ADVANCED DATABASES (CMP7214) REPORT

ONLINE VOTING MANAGEMENT SYSTEM

Module Coordinator
Dr. Parnia Samimi

Submitted By
Meera Mohan Mullamkuzhy (22185279)

MSc Big Data Analytics

Faculty of Computing, Engineering and the Built Environment

9 January 2023

Contents

List of Figures

List of Tables

1 DOMAIN DESCRIPTION

Online Voting Management is a system that allows eligible voters to cast their secure and secret ballot electronically.It is done by casting a ballot through a digital system over the internet. The system is handled by the Election Commission who administrates the voting process. Online voting management system is beneficial for the Election Commission as well as the voters as they can to get to know the candidate background and choose wisely.

We would be using MySQL Workbench for the database implementation.

The advantages of the system are as follows

- Fast and easy way of conducting election.
- Voters can view background of each candidate.
- Candidate can present themselves to voters.
- Election Commission can verify the details of candidate.
- System generated token gives more secure logins.
- Voting results can be published quickly. No manual work is needed for result calculation.

2 DATABASE ANALYSIS

This section discusses about the business situation, business rules, entity relationships etc.

2.1 BUSINESS SITUATION

A person is eligible to vote if he/she is 18 years or above. Voter Identification Card with a unique voter ID is issued to each eligible voter by the Election Commission. The voters can cast their vote irrespective of their current location. The votes can be done electronically if the voters have access to internet. Voters can login to the voting portal by entering their unique identification number provided by the Election Commission. The user will be provided with a One Time Password which has to be entered to login to the portal. Then the user is asked to undergo bio-metric verification. After successful verification, the system checks for the constituency belonged to the voter. The list of candidates participating in the election in the corresponding constituency will be displayed on the page. The voter can select the candidates and go through their details like personal information, assets, education and criminal records. Each candidate releases a manifesto for each election. Voter casts the vote and the voting status will be updated for each voter. The voter can cast a NOTA (None Of The Above) vote, if the voter finds all the candidates as non-eligible to stand for an election. The time at which the vote was made will be recorded and a receipt of the vote will be sent to the user on their respective email ID. The number of votes received by the candidate will be updated. The votes made by each voter for their individual candidate would not be recorded and would remain anonymous.

2.2 BUSINESS RULES

The business rules for the Online Voting Management System are given below

1. A unique token will be generated for each citizen for portal login.
2. The bio-metric ID of each user will be unique.
3. Each citizen belongs to a constituency whereas each constituency will have multiple citizens.
4. The voting status of each citizen will be updated as YES or NO based on the status of their vote.
5. Each citizen will be monitored by officers in Election Commission who also have the eligibility to vote.
6. A candidate may have no assets at all or several. An asset may be owned by a candidate or several.
7. A candidate may hold one or more degrees.
8. A candidate may have no criminal records or many, but each criminal record may belong to more than one candidate.
9. Every candidate is eligible to run in many elections, and there are several candidates in each election.
10. Every constituency will have numerous elections, and an election will take place in each of those constituencies.
11. Each election will only have one outcome.
12. Several parties take part in an election and there will multiple elections.
13. There will be only a candidates for each party, and each candidate may or may not have a party.
14. There will be several candidates standing for election in each constituency, and each candidate is a member of only one constituency.
15. A constituency should have at least two candidates for an election. If there is only a candidate standing for election, he will be announced as the winner.
16. Each candidate will gain a result, and each result will be specific to that candidate.
17. Each candidate publishes a manifesto which contains campaign promises.
18. Each voter can cast a NOTA vote but this remains anonymous. Total count of NOTA votes for a particular election is recorded.

2.3 LIST OF ENTITIES AND ATTRIBUTES

There are 14 entities and 41 attributes in the online voting management system.The list of the entities and the corresponding attributes are given below

Entities and Attributes List	
Entity	**Attributes**
person	voter_id first_name middle_name last_name gender date_of_birth age email_address phone_number constituency_id address_id
citizen	citizen_id
candidate	candidate_id party_id election_id
authentication	voter_id authentication_date_time
citizen_token	token
citizen_biometrics	face_id finger_print
voting _status	election_id voter_id voter_status voting_time
party	party_id party_name party_symbol candidate_id election_id
election	election_id election_date election_type result_date
constituency	constituency_id constituency_name population citizen_id candidate_id election_id party_id
candidate_asset	candidate_id asset_id asset_type valuation

candidate_education	candidate_id degree pass_year
criminal_record	candidate_id criminal_id conviction date crime_type
voting_result	candidate_id votes election_id
nota	election_id nota_votes
election_manifesto	election_id party_id candidate_id

Table 1 : *Entities and Attributes*

There are different types of attributes in the Online Voting system.

- Composite Attribute
  1. *name* of person
  2. *address* of person
- Derived Attribute
  1. *age* of person
- Multi-valued Attribute
  1. *degree* of candidate
  2. *type* of criminal_record
  3. *asset_type* of candidate_asset

2.4 ENTITY CONNECTIVITIES AND RELATIONSHIPS

This section includes the relationships between entities in the Online Voting Management System. The relationships between entities are given below.

1. **1 to 1 Relationships**
   - [citizen]1 <does> 1[citizen_token]
   - [citizen]1 <does> 1[citizen_biometric]
   - [citizen]1 <has> 1[voting_status]
   - [candidate]1 <gains> 1[voting_result]
   - [candidate]1 <publishes> 1[election_manifesto]
   - [candidate]1 <represents> 1[party]

- [election]1 <has> 1[voting_result]
- [election]1 <has> 1[nota]

2. **1 to M Relationships**
   - [citizen]1 <monitors> M[citizen]
   - [constituency]1 <belongs to> M[citizen]
   - [candidate]M <belongs to> 1[constituency]
   - [candidate]1 <contests> M[election]
3. **M to N Relationships**
   - [candidate]M <owns> N[candidate_asset]
   - [candidate]M <attains> N[candidate_education]
   - [candidate]M <holds> N[criminal_record]
   - [election]M <involves> N[party]
   - [election]M <happens> N[constituency]
4. **Recursive Relationship**
   - [citizen] <monitors> [citizen]

2.5 CARDINALITIES AND PARTICIPATION

2.5.1 Cardinalities

1. **citizen to citizen_token**
   - A *citizen* authenticates with a minimum of 1 *citizen_token*
   - A *citizen* authenticates with a maximum of 1 *citizen_token*

   Reverse :
   - A *citizen_token* is authenticated by a minimum of 1 *citizen*
   - A *citizen_token* is authenticated by a maximum of 1 *citizen*
2. **citizen to citizen_biometric**
   - A citizen authenticates with a minimum of 1 citizen_biometric
   - A citizen authenticates with a maximum of 1 citizen_biometric

   Reverse :
   - A citizen_biometric is authenticated by a minimum of 1 citizen
   - A citizen_biometric is authenticated by a maximum of 1 citizen
3. **citizen to voting_status**
   - A *citizen* has a minimum of 1 *voting_status*
   - A *citizen* has a maximum of 1 *voting_status*

   Reverse :

- A *voting_status* is for a minimum of 1 *citizen*
- A *voting_status* is for a maximum of M *citizen*

4. **citizen to citizen**

   - A *citizen* is monitored by a minimum of 1 *citizen*
   - A *citizen* is monitored by a maximum of M *citizen*

5. **citizen to constituency**

   - A *citizen* belongs to a minimum of 1 *constituency*
   - A *citizen* belongs to a maximum of 1 *constituency*

   Reverse :

   - A *constituency* is belonged to a minimum of 1 *citizen*
   - A *constituency* is belonged to a maximum of M *citizen*

6. **candidate to candidate_asset**

   - A *candidate* owns a minimum of 0 *candidate_asset*
   - A *candidate* owns a maximum of M *candidate_asset*

   Reverse :

   - A *candidate_asset* is owned by a minimum of 1 *candidate*
   - A *candidate_asset* is owned by a maximum of N *candidate*

7. **candidate to candidate_education**

   - A *candidate* attains a minimum of 1 *candidate_education*
   - A *candidate* holds a maximum of N *candidate_education*

   Reverse :

   - An *candidate_education* is attained by a minimum of 1 *candidate*
   - An *candidate_education* is attained by a maximum of M *candidate*

8. **candidate to criminal_record**

   - A *candidate* holds a minimum of 0 *criminal_record*
   - A *candidate* holds a maximum of N *criminal_record*

   Reverse :

   - A *criminal_record* is held by a minimum of 1 *candidate*
   - A *criminal_record* is held by a maximum of M *candidate*

9. **candidate to voting_result**

   - A *candidate* gains a minimum of 1 *voting_result*
   - A *candidate* gains a maximum of 1 *voting_result*

   Reverse :

   - A *voting_result* is gained by a minimum of 1 *candidate*
   - A *voting_result* is gained by a maximum of M *candidate*

10. **candidate to party**

    - A *candidate* represents a minimum of 0 *party*
    - A *candidate* represents a maximum of 1 *party*

    Reverse :

    - A *party* is represented by a minimum of 1 *candidate*
    - A *party* is represented by a maximum of 1 *candidate*

11. **candidate to constituency**

    - A *candidate* belongs to a minimum of 1 *constituency*
    - A *candidate* belongs a maximum of 1 *constituency*

    Reverse :

    - A *constituency* is belonged to a minimum of 2 *candidate*
    - A *constituency* is belonged to a maximum of M *candidate*

12. **election to constituency**

    - An *election* happens in a minimum of 1 *constituency*
    - A *election* happens in a maximum of M *constituency*

    Reverse :

    - A *constituency* has a minimum of 1 *election*
    - A *constituency* has a maximum of M *election*

13. **election to voting_result**

    - An *election* has a minimum of 1 *voting_result*
    - A *election* has in a maximum of 1 *voting_result*

    Reverse :

    - A *voting_result* has a minimum of 1 *election*
    - A *voting_result* has a maximum of 1 *election*

14. **election to party**

- An *election* involves a minimum of 1 *party*
- An *election* involves a maximum of M *party*

Reverse :

- A *party* is involved in a minimum of 1 *election*
- A *party* is involved in a maximum of M *election*

15. **election to nota**

- An *election* has a minimum of 0 *nota*
- An *election* has a maximum of 1 *nota*

Reverse :

- A *nota* has a minimum of 1 *election*
- A *nota* is has a maximum of 1 *election*

16. **candidate to election_manifesto**

- An *candidate* publishes a minimum of 1 *election_manifesto*
- An *candidate* publishes a maximum of 1 *election_manifesto*

Reverse :

- An *election_manifesto* is published by a minimum of 1 *candidate*
- An *election_manifesto* is published by a maximum of 1 *candidate*

17. **candidate to election**

- An *candidate* competes in a minimum of 1 *election*
- An *candidate* competes in a maximum of M *election*

Reverse :

- An *election* is competed by a minimum of 1 *candidate*
- An *election* is competed by a maximum of M *candidate*

2.5.2 Participation

All the entity participation involved in the Online Voting Management System is **mandatory** except the participation of candidate_asset. The entity *candidate* has an **optional** participation with *candidate_asset*.

3 DATABASE DESIGN

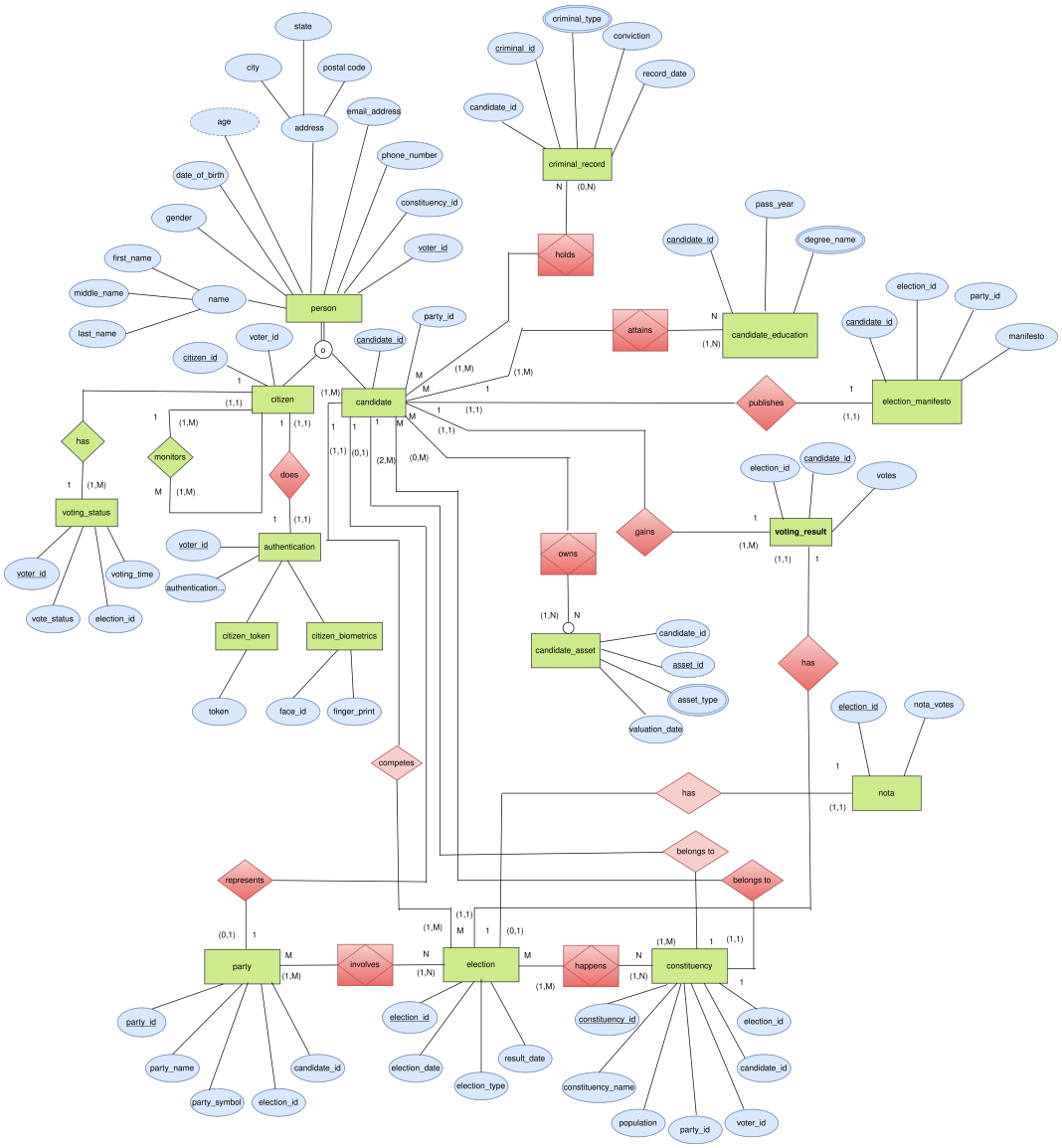

Figure 1: *ER Diagram*

4 DATABASE NORMALISATION

Normalisation is performed on all tables in order to database performance and to reduce redundancy.

1. **person**

Figure 2: *person table structure*

- **voter_id** →(first_name, middle_name, last_name, gender, date_of_birth, age, email_address, phone_number, constituency_id, address_id)
- All the attributes of **person** table fully dependent on **voter_id (PK)**
- *address* table is created to reduce the multi-valued attribute.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

2. **address**

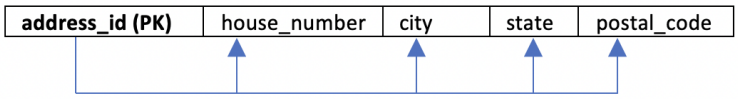

Figure 3: *address table structure*

- **address_id** →(house_number, city, state, postal_code)
- All the attributes of **address** table fully dependent on **address_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

3. **citizen**

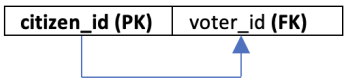

Figure 4: *citizen table structure*

- **citizen_id** →(voter_id)

- All the attributes of **citizen** table fully dependent on **citizen_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

4. **candidate**

Figure 5: *candidate table structure*

- **candidate_id** →(party_id, voter_id, election_id, constituency_id)
- All the attributes of **candidate** table fully dependent on **candidate_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

5. **criminal_record**

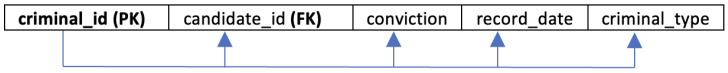

Figure 6: *crime record table structure before normalisation*

- *crime_type* table is created to remove the redundancy of *type*.

Figure 7: *crime record table structure after normalisation*

- **criminal_id** →(candidate_id, conviction, record_date, crime_type_id)
- All the attributes of **criminal_record** table fully dependent on **criminal_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

6. **crime_type**

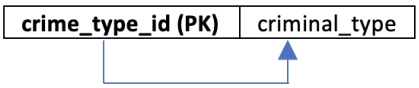

Figure 8: *crime_type table structure after normalisation*

- **crime_type_id** →(type)
- The attribute *type* of **crime_type** table fully dependent on **criminal_type_id (PK).**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

7. **candidate_education**

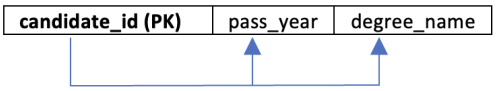

Figure 9: *candidate education table structure before normalisation*

- *degree* table is created to remove the redundancy of *degree*.

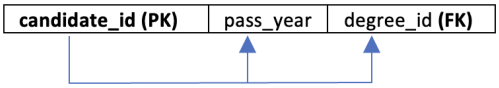

Figure 10: *candidate education table structure after normalisation*

- **candidate_id** →(pass_year, degree_id)
- All the attributes of **candidate_education** table fully dependent on **candidate_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

8. **degree**

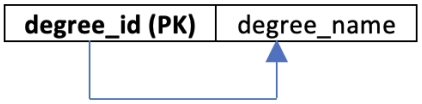

Figure 11: *degree table structure*

- **degree_id** →(degree_name)
- All attributes table **degree** fully dependent on **degree_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

9. **candidate_asset**

Figure 12: *candidate asset table structure before normalisation*

- *asset* table is created to remove the redundancy of *asset_type*.

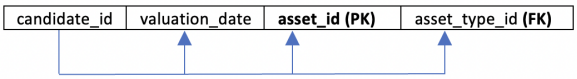

Figure 13: *candidate asset table structure after normalisation*

- **asset_id** →(valuation_date, candidate_id, asset_type_id)
- All attributes of table **candidate_asset** fully dependent on **asset_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

10. **asset**

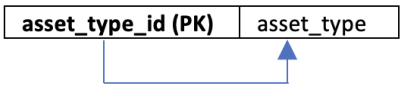

Figure 14: *asset type table structure*

- **asset_type_id** →(asset_type)
- The attribute **asset_type** of table **asset_type** is fully dependent on **asset_type_id (PK)**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

11. **voting_result**

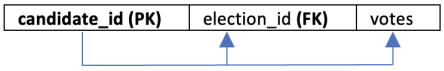

Figure 15: *asset type table structure*

- **candidate_id** →(votes, election_id)
- The attribute **votes** of the table **voting_result** is fully functionally dependent on **candidate_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

12. **constituency**

- **constituency_id** →(constituency_name, population, election_id, party_id, citizen_id, candidate_id, voter_id)
- All attributes of the table **constituency** is fully functionally dependent on **constituency_id (PK)**.

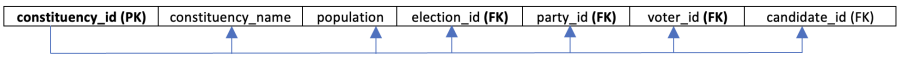

Figure 16: *constituency table structure*

- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

13. **election**

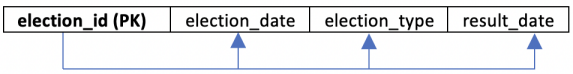

Figure 17: *election table structure before normalisation*

- *election_type* table is created to remove the redundancy of *election*.

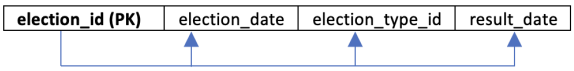

Figure 18: *election table structure after normalisation*

- **election_id** →(election_date, election_date, election_type_id, result_date)
- All attributes of table **election** fully dependent on **election_id (PK).**
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

14. **election_type**

- **election_type_id** →(election_type)
- The attribute **election_type** of the table **election_type** is fully functionally dependent on **election_type_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

15. **nota**

- **election_id** →(nota_votes)

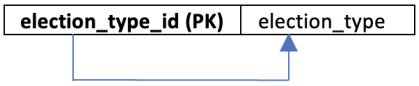

Figure 19: *election type table structure*

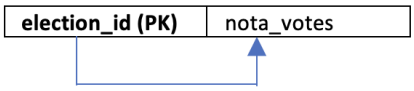

Figure 20: *nota table structure*

- The attribute **nota_votes** of the table **nota** is fully functionally dependent on **election_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

16. **election_manifesto**

Figure 21: *election manifesto table structure*

- **candidate_id** →(election_id, party_id, manifesto)
- All attributes of the table **election_manifesto** is fully functionally dependent on **candidate_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

17. **party**

- **party_id** →(party_name, party_symbol, election_id, candidate_id)
- All attributes of the table **party** is fully functionally dependent on **party_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

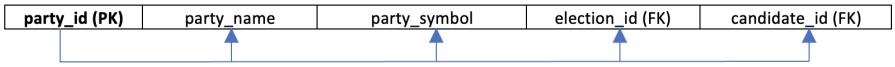

Figure 22: *party table structure*

18. **authentication**

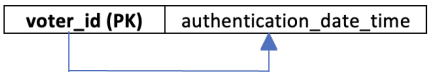

Figure 23: *authentication table structure*

- **voter_id** →(authentication_date_time)
- The attribute **authentication_date_time** of the table **authentication** is fully functionally dependent on **voter_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

19. **citizen_token**

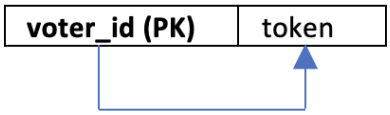

Figure 24: *citizen_token table structure*

- **voter_id** →(citizen_token)
- The attribute **token** of the table **citizen_token** is fully functionally dependent on **voter_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

20. **citizen_biometrics**

- **voter_id** →(face_id, finger_print)
- All attributes of the table **citizen_biometrics** is fully functionally dependent on **voter_id (PK)**.

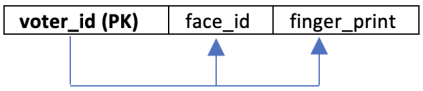

Figure 25: *citizen_bio-metrics table structure*

- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

21. **voting_status**

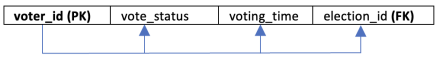

Figure 26: *voting_status table structure*

- **voter_id** →(vote_status, voting_time, election_id)
- All attributes of the table **voting_status** is fully functionally dependent on **voter_id (PK)**.
- The table is in 1NF as there are no redundancies.
- This table is in 2NF as there are no partial dependencies.
- This table is in 3NF as there are no transitive dependencies.

4.1 RELATIONAL SCHEMA

Relationship schema was implemented after normalising the tables.

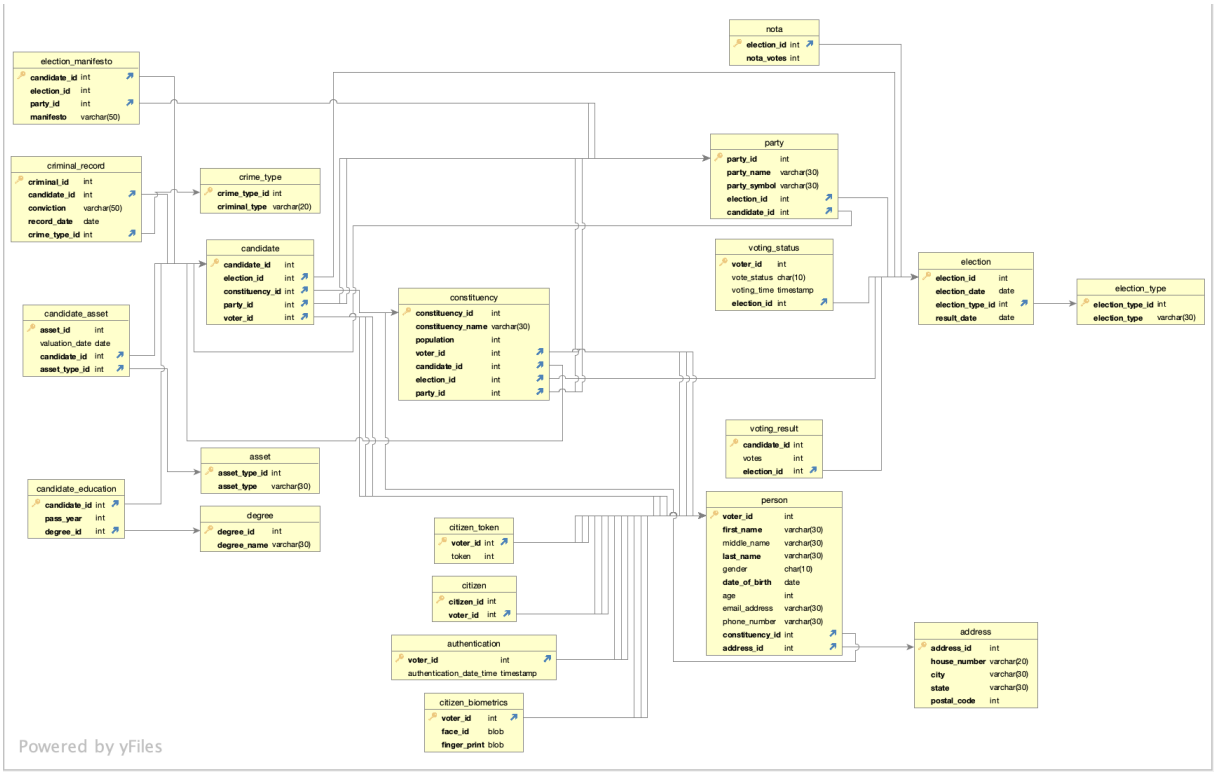

Figure 27: *Relational Schema*

5 DATABASE IMPLEMENTATION

5.1 SQL TABLE CREATION COMMANDS

5.1.1 TABLES CREATION

1. **Create person table**

```sql
CREATE TABLE person
(
    voter_id INTEGER NOT NULL PRIMARY KEY,
    first_name VARCHAR(30) NOT NULL,
    middle_name VARCHAR(30),
    last_name VARCHAR(30) NOT NULL,
    gender CHAR(10) ,
    date_of_birth DATE NOT NULL,
    age INTEGER,
    email_address VARCHAR(30) UNIQUE,
    phone_number VARCHAR(30) UNIQUE,
    constituency_id INTEGER NOT NULL,
    address_id INTEGER NOT NULL
);
```

Figure 28: *CREATE command for table person*

2. **Create address table**

```sql
CREATE TABLE address
(
    address_id INTEGER NOT NULL PRIMARY KEY,
    house_number VARCHAR(10) NOT NULL,
    city VARCHAR(30) NOT NULL,
    state VARCHAR(30) NOT NULL,
    postal_code INTEGER NOT NULL
);
```

Figure 29: *CREATE command for table address*

3. **Create citizen table**

```sql
CREATE TABLE citizen
(
    citizen_id INTEGER NOT NULL PRIMARY KEY,
    voter_id INTEGER NOT NULL
);
```

Figure 30: *CREATE command for table citizen*

4. **Create candidate table**

```sql
CREATE TABLE candidate
(
    candidate_id INTEGER NOT NULL PRIMARY KEY,
    election_id INTEGER NOT NULL,
    constituency_id INTEGER NOT NULL,
    party_id INTEGER NOT NULL,
    voter_id INTEGER NOT NULL
);
```

Figure 31: *CREATE command for table candidate*

5. **Create criminal_record table**

```sql
CREATE TABLE criminal_record
(
    criminal_id INTEGER NOT NULL PRIMARY KEY,
    candidate_id INTEGER NOT NULL,
    conviction VARCHAR(50) NOT NULL,
    record_date DATE NOT NULL,
    crime_type_id INTEGER NOT NULL
);
```

Figure 32: *CREATE command for table criminal_record*

6. **Create crime_type table**

```
CREATE TABLE crime_type
(
    crime_type_id INTEGER NOT NULL PRIMARY KEY,
    criminal_type VARCHAR(20) NOT NULL
);
```

Figure 33: *CREATE command for table crime_type*

7. **Create candidate_education table**

```
CREATE TABLE candidate_education
(
    candidate_id INTEGER NOT NULL PRIMARY KEY,
    pass_year INTEGER NOT NULL,
    degree_id INTEGER NOT NULL
);
```

Figure 34: *CREATE command for table candidate_education*

8. **Create degree table**

```
CREATE TABLE degree
(
    degree_id INTEGER NOT NULL PRIMARY KEY,
    degree_name VARCHAR(30) NOT NULL
);
```

Figure 35: *CREATE command for table degree*

9. **Create candidate_asset table**

```sql
CREATE TABLE candidate_asset
(
    asset_id INTEGER NOT NULL PRIMARY KEY,
    valuation_date DATE,
    candidate_id INTEGER NOT NULL,
    asset_type_id INTEGER NOT NULL
);
```

Figure 36: *CREATE command for table candidate_asset*

10. **Create asset table**

```sql
CREATE TABLE asset
(
    asset_type_id INTEGER NOT NULL PRIMARY KEY,
    asset_type VARCHAR(30) NOT NULL
);
```

Figure 37: *CREATE command for table asset*

11. **Create voting_result table**

```sql
CREATE TABLE criminal_record
(
    criminal_id INTEGER NOT NULL PRIMARY KEY,
    candidate_id INTEGER NOT NULL,
    conviction VARCHAR(50) NOT NULL,
    record_date DATE NOT NULL,
    crime_type_id INTEGER NOT NULL
);
```

Figure 38: *CREATE command for table criminal_record*

12. **Create constituency table**

```sql
CREATE TABLE constituency
(
    constituency_id INTEGER NOT NULL PRIMARY KEY,
    constituency_name VARCHAR(30) NOT NULL,
    population INTEGER NOT NULL,
    voter_id INTEGER NOT NULL,
    candidate_id INTEGER NOT NULL,
    election_id INTEGER NOT NULL,
    party_id INTEGER NOT NULL
);
```

Figure 39: *CREATE command for table constituency*

13. **Create election table**

```sql
CREATE TABLE election
(
    election_id INTEGER NOT NULL PRIMARY KEY,
    election_date DATE NOT NULL,
    election_type_id INTEGER NOT NULL,
    result_date DATE NOT NULL
);
```

Figure 40: *CREATE command for table election*

14. **Create election_type table**

```sql
CREATE TABLE election_type
(
    election_type_id INTEGER NOT NULL PRIMARY KEY,
    election_type VARCHAR(30) NOT NULL
);
```

Figure 41: *CREATE command for table election_type*

15. **Create party table**

```
CREATE TABLE party
(
    party_id INTEGER NOT NULL PRIMARY KEY,
    party_name VARCHAR(30) NOT NULL,
    party_symbol VARCHAR(30) NOT NULL UNIQUE,
    election_id INTEGER NOT NULL,
    candidate_id INTEGER NOT NULL
);
```

Figure 42: *CREATE command for table party*

16. **Create authentication table**

```
CREATE TABLE authentication
(
    voter_id INTEGER NOT NULL PRIMARY KEY,
    authentication_date_time TIMESTAMP
);
```

Figure 43: *CREATE command for table authentication*

17. **Create citizen_token table**

```
CREATE TABLE citizen_token
(
    voter_id INTEGER NOT NULL PRIMARY KEY,
    token INTEGER
);
```

Figure 44: *CREATE command for table citizen_token*

18. **Create citizen_biometrics table**

```sql
CREATE TABLE citizen_biometrics
(
    voter_id INTEGER NOT NULL PRIMARY KEY,
    face_id BLOB NOT NULL,
    finger_print BLOB NOT NULL
);
```

Figure 45: *CREATE command for table citizen_biometrics*

19. **Create voting_status table**

```sql
CREATE TABLE voting_status
(
    voter_id INTEGER NOT NULL PRIMARY KEY,
    vote_status CHAR(10),
    voting_time TIMESTAMP,
    election_id INTEGER NOT NULL
);
```

Figure 46: *CREATE command for table voting_status _create*

20. **Create election_manifesto table**

```sql
CREATE TABLE election_manifesto
(
    candidate_id INTEGER NOT NULL PRIMARY KEY,
    elction_id INTEGER NOT NULL,
    party_id INTEGER NOT NULL,
    manifesto VARCHAR(50) NOT NULL
);
```

Figure 47: *CREATE command for table election_manifesto*

21. **Create nota table**

```sql
CREATE TABLE nota
(
    election_id INTEGER NOT NULL PRIMARY KEY,
    nota_votes INTEGER NOT NULL
);
```

Figure 48: *CREATE command for table nota*

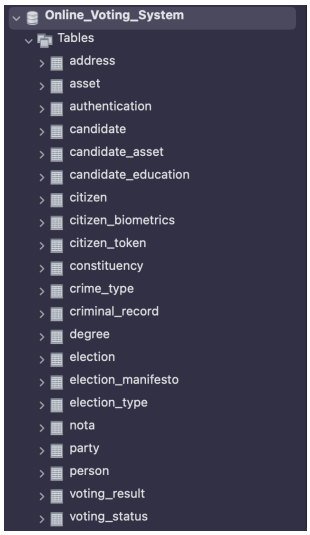

Figure 49: *List of tables created*

5.1.2 FOREIGN KEYS

```
ALTER TABLE person
ADD CONSTRAINT FK_person_constituency
FOREIGN KEY (constituency_id) REFERENCES constituency(constituency_id);

ALTER TABLE person
ADD CONSTRAINT FK_person_address
FOREIGN KEY (address_id) REFERENCES address(address_id);

ALTER TABLE citizen
ADD CONSTRAINT FK_citizen_voter
FOREIGN KEY (voter_id) REFERENCES person(voter_id);

ALTER TABLE candidate
ADD CONSTRAINT FK_candidate_party
FOREIGN KEY (party_id) REFERENCES party(party_id);

ALTER TABLE candidate
ADD CONSTRAINT FK_candidate_election
FOREIGN KEY (election_id) REFERENCES election(election_id);

ALTER TABLE candidate
ADD CONSTRAINT FK_candidate_constituency
FOREIGN KEY (constituency_id) REFERENCES constituency(constituency_id);

ALTER TABLE candidate
ADD CONSTRAINT FK_candidate_voter
FOREIGN KEY (voter_id) REFERENCES person(voter_id);

ALTER TABLE criminal_record
ADD CONSTRAINT FK_criminal_candidate
FOREIGN KEY (candidate_id) REFERENCES candidate (candidate_id);

ALTER TABLE criminal_record
ADD CONSTRAINT FK_criminal_crime_type
FOREIGN KEY (crime_type_id) REFERENCES crime_type(crime_type_id);

ALTER TABLE candidate_education
ADD CONSTRAINT FK_candidate_degree
FOREIGN KEY (degree_id) REFERENCES degree(degree_id);


ALTER TABLE candidate_education ADD CONSTRAINT FK_candidate_education
FOREIGN KEY (candidate_id) REFERENCES candidate(candidate_id);

ALTER TABLE candidate_asset
ADD CONSTRAINT FK_candidate_asset
FOREIGN KEY (asset_type_id) REFERENCES asset(asset_type_id);
```

```
ALTER TABLE constituency
ADD CONSTRAINT FK_constituency_candidate
FOREIGN KEY (candidate_id) REFERENCES candidate(candidate_id);

ALTER TABLE constituency
ADD CONSTRAINT FK_constituency_voter
FOREIGN KEY (voter_id) REFERENCES person(voter_id);

ALTER TABLE constituency
ADD CONSTRAINT FK_constituency_election
FOREIGN KEY (election_id) REFERENCES election(election_id);

ALTER TABLE constituency
ADD CONSTRAINT FK_constituency_party
FOREIGN KEY (party_id) REFERENCES party(party_id);

ALTER TABLE election
ADD CONSTRAINT FK_election_type
FOREIGN KEY (election_type_id) REFERENCES election_type(election_type_id);

ALTER TABLE party
ADD CONSTRAINT FK_party_election
FOREIGN KEY (election_id) REFERENCES election(election_id);

ALTER TABLE party
ADD CONSTRAINT FK_party_candidate
FOREIGN KEY (candidate_id) REFERENCES candidate(candidate_id);

ALTER TABLE election_manifesto
ADD CONSTRAINT FK_manifesto_candidate
FOREIGN KEY (candidate_id) REFERENCES candidate(candidate_id);

ALTER TABLE election_manifesto
ADD CONSTRAINT FK_manifesto_party
FOREIGN KEY (party_id) REFERENCES party(party_id);

ALTER TABLE voting_result ADD CONSTRAINT FK_voting_election FOR-
EIGN KEY (election_id) REFERENCES election(election_id);

ALTER TABLE voting_status ADD CONSTRAINT FK_voting_status FOR-
EIGN KEY (election_id) REFERENCES election(election_id);

ALTER TABLE nota ADD CONSTRAINT FK_nota FOREIGN KEY (elec-
tion_id) REFERENCES election(election_id);

ALTER TABLE citizen_token ADD CONSTRAINT FK_citizen_token FOR-
EIGN KEY (voter_id) REFERENCES person(voter_id);

ALTER TABLE citizen_biometrics ADD CONSTRAINT FK_citizen_biometrics
FOREIGN KEY (voter_id) REFERENCES person(voter_id);
```

ALTER TABLE authentication ADD CONSTRAINT FK_citizen_authentication
FOREIGN KEY (voter_id) REFERENCES person(voter_id);

5.1.3 CONSTRAINTS

1. A person should be 18 years or older to cast a vote
   ALTER TABLE person
   ADD CONSTRAINT age
   CHECK (age >= 18);

2. The result will be announced in a date after the election date
   ALTER TABLE election
   ADD CONSTRAINT result_date
   CHECK (result_date > election_date);

5.2 SQL INSERT COMMANDS

5.2.1 Data Insertion

1. **Insert values to table person**

   INSERT INTO person (voter_id, first_name, middle_name, last_name, gender, date_of_birth, email_address, phone_number, constituency_id) VALUES (61858, 'Smith', 'Rob', 'Jason', 'Male', '1990-08-04', 'smith@gmail.com', 0987658767, 001);
   INSERT INTO person (voter_id, first_name, middle_name, last_name, gender, date_of_birth, email_address, phone_number, constituency_id) VALUES (60989, 'Mathew', 'Zendaya', 'Female', '1989-09-04', 'zendmat@gmail.com', 09876777767, 008);
   INSERT INTO person (voter_id, first_name, middle_name, last_name, gender, date_of_birth, email_address, phone_number, constituency_id) VALUES (60653, 'Gowda', 'Richa', 'Female','1977-05-04', 'richa123@gmail.com', 09876457767, 009);

2. **Insert values to table address**

   INSERT INTO address VALUES (61858, 'Church_Road', 'Bangalore', 'Karnataka', 680877);
   INSERT INTO address VALUES (62352, 'High_Street', 'Bangalore', 'Karnataka', 680800);
   INSERT INTO address VALUES (65672, 'Newhall_Street', 'Bangalore', 'Karnataka', 680888);

3. **Insert values to table citizen**

   INSERT INTO citizen VALUES (900899, 61858);

INSERT INTO citizen VALUES (966555, 60989);
INSERT INTO citizen VALUES (966755, 606534);

4. **Insert values to table candidate**

INSERT INTO candidate VALUES (2343, 45645, 001, 878, 67565);
INSERT INTO candidate VALUES (2241, 45737, 003, 876, 67879);
INSERT INTO candidate VALUES(1563, 45986, 007, 874, 67234);

5. **Insert values to table criminal_record**

INSERT INTO criminal_record VALUES (1212, 2343, 'Bribery', '1998-07-05', 45678);
INSERT INTO criminal_record VALUES (2341, 2343, 'Murder', '2000-07-05', 34569);
INSERT INTO criminal_record VALUES (4567, 2241, 'Bribery', '2005-07-02', 67543);

6. **Insert values to table crime_type**

INSERT INTO crime_type VALUES (45678,'Bribery');
INSERT INTO crime_type VALUES (34569,'Murder');
INSERT INTO crime_type VALUES (67543,'Arson');

7. **Insert values to table candidate_education**

INSERT INTO candidate_education VALUES (2343, 1996, 111);
INSERT INTO candidate_education VALUES (2241, 2000, 121);
INSERT INTO candidate_education VALUES (1563, 2000, 131);

8. **Insert values to table degree**

INSERT INTO degree VALUES (111,'MSc International Relations');
INSERT INTO degree VALUES(121,'BSc Mathematics');
INSERT INTO degree VALUES (121,'MSc Global Business Management');
INSERT INTO degree VALUES (131,'B.Tech CSE');

9. **Insert values to table candidate_asset**

INSERT INTO candidate_asset VALUES (34567, 2014-07-07, 2343, 22);
INSERT INTO candidate_asset VALUES (38967, 2018-09-02, 2241, 21);
INSERT INTO candidate_asset VALUES (33597, 2010-07-02, 1563, 23);

10. **Insert values to table asset**

```
INSERT INTO asset VALUES (22, 'Land');
INSERT INTO asset VALUES (21,'Commercial Property');
INSERT INTO asset VALUES (23, 'Apartment');
```

11. **Insert values to table voting_result**

```
INSERT INTO voting_result VALUES (2343, 223456,441);
INSERT INTO  voting_result VALUES (2241, 23118,241);
INSERT INTO voting_result VALUES (1563, 12599,340);
```

12. **Insert values to table constituency**

```
INSERT INTO constituency VALUES (119, 'Udupi', 1235661, 900899,
2343, 441, 666);
INSERT INTO constituency VALUES (189, 'Belur',1234669, 966555, 2241,
241, 899);
INSERT INTO constituency VALUES (190, 'Mangalore',8235600, 966755,
1563, 340, 999);
```

13. **Insert values to table election**

```
INSERT INTO election VALUES (441,'2021-12-12', 11 ,'2021-12-15');
INSERT INTO election VALUES (241, '2015-08-05', 12, '2015-08-20');
INSERT INTO election VALUES(340, '2012-04-07', 13, '2012-04-15');
```

14. **Insert values to table election_type**

```
INSERT INTO election_type VALUES (11, 'General election');
INSERT INTO election_type VALUES (12,'By election');
INSERT INTO election_type VALUES (13, 'By election');
```

15. **Insert values to table party**

```
INSERT INTO party VALUES(666, 'INC', 'Lion',441,2343);
INSERT INTO party VALUES(899, 'CPI', 'Flag',241,2241);
INSERT INTO party VALUES(999, 'CPIM', 'Ladder',340,1543);
```

16. **Insert values to table authentication**

```
INSERT INTO authentication VALUES (61858,'2021-08-07 14:59:59');
INSERT INTO authentication VALUES (60989,'2015-08-05 13:59:59');
INSERT INTO authentication VALUES (60653,'2012-04-07 15:59:59');
```

17. **Insert values to table citizen_token**

INSERT INTO citizen_token VALUES (61858,112278);
INSERT INTO citizen_token VALUES (60989,348899);
INSERT INTO citizen_token VALUES (60653,237889);

18. **Insert values to table citizen_biometrics**

INSERT INTO citizen_biometrics VALUES (61858,23880, 24880);
INSERT INTO citizen_biometrics VALUES (60989,13881, 12981);
INSERT INTO citizen_biometrics VALUES (60653,33881, 99800);

19. **Insert values to table voting_voting**

INSERT INTO voting_status VALUES (61858, 'Yes', '2021-08-07 15:05:05');
INSERT INTO voting_status VALUES (60989,'Yes', '2015-08-05 14:05:05');
INSERT INTO voting_status VALUES (60653, 'Yes','2015-08-05 16:05:05');

20. **Insert values to table election_manifesto**

INSERT INTO election_manifesto VALUES (2343,441,666'Increase society integration');
INSERT INTO election_manifesto VALUES (2241,241,899,'Develop Commercial services');
INSERT INTO election_manifesto VALUES (1563,340,999,'Promote postgraduate involvement');

21. **Insert values to table nota**

INSERT INTO nota VALUES (441,200);
INSERT INTO nota VALUES (241,500);
INSERT INTO nota VALUES(340,430);

criminal_record table view is given below

criminal_id	candidate_id	conviction	date	crime_type_id
1212	2343	Bribery	1998-07-05	45678
2341	2343	Murder	2000-07-05	34569
4567	2241	Bribery	2005-07-02	67543

Figure 50: *criminal_record table view*

5.2.2 Deriving Age from Date of Birth Column

In MySQL, while creating generated columns, errors will occur if the expression contains disallowed construct (MySQL, 2022) . Hence, it's impossible to derive age from the date_of_birth column as the functions used

are non-deterministic. E.g : CURDATE(), NOW (). But, these functions can be used to retrieve age from the database using the Select statement.

Figure 51: *Derive age using SELECT statement*

5.2.3 Constraints Check

(a) Test **result date** constraint check
The result date should be after election date.

Figure 52: *Constraint check command for result_date in table election*

(b) Test **age** constraint check
Age of a person should be greater than or equal to 18 to cast a vote.

Figure 53: *Constraint check command for age in table person*

5.3 SQL TEST COMMANDS

i. Scenario 1 : **To find the winner of particular election in a particular constituency** (Meera)

```
select concat(person.first_name, " " ,person.last_name) as Winner       #combine the first and last name
from person
INNER join candidate on person.voter_id=candidate.voter_id              #finding the person details based on candidate voter_id
where candidate_id =
(SELECT max(voting_result.candidate_id) as Election_Winner              #subquery to find the candidate with maximum number of votes
FROM voting_result
INNER JOIN constituency ON voting_result.candidate_id=constituency.candidate_id     #finding voting result of candidates based on election_id
where constituency.election_id= 241 and constituency.constituency_id = 189          #and constituency_id
);
```

Figure 54: *Query to find the winner of particular election in a particular constituency*

Output :

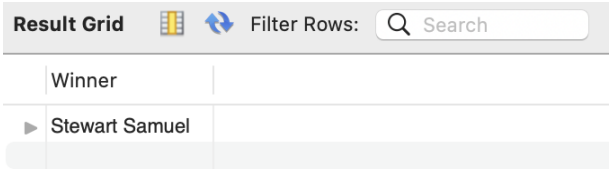

Figure 55: *Output of scenario 1*

ii. Scenario 2: **To find the types of elections held between a specified period** (Alinta)

```
select distinct(election_type.election_type)                                    #select only distinct election types
FROM election_type
INNER JOIN election ON election_type.election_type_id = election.election_type_id   #finding election types based on election_type_id
where election_date BETWEEN '2012-01-01' AND '2021-12-31';                      #specifying the date range
```

Figure 56: *Query to find the types of elections held between a specified period*

Output :

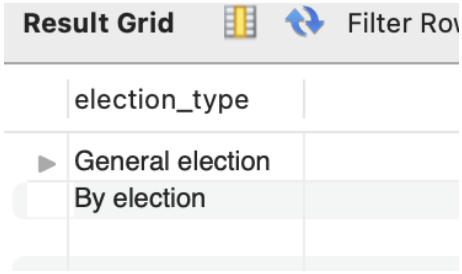

Figure 57: *Output of scenario 2*

iii. Scenario 3: **To find the elections with NOTA votes greater than a specified number** (Meera)

```
select *                    #Display all information
from election
where election_id in        #election_id will be taken from the subquery
(select election_id         #finding the election_id with nota_votes greater than 200
from nota
where nota_votes>200
);
```

Figure 58: *Query to find the elections with NOTA votes greater than a specified number*

Output :

election_id	election_date	election_type_id	result_date
241	2015-08-05	12	2015-08-20
340	2012-04-07	13	2012-04-15

Figure 59: *Output of scenario 3*

iv. Scenario 4: **To find the election ids, election types, and nota votes for each election with nota votes in descending order** (Alinta)

```
SELECT election.election_id,                                #finding election_id,election_type and nota_votes from election table
 election_type.election_type, nota.nota_votes
FROM election
INNER JOIN nota ON election.election_id = nota.election_id          #joined election and nota table
INNER JOIN election_type ON election.election_type_id = election_type.election_type_id #joined election and election_type table
group by election. election_id
order by nota_votes DESC;                                                    #to display nota_votes in descending order
```

Figure 60: *to find the election ids, election types, and nota votes for each election with nota votes in descending order*

Output :

election_id	election_type	nota_votes
241	By election	500
340	By election	430
441	General election	200

Figure 61: *Output of scenario 4*

v. Scenario 5: **To find the details of the candidates participating in a particular election** (Meera)

```
select concat(person.first_name, " " ,person.last_name) as person_name,candidate.candidate_id,
person.gender, person.date_of_birth, person.constituency_id      #to display the details of candidate from person table
From PERSON
 INNER JOIN candidate ON person.voter_id = candidate.voter_id              #joined person and candidate table to find candidate id
 where candidate.election_id = 241;                                  #specified the election_id
```

Figure 62: *Query to find the details of the candidates participating in a particular election*

Output :

person_name	candidate_id	gender	date_of_birth	constituency_id
Stewart Samuel	2241	Male	1977-05-04	119
Fahadh Raj	2242	Male	1980-05-04	189
Rahul Rani	2243	Male	1985-05-04	189

Figure 63: *Output of scenario 5*

vi. Scenario 6: **To find the voting result of all candidates in all elections**(Alinta)

```
 select concat(person.first_name, " " ,person.last_name) as person_name, candidate.candidate_id, voting_result.votes
from candidate                                          # to display details of candidate
INNER JOIN person ON candidate.voter_id = person.voter_id          #to find person details based on candidate's voter_id
INNER JOIN voting_result ON candidate.candidate_id = voting_result.candidate_id     #to find voting_result of the candidate
INNER JOIN election ON candidate.election_id = election.election_id ;   #to find candidates in a election
```

Figure 64: *Query to find the voting result of all candidates in all elections*

Output :

person_name	candidate_id	votes
Anderson Robin	1563	12599
Stewart Samuel	2241	23118
Fahadh Raj	2242	12599
Rahul Rani	2243	11000
Joseph Richard	2343	223456

Figure 65: *Output of scenario 6*

vii. Scenario 7: **To find the voting percentage of all elections held**(Alinta)

```
select constituency.population, voting_result.election_id, voting_result.votes, (votes/population) * 100 as Voting_Percentage
from constituency              #display election_id, population, votes and vote percentage
INNER JOIN voting_result ON constituency.election_id = voting_result.election_id; #joined tables to find population
```

Figure 66: *Query to find the voting percentage of all elections held*

Output :

population	election_id	votes	Voting_Percentage
1235661	441	223456	18.0839
1234669	241	23118	1.8724
1234669	241	12599	1.0204
1234669	241	11000	0.8909
8235600	340	12599	0.1530

Figure 67: *Output of scenario 7*

viii. Scenario 8: **To find the voting date and time at which maximum number of votes were casted for a particular election** (Meera)

```
select date(a.voting_time) as Peak_Vote_Date, time(a.voting_time) as Peak_Voting_time   #to display time and date separately as peak date and time.
from
(with cte as (SELECT voting_time,count(vote_status) as cnt   #query to find the voting_time and total vote counts in descending order
from voting_status
where vote_status = 'YES' and election_id = 241        #specified the voting_status and election_id
Group by voting_time)
select * from (
SELECT (@row_number:=@row_number + 1) AS row_num, voting_time,count(vote_status) as cnt             #to display the first row of the answer table from first query
FROM voting_status, (SELECT @row_number:=0) AS temp where vote_status = 'YES' and election_id = 241
Group by voting_time) as s
where row_num = 1)a;
```

Figure 68: *Query to find the peak voting date and time*

Output :

Peak_Vote_Date	Peak_Voting_time
2015-08-05	16:05:05

Figure 69: *Output of scenario 8*

ix. Scenario 9: **To find the details of the candidates belonging to a party**(Meera)

```
SELECT concat(person.first_name, " " ,person.last_name) AS Candidate, candidate.candidate_id, person.gender, person.date_of_birth,
person.constituency_id, candidate.party_id          #to display candidate details
FROM PERSON
 INNER JOIN candidate ON person.voter_id = candidate.voter_id   #joined tables to get the person details based on candidate's voter_id
 WHERE candidate.party_id = 999;                                #specified the party id
```

Figure 70: *Query to find the details of the candidates belonging to a party*

Output :

Candidate	candidate_id	gender	date_of_birth	constituency_id	party_id
Anderson Robin	1563	Male	1977-05-04	189	999
Rahul Rani	2243	Male	1985-05-04	189	999

Figure 71: *Output of scenario 9*

x. Scenario 10: **Voter looking for manifesto published by each candidate participating in a particular election in a particular constituency** (Alinta)

```sql
SELECT candidate.candidate_id, candidate.party_id, election_manifesto.manifesto
FROM election_manifesto         #to select candidate id, party id and manifesto
INNER JOIN candidate ON election_manifesto.candidate_id =candidate.candidate_id
WHERE candidate.candidate_id =      #candidate_id will be taken from subquery
(SELECT candidate.candidate_id      #subquery to find the candidate id all candidates participating for a particular election in a particular constituency
FROM candidate
INNER JOIN constituency ON candidate.candidate_id=constituency.candidate_id
WHERE constituency.election_id= 241 AND constituency.constituency_id = 189);
```

Figure 72: *Query to find the election manifesto published by each candidate participating in a particular election in a particular constituency*

Output :

candidate_id	party_id	manifesto
2241	899	Develop Commercial services

Figure 73: *Output of scenario 10*

5.4 QUERY PERFORMANCE, PLANNING AND OPTIMIZATION TECHNIQUE

A. **Range Partitioning on table election**

Consider the scenario 4 given above. A select query is used to find the election ids, election types, and nota votes for each election with nota votes in descending order.

The output of **EXPLAIN** of the query and it's execution time is given below

EXPLAIN:

```
-> Sort: nota.nota_votes DESC
    -> Table scan on <temporary>  (cost=3.80..5.49 rows=3)
        -> Temporary table with deduplication  (cost=2.95..2.95 rows=3)
            -> Nested loop inner join  (cost=2.65 rows=3)
                -> Nested loop inner join  (cost=1.60 rows=3)
                    -> Table scan on nota  (cost=0.55 rows=3)
                    -> Single-row index lookup on election using PRIMARY (election_id=nota.election_id)  (cost=0.28 rows=1)
                -> Single-row index lookup on election_type using PRIMARY (election_type_id=election.election_type_id)  (cost=0.28 rows=1)
```

Figure 74: *EXPLAIN of SELECT query before partitioning*

Figure 75: *Execution time of SELECT query before partitioning*

Create table *election_partitioned* with range partitioning.

```
CREATE TABLE election_partitioned
(
    election_id INTEGER NOT NULL PRIMARY KEY,
    election_date DATE NOT NULL,
    election_type_id INTEGER NOT NULL,
    result_date DATE NOT NULL
)
PARTITION BY RANGE (election_id)
(
   PARTITION p0 VALUES LESS THAN (300),
   PARTITION p1 VALUES LESS THAN (400),
   PARTITION p3 VALUES LESS THAN (500),
   PARTITION p4 VALUES LESS THAN MAXVALUE );

INSERT INTO election_partitioned
SELECT * FROM election;
```

Figure 76: *Range partition table creation*

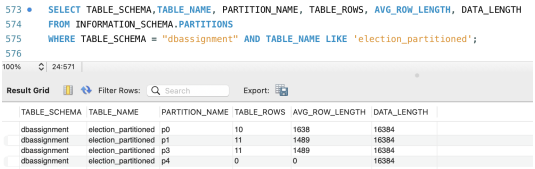

Figure 77: *Info of Range partition table*

The output of **EXPLAIN** of the query and it's execution time after range partitioning is given below

```
EXPLAIN FORMAT = TREE SELECT election_partitioned.election_id,                          #finding election_id,election_type and nota_votes fr
 election_type.election_type, nota.nota_votes
FROM election_partitioned
INNER JOIN nota ON election_partitioned.election_id = nota.election_id          #joined election and nota table
INNER JOIN election_type ON election_partitioned.election_type_id = election_type.election_type_id #joined election and election_type table
group by election_partitioned.election_id
order by nota_votes DESC;
```

Figure 78: *SELECT query used on partitioned table*

Figure 79: *Execution time of SELECT query after partitioning*

The query execution time decreased from 0.0039 seconds to 0.00094 seconds.

xi. **Indexing on table person** Consider the scenario to list name and contact details of a person whose first name like Stewart.

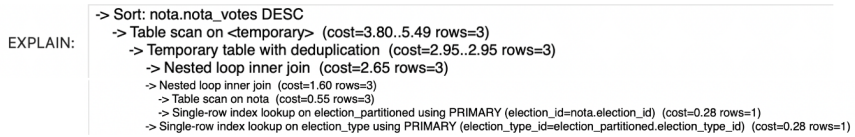

Figure 80: *EXPLAIN of SELECT query after partitioning*

Figure 81: *ANALYZE of person and EXPLAIN of SELECT query before indexing*

Created an index on the attribute first_name of the table person.

Figure 82: *Index creation*

The cost reduced from 1.05 to 0.35 and execution time reduced from 0.000014 to 0.000012.

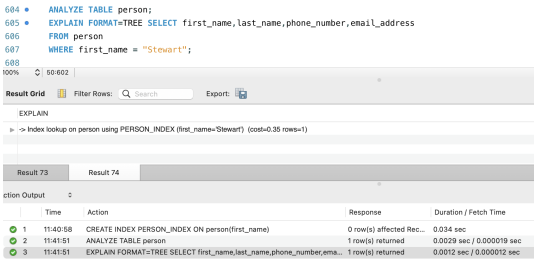

Figure 83: *ANALYZE of person and EXPLAIN of SELECT query after indexing*

5.5 CONCURRENCY CONTROL

In order to manage concurrency, locks are often used in the database world, for example to prevent one client from reading a data which another is changing. Locks does consume resources and performance can suffer if locking is not done properly. Most database servers doesn't offer much choice in terms of locking and offers what is known as row level locking which compromises on performance. MySQL, with its many storage engines - each with its own different locking policies and granularity, doesn't require a single general purpose locking solution. In our Online Voting system db, we are using Table locks which has the least overheads thus with maximum performance. Along with storage engines managing its own locks, MySQL also uses a variety of table level locks. For example MySQL uses a table level lock for ALTER TABLE regardless of the storage engine used. (Oreilly, 2022)

6 CONCLUSION

Online Voting Management System database is created to increase the efficiency of voting process for both the candidates and the voters. By developing this database, we achieved our goal to update and retrieve information quickly and efficiently.

7 REFERENCES

A. MySQL (2022) *MySQL 5.7 Reference Manual.* Available at: https://dev.mysql.com/doc/refman/5.7/en/create-table-generated-columns.html [Accessed 7 January 2022]

B. Oreilly (2022) *High Performance MySQL.* Available at: https://www.oreilly.com/library/view/high-performance-mysql/9780596101718/ch01.html [Accessed 8 January 2022]