELSEIF Candidate != 'Biden' || Candidate != 'Trump'
    THEN
    select 'The candidate should be either Biden or Trump';
    ELSEIF numMovedVotes < 0
    THEN
    select 'Integer should be only Positive';
    ELSE
    update Penna
    set
    Biden = Biden - moveBiden,
    Trump = Trump + moveBiden,

    Trump = Trump - moveTrump,
    Biden = Biden + moveTrump
    where
```

```
    precinct = Precinct_
    and
    Timestamp > timestamp_;
    END IF;
END$$

DELIMITER ;
;
```

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!