ONLINE VOTING

The Originals

virtusa

Project Team

Name	Email
Dandu Sirisha	19951A04G5@iare.ac.in
G Shiva Prasad Reddy	19951A04f9@iare.ac.in
Dudi Rajesh	19951A1266@iare.ac.in
Elayabharathi S K	elayabharathisk@gmail.com
Devaganthan Balachander	devaganthanb@yahoo.com
Deepan NP	npdeepan143@gmail.com
Donga Suresh	sureshdsuresh999@gmail.com
Dadapuram Samyuktha	19951A04E1@iare.ac.in
Dhanalaxmi Pendyala	19951A0440@iare.ac.in

Mentor: -

Revathi Gunasekaran
Lead Consultant-Business Operations
revathigunasekaran@virtusa.com

Learning Portal: -

Examly.io
candidate@iamneo.ai

virtusa

Sprint-1

Table of contents

Sprint-2

Table of contents

Sprint-3

Table of contents

Sprint-4

Table of contents

Sprint-5

Table of Contents

SPRINT- 1

1. INTRODUCTION

1.1 Introduction

A Voter Database System is a software application designed to manage the details of voters, candidates, elections, and voting results. The system provides an easy and secure way for voters to register, cast their vote, and view the election results. It also helps election officials to manage the election process efficiently and accurately. The Voter Database System is designed to be user-friendly and accessible to all eligible voters. The system stores voter information such as name, address, phone number, and AADHAAR number. It also keeps track of candidate details such as name, party affiliation, and candidate type.

1.2 Scope

The scope for an online voting system using SQL as the database management system can include the following components and features:

1. **Database Design:** Create a well-structured database schema to store information about users, candidates, ballots, votes, and other relevant data.

2. **User Registration:** Implement a user registration module to capture user details, including their name, address, and eligibility information. Store this information securely in the database.

3. **Authentication and Security:** Develop a robust authentication mechanism to verify the identity of users during the login process. Hash and securely store user passwords in the database.

4. **Ballot Creation:** Design a module to allow administrators to create and manage electronic ballots. Store the ballot information, including candidate names and positions, in the database.

5. **Vote Casting:** Develop an interface for voters to cast their votes securely. Capture and store the vote information in the database, associating it with the respective user and ballot.

6. **Vote Counting:** Implement algorithms and logic to accurately count the votes based on the voting rules and system requirements. Use SQL queries to aggregate and analyze the data stored in the database to determine the election results.

7.Result Declaration: Display the voting results in a user-friendly format, maintaining the privacy of individual voters. Retrieve and present the relevant information from the database to generate the result reports.

1.3 Out Scope

1.Offline Voting: The system focuses on online voting and does not include offline voting methods.

2.Voter Registration Verification: Extensive verification processes beyond basic criteria may be out of scope.

3.External Data Integration: Integrations with external systems or databases beyond the SQL database may not be included.

4.Complex Authentication Mechanisms: Advanced authentication mechanisms like biometrics may be out of scope.

5.Voter Education: Extensive voter education campaigns or materials may be considered out of scope.

6.Legal and Regulatory Compliance: Ensuring compliance with specific election laws and regulations is typically not part of the system.

7.External Reporting and Analysis: In-depth external reporting or data analysis may be out of scope.

8.Electoral Boundary Management: Tasks related to managing electoral boundaries or constituencies may not be included.

9.Financial Transactions: Financial processes such as campaign funding or donations may be considered out of scope.

1.4 Actors:

1.Voter: Represents individuals who participate in the voting process by casting their votes through the online voting system.

2.Administrator: Refers to authorized personnel responsible for managing and administering the online voting system. They have additional 3.privileges to create and manage elections, candidates, and user accounts.

3. System: Represents the core online voting system that facilitates the interactions between voters, administrators, and the underlying 5.database. It handles tasks such as user authentication, vote recording, result generation, and displaying election information.

4. Database: Refers to the underlying SQL database used to store and manage critical data for the online voting system, including user details, election information, candidate profiles, voting records, and audit logs.

1.5 Glossary

1. Online Voting System: A web-based application that enables voters to cast their votes electronically through an internet-enabled device.

2. SQL (Structured Query Language): A programming language used for managing relational databases. It allows for the creation, modification, and retrieval of data stored in the database.

3. User Registration: The process in which individuals provide their personal information to create an account in the online voting system, allowing them to participate in elections.

4. User Authentication: The process of verifying the identity of users logging into the online voting system to ensure that only authorized individuals can access the system and cast votes.

5. Election Management: The set of activities involved in creating, modifying, and managing elections within the online voting system. This includes defining election details, candidate lists, and timelines.

6. Vote Recording: The process of securely capturing and storing voter preferences in the database, associating each vote with the respective election and candidate.

7. Result Generation: The computation of election results based on the recorded votes. It involves tallying the votes and determining the winners for each election.

1.6 Project goals:

As of my last update in September 2021, an online voting system typically aims to achieve several key goals:

1. Accessibility: To make voting more accessible to a broader range of citizens, including those with disabilities or those who may face physical barriers to traditional voting methods.

2. Convenience: To offer a more convenient way for voters to cast their ballots from the comfort of their homes or any location with internet access, reducing the need to travel to physical polling stations.

3. Voter Participation: To increase overall voter participation and engagement by providing an easier and more user-friendly voting process.

4. Security: To ensure the integrity and security of the voting process, preventing fraud and unauthorized access.

5. Accuracy: To achieve accurate and reliable vote counting, reducing the chances of errors that might occur in manual vote counting.

6. Transparency: To make the voting process more transparent, allowing stakeholders to observe and verify the results.

7. Efficiency: To streamline the voting process, reduce administrative burdens, and provide faster results.

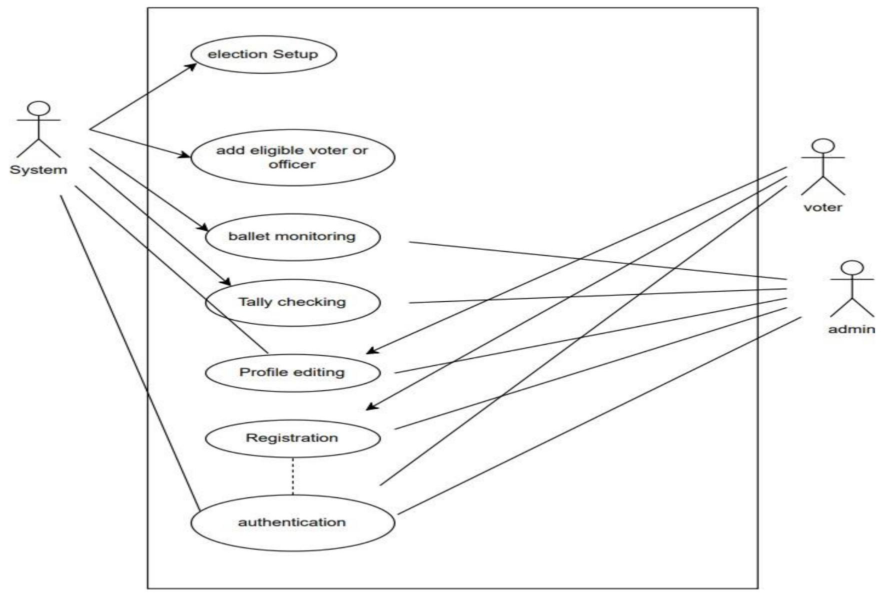
8. Cost-Effectiveness: To potentially reduce costs associated with traditional voting methods, such as printing ballots and staffing polling stations.

It's essential to note that while online voting systems have the potential to offer many benefits, they also come with significant challenges, particularly related to security and trust. As such, any online voting system must be carefully designed and thoroughly tested to meet the highest standards of security and accuracy. The goals of such a project may also vary depending on the specific country or region's legal, political, and cultural context.

2. Functional Overview

The main purpose of an online voting system using SQL is to provide a secure, efficient, and accessible platform for conducting elections.

2.1 Case Diagram



Description of the above diagram

Voter: Represents individuals who participate in the voting process. They interact with the system to cast their votes and view election information.

System: Represents the core online voting system. It manages the interactions between voters, administrators, and the database. It handles tasks such as authentication, vote recording, and displaying election information.

Administrator: Represents users with administrative privileges. They have additional capabilities to manage elections, candidates, and user accounts.

2.2 Functional Requirements/ Database

1. Election Table:

ElectionType: Type of election happening.

ElectionId: Id for type of election

2. Party Table:

PartyId: Party ID

PartyName: Party Name

Symbol: Party Symbol

PartyLeader: Party Leader

3. Candidate Type:

Candidateype: Candidate is applying for which post.

CandidateTypeID: Id of post applied for

4. Candidate Table:

CandidateID: Id for each Candidate

Aadhaar: Candidate’s Aadhaar no.

CandidateTypeID: Id of post he is standing for

Party D: Id of which party he belongs to

ElectionID: Id of which type of election is he nominated in

DistrictID: Id of District in which he is standing for

5. User Type:

UserType: Who is voting Citizen / Candidate

UserTypeID: Id of Voter

6. User Table:

It basically Contains Voter information required during login

IsActive: Is the registered voter Alive or Dead

UserTypeID: Id of User Type (Citizen / Candidate)

VoterId: Voter Id of every user

Def_Password: The system generated password that was initially given to the user, which can later be changed

Aadhaar: Aadhaar no. for reference from the Voter Table and connect information of both tables

NOTE: User table is linked with the Voter Table with the common column Aadhaar no. It basically includes the above details every person with the given Aadhaar no.

7. Voter Table:

Contains Aadhaar No., First Name, Last Name, Mother's Name, Father's Name, Sex, Birthday, Age, District ID, Phone No.

It basically contains information during registration of voter

It is linked with the User table with Aadhaar Id

8. Address:

DistrictID: Area-wise Id

Locality: Area from where he/she is voting

City: City of user

State: State of User

Zip: Pin code of locality of the user

NOTE: This table contains information about the Permanent Address of the Voter

9. Vote Table:

VoteID: Auto-assigned Id. (Nothing to do with Citizen)

VoterID: Who voted, that person's Voter Id

PartyID: Whom the person voted to

DistrictID: District ID for the region the person has voted for

CandidateID: The ID of candidate of the party they voted for

10. Result:

ResultID

CandidateID

PartyID

DistrictID

Vote_Count

2.3 Functional Relationships

a. One-to-One Binary Relationships:

1. User_Table - Voter_Table: One-to-One (1:1)

Reason: Each user must be a registered voter, and each registered voter can have only one user account.

2. Candidate_Table - User_Table: One-to-One (1:1)

Reason: Each candidate should have a corresponding user account, and each user can be associated with only one candidate.

3.Vote_table - user_table: One-to-One(1:1)

Reason: Every user can only cast one vote and one vote is taken from one user.

Every user have unique vote.

b. One-to-Many Binary Relationships:

1. Election_Table - Candidate_Table: One-to-Many (1: N)

Reason: One election can have multiple candidates, but each candidate can participate in only one election.

2. Party_Table - Candidate_Table: One-to-Many (1: N)

Reason: One political party can have multiple candidates, but each candidate can belong to only one party.

3 User_Type - User_Table: One-to-Many (1: N)

Reason: One user type can have multiple users, but each user can have only one user type.

4 Candidate_Type - Candidate_Table: One-to-Many (1:N)

Reason: One candidate type can be associated with multiple candidates, but each candidate can have only one candidate type.

5 Address - Voter_Table: One-to-Many (1:N)

Reason: One address can be associated to multiple voters, but each voter can have only one address.

6. Candidate_Table - Voter_Table: One-to-Many (1:N)

Reason: Each voter can choose to vote between multiple candidates, and each candidate can receive votes from multiple voters.

c. Many-to-One relationships:

1. Candidate_Table – Result: Many-to-One (N:1)

Reason: Candidates may be many, ultimatly the winner is only one.

2. Party_table – result: Many-to-One(N:1)

Reason: many parties can exist but only one winner is selected among multiple parties.

3. Address_table – result: Many-to-One(N:1)

Reason: Many votes can be casted from same district but only one winner can exist for one district.

2.4 Assumptions and Dependencies

1.It is assumed that users who register and participate in the online voting system meet the necessary eligibility criteria, such as age and citizenship requirements.

2.The assumption is made that voters have access to stable and reliable internet connectivity to access the online voting system.

3.User registration depends on collecting and validating user information and storing it securely in the database.

4.User authentication depends on the verification of user credentials against the stored information in the database.

5.Creating and managing elections depends on the ability to store and retrieve election details from the database.

6.Casting votes depends on the ability to record votes securely and accurately in the database.

7.Vote counting depends on retrieving and aggregating recorded votes from the database for each candidate in an election.

8.Generating election results depends on calculating the vote counts and determining the winners based on the aggregated results.

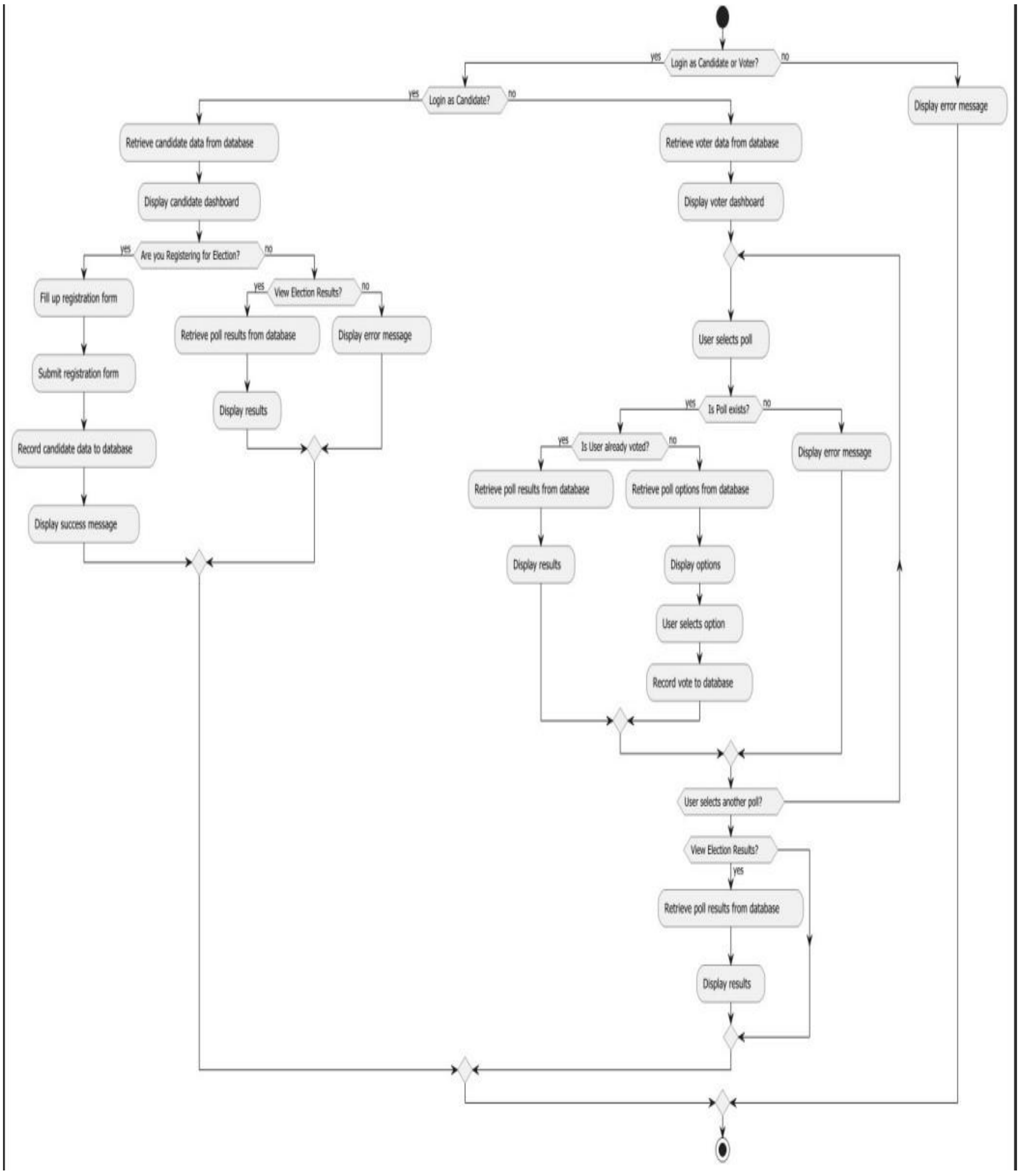
9. Displaying election results depends on retrieving and presenting the results from the database in a user-friendly format.

3. Technical Specifications

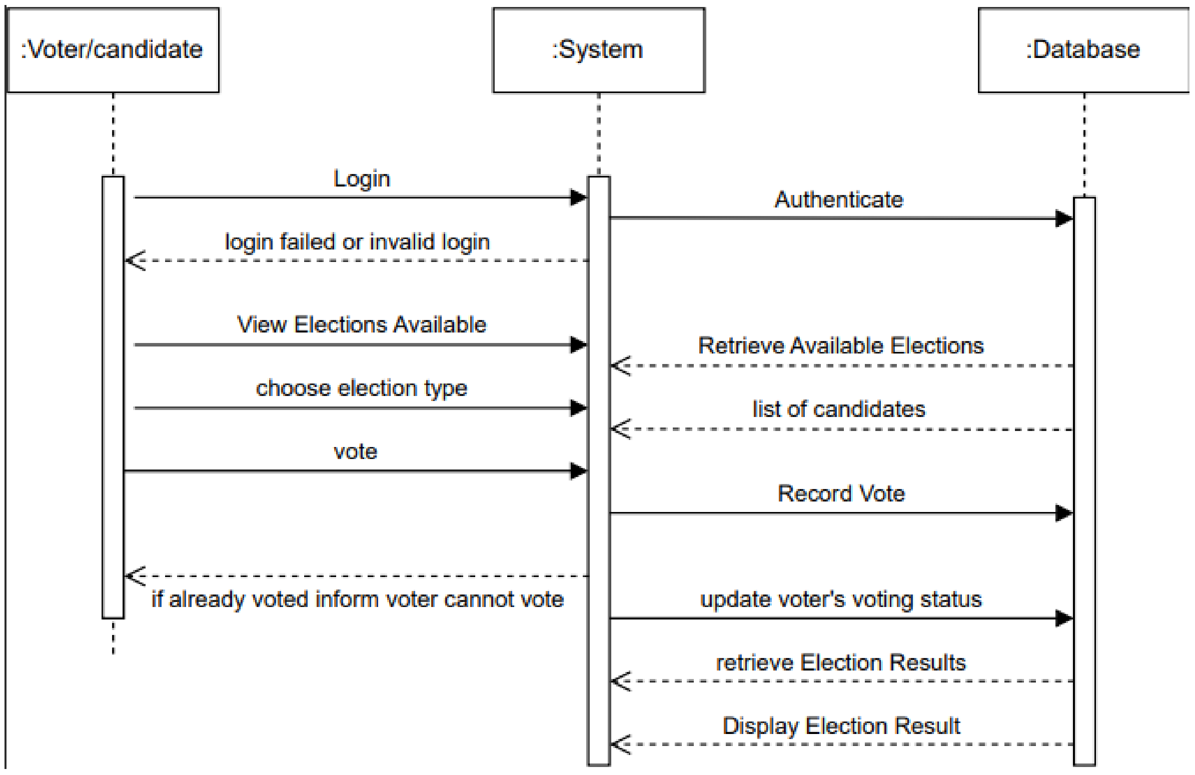
- Database: MySQL (8.0)

  Tool: MySQLworkbench

4. Activity Diagram



5. Sequence Diagram


:Voter/candidate
:System
:Database
Login
Authenticate
login failed or invalid login
View Elections Available
Retrieve Available Elections
choose election type
list of candidates
vote
Record Vote
if already voted inform voter cannot vote
update voter's voting status
retrieve Election Results
Display Election Result

In this sequence diagram:

1. The "Voter" actor initiates the sequence by logging into the system.

2. The "System" then authenticates the voter's credentials by communicating with the "Database."

3. Once authenticated, the voter requests to view the available elections.

4. The "System" retrieves the available elections from the "Database" and presents them to the voter.

5. The voter selects a specific election to vote in.

6. The "System" retrieves the details of the selected election from the "Database" and presents them to the voter.

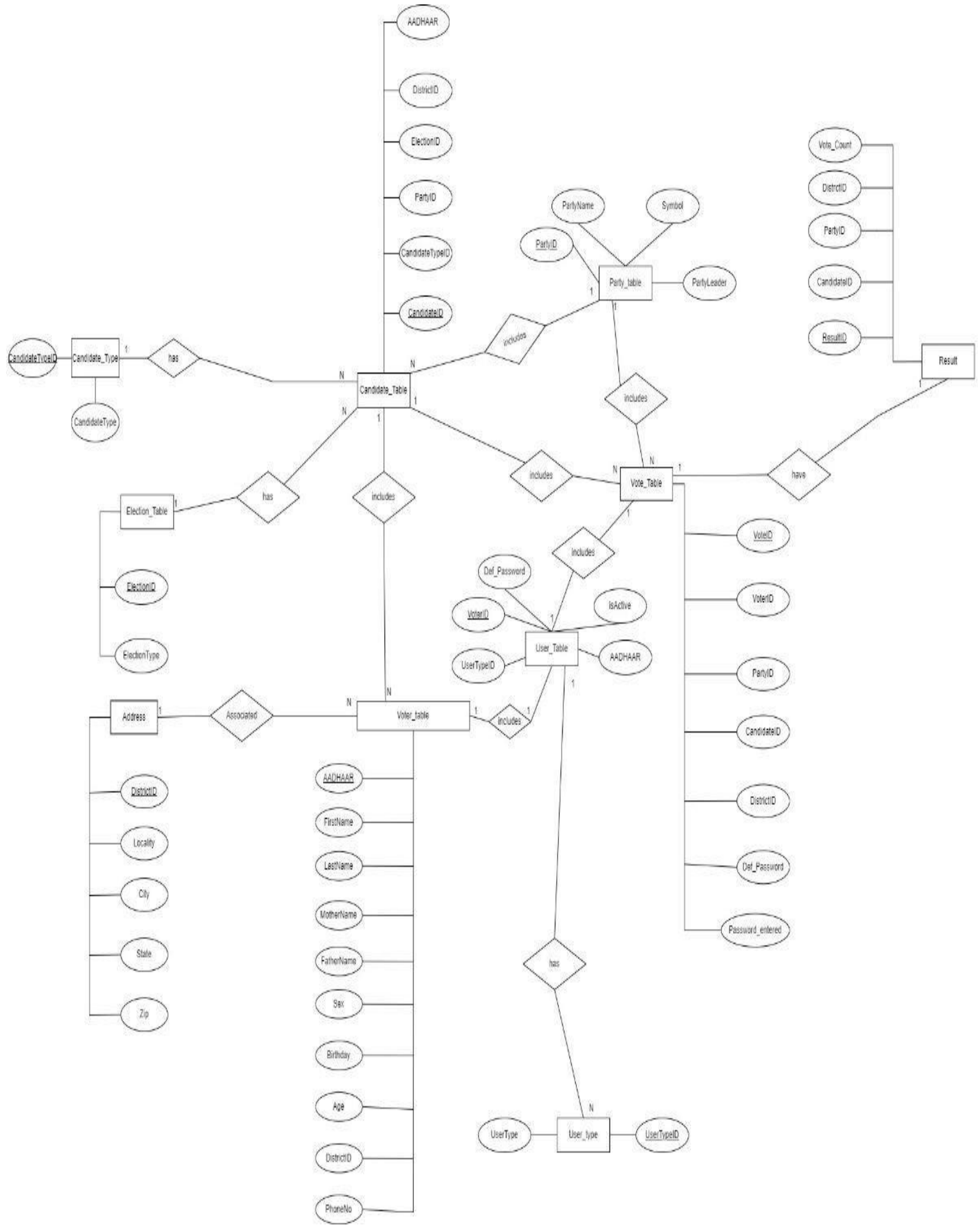
7. The voter casts their vote.

8. The "System" records the vote in the "Database" and updates the voting status of the voter.

9. The "System" retrieves the election results from the "Database."

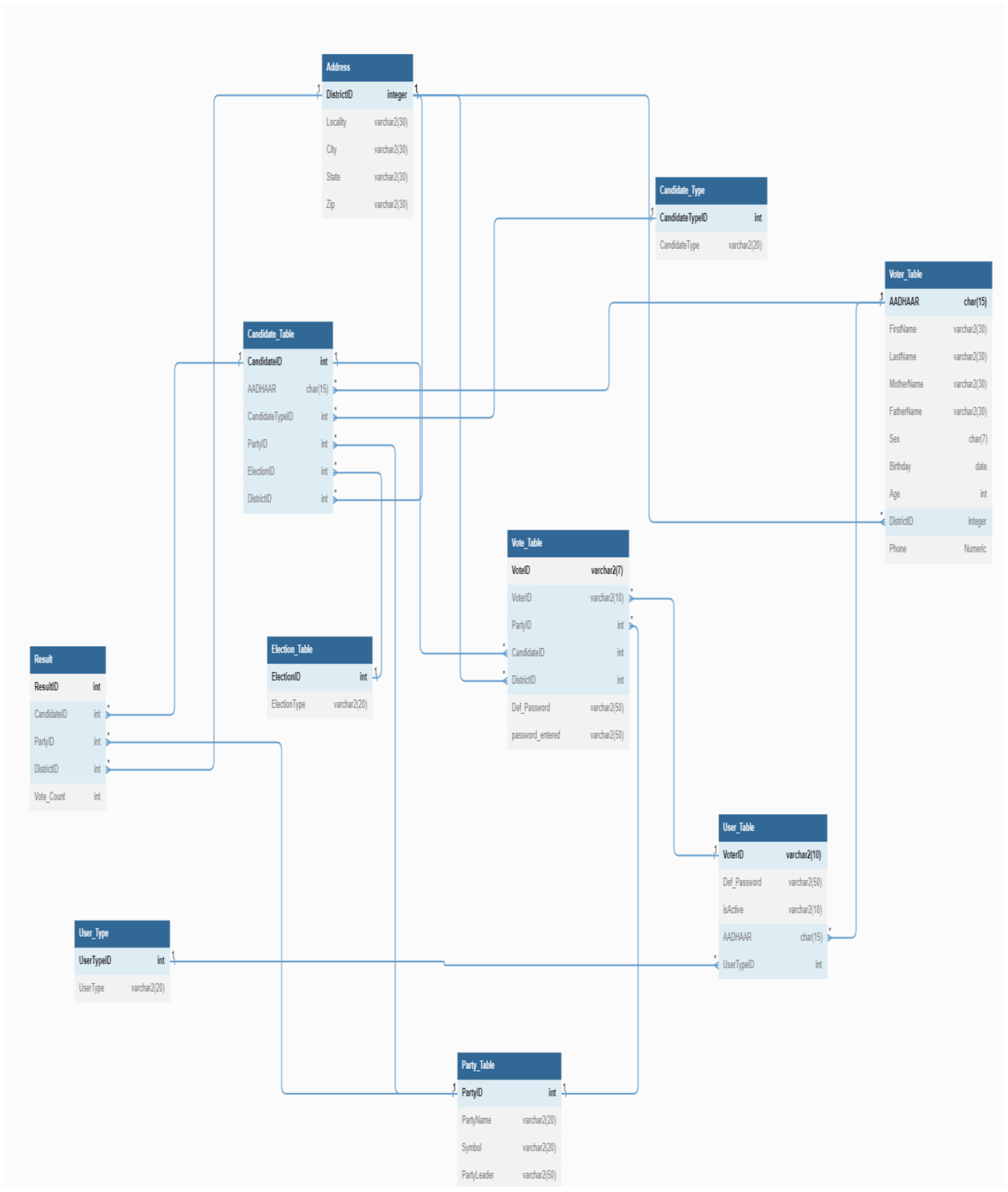
10. The "System" displays the election results to the voter.

6. ER Diagram



AADHAAR
DistrictID
ElectionID
PartyID
CandidateTypeID
CandidateID
CandidateTypeID
Candidate_Type
has
CandidateType
Candidate_Table
includes
PartyName
Symbol
PartyID
Party_table
PartyLeader
includes
Vote_Count
DistrictID
PartyID
CandidateID
ResultID
Result
have
Vote_Table
includes
Election_Table
has
ElectionID
ElectionType
includes
Def_Password
IsActive
VoterID
User_Table
AADHAAR
UserTypeID
VoteID
VoterID
PartyID
CandidateID
DistrictID
Def_Password
Password_entered
Address
Associated
Voter_table
includes
DistrictID
Locality
City
State
Zip
AADHAAR
FirstName
LastName
MotherName
FatherName
Sex
Birthday
Age
DistrictID
PhoneNo
has
UserType
User_type
UserTypeID

7. ER Schema


Address
DistrictID integer
Locality varchar2(30)
City varchar2(30)
State varchar2(30)
Zip varchar2(30)
Candidate_Type
CandidateTypeID int
CandidateType varchar2(20)
Voter_Table
AADHAAR char(15)
FirstName varchar2(30)
LastName varchar2(30)
MotherName varchar2(30)
FatherName varchar2(30)
Sex char(7)
Birthday date
Age int
DistrictID integer
Phone Numeric
Candidate_Table
CandidateID int
AADHAAR char(15)
CandidateTypeID int
PartyID int
ElectionID int
DistrictID int
Vote_Table
VoteID varchar2(7)
VoterID varchar2(10)
PartyID int
CandidateID int
DistrictID int
Def_Password varchar2(50)
password_entered varchar2(50)
Result
ResultID int
CandidateID int
PartyID int
DistrictID int
Vote_Count int
Election_Table
ElectionID int
ElectionType varchar2(20)
User_Table
VoterID varchar2(10)
Def_Password varchar2(50)
isActive varchar2(10)
AADHAAR char(15)
UserTypeID int
User_Type
UserTypeID int
UserType varchar2(20)
Party_Table
PartyID int
PartyName varchar2(20)
Symbol varchar2(20)
PartyLeader varchar2(50)

8. Conclusion

This online voting system is designed to manage voter information, allowing voters to log in and exercise their voting rights. The system encompasses all the necessary features of a voting system. Users who are above 18 years old can register their information in thedatabase. When they wish to vote, they need to log in using their unique ID and password andcan cast their vote for a single party only. The system stores the voting details in the database and calculates the results, which are then displayed. The online voting system significantly increases the percentage of voting by providing convenience and accessibility. It also reducesthe cost and time associated with the voting process. The system is user-friendly, requiring minimal time for operation and easy debugging. It leverages a database, such as SQL, to securely store and manage crucial data, including user information, election details, candidateprofiles, and voting records. SQL databases ensure reliability, scalability, and efficient data retrieval, enabling accurate vote counting and result generation. The design and implementation of the online voting system are guided by assumptions about user eligibility, internet connectivity, system security, and database reliability, ensuring a seamless and secureuser experience. Ultimately, the project's goal is to transform the electoral process by utilizingtechnology to enhance accessibility, accuracy, security, and efficiency. By developing a well-designed system that addresses these key aspects, the project aims to contribute to thedemocratic process and foster trust in the election outcomes.

SPRINT 2

2. NORMALIZATION

2.11 Relational schema:

A relational schema is a set ofrelationaltables and associated itemsthat are related to one another.All ofthe base tables, views, indexes, domains, userroles,storedmodules, and otheritemsthat a user createsto fulfill the data needs of a particular enterprise orset of applications belong to one schema.

2.12 Mapping Process:

1. Create table for every entity sets.
2. Add all its attributes to table as field.
3. Add the primary key of identifying entity set.
4. Declare all foreign key constraints.

2.13 Normalization:

• Normalization refers to the process of organizing data in a database to eliminate redundancy and data consistency. It involves applying a set of rules known as normalization rules to design a database schema that minimizes data redundancy and dependency issues. Violation of normalization occurs when these rules are not followed, resulting in a database structure that may have data anomalies or inconsistencies.

• There are different levels or forms of normalization, commonly referred to as normal forms, including First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and so on. Each normal form has specific rules that must be satisfied to achieve a higher level of normalization.

2.14 Here are some examples of violations of normalization:

1. First Normal Form (1NF) violation: In 1NF, each column of a table should contain only atomic values, meaning that it cannot contain multiple values or repeating groups. A violation would occur if a column contains multiple values separated by commas or if a table contains repeating groups.

2. Second Normal Form (2NF) violation: In 2NF, a table must meet the requirements of 1NF, and all non key attributes must be fully dependent on the entire primary key. Violations can occur if a table has a composite primary key, and a non-key attribute depends on only part of the composite key.

3. Third Normal Form (3NF) violation: In 3NF, a table must meet the requirements of 2NF, and no transitive dependencies should exist between non-key attributes. A violation would occur if a non-key attribute depends on another non-key attribute within the same table.

4. Denormalization: While not a direct violation of normalization rules, denormalization involves intentionally deviating from normalization principles for performance optimization or other specific reasons. Denormalization may reintroduce redundancy or dependencies that were eliminated during normalization.

2.15 Normalization benefits:

1. **Data Integrity:** Normalization helps maintain data integrity by minimizing data redundancy and eliminating update anomalies. With normalized tables, each piece of data is stored only once, reducing the risk of inconsistencies or contradictions that can occur when data is duplicated across multiple locations.
2. **Data Consistency:** By eliminating redundant data, normalization ensures that changes to a piece of data are reflected consistently throughout the database. Updates, inserts, and deletions are performed in a controlled and structured manner, reducing the possibility of data inconsistencies.
3. **Efficient Data Storage:** Normalization reduces data redundancy, resulting in more efficient storage utilization. By storing data in a compact and organized manner, disk space is optimized, and the overall database size can be reduced
4. **Improved Query Performance:** Normalized tables are typically structured to support efficient querying. With normalized data, queries can be written more precisely and effectively, targeting specific tables and avoiding unnecessary joins or complex conditions. This can result in improved query performance and faster response times.
5. **Simplified Data Modification:** Normalization makes it easier to modify and update data in the database. Since each piece of data is stored in only one place, modifications can be applied to a single table, reducing the complexity of updates and minimizing the chance of errors or inconsistencies.
6. **Scalability and Flexibility:** Normalized databases are generally more flexible and scalable. As the database grows and new requirements emerge, it is easier to add,

modify, or extend the database structure without causing extensive changes to the existing tables. This flexibility enables the database to adapt to evolving business needs.

7. **Maintainability and Database Design Clarity:** Normalization enhances the maintainability of the database by providing a clear and logical database structure. It improves the understandability and readability of the database schema, making iteasier for developers and administrators to work with the database and implement changes or enhancements.

A schema is said to be in 1 st Normal Form only if it satisfies the following conditions:

1. A relation will be 1NF if it contains an atomic value.
2. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
3. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

2.2 Online Voting System First Normal form

Normalization of Tables

Online Voting System is a project that involves managing various entities such as Address, VoterTable, CandidateType, ElectionTable,PartyTable, UserType, UserTable, VoteTable, and Result. The goal of normalization in database design is to eliminate data redundancy, ensure data integrity, and improve data consistency. In this document, we will analyze each entity and discuss the normalization process for achieving the first normal form (1NF), second normal form (2NF), and third normal form (3NF).

Address:

Column	DataType	Description
DistrictID	Primary Key	Unique identifier for the District
Locality		Locality of the Address
City		City of the Address
State		State of the Address
Zip		Pincode of the Address

Normalization Analysis:

1. 1NF (First Normal Form):

2. The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

Voter_Table:

Column	DataType	Description
AADHAAR	Primary Key	AADHAAR of the voter
FirstName		FirstName of the voter

LastName		LastName of the voter
MotherName		MotherName of the voter
FatherName		FatherName of the voter
Sex		Gender of the voter
Birthday		Birthday of the voter
Age		Age of the voter
DistrictID	Foreign Key	DistrictID of the voter
Phone		Phone number of the voter

Normalization Analysis:

1. 1NF (First Normal Form):

2. In this table Phone is multivalued attribute, this Table violates 1NF. A single cell must not hold a more than one value, So it renamed as Home Phone and Work Phone

Column	Data Type	Description
HomePhone		HomePhone of the voter
WorkPhone		WorkPhone of the voter

Candidate Type:

Column	DataType	Description
CandidateTypeID	PrimaryKey	Uniqueidentifierforthe CandidateID
CandidateType		Shows the Type of Candidate

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

Election Table:

Column	DataType	Description
ElectionID	Primary Key	Unique identifier for the Election
ElectionType		Shows the type of Election

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.‘

User Type:

Column	DataType	Description
UserTypeID	Primary Key	Unique identifier for the User
UserType		Shows the type of the User

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

Candidate Table:

Column	DataType	Description
CandidateID	Primary Key	Unique identifier for the Candidate
AADHAAR	Foreign Key	AADHAAR of the Candidate
CandidateTypeID	Foreign Key	Uniqueidentifierforthe CandidateID

PartyID	Foreign Key	Unique identifier for the Party
ElectionID	Foreign Key	Unique identifier for the Election

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

UserTable:

Column	**DataType**	**Description**
VoterID	Primary Key	Unique identifier for the Voter
Def_Password		Password of the User
isActive		Activity status of the User
AADHAAR	Foreign Key	AADHAAR of the User
UserTypeID	Foreign Key	Shows the Type of the User

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

Vote Table:

Column	**DataType**	**Description**
VoteID	Primary Key	Unique identifier for the Vote
VoterID	Foreign Key	Unique identifier for the Voter
PartyID	Foreign Key	Unique identifier for the Party

CandidateID	Foreign Key	Shows the Type of Candidate
DistrictID	Foreign Key	Unique identifier for the District
Def_Password		Password of the Vote Table
Password_entered		Password entered to the Vote Table

Normalization Analysis:

1. 1NF: The table is already in 1NF as each column contains atomic values, and there are no repeating groups.

Conclusion:

Now we obtained relational schema in 1st normal form which satisfies all the conditions of 1st normal form.

2.3 Online Voting system second normal form

A schema is said to be in 2nd Normal Form only if it satisfies the following conditions:

1. The table should be in first normal form
2. It should not have any Partial Dependency.

Partial Functional Dependency: The Partial Functional Dependency occurs when a non-prime attribute is functionally dependent on the part of a candidate key.

Address:

Column	**DataType**	**Description**
DistrictID	Primary Key	Unique identifier for the District
Locality		Locality of the Address
City		City of the Address
State		State of the Address
Zip		Pincode of the Address

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (DistrictID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Voter_Table:

Column	**DataType**	**Description**
AADHAAR	Primary Key	AADHAAR of the voter
FirstName		FirstName of the voter
LastName		LastName of the voter

MotherName		MotherName of the voter
FatherName		FatherName of the voter
Sex		Gender of the voter
Birthday		Birthday of the voter
Age		Age of the voter
DistrictID	Foreign Key	DistrictID of the voter
Phone		Phone number of the voter

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (AADHAAR) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Candidate Type:

Column	**DataType**	**Description**
CandidateTypeID	PrimaryKey	Uniqueidentifierforthe CandidateID
CandidateType		Shows the Type of Candidate

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (CandidateTypeID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Election Table:

Column	DataType	Description
ElectionID	Primary Key	Unique identifier for the Election
ElectionType		Shows the type of Election

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (ElectionID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Party Table:

Column	DataType	Description
PartyID	Primary Key	Unique identifier for the Party Table
PartyName		Shows the Name of the Party Table
Symbol		Shows the Symbol of the Party Table
Party Leader		Leader of the Party Table

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (PartyID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

User Type:

Column	DataType	Description
UserTypeID	Primary Key	Unique identifier for the User
UserType		Shows the type of the User

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (UserTypeID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Candidate Table:

Column	DataType	Description
CandidateID	Primary Key	Unique identifier for the Candidate
AADHAAR	Foreign Key	AADHAAR of the Candidate
CandidateTypeID	Foreign Key	Uniqueidentifierforthe CandidateID
PartyID	Foreign Key	Unique identifier for the Party
ElectionID	Foreign Key	Unique identifier for the Election

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (CandidateID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

UserTable:

Column	DataType	Description
VoterID	Primary Key	Unique identifier for the Voter
Def_Password		Password of the User
isActive		Activity status of the User
AADHAAR	Foreign Key	AADHAAR of the User
UserTypeID	Foreign Key	Shows the Type of the User

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (VoterID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Vote Table:

Column	DataType	Description
VoteID	Primary Key	Unique identifier for the Vote
VoterID	Foreign Key	Unique identifier for the Voter
PartyID	Foreign Key	Unique identifier for the Party
CandidateID	Foreign Key	Shows the Type of Candidate
DistrictID	Foreign Key	Unique identifier for the District
Def_Password		Password of the Vote Table
Password_entered		Password entered to the Vote Table

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (VoteID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Result:

Column	**DataType**	**Description**
ResultID	Primary Key	Unique identifier for the Result
CandidateID	Foreign Key	Shows the Type of Candidate
PartyID	Foreign Key	Unique identifier for the Party
DistrictID	Foreign Key	Unique identifier for the District
Vote_Count		Shows the number of Vote Count

Normalization Analysis:

1. It satisfies 1NF.
2. The table is already in 2NF as there are no partial dependencies. The primary key (ResultID) uniquely identifies each record, and all other attributes are fully dependent on the primary key.

Conclusion:

Now we obtained relational schema in 2nd normal form which satisfies all the 2 NF condition normal form.

2.4 Online Voting System Third Normal form

A schema is said to be in 3rd Normal Form only if it satisfies the following conditions:

1. The table should be in first normal form
2. It should not have any transitive Dependency.

Transitive dependency: When an indirect relationship causes functional dependency it is called Transitive dependency. If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

Address:

Column	**DataType**	**Description**
DistrictID	Primary Key	Unique identifier for the District
Locality		Locality of the Address
City		City of the Address
State		State of the Address
Zip		Pincode of the Address

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (Locality,City,State,Zip) depend directly on the primarykey(DistrictID) and no each other.

Voter_Table:

Column	**DataType**	**Description**
AADHAAR	Primary Key	AADHAAR of the voter
FirstName		FirstName of the voter
LastName		LastName of the voter
MotherName		MotherName of the voter
FatherName		FatherName of the voter
Sex		Gender of the voter
Birthday		Birthday of the voter
Age		Age of the voter
DistrictID	Foreign Key	DistrictID of the voter
Phone		Phone number of the voter

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non key attributes(FirstName, LastName, MotherName, FatherName, Sex, Birthday, Age, DidtrictID, HomePhone, WorkPhone) depend directly on the primary key(AADHAAR) and not each other.

Candidate Type:

Column	DataType	Description
CandidateTypeID	PrimaryKey	Uniqueidentifierforthe CandidateID
CandidateType		Shows the Type of Candidate

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (CandidateType)depend directly on the primary key (Customer_ID) and not on each other.

Election Table:

Column	DataType	Description
ElectionID	Primary Key	Unique identifier for the Election
ElectionType		Shows the type of Election

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (ElectionType) depend directly on the primary key (ElectionID) and not on each other

Party Table:

Column	DataType	Description
PartyID	Primary Key	Unique identifier for the Party Table
PartyName		Shows the Name of the Party Table
Symbol		Shows the Symbol of the Party Table
Party Leader		Leader of the Party Table

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (PartyName, Symbol, PartyLeader) depend directly on the primary key (PartyID) and not on each other

User Type:

Column	DataType	Description
UserTypeID	Primary Key	Unique identifier for the User
UserType		Shows the type of the User

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (UserType) depend directly on the primary key (UserTypeID) and not on each other.

Candidate Table:

Column	DataType	Description
CandidateID	Primary Key	Unique identifier for the Candidate
AADHAAR	Foreign Key	AADHAAR of the Candidate
CandidateTypeID	Foreign Key	Uniqueidentifierforthe CandidateID
PartyID	Foreign Key	Unique identifier for the Party
ElectionID	Foreign Key	Unique identifier for the Election

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes (AADHAAR, CandidateTypeID, PartyID, ElectionID) depend directly on the primary key (CandidateID) and not on each other.

UserTable:

Column	DataType	Description
VoterID	Primary Key	Unique identifier for the Voter
Def_Password		Password of the User
isActive		Activity status of the User
AADHAAR	Foreign Key	AADHAAR of the User
UserTypeID	Foreign Key	Shows the Type of the User

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-key attributes (Def_password, isActive, AADHAAR, UserType)depend directly on the primary key (VoterID) and not on each other.

Vote Table:

Column	**DataType**	**Description**
VoteID	Primary Key	Unique identifier for the Vote
VoterID	Foreign Key	Unique identifier for the Voter
PartyID	Foreign Key	Unique identifier for the Party
CandidateID	Foreign Key	Shows the Type of Candidate
DistrictID	Foreign Key	Unique identifier for the District
Def_Password		Password of the Vote Table
Password_entered		Password entered to the Vote Table

Normalization Analysis:

1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-keyattributes(VoterID,PartyID, CandidateID, DistrictID, Def_password, Pssword_entered) depend directly on the primary key (VoteID) and not on each other.

Result:

Column	**DataType**	**Description**
ResultID	Primary Key	Unique identifier for the Result
CandidateID	Foreign Key	Shows the Type of Candidate
PartyID	Foreign Key	Unique identifier for the Party
DistrictID	Foreign Key	Unique identifier for the District
Vote_Count		Shows the number of Vote Count

Normalization Analysis:

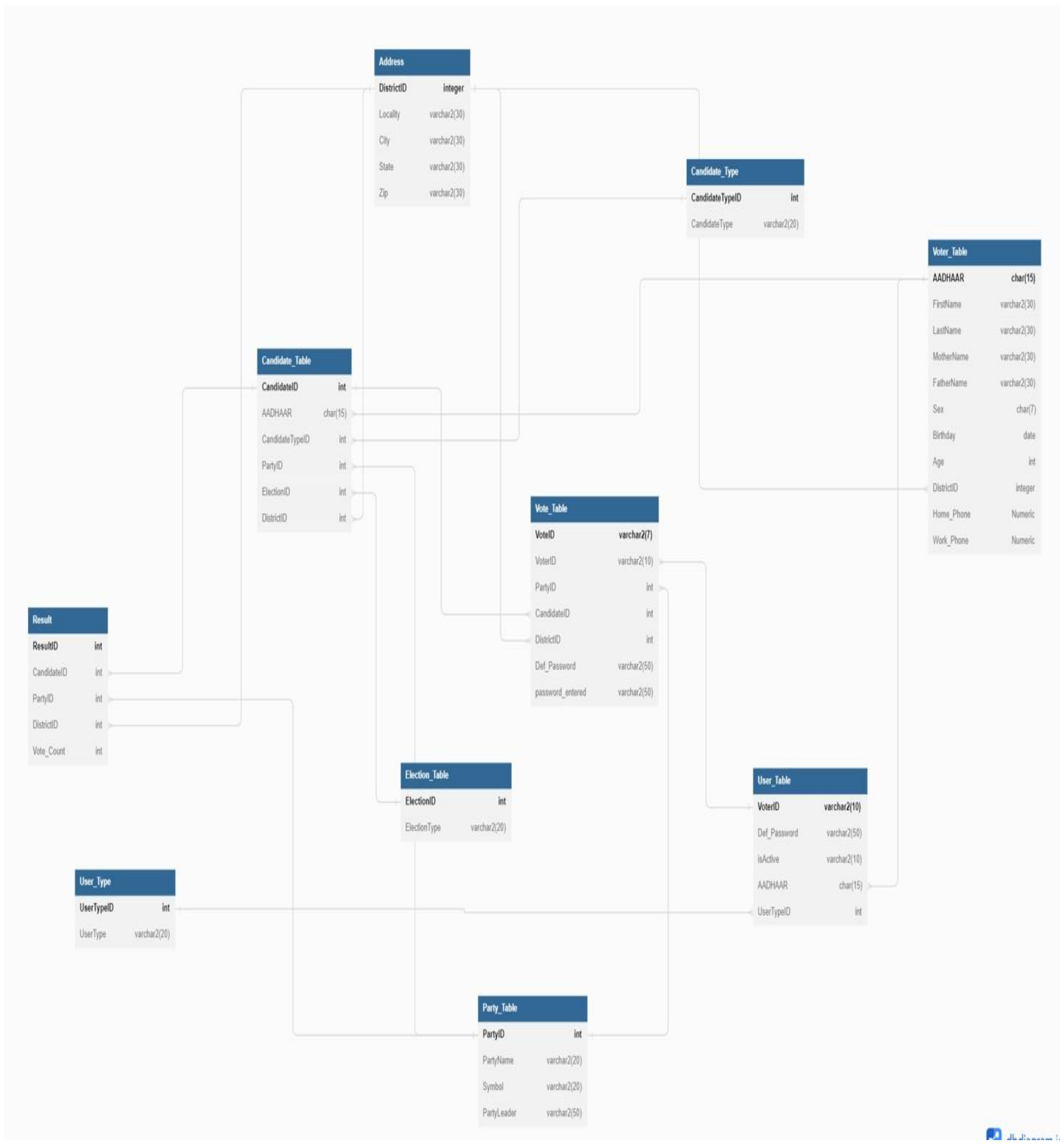
1. It satisfies 2NF.
2. The table is already in 3NF as there are no transitive dependencies. All non-key

attributes (CandidateID, PartyID, DistrictID, Vote_Count) depend directly on the primary key (ResultID) and not on each other.

Conclusion:

Now we obtained relational schema in 3st normal form which satisfies all the 3 NF condition normal form.

2.5 ER schema after Normalization



SPRINT-3

3.1 TABLE CREATION:

3.1.1 TABLE:

1. In SQL (Structured Query Language), a table represents a structured collection of data organized into rows and columns. It is a fundamental component of a relational database system and serves as a container for storing and organizing related information.

2. A table consists of the following elements:

- ❖ Columns: Also known as fields or attributes, columns define the types of data that can be stored in a table. Each column has a name and a specific data type, such as integer, string, date, or boolean. Columns provide the structure for the data and define the information that can be stored in each field.

- ❖ Rows: Also referred to as records or tuples, rows represent individual instances or entries in the table. Each row contains data that corresponds to the defined columns. The data in each row must align with the data types specified for the corresponding columns.

- ❖ Constraints: Constraints define rules and restrictions on the data stored in the table. Common constraints include the NOT NULL constraint to enforce non-null values, the UNIQUE constraint to enforce uniqueness, and the FOREIGN KEY constraint to establish relationships with other tables

- ❖ Primary Key: A primary key is a column or a combination of columns that uniquely identifies each row in the table. It ensures the integrity and uniqueness of thedata in the table, allowing for efficient data retrieval and modification.

3. Tables provide a structured and organized way to store and manage data in a database. They allow for efficient querying, sorting, filtering, and manipulation of data through SQL statements

3.1.2 Syntax:

1. The syntax for creating a table in SQL (Structured Query Language) is as follows:

```
CREATE TABLE table_name (
column1 datatype constraint,
column2 datatype constraint,
...
constraint
);
```

2. The syntax for creating a table in SQL consists of the CREATE TABLE statement followed by the table name, column definitions enclosed in parentheses, and optional constraints.

3.1.3 Schema for Table creation

1. To create a table in SQL(Structured Query Language), we can use the 'CREATE TABLE' statement.

2. This statement allows us to define the structure and properties of the table including its columns and data types.

3. Follow these steps to create a table:

    1. Begin by writing the 'CREATE TABLE' statement followed by the name of the table we want to create.

    2. Inside parentheses, define the columns of the table. Each column is specified by its name, data type and optional constraints. Separate each column definition with a comma.

    3. Add any additional table-level constraints or settings such as primary keys, foreign keys and indexes. These are specified after defining the columns and are outside the parentheses.

    4. Execute the 'CREATE TABLE' statement to create the table in database.

4. The specific syntax and available options may vary depending on the database system.

5. Choose a unique name for each column that consists of letters, numbers, and underscores. Avoid using reserved keywords.

6. Specify the data type for each column, such as ‘INTEGER’, ‘VARCHAR’, ‘DATE’ etc. The data type determines the kind of values that can be stored in the column.

7. We can apply constraints to enforce rules or conditions on the column, such as ‘PRIMARY KEY’, ‘NOT NULL’, ‘UNIQUE’,’FOREIGN KEY’ etc.

3.1.4 Table Creation

1: Address table

```
DISTRICTID NUMBER NOT NULL,
LOCALITY VARCHAR(30) NOT NULL, CITY VARCHAR(30) NOT NULL,
STATE VARCHAR(30) NOT NULL, ZIP VARCHAR(30) NOT NULL,
CONSTRAINT PK_DISTRICT PRIMARY KEY (DISTRICTID));
```

2. Voter_table

```
CREATE TABLE VOTER_TABLE (
AADHAAR CHAR(15) NOT NULL, FIRSTNAME VARCHAR2(30) NOT NULL,
LASTNAME VARCHAR2(50) NOT NULL,
MOTHERNAME VARCHAR2(30),
FATHERNAME VARCHAR2(30),
SEX CHAR(7) NOT NULL,
BIRTHDAY DATE  NOT NULL,
AGE INTEGER NOT NULL,
DISTRICTID INTEGER NOT NULL,
PHONE INTEGER NOT NULL,
CONSTRAINT PK_VOTER PRIMARY KEY (AADHAAR), CONSTRAINT
FK_DISTRICT FOREIGN KEY (DISTRICTID) REFERENCES ADDRESS
(DISTRICTID));
```

3. Candidate_table

```
CREATE TABLE CANDIDATE_TYPE (

CANDIDATETYPEID INTEGER NOT NULL, CANDIDATETYPE VARCHAR2(20) NOT NULL,

CONSTRAINT PK_CANDIDATETYPE PRIMARY KEY (CANDIDATETYPEID));
```

4.Election table

```
CREATE TABLE ELECTION_TABLE (

ELECTIONID INTEGER NOT NULL,

ELECTIONTYPE VARCHAR2 (20) NOT NULL,

CONSTRAINT PK_ELECTION PRIMARY KEY (ELECTIONID));
```

5.Party table

```
REATE TABLE PARTY_TABLE (

PARTYID INTEGER NOT NULL,

PARTYNAME VARCHAR2(20) NOT NULL,

SYMBOL VARCHAR2(20) NOT NULL,

PARTYLEADER VARCHAR2(50) NOT NULL,

CONSTRAINT PK_PARTY PRIMARY KEY (PARTYID));
```

6. user type table

```
CREATE TABLE USER_TYPE (

USERTYPEID INTEGER NOT NULL,

USERTYPE VARCHAR2(20) NOT NULL,

CONSTRAINT PK_USERTYPE PRIMARY KEY (USERTYPEID));
```

7.Candidate_table

```
CREATE TABLE CANDIDATE_TABLE (

CANDIDATEID INTEGER NOT NULL,

AADHAAR CHAR(15) NOT NULL,

CANDIDATETYPEID INTEGER NOT NULL,

PARTYID INTEGER NOT NULL,

ELECTIONID INTEGER NOT NULL,

DISTRICTID INTEGER NOT NULL,

CONSTRAINT PK_CANDIDATE PRIMARY KEY (CANDIDATEID),

CONSTRAINT FK_VOTER FOREIGN KEY (AADHAAR) REFERENCES
VOTER_TABLE(AADHAAR),

CONSTRAINT FK_DISTRICT_2 FOREIGN KEY (DISTRICTID) REFERENCES
ADDRESS (DISTRICTID),

CONSTRAINT FK_ELECTION FOREIGN KEY (ELECTIONID) REFERENCES
ELECTION_TABLE(ELECTIONID),

CONSTRAINT FK_PARTY FOREIGN KEY (PARTYID) REFERENCES PARTY_TABLE
(PARTYID),

 CONSTRAINT FK_CANDIDATETYPE FOREIGN KEY (CANDIDATETYPEID)
REFERENCES CANDIDATE_TYPE(CANDIDATETYPEID));
```

8.user table

```
CREATE TABLE USER_TABLE(

VOTERID VARCHAR2(10) NOT NULL,

DEF_PASSWORD VARCHAR2(50) NOT NULL,

ISACTIVE VARCHAR2(10) NOT NULL,

AADHAAR CHAR(15) NOT NULL,

USERTYPEID INTEGER NOT NULL,

CONSTRAINT PK_USER PRIMARY KEY (VOTERID),
```

CONSTRAINT FK VOTER_2 FOREIGN KEY (AADHAAR) REFERENCES VOTER_TABLE(AADHAAR),

CONSTRAINT FK_USERID FOREIGN KEY (USERTYPEID) REFERENCES USER_TYPE(USERTYPEID));

9.vote table

CREATE TABLE VOTE_TABLE(

VOTEID VARCHAR2(7) NOT NULL,

VOTERID VARCHAR2(10) NOT NULL,

PARTYID INTEGER NOT NULL,

CANDIDATEID INTEGER NOT NULL,

DISTRICTID INTEGER NOT NULL,

DEF_PASSWORD VARCHAR2(50) NOT NULL,

PASSWORD_ENTERED VARCHAR2(50) NOT NULL,

CONSTRAINT PK_VOTE PRIMARY KEY (VOTEID),

CONSTRAINT FK_VOTERID FOREIGN KEY (VOTERID) REFERENCES USER_TABLE (VOTERID),

CONSTRAINT FK_CANDIDATEID FOREIGN KEY (CANDIDATEID) REFERENCES CANDIDATE_TABLE(CANDIDATEID)

CONSTRAINT FK_DISTRICT 4 FOREIGN KEY (DISTRICTID) REFERENCES ADDRESS (DISTRICTID),

CONSTRAINT FK_PARTY_2 FOREIGN KEY (PARTYID) REFERENCES PARTY_TABLE (PARTYID));

10. Result

CREATE TABLE RESULT

RESULTID INTEGER NOT NULL,

CANDIDATEID INTEGER NOT NULL,

PARTYID INTEGER NOT NULL, DISTRICTID INTEGER NOT NULL,

VOTE COUNT INTEGER NOT NULL,

CONSTRAINT PK_RESULT PRIMARY KEY (RESULTID),

CONSTRAINT FK_CANDIDATEID_2 FOREIGN KEY (CANDIDATEID) REFERENCES CANDIDATE_TABLE(CANDIDATEID),

CONSTRAINT FK_DISTRICT_5 FOREIGN KEY (DISTRICTID) REFERENCES ADDRESS(DISTRICTID),

CONSTRAINT FK_PARTY_3 FOREIGN KEY (PARTYID) REFERENCES PARTY_TABLE (PARTYID));

3.2 FETCH SINGLE TABLE RECORD:

• Fetching a single table record refers to retrieving a specific row or record from a database table.

• To retrieve a single record from a table in SQL(Structured Query Language), you can use the 'SELECT' statement along with appropriate filtering conditions.

• Follow these steps to fetch a single record:

1. To fetch a single table record, you typically use the 'SELECT' statement in SQL. The 'SELECT' statement allows you to specify the columns you want to retrieve from the table.

2. By applying filtering conditions using the WHERE clause, you can narrow down the result set to match specific criteria.

3. Additionally, you can use clauses like 'LIMIT' or 'TOP' (depending on the database system) to limit the number of records returned to just one.

• The purpose of fetching a single table record is to obtain specific information or data from the table for further processing or display.

• This could involve retrieving a user's details, fetching a specific order, accessing a particular product's information, or other similar use cases. By fetching only the necessary record, you can minimize data retrieval and improve query performance

1: Write a SQL query to list every record from the address table.

select * from address

2: Write a SQL Query to List out all the records from the voter_table

select * from voter_table

3: List out all the records from the candidate_type table

select * from candidate_type

4: List out all the records from the election_table

select * from election_table

5: List out all the records from the party_table

select * from party_table

6: List out all the records from the user_type table

select * from user_type

7: List out all the records from the candidate_table

select * from candidate_table

8: List out all the records from the user_table

select * from user_table

9: List out all the records from the vote_table

select * from vote_table

10: List out all the female records from the voter_table Note: 'F' indicates Female Voters

select * from voter_table where sex=’F’;

11: List out all the voters details whose age is between 30 and 40 from the voter_table

select * from voter_table where age between 30and 40;

12: List out all the details from the party_table where the party name starts with the letter B

select * from party_table where partyname like ‘%B’;

13: Write a SQL Query to list the candidate id and first four aadhaar numbers from candidate_table.

select candidateid,substr(aadhaat,1,4) as Extractstring from candidate_table;

14 : List out all votes details from vote_table where the length of the voteid is 2.

select * from voter_table wherelength(voterid)=2;

15: List out all voters whose date of birth is above 1990 from voter_table.

```
select * from voter_table where extract(year from birthday)>1990;
```

3.3 Fetch Record From Multiple Table

• Fetching records from multiple tables in SQL involves retrieving data from more than one table in a single query.

• This allows us to combine related information from different tables to obtain a complete result set.

• Follow these steps to fetch records from multiple tables :

1. Begin by writing the ‘SELECT’ statement followed by the column(s) we want to retrieve from the tables.
2. Specify the tables from which we want to fetch the records using the ‘FROM’ keyword, followed by the table names. We can list multiple tables, separating them with commas.
3. If there are relationships between the tables, use the appropriate joinconditions in the ‘JOIN’clause to establish the relationships. Commontypes of joins include ‘INNER JOIN’, ‘LEFT JOIN’, ‘RIGHT JOIN’ and ‘FULL JOIN’. Specify the join conditions using the ‘ON’keyword.
4. If necessary, apply additional conditions to filter the records using the ‘WHERE’ clause. Specify the conditions that the records must satisfy.
5. Execute the ‘SELECT’ statement to fetch the records from the multiple tables.

• By following the steps outlined above, you can effectively fetch records frommultiple tables in SQL.

1: Select all the details related to candidate details (such as candidateid, partyname, aadhaar, firstname, lastname, mothername, fathername, sex, birthday, age, districtid, phone) using joins

```
select c.customerid,p.partyname,v.* from candidate_table
Join party_table p on p.partyid=c.partyid
Join voter_table v on v.aadhaar=c.aadhaar;
```

2: Find which voter to which party name and display of party name and voterid.

```
 select p.partyname, v.voterid
From party_table p join vote_table v
On v.partyid = p.partyid;
```

3: count BSP party's vote count from voters who cast their votes and the BSP party's id is 14

```
Select count(*) as  from vote_table where partyid=14;
```

4: Write a query to make the left join between address table and voter_table

```
 select * from voter_table v
 left join address a on v.districtid=a.districtid;
```

5: Display the following details of voters who should be active. Having to diplay firstname , lastname , age and phone number of the voters with the above condition and if they are active then it shows 'yes' or else'no'

```
 select firstname , lastname , age, phone from voter_table v
Join user_table u on v.aadhaar=u.aadhaar where u.isactive='yes';
```

6: Display the usertype of citizen's voterid, def_password and aadhaar details.usertypeid of citizen is 2.

```
 select voterid,def_password,aadhaar from user_table u
Left join user_type v on u.usertypeid=v.usertypeid
Where u.usertypeid=2;
```

7: write a query to right join between election_type and candidate_table and also write the condition to check whether the election type is 'state assembly' wheras its id is 201.

```
 select * from candidate_table  c
Right join election_table e on c.electionid=e.electionid
Where c.electionid=201;
```

8: Display the usertype of the candidate's voterid , is active status and aadhaar details

Note: usertypeid of candidate is 1

```
 select voterid , isactive, aadhaar from user_table u
Left join user_type v on u.usertypeid=v.usertypeid
Where u.usertypeid=1;
```

9: write a sql query to count of votes received by a political party with partyid=13 from a vote_table that is joined with party_table on the basis of the common partyid column.

```
 select count(*) as count from vote_table v left join party_table p on v.partyid=p.partyid where
v.partyid=13;
```

10: write a sql query to retrieve the record from candidate_table and address table where the districtid in the candidate_table matches with the districtid in the address tables usinh left join.

```
 select * from candidate_table c left join address a on c.districtid=a.districtid;
```

3.4 SUBQUERY

1. A subquery, also known as an inner query or nested query, is a query nested within another query in SQL (Structured Query Language). It allows you to retrieve data from one table based on the results of another query or perform calculations on a subset of data.
2. To use a subquery, you typically enclose the inner query within parentheses and place it within the larger outer query. The result of the inner query is then used in the outer query to further filter or manipulate the data.
3. Subqueries can be used in various parts of a SQL statement, including the SELECT, FROM, WHERE, and HAVING clauses. They can help solve complex problems by breaking down the tasks into smaller, more manageable steps.
4. Subqueries can be used to perform tasks such as filtering records based on a condition, retrieving aggregated data, performing calculations, and creating temporaryresult sets for further processing.

1 Write a SQL Query to retrieve the candidateld and aadhaar columns from the candidate_table where the partyid matches with the partyid retrieved from the party_table where the symbol column equals 'Lotus'.

```
select candidateld, aadhaar from candidate _table c
Where partyid in
(select partyid from party_table where symbol=’LOTUS’);
```

2: Write a SQL Query to retrieve the voterid and def_password columns from the user table where the usertypeid matches with the usertypeid retrieved from the user_type table where the usertype column equals 'Citizen.

```
 select voterid, def_password user_table where usertypeid in
(select usertypeid from user_type where usertype=’Citizen’;
```

3: Write a SQL Query to display only the voter id and their aadhaar information from user_table who is not in the vote_table record

```
 select voterid, aadhaar from user_table 2 where voterid not in
(select voterid from vote_table);
```

4: Write a SQL Query to Display the firstname, lastname and phone number details from voter_table who is not active now. Note: If voter is active then it shows yes' else 'no'.

```
select firstname, lastname, phone from voter_table
Where aadhaar in
(select aadhaar from user_table where isactive='no');
```

5: Display the firstname, lastname and age in ascending order from voter table where the voters are of type 'Candidate whose id is 1

```
 select firstname, lastname, age from voter_table v
Where aadhaar in
(select aadhaar from candidate_table )
Order by age;
```

6: Display the firstname, lastname and age in ascending order from voter_table where the voters are of type 'Citizen' whose id is 2

```
 select firstname, lastname, age from voter_table
Where aadhaar in
(select aadhaar from user_table where usertypeid=2 )
Order by age;
```

7: Display the firstname, lastname and aadhaar details from voter_table who cast their vote to BSP Party

```
 select firstname, lastname, aadhaar from voter_table
 where aadhaar in
(select aadhaar from candidate_table c,party_table p
where c.partyid-p.partyid and partyname='BSP');
```

8: Display the firstname, lastname, aadhaar and age in descending order from voter_table who cast their vote to INC party

```
select firstname, lastname, aadhaar,  age from voter_table v
Where age in (30,40,43,46,48) and firstname!=’Harsh’
Order by age desc;
```

9: Write a SQL Query to Display the voter details of their firstname, lastname and birthday from voter_table who belongs to Maharashtra state.

```
select firstname, lastname, birthday from voter_table
where districtid in
(select districtid from address where state=''Maharashtra');
```

10: Write a SQL Query to Display the voter details of their firstname, lastname and age in asc order from voter_table who belongs to Pune city.

```
select firstname, lastname, age from voter_table
Where districtid in
(select districtid  from  address  where city=’Pune)  Order by age;
```

SPRINT-4

4.1 Stored Procedures:

4.1.1 Procedures:

1. A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language.

2.Procedure always contains a name, parameter lists, and SQL statements.

4.1.2 Syntax:

The following syntax is used to create procedures in SQL.

```
Create Procedure [Procedure Name] ([Parameter 1],[Parameter 2], [Parameter 3] )
 Begin
            SQL Queries.
           End;
```

1. The name of the procedure must be specified after the Create Procedure keyword.
2. After the name of the procedure, the list of parameters must be specified in the parenthesis. The parameter list must be comma-separated.
3. The SQL Queries and code must be written between BEGIN and END keywords.

1. **Write a stored procedure to remove a record from voter_table**

```
Create or replace procedure DELETEVOTE(p_voteid in varchar) as
tmp number:=0;
begin
select count(*) into tmp from voter_table;
delete from vote_table where voteid=p_voteid;
select count(*) into tmp
from vote_table;
```

```
end;

/
```

2. **Write a stored procedure in PLSQL to delete a record from candidate_table, have to delete the partyid of 15**

```
Create or replace procedure DELETECANDIDATE
as
cnt int:=0;
cnt int:=0;
begin
select count(*) into cnt from candidate_table;
delete from candidate_table where partyid=15;
select count(*) into cnt1 from candidate_table;
dbms_output.put_line(Before deleting one record from candidate table, the
count is: ‘      ||cnt);
dbms_output.put_line(After deleting one record from candidate table, the
count is: ‘  ||cnt1);
end ;
/
```

3. **Write a stored procedure in PLSQL to insert a new record to the address table**

```
Create or replace procedure INERTNEWADDRESS

(d_id in int,l_loc in varchar,c_city in varchar, state in varchar, zip
invarchar)

As

cnt number:=0;

begin

select count(*) into cnt from address;

insert into address values(d_id, l_loc, c_city, state, zip);

select count(*) into cnt from address;

end;

/
```

4. **Write a stored procedure in PLSQL to update the party table of PartyID of 14's symbol as tiger**.

```
Create or replace procedure UPDATEPARTYTABLE
(p_id in number, s_symbol in varchar)
As
Begin
update party_table
set symbol=s_symbol where party_id=p_id;
end;
/
```

5. **Write a stored procedure in PLSQL to count the total records inserted in user_table.**

```
Create or replace procedure TOTALUSERS return number
is
  tot number:=0;
  begin
  select count(*) into tot
  from  user_table;
  return tot;
 end;
 /
```

6. **Write a user Defined function in PLSQL to count the total votes of the BSP party (party id: 14)**

```
Create or replace function TOTALVOTESFORBJP return number
As
cnt number:=0;
begin
select count(*) into cnt from vote_table
```

```
where partyid=14;
return cnt;
end;
/
```

7. **Write a user-defined function in PLSQL to count the total active users inserted in the user_table**

```
Create or replace function TOTALUSERS return number
as
cnt number:=0;
begin
select count(*) into cnt from user_table
where isactive='Yes';
return cnt;
end;
/
```

4.2 Cursors

4.2.1 Cursors:

1. A cursor in database is a construct which allows you to iterate/traversal the records of a table.

2. In MySQL we can use cursors with in a stored program such as procedures, functions.

4.2.2 Syntax:

The following syntax is used to create cursors in SQL.

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
OPEN cursor_name;
 FETCH cursor_name INTO variables list;
CLOSE cursor_name;
```

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

1. The cursor declaration must be after any variable declaration.
2. The OPEN statement initializes the result set for the cursor.
3. The FETCH statement is used to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.
4. Deactivate the cursor and release the memory associated with it using the CLOSE statement.
5. To declare a NOT FOUND handler. The finished is a variable to indicate that the cursor has reached the end of the result set.

1. Write a cursor in PLSQL to display party id and party name from party_table

```
declare
cursor deep
is
select partyid,partyname
from party_table;
begin
for i in deep loop
dbms_output.put_line(i.partyid||' '||i.partyname);
end loop;
end;
/
```

2. Write a cursor in PLSQL to display firstname From voter_table

```
declare
cursor deep
is
select firstname
from voter_table;
j int:=0;
is
begin
select count(*)into j from voter_table;
for i in deep loop
```

```
dbms_output.put_line( ‘Voters Fetched:  ‘||i.firstname);
end loop;
dbms_output.put_line(‘Total rows fetched is  ‘||j);
end;
/
```

3. **Write a cursor in PLSQL to display all voters firstname from voter_table and count how many details you fetched as well.**

```
declare
cursor deep is
select firstname
from voter_table;
j int:=0;
 begin
 select count(*) into j from voter_table;
 dbms_output.put_line(‘Voters Fetched:  ‘||i.firstname);
 end loop;
 dbms_output.put_line(‘Total rows fetched is  ‘||j);
 end;
/
```

4.3Triggers:

4.3.1 Triggers:

1. Triggers are special types of stored procedures that are automatically executed or fired in response to specific events, such as data manipulation (insert, update, delete) on a table.

2. Triggers allow you to define custom actions or business rules that should be executed before or after the triggering event occurs.

4.3.2 syntax:

```
 CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
 [OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
 Declaration-statements
BEGIN
Executable-statements
EXCEPTION
 Exception-handling-statements
END;
```

1. **Create a trigger in PLSQL that gets triggered when we enter a record in Voter_table.Already Voter_table created when we insert the record in the Voter_tableinsert the FirstName into the LIST table before inserting the records in the voter table and display the first name.**

NOTE:LIST Table already Created

```
Create or replace trigger LIST
Before insert on Voter_table
For each row
Begin
insert into LIST (Name) values (:new.FirstName);
 end;
 /
```

2. **Create a trigger in PLSQL that gets triggered when we delete a record in Voter_table.**

   insert the female firstname into the LIST table before deleting the records in the voter table.

```
Create or replace trigger LIST

Before delete on Voter_table

for each row

When (old.sex='F')

Begin
Insert into LIST (NAME) values (:new.FirstName);
End;
/
```

3. **Create a trigger in PLSQL that gets triggered when we enter a record in the Voter_table.Already these records are inserted in the Voter_table.**

   Insert the male Firstname,sex into the LIST table before deleting the record into the voter table and display FirstName, sex.

```
Create or replace trigger LIST
Before delete on Voter_table
For each row
When (old.sex='M');
Begin
Insert into LIST (Firstname, sex) values (:old.FirstName,:old.sex);
End;
 /
```

4. **Create a trigger in PLSQL that gets triggered when we entered a record in Voter_table.**
   Already these records are inserted in the Voter_Table.

   (3591 4628 3661, Akash', 'Singh, Aishwarya, Bhavesh', 'M',1984-02-16,37,234,9623412913);
   (7820 3429 4038','Shlok, Agarwal, Aparna', 'Girish', 'M",''1988-02-04,33,234,9722768470);
   (6169 5028 5641, Rashid, 'Khan, Indira, Abhay,M,1976-10-17,44,235,9414321457);

```
create or replace trigger LIST
after update on voter_table for each row
begin
if updating then
insert into LIST(name, sex) values (:new.firstname, :old.sex);
end if;
end;
/
```

SPRINT 5

5.1 Performance Tuning:

SQL tuning is the process of enhancing SQL queries to speed up the performance of your server. Its main goal is to shorten the time it takes for a user to receive a response after sending a query and to utilize fewer resources in the process. The idea is that users can occasionally produce the same intended result set with a faster-running query.SQL performance tuning is speeding up queries against a relational database.

There is not just one tool or method for optimizing SQL speed. Instead, it's a set of procedures that utilize a variety of methods, and procedures. Let's talk about some of the major factors that will influence how many computations you must perform and how long it takes for your query to run:

1. **Table size:** Performance may be impacted if your query hits one or more tables with millions of rows or more.
2. **Joins:** Your query is likely to be slow if it joins two tables in a way that significantly raises the number of rows in the return set.
3. **Aggregations:** Adding several rows together to create a single result needs more processing than just retrieving those values individually.
4. **Other users executing queries:** The more queries a database has open at once, the more it must process at once, and the slower it will all be. It can be particularly problematic if other users are using a lot of resources to perform queries that meet some of the aforementioned requirements.

Ways to Find Slow SQL Queries in SQL Server to Measure SQL Server Performance:

1. Create an Execution Plan: It is essential to be able to create execution plans, which you can accomplish with SQL Server Management Studio, in order to diagnosedelayed queries. After the queries run, actual execution plans are generated. To create an execution plan:

- Start by selecting "Database Engine Query" from the toolbar of SQL Server Management Studio.
- Enter the query after that, and then select "Include Actual Execution Plan" from the Query option.

- It’s time to run your query at this point. You can do that by pressing F5 or the “Execute” toolbar button.
- The execution plan will then be shown in the results pane, under the “Execution Pane” tab, in SQL Server Management Studio.

2. Monitor Resource Usage: The performance of a SQL database is greatly influenced by resource use. Monitoring resource use is important since you can’t improve what you don’t measure. Use the System Monitor tool on Windows to evaluate SQL Server’s performance. You may view SQL Server objects, performance counters, and other object activity with it. Simultaneously watch Windows and SQL Server counters with System Monitor to see if there is any correlation between thetwo services’ performance.

3. Use SQL dynamic management views (DMV) to Find Slow Queries: The abundance of dynamic management views (DMVs) that SQL Server includes is one of its best features. There are many of them, and they may offer a lot of knowledge on a variety of subjects.

Optimizing a Query for SQL:

1. SELECT fields instead of using SELECT *.
2. Avoid SELECT DISTINCT.
3. Create queries with INNER JOIN (not WHERE or cross join).
4. Use WHERE instead of HAVING to define filters.
5. Use wildcards at the end of a phrase only.
6. Use LIMIT to sample query results.
7. Run your query during off-peak hours.

5.2 Indexes:

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to refer all pages in a book that discuss a certain topic, you first refer to the index, which lists all the topics alphabetically along with their page numbers.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index. The **CREATE INDEX** Command

The basic syntax of a CREATE INDEX is as follows.

CREATE INDEX index_name ON table_name;

- **Single-Column Indexes:** A single-column index is created based on only one table column. The basic syntax is as follows.

  CREATE INDEX index_name ON table_name (column_name);

- **Unique Indexes:** Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

  CREATE UNIQUE INDEX index_name on table_name (column_name);

- **Composite Indexes:** A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

  CREATE INDEX index_name on table_name (column1, column2);

  Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions. Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

- Implicit Indexes: Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created forprimary key constraints and unique constraints. 3.

5.3 LEAD( ) and LAG( ) Function:

The LEAD() and LAG() function in MySQL are used to get preceding and succeeding value of any row within its partition. These functions are termed as nonaggregate Window functions.

The Window functions are those functions which perform operations for each row of the partition or window. These functions produce the result for each query row unlikely to the aggregate functions that group them and results in a single row. The row on which operation occur is termed as current row. The set of rows which are related to current row or using which function operates on current row is termed as Window.

The LAG() function is used to get value from row that precedes the current row.

Syntax: **LEAD(expr, N, default) OVER (Window_specification | Window_name)**

The LEAD() function is used to get value from row that succeeds the current row.

Syntax: **LAG(expr, N, default) OVER (Window_specification | Window_name)**

Parameters used:

1. **expr:** It can be a column or any built-in function.
2. **N:** It is a positive value which determine number of rows preceding/succeeding the current row. If it is omitted in query then its default value is 1.
3. **default:** It is the default value return by function in-case no row precedes/succeedes the current row by N rows. If it is missing then it is by default NULL.
4. **OVER():** It defines how rows are partitioned into groups. If OVER() is empty then function compute result using all rows.
5. **Window_specification:** It consist of query partition clause which determines how the query rows are partitioned and ordered.
6. **Window_name:** If window is specified elsewhere in the query then it is referenced using this Window_name.

ORACLE WORKSPACE

1. **Write a SQL query to find a total count of female candidates participating in each year.**

```
SELECT YEAR, SUM(CASE WHEN CAND_SEX = 'F' THEN 1 ELSE 0 END) AS
FemaleCandidateCount
FROM ELECTION
GROUP BY YEAR;
```

2. **Write a SQL query to find the total candidates who participated in the election at each state in each year.**

```
SELECT ST_NAME, YEAR, COUNT(*) AS total_candidates
FROM ELECTION
GROUP BY ST_NAME, YEAR;
```

3. **Write a SQL query to find total votes BJP got in each state in the year 1987.**

```
SELECT ST_NAME, SUM(TOTVOTPOLL) AS Total_Votes_ FROM ELECTION
WHERE YEAR = 1987 AND PARTYNAME = 'BJP'
GROUP BY ST_NAME;
```

4. **Write a SQL query to find the total candidates who participated in the election in each state in the year 2004.**

```
SELECT ST_NAME, COUNT(*) AS Ttotal_Candidatess_
FROM ELECTION
WHERE YEAR = 2004
GROUP BY ST_NAME;
```

5. **Write a SQL query to what are the top 5 parties that got the most votes in Uttar Pradesh in the year 2014.**

```
SELECT PARTYNAME, SUM(TOTVOTPOLL) AS Total_Votes
FROM ELECTION
WHERE ST_NAME = 'UTTAR PRADESH' AND YEAR = 2014
GROUP BY PARTYNAME
ORDER BY Total_Votes DESC
FETCH FIRST 5 ROWS ONLY
```

CONTRIBUTION

SPRINT **CONTRIBUTION**

Sprint-1 **SRS Document :-**

- Dudi Rajesh
- Dandu Sirisha
- Donga Suresh
- Elayabharathi Sk
- Dhanalaxmi Pendyala

ER Diagram

- G Shiva Prasad Reddy
- Devanaganthan B Balachander
- Deepan NP
- Dadapuram Samyuktha

Sprint-2 **Relation Schema in 1NF and Document :-**

- Dhanalaxmi Pendyala
- Dadapuram Samyuktha
- Dandu Sirisha

Relation schema in 2NF and Document :-

- Dudi Rajesh
- Elayabharathi Sk
- Donga Suresh

Relation schema in 3NF and Document :-

- Deepan NP
- Devanaganthan B Balachander
- G Shiva Prasad Reddy

Sprint-3	**Individual Task**
	Final Document for sprint-3 :-
	G Shiva Prasad Reddy, Dhanalaxmi Pendyala, Dadapuram Samyuktha
Sprint-4	**Individual Task**
	Final Document for sprint-4 :-
	Deepan NP, Devanaganthan B Balachander, Elayabharathi Sk
Sprint-5	**Individual Task**
	Final Document for sprint 5:-
	Dudi Rajesh, Donga Suresh, Dandu Sirisha
Workspace	**Team Work**
	Final Document for workspace - Team