ABSTRACT

The entitled project "**EVENT MANAGEMENT SYSTEM**" is made keeping in mind all the aspects of the Events. The system allows only registered users to login and new users are allowed to register on the application. The project provides most of the basic functionality required for an event. It allows the user to select from a list of event types. It will be capable of doing all the necessary operations/functions that are done in any Event for example- registration of events, ticketing, login-logout, transactions, history of payments, minimalistic UI, UPI, sharing, etc. Since all the work that is to be done by this software can also be done manually, but this consumes time, man power and energy. So, this software will be a relief to those who have to do all this work manually. The knowledge of computers and programming has become a basic skill needed to survive in present society.

The motive is to make such kind of a software that is very easy to use, minimalistic in design, with good UI/UX. There wouldn't be need of any training and the person who does not have much knowledge of computers can also use this. All the history of the customers will be stored for further enquiries.

CONTENTS

Chapter 1

INTRODUCTION

This application is used by two users:

1) Admin
2) Coordinator

The main aim is to automate the entire day to day activities of Events like: Logins, new registrations, number of events, transactions, ticketing, UPI and updating.

The coordinators can login to the login page and can register an event according to the participant's choice by providing the necessary credentials. The Coordinator updates the event records, as only coordinators are authorized.

"EVENT MANAGEMENT SYSTEM" has been designed to computerize the following functions that are performed by the system:

- Event details functions
- Registration details
- Login - Logout details
- Event Registration facility
- M-ticket generation
- Security & Encryption
- Responsive UI/UX
- Encrypted Database
- Secure Authentication

1.1 OBJECTIVES

During several decades, personnel function has been transformed from a relatively obscure record keeping staff to a central and top-level management function. There are many factors that have influenced this transformation like technological advances, professionalism and general recognition of human beings as most important resources.

- A coordinator-based management system is designed to handle all the primary information required to register the participants for their choice of events. Separate database is maintained to handle all the details required for the correct registration and transaction.
- The intention is to introduce more user-friendliness in the various activities such as record updating, maintenance, and searching.
- The searching of record has been made quite simple as all the details of the participant can be obtained by simply keying the USN of the participant.
- Similarly, record maintenance and updating can also be accomplished by using the USN of participant with all details being automatically generated. These details are also being automatically updated in the FILE, thus keeping record up-to-date.
- The entire information has been maintained in the database or FILEs and whoever wants to retrieve can't retrieve, as only authorized Coordinators can retrieve the necessary information which can be easily accessible from the FILE/Database.

1.2 SCOPE

PERFORMANCE: Manual handling of the record is time consuming and highly prone to error. Hence to improve the performance, computerized and user-friendly system is undertaken.

EFFICIENCY: Efficiency of the system is a basic need. So, whenever the new participant submits his/her details, it has to be updated automatically. It works on all platforms like Mac, Windows, Linux, iOS, Android, etc.

CONTROL: The complete control is under the admin who has the authority to access, and illegal access is not permitted. Coordinators are given authority to register and they have the rights just to see the entries and not to change the records or any entries.

SECURITY: This is the main criteria of the proposed system. Since illegal access may cause unnecessary actions, so security has been enforced. Here, in this system, we are using 256 Bit AES encryption (256 Bit Advanced Encryption Standard).

Chapter 2

SYSTEM SPECIFICATION

2.1 HARDWARE REQUIREMENTS

Minimum Req.

PROCESSOR: i3 4th gen

RAM: 4GB

HARD DISK: 40MB

Recommended Req.

PROCESSOR: i9 10^{th} gen

RAM: 32GB

HARD DISK: 4TB

2.2 SOFTWARE REQUIREMENTS

OS: Cross platform, 64 bit

CODING LANGUAGE:

- FRONT END: Electron-JS (NODE-JS)

  Electron-JS:

  The Electron framework lets you write cross-platform desktop applications using JavaScript, HTML and CSS. It is based on Node.js and Chromium and is used by the Atom editor and many other apps.

  Node-JS:

  It is an open source, cross-platform, runtime environment that allows developers to create all kinds of server-side tools and applications in JavaScript. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

- BACK END: SQL

  It is a relational database whose components (tables, forms, queries) are linked (related). The linkages between database components are created by making relationship links between them. The relationship can be:

  - One component and another (one-one relationship)
  - One component related to several other components(one-many)
  - Several database components(many-many)

Creation of relationships between the database components reduces data redundancies and enhances ease of access of the information.

Chapter 3

DESIGN

3.1 Description of Event Management Database System

- The details of events are stored in the Events table respective with all its attributes.

  Each entity (USN, R_ID, E_ID, Name, Section, Department, Coordinators) contains primary keys, and foreign keys.
- The above-mentioned key combination is used as a composite key here.
- There are one-to-one and one-to-many relationships available between USN and other attributes of other entities.
- All the entities are normalized and reduce duplicity of records.
- Indexing is implemented on each of the tables of the Event Management System for fast query execution.

There would be many events conducted in a College fest. Each event is identified by its E_ID. Each event provides many registrations. Each event is performed on the mentioned venue and at the mentioned time. An event has many numbers of rules. Each student can register for multiple events.

Each participant's name, USN, E_ID are stored in the registration table. Each student is having a unique USN. Each event is assigned with a coordinator and the participants who are registered to the events will be updated regarding the event by the assigned coordinator of the event.

The coordinator signs in to his profile and registers the event according to the participant's requirement. The participant is needed to give details to register for the events. After registering for the event, the participants are provided with a unique R_ID known as Registration id. The payment is done based on the tariff (price) of the event via any payment mode that is using UPI or CASH. The M-ticket is generated with the QR CODE based on the R_ID of the event.

The R_ID is responsible for keeping track of the status of the students and registration of the events. The details of the students and the event registration are accessible only by the coordinator. The payment is handled by a desk coordinator. After the participant has registered, the M-TICKET is generated on spot.

STEP 1:

Entities are:

1. Students
2. Events
3. Registration
4. Coordinators
5. Transaction
6. Authorization

STEP 2:

RELATIONS:

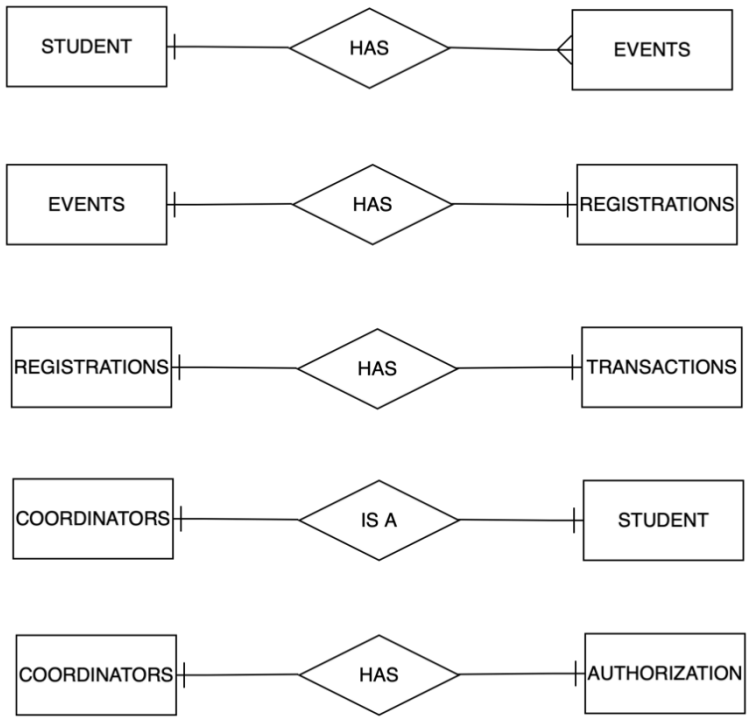

Figure 3.1: relation between the existing tables in the database.

STEP 3:

Key attributes:

- STUDENTS - USN
- EVENTS – E_ID
- COORDINATOR - USN
- REGISTRATION – R_ID
- RULES – E_ID, RULE_NO
- TRANSACTION - R_ID
- AUTHORIZATION - USN

STEP 4:

Other attributes

- STUDENTS - USN, DEPT, PHONE, SECTION, SEM, NAME
- EVENTS - TEAM COUNT, E_ID, NAME, COLOR, VENUE, DURATION, TIME, DATE, CATEGORY, PRICE, RULES
- REGISTRATION - R_ID, E_ID, DESK_USN, TIMESTAMP, USN
- TRANSACTION - AMOUNT, R_ID, MODE, STATUS
- COORDINATORS - ROLE, USN, EVENT
- AUTHORIZATION - PASSWORD, USN

SCHEMA DIAGRAM

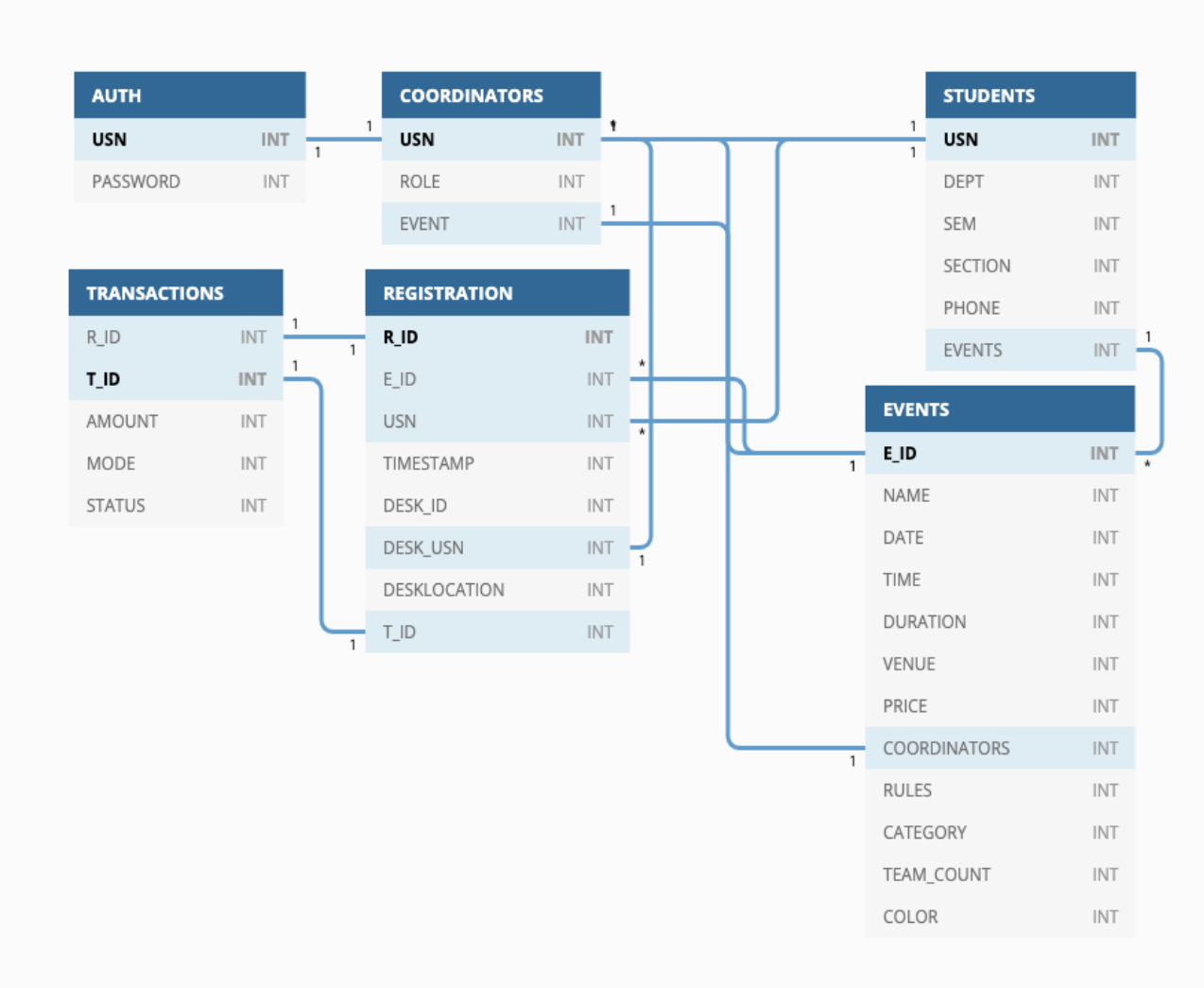

Figure 3.2: Relational schema of the tables present in the database.

STEP 5:

ER DIAGRAM

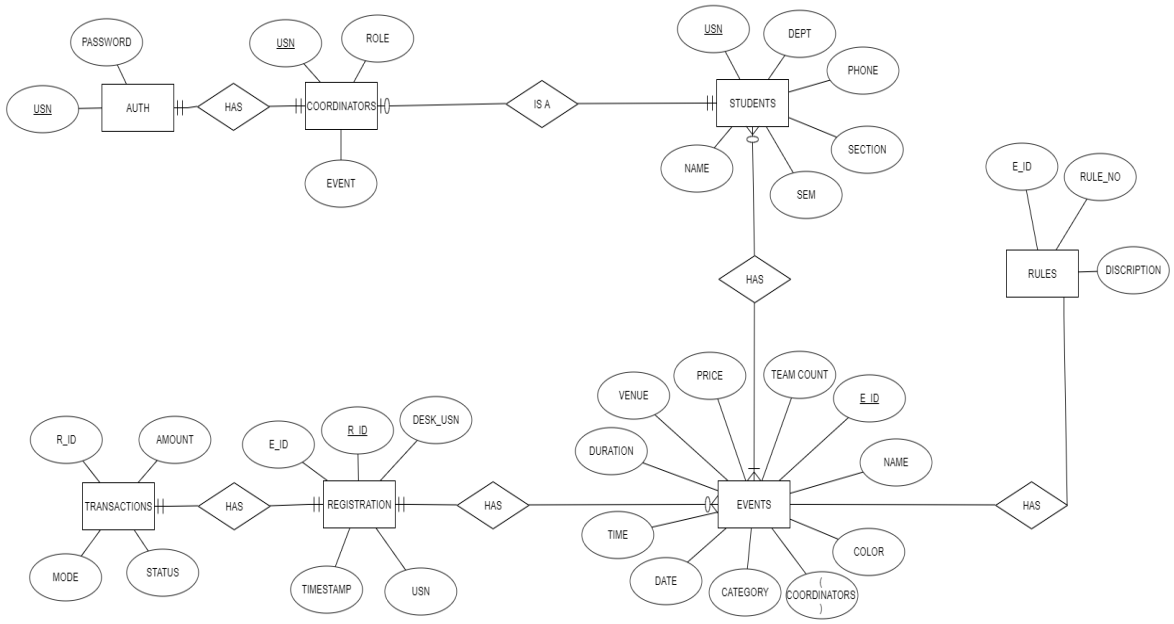

Figure 3.3: ER DIAGRAM for the database

3.2 NORMALISATION

The basic Objectives of normalization is to reduce redundancy, which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of data stored. Relations are normalized so that when a relation in the database is to be altered during the lifetime of the database, information is not. The type of alterations normally needed for relation is:

- Insertion of new data values to relation. This should be possible without being forced to leave blank fields for some attributes.
- Deletion of a tuple, namely, a row of a relation. This should be possible without losing vital information unknowingly.

Functional Dependency:

As the concept of dependency is very important, it is essential that it should be understood first and then proceeded to the idea of normalization. There is no fool-proof algorithmic method of identifying dependency.

Properties of normalized relations:

Ideal relations after normalization should have the following properties:

- No data values should be duplicated in different rows unnecessarily.
- A value must be specified (and required) for every attribute in a row.
- Each relation should be self-contained. In other words, if a row from a relation is deleted, important information should not be accidentally lost.
- When a row is added to a relation, other relations in the database should not be affected.
- A value of an attribute in a tuple may be changed independent of other tuples in the relation and other relations.

Consider the EVENTS table (refer to the schema diagram on figure 3.2).

The prime attributes identified are the attributes which is part of candidate key.

The non-prime attributes are not part of primary key.

When the EVENTS table is split into two tables having the following attributes:

Before Normalization-

Attributes rules was present in the event table before the normalization. (see table 3.1)

Rules attribute stores all the description about the event with multiple points which was containing multiple values.

Table 3.1: EVENTS before any Normalization.

E_ID	NAME	DATE	TIME	DURATION	VENUE	PRICE	COORDINATORS	RULES	CATEGORY	TEAM_COUNT	COLOR
E1001	Code it	2019-11-15	10:00	60	LH-201	80	1AM17CS102	The team should complete the task in given time	TECHNICAL	2	YELLOW
E1002	Fashion show	2019-11-15	17:00	15	MAIN STAGE	800	1AM17CS103	The team must not use any vulgar clothes	MAIN STAGE	8	RED
E1003	Group singing	2019-11-15	10:00	7	MAIN STAGE	400	1AM17CS104	To be completed in given time	MAIN STAGE	5	RED
E1004	Group dance	2019-11-15	11:00	7	MAIN STAGE	500	1AM17CS105	To be completed in given time	MAIN STAGE	6	RED
E1005	Solo singing	2019-11-15	12:00	5	MAIN STAGE	100	1AM17CS106	To be completed in given time	MAIN STAGE	1	RED

1NF

- Each table cell should contain a single value.
- Each record needs to be unique.

According to table (3.2 and 3.3) rules attribute is made into a different table by this each table contains single values and each record is unique.

Therefore, the given tables are in 1NF (1st Normal Form)

2NF

- Be in 1NF
- Single Column Primary Key

According to table (3.2 and 3.3) rules attribute is made into a different table by this each column will be having single column primary key.
Therefore, the given tables are in 2NF (2nd Normal Form)

3NF

- Be in 2NF
- Rule 2- Has no transitive functional dependencies

According to table (3.2 and 3.3) rules attribute is made into a different table by this the two tables does not have transitive functional dependencies.
Therefore, the given tables are in 3NF (3rd Normal Form)

BCNF

- Be in 3NF
- Should not have more than one Candidate Key

According to table (3.2) rules attribute is made into a different table by this there are no multiple candidate keys. There are no other attributes in event table 3.2 which can be a candidate key.
Therefore, the given tables are in BCNF (BCNF Normal Form)

Table 3.2 EVENTS table normalized

E_ID	NAME	DATE	TIME	DURATION	VENUE	PRICE	COORDINATORS	CATEGORY	TEAM_COUNT	COLOR
E1001	Code it	2019-11-15	10:00	60	LH-201	80	1AM17CS102	TECHNICAL	2	YELLOW
E1002	Fashion show	2019-11-15	17:00	15	MAIN STAGE	800	1AM17CS103	MAIN STAGE	8	RED
E1003	Group singing	2019-11-15	10:00	7	MAIN STAGE	400	1AM17CS104	MAIN STAGE	5	RED
E1004	Group dance	2019-11-15	11:00	7	MAIN STAGE	500	1AM17CS105	MAIN STAGE	6	RED
E1005	Solo singing	2019-11-15	12:00	5	MAIN STAGE	100	1AM17CS106	MAIN STAGE	1	RED

Table 3.3 RULES TABLE

E_ID	RULE_NO	RULES
1	1	The team should complete the task in given time
1	2	No usage of internet
2	1	The team must not use any vulgar clothes
2	2	Should be based on some theme
2	3	Maximum of 8 persons in a team
3	1	To be completed in given time
3	2	Maximum of 6 persons in a team
3	3	To get their own bgms
4	1	To be completed in given time

3.3 TRIGGERS

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

BEFORE and AFTER of Trigger:
BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

```
CREATE TRIGGER `ENCRYPT_INSERT` BEFORE INSERT ON `auth`
 FOR EACH ROW SET new.PASSWORD = AES_ENCRYPT(new.PASSWORD, 'nish')
```

The above trigger is executed after a new record is added to the auth table. It executes AES_ENCRYPT() function on the entered password using the given encryption key and stores the encrypted result in the table.

```
CREATE TRIGGER `TRANSACT` AFTER INSERT ON `registration`
 FOR EACH ROW INSERT INTO transactions(R_ID, AMOUNT) VALUES(NEW.R_ID, (SELECT
PRICE FROM events WHERE events.E_ID=NEW.E_ID))
```

The above trigger is executed when a new record in the registration table is added. This trigger inserts a new record in the transactions table containing the same R_ID as the new registration and Registration fee from events table for the respective E_ID.

3.4 Procedures

Stored Procedures are created to perform one or more DML operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not returns a value.

The most important part is parameters. Parameters are used to pass values to the Procedure. There are 3 different types of parameters, they are as follows:

IN:

This is the Default Parameter for the procedure. It always receives the values from calling program.

OUT:

This parameter always sends the values to the calling program.

IN OUT:

This parameter performs both the operations. It Receives value from as well as sends the values to the calling program.

REGISTER

If USN is not present in students table then add student USN, name, phone, Sem., section, department.

If the USN, E_ID pair does not exist in the registration table then

- Add to registration as E_ID, USN, DESK_USN.
- Trigger will add transaction automatically as pending if not it will show error as duplicate registration.
- The transaction is updated as amount, mode as payment and status as ‘paid’ where R_ID =R_ID.
- Display the R_ID for the corresponding registration.

If the above statements become false then it will give error message saying “Duplicate registration”.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `REGISTER`(IN `_E_ID` INT(11), IN
`_USN` VARCHAR(20), IN `_NAME` VARCHAR(20), IN `_PHONE` VARCHAR(20), IN `_SEM`
INT(11), IN `_SECTION` VARCHAR(20), IN `_PAY` VARCHAR(20), IN `_DESKUSN`
VARCHAR(20), IN `_DEPT` VARCHAR(20))
    MODIFIES SQL DATA
```

```
BEGIN
DECLARE _COUNT INT(11);
DECLARE _R_ID INT(11);
DECLARE _EXISTCOUNT INT(11);
   SELECT
      COUNT(*)
           INTO _COUNT
   FROM
      students
   WHERE
      USN = _USN;

IF (_COUNT = 0) THEN
   INSERT INTO students(
      USN,
      NAME,
      PHONE,
      SEM,
      SECTION,
      DEPT
   )
VALUES(
   _USN,
   _NAME,
   _PHONE,
   _SEM,
   _SECTION,
   _DEPT
);
END IF;
SELECT COUNT(*) INTO _EXISTCOUNT FROM registration WHERE USN=_USN AND
E_ID=_E_ID;
IF (_EXISTCOUNT = 0) THEN
INSERT INTO registration(E_ID, USN, DESK_USN)
VALUES(_E_ID, _USN, _DESKUSN);

   SELECT
      R_ID
   INTO _R_ID
   FROM
      registration
   WHERE
      E_ID = _E_ID AND USN = _USN
      ORDER BY `TIMESTAMP` LIMIT 1 ;

UPDATE
   transactions
SET
   `MODE` = _PAY,
   `STATUS` = 'PAID'
WHERE
   R_ID = _R_ID;
   SELECT _R_ID LIMIT 1;
ELSE
      SIGNAL SQLSTATE '45000'
```

```
        SET MESSAGE_TEXT = 'Duplicate registration for USN, E_ID pair';
END IF;
END$$
DELIMITER ;
```

Login

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `LOGIN`(IN `_USN` VARCHAR(11), IN
`PASS` VARCHAR(30))
   READS SQL DATA
BEGIN

IF ((SELECT AES_ENCRYPT(PASS, 'nish') AS `PASSWORD`) = (SELECT `PASSWORD` FROM
`auth` WHERE `USN`=_USN)) THEN
SELECT true AS RESPONSE;
ELSE
SELECT false AS RESPONSE;
END IF;
END$$
DELIMITER ;
```

Change password

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `CHANGEPASS`(IN `_USN`
VARCHAR(11), IN `OLDPASS` VARCHAR(30), IN `NEWPASS` VARCHAR(30))
   MODIFIES SQL DATA
BEGIN

IF ((SELECT AES_ENCRYPT(OLDPASS,'nish') AS `PASSWORD`) = (SELECT `PASSWORD`
FROM auth WHERE USN=_USN)) THEN
      UPDATE auth SET `PASSWORD`= AES_ENCRYPT(NEWPASS,'nish') WHERE USN=_USN;
ELSE
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Incorrect Password. Please check the
old password you have entered.';
END IF;
END$$
DELIMITER ;
```

Chapter 4

IMPLEMENTATION AND CODING

4.1 Source Code

FILE: main.js

```
const path = require('path')

const {
      app,
      BrowserWindow,
      dialog,
      ipcMain,
} = require('electron')

require('electron-reload')(__dirname, {
   electron: require(`${__dirname}/node_modules/electron`)
});

let loginWin, eventWin, viewEventWin, userWin, loginUSN, aboutWin, isAdmin = false
const Window = require('./js/Window')

const db = require('./js/db')
const showError = require('./js/showError')


createWindow = () => {
      const {screen} = require('electron')
      var dimensions = screen.getPrimaryDisplay().size, zFactor = 1440/dimensions.width;

      loginWin = new Window({
            file: 'index.html',
            zoom: zFactor,
            width: 700,
            height: 500,
      })

      loginWin.once('show', () => {
            db.connect((err) => {if (err) showError(err)});
      })

      ipcMain.on('event-window', (e, usn) => {
   if (!eventWin) {
    eventWin= new Window({
              file: 'events.html',
              zoom: zFactor,
              width: 1079,
            height: 720,
            parent: loginWin
     })
```

```
        loginUSN = usn

        eventWin.once('show', () => {
                db.query(`SELECT ROLE as ROLE FROM coordinators where USN =
\'${loginUSN}\' `, function(err, result, fields) {
                        if (err) showError(err)
                        else isAdmin = (result[0]['ROLE'] != "Coordinator") ? true : false;
                });
                db.query("SELECT E_ID,NAME,CATEGORY,COLOR FROM events order by case
when category = 'MAIN STAGE' then 0 else 1 end, category ", function(err, result, fields) {
                        if (err) showError(err)
                        else eventWin.webContents.send('events', result , isAdmin)
                });
      });

     // cleanup
     eventWin.on('closed', () => {
       eventWin = null
     })
   }
 })

 ipcMain.on('view-event-window', (e, id) => {
            if (!viewEventWin) {
                viewEventWin = new Window({
                        file: 'view_event.html',
                        zoom: zFactor,
                        width: 800,
                        height: 680,
                        parent: eventWin,
                })

                viewEventWin.once('show', () => {
                        db.query(`SELECT e.* , s.name  as a, s.phone as b FROM events e, coordinators c,
students s WHERE E_ID=\'${id}\' and e.COORDINATOR = c.usn and s.usn = e.COORDINATOR`,
function(err, result, fields) {
                                if (err) showError(err)
                                else viewEventWin.webContents.send('view-event', result, loginUSN)
                        });
                })

                // cleanup
                viewEventWin.on('closed', () => {
                        viewEventWin = null
                })
            }
      })

      ipcMain.on('edit-event-window', (e, id) => {
            if (!viewEventWin) {
                viewEventWin = new Window({
                        file: 'edit_event.html',
                        zoom: zFactor,
                        width: 620,
```

```
                    height: 530,
                    parent: eventWin
                })

                viewEventWin.once('show', () => {
                    (id != 'add') ? db.query(`SELECT e.* , s.name  as a, s.phone as b FROM events e,
coordinators c, students s WHERE E_ID=\'${id}\' and e.COORDINATOR = c.usn and s.usn =
e.COORDINATOR`, function(err, result, fields) {
                        if (err) showError(err)
                        else viewEventWin.webContents.send('edit-event', result[0])
                    }) : viewEventWin.webContents.send('edit-event')
                })

                // cleanup
                viewEventWin.on('closed', () => {
                    viewEventWin = null
                    db.query(`SELECT ROLE as ROLE FROM coordinators where USN =
\'${loginUSN}\' `, function(err, result, fields) {
                        if (err) showError(err)
                        else isAdmin = (result[0]['ROLE'] != "Coordinator") ? true : false;
                    });
                    db.query("SELECT E_ID,NAME,CATEGORY,COLOR FROM events order by
case when category = 'MAIN STAGE' then 0 else 1 end, category ", function(err, result, fields) {
                        if (err) showError(err)
                        else eventWin.webContents.send('events', result , isAdmin)
                    });
                })
            }
        })

        ipcMain.on('user-window', () => {
            if (!userWin) {
                userWin = new Window({
                    file: 'user.html',
                    zoom: zFactor,
                    width: 300,
                    height: 350,
                    parent: eventWin
                })

                userWin.once('show', () => {
                    db.query(`SELECT  * FROM auth a,coordinators c,students s WHERE
a.USN=\'${loginUSN}\' and a.USN=c.USN and a.USN=s.USN`, function(err, result, fields) {
                        if (err) showError(err)
                        else userWin.webContents.send('view-user', result[0])
                    });
                })

                // cleanup
                userWin.on('closed', () => {
                    userWin = null
                })
            }
        })
```

```
	ipcMain.on('edit-user-window', (e, id) => {
		if (!userWin) {
			userWin = new Window({
				file: 'edit_user.html',
				zoom: zFactor,
				width: 740,
				height: 620,
				parent: eventWin
			})

			userWin.once('show', () => {
				db.query(`SELECT  * FROM auth a,coordinators c,students s WHERE
a.USN=\'${loginUSN}\' and a.USN=c.USN and a.USN=s.USN`, function(err, result, fields) {
					if (err) showError(err)
					else userWin.webContents.send('edit-user', result[0])
				});
			})

			// cleanup
			userWin.on('closed', () => {
				userWin = null
			})
		}
	})

	ipcMain.on('about-window', () => {
		if (!aboutWin) {
			aboutWin = new Window({
				file: 'about.html',
				zoom: zFactor,
				width: 450,
				height: 350,
				parent: loginWin
			})

			// cleanup
			aboutWin.on('closed', () => {
				aboutWin = null
			})
		}
	})

	loginWin.on('closed', () => {
		loginWin = null
	})
}

app.on('ready', createWindow)

app.on('window-all-closed', () => {
	app.quit()
})
```

FILE: db.js

```
var mysql = require('mysql');
```

```
const db = mysql.createConnection({
        host: "localhost",
        user: "root",
        password: "",
        database: "frontdesk"
});

module.exports = db;
```

FILE: edit_event.js

```
'use strict'

const { ipcRenderer, remote } = require('electron')
const db = require('./js/db')
const showError = require('./js/showError')
const moment = require('moment');

const setValue = (field, value) => {
    document.getElementById(field).value = value;
}

let color = ''

const setColor = () => {
    document.getElementById('body').querySelectorAll('.item-container').forEach(i => {
        i.style.width = '60px'
        i.style.height = '40px'
        i.style.border = '0px'
    })
    let selectedItem = document.getElementById(color);
    selectedItem.style.border = '5px solid #E6672F'
    selectedItem.style.width = '63px'
    selectedItem.style.height = '43px'
}

ipcRenderer.on('edit-event', (e, eventData) => {

    if(!eventData) document.getElementById('header').innerHTML = 'Add Event'
    else {
        setValue('name',eventData['NAME'])
        setValue('category',eventData['CATEGORY'])
        setValue('venue',eventData['VENUE'])
        setValue('price',eventData['PRICE'])
        setValue('count',eventData['TEAM_COUNT'])
        setValue('coordinator',eventData['COORDINATOR'])
        setValue('time', moment(eventData['TIME'], "HH:mm:ss", true).format("HH:mm:ss"))
        setValue('date', moment(eventData['DATE']).format("YYYY-MM-DD"))
        setValue('duration', eventData['DURATION'])
        color = eventData['COLOR'] || 'RED'
        setColor()
    }
    document.getElementById('body').querySelectorAll('.item-container').forEach(i => {
        i.addEventListener('click', (e) => {
```

```
            color = i.id;
            setColor();
        })
    })


    document.getElementById('change2').addEventListener('click', (e) => {
        e.preventDefault();

          db.query( eventData ? `UPDATE events SET NAME =
\'${document.getElementById("name").value}\', COLOR=\'${color}\', TIME =
\'${moment(document.getElementById("time").value , "HH:mm", true).format("HH:mm:ss")}\',
DURATION = ${document.getElementById("duration").value}, DATE =
\'${document.getElementById("date").value}\', VENUE =
\'${document.getElementById("venue").value}\', PRICE =
\'${document.getElementById("price").value}\', COORDINATOR =
\'${document.getElementById("coordinator").value}\', TEAM_COUNT =
${document.getElementById("count").value}, CATEGORY =
\'${document.getElementById("category").value}\' WHERE E_ID = ${eventData["E_ID"]}` :
          `CALL
ADDEVENT(\'${document.getElementById("name").value}\',\'${document.getElementById("date").
value}\',\'${moment(document.getElementById("time").value , "HH:mm",
true).format("HH:mm:ss")}\',${document.getElementById("duration").value},
\'${document.getElementById("venue").value}\', \'${document.getElementById("price").value}\',
\'${document.getElementById("coordinator").value}\',
\'${document.getElementById("category").value}\', ${document.getElementById("count").value},
\'${color}\')`, function(err, result, fields) {
            if (err) showError(err)
            else{
                document.getElementById('body').innerHTML = "<center><img src=\"images/tick.gif\"
alt=\"tick\" class=\"tick\" id=\"tick\" width=\"380px\" style=\"margin-top:60px;\"></center>"

                    setTimeout(function() {
                        setInterval(function() {
                            $('#tick').attr('src', $('#tick').attr('src'))
                        }, 1)
                        document.getElementById('body').className = 'hidden35';
                    remote.getCurrentWindow().close()
                    }, 1800)
            }
        });
    })
})
```

FILE: edit_user.js

```
'use strict'

const { ipcRenderer, remote } = require('electron')
const db = require('./js/db')
const showError = require('./js/showError')

const setValue = (field, value) => {
    document.getElementById(field).value = value;
}
```

```
const setText = (field, value) => {
   document.getElementById(field).innerHTML = value;
}

let userList = '', selectedUSN = '' , oldPass = ''

const viewEditUser = (user) => {
   selectedUSN = user['USN']
   document.getElementById('userInfo').innerHTML = '<b>Edit Coordinator<br>USN: </b>' +
user['USN']
   document.getElementById('AllUsers').style.display = 'none'
   document.getElementById('changeForm').style.display = 'block'

   setValue('name', user['NAME'])
   setValue('phone', user['PHONE'])
   setValue('dept', user['DEPT'])
   setValue('sem', user['SEM'])
   setValue('section', user['SECTION'])
   setValue('role', user['ROLE'])
   oldPass = user['PASSWORD']
}

const updateCoord = (name, phone, dept, sem, section, role) => {
       db.query(`update students set
           DEPT=\'${dept}\',
           SEM=${sem},
           SECTION=\'${section}\',
           PHONE=${phone},
           NAME=\'${name}\'
           WHERE USN=\'${selectedUSN}\'`,
             function(err, result, fields) {
                   if (err) showError(err)
                   else db.query(`update coordinators set
                ROLE = \'${role}\'
                WHERE USN=\'${selectedUSN}\'`,
                        function(err2, result, fields) {
                              if (err2) showError(err2)
                              else {
                    document.getElementById('changeForm').style.display = 'none';
                    document.getElementById('body').innerHTML = "<center><img
src=\"images/tick.gif\" alt=\"tick\" class=\"tick\" id=\"tick\" width=\"260px\" style=\"margin-
top:130px;\"></center>"
                    document.getElementById('body').style.display = 'block';
                    document.getElementById('tick').className = 'visible15';

                    setTimeout(function() {
                       setInterval(function() {
                          $('#tick').attr('src', $('#tick').attr('src'))
                       }, 1)
                       document.getElementById('tick').className = 'hidden35';
                       remote.getCurrentWindow().close()
                    }, 1800)
                }
                        })
```

```
            });
}

ipcRenderer.on('edit-user', (e, userData) => {
   document.getElementById('changeForm').style.display = 'none'

   document.getElementById('userInfo').innerHTML =
      `<b>USN:</b> ${userData['USN']}        
      <b>Name:</b> ${userData['NAME']}        
      <b>Class:</b> ${userData['DEPT']} ${userData['SEM']} ${userData['SECTION']}
      <br>
      <b>Phone:</b> +91 ${userData['PHONE']}        
      <b>Role:</b> ${userData['ROLE']}`


      db.query(`SELECT  s.*, c.*, AES_DECRYPT(a.PASSWORD, 'nish') as PASSWORD FROM
auth a,coordinators c,students s WHERE a.USN=c.USN and a.USN=s.USN`, function(err, result,
fields) {
         if (err) showError(err)
         else {
            result.forEach((r, index) => {
               userList += `<tr id="${index}" class="tableItem">
                           <td>${r['USN']}</td>
                           <td>${r['NAME']}</td>
                           <td>${r['PHONE']}</td>
                           <td>${r['DEPT']} ${r['SEM']} ${r['SECTION']}</td>
                           <td>${r['ROLE']}</td>
                        </tr>`
            })
            document.getElementById('tableBody').innerHTML = userList

            document.getElementById('body').querySelectorAll('.tableItem').forEach(i => {
               i.addEventListener('click', () => {viewEditUser(result[i.id])})
            })
         }
      });

   document.getElementById('editButton').addEventListener('click', (e) => {
      e.preventDefault();

      let name = document.getElementById("name").value,
         phone = document.getElementById("phone").value,
         dept = document.getElementById("dept").value,
         sem = document.getElementById("sem").value,
         section = document.getElementById("section").value,
         role = document.getElementById("role").value,
         newpass = document.getElementById("pass").value,
         newpass2 = document.getElementById("confirm").value

      if (newpass.length>0 || newpass2.length>0) {
         if(newpass==newpass2) {
            db.query(`CALL CHANGEPASS(\'${selectedUSN}\',\'${oldPass}\',\'${newpass}\')`, (err,
result, fields) => {
               if(err) showError(err)
               else updateCoord(name, phone, dept, sem, section, role)
```

```
            })
        } else showError("Passwords do not match.")
    } else updateCoord(name, phone, dept, sem, section, role)
  })

})
```

FILE: index.js

```
'use strict'

const { ipcRenderer } = require('electron')
const db = require('./js/db')
const showError = require('./js/showError')

const debug = !true

document.getElementById('login').addEventListener('click', !debug ? (e) => {
  e.preventDefault();

  let usn = document.getElementById('username').value.toUpperCase();
  let pass = document.getElementById('pass').value;

  db.query(`CALL LOGIN(\'${usn}\', \'${pass}\')`, (err, result, fields) =>
    result[0][0]['RESPONSE'] ? ipcRenderer.send('event-window', usn) : showError("Invalid
Username or Password")
  )
} : () => ipcRenderer.send('event-window', "1AM17CS101"));

document.getElementById('aboutButton').addEventListener('click', () => {
  ipcRenderer.send('about-window')
});
```

FILE: view_event.js

```
'use strict'

const { ipcRenderer } = require('electron')
const moment = require('moment');
var QRCode = require('qrcode')
const db = require('./js/db')
const showError = require('./js/showError')

String.prototype.capitalizeEachWord = function() {
    this.toLowerCase();
    let str = this.split(" ");

    for (var i = 0, x = str.length; i < x; i++)
        str[i] = str[i][0].toUpperCase() + str[i].substr(1);

    return str.join(" ");
}

let detailsContent, registerContent, eventData, userData, registerStatus = false, rulesList = '',
DeskUSN, payment, regList = '';
```

```
const collapseLeft = function() {
	document.querySelector('.container-left').style.width = '450px';
	document.getElementById('categoryContainer').className = 'hidden35';
	document.getElementById('body').className = 'hidden35';

	document.getElementById('body').innerHTML = registerContent;

	document.getElementById('body').className = 'visible35';

	document.getElementById('register').addEventListener('click', (e) => {
		e.preventDefault();

		userData = {
			USN: document.getElementById('USN').value,
			name: document.getElementById('Name').value,
			phone: document.getElementById('phone').value,
			sem: document.getElementById('sem').value,
			sec: document.getElementById('section').value,
			dept: document.getElementById('dept').value,
		}

		register(eventData['E_ID'], userData, DeskUSN)
	});

	document.getElementById('USN').addEventListener('input', () => {
		checkUSN(document.getElementById('USN').value)
	});
}

const expandLeft = function() {
	document.querySelector('.container-left').style.width = '720px';
	document.getElementById('categoryContainer').className = 'visible15';
	document.getElementById('body').className = 'hidden35';

	rulesList = ''
	getRules(eventData['E_ID'])
	document.getElementById('body').innerHTML = detailsContent;

	document.getElementById('body').className = 'visible35';
}

const checkUSN = (usn) => {
	if(usn.length == 10) {
		document.getElementById('body').querySelectorAll('.fetchable').forEach(f => {
			f.style.display = 'flex';
		})
		db.query(`SELECT * FROM students WHERE USN=\'${usn}\'`, (err, result, fields) => {
			if (result.length>0) {
				document.getElementById('Name').value = result[0]['NAME'];
				document.getElementById('phone').value = result[0]['PHONE'];
				document.getElementById('sem').value = result[0]['SEM'];
				document.getElementById('section').value = result[0]['SECTION'];
				document.getElementById('dept').value = result[0]['DEPT'];
			} else {
				document.getElementById('Name').value = "";
```

```
                    document.getElementById('phone').value = "";
                    document.getElementById('sem').value = "";
                    document.getElementById('section').value = "";
                    document.getElementById('dept').value = "";
                }
            })
        } else {
            document.getElementById('body').querySelectorAll('.fetchable').forEach(f => {
                f.style.display = 'none';
            })
        }
}

const register = (e_id, user) => {
    db.query(`CALL REGISTER(${e_id}, \'${user.USN}\', \'${user.name}\', \'${user.phone}\',
${user.sem}, \'${user.sec}\', \'${payment}\', \'${DeskUSN}\', \'${user.dept}\')`, (err, result, fields) =>
{
            if(err) showError(err)
            else {
                registerStatus = true;
                document.getElementById('body').className = 'hidden15';
                document.getElementById('body').className = 'hidden15';
                document.getElementById('eventHeader').style.display = 'none';
                document.getElementById('categoryContainer').style.display = 'none';
                document.getElementById('hr').style.display = 'none';
                document.getElementById('body').innerHTML = ""
                document.getElementById('body').className = 'visible15';
                document.getElementById('body').innerHTML = "<img src=\"images/tick.gif\"
alt=\"tick\" class=\"tick\" id=\"tick\" width=\"380px\">"
                document.getElementById('qr').style.visibility = 'visible';

                setTimeout(function() {
                    setInterval(function() {
                        $('#tick').attr('src', $('#tick').attr('src'))
                    }, 1)
                    document.getElementById('body').className = 'hidden35';
                    document.getElementById('containerLeft').style.width = '0px';

                    document.getElementById('backText').innerHTML = "<span class=\"fa-icon body-
items\" data-placeholder=\"&#xf030;\" style=\"color: #CCC;\"></span><br>Screenshot this M-
Ticket<br>and share it with the participants"
                }, 1800)

                let qrData = Buffer.from(`{"R_ID": ${result[0][0]['_R_ID']}}`).toString('base64');
                console.log(qrData);
                QRCode.toCanvas(qr, qrData, {scale: 3.5}, (error) => {
                  if (error) console.error(error)
                })

                document.getElementById('ticketSub').innerHTML +=
`<br><br>ADMITS<br><b>${user.name}</b><br><br><br>Coordinated by
${eventData['a']}<br>(+91 ${eventData['b']})`
            }
        })
```

```
}

const getRules = (e_id) => {
	db.query(`SELECT * FROM rules WHERE E_ID=\'${e_id}\' ORDER BY RULE_NO`, (err,
result, fields) => {
		if(!err) {
			let r;
			result.forEach((r) => {
				rulesList += r['RULE_NO'] + '. ' + r['RULES'] + '<br>'
			})
			document.getElementById('rlist').innerHTML = rulesList.length>0 ? rulesList : "No
rules available";
		} else console.log(err);
	})
	getRegistrations(e_id);
}

const setPayment = (mode) => {
	payment = mode;
}

ipcRenderer.on('view-event', (e, obj, loginUSN) => {
	eventData = obj[0];

	DeskUSN = loginUSN

	// Heading
	document.getElementById('eventHeader').innerHTML =
eventData['NAME'].capitalizeEachWord()

	//Category Label
	document.getElementById('categoryContainer').innerHTML =
		`<div class="item-container-min gradient-${eventData['COLOR'].toUpperCase()}">
  ${eventData['CATEGORY'].toUpperCase()}
  </div>`
	//Event desription
	registerContent = `<form class="login100-form validate-form p-b-33 p-t-5">

	<div class="container-login100-form-btn" style="margin-top:10%;">
	  <button id="register" class="login100-form-btn gradient-${obj[0]['COLOR'].toUpperCase()}-
left">
	   REGISTER
	  </button>
	</div>
  </form>`

	getRules(eventData['E_ID']);

	detailsContent = `<span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}"
data-placeholder="&#xf073;"></span>
		${moment(eventData['DATE']).format("MMM DD,
YYYY")}      
		<span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}" data-
placeholder="&#xf017;"></span>
```

```
                ${moment(eventData['TIME'], "HH:mm:ss", true).format("hh:mm A")} -
${moment(eventData['TIME'], "HH:mm:ss", true).add(eventData['DURATION'],
'minutes').format("hh:mm A")}<br><br>
                 <span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}" data-
placeholder="&#xf02d;"></span>
               <div id="rlist" style="position: relative; top: -40px; left: 70px; margin-bottom: -
70px;">${rulesList}</div><br><br>
               <span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}" data-
placeholder="&#xf0c0;"></span>
                ${eventData['TEAM_COUNT']} in a
team        
                              <span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}"
data-placeholder="&#xf0d6;"></span>
                              ${eventData['PRICE'] ? "Rs. " + eventData['PRICE'] :
"Free"}        
             <span class="fa-icon body-items ${eventData['COLOR'].toUpperCase()}" data-
placeholder="&#xf2bb;"></span>
               ${eventData['a']} (+91 ${eventData['b']})<br><br>`



    document.getElementById('body').innerHTML = detailsContent;
    // Ticket
    let gradient = getComputedStyle(document.querySelector('.gradient-' +
eventData['COLOR'].toUpperCase())).background.split('-webkit-linear-gradient(top, ')[1].split('
100%)')[0].split(' 0%, ');
    document.getElementById('grad1').innerHTML += `<stop offset="0%" style="stop-color:
${gradient[0]}" />
  <stop offset="100%" style="stop-color: ${gradient[1]}" />`

    document.getElementById('ticketContent').innerHTML += `<p class="ticket-content-head no-
select">${eventData['NAME'].capitalizeEachWord()}</p>
                              <p class="ticket-content-sub no-select"
id="ticketSub">${moment(eventData['DATE']).format("MMM DD, YYYY")}, 
                              ${moment(eventData['TIME'], "HH:mm:ss", true).format("hh:mm
A")}</p>
                                                  <div class="qr-container no-
select"><canvas id="qr" class="qr"></canvas></div>`;

    document.getElementById('ticketContainer').addEventListener('click', () => {
        if(!registerStatus) {
            if (document.querySelector('.container-left').style.width != '450px')
                collapseLeft();
            else expandLeft();
        }
    });

    QRCode.toCanvas(qr, "Please Register First", {scale: 3.5}, (error) => {
      if (error) console.error(error)
    })


})
```

4.4 Database Design

Creation Of Tables

AUTH

```
CREATE TABLE `auth` (
  `USN` varchar(10) NOT NULL,
  `PASSWORD` varchar(30) NOT NULL
)

ALTER TABLE `auth`
  ADD PRIMARY KEY (`USN`);

ALTER TABLE `auth`
  ADD CONSTRAINT `auth_ibfk_1` FOREIGN KEY (`USN`) REFERENCES `coordinators`
(`USN`) ON DELETE CASCADE;
```

EVENTS

```
CREATE TABLE `events` (
  `E_ID` int(11) NOT NULL,
  `NAME` varchar(20) NOT NULL,
  `DATE` date NOT NULL,
  `TIME` time NOT NULL,
  `DURATION` int(11) NOT NULL,
  `VENUE` varchar(20) NOT NULL,
  `PRICE` int(11) NOT NULL,
  `COORDINATOR` varchar(20) DEFAULT NULL,
  `CATEGORY` varchar(20) NOT NULL,
  `TEAM_COUNT` int(11) NOT NULL,
  `COLOR` varchar(20) NOT NULL
)

ALTER TABLE `events`
  ADD PRIMARY KEY (`E_ID`),
  ADD KEY `events_ibfk_1` (`COORDINATOR`);

ALTER TABLE `events`
  ADD CONSTRAINT `events_ibfk_1` FOREIGN KEY (`COORDINATOR`) REFERENCES
`coordinators` (`USN`);
```

REGISTRATION

```
CREATE TABLE `registration` (
  `R_ID` int(11) NOT NULL,
  `E_ID` int(11) NOT NULL,
  `USN` varchar(20) NOT NULL,
  `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `DESK_USN` varchar(20) NOT NULL
)

ALTER TABLE `registration`
  ADD PRIMARY KEY (`R_ID`),
  ADD KEY `E_ID` (`E_ID`),
```

```
  ADD KEY `USN` (`USN`),
  ADD KEY `registration_ibfk_4` (`DESK_USN`);

ALTER TABLE `registration`
  ADD CONSTRAINT `registration_ibfk_2` FOREIGN KEY (`E_ID`) REFERENCES `events`
(`E_ID`) ON DELETE CASCADE,
  ADD CONSTRAINT `registration_ibfk_3` FOREIGN KEY (`USN`) REFERENCES `students`
(`USN`) ON DELETE CASCADE,
  ADD CONSTRAINT `registration_ibfk_4` FOREIGN KEY (`DESK_USN`) REFERENCES
`students` (`USN`);
```

RULES

```
CREATE TABLE `rules` (
  `E_ID` int(11) NOT NULL,
  `RULE_NO` int(11) NOT NULL,
  `RULES` varchar(200) NOT NULL
)

ALTER TABLE `rules`
  ADD PRIMARY KEY (`E_ID`,`RULE_NO`);

ALTER TABLE `rules`
  ADD CONSTRAINT `rules_fk1` FOREIGN KEY (`E_ID`) REFERENCES `events` (`E_ID`);
```

STUDENTS

```
CREATE TABLE `students` (
  `USN` varchar(20) NOT NULL,
  `DEPT` varchar(20) NOT NULL,
  `SEM` int(11) NOT NULL,
  `SECTION` varchar(20) NOT NULL,
  `PHONE` varchar(20) NOT NULL,
  `NAME` varchar(20) NOT NULL
)

ALTER TABLE `students`
  ADD PRIMARY KEY (`USN`);
```

TRANSACTIONS

```
CREATE TABLE `transactions` (
  `R_ID` int(11) NOT NULL,
  `AMOUNT` int(11) NOT NULL DEFAULT '0',
  `MODE` varchar(20) DEFAULT NULL,
  `STATUS` varchar(20) NOT NULL DEFAULT 'PENDING'
)

ALTER TABLE `transactions`
  ADD PRIMARY KEY (`R_ID`),
  ADD KEY `R_ID` (`R_ID`);

ALTER TABLE `transactions`
  ADD CONSTRAINT `trans_ibfk_1` FOREIGN KEY (`R_ID`) REFERENCES `registration`
(`R_ID`) ON DELETE CASCADE;
```

COORDINATORS

```
CREATE TABLE `coordinators` (
  `USN` varchar(20) NOT NULL,
  `ROLE` varchar(20) NOT NULL
)

ALTER TABLE `coordinators`
  ADD PRIMARY KEY (`USN`);

ALTER TABLE `coordinators`
  ADD CONSTRAINT `coordinators_ibfk_1` FOREIGN KEY (`USN`) REFERENCES `students`
(`USN`) ON DELETE CASCADE;
```

REGISTRATION

```
CREATE TABLE `registration` (
  `R_ID` int(11) NOT NULL,
  `E_ID` int(11) NOT NULL,
  `USN` varchar(20) NOT NULL,
  `TIMESTAMP` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `DESK_USN` varchar(20) NOT NULL
)

ALTER TABLE `registration`
  MODIFY `R_ID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=89;
```

CONCLUSION

The application is based on the operational aspects of an event and a study of all the functions of the different operations done in an event.

It gives complete exposure of the requirement of the management for smooth functioning. The management policies differ from one event to another in thus outlook but there is not much difference in the functioning of the events.

There are different forms and tables are used. The data is stored in tables automatically. However, the whole system cannot be changed, but the computerized system designed not only saves time but at the same time reduces labor & expenditure. In the traditional system, there were lot of irregularities founds in generating data to where as in modified and computerized system in every problem overcome with the press of button. This system provides the security from loss, disclosure, modification and destruction of data. This system provides integrity of proper functioning of programs.

REFERENCES

[1] : www.google.com

[2] : Fundamentals of Database Systems, Seventh Edition, Navathe, Pearson.

[3] : The Complete Reference J2EE, Seventh Edition, Mc-Graw Hill.

[4] : www.w3schools.com

[5] : www.github.com

[6] : https://bootstrapdocs.com/

[7] : https://bootstrapdocs.com/

[8] : https://mariadb.com/kb/en/library/documentation/

[9] : https://devdocs.io/html/

[10] : https://devdocs.io/javascript/

[11] : https://devdocs.io/css/

[12] : https://nodejs.org/en/docs/