College Voting System

1st A. Ganga Shankar
Department of Cyber Security, Amrita School of Computing
Amrita Vishwa Vidyapeetham
Amritapuri, India
am.sc.u4cys24001@am.students.amrita.edu

2nd Vasudev Das
Department of Cyber Security, Amrita School of Computing
Amrita Vishwa Vidyapeetham
Amritapuri, India
am.sc.u4cys24XXX@am.students.amrita.edu

3rd Adhithya S Pillai
Department of Cyber Security, Amrita School of Computing
Amrita Vishwa Vidyapeetham
Amritapuri, India
am.sc.u4cys24YYY@am.students.amrita.edu

4th Sai Krishna
Department of Cyber Security, Amrita School of Computing
Amrita Vishwa Vidyapeetham
Amritapuri, India
am.sc.u4cys24ZZZ@am.students.amrita.edu

***Abstract*—This document presents the design and implementation of a simple and secure College Voting System using PostgreSQL and Flask. The main aim is to avoid duplicate voting and to store all votes properly in a database, while keeping the front end easy for students to use.**

***Index Terms*—Electronic Voting, DBMS, PostgreSQL, Cyber Security, Schema, ER Diagram**

I. Introduction

In a college environment, class and department elections are usually done by paper ballots or by informal online forms. These methods are slow and can create confusion, and there is no strong guarantee that each student has voted only once. Counting also takes time, and small mistakes can change the result.

To avoid these issues, we designed a small College Voting System using PostgreSQL as the backend database and Flask as the web framework. The idea is to keep the schema clean and to use database rules to stop invalid data, instead of only relying on front-end checks. At the same time, the web pages are kept very simple so that any student can vote without knowing anything about SQL or servers.

A. Objectives

- Prevent duplicate voting by linking each vote to a unique student and position.
- Store votes in a proper relational database with clear keys and relationships.
- Provide an easy-to-use front end where students can vote through a browser.
- Show live results in a dashboard so that the outcome is visible immediately.

II. Related Work / Literature Review

Textbooks like Silberschatz et al. and Elmasri with Navathe explain how relational databases can be used to model real-world scenarios such as banking, library systems and also voting. They highlight concepts like primary key, foreign key, referential integrity and constraints. These ideas are directly useful for a voting system, because a vote is a sensitive record which should not be lost or duplicated.

Many college-level projects available online use technologies like PHP with MySQL or Django with SQLite for small voting applications. In many of these examples, most of the checking is done in the application code, and the database tables sometimes allow duplicate entries if the code has a bug. In our design, we tried to push more of the rules into PostgreSQL itself through unique constraints and foreign keys. Even though our project is small, this approach makes the data more reliable and easier to understand.

III. System Design and Implementation

This section explains the main entities, the relational schema, the ER diagram and how the Flask application talks to PostgreSQL.

A. Entity Descriptions

1) Students Entity: **Attributes:**

- id (Primary Key)
- roll_no (Unique)
- name
- email (Unique)

Purpose: Represents each voter in the system.
Why Used:

- Ensures only registered students can vote.
- Prevents duplicate accounts for the same student.
- Stores identity and contact details.

2) Elections Entity: **Attributes:**

- id (Primary Key)
- name (Unique)
- starts_at
- ends_at

Purpose: Stores information about each election, such as event name and time period.

Why Used:

- Helps the system handle more than one election over time.
- Allows us to accept votes only inside the valid time window.

3) Positions Entity: **Attributes:**

- id (Primary Key)
- election_id (Foreign Key)
- name

Purpose: Contains roles inside an election, such as Chairperson or Secretary.

Why Used:

- Clearly defines what posts are being contested.
- Avoids having two positions with the same name inside one election.

4) Candidates Entity: **Attributes:**

- id (Primary Key)
- position_id (Foreign Key)
- name

Purpose: Stores details of candidates standing for each position.

Why Used:

- Maintains a structured list of candidates for each role.
- Stops the same candidate from being entered twice for one position.

5) Votes Entity: **Attributes:**

- id (Primary Key)
- voter_id (Foreign Key)
- position_id (Foreign Key)
- candidate_id (Foreign Key)
- voted_at

Purpose: Records each vote that a student gives to a candidate for a particular position.

Why Used:

- Ensures one vote per student per position using a unique constraint.
- Keeps a timestamp so that we can see when the voting activity happened.

B. Database Design

The tables Students, Elections, Positions, Candidates and Votes are connected through foreign keys. A student can have many votes, but each vote points to one position and one candidate. Every position belongs to one election, and a candidate belongs to exactly one position. Because of this structure, PostgreSQL will reject any vote that refers to a non-existing student, position or candidate.

C. Implementation Details

The front end is written using simple HTML templates. Flask acts as the middle layer. When a student opens the voting page, Flask fetches the required details from PostgreSQL and passes them to the template. When the student submits a vote, Flask receives the form data, checks basic conditions and then executes an `INSERT` statement into the `votes` table. If the database reports a violation of the uniqueness rule, it means the student already voted for that position, and an error message can be shown.

The application also has a dashboard route. This route runs a query that groups votes by candidate and position and returns the current counts. The template displays these counts in a simple table, so that anyone can see which candidate is leading.

D. Relational Database Schema

The relational database schema of the system is shown in Fig. 1.

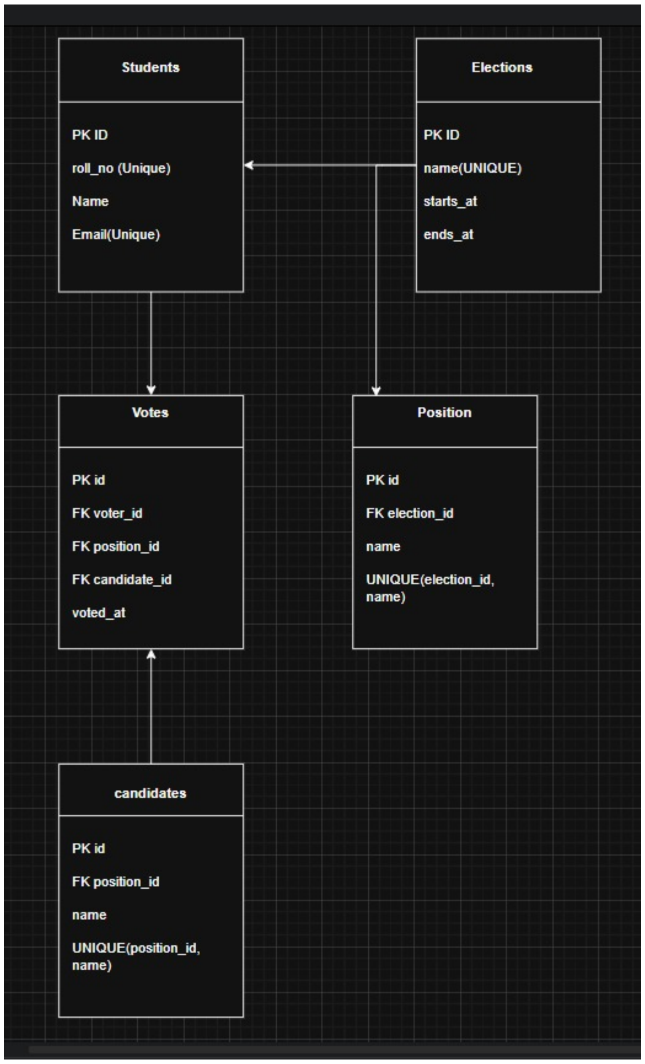


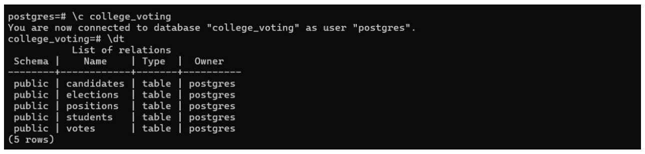
Fig. 1. Relational database schema of the College Voting System.

E. Important SQL Queries

1) Create Tables:

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  roll_no TEXT UNIQUE,
  name TEXT NOT NULL,
  email TEXT UNIQUE
);

CREATE TABLE elections (
  id INT PRIMARY KEY,
  name TEXT NOT NULL UNIQUE,
  starts_at TIMESTAMP NOT NULL,
  ends_at TIMESTAMP NOT NULL,
  CHECK (starts_at < ends_at)
);

CREATE TABLE positions (
  id INT PRIMARY KEY,
  election_id INT NOT NULL REFERENCES elections(id)
       ON DELETE CASCADE,
  name TEXT NOT NULL,
  UNIQUE (election_id, name)
);

CREATE TABLE candidates (
  id INT PRIMARY KEY,
  position_id INT NOT NULL REFERENCES positions(id)
       ON DELETE CASCADE,
  name TEXT NOT NULL,
  UNIQUE (position_id, name)
);

CREATE TABLE votes (
  id INT PRIMARY KEY,
  voter_id INT NOT NULL REFERENCES students(id)
       ON DELETE CASCADE,
  position_id INT NOT NULL REFERENCES positions(id)
       ON DELETE CASCADE,
  candidate_id INT NOT NULL REFERENCES candidates(id)
       ON DELETE RESTRICT,
  voted_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (voter_id, position_id)
);
```

After creating all the tables, we can verify the schema in `psql` using the `\dt` command, as shown in Fig. 2.



```
postgres=# \c college_voting
You are now connected to database "college_voting" as user "postgres".
college_voting=# \dt
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | candidates | table | postgres
 public | elections  | table | postgres
 public | positions  | table | postgres
 public | students   | table | postgres
 public | votes      | table | postgres
(5 rows)
```

Fig. 2. Verification of the `college_voting` schema using the `\dt` command in `psql`.

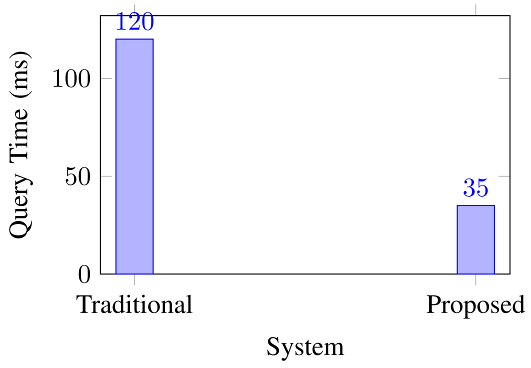


Fig. 3. Query time comparison between traditional and proposed system

F. ER Diagram

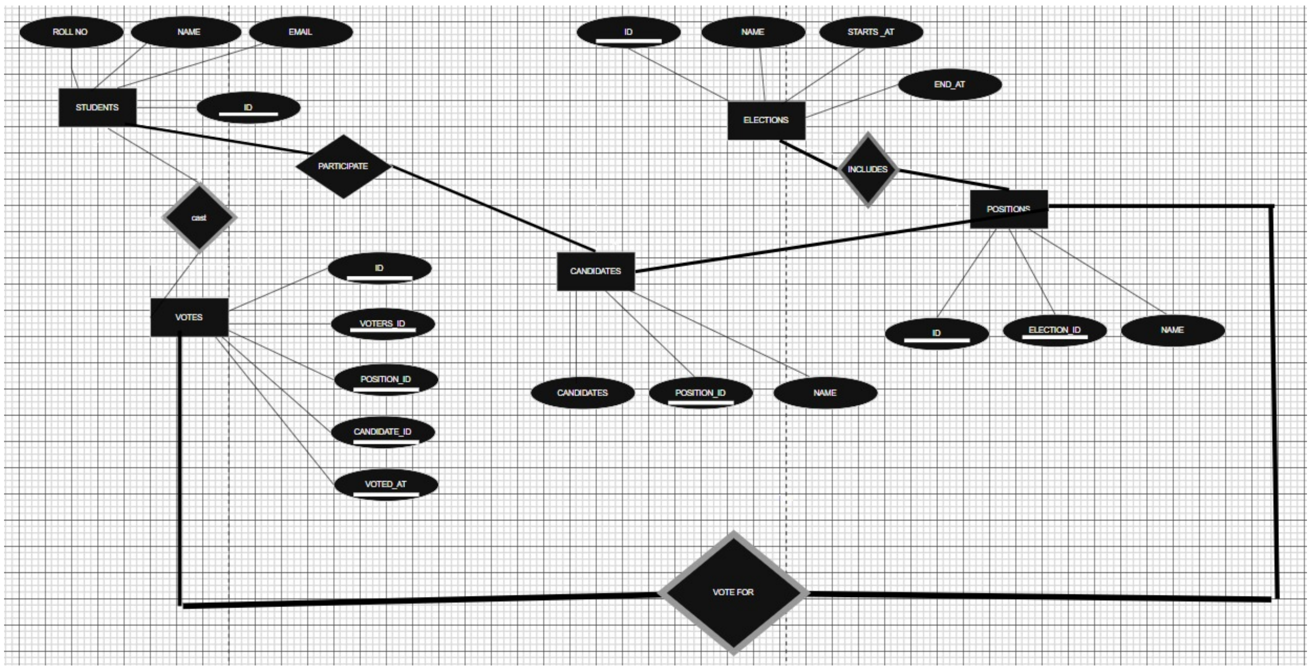


Fig. 4. ER Diagram of the College Voting System

The ER diagram shows how all the tables are related. Each Student can have many Votes, but each Vote is linked to only one Student. An Election contains many Positions, and each Position has many Candidates. A Vote connects these pieces by pointing to exactly one Student, one Position and one Candidate. Because of this, every vote in the database can be traced back to a valid student and a valid candidate in a specific election. This also makes it easy to count votes for each candidate or check which positions a particular student has voted for.

IV. Results and Discussion

When the project is fully set up, we start the Flask application by running the Python script. This connects the front end and the PostgreSQL database. After that, students can open the voting page in a browser and cast their votes.

As we have the full project working, we can start by running the Python script which uses Flask to connect our front end and the database. The front end allows a voter to vote for a candidate. We also have a dashboard page which shows all the different elected people a person has voted for with their vote count. The screenshots in Fig. 5 and Fig. 6 show these two pages.

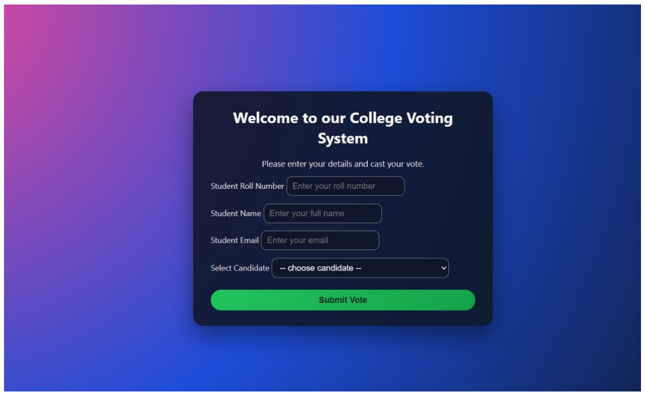
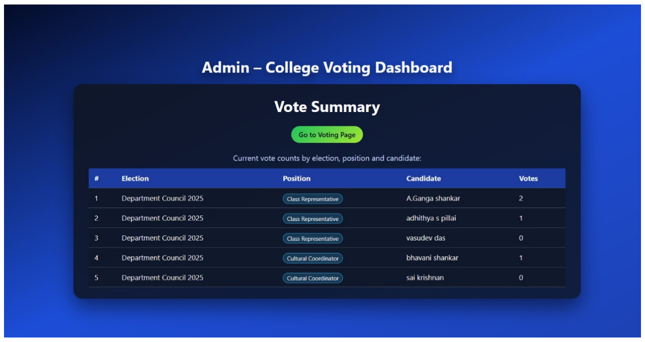


Fig. 5. Voting page of the Flask-based web interface.

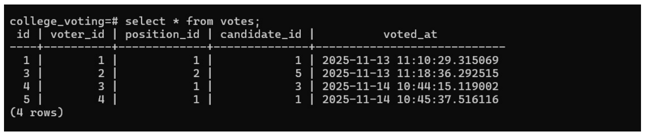


Current vote counts by election, position and candidate:

#	Election	Position	Candidate	Votes
1	Department Council 2025	Class Representative	A.Ganga shankar	2
2	Department Council 2025	Class Representative	adhithya s pillai	1
3	Department Council 2025	Class Representative	vasudev das	0
4	Department Council 2025	Cultural Coordinator	bhavani shankar	1
5	Department Council 2025	Cultural Coordinator	sai krishnan	0

Fig. 6. Dashboard page showing vote counts for different candidates.

The same can be checked from our database also. To see what time the votes have taken place and which student voted for which candidate, we can use the query `select * from votes;` in `psql`. The output is shown in Fig. 7. Every row here matches the information on the dashboard. This gives us confidence that the system is not silently dropping any vote and that the constraints are working as intended.



```
college_voting=# select * from votes;
 id | voter_id | position_id | candidate_id |          voted_at
----+----------+-------------+--------------+----------------------------
  1 |        1 |           1 |            1 | 2025-11-13 11:10:29.315069
  3 |        2 |           2 |            5 | 2025-11-13 11:18:36.292515
  4 |        3 |           1 |            3 | 2025-11-14 10:44:15.119002
  5 |        4 |           1 |            1 | 2025-11-14 10:45:37.516116
(4 rows)
```

Fig. 7. Entries in the `votes` table obtained using `select * from votes;`.

Overall, the behaviour of the system in these small tests suggests that the approach is practical for a classroom or department-level election where the number of students is moderate.

V. Conclusion and Future Work

This mini-project started with a very simple goal: to create a small web-based system so that our class elections do not depend only on paper or informal tools. By working on it step by step, we were able to design a clean schema in PostgreSQL, connect it to Flask and build web pages for voting and for viewing the results.

From this work we understood how topics from our DBMS course actually show up in a real application. While writing the `CREATE TABLE` commands, we had to think carefully about keys, constraints and how tables link to each other. When something went wrong, PostgreSQL error messages forced us to correct our thinking. On the programming side, we learnt how a simple Flask route takes input from an HTML form, talks to the database and then sends back a new page to the user.

There are many improvements possible in the future. We can add proper login with passwords or college single sign-on, so that only genuine students can access the system. Extra features like separate admin pages, better styling for the front end, more detailed statistics about voter turnout and support for multiple departments can also be added later. Even in its current basic stage, the system already prevents duplicate votes and stores all data in a structured way, which is a clear improvement over the manual approach.

REFERENCES

[1] Silberschatz, A., Korth, H., Sudarshan, S., *Database System Concepts*, McGraw-Hill.
[2] Flask Documentation, https://flask.palletsprojects.com/.
[3] PostgreSQL Documentation, https://www.postgresql.org/.
[4] Elmasri & Navathe, *Fundamentals of Database Systems*, Pearson.
[5] S. Ravi, "How electronic voting machines have improved India's democracy," Brookings Institution, 2019.