CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Current rental systems have the major disadvantage of not allowing owners to keep track of tenant details easily. This can also generate a rent receipt that exempts the tenants from taxes. Many tenants tend to pay rent online as it's more convenient as it can be done from the convenience of sitting at home. Upon payment of the rent, they will receive their rent receipt. This is also error-free.

1.2 OBJECTIVES

- Once developed, the application will provide services to rent collection systems.
- In addition to keeping track of income, owners must make sure to stay on top of their tax obligations. This would help overcome that and provide a database to store the files.
- A person who pays rent for a furnished/unfurnished apartment can claim a deduction for the rent under Section 80(GG) of the I-T Act, provided he has not received HRA as part of his salary by providing Form 10B. This project could be exempt from some of their taxes, which is a wonderful idea.

1.3 EXPECTED OUTCOMES

- Keep track of all tenant details.
- Helps view or update any tenant detail.
- Prints rent receipt.

1.4 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

a. A Personal Computer
b. Minimum of 2gb RAM
c. 64 bit Operating system
d. Windows 7 or above

Software requirements:

1. MySQL
2. Pycharm

CHAPTER 2

FUNDAMENTALS OF PYTHON

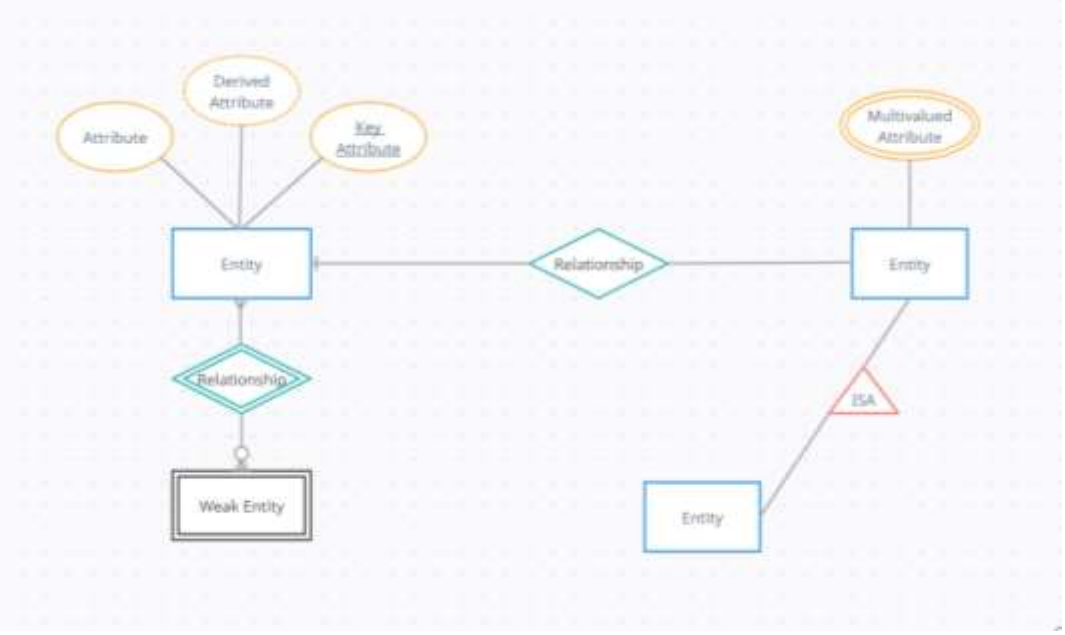
2.1 INTRODUCTION TO PYTHON

Python is a commonly and extensively used general-purpose, high-level programming language. In Python, the highest level of programming is interpreted, object-oriented, and has a dynamic semantics. The built-in data structures and dynamic typing make it ideal for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. In addition to being easy to learn, Python's syntax emphasizes readability, so it reduces the cost of maintaining a program. Python supports modules and packages, which encourage program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

2.2 TYPES OF DATA MODELS

The Data Model gives us an idea of how the final system would look after it has been fully implemented. It specifies the data items as well as the relationships between them. In a database management system, data models are often used to show how data is connected, stored, accessed, and changed. We portray the information using a set of symbols and language so that members of an organization may understand and comprehend it and then communicate.

Though there are other data models in use today, the Relational model is the most used. Aside from the relational model, there are a variety of different data models that we shall discuss in-depth in this article.


Derived Attribute
Attribute
Key Attribute
Multivalued Attribute
Entity
Relationship
Entity
Relationship
ISA
Weak Entity
Entity

2.2.1 ENTITY-RELATIONSHIP (ER) MODEL

The Entity-Relationship (ER) Model is an attractive high level conceptual data model. It has an entity which may be an object with a physical existence like a particular car, house, person or employee or it may be an object with a coneptual existence like an organization, a profession, or a university course. Each enity has attribtes—the defnite properies that characterize it. For exmple, a student entty my be described by the student's name, age, address, USN etc.

2.2.2 RELATIONAL MODEL

This is the **most widely accepted data model.**In this model, the database is represented as a **collection of relations in the form of rows and columns of a two-dimensional table.** Each row is known as a tuple (a tuple contains all the data for an individual record) while each column represents an attribute.

2.2.3 OBJECT ORIENTED DATA MODEL

As suggested by its name, the object-oriented data model is a combination of object-oriented programming, and relational data model. In this data model, the data and their relationship are represented in a single structure which is known as an object. Since data is stored as objects we can easily store **audio, video, images,** etc in the database which was very difficult and inconvenient to do in the relational model. As shown in the image below two objects are connected with each other through links.

CHAPTER 3

DESIGN

3.1 E-R DIAGRAM OF THE PROJECT

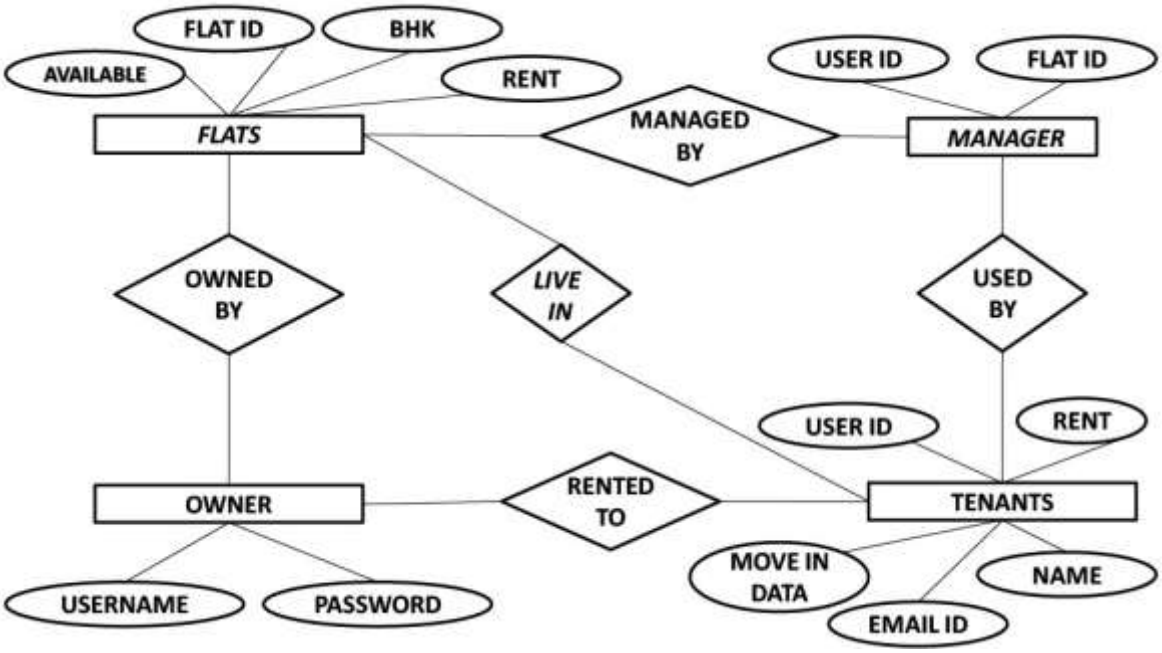


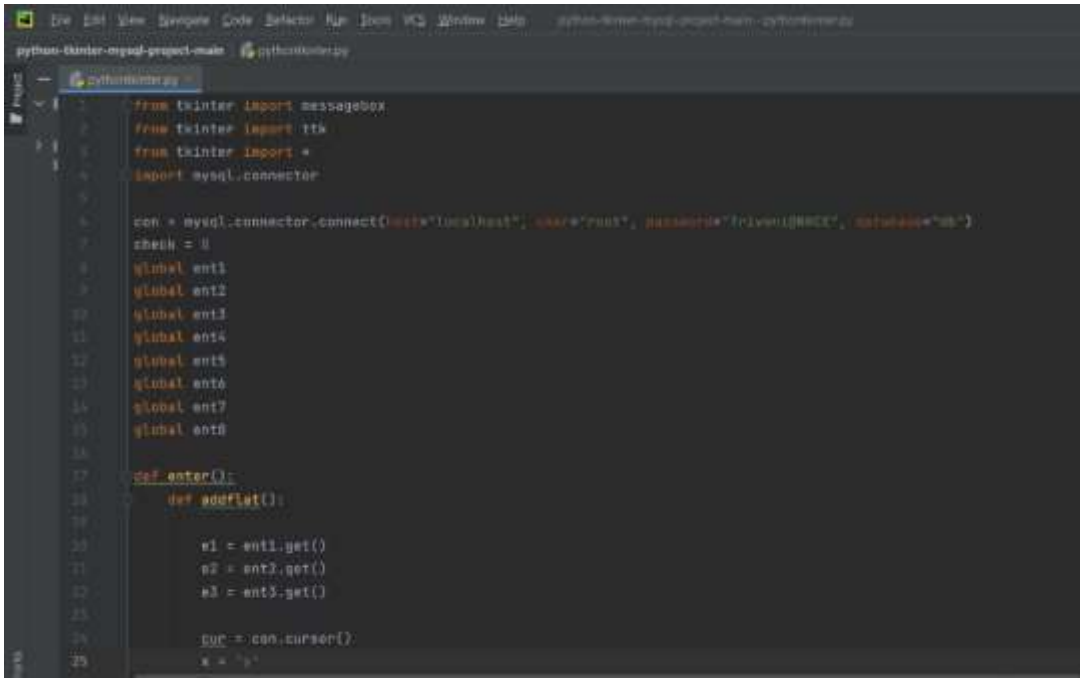
Figure 3.1: E-R Model of Ez-rentalz

This is an Entity relational diagram with 4 entities namely flats, manages, owner, tenants. They are related to each other with different relationships. Flats are owned by the owner, flats are managed by the manager. Owner rents the flats to the tenant, tenants live in the flat. Each entity has its own table in the database.

Each entity has its own attributes. Flats have attributes like available, flat_id, bhk, rent where flat_id is the primary key and the foreign key that links it to the manager table. Owner has attributes such as username and password. Tenants have attributes like user_id, email id, name, rent, move in date. The user_id is the primary key and the foreign key that links it to the manager table.

CHAPTER 4

IMPLEMENTATION IN PYTHON



```
from tkinter import messagebox
from tkinter import ttk
from tkinter import *
import mysql.connector

con = mysql.connector.connect(host="localhost", user="root", password="friyeni@BRCE", database="db")
check = 0
global ent1
global ent2
global ent3
global ent4
global ent5
global ent6
global ent7
global ent8

def enter():
    def addflat():

        e1 = ent1.get()
        e2 = ent2.get()
        e3 = ent3.get()

        cur = con.cursor()
        x = '>'
```

```
        x = 'y'
        cur.execute('insert into flats(flat_id,bhk,rent,available) values(%s,%s,%s,%s)', (e1, e2, e3, x))
        con.commit()
        flatid.delete(0, END)
        bhk.delete(0, END)
        price.delete(0, END)

        messagebox.showinfo('success', 'flat inserted!')

    # frameland()

    # Function for deleting a flat
    def delflat():
        e1 = ent1.get()
        e2 = ent2.get()
        e3 = ent3.get()
        cur = con.cursor()
        x = 'y'
        cur.execute('delete from flats where flat_id=%s and available=%s', (e1, x))
        con.commit()
        flatid.delete(0, END)
        bhk.delete(0, END)
        price.delete(0, END)
        messagebox.showinfo('success', 'flat deleted!')
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help python-tkinter-mysql-project-main - pythontkinter.py

python-tkinter-mysql-project-main > pythontkinter.py

pythontkinter.py

```
    # creating the main window
    root = Toplevel()
    root.geometry("500x500")
    root.title("EZRENTALZ")

    # creating a notebook for tabs(owner,book,return,receipt)
    notebook = ttk.Notebook(root)
    notebook.pack()

    # creating 4 frames
    frame1 = Frame(notebook, width=500, height=500)
    frame2 = Frame(notebook, width=500, height=500)
    frame3 = Frame(notebook, width=500, height=500)
    frame4 = Frame(notebook, width=500, height=500)

    frame1.pack(fill="both", expand=1)
    frame2.pack(fill="both", expand=1)
    frame3.pack(fill="both", expand=1)
    frame4.pack(fill="both", expand=1)

    notebook.add(frame1, text="owner")
    notebook.add(frame2, text="book")
    notebook.add(frame3, text="return")
    notebook.add(frame4, text="receipt")
```

```python
ent1 = IntVar()
ent2 = StringVar()
ent3 = IntVar()

# adding and droping flat window
lap = Label(frame1, text="WELCOME TO EZ-RENTALS").pack()

lab1 = Label(frame1, text="FLATID").place(x=150, y=50)
flatid = Entry(frame1, textvariable=ent1)
flatid.place(x=200, y=50)

lab2 = Label(frame1, text="BHK").place(x=150, y=80)
bhk = Entry(frame1, textvariable=ent2)
bhk.place(x=200, y=80)

lab3 = Label(frame1, text="RENT").place(x=150, y=110)
price = Entry(frame1, textvariable=ent3)
price.place(x=200, y=110)

but1 = Button(frame1, text="ADD FLAT", command=addflat).place(x=200, y=200)
but2 = Button(frame1, text="DROP FLAT", command=delflat).place(x=300, y=200)

# butshow=Button(frame1,text="GO TO BOOK",command=show).place(x=250,y=400)
```

```python
# def frame2add():
# displaying available flats
lapf = Label(frame2, text="AVAILABLE FLATS").pack()

treescroll = Frame(frame2)
treescroll.pack()

scroll = Scrollbar(frame2)
scroll.pack(side=RIGHT, fill=Y)
tree = ttk.Treeview(frame2, yscrollcommand=scroll.set)  # xscrollcommand=scroll.set)
scroll.config(command=tree.yview)

# vsb=Scrollbar(root,orient="vertical")
# vsb.configure(command=tree.yview)
# tree.configure(yscrollcommand=vsb.set)
# vsb.pack(fill=Y,side=RIGHT)

tree['show'] = 'headings'
s = ttk.Style(root)
s.theme_use('clam')
tree.pack()

tree['columns'] = ("flat_id", "bhk", "rent","available")
cur = con.cursor()
```

```python
cur.execute("select flat_id,bhk,rent,available from flats ")
# con.commit()
# cur.close()
tree.column("flat_id", width=50, minwidth=50, anchor=CENTER)
tree.column("bhk", width=100, minwidth=100, anchor=CENTER)
tree.column("rent", width=150, minwidth=150, anchor=CENTER)
tree.column("available", width=90, minwidth=90, anchor=CENTER)


# headings
tree.heading("flat_id", text="flatid", anchor=CENTER)
tree.heading("bhk", text="bhk", anchor=CENTER)
tree.heading("rent", text="rent", anchor=CENTER)
tree.heading("available", text="available", anchor=CENTER)


i = 0
for row in cur:
    tree.insert('', i, text='', values=(row[0], row[1], row[2], row[3]))
    i += 1

con.commit()

# vsb=Scrollbar(root,orient="vertical")
```

```python
ent4 = IntVar()
ent5 = StringVar()
ent6 = StringVar()
ent7 = StringVar()
ent8 = StringVar()

# Function to book a flat
def bookflat():
    a = ent4.get()
    b = ent5.get()
    c = ent6.get()
    d = ent7.get()
    e = ent8.get()
    cur = con.cursor()
    cur.execute("insert into user(name,email,startdate,lot) values(%s,%s,%s,%s)", (b, c, d, e))
    con.commit()
    cur = con.cursor()
    cur.execute("select user_id from users where name=%s and email=%s", (b, c))
    ac = 0
    for r in cur:
        ac = r[0]
    con.commit()
    cur = con.cursor()
```

```
            a = 'n'
            cur.execute("insert into manages(user_id,flat_id) values(%s,%s)", (ac, a))
            a = 'n'
            cur.execute("update flats set available=%s where flat_id=%s", (x, a))
            con.commit()
            # cur.execute("update manages set user_id = (select user_id from users natural join flats where name=%s and email=%s)",(n,e))
            # con.commit()
            # cur.execute("delete from users where user_id not in (select  user_id from manages)")
            # con.commit()
            # messagebox.showinfo("success","flat booked")
            # cur.execute("delete from manages where manages.flat_id=%s in (select flat_id from flats where available='n')",(a))
            # con.commit()
            cur.execute("delete from manages where flat_id not in (select flat_id from flats)")
            con.commit()
            cur.execute("delete from users where user_id not in (select user_id from manages)")
            con.commit()
            messagebox.showinfo("success", "flat booked")
            flatbook.delete(0, END)
            flatname.delete(0, END)
            flatmail.delete(0, END)
            flatdate.delete(0, END)
            flatloc.delete(0, END)

        # under development!!!
        def returnflat():
```

```
        def returnflat():
            root1 = Toplevel()
            root1.geometry("600x400")
            root1.title("RETURN FLAT")
            lab = Label(root1, text="FLATS TO BE RETURNED").pack()
            tree1 = ttk.Treeview(root1)
            tree1["show"] = "headings"
            s = ttk.Style(root1)
            s.theme_use("clam")
            tree1["columns"] = ("flatid", "userid", "name", "email")
            tree1.column("flatid", width=70, minwidth=70, anchor=CENTER)
            tree1.column("userid", width=70, minwidth=70, anchor=CENTER)
            tree1.column("name", width=150, minwidth=150, anchor=CENTER)
            tree1.column("email", width=150, minwidth=150, anchor=CENTER)

            tree1.heading("flatid", text="flatid", anchor=CENTER)
            tree1.heading("userid", text="userid", anchor=CENTER)
            tree1.heading("name", text="name", anchor=CENTER)
            tree1.heading("email", text="email", anchor=CENTER)

            cur = con.cursor()
            cur.execute(
                "select flat_id,user_id,name,email from manages natural join users natural join flats where available='n'")
            i = 0
            for row in cur:
```

```
        for com in cur:
            tree1.insert('', i, text='', values=(row[0], row[1], row[2], row[3]))
            i += 1
        con.commit()

        def reflat():
            s1 = bt1.get()
            s2 = bt2.get()
            s3 = bt3.get()
            cur = con.cursor()
            y = 'y'
            cur.execute("update flats set available=%s where flat_id=%s", (y, s1))
            con.commit()
            cur.execute("delete from manages where flat_id=%s and user_id=%s", (s1, s2))
            con.commit()
            messagebox.showinfo("flat returned", "thank u come again!")
            b1.delete(0, END)
            b2.delete(0, END)
            b3.delete(0, END)

        bt1 = IntVar()
        bt2 = StringVar()
        bt3 = StringVar()

        ll = Label(root1, text="FLATID").place(x=210, y=270)
```


```
        for com in cur:
            tree1.insert('', i, text='', values=(row[0], row[1], row[2], row[3]))
            i += 1
        con.commit()

        def reflat():
            s1 = bt1.get()
            s2 = bt2.get()
            s3 = bt3.get()
            cur = con.cursor()
            y = 'y'
            cur.execute("update flats set available=%s where flat_id=%s", (y, s1))
            con.commit()
            cur.execute("delete from manages where flat_id=%s and user_id=%s", (s1, s2))
            con.commit()
            messagebox.showinfo("flat returned", "thank u come again!")
            b1.delete(0, END)
            b2.delete(0, END)
            b3.delete(0, END)

        bt1 = IntVar()
        bt2 = StringVar()
        bt3 = StringVar()

        ll = Label(root1, text="FLATID").place(x=210, y=270)
```

```
        l1 = Label(root1, text="FLATID").place(x=210, y=270)
        b1 = Entry(root1, textvariable=bt1)
        b1.place(x=260, y=270)

        l2 = Label(root1, text="DOORID").place(x=210, y=300)
        b2 = Entry(root1, textvariable=bt2)
        b2.place(x=260, y=300)

        l3 = Label(root1, text="NAME").place(x=210, y=330)
        b3 = Entry(root1, textvariable=bt3)
        b3.place(x=260, y=330)

        bn = Button(root1, text="RETURN FLAT", command=reflat).place(x=250, y=370)

        vsb = ttk.Scrollbar(root1, orient="vertical")
        vsb.configure(command=tree1.yview)
        tree1.configure(yscrollcommand=vsb.set)
        vsb.pack(fill=Y, side=RIGHT)

        tree1.pack()
        root1.mainloop()

    lab4 = Label(frame2, text="FLATID").place(x=150, y=270)
    flatbook = Entry(frame2, textvariable=ent4)
    flatbook.place(x=200, y=270)
```

```
    lab5 = Label(frame2, text="NAME").place(x=150, y=300)
    flatname = Entry(frame2, textvariable=ent5)
    flatname.place(x=200, y=300)

    lab6 = Label(frame2, text="EMAIL").place(x=150, y=330)
    flatmail = Entry(frame2, textvariable=ent6)
    flatmail.place(x=200, y=330)

    lab7 = Label(frame2, text="DATE").place(x=150, y=360)
    flatdate = Entry(frame2, textvariable=ent7)
    flatdate.place(x=200, y=360)

    lab8 = Label(frame2, text="DIST").place(x=150, y=390)
    flatloc = Entry(frame2, textvariable=ent8)
    flatloc.place(x=200, y=390)

    but1 = Button(frame2, text="BOOK FLAT", command=bookflat).place(x=200, y=420)
    bin = Button(frame2, text="RETURN", command=enter).place(x=200, y=450)

    lz = Label(frame3, text="RETURN FLAT PORTAL").place(x=190, y=10)
    but4 = Button(frame3, text="GO TO RETURN FLAT", command=returnflat).place(x=200, y=100)

    def rec():
        g = uname.get()
```

```
python-tkinter-mysql-project-main   pythontkinter.py
pythontkinter.py
        def rec():
            a = uname.get()
            b = rent.get()
            c = month.get()
            mm = " Rent Received from " +a+ " \n for an amount of Rs." +b+ " \n for the  month " +c+ " against \n " \
            "the settlement of amount agreed \n by Trivedi. This amount is \n referring to the confirmation of \n " \
            "compliance of rent(s) agreement. \n \n TRIVEDI A "
            la = Label(frame6, text= mm, font=("Calibri","15"), justify=LEFT, wraplength=2, relief= "solid").place(x=10, y=290)


        la = Label(frame4, text="RENT RECEIPT ").place(x=10, y=10)
        la = Label(frame4, text="name").place(x=50, y=40)
        la = Label(frame4, text="month").place(x=200, y=40)
        la = Label(frame4, text="rent").place(x=150, y=40)

        uname = StringVar()
        rent = StringVar()
        month = StringVar()

        u = Entry(frame4, textvariable=uname)
        u.place(x=50, y=80)
        r = Entry(frame4, textvariable=rent)
        r.place(x=150, y=80)
        m = Entry(frame4, textvariable=month)
        m.place(x=200, y=80)
```

```
python-tkinter-mysql-project-main   pythontkinter.py
pythontkinter.py
        but7 = Button(frame4, text="Print", command=rec).place(x=300, y=150)

        # tree.pack()
        frame2.mainloop()
        root.mainloop()

    def subatt():
        c = 0
        username = a.get()
        password = b.get()
        cur.execute("select *from owner")
        for i in cur:
            if i[0] == username and i[1] == password:
                global check
                c = 1
                check = 1
        if c == 1:
            messagebox.showinfo("valid!", "welcome!")
            enter()
        else:
            messagebox.showwarning("not valid!!", "try again!")


    r = Tk()
    r.title(" RE RENTAL ")
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help   python-tkinter-mysql-project-main - pythontkinter.py
python-tkinter-mysql-project-main   pythontkinter.py
pythontkinter.py
            messagebox.showwarning("not valid!!", "try again!")


    r = Tk()
    r.title(" RZ-RENTALZ ")
    r.geometry("500x500")
    s = ttk.Style(r)
    s.theme_use("clam")
    cur = con.cursor()
    a = StringVar()
    b = StringVar()
    t = Label(r, text="SIGNUP/LOGIN").pack()
    user = Label(r, text="USERNAME").place(x=150, y=150)
    g = Entry(r, textvariable=a)
    g.place(x=230, y=150)
    passw = Label(r, text="PASSWORD").place(x=150, y=180)
    h = Entry(r, textvariable=b)
    h.place(x=230, y=180)
    bo = Button(r, text="SUBMIT", command=submit)
    bo.place(x=250, y=220)
    # b5=Button(r,text="REFRESH",command=enter)
    # b5.place(x=250,y=250)
    con.commit()
    r.mainloop()
```

CHAPTER 5

IMPLEMENTATION IN MY SQL

```
MySQL 8.0 Command Line Client - Unicode

mysql> use db;
Database changed
mysql> show tables;
+---------------+
| Tables_in_db |
+---------------+
| flats         |
| manages       |
| owner         |
| users         |
+---------------+
4 rows in set (0.04 sec)

mysql> select * from flats;
+---------+------+-------+-----------+
| flat_id | bhk  | rent  | available |
+---------+------+-------+-----------+
|     101 | 1bhk | 25000 | y         |
|     102 | 2bhk | 45000 | y         |
|     103 | 1bhk | 20000 | n         |
|     104 | 3bhk | 60000 | y         |
|     201 | 1bhk | 25000 | n         |
|     202 | 2bhk | 50000 | n         |
|     203 | 1bhk | 20000 | y         |
|     204 | 3bhk | 55000 | n         |
+---------+------+-------+-----------+
8 rows in set (0.01 sec)
```

Figure 5.1: MySQL Implementation-1

```
MySQL 8.0 Command Line Client - Unicode
mysql> select * from manages;
+---------+---------+
| user_id | flat_id |
+---------+---------+
|     508 |     103 |
|     509 |     201 |
|     510 |     202 |
|     511 |     204 |
+---------+---------+
4 rows in set (0.00 sec)

mysql> select * from users;
+---------+--------+------------------+------------+-------+
| user_id | name   | email            | startdate  | loc   |
+---------+--------+------------------+------------+-------+
|     508 | vinay  | vinay@gmail.com  | 2023-01-03 | 25000 |
|     509 | swetha | swetha@gmail.com | 2022-12-12 | 25000 |
|     510 | chinty | chinty@gmail.com | 2022-11-11 | 49000 |
|     511 | teju   | tej@yahoo.com    | 2022-06-06 | 60000 |
|     512 | veena  | veena@gmail.com  | 2023-01-01 | 45000 |
+---------+--------+------------------+------------+-------+
5 rows in set (0.00 sec)

mysql> select * from owner;
+----------+----------+
| username | password |
+----------+----------+
| triveni  | 123      |
+----------+----------+
1 row in set (0.01 sec)
```

Figure 5.2: MySQL Implementation-2

CHAPTER 6

RESULTS

6.1. OUTPUT

Figure 6.1: Login

Figure 6.2: Add/Drop a flat

Figure 6.3: Book a flat

Figure 6.4: Availability changes

Figure 6.5: Return a flat

Figure 6.6: Rent Receipt

CHAPTER 7

CONCLUSION

The mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report.

As this sector continues to grow, we still lack the technology where the owner can easily keep track of tenant details and immediately produce a rent receipt. A rent receipt is considered most helpful as it helps the tenant exempt themselves from unwanted taxes. Taxes are paid by every citizen to the government for various reasons. They are useful to keep the country's stability balanced. But also, all of us want as much money we earn or save for ourselves. An individual paying rent for a furnished/unfurnished accommodation can claim the deduction for the rent paid under Section 80(GG) of the I-T Act, provided he is not paid HRA as a part of his salary by furnishing Form 10B. This project would be a great idea to exempt a small amount of their taxes.