Project report

on

CAREER PORTAL

Submitted by:

SARAVANA KUMAR B

CHAPTER 1

INTRODUCTION

1.1 Introduction

Python uses Tkinter (Tk=Tool kit; inter=interface) to create GUI (Graphical User Interface). GUI helps to create a display interface in windows which is being used to take input from users and displaying output. Inserting and accessing data is happening with the help of backend through SQLite3 connection.

The create table command has been implemented in Python from which the database has been invoked to create the tables as well as the insert queries inserts the values into that table. Applications such as Applying a job, showing details of all the available jobs, viewing specific jobs and viewing application status, etc. have been done in Python as a frontend which is processed in backend with the help of SQL in SQLite database.

The project contains entire information about the applicant and employer. In the SQLite four tables (Applicant, Employer, Job, Status) have been created in the backend. All table has its own attributes and are related to each other through various participation and relationship which is described briefly below in coming chapters.

1.2 Course Objectives

- To learn how to design and program Python applications.
- To learn how to build and package Python modules for reusability.
- To learn how to design Object oriented programs.
- To learn how to use class inheritance in Python for reusability.
- To learn how to use exception handling in Python applications for error handling.
- To make better understanding of lists, tuples, dictionaries in Python programs.
- To understand why Python is a useful scripting language for developers.

1.3 Problem Definition

- Create a GUI based application for facilitating the recruitment process using a database.
- Store the records in a database and access it through the GUI.
- Assist the employer by optimizing the recruitment process and increasing the quality of candidates. With a recruitment system in place, an employer can ensure that candidate applications are processed quickly and accurately during the hiring process.
- Make the recruitment process as easy as possible for the candidate using a User-friendly interface. A good recruitment system is not just there to serve the employer, but also the candidate.

- Provide an innovative, cost efficient and effective recruitment system.
- Increase the quality and quantity of applicants applying for vacancies.

1.4 Outcomes of Project Work

- The project mainly focusses on providing an efficient and productive network between the Recruiters and the Applicants in the form of this application portal.
- Effectiveness and enhancement of the recruitment process.
- Dramatically reduces paper work and administrative work.
- Streamlines the recruitment process and reduces the corporate's overall recruitment cycle.
- Reduces the operational overheads of the company as well as ensures the accessibility of various opportunities of the Applicant.
- Provides a 2-Way Network between the recruiter and the applicant and thus makes the process of Job Recruitment easier and productive
- Delivers the best possible solution for a recruiting organization as well as an applicant.

CHAPTER 2

REQUIREMENTS AND DESIGN

2.1 Hardware Requirements:

- Processor: Intel core i3
- Ram: at least 1gb
- Hard Disk: 10gb or higher
- Mouse & Keyboard

2.2 Software Requirements:

- Operating System: Windows 7 or later
- Platform: Python IDE or Jupiter Notebook or Pycharm
- Database Software: SQLite3

CHAPTER 3

DATA MODELS AND ER MODELS

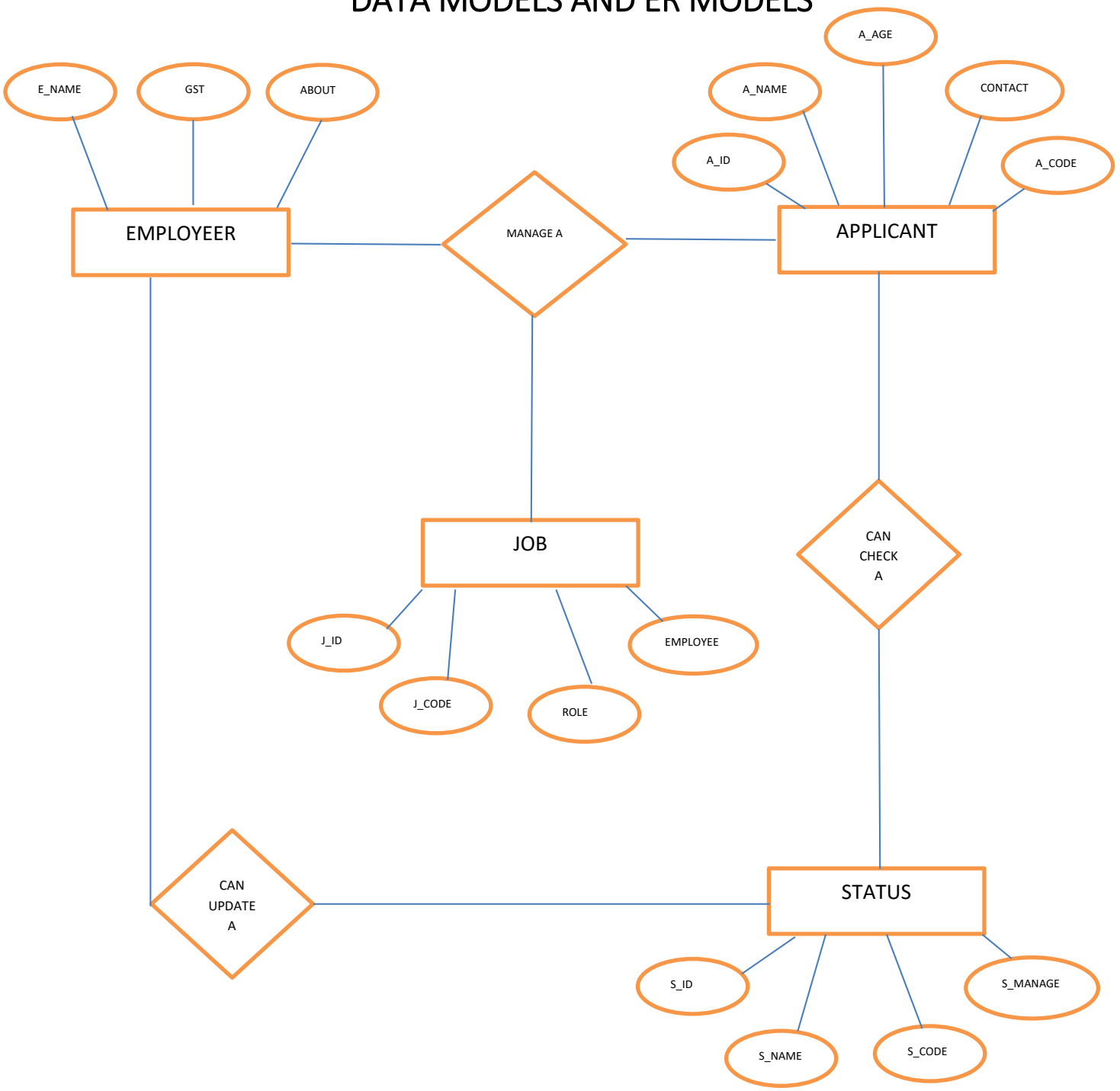

Fig 3.1 ER diagram

3.1 Entity and Attributes:

Entity type is collection of entities which consists of various attributes which may be of various data types and can have various constraints. While selecting an entity we must always care of two things, does that selected entity have enough members and also enough attributes. If the entity satisfies both the conditions, then such entities are called good entity but if the entity fails to satisfy one of these conditions then such entities are considered to be bad entities. Attributes for each entity should have a proper data type assigned to it and may or may not have constraints.

- Applicant (A_id, A_name, A_age, A_contact, A_code)
- Employeer (E_id, E_name, Gst, About)
- Job (J_id, J_code, Salary)
- Status (S_Id, S_name, S_code, message)

3.2 Keys

Keys is a constraint used while defining attributes in a table. Keys is used to identify a row in a table. Keys plays a vital role in finding the relation between two tables. It helps you uniquely identify a row in a table by combination of one or more columns in that table. There are various keys which has various properties, one of them which is widely used is Primary key. Primary key is a unique identification of a table that is used while combining tables and it can never be NULL. In a single table more than one column can be primary key. When this primary key column is used in other tables then that becomes Foreign key and its values can be NULL. Both primary and foreign key plays important role in determining relation between two tables. Unique key is also one of them which ensures that the particular column has unique values.

3.2.1 Primary key:

- A_id
- E_id
- E_id
- S_id

3.3 Relationship and Participation:

- The EMPLOYEER entity has attributes like E_name, gst, about
  E_name is the primary key of this entity
- The APPLICANT entity has attributes like A_id, A_name, age, contact, A_code
  A_code is the primary key of this entity
- The JOB entity has attributes like J_id, J_code, role, J_emp
  J_code is the primary key of this entity
- The Status entity has attributes like S_id, S_name, S_code, S_manage
  S_code is the primary key of this entity

3.4 Problem Statement :

- The project mainly focusses on providing an efficient and productive network between the Recruiters and the Applicants in the form of this application portal.
- Effectiveness and enhancement of the recruitment process.
- Dramatically reduces paper work and administrative work.
- Streamlines the recruitment process and reduces the corporate's overall recruitment cycle.
- Reduces the operational overheads of the company as well as ensures the accessibility of various opportunities of the Applicant.
- Provides a 2-Way Network between the recruiter and the applicant and thus makes the process of Job Recruitment easier and productive
- Delivers the best possible solution for a recruiting organization as well as an applicant.

CHAPTER 4

RELATIONAL MODEL CONCEPTS AND SCHEMA

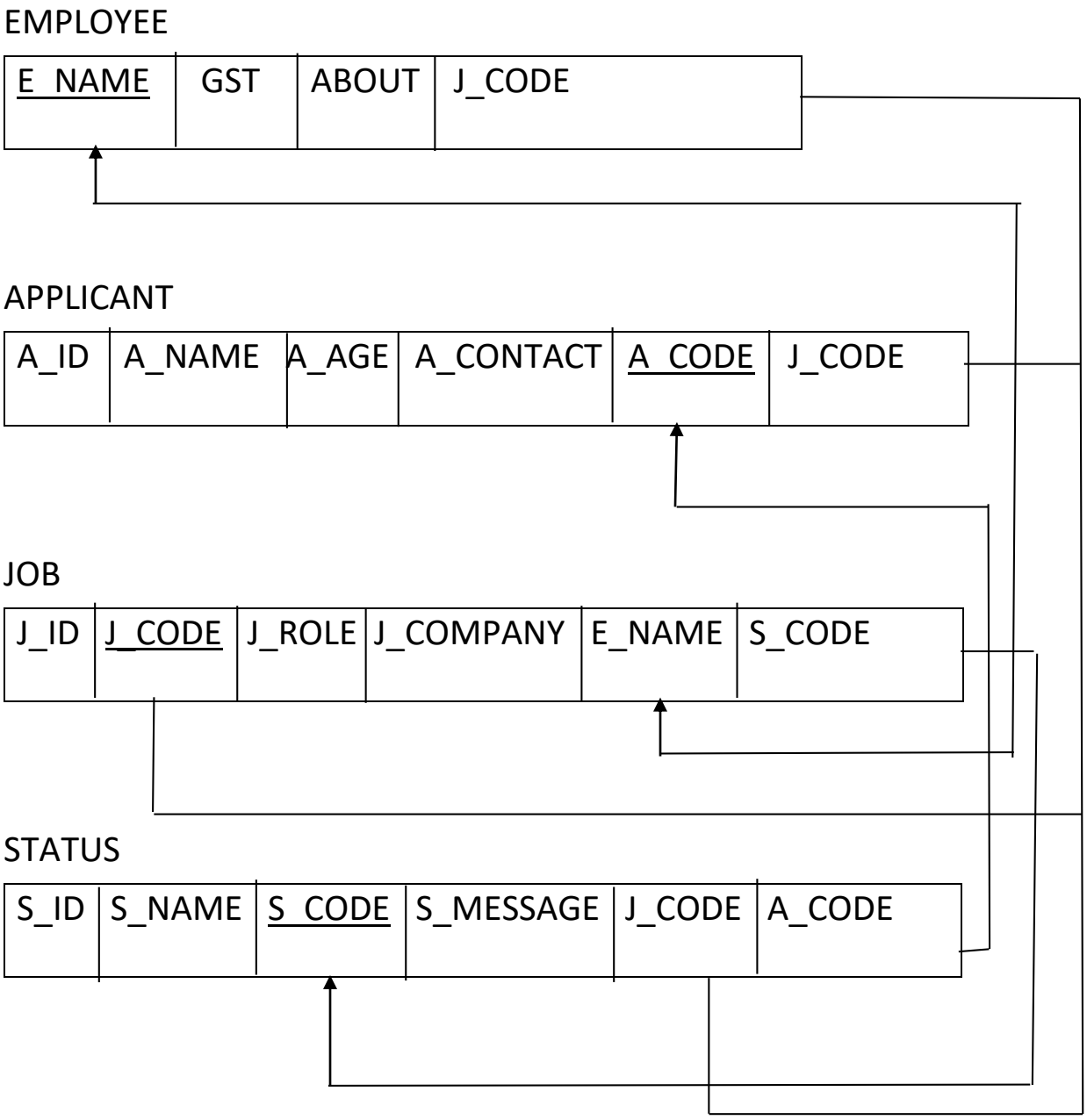

Fig 4.2 Relational Schema

CHAPTER 5

SQL

SQL stands for Structured Query Language. It's an standard computer language for managing the relational database and manipulation of data. It is used to do all the operation in database like creation of schema, table, inserting, updating, deleting and retrieving rows. SQL is used by various database management system like: MySQL, oracle, vertica, Sybase, etc.
There are various SQL commands which are listed below:

5.1 DDL COMMANDS

DDL stands for Data Definition Language. These are the commands used for schema or the table definition manipulation but not for data. There are various DDL commands which are listed below:

5.1.1 Create

This command is used to create a table and objects like view, triggers, assertions, etc. in database.
Syntax: CREATE TABLE <table_name> VALUES (column1, column2,….);

e.g.: ('CREATE TABLE IF NOT EXISTS Job(Id INTEGER PRIMARY KEY, Code VARCHAR NOT NULL, Role VARCHAR NOT NULL, Employer VARCHAR NOT NULL) ')

5.1.2 Drop

This command is used to delete table or the objects from the database.
Syntax: DROP TABLE <table_name>;
e.g.: DROP TABLE Job;

5.1.3 Alter

This command is used to alter the table structure like table definitions, constraints, delete columns, add columns, etc.
Syntax: ALTER TABLE <table_name> <option> <commands>;
e.g.: ALTER TABLE job DROP COLUMN PWD;

5.1.4 Truncate:

This command is used to delete the contents of the table including all the spaces.
Syntax: TRUNCATE TABLE <table_name>;
e.g.: TRUNCATE TABLE JOB;

5.2 DQL COMMANDS

DQL stands for Data Query Language for performing database queries.

5.2.1 Select

This command is used to retrieve the data from the database. It is one of the widely used command and is complex.

Syntax: SELECT <column_name> FROM <table_name> WHERE <condition>;

e.g.: ('SELECT * FROM Job WHERE Code = ?';

5.3 DML commands

DML stands for Data Manipulation Language and is used to manipulate the rows in the table. The rows in the table can be updated, deleted, inserted, etc.

5.3.1 Insert:

This command is used to insert the tuples into the table.

Syntax: INSERT INTO <table_name> <column_name> VALUES <column_values>;

e.g.: ('INSERT INTO Employer(Name, GST, About) VALUES('SK','rr5432','ssss');

5.3.2 Update:

This command is used to update the existing values of the tuple. Variable values can be updated using this command.

Syntax: UPDATE <table_name> SET <column_name>=<values> WHERE <condition>;

e.g.: ("""UPDATE Status SET Message = ? WHERE Name = ? AND Code = ? """

5.4 TCL command

TCL stands for Transaction Control Language which deals with the transaction within the database.

5.4.1 COMMIT

This command is used to commit the transaction so that the previous transaction will be successfully saved into the database. Once commit is done it is not possible to rollback.

Syntax: COMMIT;

5.4.2 ROLLBACK

This command is used to rollback/ undo the transaction if any error occurs.

Syntax: ROLLBACK;

CHAPTER 6

PYTHON FEATURES

6.1 Broad Standard Library

Python has huge collection of defined library which makes very easy to code in python. It's library is portable and compatible with all platforms like Macintosh, UNIX, and Windows. You donot have to write your own code for each and every thing as it provides rich sets of modules and functions. It has various libraries for web browsing, regular expressions, etc.

6.2 Interpreted Language

Python is one of the Interpreted Language as its code is executed line by line at a time. It is not required to compile our code like in other languages like java, c++, etc. which makes it easier to debug our code. The source code of python is converted into an immediate form called byte code.

6.3 Support for GUI Programming

Python provides various modules like PyQt, Tkinter, wxPython through which user can created Graphical User Interface for mobile applications. The most popular for creating graphical apps using python is PyQt5. Tkinter also provides all of the required options to create a beautiful user interface even Gaffer is made importing Tkinter module. This user interface can be connected to the backend using any one of the DBMS also supported by python, makes it more beautiful.

6.4 Object Oriented Programming Language

Python is an object oriented programming language which include the concept of class and object. It support all OOPs concepts like inheritance, data abstraction, polymorphism, encapsulation etc.

6.5 Scalable and Extendable

Python provides a better structure and support for large programs than shell scripting. You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

CHAPTER 7

TKINTER WIDGETS

7.1 Frame

A frame is a container which associates other widgets. It is mainly used for grouping and organizing widgets. A bunch of labels, entry, buttons, etc. can be added into the frame and moving frame alone moves other widgets too. Various options like: bg, bd, cursor, relief, width, etc. can be used to configure frame.
Syntax: Frame(window_name, options)

7.2 Label

A label is a widget which is used to display non-editable text. Label infact is also used to display images using Photo Image module. The most commonly used label is with 'text' configuration option and can change this at any time. Label makes use of many options like fg, bg, font, width, height, etc.
Syntax: Label(window_name, options)
We can add an image into label as below:
project_img=PhotoImage(file="path with file name")
project_img_label=ttk.Label (captainDashboard, image=project_img)

7.3 Entry

A entry is a single line text field user can use to type anything. It's mostly used in log in form for retrieving username and password. It has a special property to hide/ encrypt the text typed by user by using "show='*' "option which replaces each and every letter with the specified symbol/letter (in this case every letter typed by user is encrypted with '*').
Syntax: Entry(window_name, options)

7.4 Button

Button is one of the widely used widget among all in GUI with Tkinter. It is a functional widget that is clickable and on click it performs some action defined in the command option. It is used for linking two functions. They can display text or images same as labels, but also have a whole range of new options used to control their behavior.
Syntax: Button(window_name, options)

7.5 Listbox

Listbox displays a list of contents which a user interact with and user can accept any number of times. It looks like a column of a tables that displays values in various rows. It provide option to browse, select multiple, select single through selectmode option. It also offers other variety of options like: bg, fg, font, height, width, highlightcolor, etc.

Syntax: Listbox(window_name, options)

7.6 ScrolledText

This widget provides the feature of multiple line input field with scrollbar wherein user can type multiple lines of text. This very much useful for typing paragraphs, letter, essays, etc. It also supports various options like: height, width, etc.
Syntax: scrolledtext.ScrolledText(window_name, options)

CHAPTER 8

IMPLEMENTATION

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import sqlite3
import sys
import os
```

Fig 8.1 Importing required Python modules

```
class MainPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        label = tk.Label(self, text="MAIN MENU", font = HEADING_FONT)
        option1 = tk.Label(self, text="- APPLICANT PORTAL", font = HEADING_FONT)
        option2 = tk.Label(self, text="- EMPLOYER PORTAL", font = HEADING_FONT)


        label.grid(row=0, columnspan= 2,padx = 40, pady=40)
        option1.grid(row=1,column=1,padx = 40, pady=40)
        option2.grid(row=2,column=1,padx = 40, pady=40)

        button1 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ApplicantPage))
        button1.grid(row=1, column=2, sticky = "es",padx = 40, pady=40 )
        button2 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(EmployerPage))
        button2.grid(row=2, column=2, sticky = "es",padx = 40, pady=40 )
```

Fig 8.2 MAINFRAME

```
def create_job_table():
    c.execute('CREATE TABLE IF NOT EXISTS Job(Id INTEGER PRIMARY KEY, Code VARCHAR NOT NULL, Role VARCHAR NOT NULL, Employer VARCHAR NOT NULL) ')

def create_employer_table():
    c.execute('CREATE TABLE IF NOT EXISTS Employer(Id INTEGER PRIMARY KEY, Name VARCHAR NOT NULL, GST VARCHAR NOT NULL, About VARCHAR NOT NULL) ')

def create_applicant_table():
    c.execute('CREATE TABLE IF NOT EXISTS Applicant(Id INTEGER PRIMARY KEY, Name VARCHAR NOT NULL, Age INTEGER NOT NULL, Contact VARCHAR NOT NULL, Code VARCHAR NOT NULL) ')

def create_status_table():
    c.execute('CREATE TABLE IF NOT EXISTS Status(Id INTEGER PRIMARY KEY, Name VARCHAR NOT NULL, Code VARCHAR NOT NULL, Message VARCHAR NOT NULL) ')
```

Fig 8.3 Create Table

- create_employer_table()
- create_status_table()
- create_job_table()
- create_applicant_table()

```
def insert_applicant( name, age, contact, code):
    c.execute('INSERT INTO Applicant(Name, Age, Contact, Code) VALUES(?,?,?,?)',(name, age, contact, code,))
    conn.commit()
    messagebox.showinfo(title = "Success", message = "Job Application to Employer successfuly sent")

def insert_employer(name, gst, about):
    c.execute('INSERT INTO Employer(Name, GST, About) VALUES(?,?,?)',(name, gst, about,))
    conn.commit()
    messagebox.showinfo(title = "Success", message = "New Employer was successfully added")
    restart_program()

def insert_job(code, role, employer):
    c.execute('INSERT INTO Job(Code, Role, Employer) VALUES(?,?,?)',(code, role, employer,))
    conn.commit()
    messagebox.showinfo(title = "Success", message = "New Job was successfully added")
    restart_program()

def insert_status(name, code, message):
    c.execute('INSERT INTO Status(Name, Code, Message) VALUES(?,?,?)',(name, code, message,))
    conn.commit()
    restart_program()
```

Fig 8.4 Inserting

- Inserting employer
- Inserting status
- Inserting job
- Inserting applicant

```
class ApplicantPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        label = tk.Label(self, text="APPLICANT PORTAL", font = HEADING_FONT)
        option1 = tk.Label(self, text="- APPLY FOR A JOB", font = LARGE_FONT)
        option2 = tk.Label(self, text="- VIEW ALL JOBS", font = LARGE_FONT)
        option3 = tk.Label(self, text="- VIEW SPECIFIC JOBS", font = LARGE_FONT)
        option4 = tk.Label(self, text="- VIEW APPLICATION STATUS", font = LARGE_FONT)

        label.grid(row=1, column=1,columnspan=2, padx = 20, pady=20)
        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)
        option4.grid(row=5,column=1,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(MainPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button2 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ApplyPage))
        button2.grid(row=2, column = 2, sticky = "es",padx = 20, pady=20 )
        button3 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ViewAllJobPage))
        button3.grid(row=3, column = 2, sticky = "es",padx = 20, pady=20 )
        button4 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ViewSpecificJobPage))
        button4.grid(row=4, column = 2, sticky = "es",padx = 20, pady=20 )
        button5 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ViewStatusPage))
        button5.grid(row=5, column = 2, sticky = "es",padx = 20, pady=20 )
```

Fig 8.5 Applicant Page

VIEW ALL JOBS or SEARCH SPECIFIC JOB by CODE, EMPLOYER or ROLE you can either view all jobs posted or view specific jobs according to employer or Job Unique ID or Role that the job offers from here you can note down the job unique ID which you want to apply for

```
class ApplyPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        def wrapper(name, age, contact, code):
            c.execute('SELECT * FROM Job WHERE Code = ?',(code.get(),))
            info = c.fetchall()
            if len(info) == 0:
                messagebox.showinfo(title = "Caution", message = "No such job exists")
                controller.show_frame(ApplyPage)
            else:
                insert_applicant( name.get(), age.get(), contact.get(), code.get())
                message =  "Not Yet Viewed"
                insert_status(name.get(), code.get(), message)

        option1 = tk.Label(self, text="- NAME ", font = LARGE_FONT)
        option2 = tk.Label(self, text="- AGE", font = LARGE_FONT)
        option3 = tk.Label(self, text="- CONTACT", font = LARGE_FONT)
        option4 = tk.Label(self, text="- CODE", font = LARGE_FONT)

        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)
        option4.grid(row=5,column=1,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=2,column=2,padx = 20, pady=20)
        entry2 = tk.Entry(self)
        entry2.grid(row=3,column=2,padx = 20, pady=20)
        entry3 = tk.Entry(self)
        entry3.grid(row=4,column=2,padx = 20, pady=20)
        entry4 = tk.Entry(self)
        entry4.grid(row=5,column=2,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(ApplicantPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button1 = ttk.Button(self, text="Store",
                             command= lambda: wrapper(entry1, entry2, entry3, entry4))
        button1.grid(row=6, columnspan = 3, padx = 20, pady=20 )
```

Fig 8.6 Applying Page

- To apply for a job you need a valid Job Unique Id which will be validated
- So make sure you have noted the correct job id from the VIEW ALL/SPECIFIC JOBS section

```
class ViewSpecificJobPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        def wrapper(entry):
            c.execute('SELECT * FROM Job WHERE Role = ?',(entry.get(),))
            info = c.fetchall()
            c.execute('SELECT * FROM Job WHERE Employer = ?',(entry.get(),))
            info2 = c.fetchall()
            c.execute('SELECT * FROM Job WHERE Code = ?',(entry.get(),))
            info3 = c.fetchall()
            if len(info) == 0 & len(info2) == 0 & len(info3) == 0:
                messagebox.showinfo(title = "Caution", message = "No such job exists")
            else:
                ROW = info
                ROW.extend(info2)
                ROW.extend(info3)
                controller.dynamic_page(DisplaySpecific, parent, ROW)
                controller.show_frame(DisplaySpecific)


        query = tk.Label(self, text="- ENTER ROLE / EMPLOYER / CODE: ", font = LARGE_FONT)
        option1 = tk.Label(self, text="ID", font = LARGE_FONT)
        option2 = tk.Label(self, text="CODE", font = LARGE_FONT)
        option3 = tk.Label(self, text="ROLE", font = LARGE_FONT)
        option4 = tk.Label(self, text="EMPLOYER", font = LARGE_FONT)

        query.grid(row=1,column=1,columnspan = 2,padx = 20, pady=20)
        option1.grid(row=2,column=0,padx = 20, pady=20)
        option2.grid(row=2,column=1,padx = 20, pady=20)
        option3.grid(row=2,column=2,padx = 20, pady=20)
        option4.grid(row=2,column=3,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=1,column=3,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(ApplicantPage))
        button1.grid(row=0, columnspan = 2, padx = 20, pady=20 )
        button2 = ttk.Button(self, text="Fetch",
                             command= lambda: wrapper(entry1))
        button2.grid(row=1, column = 4, padx = 20, pady=20 )
```

Fig 8.7 View Specific Page

- To apply for a job you need a valid Job Unique Id which will be validated
- So make sure you have noted the correct job id from the VIEW ALL/SPECIFIC JOBS section

```
class ViewStatusPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        def wrapper(entry1, entry2):
            c.execute('SELECT * FROM Status WHERE Name = ? AND Code = ?',(entry1.get(),entry2.get(),))
            info = c.fetchall()
            if len(info) == 0:
                messagebox.showinfo(title = "Caution", message = "No such application was submitted")
            else:
                ROW = info[0]
                controller.dynamic_page(DisplayStatus, parent, ROW)
                controller.show_frame(DisplayStatus)


        query1 = tk.Label(self, text="- ENTER YOUR NAME: ", font = LARGE_FONT)
        query2 = tk.Label(self, text="- ENTER JOB CODE: ", font = LARGE_FONT)

        query1.grid(row=1,column=1,padx = 20, pady=20)
        query2.grid(row=2,column=1,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=1,column=2,padx = 20, pady=20)
        entry2 = tk.Entry(self)
        entry2.grid(row=2,column=2,padx = 20, pady=20)


        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(ApplicantPage))
        button1.grid(row=0, columnspan = 2, padx = 20, pady=20 )
        button1 = ttk.Button(self, text="Fetch",
                             command= lambda: wrapper(entry1,entry2))
        button1.grid(row=3, columnspan = 2, padx = 20, pady=20 )
```

Fig 8.8 View Status

- Enter Applicants name and Job unique Id and make sure you have applied for the job
- Before you check the status of your job application as it will be validated

```
class ViewAllJobPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        option1 = tk.Label(self, text="ID", font = LARGE_FONT)
        option2 = tk.Label(self, text="CODE", font = LARGE_FONT)
        option3 = tk.Label(self, text="ROLE", font = LARGE_FONT)
        option4 = tk.Label(self, text="EMPLOYER", font = LARGE_FONT)

        option1.grid(row=1,column=0,padx = 20, pady=20)
        option2.grid(row=1,column=1,padx = 20, pady=20)
        option3.grid(row=1,column=2,padx = 20, pady=20)
        option4.grid(row=1,column=3,padx = 20, pady=20)


        with conn:
            c = conn.cursor()
            c.execute('SELECT * FROM Job')
            index=2
            for row in c.fetchall():
                tk.Label(self, text=row[0], font = LARGE_FONT).grid(row=index, column=0,padx = 20, pady=20)
                tk.Label(self, text=row[1], font = LARGE_FONT).grid(row=index, column=1,padx = 20, pady=20)
                tk.Label(self, text=row[2], font = LARGE_FONT).grid(row=index, column=2,padx = 20, pady=20)
                tk.Label(self, text=row[3], font = LARGE_FONT).grid(row=index, column=3,padx = 20, pady=20)
                index+=1

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(ApplicantPage))
        button1.grid(row=0,column = 1,columnspan = 2, padx = 20, pady=20 )
```

Fig 8.9 View All Job

```
class EmployerPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        label = tk.Label(self, text="EMPLOYER PORTAL", font = HEADING_FONT)
        option1 = tk.Label(self, text="- REGISTER NEW EMPLOYER", font = LARGE_FONT)
        option2 = tk.Label(self, text="- POST A JOB", font = LARGE_FONT)
        option3 = tk.Label(self, text="- VIEW APPLICANTS", font = LARGE_FONT)
        option4 = tk.Label(self, text="- CONTACT AN APPLICANT", font = LARGE_FONT)

        label.grid(row=1, column=1,columnspan=2, padx = 20, pady=20)
        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)
        option4.grid(row=5,column=1,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(MainPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button2 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(RegisterPage))
        button2.grid(row=2, column = 2, sticky = "es",padx = 20, pady=20 )
        button3 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(PostJobPage))
        button3.grid(row=3, column = 2, sticky = "es",padx = 20, pady=20 )
        button4 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ViewApplicantPage))
        button4.grid(row=4, column = 2, sticky = "es",padx = 20, pady=20 )
        button5 = ttk.Button(self, text="...",
                             command= lambda: controller.show_frame(ContactApplicantPage))
        button5.grid(row=5, column = 2, sticky = "es",padx = 20, pady=20 )
```

Fig 8.10 Employer – Register Employer

```
class PostJobPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        option1 = tk.Label(self, text="- CODE ", font = LARGE_FONT)
        option2 = tk.Label(self, text="- ROLE", font = LARGE_FONT)
        option3 = tk.Label(self, text="- EMPLOYER", font = LARGE_FONT)

        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=2,column=2,padx = 20, pady=20)
        entry2 = tk.Entry(self)
        entry2.grid(row=3,column=2,padx = 20, pady=20)
        entry3 = tk.Entry(self)
        entry3.grid(row=4,column=2,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(EmployerPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button1 = ttk.Button(self, text="Store",
                             command= lambda: insert_job(entry1.get(), entry2.get(), entry3.get(),))
        button1.grid(row=6, columnspan = 3, padx = 20, pady=20 )
```

Fig 8.11 Posting Job

- To post a job you need to give a unique job code to the job which will be used throughout
- Like the job unique ID so you need to remember it as well
- REMEMBER you cannot post a job without an employer, since there is a validation
- Thus the employer has to be registered first only then the name will be accepted by the system

```
class RegisterPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        option1 = tk.Label(self, text="- NAME ", font = LARGE_FONT)
        option2 = tk.Label(self, text="- GST", font = LARGE_FONT)
        option3 = tk.Label(self, text="- ABOUT", font = LARGE_FONT)

        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=2,column=2,padx = 20, pady=20)
        entry2 = tk.Entry(self)
        entry2.grid(row=3,column=2,padx = 20, pady=20)
        entry3 = tk.Entry(self)
        entry3.grid(row=4,column=2,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(EmployerPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button1 = ttk.Button(self, text="Store",
                             command= lambda: insert_employer(entry1.get(), entry2.get(), entry3.get(),))
        button1.grid(row=6, columnspan = 3, padx = 20, pady=20 )
```

Fig 8.12 Registring Job

```
class ContactApplicantPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)

        def wrapper(name, code, message):
            c.execute('SELECT * FROM Status WHERE Name = ? AND Code = ?',(name.get(), code.get(),))
            info = c.fetchall()
            if len(info) == 0:
                messagebox.showinfo(title = "Caution", message = "No such Application Exists")
            else:
                update_status( name.get(), code.get(), message.get())
                messagebox.showinfo(title = "Success", message = "Message Sent Successfully")
                controller.show_frame(EmployerPage)


        option1 = tk.Label(self, text="- NAME ", font = LARGE_FONT)
        option2 = tk.Label(self, text="- CODE", font = LARGE_FONT)
        option3 = tk.Label(self, text="- MESSAGE", font = LARGE_FONT)

        option1.grid(row=2,column=1,padx = 20, pady=20)
        option2.grid(row=3,column=1,padx = 20, pady=20)
        option3.grid(row=4,column=1,padx = 20, pady=20)

        entry1 = tk.Entry(self)
        entry1.grid(row=2,column=2,padx = 20, pady=20)
        entry2 = tk.Entry(self)
        entry2.grid(row=3,column=2,padx = 20, pady=20)
        entry3 = tk.Entry(self)
        entry3.grid(row=4,column=2,padx = 20, pady=20)

        button1 = ttk.Button(self, text="Back",
                             command= lambda: controller.show_frame(EmployerPage))
        button1.grid(row=0, columnspan = 3, padx = 20, pady=20 )
        button1 = ttk.Button(self, text="Store",
                             command= lambda: wrapper(entry1, entry2, entry3))
        button1.grid(row=6, columnspan = 3, padx = 20, pady=20 )
```


Fig 8.13 Contact An Applicant

- Enter Applicants name and Job unique Id and make sure you have applied for the job
- before you check the status of your job application as it will be validated

CHAPTER 10

CONCLUSION

This application is totally developed by using python programming language and SQLITE3as the database application. Python being a simple and easy to code programming language, creating GUI's is simple and easy to implement for real world applications. And linking python to database using SQLITE3is also easily done using the SQLITE3 connector. So because all these advantages of python with GUI's and databases it will be easy and simple to use or create any application for real world usage. The project mainly focusses on providing an efficient and productive network between the Recruiters and the Applicants in the form of this application portal. Effectiveness and enhancement of the recruitment process. Dramatically reduces paper work and administrative work. Streamlines the recruitment process and reduces the corporate's overall recruitment cycle. Reduces the operational overheads of the company as well as ensures the accessibility of various opportunities of the Applicant. Provides a 2-Way Network between the recruiter and the applicant and thus makes the process of Job Recruitment easier and productive. Delivers the best possible solution for a recruiting organization as well as an applicant.