ELECTION PROJECT

Vasanth Banumurthy

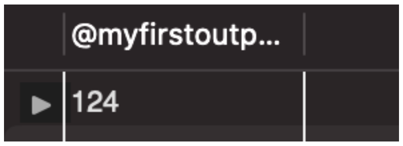
Part 1 (30%) – Powering simple interface to Penna table.

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.

```
DROP PROCEDURE IF EXISTS API1;


DELIMITER $$
CREATE PROCEDURE API1(IN C VARCHAR(30), IN T DATETIME, IN P VARCHAR(100), OUT V INT)
BEGIN
    DECLARE lastvote INT;
    DECLARE tvoteprime INT;
    DECLARE tvote INT;

    IF ( strcmp(C,"Biden" ) <> 0 AND strcmp(C, "Trump") <> 0)THEN
        SELECT "incorrect candidate";
        END IF;
    IF NOT EXISTS (SELECT DISTINCT Penna.precinct FROM Penna WHERE P = Penna.precinct) THEN
        SELECT "incorrect precinct";
        END IF;

    SELECT
    CASE
        WHEN C = 'Biden' THEN Penna.Biden
        WHEN C = 'Trump' THEN Penna.Trump
    END AS can
    INTO lastvote
    FROM testDB.Penna
    WHERE Penna.precinct = P AND Penna.Timestamp = (SELECT MAX(Penna.Timestamp)) LIMIT 1;

    SELECT Penna.totalvotes INTO tvoteprime
    FROM testDB.Penna
    WHERE Penna.precinct = P AND  Penna.Timestamp = (SELECT MAX(Penna.Timestamp) WHERE Penna.Timestamp < T)
LIMIT 1;

    SELECT
    CASE
        WHEN C = 'Biden' THEN Penna.Biden
        WHEN C = 'Trump' THEN Penna.Trump
    END AS votes
    INTO tvote
    FROM testDB.Penna
    WHERE Penna.precinct = P AND Penna.Timestamp = T;

    IF T < (SELECT MIN(Penna.Timestamp) FROM Penna) THEN SET V = 0;
    ELSEIF T > (SELECT MAX(Penna.Timestamp) FROM Penna) THEN SET V = lastvote;
    ELSEIF T IN (SELECT Penna.Timestamp FROM Penna) THEN SET V = tvote;
    ELSE SET V = tvoteprime;
    END IF;
END $$
DELIMITER ;
```

```
SET @myfirstoutput = 0;
CALL API1('Biden', '2020-11-07 03:01:22',
'Adams Township - Dunlo Voting Precinct',
@myfirstoutput);
SELECT @myfirstoutput;
```

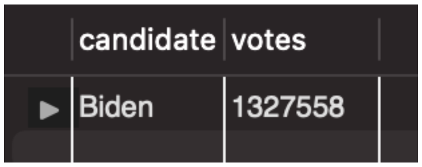


```
@myfirstoutp...
124
```

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.

```
DROP PROCEDURE IF EXISTS API2;
DELIMITER $$
CREATE PROCEDURE API2(IN T DATE)
BEGIN


SELECT
CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN 'Biden'
    ELSE 'Trump'
END AS candidate,
CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN SUM(Penna.Biden)
    ELSE SUM(Penna.Trump)
END AS votes
FROM Penna
WHERE Penna.Timestamp = (SELECT MAX(Penna.Timestamp) FROM Penna WHERE
DATE(Penna.Timestamp) = '2020-11-06');
END $$
DELIMITER ;
```

```
CALL API2('2020-11-06');
```

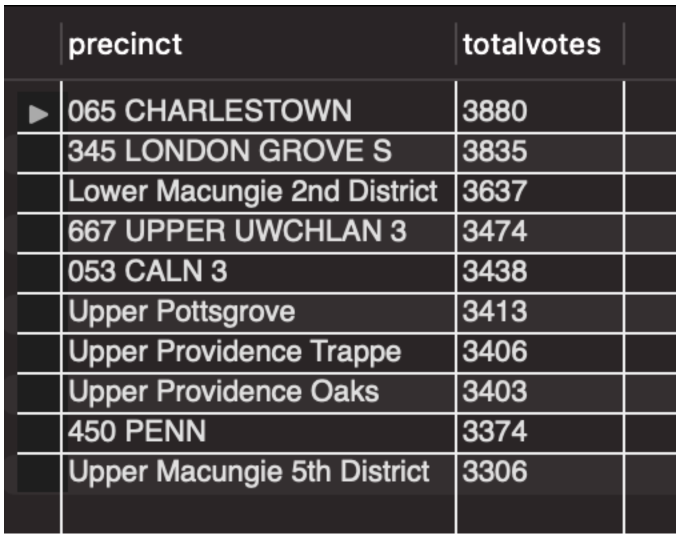


```
candidate  votes
Biden      1327558
```

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.

```
DROP PROCEDURE IF EXISTS API3;

DELIMITER $$
CREATE PROCEDURE API3(IN C VARCHAR(30))
BEGIN

IF ( strcmp(C,"Biden" ) <> 0 AND strcmp(C, "Trump") <> 0)THEN
SELECT "incorrect candidate";
END IF;

IF C = 'Biden'
THEN
(SELECT Penna.precinct, Penna.totalvotes
FROM Penna
WHERE Penna.Biden > (Penna.totalvotes / 2) AND DATE(Penna.Timestamp) = '2020-11-11'
GROUP BY Penna.precinct, Penna.totalvotes
ORDER BY Penna.totalvotes DESC LIMIT 10);
ELSE
(SELECT Penna.precinct, Penna.totalvotes
FROM Penna
WHERE Penna.Trump > (Penna.totalvotes / 2) AND DATE(Penna.Timestamp) = '2020-11-11'
GROUP BY Penna.precinct, Penna.totalvotes
ORDER BY Penna.totalvotes DESC LIMIT 10);
END IF;
END $$
DELIMITER ;
```

```
CALL API3('Biden');
```

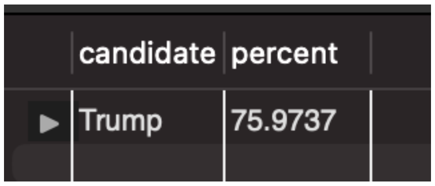


precinct	totalvotes
065 CHARLESTOWN	3880
345 LONDON GROVE S	3835
Lower Macungie 2nd District	3637
667 UPPER UWCHLAN 3	3474
053 CALN 3	3438
Upper Pottsgrove	3413
Upper Providence Trappe	3406
Upper Providence Oaks	3403
450 PENN	3374
Upper Macungie 5th District	3306

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.

```
DROP PROCEDURE IF EXISTS API4;
DELIMITER $$
CREATE PROCEDURE API4(IN P VARCHAR(150))
BEGIN

 IF NOT EXISTS (SELECT DISTINCT Penna.precinct FROM Penna WHERE P =
Penna.precinct) THEN
        SELECT "incorrect precinct";
        END IF;
SELECT
    CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN 'Biden'
    ELSE 'Trump'
    END AS candidate,

    CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN (SUM(Penna.Biden) / SUM(Penna.totalvotes) * 100)
    ELSE (SUM(Penna.Trump) / SUM(Penna.totalvotes) * 100)
    END AS percent
FROM Penna
WHERE Penna.precinct = P;
END $$
DELIMITER ;
```

```
CALL API4('Adams Township - Elton Voting Precinct');
```

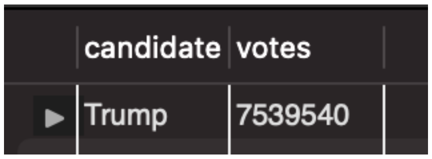


```
  candidate  percent
▶ Trump      75.9737
```

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.

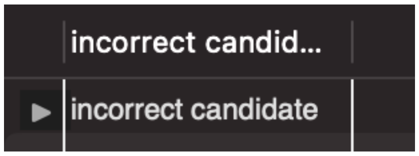
```
DROP PROCEDURE IF EXISTS API5;
DELIMITER $$
CREATE PROCEDURE API5(IN P VARCHAR(150))
BEGIN
    SELECT
    CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN 'Biden'
    ELSE 'Trump'
END AS candidate,
    CASE WHEN
    (
        SUM(Penna.Biden) > SUM(Penna.Trump)
    )
    THEN SUM(Penna.Biden)
    ELSE SUM(Penna.Trump)
END AS votes
FROM Penna
WHERE (SELECT(LOCATE(P, Penna.precinct) <> 0));
END $$
DELIMITER ;
```

```
CALL API5('Township');
```

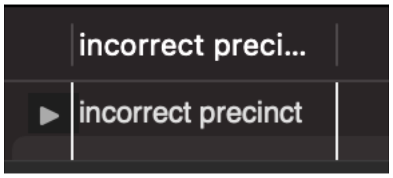


candidate	votes
Trump	7539540

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().



incorrect candid...
incorrect candidate



incorrect preci...
incorrect precinct

```
Penna.Timestamp = (SELECT MAX(Penna.Timestamp) WHERE
Penna.Timestamp < T)
```

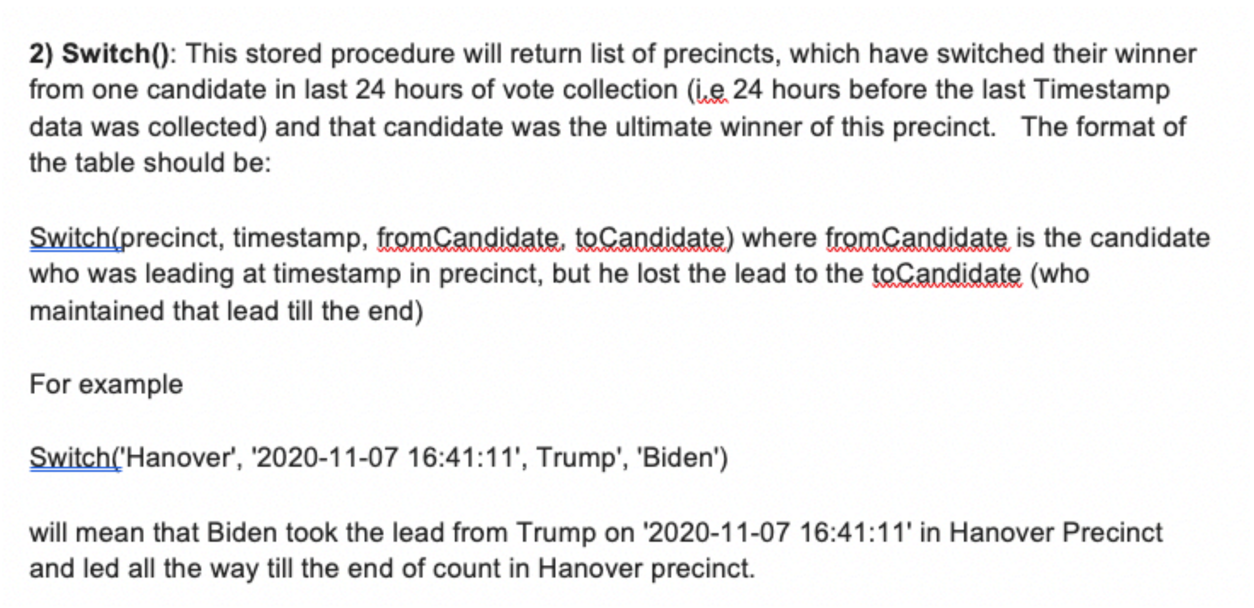
Part 2 (30%)

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..

```
CREATE TABLE newPenna
(
    new_Precinct VARCHAR(45),
    new_Timestamp DATETIME,
    new_Votes INT,
    new_Biden INT,
    new_Trump INT
);

INSERT INTO newPenna(new_Precinct,new_Timestamp)
SELECT Penna.precinct, Penna.Timestamp
FROM Penna;

DROP PROCEDURE IF EXISTS newPenna;
DELIMITER $$
CREATE PROCEDURE newPenna()
BEGIN
DECLARE var_count int DEFAULT 0;
DECLARE var_end_count int DEFAULT 0;
DECLARE P VARCHAR(150);
DECLARE T DATETIME;

DECLARE cur CURSOR FOR
SELECT newPenna.new_Precinct, newPenna.new_Timestamp
FROM newPenna;



SET var_count = 0;
SELECT count(*) INTO var_end_count FROM newPenna;

OPEN cur;

WHILE var_count < var_end_count DO
FETCH NEXT FROM cur INTO P, T;
INSERT INTO newPenna(new_Votes, newBiden, newTrump)
SELECT Penna.totalvotes -
(SELECT Penna.totalvotes
FROM Penna
WHERE Penna.Timestamp =
(SELECT MAX(Penna.Timestamp) WHERE Penna.Timestamp < T) LIMIT 1),
Penna.Biden - (SELECT Penna.Biden
FROM Penna
WHERE Penna.Timestamp =
(SELECT MAX(Penna.Timestamp) WHERE Penna.Timestamp < T) LIMIT 1),
Penna,Trump - (SELECT Penna.Trump
FROM Penna
WHERE Penna.Timestamp =
(SELECT MAX(Penna.Timestamp) WHERE Penna.Timestamp < T) LIMIT 1)
FROM Penna
WHERE Penna.precinct = P AND Penna.Timestamp = T;

SET var_count = var_count + 1;
END WHILE;
CLOSE cur;
END $$
DELIMITER ;



CALL newPenna();
```



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:

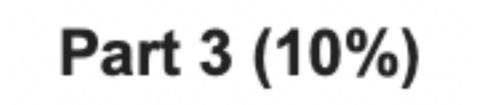
Switch(precinct, timestamp, fromCandidate, toCandidate) where fromCandidate is the candidate who was leading at timestamp in precinct, but he lost the lead to the toCandidate (who maintained that lead till the end)

For example

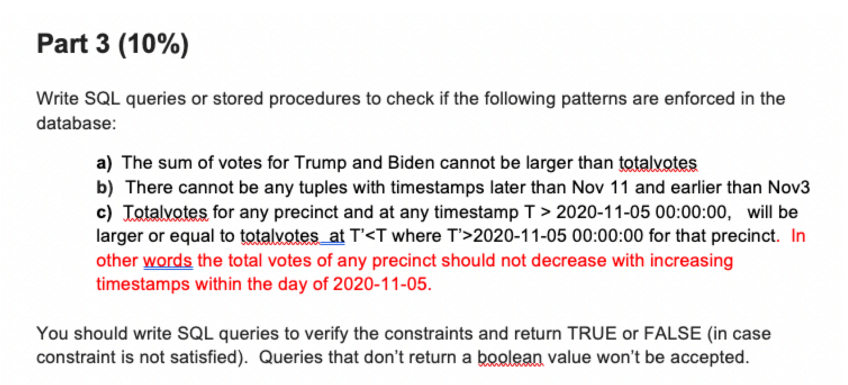
Switch('Hanover', '2020-11-07 16:41:11', Trump', 'Biden')

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.

```
SELECT p1.precinct
FROM ( SELECT * FROM Penna WHERE Penna.Timestamp >
'2020-11-11 00:00:00') AS p1,
( SELECT * FROM Penna WHERE Penna.Timestamp >
'2020-11-11 00:00:00') AS p2
WHERE p1.precinct = p2.precinct AND p1.Trump > p1.Biden
AND p2.Trump < p2.Biden
UNION
SELECT p1.precinct
FROM ( SELECT * FROM Penna WHERE Penna.Timestamp >
'2020-11-11 00:00:00') AS p1,
( SELECT * FROM Penna WHERE Penna.Timestamp >
'2020-11-11 00:00:00') AS p2
WHERE p1.precinct = p2.precinct AND p1.Trump < p1.Biden
AND p2.Trump > p2.Biden;
```



Part 3 (10%)



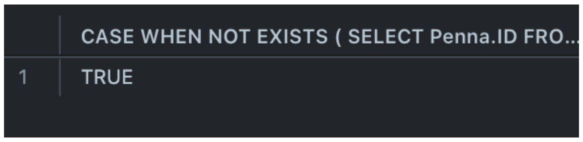
Part 3 (10%)

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.

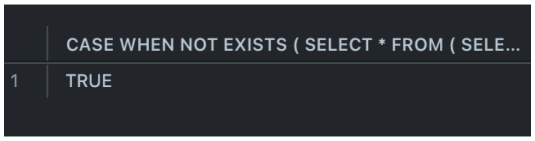
You should write SQL queries to verify the constraints and return TRUE or FALSE (in case constraint is not satisfied). Queries that don't return a boolean value won't be accepted.

```
-- A --

SELECT
    CASE WHEN NOT EXISTS
    (
        SELECT Penna.ID
        FROM Penna
        GROUP BY Penna.ID
        HAVING (SUM(Penna.Biden) + SUM(Penna.Trump)) > SUM(Penna.totalvotes)
    )
    THEN 'TRUE'
    ELSE 'FALSE'
END;
```



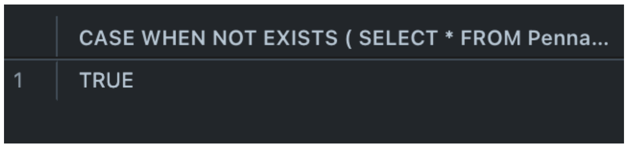
```
   CASE WHEN NOT EXISTS ( SELECT Penna.ID FRO...
1  TRUE
```

```
-- B --

SELECT
    CASE WHEN NOT EXISTS
    (
        SELECT *
        FROM Penna
        WHERE DATE(Penna.Timestamp) < '2020-11-03' OR DATE(Penna.Timestamp) >
'2020-11-11'
    )
    THEN 'TRUE'
    ELSE 'FALSE'
END;
```



```
   CASE WHEN NOT EXISTS ( SELECT * FROM ( SELE...
1  TRUE
```

```
-- C --

SELECT
    CASE WHEN NOT EXISTS
    (
        SELECT *
        FROM
        (
        SELECT Penna.totalvotes as tvotes,
        LEAD(totalvotes) OVER(ORDER BY totalvotes) as next_vote_count
        FROM Penna
        WHERE DATE(Penna.Timestamp) = '2020-11-05'
        ) as data
        WHERE next_vote_count < tvotes
    )
    THEN 'TRUE'
    ELSE 'FALSE'
END;
```



```
   CASE WHEN NOT EXISTS ( SELECT * FROM Penna...
1  TRUE
```

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.

```
SET SQL_SAFE_UPDATES = 0;
```

```
DROP TABLE IF EXISTS Updated_Tuples;
CREATE TABLE Updated_Tuples LIKE Penna;

DROP TABLE IF EXISTS Inserted_Tuples;
CREATE TABLE Inserted_Tuples LIKE Penna;

DROP TABLE IF EXISTS Deleted_Tuples;
CREATE TABLE Deleted_Tuples LIKE Penna;
```

TESTING

```
DROP TRIGGER update_op;
DELIMITER $$
CREATE TRIGGER update_op BEFORE UPDATE ON Penna
FOR EACH ROW
BEGIN
INSERT INTO Updated_Tuples
SELECT * FROM Penna
WHERE Penna.ID = OLD.ID AND Penna.Timestamp =
OLD.Timestamp;
END  $$
DELIMITER ;
```

```
SELECT *
FROM Updated_Tuples;

DESCRIBE Penna;


UPDATE Penna
SET Penna.state = 'NJ'
WHERE Penna.ID = 1 AND Penna.Timestamp =
'2020-11-04 03:58:36';

SELECT *
FROM Penna
WHERE Penna.ID = 1;
```

TESTING

```
DROP TRIGGER insert_op;
DELIMITER $$
CREATE TRIGGER insert_op AFTER INSERT ON Penna
FOR EACH ROW
BEGIN
INSERT INTO Inserted_Tuples
VALUES(NEW.ID, NEW.Timestamp, NEW.state,
NEW.locality, NEW.precinct, NEW.geo,
NEW.totalvotes,NEW.Biden, NEW.Trump,
NEW.filestamp);
END  $$
DELIMITER ;
```

```
INSERT INTO Penna
VALUES (1, '2020-11-04 03:58:35', 'NJ',
'Middlesex', 'Monroe Twp.', '42021-MON TWP', 35,
30, 5, 'NOVEMBER_04_2020_013100.json');

SELECT *
FROM Inserted_Tuples;
```

TESTING

```
DROP TRIGGER delete_op;
DELIMITER $$
CREATE TRIGGER delete_op BEFORE DELETE ON Penna
FOR EACH ROW
BEGIN
INSERT INTO Deleted_Tuples
SELECT * FROM Penna
WHERE Penna.ID = OLD.ID AND Penna.Timestamp =
OLD.Timestamp;
END  $$
DELIMITER ;
```

```
DELETE FROM Penna
WHERE Penna.precinct = 'Monroe Twp.';

SELECT *
FROM Deleted_Tuples;

SELECT *
FROM Penna
WHERE Penna.precinct = 'Monroe Twp.';
```

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".

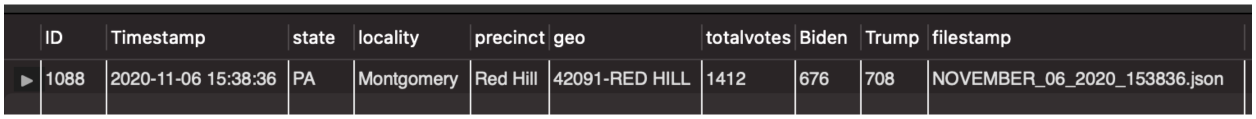
d) Of course if *CoreCandidate* is neither Trump nor Biden, *MoveVotes()* should say "Wrong Candidate".

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.

```
SELECT *
FROM Penna
WHERE Penna.precinct = 'Red Hill' AND
Penna.Timestamp = '2020-11-06 15:38:36';
```



ID	Timestamp	state	locality	precinct	geo	totalvotes	Biden	Trump	filestamp
1088	2020-11-06 15:38:36	PA	Montgomery	Red Hill	42091-RED HILL	1412	676	708	NOVEMBER_06_2020_153836.json

```
DELIMITER $$
CREATE PROCEDURE MoveVotes(
    IN P VARCHAR(150),
    IN T TIMESTAMP,
    IN C VARCHAR(100),
    IN Moved_vote INT)
BEGIN

    IF ( strcmp(C,"Biden") <> 0 AND strcmp(C, "Trump") <> 0 )THEN
        SELECT "incorrect candidate";
    END IF;
    IF NOT EXISTS (SELECT DISTINCT Penna.precinct FROM Penna WHERE P =
Penna.precinct) THEN
        SELECT "incorrect precinct";
    END IF;
    IF NOT EXISTS (SELECT DISTINCT Penna.Timestamp FROM Penna WHERE T =
Penna.Timestamp) THEN
        SELECT "incorrect timestamp";
    END IF;
    IF strcmp(C, "Biden") = 0 THEN
        UPDATE Penna
        SET
            Penna.Biden = Penna.Biden - Moved_vote,
            Penna.Trump = Penna.Trump + Moved_vote
        WHERE Penna.precinct = P and Penna.Timestamp= T;
    END IF;
    IF strcmp(C, "Trump") = 0 THEN
        UPDATE Penna
        SET
            Penna.Biden = Penna.Biden + Moved_vote,
            Penna.Trump = Penna.Trump - Moved_vote
        WHERE Penna.precinct = P AND Penna.Timestamp = T;
    END IF;
END;
DELIMITER ;
```

Any Questions?

Reach out to vb331@scarletmail.rutgers.edu