

CONTACT MANAGEMENT SYSTEM

Advance Data Base

603-C

Wizards at work



Sacred Heart University
School of Computer Science & Engineering
The Jack Welch College of Business & Technology

Submitted To:
Dr. Reza Sadeghi

Spring 2022

Final Project Report of Contact Management System

Team Name

Wizards at Work

Team Members

- | | |
|-----------------------------|--|
| 1. Prudhvi Sai Akhil Thumu | thumup@mail.sacredheart.edu (Team Head) |
| 2. Sandeep Yepuri | yepuris@mail.sacredheart.edu (Team Member) |
| 3. Nikhilender Reddy Baddam | baddamn2@mail.sacredheart.edu (Team Member) |
| 4. Chetan Sai Tallamudi | tallamudic@mail.sacredheart.edu (Team Member) |

Roles of Team Members

1. Prudhvi sai akhil Thumu.

I'm Prudhvi sai akhil Thumu, Graduate Computer Science student at Sacred Heart University. Completed my Under graduation from Gitam University in India and, I acquired experience in IT industry as a cloud DevOps Engineer. My role in this project is to be Monitoring the team activities as a team lead, involving in the development of GUI and Database system for Contact management System and Creating Different Tables which are essential for the Contact Management System (CMS).

2. Sandeep Yepuri.

I'm Sandeep Yepuri, Graduate Computer Science student at Sacred Heart University. Completed my Under graduation from KLU University in India and, I acquired experience in IT industry as a full stack developer. My role in this project to develop UI interface for contact management system and will share some work in SQL.

3. Chetan Sai Tallamudi.

I'm Chetan Sai Tallamudi, Graduate Computer Science student at Sacred Heart University. Completed my Under graduation from St Joseph in India. My role in this project to develop in order to connect the GUI with backend database and will do some part of work in database.

3. Nikhilender Reddy Baddam.

I'm Nikhilender Reddy Baddam, Graduate Computer Science student at Sacred Heart University. Completed my Under graduation from VNR VJIET institute of Technology in India. I acquired experience in IT industry as a Python developer. My role in this project to develop the database and will share work in developing GUI for Contact Management System and SQL tables.

Table of Contents

Table of Figures.....	5
Project description.....	6-7
Entity Relationship Diagram Description.....	7-8
Enhanced Entity Relationship Diagram Description.....	8-9
Database Description.....	10-13
Loading data & Performance enhancements	13-20
Graphical User Interface.....	20-36
Conclusion and Future work.....	36
GitHub Repository Address.....	37
References.....	37

Table of Figures

Figure1: ER Diagram	8
Figure2: EER Diagram	9
Figure3: Execution Plan	18
Figure 4: CMS login Page	21
Figure 5: Admin Main menu Page	24
Figure 6: User Main menu Page	27
Figure 7: Action page to add a user	30
Figure 8: Action page to delete user.....	34

Project Description

Contact Management System (CMS) [1] stores different types of information such as Telephone number, Mailing Address, Contact name, Phone number, Fax Number, Home number and Address. The CMS will store the user data in distinct SQL tables with different user types.

- CMS can ask the admin username and password to login and password should contain 8 characters. CMS can be able to change the admin user and Password.
- Admin user or root user can add new user to the CMS by adding the username and password to the database which cannot done by the normal user.
- Admin user can remove user from the CMS by removing username, password and any other related data.
- Every user can be able to add the contact information like first name, sur name, phone number, workplace number, workplace address, home address, zip code, fax number, email address, gender and age.
- Every user can be able to remove the contact information
- Every user can be able to edit the contact details
- CMS can display the information using various search options where users are able to search with contact number to get other details like name and mail address
- CMS graphical user interface (GUI) is very user friendly where it can show the warnings where the user is trying to put the contact information which is exist in tables.
- CMS GUI shows a beautiful welcome page
- CMS GUI can show all available options and functions to the end user
- CMS can show the reports in the tabular form

- CMS can be able to provide the exit functionality on the GUI

Entity Relation Diagram Description:

Entity Relation diagram [2] represents the basic design upon which database is built. The main entities of contact management system are the User, Login, Permission, Company, Customer, Address etc. Each entity will have the primary key and some entities will contain the foreign keys. In our entity relationship diagram, we have also showed different functionalities like partial participation, total participation, multivalued attribute, composite attribute etc.

1. Partial Participation - User table
2. Total Participation – Customer table
3. Multivalued attribute – emails and contacts (single user may have multiple emails and multiple contacts)
4. Composite attribute – Name (F_name, M_name, L_name)
5. Derived attribute – age of the user can be derived from the date of birth of the user.
6. Weak entities – Email table and contact's table

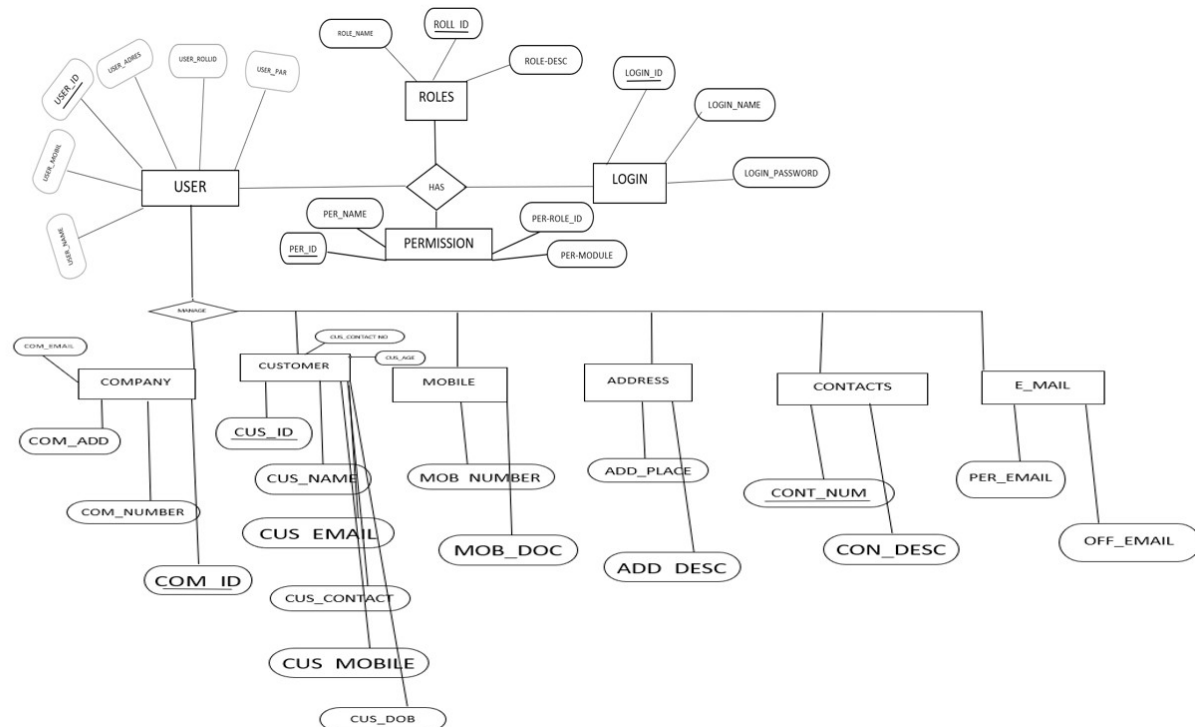


Figure 1: Entity Relationship Diagram for CMS

Enhanced Entity Relation Diagram Description:

Enhanced Entity Diagram [3] helps us create and maintain detailed databases through high level models and they are developed based on the ER diagram. All the tables shown below are to implement the database for Contact Management System (CMS). Data which is going to store inside the CMS database is purely end user information who are working in different organizations and having different roles. For CMS to maintain the relationship between among the entities in the tables used one to many relationships and one to one relationship.

Enhanced Entity Relation diagram for CMS consists of 10 tables which are stated below

1. Login (login_id, login_role_id, login_username, user_password)
2. User (user_id, user_name, user_mobile, user_address, user_email, user_rolid, user_perid)
3. Roles (role_id, role_name, role_desc)
4. Address (add_place, add_desc, add_zip)

5. Mobile (mobile_des, mob_num)
6. Permission (per_id, per_role_id, per_name, per_module)
7. Contacts (con_num, con_des)
8. Company (company_id, company_name, company_add, company_num, company_mail, company_lev)
9. Email (per_email,office_mail)
10. Customer (cus_id, cus_name, cus_mobile, cus_email, cus_add, cus_comid, company_com_id, cus_dob,cus_age)

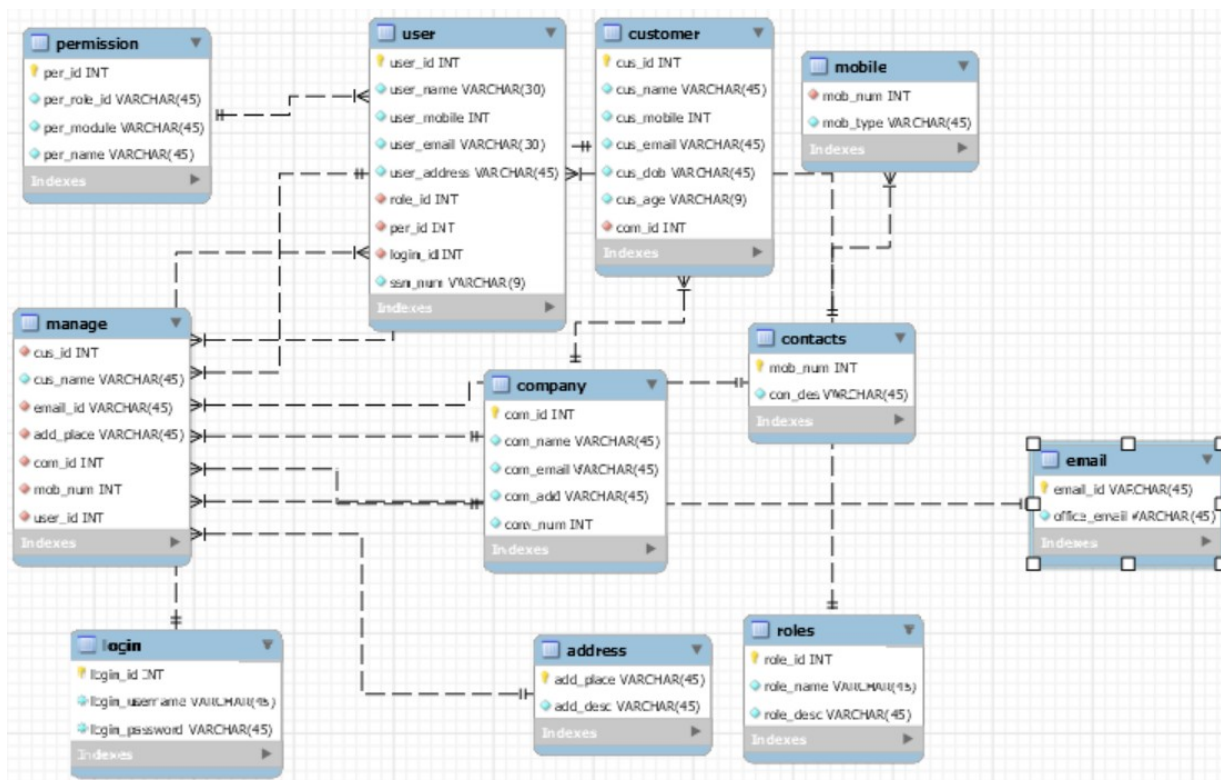
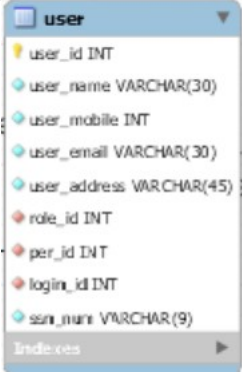


Figure 2: Enhanced Entity Relationship Diagram for CMS

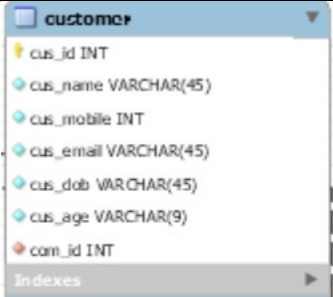
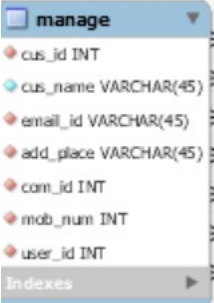
Database Description:

This Database is designed as per the EER model diagram which is shown above. All the tables and entities are created with the help SQL queries Containing Primary Keys, foreign keys and Null keys.

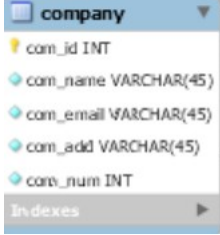
In all most all the tables Primary keys (Pkey) [4] and Foreign Keys (Fkey) [4] are defined which are having a unique value for each column.

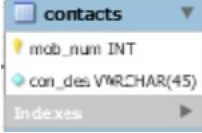

Table Name	Query	EER Model for Table	Description	Pkey	Fkey
user	<pre>CREATE TABLE `user` (user_id int NOT NULL, user_name varchar(30) NOT NULL, user_mobile INT NOT NULL, user_email varchar(30) NOT NULL, user_address varchar(45) NOT NULL , role_id INT NOT NULL, per_id INT NOT NULL, login_id int NOT NULL, ssn_num varchar(9) NOT NULL, PRIMARY KEY (`user_id`), FOREIGN KEY (role_id) REFERENCES ROLES(role_id), FOREIGN KEY (login_id) REFERENCES login(login_id), FOREIGN KEY (per_id) REFERENCES PERMISSION(per_id));</pre>		The purpose of User table is to manage the customer's data	Yes	Yes

Contact management system_Final Project Report_Wizards at work

customer	<pre>CREATE TABLE `customer` (cus_id int NOT NULL, cus_name varchar(45) NOT NULL, cus_mobile INT NOT NULL, cus_email varchar(45) NOT NULL, cus_dob varchar(45) NOT NULL, cus_age varchar(9) NOT NULL, com_id INT NOT NULL, PRIMARY KEY (`cus_id`), FOREIGN KEY (com_id) REFERENCES company(com_id));</pre>		The Purpose of customer table is to store the customer data.	Yes	Yes
manage	<pre>CREATE TABLE `manage` (cus_id int NOT NULL, cus_name varchar(45) NOT NULL, email_id varchar(45) NOT NULL, add_place varchar(45) NOT NULL, com_id INT NOT NULL, mob_num int NOT NULL, user_id INT NOT NULL, FOREIGN KEY (cus_id) REFERENCES customer(cus_id), FOREIGN KEY (email_id) REFERENCES email(email_id), FOREIGN KEY (add_place) REFERENCES address(add_place), FOREIGN KEY (com_id) REFERENCES company(com_id), FOREIGN KEY (com_id) REFERENCES company(com_id), FOREIGN KEY (mob_num) REFERENCES contacts(mob_num), FOREIGN KEY (user_id) REFERENCES user(user_id));</pre>		The purpose of manage table is for user to manage the customer data.	No	Yes

Contact management system_Final Project Report_Wizards at work

company	CREATE TABLE `company` (com_id int NOT NULL, com_name varchar(45) NOT NULL, com_email varchar(45) NOT NULL, com_add varchar(45) NOT NULL, com_num int NOT NULL, PRIMARY KEY (`com_id`));		The purpose of company table is to add or manipulate the customer's company data.	Yes	No
permission	CREATE TABLE `permission` (per_id INT NOT NULL, per_role_id varchar(45) NOT NULL, per_module varchar(45) NOT NULL, per_name varchar(45) NOT NULL, PRIMARY KEY (`per_id`));		The purpose of Permissions table to assign different permissions to end user.	Yes	No
login	CREATE TABLE `login` (login_id int NOT NULL, login_username varchar(45) NOT NULL, login_password varchar(45) NOT NULL, PRIMARY KEY (`login_id`));		The purpose of Login table is to store the login credentials like username and password etc	Yes	No
roles	CREATE TABLE `roles` (role_id INT NOT NULL, role_name varchar(45) NOT NULL , role_desc varchar(45) NOT NULL , PRIMARY KEY (`role_id`));		The purpose of roles table is to assign the different roles to user like admin, developer etc.	Yes	No
address	CREATE TABLE `address` (add_place varchar(45) NOT NULL, add_desc varchar(45) NOT NULL, PRIMARY KEY (`add_place`));		The purpose of address table is to add or manipulate the customer's address.	Yes	No
email	CREATE TABLE `email` (email_id varchar(45) NOT NULL, office_email varchar(45) NOT NULL, PRIMARY KEY (`email_id`));		The purpose of email table is to add or manipulate the customer's email.	Yes	No

	PRIMARY KEY (`email_id`));				
contacts	CREATE TABLE `contacts` (mob_num int NOT NULL, con_des varchar(45) NOT NULL, PRIMARY KEY (`mob_num`));		The purpose of contacts table is to add or manipulate the customer's contacts.	Yes	No
mobile	CREATE TABLE `contacts` (mob_num int NOT NULL, con_des varchar(45) NOT NULL, PRIMARY KEY (`mob_num`));		The purpose of mobile table is to add or manipulate the customer's contacts.	Yes	No

Loading data & performance enhancements:

Importing data:

Table Name	Query	Description
User	<pre> Insert into user(user_id,user_name,user_mobile,user_email,user_address,role_id,per_id,login_id,ssn_num)values (1,'Sandeep',2,'sandeep214@gmail.com','186, lincoln',3,1,91222,'564776'), (2,'Akhil',2,'akhndeep214@gmail.com','186, lincoln',3,1,91222,'564776'), (3,'Nikhil',2,'nikhil@gmail.com','186, lincoln',3,1,91222,'564776'), (4,'Chethan',4,'chethan14@gmail.com','186, lincoln',4,1,91222,'564776'); </pre>	Here we are inserting the values into the User table. Our application has different kinds of user and user will have different roles according to their permissions.
Login	<pre> insert into login(login_id,login_username,login_password) values('32354','sacreadheart','SHU@345'), ('90282','kites','Pru@345'), ('67533','university','Ak@6323'), ('63276','posst','mypass'), ('821629','kiytes','uni@1P'), ('91222','leosa','post%\$'), ('78632','lions','Peudjj@sj'), </pre>	Here we are inserting the data into the login table. In this table we have information like login ids, username, and password.

	('92329','school','Liqjs'), ('83728','hshes','Insjs'), ('656757','yrsyg','\$^uyt');	
Permission	Insert into permission(per_id, per_role_id, per_module, per_name) values (1,2,'ALL','Access to all modules/pages'),(2,3,'Limited','Access to only some pages'),(3,4,'No Permission','No Access to any modules'), (4,2,'hello','Access to all modules/pages'), (5,3,'Limited','Access to only some pages'), (7,8,'No Permission','No Access to any modules'), (9,2,'ALL','Access to all modules/pages'), (10,2,'Limited','Access to only some pages'), (11,4,'No Permission','No Access to any modules');	Here we are inserting the data into the permission table, and we have data like permission id, permission role, permission name. There will be different permissions for different users.

B. Manipulating the data:

In this section, we manipulated the data which exist in the tables with the help of ALTER and UPDATE commands.

Using ALTER command, we created a new column in the user table and updating the new column with some data with the help of UPDATE command.

Table Name: User

Before Manipulation of data:

user_id	user_name	user_mobile	user_email	user_address	role_id	per_id	login_id	ssn_num
1	Sandeep	2	sandeep214@gmail.com	186, lincoln	3	1	91222	564776
2	Akhil	2	akhndeep214@gmail.com	186, lincoln	3	1	91222	123
3	Nikhil	2	nikhil@gmail.com	186, lincoln	3	1	91222	564776
4	Chethan	4	chethan14@gmail.com	186, lincoln	4	1	91222	564776
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Query with Alter:

Here we are adding the new column to the user table.

Alter table user

add New_SSN varchar(10);

After Manipulation of data with Alter:

	user_id	user_name	user_mobile	user_email	user_address	role_id	per_id	login_id	ssn_num	New_SSN
▶	1	Sandeep	2	sandeep214@gmail.com	186, lincoln	3	1	91222	564776	NULL
	2	Akhil	2	akhndeep214@gmail.com	186, lincoln	3	1	91222	123	NULL
	3	Nikhil	2	nikhil@gmail.com	186, lincoln	3	1	91222	564776	NULL
	4	Chethan	4	chethan14@gmail.com	186, lincoln	4	1	91222	564776	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Query with Update:

Here we are assigning the value to the new column.

Update user

set New_SSN = '98273792'

where user_id =2;

After Manipulation of data with Update:

user_id	user_name	user_mobile	user_email	user_address	role_id	per_id	login_id	ssn_num	New_SSN
1	Sandeep	2	sandeep214@gmail.com	186, lincoln	3	1	91222	564776	NULL
2	Akhil	2	akhndeep214@gmail.com	186, lincoln	3	1	91222	123	98273792
3	Nikhil	2	nikhil@gmail.com	186, lincoln	3	1	91222	564776	NULL
4	Chethan	4	chethan14@gmail.com	186, lincoln	4	1	91222	564776	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Table Name: User

Before Manipulation of data:

role_id	role_name	role_desc
1	Admin	Adminstrator
2	User	Normal User
3	Super Admin	Person who have high level of access
4	Guest	Guest to the application
NULL	NULL	NULL

Query:

By using this command, we are renaming the role_desc to role_detail.

Alter table roles

rename column role_desc to role_detail;

After Manipulation of data:

role_id	role_name	role_detail
1	Admin	Adminstrator
2	User	Normal User
3	Super Admin	Person who have high level of access
4	Guest	Guest to the application
NULL	NULL	NULL

Query with update:

By using this command, we are changing the roles.

Update roles

set role_name = 'Admin', role_detail= 'Adminstrator'

where role_id =4;

After Manipulation of data with Update:

role_id	role_name	role_detail
1	Admin	Adminstrator
2	User	Normal User
3	Super Admin	Person who have high level of access
4	Admin	Adminstrator
NULL	NULL	NULL

C. Optimizing the data:

In the section of data optimization, we joined two different tables which are having the same column name (foreign key) with help of Inner Join and Group By command.

Using the count function, we counted the number of records presented for each user in the roles table and new column having the number of records.

Using Group by command, joined the existing table username and the newly created table number of roles.

User table:

user_id	user_name	user_mobile	user_email	user_address	role_id	per_id	login_id	ssn_num	New_SSN
1	Sandeep	2	sandeep214@gmail.com	186, lincoln	3	1	91222	564776	NULL
2	Akhil	2	akhndeep214@gmail.com	186, lincoln	3	1	91222	123	98273792
3	Nikhil	2	nikhil@gmail.com	186, lincoln	3	1	91222	564776	NULL
4	Chethan	4	chethan14@gmail.com	186, lincoln	4	1	91222	564776	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Roles table:

role_id	role_name	role_detail
1	Admin	Adminstrator
2	User	Normal User
3	Super Admin	Person who have high level of access
4	Admin	Adminstrator
NULL	NULL	NULL

Query:

Here we are evaluating by using the **GROUPBY** and **LEFT JOIN**. By running the below query, will get a table containing two rows user_name and NumberOfRoles. With the help of left join joined the two tables which are having the same column role_id in both tables roles and user.

SELECT user.user_name,COUNT(roles.role_id) AS NumberOfRoles FROM roles

LEFT JOIN user ON roles.role_id = user.role_id

GROUP BY user_name;

Evaluated results:

user_name	NumberOfRoles
NULL	2
Sandeep	1
Akhil	1
Nikhil	1
Chethan	1

Execution Plan:

Execution plans can tell you how a query will be executed, or how a query was executed. They are therefore the DBA's primary means of troubleshooting a poorly performing query. Rather than guess at why a given query is performing thousands of scans, putting your I/O through the roof, you can use the execution plan to identify the exact piece of SQL code that is causing the problem. For example, it may be scanning an entire table-worth of data when, with the proper index, it could simply backpack out only the rows you need. All this and more are displayed in the execution plan.

Below diagram represents the way how it joined the two different tables user and roles.

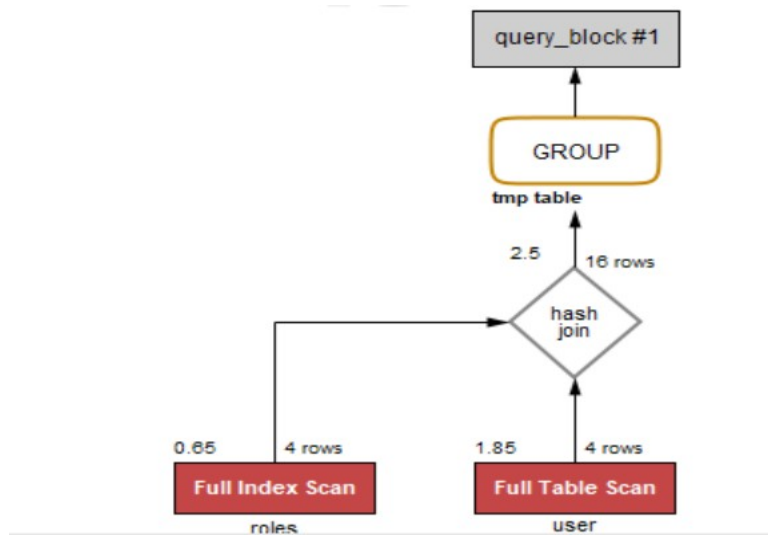


Figure3: Execution Plan

Query statistics:

The Query Stats SQL editor results tab (see the next two figures) uses Performance Schema data to gather key statistics collected for executed query, such as timing, temporary tables, indexes, joins, and more.

Checking the optimization by inserting the single instance and multiple instances.

Query:

Insert into roles (role_id, role_name, role_detail)

values

(5,'Admin','Administrator');

Execution time with one set of values:

The below figure shows execution time one set of values, we can clearly observe that timing as measured by the server for the execution 0.00.000.00032540 and table lock wait time 0:00:0.00011400 and timing as measured at client side for the

Execution 0:00:0.00000000.

By comparing these values with the values of execution time with multiple set of values we can clearly compare the optimization.

Query Statistics	
Timing (as measured at client side): Execution time: 0:00:0.00000000	Joins per Type: Full table scans (Select_scan): 1 Joins using table scans (Select_full_join): 0 Joins using range search (Select_full_range_join): 0 Joins with range checks (Select_range_check): 0 Joins using range (Select_range): 0
Timing (as measured by the server): Execution time: 0:00:0.00032540 Table lock wait time: 0:00:0.00011400	Sorting: Sorted rows (Sort_rows): 0 Sort merge passes (Sort_merge_passes): 0 Sorts with ranges (Sort_range): 0 Sorts with table scans (Sort_scan): 0
Errors: Had Errors: NO Warnings: 0	Index Usage: No Index used
Rows Processed: Rows affected: 0 Rows sent to client: 5 Rows examined: 5	Other Info: Event Id: 299 Thread Id: 51
Temporary Tables: Temporary disk tables created: 0 Temporary tables created: 0	

Execution time with multiple set of values:

Execution time with one set of values:

The below figure shows execution time one set of values, we can clearly observe that timing as measured by the server for the execution 0.00.000.00039910 and table lock wait time 0:00:0.00019200 and timing as measured at client side for the

Execution 0:00:0.00000000.

By comparing these values with the values of execution time with one set of values we can clearly compare the optimization. By this we can conclude that it is better to insert the multiple values than to insert single values at a time.

Query Statistics

Timing (as measured at client side):

Execution time: 0:00:0.00000000

Timing (as measured by the server):

Execution time: 0:00:0.00039910

Table lock wait time: 0:00:0.00019200

Errors:

Had Errors: NO

Warnings: 0

Rows Processed:

Rows affected: 0

Rows sent to client: 17

Rows examined: 17

Temporary Tables:

Temporary disk tables created: 0

Temporary tables created: 0

Joins per Type:

Full table scans (Select_scan): 1

Joins using table scans (Select_full_join): 0

Joins using range search (Select_full_range_join): 0

Joins with range checks (Select_range_check): 0

Joins using range (Select_range): 0

Sorting:

Sorted rows (Sort_rows): 0

Sort merge passes (Sort_merge_passes): 0

Sorts with ranges (Sort_range): 0

Sorts with table scans (Sort_scan): 0

Index Usage:

No Index used

Other Info:

Event Id: 333

Thread Id: 51

Graphical User Interface:

A. Login Page

Below figure illustrates the Login page of Contact Management System (CMS),

Which Contains the two login buttons for two respective user's admin and user.

Admin have the privileges to create a user, alter the details of user and delete the user from CMS.

In order to login to the CMS dashboard, user should be created by admin and provides the username and password to the end user.

Once the user gets the Username and password, CMS allows the user to change the password as the per the user requirement and user should be able to add the required information.

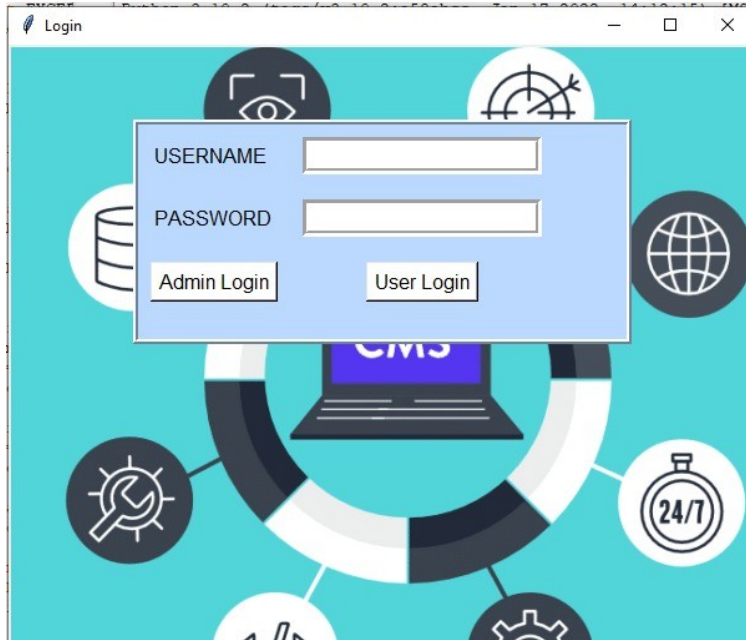


Figure 4 : CMS login Page, only the figure is taken from the source[6]

Code for Login page:

```
from AdminPage import *
from SearchBook import *
from tkinter import *
from PIL import ImageTk
from tkinter import messagebox
from UserPage import *

import pymysql
# Add your own database name and password here to reflect in the code
mypass = "S@nde780yepuri"
mydatabase="libpos"
print('before')
con =
pymysql.connect(host="localhost",user="root",password=mypass,database=mydatabase)
print('after')
cur = con.cursor()

root = Tk()
```

```
root.title("Login")
root.geometry("600x480")

def calladmin():
    usern = username_box.get()
    passw = password_box.get()
    if usern == "" or passw == "":
        messagebox.showinfo("Error", "All fields are required")
    else:
        try:
            cur.execute("select * from user2 where user_name=%s and user_password=
%s", (usern, passw))
            op = cur.fetchone()
            print(op[1])
            if op == None:
                messagebox.showinfo("Error", "Invalid Username or Password")
            else:
                root.destroy()
                mainmenu()

        except EXCEPTION as es:
            messagebox.showerror("Error"f"Error Due to: {str(es)}")

def callUser():
    usern = username_box.get()
    passw = password_box.get()
    if usern == "" or passw == "":
        messagebox.showinfo("Error", "All fields are required")
    else:
        try:
            cur.execute("select * from user2 where user_name=%s and user_password=
%s", (usern, passw))
            op = cur.fetchone()
            # print(op)
            if op == None:
                messagebox.showinfo("Error", "Invalid Username or Password")
```

```
        else:
            root.destroy()
            mainmenu1(op[0])

    except EXCEPTION as es:
        messagebox.showerror("Error"f"Error Due to: {str(es)}")

bg = ImageTk.PhotoImage(file="CMS-Home-Image.jpg")
bg_image = Label(root, image=bg).grid()

del_frame = Frame(root, bd=4, relief=RIDGE, bg="#BBD8FF")
del_frame.place(x=100, y=60, width=400, height=180)

# Label for username and password
username = Label(del_frame, text="USERNAME", bg="#BBD8FF", fg="black", font=15)
username.grid(row=12, column=0, sticky=W, padx=10, pady=10)
password = Label(del_frame, text="PASSWORD", bg="#BBD8FF", fg="black", font=15)
password.grid(row=14, column=0, sticky=W, padx=10, pady=10)

# Text boxes for username and password
global username_box
username_box = Entry(del_frame, font=15, bd=5, relief=GROOVE)
username_box.grid(row=12, column=1, pady=10, padx=10, sticky="w")

global password_box
password_box = Entry(del_frame, font=15, bd=5, relief=GROOVE)
password_box.grid(row=14, column=1, pady=10, padx=10, sticky="w")

usern = username_box.get()
passw = password_box.get()
```

```
# Login Button for student and admin
adminlogin = Button(del_frame, text="Admin Login", bg="white", fg="black", font=15,
command=calladmin)
adminlogin.grid(row=20, column=0, pady=10, padx=10)

studentlogin = Button(del_frame, text="User Login",bg="white", fg="black", font=15,
command=callUser)
studentlogin.grid(row=20, column=1, pady=10, padx=10)

root.mainloop()
```

B. Main Menu Page:

The below figure illustrates Admin main menu page containing two buttons to create a user and delete a user namely Add a new user and Delete a User.

After successful admin login, the admin main menu page is displayed where admin is allowed to create a new user which results in creating a new user in contact management system (CMS) and have the access to delete a user which results in deleting the existing user.

The user got deleted from contact management system (CMS) is no longer is able to login to the CMS dashboard.

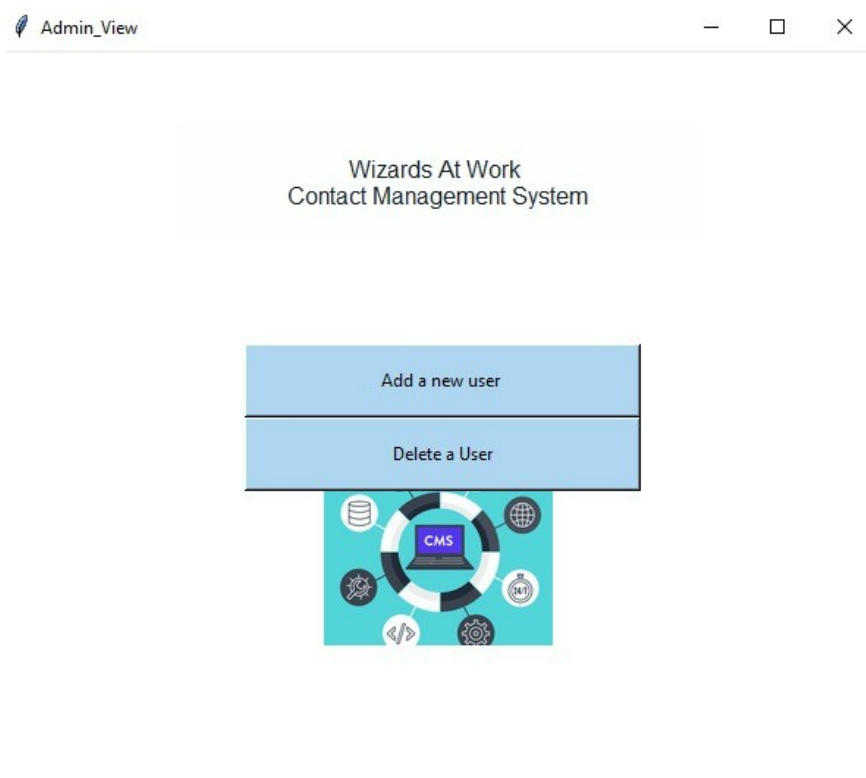


Figure 5: Admin Main menu Page

Code for Admin Main menu Page:

```
from tkinter import *
from PIL import ImageTk,Image
import pymysql
from tkinter import messagebox
from AddUser import *
from DeleteBook import *
from SearchBook import *
from UpdateBook import *
from showAllRecord import *
# Add your own database name and password here to reflect in the code
mypass = "S@nde780yepuri"
mydatabase="libpos"

con =
pymysql.connect(host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()
```

```
def mainmenu():
    root = Tk()
    root.title("Admin_View")
    root.minsize(width=400, height=400)
    root.geometry("600x500")

    # Take n greater than 0.25 and less than 5
    same = True
    n = 0.25

    # Adding a background image
    background_image = Image.open("CMS-Home-Image.jpg")
    [imageSizeWidth, imageSizeHeight] = background_image.size

    newImageSizeWidth = int(imageSizeWidth * n)
    if same:
        newImageSizeHeight = int(imageSizeHeight * n)
    else:
        newImageSizeHeight = int(imageSizeHeight / n)

    background_image = background_image.resize((newImageSizeWidth,
newImageSizeHeight), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(background_image)

    Canvas1 = Canvas(root)

    Canvas1.create_image(300, 340, image=img)
    Canvas1.config(bg="white", width=newImageSizeWidth, height=newImageSizeHeight)
    Canvas1.pack(expand=True, fill=BOTH)

    headingFrame1 = Frame(root, bg="#FDFEFE", bd=5)
    headingFrame1.place(relx=0.2, rely=0.1, relwidth=0.6, relheight=0.16)
```

```
headingLabel = Label(headingFrame1, text="Wizards At Work \nContact Management
System", bg='#FDFFEFE', fg='#17202A',
                    font='40')
headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)

btn1 = Button(root, text="Add a new user ", bg='#AED6F1', fg='black',
command=lambda: addUser(0))
btn1.place(relx=0.28, rely=0.4, relwidth=0.45, relheight=0.1)

btn2 = Button(root, text="Delete a User", bg='#AED6F1', fg='black',
command=delete)
btn2.place(relx=0.28, rely=0.5, relwidth=0.45, relheight=0.1)

root.mainloop()
```

The below figure illustrates User main menu page containing three buttons to create a user and delete a user namely View your contacts, add a contact, Delete contact.

After successful user login, the user main menu page is displayed where user is allowed to view a user which results in viewing the contacts in contact management system (CMS) and have the access to add a contact which results in adding the contacts.

User has a access to delete the contact which results in deleting the existing contacts.

The user got deleted from contact management system (CMS) is no longer can login to the CMS dashboard.

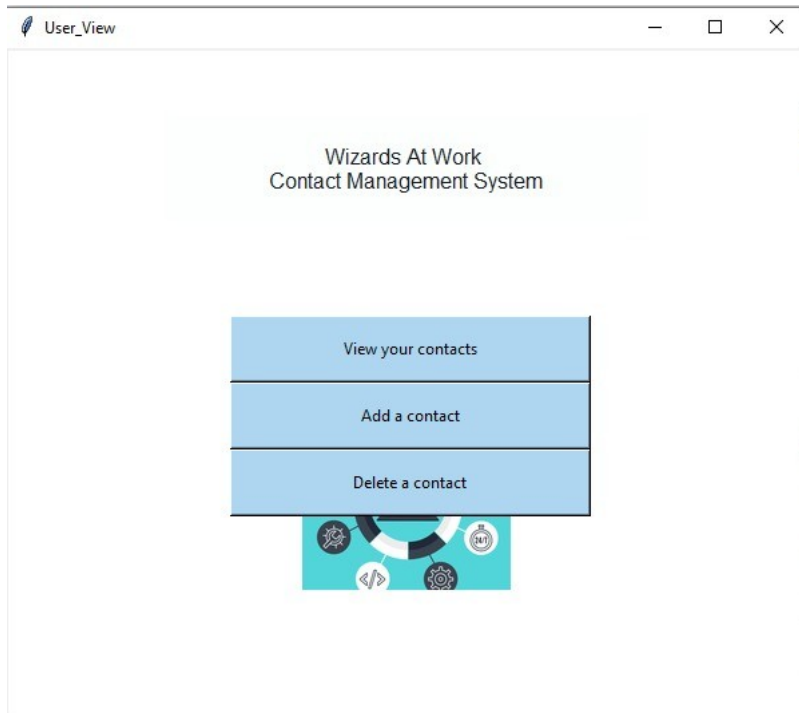


Figure 6: User Main menu Page

Code for User Main menu Page:

```
from tkinter import *
from PIL import ImageTk,Image
import pymysql
from tkinter import messagebox
from AddUser import *
from DeleteBook import *
from SearchBook import *
from UpdateBook import *
from showAllRecord import *

# Add your own database name and password here to reflect in the code
mypass = "S@nde780yepuri"
mydatabase="libpos"

con =
pymysql.connect(host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()
```

```
def mainmenu1(userId):
    root = Tk()
    root.title("User_View")
    root.minsize(width=400, height=400)
    root.geometry("600x500")

    # Take n greater than 0.25 and less than 5
    same = True
    n = 0.25

    # Adding a background image
    background_image = Image.open("CMS-Home-Image.jpg")
    [imageSizeWidth, imageSizeHeight] = background_image.size

    newImageSizeWidth = int(imageSizeWidth * n)
    if same:
        newImageSizeHeight = int(imageSizeHeight * n)
    else:
        newImageSizeHeight = int(imageSizeHeight / n)

    background_image = background_image.resize((newImageSizeWidth,
newImageSizeHeight), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(background_image)

    Canvas1 = Canvas(root)

    Canvas1.create_image(300, 340, image=img)
    Canvas1.config(bg="white", width=newImageSizeWidth, height=newImageSizeHeight)
    Canvas1.pack(expand=True, fill=BOTH)

    headingFrame1 = Frame(root, bg="#FDFEFE", bd=5)
    headingFrame1.place(relx=0.2, rely=0.1, relwidth=0.6, relheight=0.16)
```

```
headingLabel = Label(headingFrame1, text="Wizards At Work \nContact Management
System", bg='#FDFEFE', fg='#17202A',
                    font='40')
headingLabel.place(relx=0, rely=0, relwidth=1, relheight=1)

btn1 = Button(root, text="Add a new contact ", bg='#AED6F1', fg='black',
command=lambda: addUser(userId))
btn1.place(relx=0.28, rely=0.4, relwidth=0.45, relheight=0.1)

btn1 = Button(root, text="show all contacts", bg='#AED6F1', fg='black',
command=lambda: showAll(userId))
btn1.place(relx=0.28, rely=0.5, relwidth=0.45, relheight=0.1)

btn2 = Button(root, text="Delete a contact", bg='#AED6F1', fg='black',
command=delete)
btn2.place(relx=0.28, rely=0.6, relwidth=0.45, relheight=0.1)

root.mainloop()
```

Action Pages:

The below figure illustrates action page to add a user with admin in contact management system (CMS).

This page contains some textboxes where it's required to enter the user information like username, user mobile, user email, user address and user password.

Mainly there are two buttons Submit and Quit presented in this page. Once the admin user enters the valid details of the end user and clicks on submit will create a user and records the user details in the database.

The purpose of using the quit button is to terminate the user creation. Once admin clicks on the Quit button admin will navigated back to the admin main menu page.

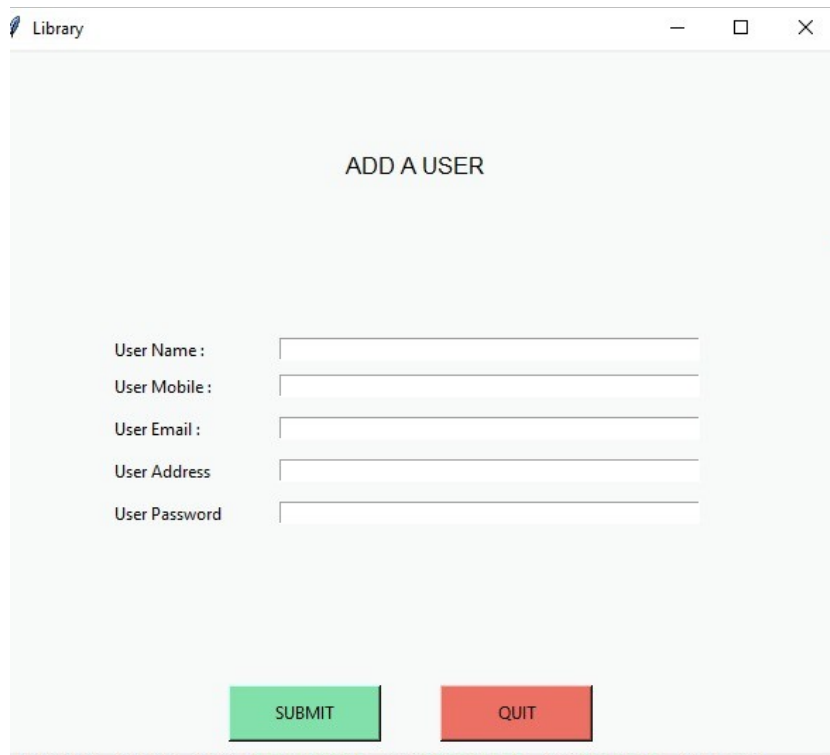


Figure 7: Action page to add a user

Code for Action Page:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql

def userRegister(userId):
    userName1 = userName.get()
    mobileNum1 = mobileNum.get()
    emailAddress1 = emailAddress.get()
    userAddress1 = userAddress.get()
    userPassword1 = userPassword.get()

    if userName1 != "" and mobileNum1 != "" and emailAddress1 != "" and
    userAddress1 != "" and userPassword1 != "" :
        insert_Data = "Insert into user2 (user_name,user_mobile,user_email,
```

Contact management system_Final Project Report_Wizards at work

```
user_address,user_Password, role_id) value (%s,%s,%s,%s,%s,%s)"
    value = (userName1, mobileNum1, emailAddress1, userAddress1, userPassword1,
7)
    cur.execute(insert_Data, value)
    con.commit()
    messagebox.showinfo("Info", "Record Inserted")
    cur.execute("select * from user2 where user_name=%s", userName1)
    op = cur.fetchone()
    insert_junctionTable = "Insert into userContacts2(parent_user_id,
contact_user_id) value (%s, %s)"
    value = (userId, op[0])
    cur.execute(insert_junctionTable, value)
    con.commit()
    messagebox.showinfo("Info", "Junction table Inserted")
else:
    messagebox.showinfo("Info", "Enter Valid Records")

print(bookDescription)
print(bookTitle)
print(bookCategory)
print(bookAuthorName)
print(bookPublication)
print(bookPrice)
print(bookISBN)

root.destroy()

def addBook(args):

    print(args)
    global userName, mobileNum, emailAddress, userAddress, userPassword, bookInfo6,
bookInfo7, Canvas1, con, cur, bookTable, root

    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")
```


Contact management system_Final Project Report_Wizards at work

```
# Add your own database name and password here to reflect in the code
mypass = "S@nde780yepuri"
mydatabase = "libpos"

con =
pymysql.connect(host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()

Canvas1 = Canvas(root)

Canvas1.config(bg="#F8F9F9")
Canvas1.pack(expand=True,fill=BOTH)

headingFrame1 = Frame(root,bg="#F8F9F9",bd=5)
headingFrame1.place(relx=0.25,relx=0.1,relwidth=0.5,relheight=0.13)

headingLabel = Label(headingFrame1, text="ADD A USER",bg='#F8F9F9', fg='black',
font=15)
headingLabel.place(relx=0,relx=0, relwidth=1, relheight=1)

labelFrame = Frame(root,bg='#F8F9F9')
labelFrame.place(relx=0.1,relx=0.4,relwidth=0.8,relheight=0.4)

# Book ISBN
lb1 = Label(labelFrame,text="User Name : ",bg='#F8F9F9', fg='black')
lb1.place(relx=0.05,relx=0.02, relheight=0.08)

userName = Entry(labelFrame)
userName.place(relx=0.3,relx=0.02, relwidth=0.62, relheight=0.08)

# Title
lb2 = Label(labelFrame,text="User Mobile : ",bg='#F8F9F9', fg='black')
lb2.place(relx=0.05,relx=0.15, relheight=0.08)

mobileNum = Entry(labelFrame)
mobileNum.place(relx=0.3,relx=0.15, relwidth=0.62, relheight=0.08)
```

```
# Book Author
lb3 = Label(labelFrame,text="User Email : ",bg='#F8F9F9', fg='black')
lb3.place(relx=0.05,rely=0.30, relheight=0.08)

emailAddress = Entry(labelFrame)
emailAddress.place(relx=0.3,rely=0.30, relwidth=0.62, relheight=0.08)

# Book Category
lb4 = Label(labelFrame,text="User Address",bg='#F8F9F9', fg='black')
lb4.place(relx=0.05,rely=0.45, relheight=0.08)

userAddress = Entry(labelFrame)
userAddress.place(relx=0.3,rely=0.45, relwidth=0.62, relheight=0.08)

# Book Category
lb4 = Label(labelFrame,text="User Password",bg='#F8F9F9', fg='black')
lb4.place(relx=0.05,rely=0.60, relheight=0.08)

userPassword = Entry(labelFrame)
userPassword.place(relx=0.3,rely=0.60, relwidth=0.62, relheight=0.08)

#Submit Button
SubmitBtn = Button(root,text="SUBMIT",bg='#82E0AA', fg='black',command = lambda:
userRegister(args))
SubmitBtn.place(relx=0.28,rely=0.9, relwidth=0.18,relheight=0.08)

quitBtn = Button(root,text="QUIT",bg='#EC7063', fg='black', command=root.destroy)
quitBtn.place(relx=0.53,rely=0.9, relwidth=0.18,relheight=0.08)

root.mainloop()
```

Delete User:

The below figure illustrates action page to delete a user with admin in contact management system (CMS).

This page contains some textboxes where it's required to enter the username.

Mainly there are two buttons Delete and Quit presented in this page. Once the admin user enters the username of the end user and clicks on submit will delete from the database.

The purpose of using the quit button is to terminate the user deletion. Once admin clicks on the Quit button admin will navigated back to the admin main menu page.

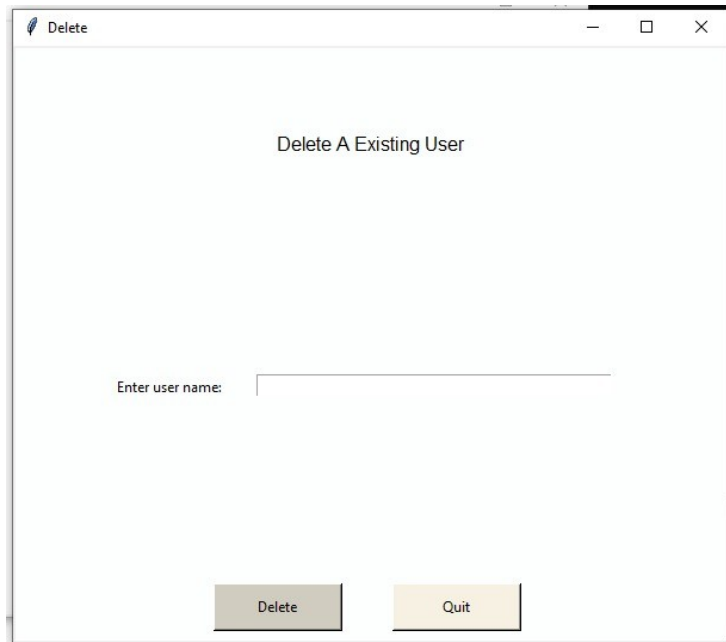


Figure 8: Action page to delete user

Code for Action Page:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql

# Add your own database name and password here to reflect in the code
mypass = "S@nde780yepuri"
mydatabase="libpos"

con =
pymysql.connect(host="localhost",user="root",password=mypass,database=mydatabase)
```

```
cur = con.cursor()
```

```
def deleteUser():
```

```
    userName = userInfo.get()
```

```
    getUserId = "select * from User1 where user_name = '%s'" % userName
```

```
    deleteByUserName = "Delete from User2 where user_name = '%s'" % userName
```

```
    try:
```

```
        cur.execute(getUserId)
```

```
        result = cur.fetchone()
```

```
        deleteJuntionTableRecords = "Delete from userContacts2 where parent_user_id = '%s'" % result[0]
```

```
        cur.execute(deleteJuntionTableRecords)
```

```
        cur.execute(deleteByUserName)
```

```
        con.commit()
```

```
        messagebox.showinfo("Information", "Record Deleted")
```

```
    except:
```

```
        messagebox.showinfo("Please check Book ID")
```

```
print(deleteByUserName)
```

```
root.destroy()
```

```
def delete():
```

```
    global userInfo,Canvas1,con,cur,root
```

```
    root = Tk()
```

```
    root.title("Delete")
```

```
    root.minsize(width=400,height=400)
```

```
root.geometry("600x500")

Canvas1 = Canvas(root)

Canvas1.config(bg="#FDFEFE")
Canvas1.pack(expand=True,fill=BOTH)

headingFrame1 = Frame(root,bg="#FDFEFE",bd=5)
headingFrame1.place(relx=0.25,relx=0.1,relwidth=0.5,relheight=0.13)

headingLabel = Label(headingFrame1, text="Delete A Existing User", bg='#FDFEFE',
fg='black', font=15)
headingLabel.place(relx=0,relx=0, relwidth=1, relheight=1)

labelFrame = Frame(root,bg='#FDFEFE')
labelFrame.place(relx=0.1,relx=0.3,relwidth=0.8,relheight=0.5)

# Book ID to Delete
lb1 = Label(labelFrame,text="Enter user name: ", bg='#FDFEFE', fg='black')
lb1.place(relx=0.05,relx=0.5)

userInfo = Entry(labelFrame)
userInfo.place(relx=0.3,relx=0.5, relwidth=0.62)

#Submit Button
SubmitBtn = Button(root,text="Delete",bg='#d1ccc0',
fg='black',command=deleteUser)
SubmitBtn.place(relx=0.28,relx=0.9, relwidth=0.18,relheight=0.08)

quitBtn = Button(root,text="Quit",bg='#f7f1e3', fg='black', command=root.destroy)
quitBtn.place(relx=0.53,relx=0.9, relwidth=0.18,relheight=0.08)

root.mainloop()
```

Conclusion and Future work:

From this Project Contact Management System (CMS) we have learned how to work with databases, normalization, entity relationships, sql commands for creating,

inserting and manipulating the tables, database optimization, database connection, graphical user interface implementation with python and database security. All these concepts helped us to implement an end-to-end user interface for contact management system.

There is a scope in extending the contact management system by making it more dynamic and security can be increased and there is also scope for implementing the more functionalities for the user and admin where they can perform additional operations.

GitHub Repository Address

GitHub Repository address for the Contact Management System (CMS) is the following: <https://github.com/prudhviakhil619/contact-management-system>

References

[1] Contact Management System: <https://monday.com/blog/project-management/contact-management-database/>

[2] Entity Relationship Diagram: [https://en.wikipedia.org/wiki/Entity](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model)
relationship_model

[3] Enhanced Entity Relationship Diagram: <https://www.differencebetween.com/difference-between-er-and-vs-eer-diagram/>

[4] Primary Key and Foreign Key: <https://www.geeksforgeeks.org/difference-between-primary-key-and-foreign-key/>

[5] Optimization: https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby1

[6] CMS logo: <https://soffront.com/blog/features-free-contact-management-software/>

[7] Warehouse Management System: <https://github.com/RezaSadeghiWSU/Warehouse-Management-System>

