

A Report on

# Developing a Voting Management System

Prepared For

Dr Amit Dua

(Instructor In-Charge)

CS F212 - Database Management



Prepared By

Chaitanya Sethi - 2020B3A71961P

Stavya Puri - 2020B5A70912P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI  
SECOND SEMESTER AY 2022-23  
JANUARY - MAY 2023

## DECLARATION STATEMENT

We hereby declare that this project was solely written by the members of this group. We are aware that the incorporation of code/material from other sources will be treated as plagiarism, subject to the custom and usage of the subject, according to the guidelines provided by the institution. We are fully responsible for its contents and declare all contents plagiarism-free. Wherever necessary, all the references are duly quoted, and a bibliography is provided at the end.

Chaitanya Sethi	2020B3A71961P	
Stavya Puri	2020B5A70912P	

## Acknowledgement

The success of this report would not have been possible without the assistance and guidance of our teachers. We would like to thank the Vice-Chancellor (BITS Pilani University), Dr Souvik Bhattacharya and the Director of BITS Pilani (Pilani Campus), Prof. Sudhir Kumar Barai. We would thank our instructors and mentors for this assignment dedicated to International Economics, Prof. Dr Amit Dua (Instructor In-Charge) and his skilled team of instructors and scholars for allowing us to conduct this study under the course **CS F212 - Database Management** and for guiding us throughout the semester to make the project comprehensive through his insights and knowledge. Lastly, we would also like to thank our family members and close friends for their moral support.

## Table of Contents

Declaration.....	1
Acknowledgement.....	2
Table of Contents.....	3
1. Introduction.....	4
2. Enhanced-Entity(or Entity Relationship) Relationship Diagram.....	5
3. Key Logical Assumptions and Working.....	6
4. Relational Schema.....	8
4.1. Functional Dependencies, Conversions and Normalisations.....	8
4.1.1 User Entity.....	8
4.1.2 Candidate Entity.....	8
4.1.3 Election Entity.....	9
4.1.4. Administrator Entity.....	10
4.1.5. Votes_For Relation .....	10
4.1.6. Manages Relation.....	11
4.1.7. Runs_For Relation.....	11
5. SQL Queries - Output Screenshots.....	14
5.1. Relational Definition - Create all tables.....	14
5.2. Insert User.....	18
5.3. Insert Candidate.....	19
5.4. Register a vote - register_vote.....	20
5.5. Insert election - insert_election.....	21
5.6. Retrieve no. of votes for Candidate (in given election).....	22
5.7. Retrieve the percentage of votes for a candidate.....	23
5.8. Retrieve the winner of the election.....	24
5.9. Retrieve the rank of each candidate.....	24
5.10. Retrieve the total number of votes cast for an election.....	24
5.11. Update the user's information.....	25
5.12. Delete a specific user's information.....	26
6. Video Links.....	27

## **1. INTRODUCTION**

A software programme called a "voting management system" is intended to administer the election process in a secure and transparent way. Voter registration, candidate registration, voting, and vote tallying are all made easier using it. Elections of all kinds, including those for national, state, local, and organisational offices, can be conducted using the system.

In order to ensure fair and transparent elections, a voting management system is essential for public administration. The technology offers a safe and effective way for voters to cast their ballots, as well as a way for candidates to sign up and keep track of the election. Additionally, it aids in avoiding fraud and other abnormalities that can jeopardise the election's credibility.

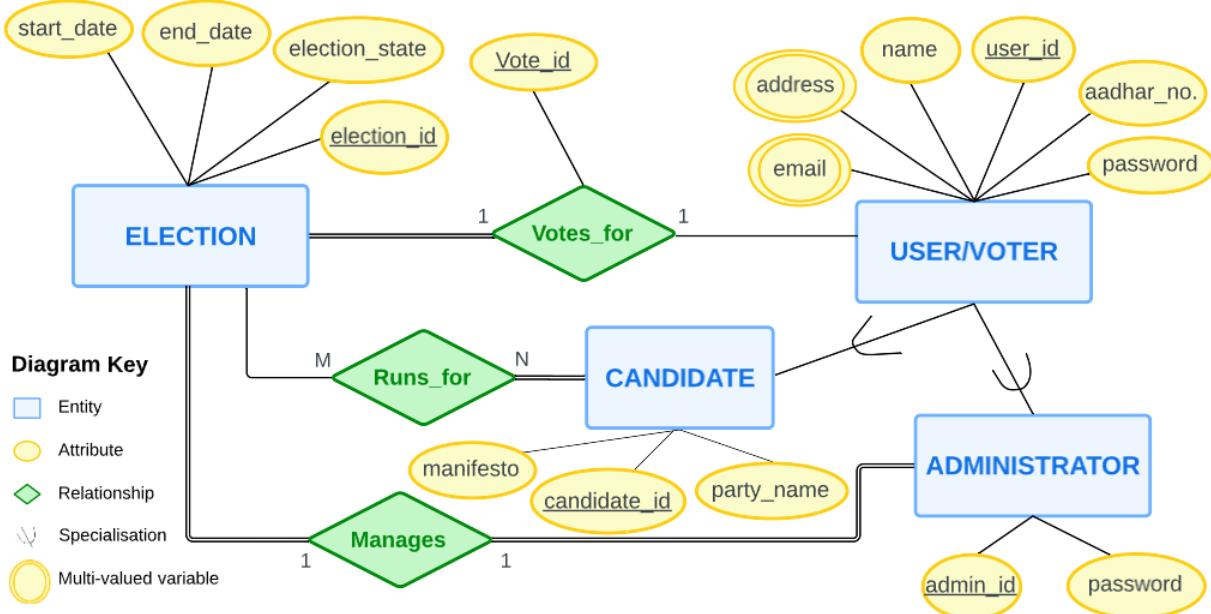
Public administration may make sure that all qualified voters can participate in the election process and that the results are reliable and accurate by implementing a voting management system. The technique can also aid in minimising the expense and time involved with conventional paper-based voting procedures.

In order to ensure fair and transparent elections and respect the democratic norms of free and fair elections, public administration must use a voting management system.

In the above spirit, as tasked, a Voting Management System has been developed using MySQL, starting with an Enhanced Entity Model, Relational Schema, which has been translated into actual SQL queries provided and implemented. The system is capable of registering(inserting) users(voters), candidates, and administrators for elections, creating new elections and generating reports of election winners, ranking and total votes cast.

## 2. Enhanced-Entity(or Entity Relationship) Relationship Diagram

This section explores the logical assumptions and the working of the EER.



**Figure 1. ENHANCED ENTITY RELATIONSHIP DIAGRAM**

**Figure 1. Enhanced Entity Relationship Diagram**

Here, we have created Entities ELECTION, USER/VOTER, CANDIDATE AND ADMINISTRATOR. We have also defined Relationships VOTES\_FOR, RUNS\_FOR, and MANAGES. We have considered specialisation that is ADMINISTRATOR and CANDIDATE as the children of USER as they also have voting rights. Cardinalities between Entities on the basis of various Relationships are shown above.

### **3. KEY LOGICAL ASSUMPTIONS AND WORKING**

As a complex and multi-variable system, certain logical assumptions have been considered alongside logical working for certain functions involved, further solidifying the understanding of the above-discussed EER diagram.

#### **General Assumptions**

- 1) The user is a generalisation of the candidate, administrator and other voters, as the candidates running in the election or the administrator handling the election also have the right to vote. This allows us to store all attributes of the candidate and administrator only once under user and reduce redundancy of repeated storage and update.
- 2) As per the standard voting regime, multiple candidates can stand for multiple elections across various states, but a voter can vote only once in a particular election. Additionally, each election is managed by a single administrator.
- 3) The administrator is a fair and just person who does not manipulate the voting system based on his personal interests.
- 4) Each election has an administrator, who manages the election and has the exclusive right to extract the election results, reports and ranks from the database system.
- 5) The Voting System does not provide tie-resolutions, as it is assumed that there might be coalitions and other political propaganda which is uncertain and outside the scope of the database.

#### **Function/Working-based Assumptions**

- 6) If a new candidate is being inserted into the system, he/she must enter all attributes corresponding to the user first, which only occurs if the user does not exist already. After this, it is checked whether he is already an administrator or a candidate for the same election already. If not, he is added as a candidate to the particular election through the election\_id, which is determined as the candidate must also provide election\_id. Else, insertion is rejected.
- 7) If a new administrator is being entered, he/she must enter all attributes corresponding to the user first, which only occurs if the user does not exist already. After this, it is checked whether he is already a candidate or an administrator of other elections. In case of any of the above, the insertion is rejected.
- 8) Deletion and updation of the user lead to corresponding changes in the candidate's/administrator's values' deletion or updation as well, as the user is a generalisation of these entities as earlier specified.

- 9) While inserting an election, a corresponding administrator from the existing users/voters must also be assigned to manage it. Hence, along with election attributes, administrator's attributes are also provided, which includes user\_id and a unique administrator id. Then the user\_id is subject to checks
1. If the user does not exist, insertion is rejected
  2. If the user exists but
    - a. Is already a candidate, insertion is rejected as candidates are not assigned as election administrators
    - b. Inserted admin\_id matches already an administrator, insertion is rejected as one election is managed by a single administrator.
    - c. user\_id matches any previously added administrator's user id, then insertion is rejected as user\_id, and admin\_id are to be unique for each election.

If all the above constraints are satisfied, the insertion occurs in the order:

- a) Election
- b) Administrator
- c) Manages

## 4. RELATIONAL SCHEMA

Given the above assumptions, constraints, and the EER diagram, the development of a relational schema, observations of functional dependencies involved, and normalisation may be conducted to form a relational schema in the desired 3NF Normal form.

### 4.1. Functional Dependencies, Conversions and Normalisations

*We have proved by first checking the 3NF Property that is Absence of Transitive Dependence and then going further to check for 2NF and 1NF as a Relation in 3NF is already in 2NF and 1NF.*

#### 4.1.1. User Entity

USER											
user_id	INT	name	VARCHAR	address	VARCHAR	email	VARCHAR	aadhar_no.	INT	password	VARCHAR

The user entity has six attributes with their corresponding input types as shown here.

**Primary Key:** user\_id

The functional dependencies involved here are

user\_id → name, address, email, aadhar\_no., password

aadhar\_no. → name, address, email, aadhar\_no., password

For the given entity and functional dependencies, all the left attributes are superkeys, and hence 3NF(BCNF) form is achieved as **no transitive dependencies are formed**.

#### 4.1.2. Candidate Entity

CANDIDATE						
candidate_id	INT	user_id	party_name	VARCHAR	manifesto	VARCHAR

The candidate entity, being a specialisation of the user entry, has additional two entries as party\_name and manifesto, with

**Primary Key:** candidate\_id

**Foreign Key:** user\_id

(referencing User table, identifies the candidate's basic attributes as he is also a voter)

The functional dependencies involved here are

**candidate\_id** → user\_id, party\_name, manifesto

*user\_id* → candidate\_id, party\_name, manifesto

Having chosen candidate\_id as the Primary Key, it can be observed that both the functional dependencies involved here have their left attributes as a superkey, ensuring no transitive dependency occurs. Thus, we achieve a **3NF(BCNF)** again using similar argument.

#### 4.1.3. Election Entity

ELECTION					
election_id	INT	Election_State	VARCHAR	start_date	VARCHAR

The Election entity has four attributes, with their corresponding input types as shown here.

**Primary Key:** election\_id

The functional dependencies involved here are

**election\_id** → Election\_State, start\_date, end\_date

Election\_State, start\_date → **election\_id**, end\_date

Election\_State, end\_date → **election\_id**, start\_date

The second and third functional dependencies are constraints that in a state Elections can't start or end on the same date. as seen from the above function dependencies left attributes are super keys and hence transitive dependencies are not maintained. Hence we can conclude a 3NF form.

#### 4.1.4. Administrator Entity

ADMINISTRATOR		
admin_id	INT	password
		user_id

The administrator entity, being a specialisation of the user entry, has one additional entry as a password, with

**Primary Key:** admin\_id

*Foreign Key:* user\_id

(referencing User table, identifies the admin's basic attributes as he is also a voter)

admin\_id → user\_id, password

user\_id → admin\_id, password

Again, from the Similar logic of having the left attributes are super keys transitive depends cannot exist, therefore leading to a 3NF form once again.

#### 4.1.5. Votes\_for relation

VOTES_FOR			
Vote_id INT	election_id	user_id	candidate_id

Moving on to the relations involved, this relation holds a primary key itself to denote votes uniquely, as the primary key.

**Primary Key:** vote\_id

*Foreign Keys:*

(i) election\_id (referencing the Election table, identifies the election the vote was cast in)

(ii) user\_id (referencing the User table, identifies the user who cast the vote)

(iii) candidate\_id (referencing the Candidate table, identifies the candidate the user voted for)

The functional dependencies involved are:

*election\_id, user\_id* → **vote\_id, candidate\_id**

**vote\_id** → *election\_id, user\_id, candidate\_id*

The unique primary key vote\_id is used to register a vote. Additionally, the election ID along with the user ID of the voter can also determine every other attribute as given an election and voter we can get the vote because only one vote can be cast per voter in an election.

Here, since all the left attributes of both function dependencies are super key hence 3NF, that is transitivity is not achieved, rather BCNF is achieved.

#### 4.1.6. Manages Relationship

MANAGES	
election_id	admin_id

Manages being a relationship between administrator and election has no additional attributes or primary keys.

*Foreign Keys:*

*election\_id* (referencing the Election table, identifying which election is being maintained)

*admin\_id* (referencing the User table, identifies the administrator responsible for the election management)

Involved Functional Dependencies are:

*admin\_id* → *election\_id*

*election\_id* → *admin\_id*

Only a single user can become the administrator for an election, and there is a one to one correspondence. 3NF is maintained as both left attributes are superkeys and transitive dependency cannot exist.

#### 4.1.7. Runs\_for Relationship

RUNS _FOR	
election_id	candidate_id

Similar to manages runs\_for relationship is between a candidate and the elections he or she is running for. It also has no additional attributes or primary keys.

*Foreign Keys:*

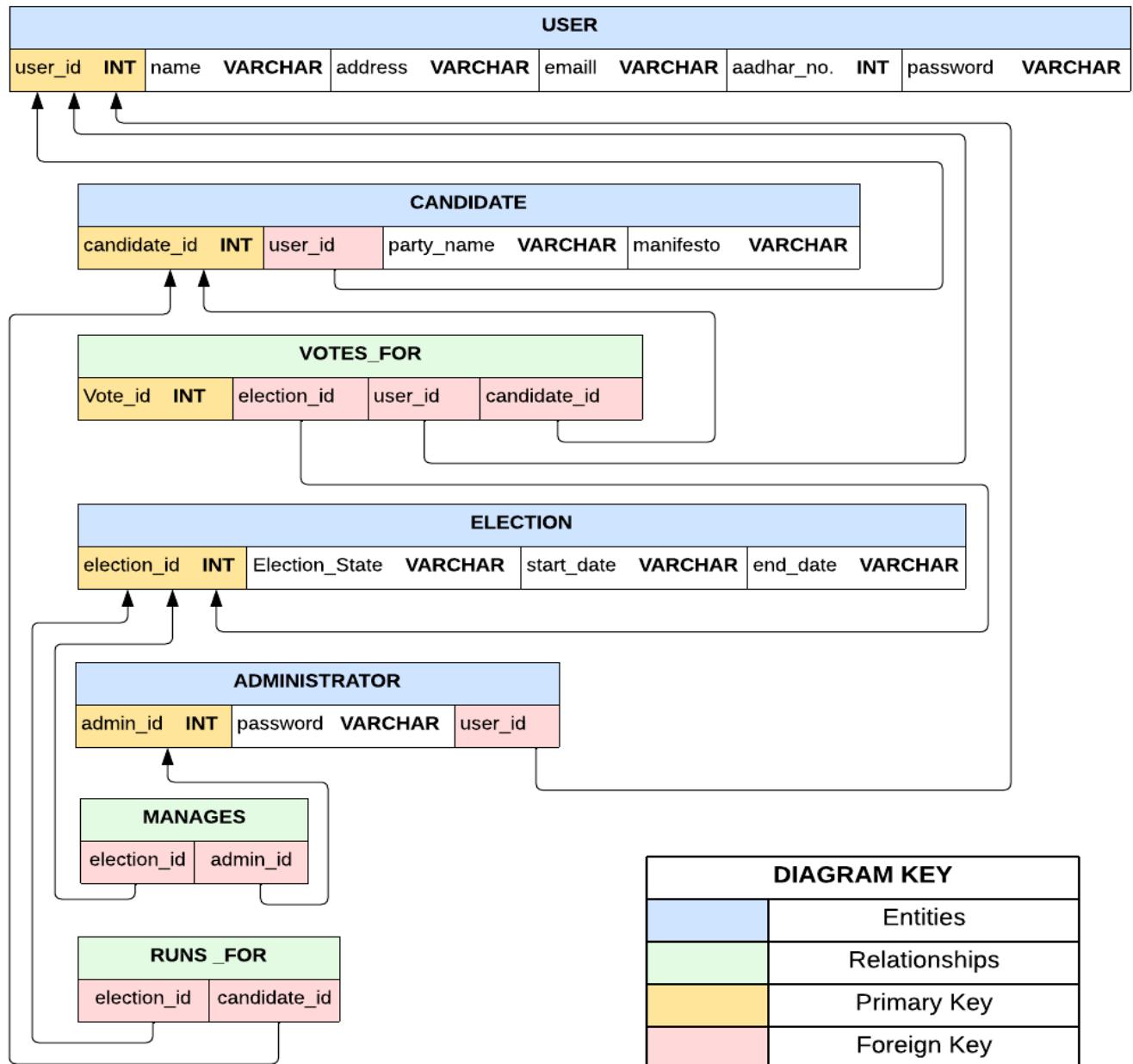
*election\_id* ( referencing the Election table, identifying which election is the candidate running for)

*candidate\_id* (referencing the candidate table, identifying which candidate is running for the election corresponding to the election\_id)

No functional dependency as this is a relation with cardinality N:M. Thus, 3NF is maintained.

Through the above functional dependence observation and normalisation process, we now have the following relational schema, where the foreign key constraints across all relationships and entities are depicted through arrows:

Figure 2: Relational Schema - Voting Management System



## 5. SQL Queries - Output Screenshot

With all the theoretical and diagrammatical concepts in place, the following SQL queries which were provided have been implemented

1. Create all the necessary tables, such as candidates table, users table, votes table, elections table etc.
2. Insert a new user into the user table.
3. Insert a new candidate into the candidate table.
4. Write a query to register a vote.
5. Insert an election into the elections table.
6. Retrieve the number of votes for a candidate.
7. Retrieve the percentage of votes for a candidate.
8. Retrieve the winner of the election.
9. Retrieve the rank of each candidate based on the number of votes received.
10. Retrieve the total number of votes cast for an election.
11. Update the user's information.
12. Delete a specific user's information.

### 5.1. Relational Definition - Create all tables

The database creation begins with the creation of all the tables involved (seven in number), which is executed by the Relational\_Definition Query:

Relational\_Definition x Data\_Values Query\_Runner Query\_Checker

```

15 • CREATE TABLE IF NOT EXISTS voter
16   (
17     user_id INT,
18     name VARCHAR(255) NOT NULL DEFAULT '',
19     address VARCHAR(255) NOT NULL DEFAULT '',
20     email VARCHAR(255) NOT NULL DEFAULT '',
21     password VARCHAR(255) NOT NULL DEFAULT '',
22     aadhar_no INT UNIQUE NOT NULL DEFAULT '0',
23     PRIMARY KEY(user_id)
24   );
25 • describe voter;
26

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result 1 x Result 2 Result 3 Result 4 Result 5 Result 6 Result 7 | Read Only

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	
name	varchar(255)	NO			
address	varchar(255)	NO			
email	varchar(255)	NO			
password	varchar(255)	NO			
aadhar_no	int	NO	UNI	0	

Relational\_Definition x Data\_Values Query\_Runner Query\_Checker

```

29 • CREATE TABLE IF NOT EXISTS candidate
30   (
31     user_id INT,
32     candidate_id INT,
33     party_name VARCHAR(255) NOT NULL DEFAULT '',
34     manifesto VARCHAR(255) NOT NULL DEFAULT '',
35     PRIMARY KEY(candidate_id),
36     CONSTRAINT FK_CV FOREIGN KEY (user_id) REFERENCES voter(user_id) ON DELETE CASCADE ON UPDATE CASCADE
37   );
38 • describe candidate;
39
40

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result 1 x Result 2 Result 3 Result 4 Result 5 Result 6 Result 7 | Read Only

Field	Type	Null	Key	Default	Extra
user_id	int	YES	MUL	NULL	
candidate_id	int	NO	PRI	NULL	
party_name	varchar(255)	NO			
manifesto	varchar(255)	NO			

Relational\_Definition Data\_Values Query\_Runner Query\_Checker

39  
40  
41  
42 • CREATE TABLE IF NOT EXISTS election  
43 (     election\_id INT AUTO\_INCREMENT,  
44     Election\_State VARCHAR(255) NOT NULL DEFAULT ' ',  
45     Start\_Date VARCHAR(255) NOT NULL DEFAULT ' ',  
46     End\_Date    VARCHAR(255) NOT NULL DEFAULT ' ',  
47     PRIMARY KEY(election\_id),  
48     UNIQUE(Election\_State, Start\_Date),  
49     UNIQUE(Election\_State, End\_Date)  
50 );  
51  
52 • ALTER TABLE election AUTO\_INCREMENT=10000001;  
53 • describe election;  
54  
55  
56  
57 • CREATE TABLE IF NOT EXISTS votes for

Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]

	Field	Type	Null	Key	Default	Extra
▶	election_id	int	NO	PRI	NULL	auto_increment
	Election_State	varchar(255)	NO	MUL		
	Start_Date	varchar(255)	NO			
	End_Date	varchar(255)	NO			

Result 1 Result 2 Result 3 X Result 4 Result 5 Result 6 Result 7 ⓘ Read Only

Result Grid  
Form Editor

Relational\_Definition   Data\_Values   Query\_Runner   Query\_Checker

56  
57 • CREATE TABLE IF NOT EXISTS votes\_for  
58 (  
59     vote\_id INT AUTO\_INCREMENT,  
60     election\_id INT,  
61     candidate\_id INT,  
62     user\_id INT,  
63     PRIMARY KEY(vote\_id),  
64     CONSTRAINT FK\_VFU FOREIGN KEY (user\_id) REFERENCES voter(user\_id) ON DELETE CASCADE ON UPDATE CASCADE;  
65     CONSTRAINT FK\_VFE FOREIGN KEY (election\_id) REFERENCES election(election\_id) ON DELETE CASCADE ON UPDATE CASCADE;  
66     CONSTRAINT FK\_VFC FOREIGN KEY (candidate\_id) REFERENCES candidate(candidate\_id) ON DELETE CASCADE ON UPDATE CASCADE;  
67     UNIQUE(election\_id, user\_id)  
68 );  
69 • ALTER TABLE votes\_for AUTO\_INCREMENT=10000001;  
70 • describe votes\_for;  
71  
72  
73  
74 • CREATE TABLE IF NOT EXISTS administrator

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Field	Type	Null	Key	Default	Extra
vote_id	int	NO	PRI	NULL	auto_increment
election_id	int	YES	MUL	NULL	
candidate_id	int	YES	MUL	NULL	
user_id	int	YES	MUL	NULL	

Result 1   Result 2   Result 3   Result 4   Result 5   Result 6   Result 7   Read Only

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

- sakila
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sys
- vms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- world

Administration Schemas

No object selected

Relational\_Definition   Data\_Values   Query\_Runner   Query\_Checker

69 • ALTER TABLE votes\_for AUTO\_INCREMENT=10000001;  
70 • describe votes\_for;  
71  
72  
73  
74 • CREATE TABLE IF NOT EXISTS administrator  
75 (  
76     admin\_id INT,  
77     admin\_password VARCHAR(255) NOT NULL DEFAULT ' ',  
78     user\_id INT,  
79     PRIMARY KEY(admin\_id),  
80     CONSTRAINT FK\_AV FOREIGN KEY(user\_id) REFERENCES voter(user\_id) ON DELETE CASCADE ON UPDATE CASCADE ,  
81     UNIQUE (admin\_id, user\_id)  
82 );  
83 • describe administrator;  
84  
85  
86  
87 • CREATE TABLE IF NOT EXISTS managers

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Field	Type	Null	Key	Default	Extra
admin_id	int	NO	PRI	NULL	
admin_password	varchar(255)	NO			
user_id	int	YES	MUL	NULL	

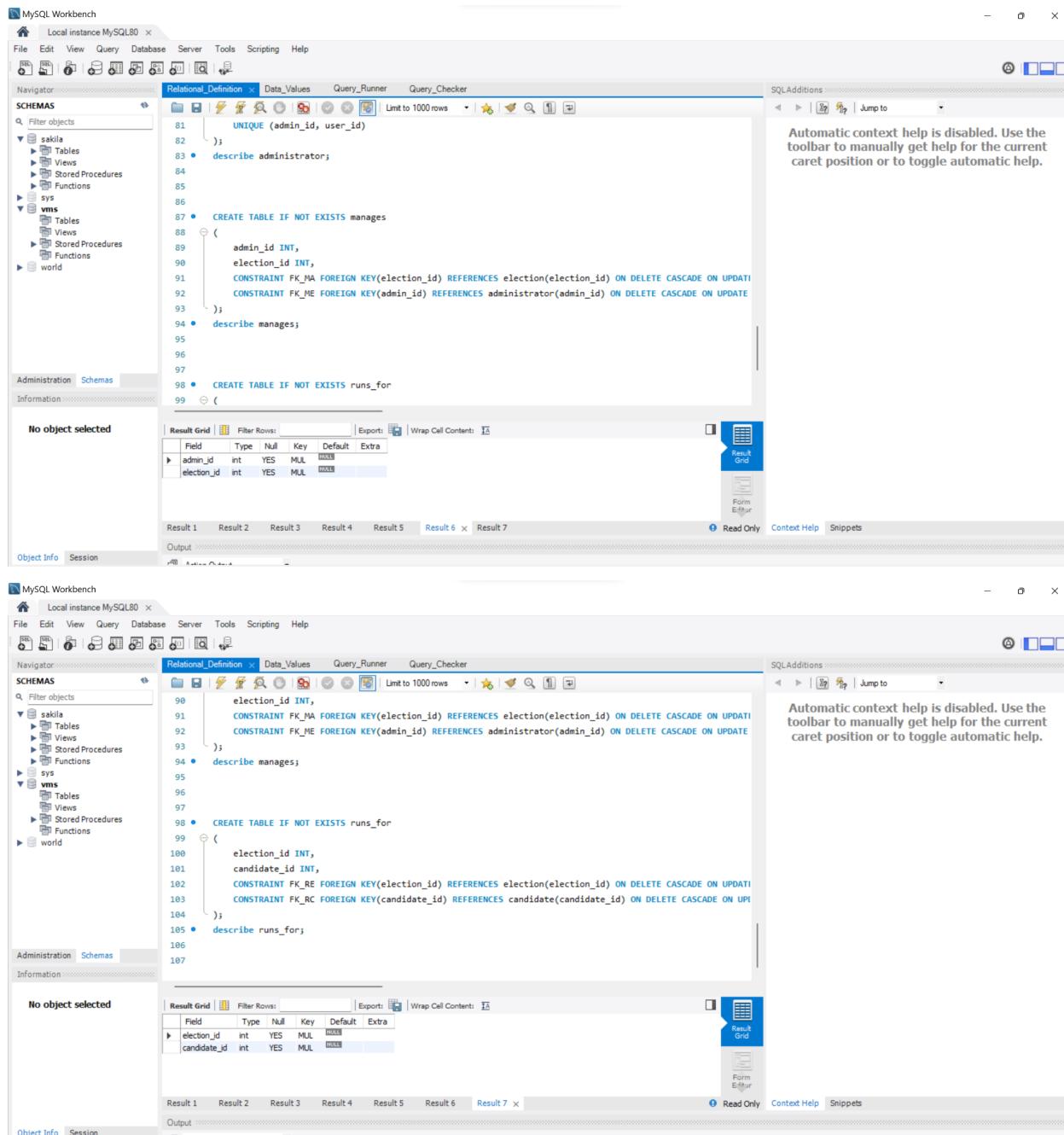
Result 1   Result 2   Result 3   Result 4   Result 5   Result 6   Result 7   Read Only   Context Help   Snippets

Output:

Object Info Session

## VOTING MANAGEMENT SYSTEM

17



## 5.2. Insert User

**MySQL Workbench**

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator Relational\_Definition Data\_Values Query\_Banner x Query\_Checker

Limit to 1000 rows

```

9 -- SELECT * FROM runs_for;
10
11
12
13 /*2. INSERT A NEW USER INTO USERS(VOTER) TABLE */
14 • DROP PROCEDURE IF EXISTS insert_user;
15 DELIMITER $$;
16 • CREATE procedure insert_user(user_entered INT, user_name varchar(255), address varchar(255),
17 email varchar(255), password varchar(255), aadhar_no INT)
18 MODIFIES SQL DATA
19 BEGIN
20 IF (user_entered IN (SELECT user_id from voter)) THEN
21 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT="USER EXISTS!";
22 ELSE
23 INSERT INTO voter (user_id, name, address, email, password, aadhar_no)
24 VALUES (user_entered, user_name, address, email, password, aadhar_no);
25 END IF;
26 END$$
27 DELIMITER ;
28
29
--
```

No object selected

Object Info Session

Output

#	Time	Action	Message	Duration / Fetch
91	18:59:57	INSERT INTO votes_for(vote_id, election_id, candidate_id, user_id) VALUES (10000025, 1000000...)	1 row(s) affected	0.000 sec
92	18:59:57	INSERT INTO votes_for(vote_id, election_id, candidate_id, user_id) VALUES (10000026, 1000000...)	1 row(s) affected	0.000 sec
93	18:59:57	INSERT INTO votes_for(vote_id, election_id, candidate_id, user_id) VALUES (10000027, 1000000...)	1 row(s) affected	0.000 sec
94	18:59:57	INSERT INTO voter(user_id, name, address, email, password, aadhar_no) VALUES ('12345678', 'Vijayu', '1237462 Bandra Road', 'jayu.orci@hotmail.ca', 'A4W3KSDTY90Y', '388671625')	1 row(s) affected	0.000 sec

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator Relational\_Definition Data\_Values Query\_Banner x Query\_Checker

Limit to 1000 rows

```

1 /* SAMPLE QUERIES THAT CAN BE RUN. TO CHECK THE CONSTRAINTS OF UNIQUENESS AND FOREIGN KEYS YOU CAN ADD QUERIES AS PER YOUR REQUIREMENT
2 USE vms;
3
4 • SELECT * FROM voter;
5 • call insert_user('12345678', 'Vijayu', '1237462 Bandra Road', 'jayu.orci@hotmail.ca', 'A4W3KSDTY90Y', '388671625');
6 • SELECT * FROM voter;
7
8
9 • SELECT * FROM candidate;
```

No object selected

Object Info Session

Result Grid

user_id	name	address	email	password	aadhar_no							
10392149	Halla Costa	469-2173 Portbtor Road	fels.orci@hotmail.ca	YQ41TQ06YA	141608543							
10816078	Tyler Dotson	P.O. Box 112, 1521 Malessada Road	in.fauibus@outlook.edu	DOL1KvUJS	427080444							
12345678	Vijayu	1237462 Bandra Road	jayu.orci@hotmail.ca	A4W3KSDTY90Y	388671625							
13946744	Sopoline Pope	Ap #170-1975 Dolor, Street	maceenas.mi@google.edu	VOO29WM1M4H	871452405							
14809620	Savannah Montgomery	P.O. Box 622, 1349 Moleste Av.	loboris.class@protonmail.ca	WBF40CE0P	225903750							
19993116	Lena Strickland	2122 Leo Ave	blandit@hotmail.com	VEP83VD3LG	110976137							
24026930	Josiah Chang	Ap #391-2625 Nunc Ave	leo@cloud.net	XUF6SWV74Q0M	898638575							
24555631	Sophia Parks	P.O. Box 642, 4759 Eu Ave	quis.arcu@cloud.org	I0H40CZQ2X0	134082328							
30403329	Alana Mann	841-5195 In Street	mauris.integer.sem@outlook.ca	QK1W70XK6B0	914997836							
32920330	Jin Can	Ap #805-569 Felis, Rd.	eu.eros@cloud.net	DQQ3QHQJPD	329802400							
34120653	Knox Durham	Ab #717-1928 Indutn Rd.	tellus@boudle.com	DY168DXKJU	546001195							
voter_1	voter_2	x candidate_3	candidate_4	votes_for_5	election_7	election_8	Result_9	Result_10	Result_11	F	Apply	Revert

Output

#	Time	Action	Message	Duration / Fetch
119	19:01:12	SELECT * FROM voter LIMIT 0, 1000	32 row(s) returned	0.000 sec / 0.000 sec
120	19:01:12	call delete_user_info(14809620)	1 row(s) affected	0.000 sec
121	19:01:12	SELECT * FROM voter LIMIT 0, 1000	31 row(s) returned	0.000 sec / 0.000 sec

### 5.3. Insert Candidate

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator: Relational\_Definition Data\_Values Query\_Runner Query\_Checker

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```

31 • /* 3. INSERT A NEW CANDIDATE INTO CANDIDATE TABLE AFTER INSERTING INTO USER(VOTER) TABLE DUE TO SPECIALIZATION*/
32 DROP PROCEDURE IF EXISTS insert_candidate;
33 DELIMITER $$;
34 • CREATE procedure insert_candidate(election_entered INT, candidate_entered INT, user_entered INT,
35 user_name varchar(255), address varchar(255), email varchar(255), password varchar(255),
36 aadhar_no INT, party VARCHAR(255), manifesto VARCHAR(255))
37 MODIFIES SQL DATA
38 BEGIN
39     IF (user_entered IN (SELECT user_id from voter)) THEN
40         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT="USER EXISTS!";
41     ELSE
42         INSERT INTO voter (user_id, name, address, email, password, aadhar_no)
43             VALUES(user_entered, user_name, address, email, password, aadhar_no);
44     END IF;
45     IF (candidate_entered IN (SELECT candidate_id from candidate)) THEN
46         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT="CANDIDATE EXISTS!";
47     ELSE
48         INSERT INTO candidate (user_id, candidate_id, party_name, manifesto )
49             VALUES (user_entered, candidate_entered, party, manifesto);
50     END IF;
51     IF (election_entered IN (SELECT election_id from election)) THEN
52         INSERT INTO runs_for (candidate_id, election_id)
53             VALUES (candidate_entered, election_entered);
54     ELSE
55         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT="ELECTION DOES NOT EXISTS!";
56     END IF;
57 END$$

```

No object selected

Object Info Session Output

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator: Relational\_Definition Data\_Values Query\_Runner Query\_Checker

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```

7
8
9 • SELECT * FROM candidate;
10 • call insert_candidate(10000002, 31809980, 75431171, 'koko', 'ghar', 'sjdfgsjhyudfg', 'AJSHFGSF', 1298946788, 'CONGRESS', 'NO');
11 • SELECT * FROM candidate;
12
13
14 • SELECT * FROM votes_for;

```

No object selected

Object Info Session Output

Action Output

#	Time	Action	Message	Duration / Fetch
102	19:01:12	call insert_user('12345678', 'Vijayu', '1237462 Bandra Road', 'jayu.ori@hotmail.ca', 'AAHJKSDTY90Y...', '1 row(s) affected')	1 row(s) affected	0.000 sec
103	19:01:12	SELECT * FROM voter LIMIT 0, 1000	31 row(s) returned	0.000 sec / 0.000 sec
104	19:01:12	SELECT * FROM candidate LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
105	19:01:12	call insert_candidate(10000002, 31809980, 75431171, 'koko', 'ghar', 'sjdfgsjhyudfg', 'AJSHFGSF', 1298946788, 'CONGRESS', 'NO')	1 row(s) affected	0.000 sec
106	19:01:12	SELECT * FROM candidate LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
107	19:01:12	SELECT * FROM votes_for LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
108	19:01:12	call insert_votes_for('10000002', '31809980', '75431171', '1298946788', 'CONGRESS', 'NO')	1 row(s) affected	0.000 sec

## 5.4. Register a vote - register\_vote

**MySQL Workbench - Local instance MySQL80**

**Relational\_Definition**

```

61
62 • /* 4. REGISTER A VOTE */
63 DROP PROCEDURE IF EXISTS register_vote;
64 DELIMITER $$;
65 • CREATE PROCEDURE register_vote(elec_state varchar(255), elec_date varchar(255), cand_id int, user_id int)
66 MODIFIES SQL DATA
67 BEGIN
68     DECLARE elec_id int;
69     IF (user_id IN (SELECT user_id FROM voter)) THEN
70         SELECT election_id FROM election WHERE elec_state = Election_State AND elec_date = Start_Date INTO elec_id;
71         INSERT INTO votes_for(election_id, candidate_id, user_id) VALUES (elec_id, cand_id, user_id);
72     ELSE
73         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "USER DOES NOT EXIST!";
74     END IF;
75 END$$
76
77
78

```

**Output**

Action	Time	Action	Message	Duration / Fetch
CREATE procedure insert_user	19.05.18	user_entered INT, user_name varchar(255), address varchar(255), em...	0 row(s) affected	0.000 sec
/* 3. INSERT A NEW CANDIDATE INTO CANDIDATE TABLE AFTER INSERTING INTO USER(VOTER)	19.05.18		0 row(s) affected	0.000 sec
CREATE procedure insert_candidate	19.05.18	election_entered INT, candidate_entered INT, user_entered INT...	0 row(s) affected	0.016 sec
/* 4 REGISTER A VOTE */ DROP PROCEDURE IF EXISTS register_vote	19.05.18		0 row(s) affected	0.000 sec
CREATE PROCEDURE register_vote(elec_state varchar(255), elec_date varchar(255), cand_id int, user_id int)	19.05.18		0 row(s) affected	0.000 sec
/* 5. INSERT AN ELECTION INTO THE ELECTION TABLE UNDER THE ASSUMPTION THAT A C...	19.05.18		0 row(s) affected	0.000 sec
CREATE PROCEDURE insert_election	19.05.18	election_id INT, election_name varchar(255), election_start...	0 row(s) affected	0.000 sec

**MySQL Workbench - Local instance MySQL80**

**Relational\_Definition**

```

10 • call insert_candidate(10000002, 31809980, 75431171, 'koko', 'ghar', 'sjdfgsjhyudfg', 'AJSHFGSF', 1298946788, 'CONGRESS', 'NO');
11 • SELECT * FROM candidate;
12
13
14 • SELECT * FROM votes_for;
15 • call register_vote('Punjab', '29-01-2013', 59559032, 97651571);
16 • SELECT * FROM votes_for;
17

```

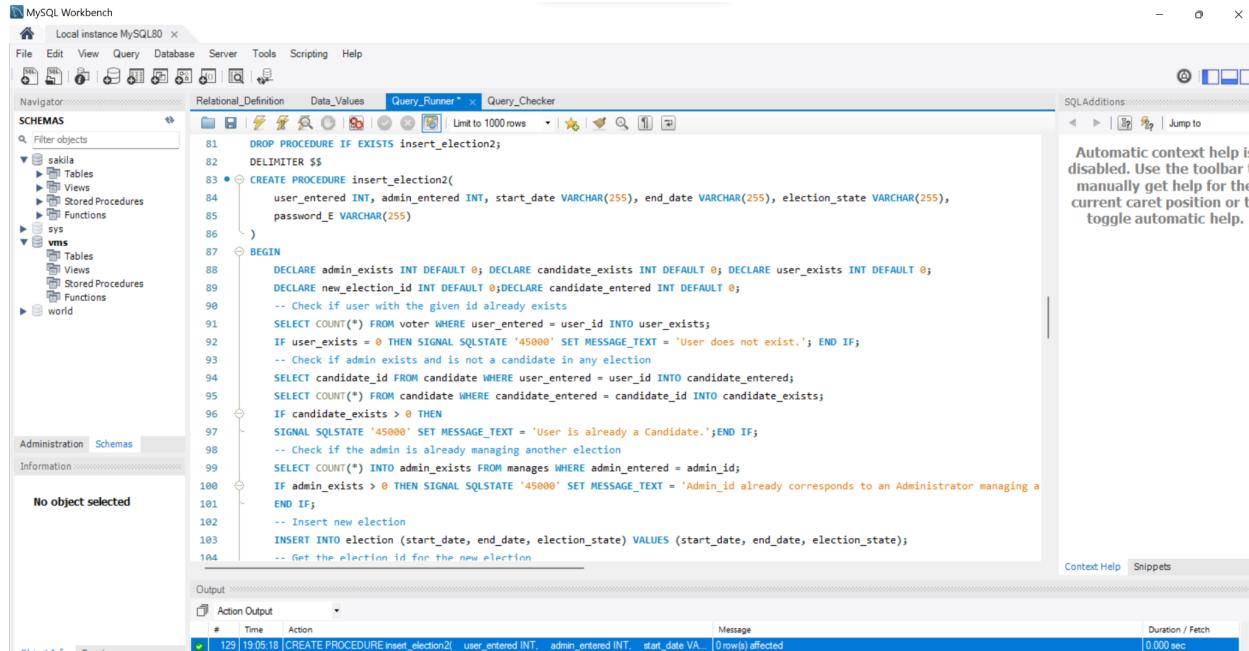
**Result Grid**

vote_id	election_id	candidate_id	user_id
10000025	10000004	89436412	83739294
10000026	10000005	89436412	86379979
10000027	10000001	463216412	88018769
10000028	10000004	78613292	93614661
10000029	10000002	89436412	95474693
10000030	10000006	89436412	97651571
10000031	10000005	59559032	97651571
10000032	10000005	59559032	97651571

**Output**

Action	Time	Action	Message	Duration / Fetch
call insert_candidate	19.01.12	(10000002, 31809980, 75431171, 'koko', 'ghar', 'sjdfgsjhyudfg', 'AJSHFGSF', 129...	1 row(s) affected	0.000 sec
SELECT * FROM candidate	19.01.12	LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
SELECT * FROM votes_for	19.01.12	LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec
call register_vote	19.01.12	('Punjab', '29-01-2013', 59559032, 97651571)	1 row(s) affected	0.000 sec
SELECT * FROM votes_for	19.01.12	LIMIT 0, 1000	31 row(s) returned	0.000 sec / 0.000 sec
SELECT * FROM election	19.01.12	LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
call insert_election	19.01.12	(10000002, 'CONGRESS', 'NO')	1 row(s) affected	0.000 sec

## 5.5. Insert election - insert\_election



The screenshot shows the MySQL Workbench interface with the 'Query\_Runner' tab selected. The code area contains a stored procedure definition:

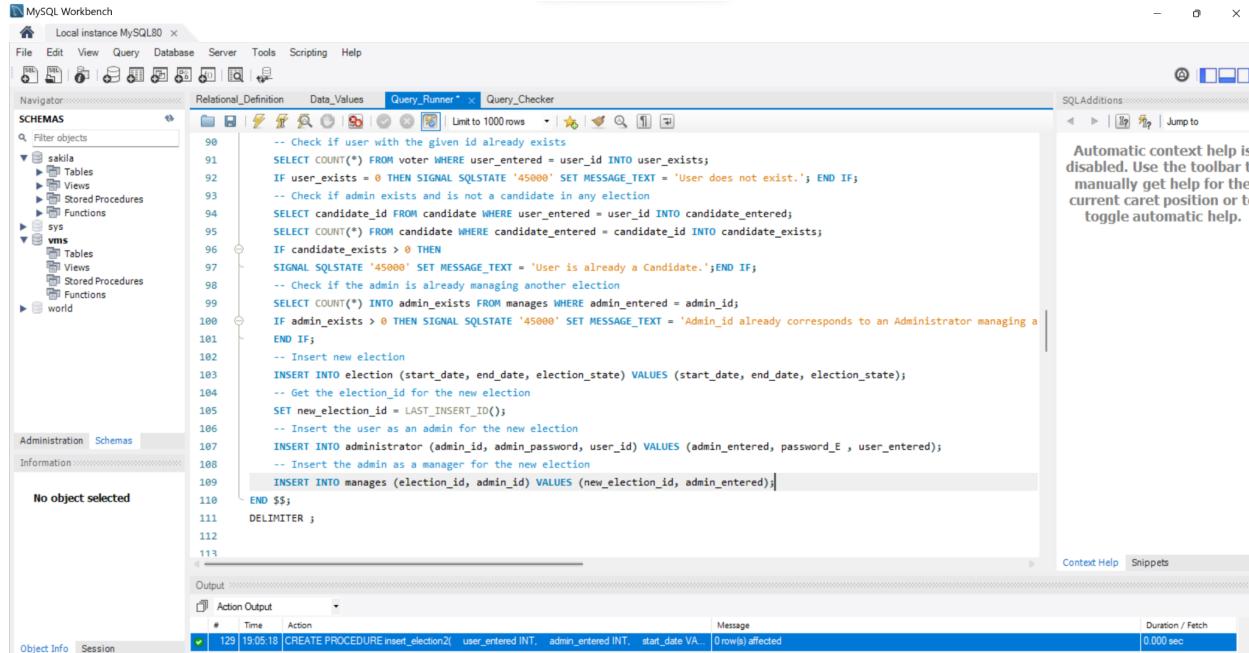
```

81  DROP PROCEDURE IF EXISTS insert_election2;
82  DELIMITER $$;
83  CREATE PROCEDURE insert_election2(
84      user_entered INT, admin_entered INT, start_date VARCHAR(255), end_date VARCHAR(255), election_state VARCHAR(255),
85      password_E VARCHAR(255)
86  )
87  BEGIN
88      DECLARE admin_exists INT DEFAULT 0; DECLARE candidate_exists INT DEFAULT 0; DECLARE user_exists INT DEFAULT 0;
89      DECLARE new_election_id INT DEFAULT 0;DECLARE candidate_entered INT DEFAULT 0;
90      -- Check if user with the given id already exists
91      SELECT COUNT(*) FROM voter WHERE user_entered = user_id INTO user_exists;
92      IF user_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User does not exist.'; END IF;
93      -- Check if admin exists and is not a candidate in any election
94      SELECT candidate_id FROM candidate WHERE user_entered = user_id INTO candidate_entered;
95      SELECT COUNT(*) FROM candidate WHERE candidate_entered = candidate_id INTO candidate_exists;
96      IF candidate_exists > 0 THEN
97          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User is already a Candidate.';END IF;
98      -- Check if the admin is already managing another election
99      SELECT COUNT(*) INTO admin_exists FROM manages WHERE admin_entered = admin_id;
100     IF admin_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Admin_id already corresponds to an Administrator managing a';
101     END IF;
102     -- Insert new election
103     INSERT INTO election (start_date, end_date, election_state) VALUES (start_date, end_date, election_state);
104     -- Get the election_id for the new election
105     SET new_election_id = LAST_INSERT_ID();
106     -- Insert the user as an admin for the new election
107     INSERT INTO administrator (admin_id, admin_password, user_id) VALUES (admin_entered, password_E , user_entered);
108     -- Insert the admin as a manager for the new election
109     INSERT INTO manages (election_id, admin_id) VALUES (new_election_id, admin_entered);
110  END $$;
111  DELIMITER ;
112
113

```

The output pane shows the execution of the command:

#	Time	Action	Message	Duration / Fetch
129	19:05:18	CREATE PROCEDURE insert_election2( user_entered INT, admin_entered INT, start_date VA... )	0 rows affected	0.000 sec

The second screenshot shows the same MySQL Workbench interface with a different implementation of the stored procedure:

```

90      -- Check if user with the given id already exists
91      SELECT COUNT(*) FROM voter WHERE user_entered = user_id INTO user_exists;
92      IF user_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User does not exist.'; END IF;
93      -- Check if admin exists and is not a candidate in any election
94      SELECT candidate_id FROM candidate WHERE user_entered = user_id INTO candidate_entered;
95      SELECT COUNT(*) FROM candidate WHERE candidate_entered = candidate_id INTO candidate_exists;
96      IF candidate_exists > 0 THEN
97          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User is already a Candidate.';END IF;
98      -- Check if the admin is already managing another election
99      SELECT COUNT(*) INTO admin_exists FROM manages WHERE admin_entered = admin_id;
100     IF admin_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Admin_id already corresponds to an Administrator managing a';
101     END IF;
102     -- Insert new election
103     INSERT INTO election (start_date, end_date, election_state) VALUES (start_date, end_date, election_state);
104     -- Get the election_id for the new election
105     SET new_election_id = LAST_INSERT_ID();
106     -- Insert the user as an admin for the new election
107     INSERT INTO administrator (admin_id, admin_password, user_id) VALUES (admin_entered, password_E , user_entered);
108     -- Insert the admin as a manager for the new election
109     INSERT INTO manages (election_id, admin_id) VALUES (new_election_id, admin_entered);
110  END $$;
111  DELIMITER ;
112
113

```

The output pane shows the execution of the command:

#	Time	Action	Message	Duration / Fetch
129	19:05:18	CREATE PROCEDURE insert_election2( user_entered INT, admin_entered INT, start_date VA... )	0 rows affected	0.000 sec

```

16 •   SELECT * FROM votes_for;
17
18
19 •   SELECT * FROM election;
20 •   call insert_election2('61906143','782368540','17-03-2023','24-04-2023','Telangana','isujsgsrdivjb');
21 •   SELECT * FROM election;
22
23

```

election_id	Election_State	Start_Date	End_Date
10000001	Meghalaya	15-02-2022	25-02-2022
10000002	Gujarat	31-01-2013	07-02-2013
10000003	Chandigarh	15-02-2022	24-02-2022
10000004	Uttar Pradesh	29-06-2012	06-07-2012
10000005	Punjab	29-01-2013	07-02-2013
10000006	Uttar Pradesh	21-10-2019	29-10-2019
10000007	Telangana	17-03-2023	24-04-2023
NULL	NULL	NULL	NULL

No object selected

voter 1 voter 2 candidate 3 candidate 4 votes\_for5 votes\_for6 election 7 election 8 × Result 9 Result 10 Result 11 F Apply Revert Context Help Snippets

Output:

Action Output	Time	Action	Message	Duration / Fetch
109	19:01:12	SELECT * FROM votes_for LIMIT 0, 1000	31 row(s) returned	0.000 sec / 0.000 sec
110	19:01:12	SELECT * FROM election LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
111	19:01:12	call insert_election2('61906143','782368540','17-03-2023','24-04-2023','Telangana','isujsgsrdivjb')	1 row(s) affected	0.016 sec
112	19:01:12	SELECT * FROM election LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
113	19:01:12	call get_candidate_vote_count('89436412', '10000002')	1 row(s) returned	0.000 sec / 0.000 sec

## 5.6. Retrieve no. of votes for Candidate (in given election)

```

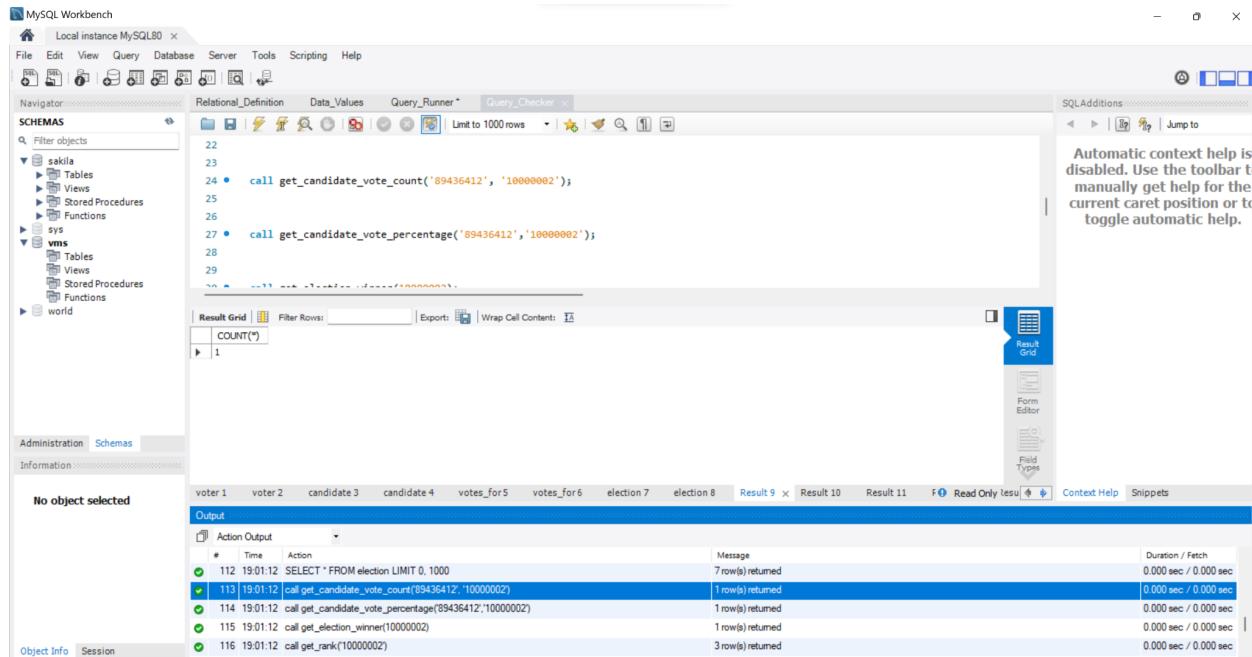
130
131     END $$;
132     DELIMITER ;
133
134
135
136
137 •   /*6. GETS VOTE COUNT IN PARTICULAR ELECTION FOR GIVEN CANDIDATE*/
138     DROP PROCEDURE IF EXISTS get_candidate_vote_count;
139     DELIMITER $$
140 •   CREATE PROCEDURE get_candidate_vote_count(candidate_entered INT, election_entered INT)
141     BEGIN
142         SELECT COUNT(*) FROM votes_for
143         WHERE (candidate_entered = candidate_id AND election_entered = election_id);
144     END $$
145     DELIMITER ;
146
147
148
149
150 •   /*7. GETS PERCENTAGE OF VOTES FOR GIVEN CANDIDATE IN GIVEN ELECTION */

```

No object selected

Output:

Action Output	Time	Action	Message	Duration / Fetch
130	19:05:18	/6. GETS VOTE COUNT IN PARTICULAR ELECTION FOR GIVEN CANDIDATE/ DROP PROCEDURE IF EXISTS get_candidate_vote_count;	0 row(s) affected	0.000 sec
131	19:05:18	CREATE PROCEDURE get_candidate_vote_count(candidate_entered INT, election_entered INT) BEGIN SELECT COUNT(*) FROM votes_for WHERE (candidate_entered = candidate_id AND election_entered = election_id); END \$\$	0 row(s) affected	0.000 sec
132	19:05:18	/7. GETS PERCENTAGE OF VOTES FOR GIVEN CANDIDATE IN GIVEN ELECTION / DROP PROCEDURE IF EXISTS get_candidate_vote_count;	0 row(s) affected	0.000 sec
133	19:05:18	CREATE PROCEDURE get_candidate_vote_percentage(candidate_entered INT, election_entered INT) BEGIN SELECT COUNT(*) AS total_votes, SUM(votes_for) AS candidate_votes FROM votes_for WHERE (candidate_entered = candidate_id AND election_entered = election_id); END \$\$	0 row(s) affected	0.000 sec
134	19:05:18	/8. GIVES WINNER OF AN ELECTION/ DROP PROCEDURE IF EXISTS get_election_winner;	0 row(s) affected	0.000 sec



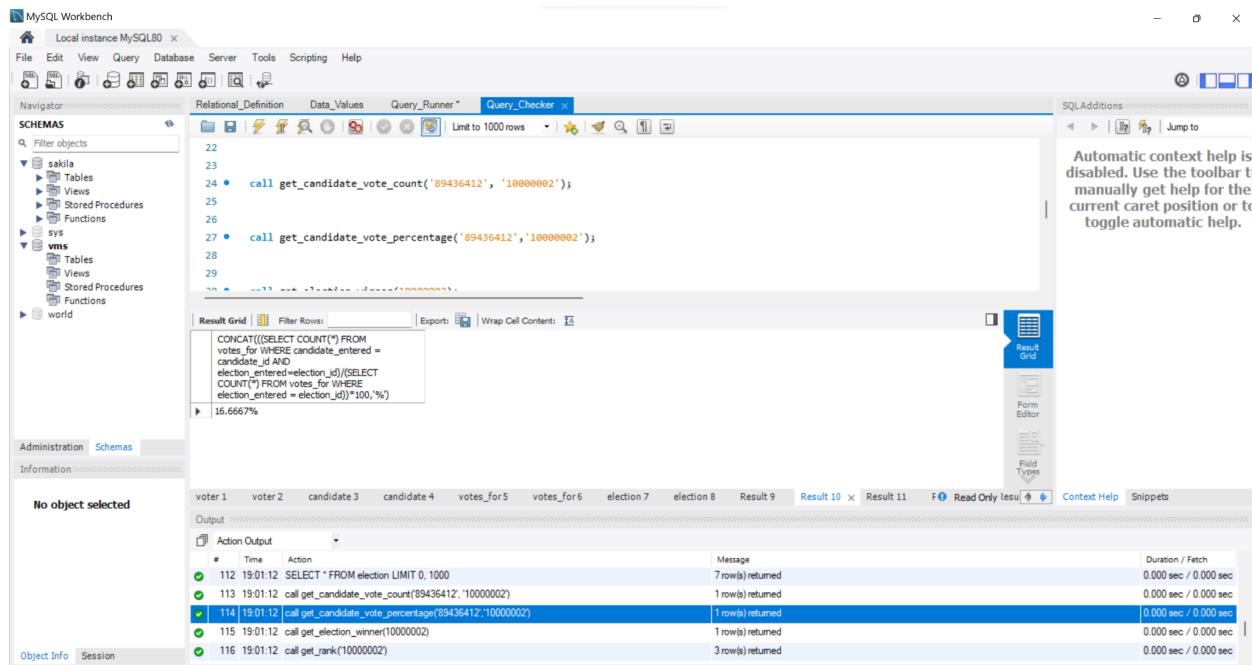
The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL80, Schemas: sakila, sys, vms, world.
- Query Editor:** Contains the following SQL code:
 

```

22
23
24 • call get_candidate_vote_count('89436412', '10000002');
25
26
27 • call get_candidate_vote_percentage('89436412','10000002');
28
29
      
```
- Result Grid:** Shows the output of the last query, which is a single row with COUNT(\*) = 1.
- Action Output:** Shows the history of actions taken during the session, including the execution of the stored procedures.

#### 4.7. Retrieve the percentage of votes for a candidate.



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL80, Schemas: sakila, sys, vms, world.
- Query Editor:** Contains the following SQL code:
 

```

22
23
24 • call get_candidate_vote_count('89436412', '10000002');
25
26
27 • call get_candidate_vote_percentage('89436412','10000002');
28
29
      
```
- Result Grid:** Shows the output of the last query, which is a single row with the value 16.667%.
- Action Output:** Shows the history of actions taken during the session, including the execution of the stored procedures.

### **5.8. Retrieve the winner of the election.**

**5.9. Retrieve the rank of each candidate based on the number of votes received.**

The screenshot shows the MySQL Workbench interface with the following details:

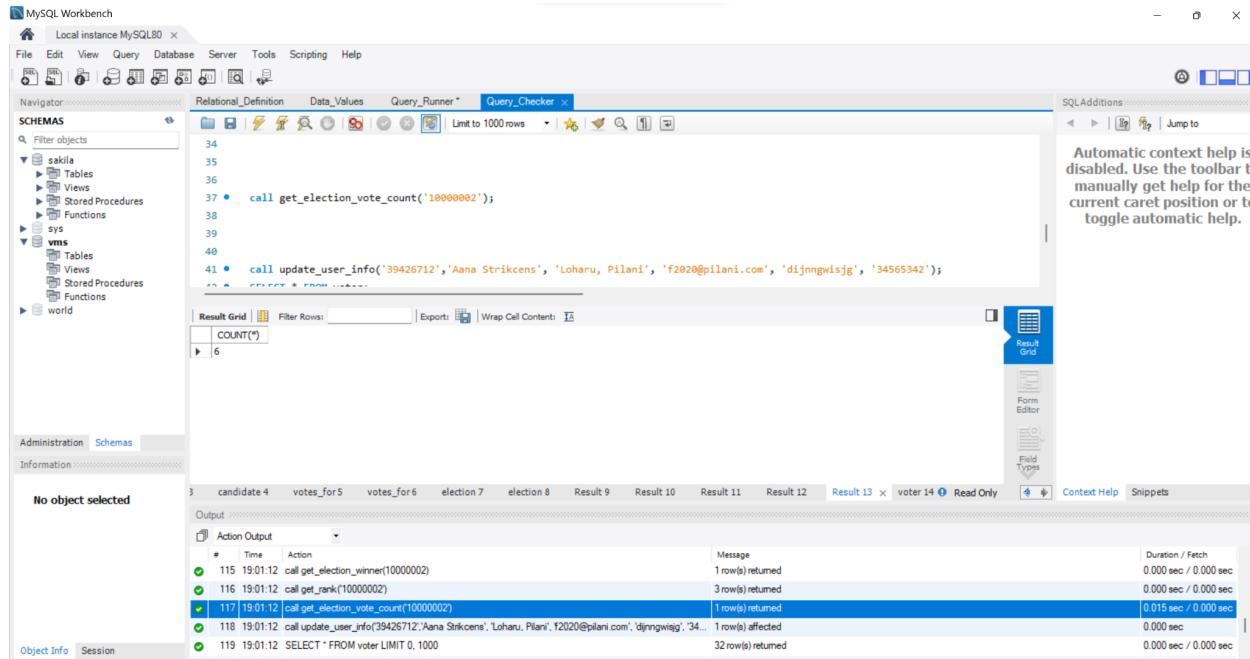
- File**, **Edit**, **View**, **Query**, **Database**, **Server**, **Tools**, **Scripting**, **Help** menu.
- Navigator** pane on the left showing **SCHEMAS**: **sakila** (Tables, Views, Stored Procedures, Functions), **sys** (Functions), **vms** (Tables, Views, Stored Procedures, Functions), and **world**.
- Relational Definition**, **Data Values**, **Query Runner**, **Query Checker** tabs at the top.
- SQLAdditions** toolbar on the right with icons for back, forward, search, and jump to.
- Automatic context help** message: "Automatic context help is disabled. Use the toolbar or manually get help for the current caret position or toggle automatic help."
- Result Grid** pane showing the output of the following query:

```
28
29
30 • call get_election_winner(10000002);
31
32
33 • call get_rank('10000002');
34
35
36
```

candidate_id	vote_count	ranking
46321648	3	1
59559032	2	2
89436412	1	3
- Administration** and **Schemas** tabs in the bottom-left.
- Information** tab in the bottom-left.
- No object selected** message in the bottom-left.
- Object Info** and **Session** tabs in the bottom-left.
- Action Output** pane at the bottom showing log entries:

#	Time	Action	Message	Duration / Fetch
112	19:01:12	SELECT * FROM election LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
113	19:01:12	call get_candidate_vote_count(89436412, '10000002')	1 row(s) returned	0.000 sec / 0.000 sec
114	19:01:12	call get_candidate_vote_percentage(89436412, '10000002')	1 row(s) returned	0.000 sec / 0.000 sec
115	19:01:12	call get_election_winner(10000002)	1 row(s) returned	0.000 sec / 0.000 sec
116	19:01:12	call get_rank('10000002')	3 row(s) returned	0.000 sec / 0.000 sec

## 5.10. Retrieve the total number of votes cast for an election.

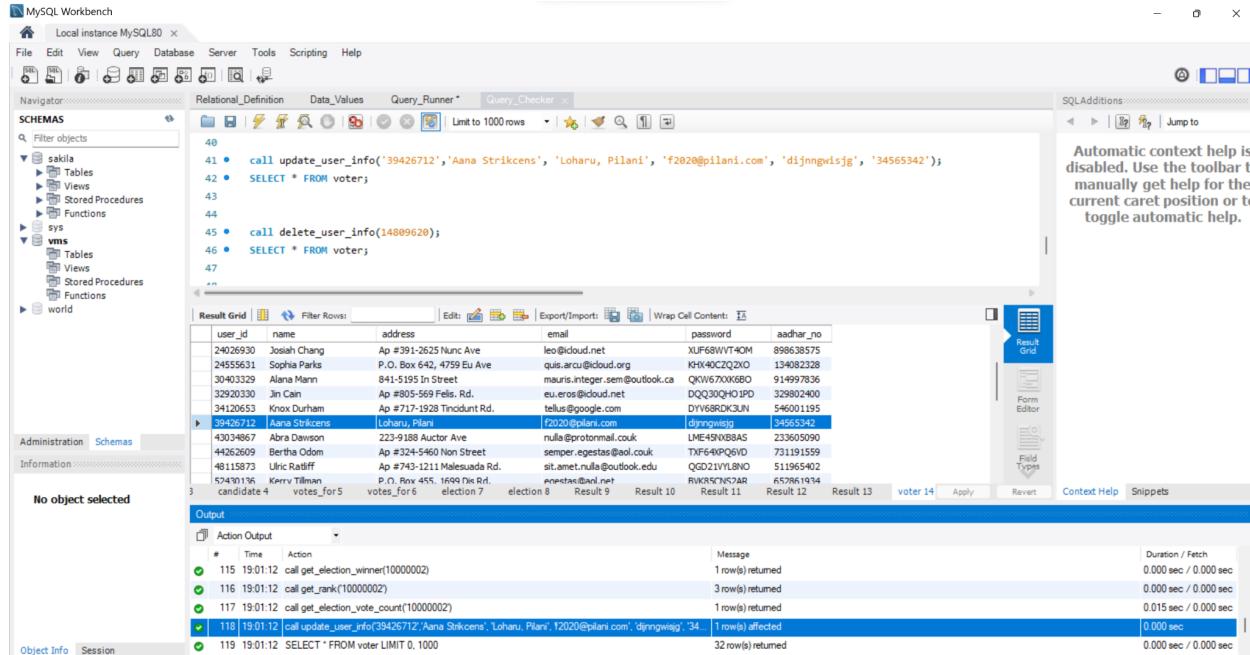


The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `vms` containing tables like `candidate`, `voter`, and `election`.
- Query Editor:** Contains the following SQL code:
 

```
34
35
36
37 •  call get_election_vote_count('10000002');
38
39
40
41 •  call update_user_info('39426712','Aana Strikcens', 'Loharu, Pilani', 'f2020@pilani.com', 'dijnngwisjg', '34565342');
-- •  returns a row ...
```
- Result Grid:** Displays the result of the `get\_election\_vote\_count` call, showing a single row with COUNT(\*) = 6.
- Action Output:** Shows the log of actions taken during the session, including the execution of the stored procedure and the update query.

## 5.11. Update the user's information.



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `vms` containing tables like `candidate`, `voter`, and `election`.
- Query Editor:** Contains the following SQL code:
 

```
40
41 •  call update_user_info('39426712','Aana Strikcens', 'Loharu, Pilani', 'f2020@pilani.com', 'dijnngwisjg', '34565342');
42 •  SELECT * FROM voter;
43
44
45 •  call delete_user_info(14809620);
46 •  SELECT * FROM voter;
47
```
- Result Grid:** Displays the result of the `update\_user\_info` call, showing the updated voter record for voter ID 39426712.
- Action Output:** Shows the log of actions taken during the session, including the execution of the update query, the selection of all voters, and the deletion of a voter record.

## 5.12. Delete a specific user's information.

### Before Delete

The screenshot shows the MySQL Workbench interface with the 'voter' table selected in the 'Result Grid'. The table contains 15 rows of data, including the entry for Savanna Montgomery. The columns are: user\_id, name, address, email, password, and aadhar\_no. The 'Output' tab at the bottom shows the history of SQL statements executed, including the insertion of Savanna's data and the subsequent delete operation.

user_id	name	address	email	password	aadhar_no
10392149	Halla Acosta	469-2173 Portitor Road	fels.ord@hotmail.ca	YQX41TQO6VA	141608543
10816078	Tyler Dotson	P.O. Box 112, 1521 Malesuada Road	in.fauibus@outlook.edu	DOL14KVU9PS	427080444
12345678	Vijayu	1237462 Bandra Road	jayu.or@hotmail.ca	AAH3KSDTY90Y	388671625
13946744	Sopoline Pope	Ap #170-1975 Dolor, Street	maeemas.mi@google.edu	VOO29WMWIMH	871452405
14809620	Savannah Montgomery	P.O. Box 632, 1349 Maleste Av.	labor.tis.class@protonmail.ca	WRF4DCEB0P	235903750
19993116	Lana Strickland	2122 Leo Ave	blanit@hotmail.com	VEP93ID5L6	110976137
24026930	Josiah Chang	Ap #391-2625 Nunc Ave	leo@icloud.net	XUF68WV740M	898638575
24556311	Sophia Parks	P.O. Box 642, 4759 Eu Ave	quis.aru@cloud.org	KHX40CZQ2X0	134082328
30403329	Alana Mann	841-5195 In Street	maurus.integer.sem@outlook.ca	QK1W670XK6B0	914997836
32926763	Jin Can	Ap #805-569 Felis, Rd.	eu.eros@icloud.net	DDQ30QHQ1PD	329802400
34179633	Kenny DuBois	An #717-1075 Tincidunt Rd.	tel.eBrookle.nnn	PMYK8RNHC1P9	546001165
candidate 4	votes_for 5	votes_for 6	election 7	election 8	Result 9
					Result 10
					Result 11
					Result 12
					Result 13
					voter 14

Action Output:

```

# Time Action Message Duration / Fetch
118 19.01.12 call update_user_info('39426712','Aana Strikcens', 'Loharu, Pilani', 'f2020@pilani.com', 'dijnngwjsjg', '34565342'); 1 row(s) affected 0.000 sec
119 19.01.12 SELECT * FROM voter LIMIT 0, 1000 32 row(s) returned 0.000 sec / 0.000 sec
120 19.01.12 call delete_user_info(14809620); 1 row(s) affected 0.000 sec
121 19.01.12 SELECT * FROM voter LIMIT 0, 1000 31 row(s) returned 0.000 sec / 0.000 sec
122 19.05.18 DROP PROCEDURE IF EXISTS insert_user; 0 row(s) affected 0.000 sec

```

### After Delete

The screenshot shows the MySQL Workbench interface with the 'voter' table selected in the 'Result Grid'. The table now contains 14 rows of data, with Savanna Montgomery's entry removed. The 'Output' tab at the bottom shows the history of SQL statements executed, including the insertion of Savanna's data and the subsequent delete operation.

user_id	name	address	email	password	aadhar_no
10392149	Halla Acosta	469-2173 Portitor Road	fels.ord@hotmail.ca	YQX41TQO6VA	141608543
10816078	Tyler Dotson	P.O. Box 112, 1521 Malesuada Road	in.fauibus@outlook.edu	DOL14KVU9PS	427080444
12345678	Vijayu	1237462 Bandra Road	jayu.or@hotmail.ca	AAH3KSDTY90Y	388671625
13946744	Sopoline Pope	Ap #170-1975 Dolor, Street	maeemas.mi@google.edu	VOO29WMWIMH	871452405
19993116	Lana Strickland	2122 Leo Ave	blanit@hotmail.com	VEP93ID5L6	110976137
24026930	Josiah Chang	Ap #391-2625 Nunc Ave	leo@icloud.net	XUF68WV740M	898638575
24556311	Sophia Parks	P.O. Box 642, 4759 Eu Ave	quis.aru@cloud.org	KHX40CZQ2X0	134082328
30403329	Alana Mann	841-5195 In Street	maurus.integer.sem@outlook.ca	QK1W670XK6B0	914997836
32926763	Jin Can	Ap #805-569 Felis, Rd.	eu.eros@icloud.net	DDQ30QHQ1PD	329802400
34179633	Kenny DuBois	An #717-1075 Tincidunt Rd.	tel.eBrookle.nnn	PMYK8RNHC1P9	546001165
candidate 4	votes_for 5	votes_for 6	election 7	election 8	Result 9
					Result 10
					Result 11
					Result 12
					Result 13
					voter 14

Action Output:

```

# Time Action Message Duration / Fetch
118 19.01.12 call update_user_info('39426712','Aana Strikcens', 'Loharu, Pilani', 'f2020@pilani.com', 'dijnngwjsjg', '34565342'); 1 row(s) affected 0.000 sec
119 19.01.12 SELECT * FROM voter LIMIT 0, 1000 32 row(s) returned 0.000 sec / 0.000 sec
120 19.01.12 call delete_user_info(14809620); 1 row(s) affected 0.000 sec
121 19.01.12 SELECT * FROM voter LIMIT 0, 1000 31 row(s) returned 0.000 sec / 0.000 sec
122 19.05.18 DROP PROCEDURE IF EXISTS insert_user; 0 row(s) affected 0.000 sec

```

(Savanna Montgomery's entry is missing from the table, as it is deleted)

## Links for Video

CHAITANYA SETHI - 2020B3A71961P

<https://drive.google.com/file/d/1EKmC-feXqt7ngMpqBk3CrVRMz8PQY0KP/view?usp=sharing>

**\*\*Note:** In the end, the candidate\_vote\_percentage is different as one candidate was earlier deleted by the delete user query when I initially had run the code once. Hence, only 5 votes remained, out of which the winner had 3, leading to a 60% percentage. The repeating values issue has been corrected in the uploaded code, please check.

STAVYA PURI - 2020B5A70912P

[https://drive.google.com/file/d/1dRX-kY-PgfdpJwKYeVPauehYHpN6Vy\\_W/view?usp=share\\_link](https://drive.google.com/file/d/1dRX-kY-PgfdpJwKYeVPauehYHpN6Vy_W/view?usp=share_link)