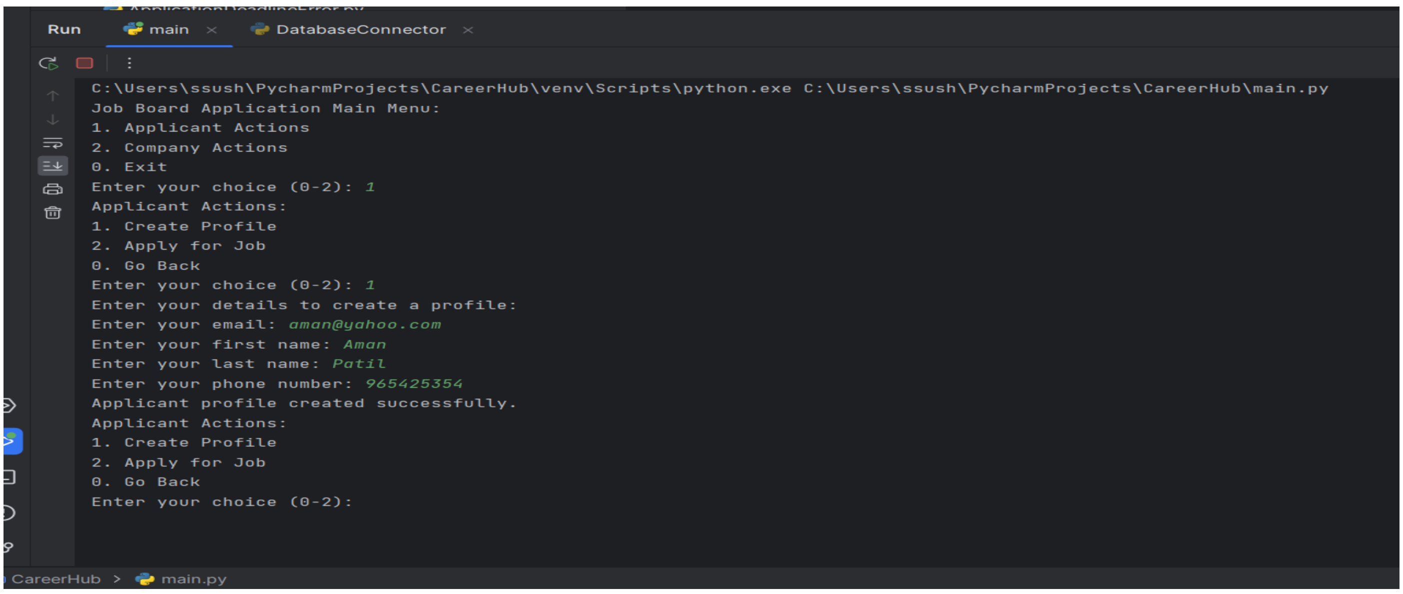
Coding Challenges 2: CareerHub, The Job Board

Name:- Sushant Kumar Singh

Problem Statement:

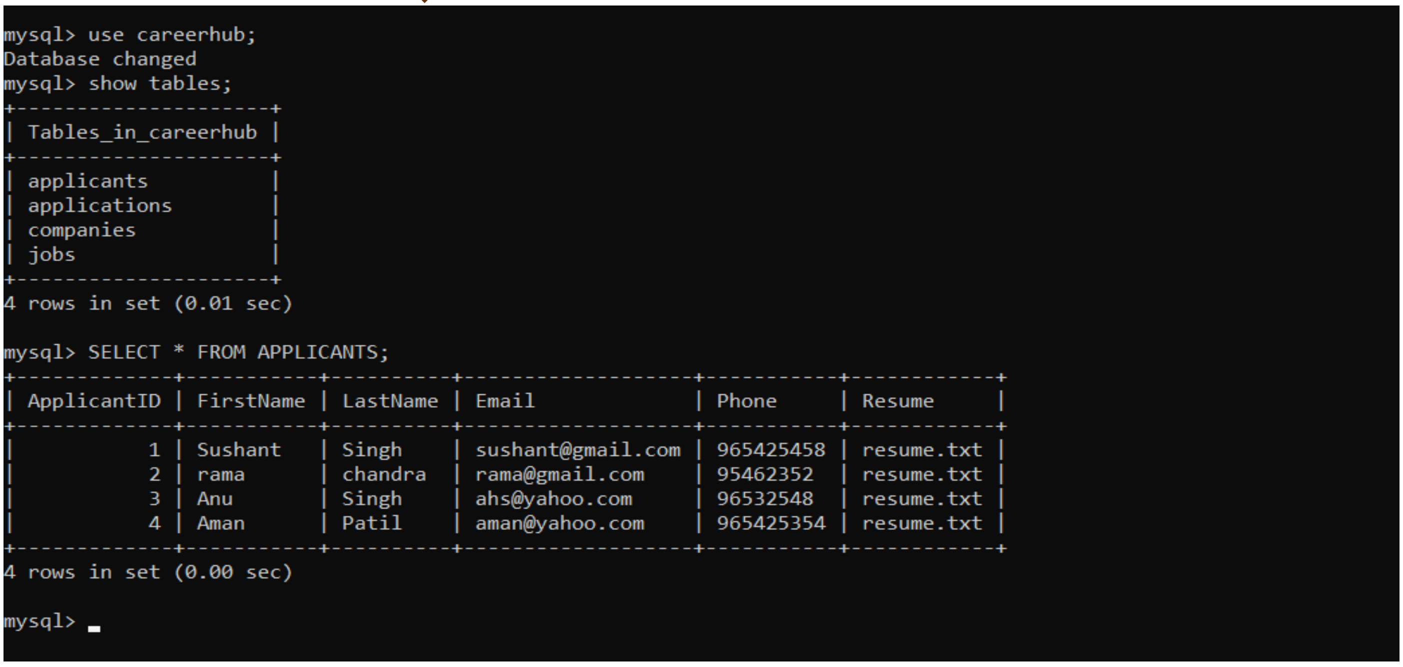
A Job Board scenario is a digital platform or system that facilitates the process of job searching and recruitment. In this scenario, various stakeholders, such as job seekers, companies, and recruiters, use the platform to post, search for, and apply to job opportunities.
Create SQL Schema from the application, use the class attributes for table column names.

SOLVED PROBLEM STATEMENT



```
C:\Users\ssush\PycharmProjects\CareerHub\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\CareerHub\main.py
Job Board Application Main Menu:
1. Applicant Actions
2. Company Actions
0. Exit
Enter your choice (0-2): 1
Applicant Actions:
1. Create Profile
2. Apply for Job
0. Go Back
Enter your choice (0-2): 1
Enter your details to create a profile:
Enter your email: aman@yahoo.com
Enter your first name: Aman
Enter your last name: Patil
Enter your phone number: 965425354
Applicant profile created successfully.
Applicant Actions:
1. Create Profile
2. Apply for Job
0. Go Back
Enter your choice (0-2):
```

OUR GIVEN INPUT SUCCESSEFULL SAVEED IN OUR DATABASE



```
mysql> use careerhub;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_careerhub |
+---------------------+
| applicants          |
| applications        |
| companies           |
| jobs                |
+---------------------+
4 rows in set (0.01 sec)

mysql> SELECT * FROM APPLICANTS;
+-------------+-----------+----------+-------------------+-----------+------------+
| ApplicantID | FirstName | LastName | Email             | Phone     | Resume     |
+-------------+-----------+----------+-------------------+-----------+------------+
|           1 | Sushant   | Singh    | sushant@gmail.com | 965425458 | resume.txt |
|           2 | rama      | chandra  | rama@gmail.com    | 95462352  | resume.txt |
|           3 | Anu       | Singh    | ahs@yahoo.com     | 96532548  | resume.txt |
|           4 | Aman      | Patil    | aman@yahoo.com    | 965425354 | resume.txt |
+-------------+-----------+----------+-------------------+-----------+------------+
4 rows in set (0.00 sec)

mysql> _
```

1.Create and implement the mentioned class and the structure in your application.

JobListing Class:

Attributes:

• JobID (int): A unique identifier for each job listing.

• CompanyID (int): A reference to the company offering the job.

• JobTitle (string): The title of the job.

• JobDescription (string): A detailed description of the job.

• JobLocation (string): The location of the job.

• Salary (decimal): The salary offered for the job.

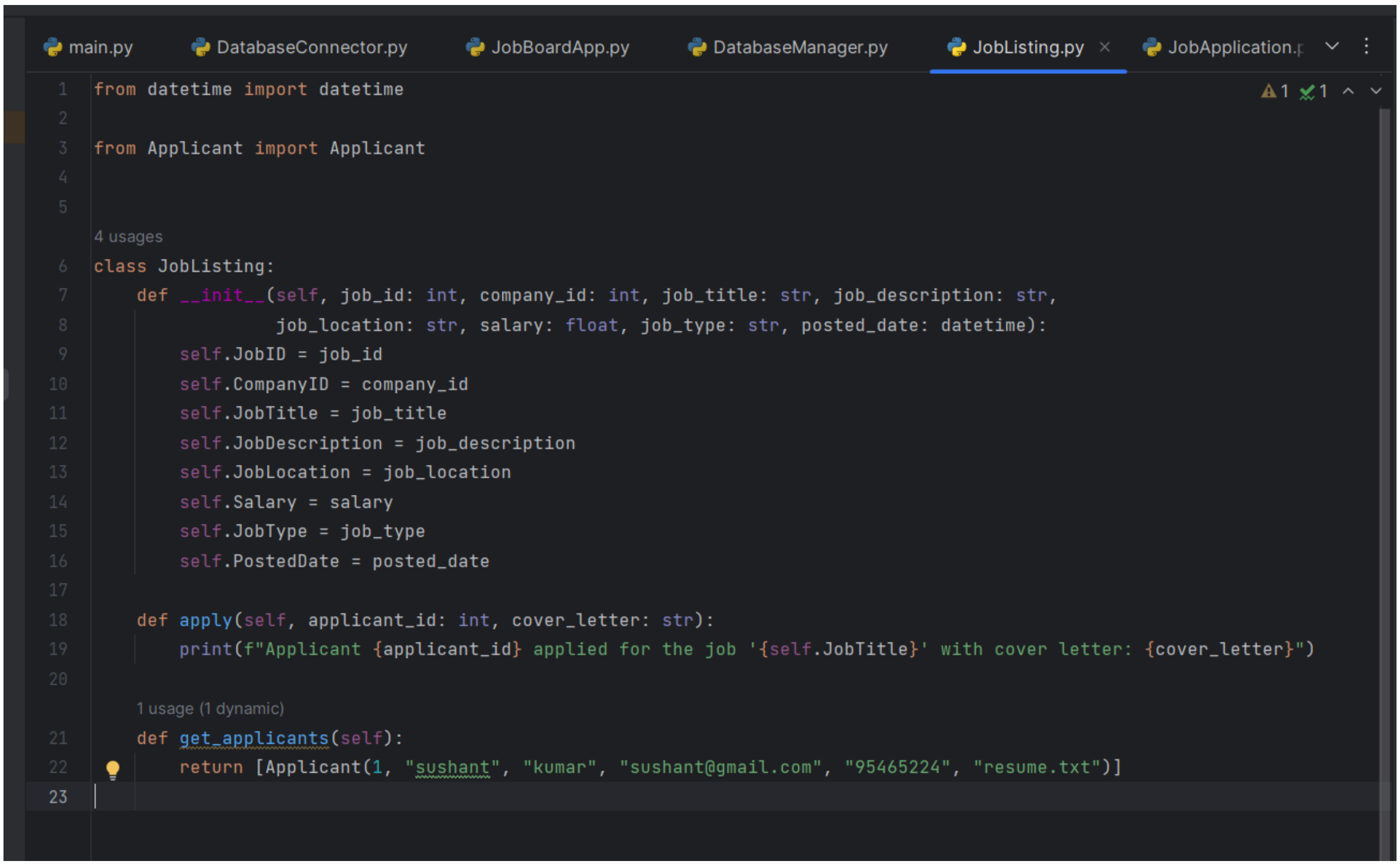
• JobType (string): The type of job (e.g., Full-time, Part-time, Contract).

• PostedDate (DateTime): The date when the job was posted.

Methods:

• Apply(applicantID: int, coverLetter: string): Allows applicants to apply for the job by providing their ID and a cover letter.

• GetApplicants(): List<Applicant>: Retrieves a list of applicants who have applied for the job.



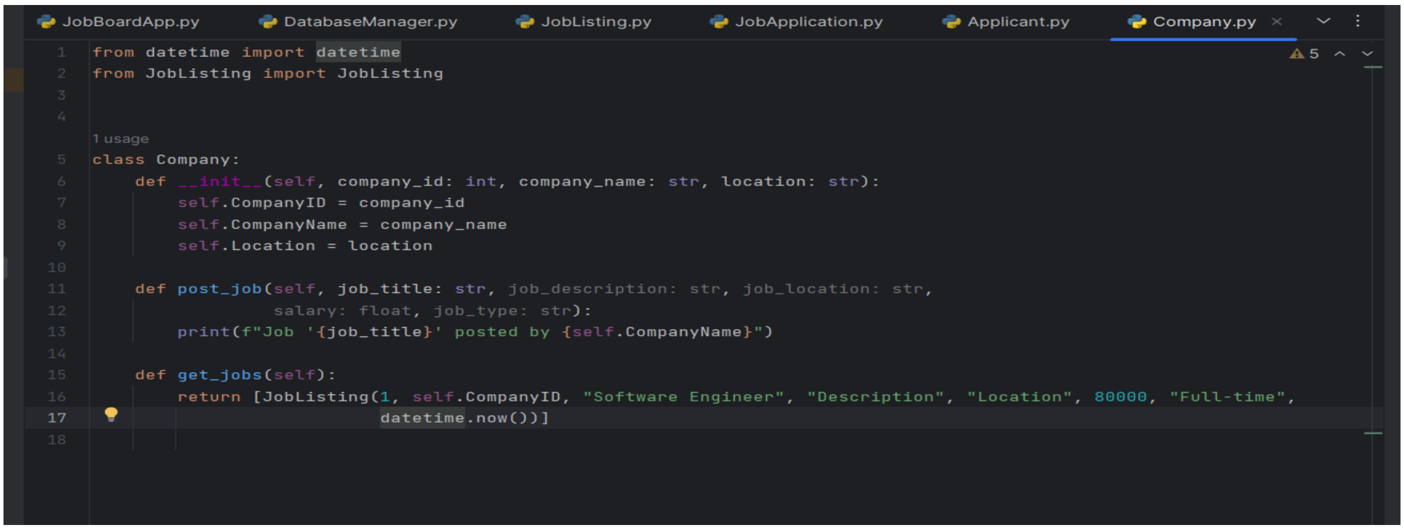
```python
from datetime import datetime

from Applicant import Applicant


4 usages
class JobListing:
    def __init__(self, job_id: int, company_id: int, job_title: str, job_description: str,
                 job_location: str, salary: float, job_type: str, posted_date: datetime):
        self.JobID = job_id
        self.CompanyID = company_id
        self.JobTitle = job_title
        self.JobDescription = job_description
        self.JobLocation = job_location
        self.Salary = salary
        self.JobType = job_type
        self.PostedDate = posted_date

    def apply(self, applicant_id: int, cover_letter: str):
        print(f"Applicant {applicant_id} applied for the job '{self.JobTitle}' with cover letter: {cover_letter}")

    1 usage (1 dynamic)
    def get_applicants(self):
        return [Applicant(1, "sushant", "kumar", "sushant@gmail.com", "95465224", "resume.txt")]
```

Company Class:
Attributes:

- CompanyID (int): A unique identifier for each company.
- CompanyName (string): The name of the hiring company.
- Location (string): The location of the company.

Methods:

- PostJob(jobTitle: string, jobDescription: string, jobLocation: string, salary: decimal, jobType: string): Allows a company to post a new job listing.
- GetJobs(): List<JobListing>: Retrieves a list of job listings posted by the company.



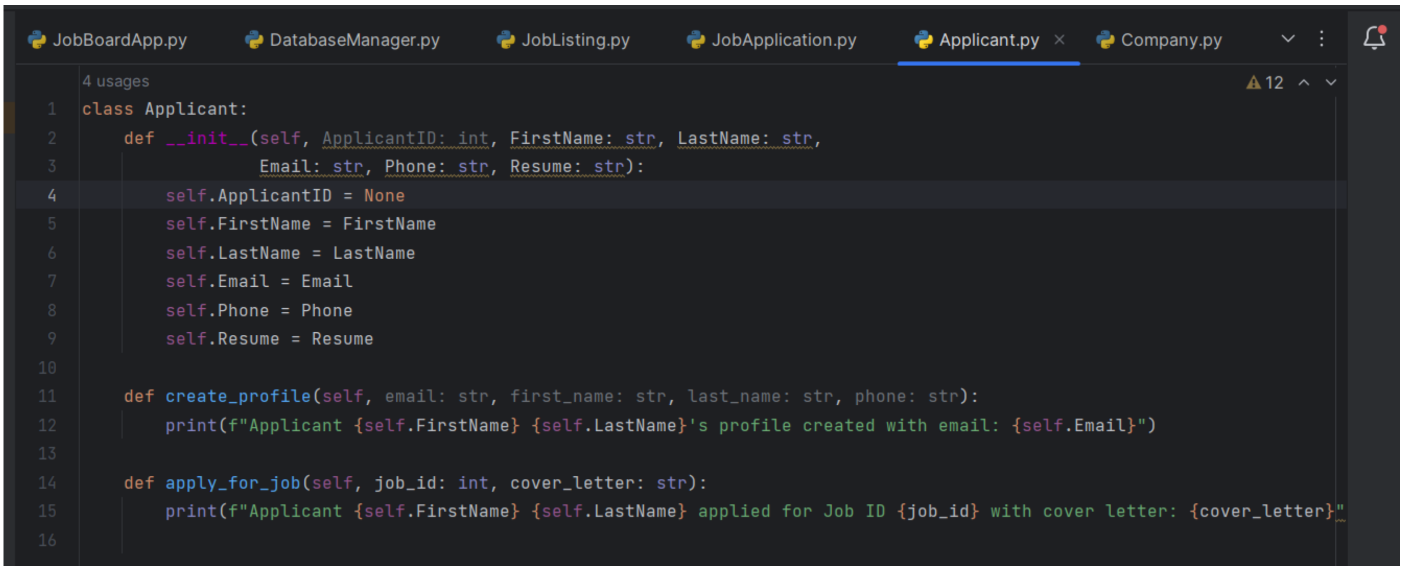
```python
from datetime import datetime
from JobListing import JobListing


class Company:
    def __init__(self, company_id: int, company_name: str, location: str):
        self.CompanyID = company_id
        self.CompanyName = company_name
        self.Location = location

    def post_job(self, job_title: str, job_description: str, job_location: str,
                 salary: float, job_type: str):
        print(f"Job '{job_title}' posted by {self.CompanyName}")

    def get_jobs(self):
        return [JobListing(1, self.CompanyID, "Software Engineer", "Description", "Location", 80000, "Full-time",
                           datetime.now())]

```

Applicant Class:
Attributes:

- ApplicantID (int): A unique identifier for each applicant.
- FirstName (string): The first name of the applicant.
- LastName (string): The last name of the applicant.
- Email (string): The email address of the applicant.
- Phone (string): The phone number of the applicant.
- Resume (string): The applicant's resume or a reference to the resume file.

Methods:

- CreateProfile(email: string, firstName: string, lastName: string, phone: string): Allows applicants to create a profile with their contact information.

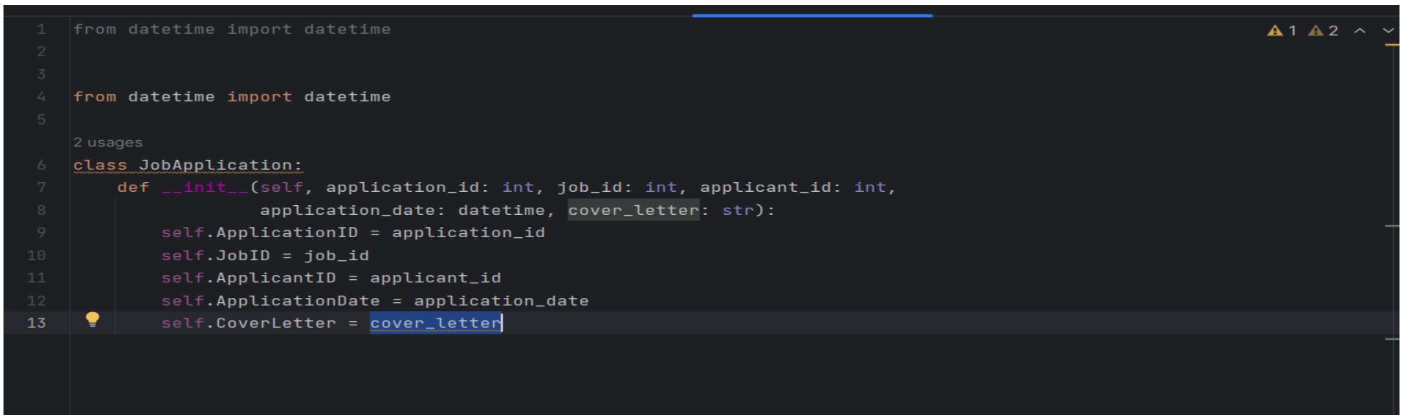


```python
class Applicant:
    def __init__(self, ApplicantID: int, FirstName: str, LastName: str,
                 Email: str, Phone: str, Resume: str):
        self.ApplicantID = None
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.Phone = Phone
        self.Resume = Resume

    def create_profile(self, email: str, first_name: str, last_name: str, phone: str):
        print(f"Applicant {self.FirstName} {self.LastName}'s profile created with email: {self.Email}")

    def apply_for_job(self, job_id: int, cover_letter: str):
        print(f"Applicant {self.FirstName} {self.LastName} applied for Job ID {job_id} with cover letter: {cover_letter}"

```

JobApplication Class:

Attributes:

- ApplicationID (int): A unique identifier for each job application.
- JobID (int): A reference to the job listing.
- ApplicantID (int): A reference to the applicant.
- ApplicationDate (DateTime): The date and time when the application was submitted.
- CoverLetter (string): The cover letter submitted with the application.

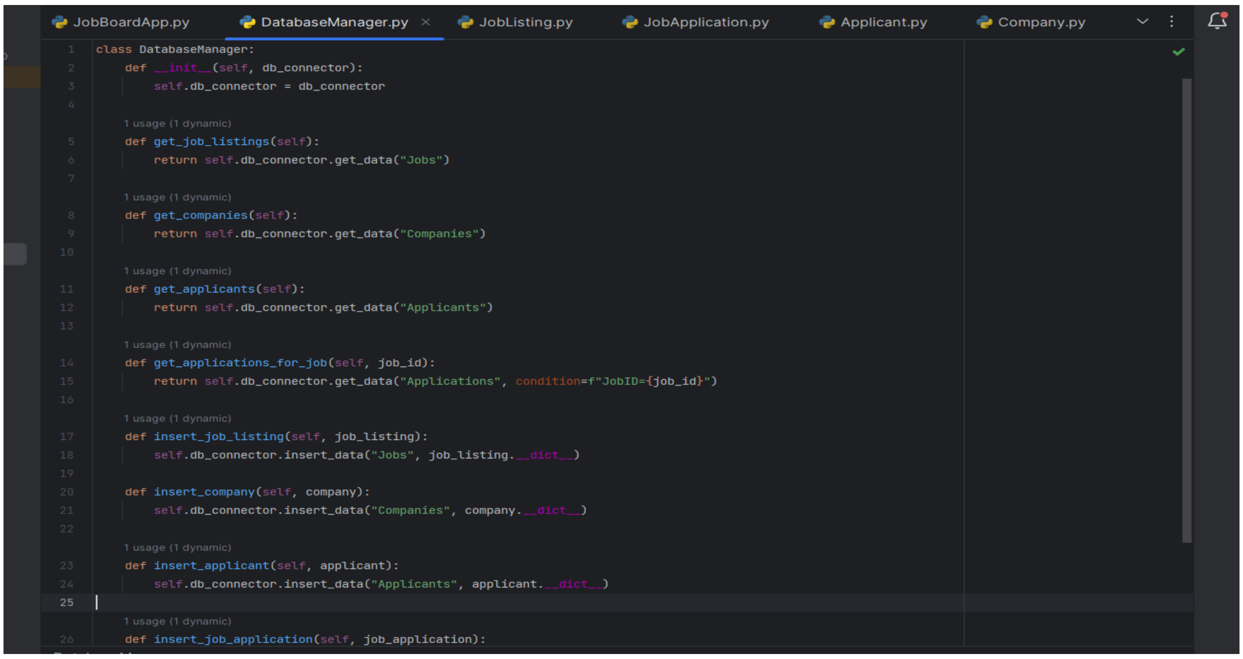


```python
from datetime import datetime


from datetime import datetime

2 usages
class JobApplication:
    def __init__(self, application_id: int, job_id: int, applicant_id: int,
                 application_date: datetime, cover_letter: str):
        self.ApplicationID = application_id
        self.JobID = job_id
        self.ApplicantID = applicant_id
        self.ApplicationDate = application_date
        self.CoverLetter = cover_letter
```

2.DatabaseManager Class:

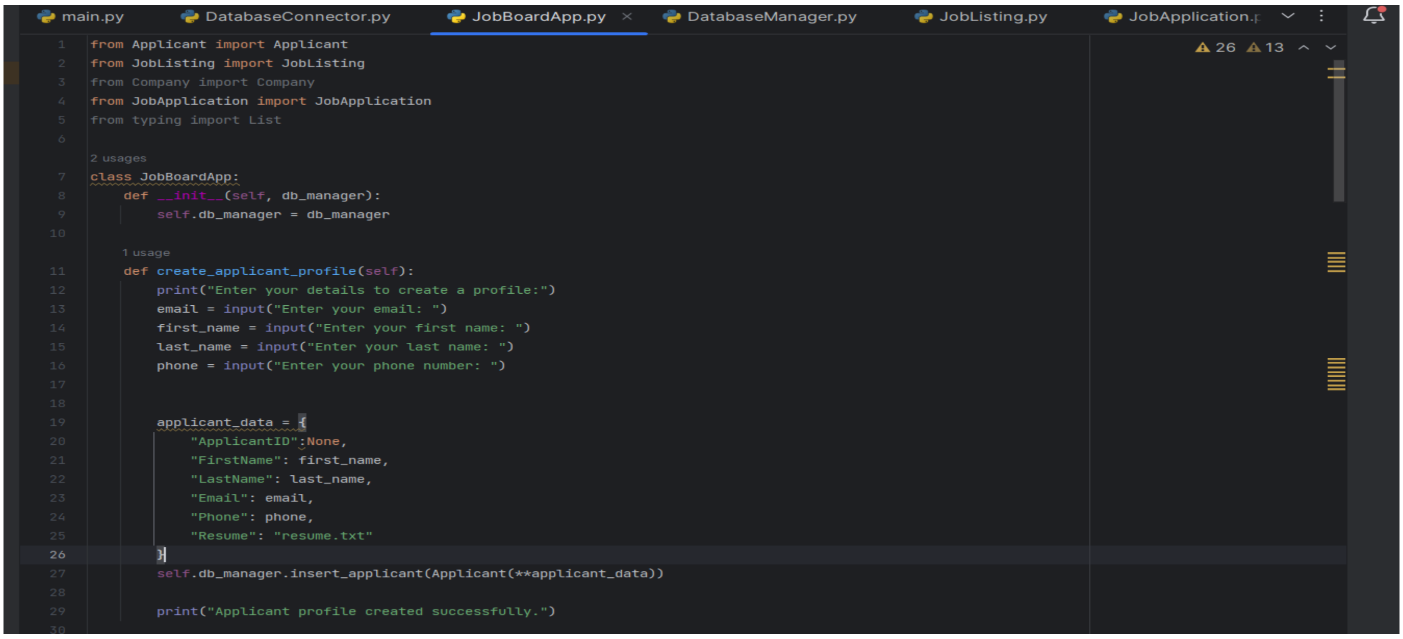
Methods:

- InitializeDatabase(): Initializes the database schema and tables.
- InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.
- InsertCompany(company: Company): Inserts a new company into the "Companies" table.
- InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.
- InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications"
- GetJobListings(): List<JobListing>: Retrieves a list of all job listings.
- GetCompanies(): List<Company>: Retrieves a list of all companies.
- GetApplicants(): List<Applicant>: Retrieves a list of all applicants.
- GetApplicationsForJob(jobID: int): List<JobApplication>: Retrieves a list of job applications for a specific job listing.

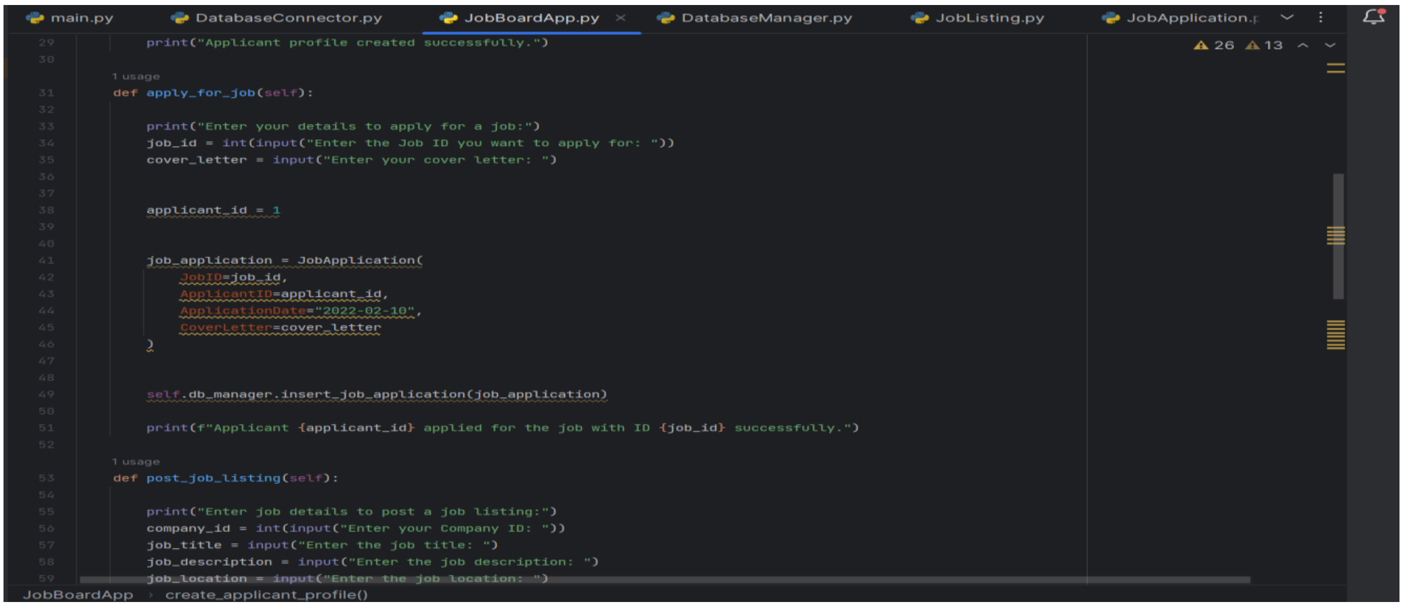


```python
class DatabaseManager:
    def __init__(self, db_connector):
        self.db_connector = db_connector

    1 usage (1 dynamic)
    def get_job_listings(self):
        return self.db_connector.get_data("Jobs")

    1 usage (1 dynamic)
    def get_companies(self):
        return self.db_connector.get_data("Companies")

    1 usage (1 dynamic)
    def get_applicants(self):
        return self.db_connector.get_data("Applicants")

    1 usage (1 dynamic)
    def get_applications_for_job(self, job_id):
        return self.db_connector.get_data("Applications", condition=f"JobID={job_id}")

    1 usage (1 dynamic)
    def insert_job_listing(self, job_listing):
        self.db_connector.insert_data("Jobs", job_listing.__dict__)

    def insert_company(self, company):
        self.db_connector.insert_data("Companies", company.__dict__)

    1 usage (1 dynamic)
    def insert_applicant(self, applicant):
        self.db_connector.insert_data("Applicants", applicant.__dict__)


    1 usage (1 dynamic)
    def insert_job_application(self, job_application):
```

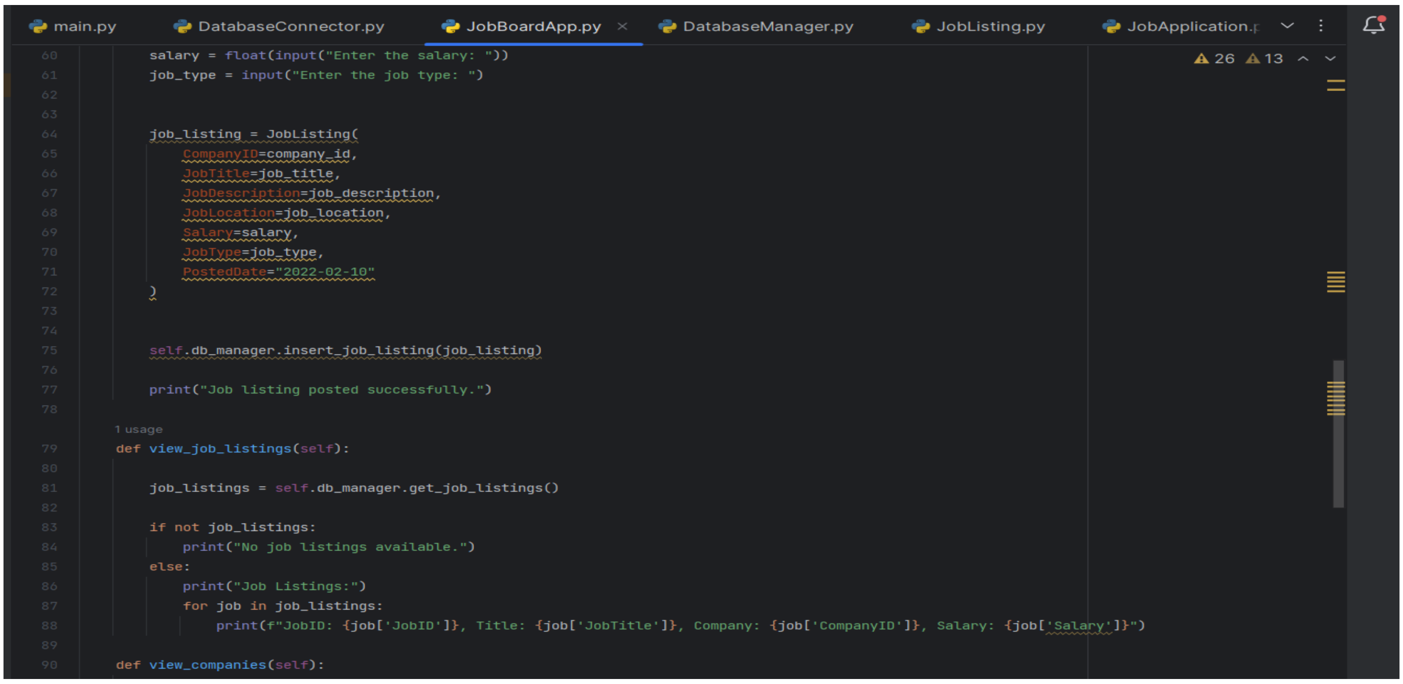
JOBBOARDAPP.PY CLASS



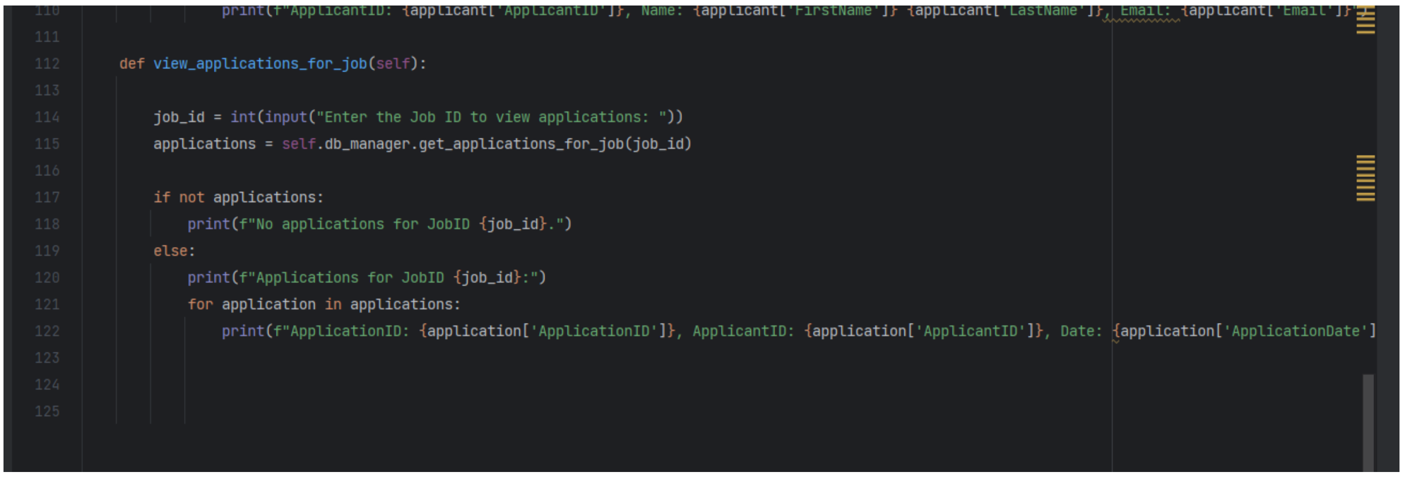
```python
from Applicant import Applicant
from JobListing import JobListing
from Company import Company
from JobApplication import JobApplication
from typing import List

2 usages
class JobBoardApp:
    def __init__(self, db_manager):
        self.db_manager = db_manager

    1 usage
    def create_applicant_profile(self):
        print("Enter your details to create a profile:")
        email = input("Enter your email: ")
        first_name = input("Enter your first name: ")
        last_name = input("Enter your last name: ")
        phone = input("Enter your phone number: ")


        applicant_data = {
            "ApplicantID":None,
            "FirstName": first_name,
            "LastName": last_name,
            "Email": email,
            "Phone": phone,
            "Resume": "resume.txt"
        }
        self.db_manager.insert_applicant(Applicant(**applicant_data))

        print("Applicant profile created successfully.")
```



```python
        print("Applicant profile created successfully.")

    1 usage
    def apply_for_job(self):

        print("Enter your details to apply for a job:")
        job_id = int(input("Enter the Job ID you want to apply for: "))
        cover_letter = input("Enter your cover letter: ")


        applicant_id = 1


        job_application = JobApplication(
            JobID=job_id,
            ApplicantID=applicant_id,
            ApplicationDate="2022-02-10",
            CoverLetter=cover_letter
        )


        self.db_manager.insert_job_application(job_application)

        print(f"Applicant {applicant_id} applied for the job with ID {job_id} successfully.")

    1 usage
    def post_job_listing(self):

        print("Enter job details to post a job listing:")
        company_id = int(input("Enter your Company ID: "))
        job_title = input("Enter the job title: ")
        job_description = input("Enter the job description: ")
        job_location = input("Enter the job location: ")
```

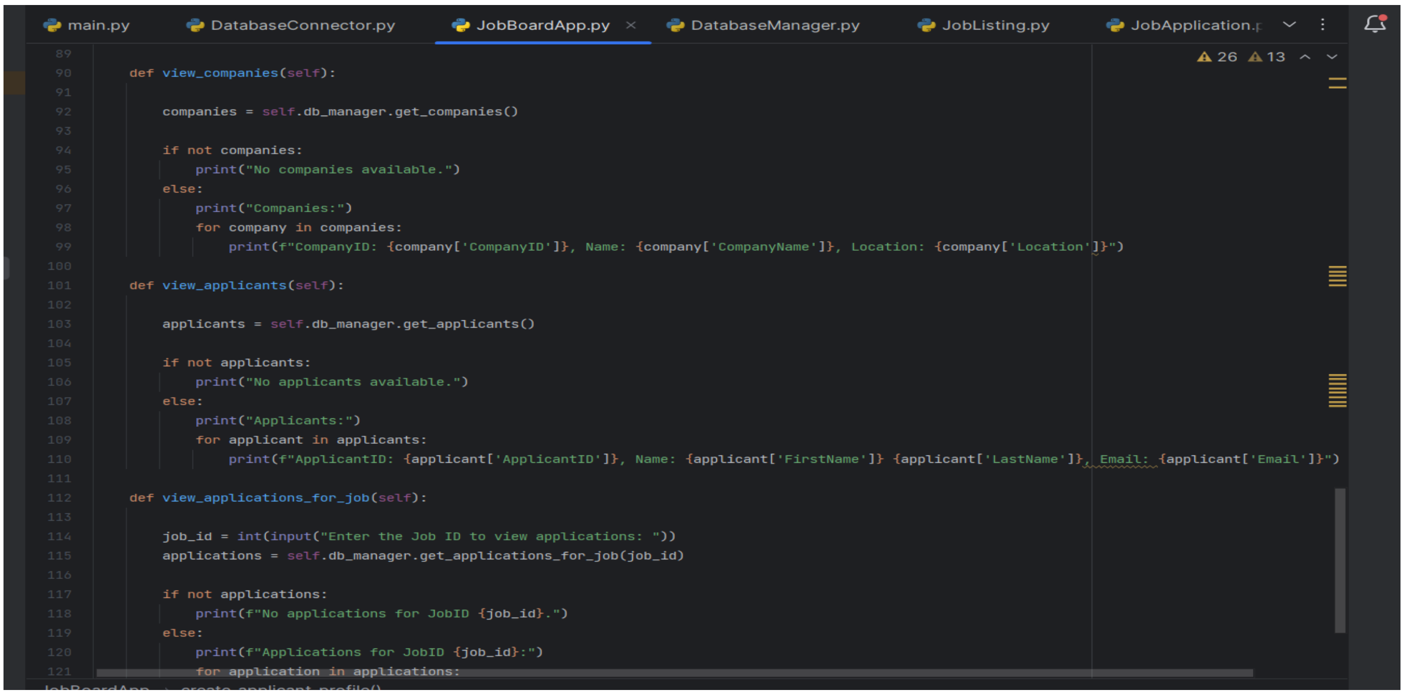


```python
        salary = float(input("Enter the salary: "))
        job_type = input("Enter the job type: ")


        job_listing = JobListing(
            CompanyID=company_id,
            JobTitle=job_title,
            JobDescription=job_description,
            JobLocation=job_location,
            Salary=salary,
            JobType=job_type,
            PostedDate="2022-02-10"
        )


        self.db_manager.insert_job_listing(job_listing)

        print("Job listing posted successfully.")

    1 usage
    def view_job_listings(self):

        job_listings = self.db_manager.get_job_listings()

        if not job_listings:
            print("No job listings available.")
        else:
            print("Job Listings:")
            for job in job_listings:
                print(f"JobID: {job['JobID']}, Title: {job['JobTitle']}, Company: {job['CompanyID']}, Salary: {job['Salary']}")

    def view_companies(self):
```

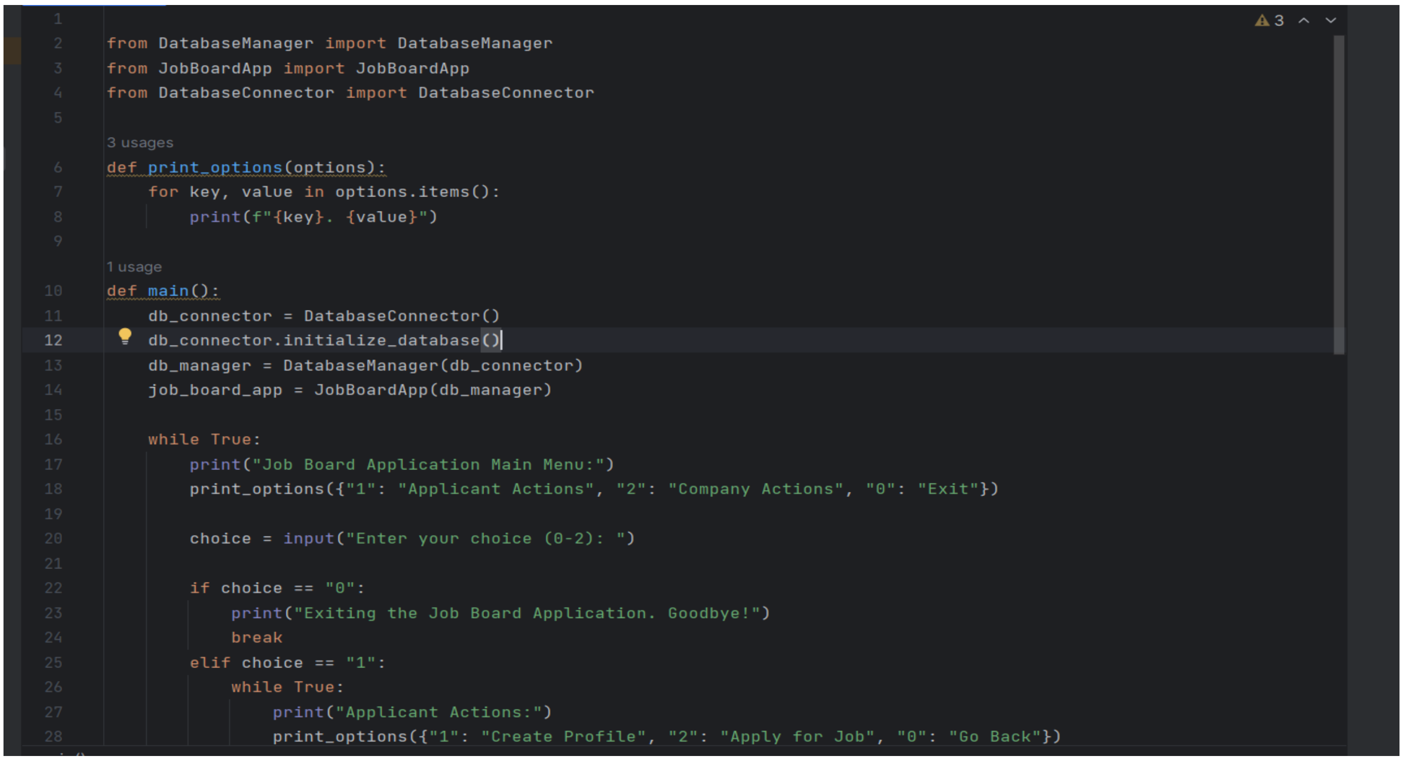


```python
                print(f"ApplicantID: {applicant['ApplicantID']}, Name: {applicant['FirstName']} {applicant['LastName']}, Email: {applicant['Email']}")

    def view_applications_for_job(self):

        job_id = int(input("Enter the Job ID to view applications: "))
        applications = self.db_manager.get_applications_for_job(job_id)

        if not applications:
            print(f"No applications for JobID {job_id}.")
        else:
            print(f"Applications for JobID {job_id}:")
            for application in applications:
                print(f"ApplicationID: {application['ApplicationID']}, ApplicantID: {application['ApplicantID']}, Date: {application['ApplicationDate']
```

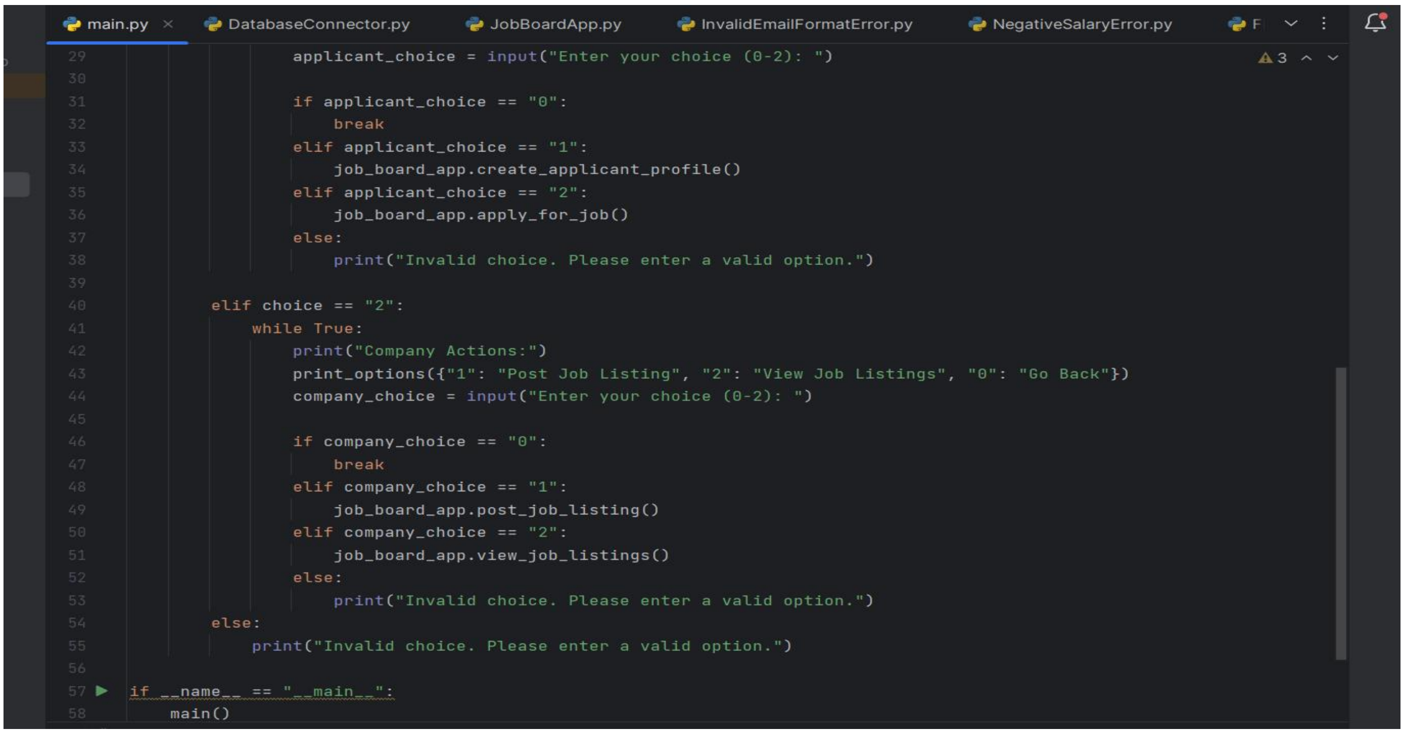
MAIN.PY CLASS



```python

    def view_companies(self):

        companies = self.db_manager.get_companies()

        if not companies:
            print("No companies available.")
        else:
            print("Companies:")
            for company in companies:
                print(f"CompanyID: {company['CompanyID']}, Name: {company['CompanyName']}, Location: {company['Location']}")

    def view_applicants(self):

        applicants = self.db_manager.get_applicants()

        if not applicants:
            print("No applicants available.")
        else:
            print("Applicants:")
            for applicant in applicants:
                print(f"ApplicantID: {applicant['ApplicantID']}, Name: {applicant['FirstName']} {applicant['LastName']}, Email: {applicant['Email']}")

    def view_applications_for_job(self):

        job_id = int(input("Enter the Job ID to view applications: "))
        applications = self.db_manager.get_applications_for_job(job_id)

        if not applications:
            print(f"No applications for JobID {job_id}.")
        else:
            print(f"Applications for JobID {job_id}:")
            for application in applications:
```



```python

from DatabaseManager import DatabaseManager
from JobBoardApp import JobBoardApp
from DatabaseConnector import DatabaseConnector

3 usages
def print_options(options):
    for key, value in options.items():
        print(f"{key}. {value}")

1 usage
def main():
    db_connector = DatabaseConnector()
    db_connector.initialize_database()
    db_manager = DatabaseManager(db_connector)
    job_board_app = JobBoardApp(db_manager)

    while True:
        print("Job Board Application Main Menu:")
        print_options({"1": "Applicant Actions", "2": "Company Actions", "0": "Exit"})

        choice = input("Enter your choice (0-2): ")

        if choice == "0":
            print("Exiting the Job Board Application. Goodbye!")
            break
        elif choice == "1":
            while True:
                print("Applicant Actions:")
                print_options({"1": "Create Profile", "2": "Apply for Job", "0": "Go Back"})
```



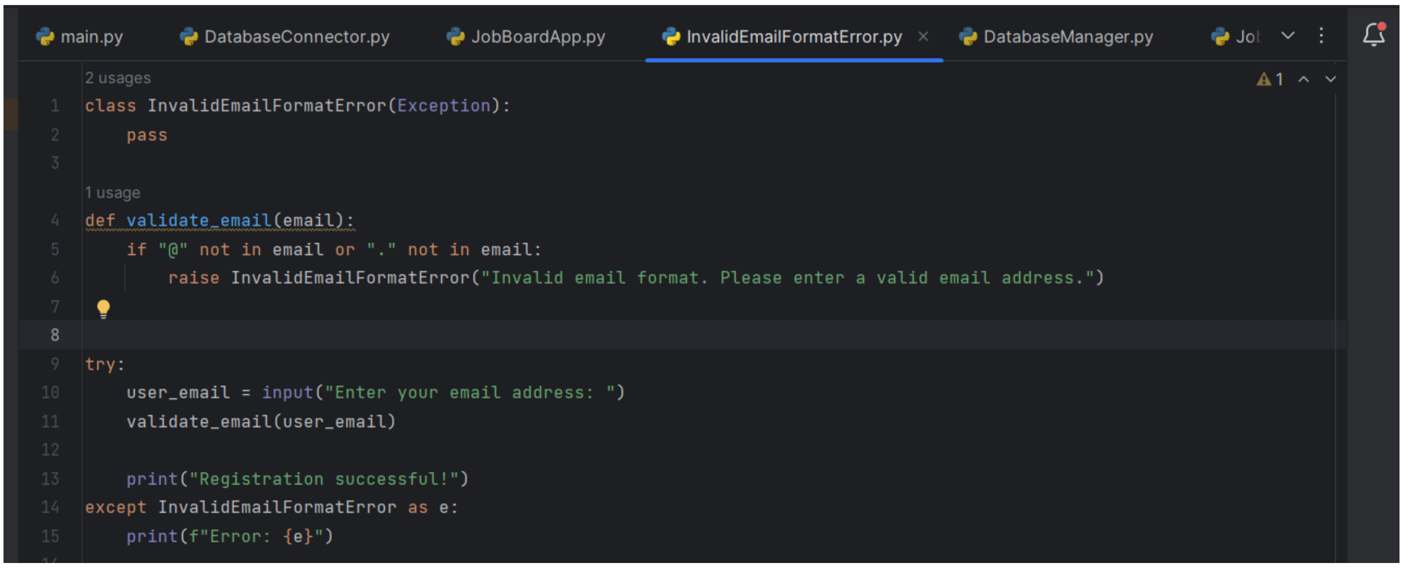
```python
                applicant_choice = input("Enter your choice (0-2): ")

                if applicant_choice == "0":
                    break
                elif applicant_choice == "1":
                    job_board_app.create_applicant_profile()
                elif applicant_choice == "2":
                    job_board_app.apply_for_job()
                else:
                    print("Invalid choice. Please enter a valid option.")

        elif choice == "2":
            while True:
                print("Company Actions:")
                print_options({"1": "Post Job Listing", "2": "View Job Listings", "0": "Go Back"})
                company_choice = input("Enter your choice (0-2): ")

                if company_choice == "0":
                    break
                elif company_choice == "1":
                    job_board_app.post_job_listing()
                elif company_choice == "2":
                    job_board_app.view_job_listings()
                else:
                    print("Invalid choice. Please enter a valid option.")
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

3.Exceptions handling

Create and implement the following exceptions in your application.

• **Invalid Email Format Handling:**

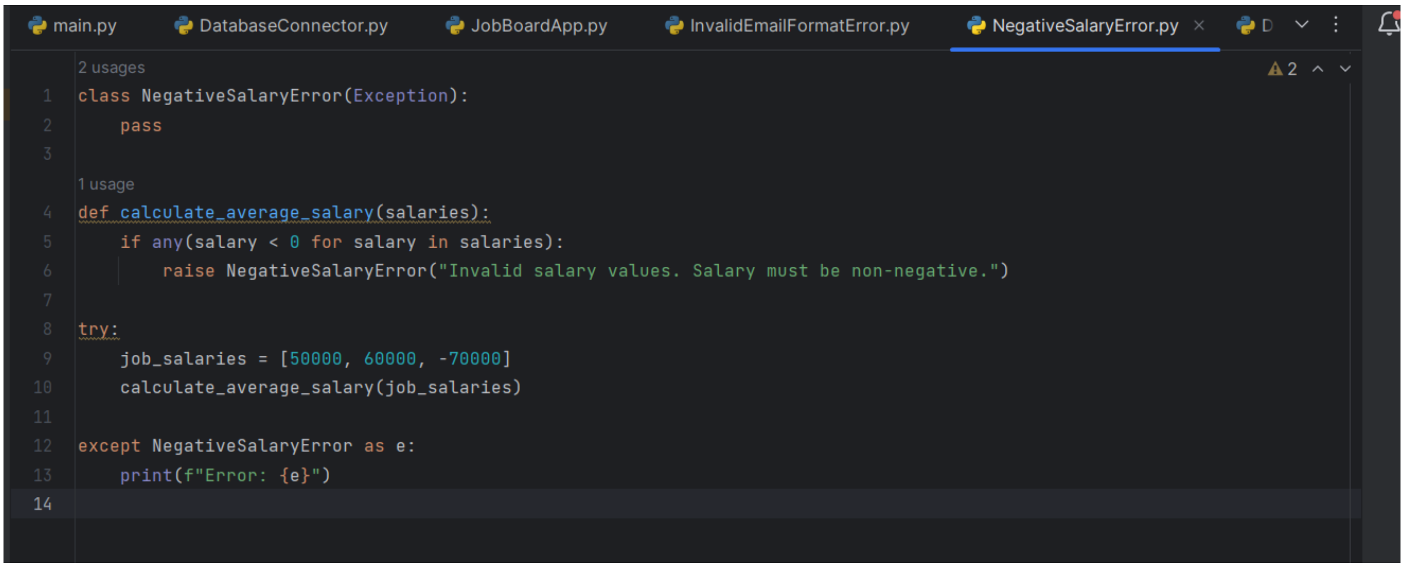
o In the Job Board application, during the applicant registration process, users are required to enter their email addresses. Write a program that prompts the user to input an email address and implement exception handling to ensure that the email address follows a valid format (e.g., contains "@" and a valid domain). If the input is not valid, catch the exception and display an error message. If it is valid, proceed with registration



```python
class InvalidEmailFormatError(Exception):
    pass

def validate_email(email):
    if "@" not in email or "." not in email:
        raise InvalidEmailFormatError("Invalid email format. Please enter a valid email address.")


try:
    user_email = input("Enter your email address: ")
    validate_email(user_email)

    print("Registration successful!")
except InvalidEmailFormatError as e:
    print(f"Error: {e}")
```

Salary Calculation Handling:

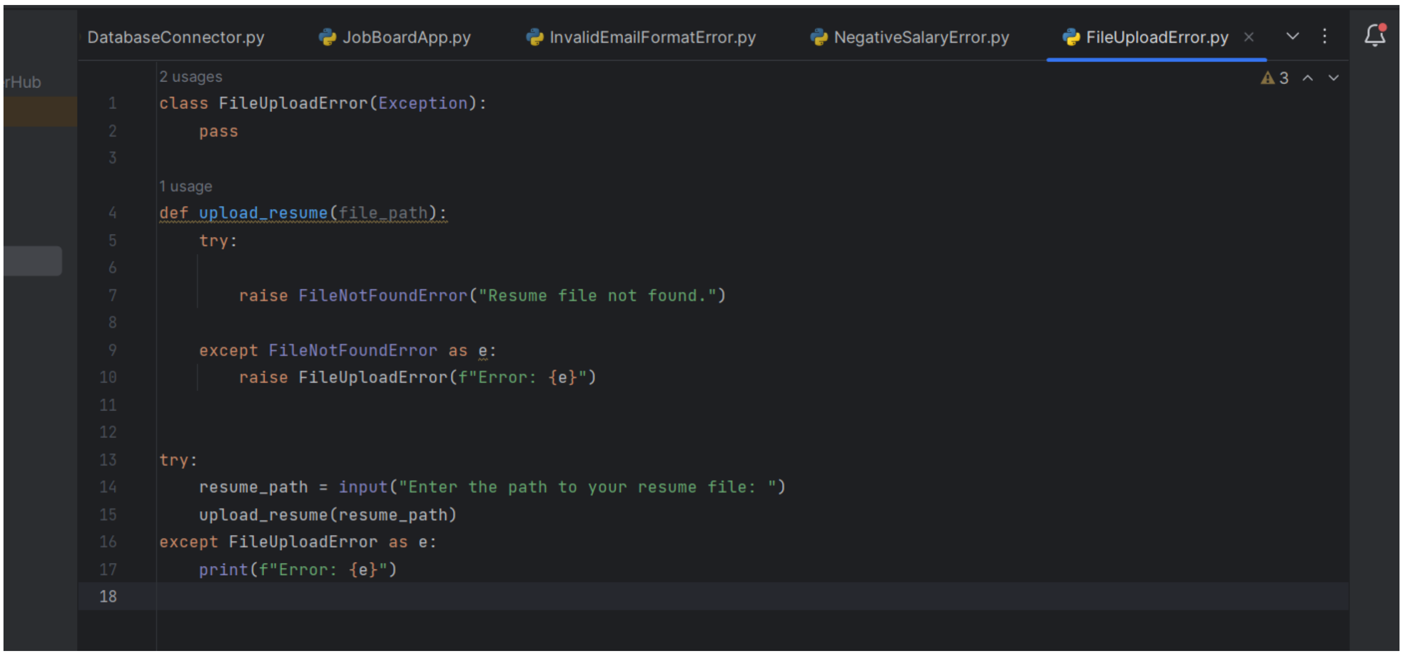
o Create a program that calculates the average salary offered by companies for job listings. Implement exception handling to ensure that the salary values are non-negative when computing the average. If any salary is negative or invalid, catch the exception and



```python
class NegativeSalaryError(Exception):
    pass

def calculate_average_salary(salaries):
    if any(salary < 0 for salary in salaries):
        raise NegativeSalaryError("Invalid salary values. Salary must be non-negative.")

try:
    job_salaries = [50000, 60000, -70000]
    calculate_average_salary(job_salaries)

except NegativeSalaryError as e:
    print(f"Error: {e}")
```

• **File Upload Exception Handling:**

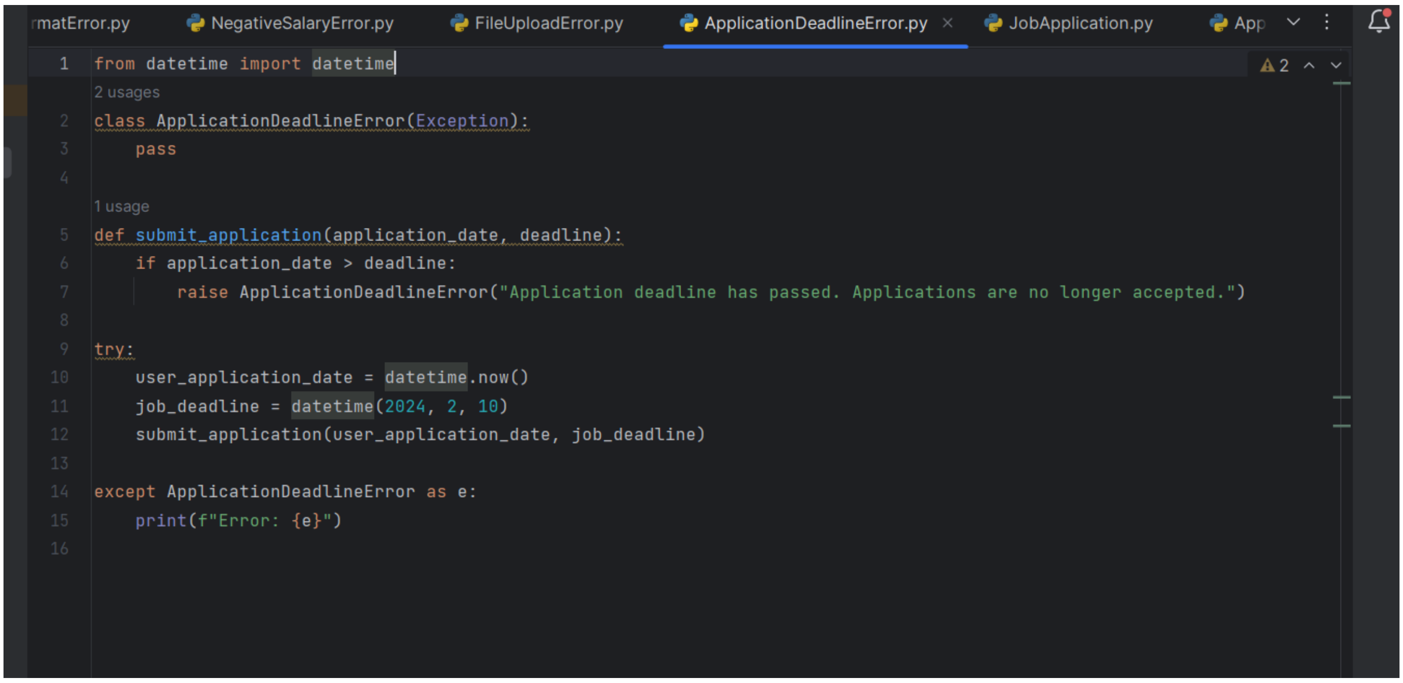
o In the Job Board application, applicants can upload their resumes as files. Write a program that handles file uploads and implements exception handling to catch and handle potential errors, such as file not found, file size exceeded, or file format not supported. Provide appropriate error messages in each case.



```python
class FileUploadError(Exception):
    pass

def upload_resume(file_path):
    try:

        raise FileNotFoundError("Resume file not found.")

    except FileNotFoundError as e:
        raise FileUploadError(f"Error: {e}")


try:
    resume_path = input("Enter the path to your resume file: ")
    upload_resume(resume_path)
except FileUploadError as e:
    print(f"Error: {e}")
```

• Application Deadline Handling:

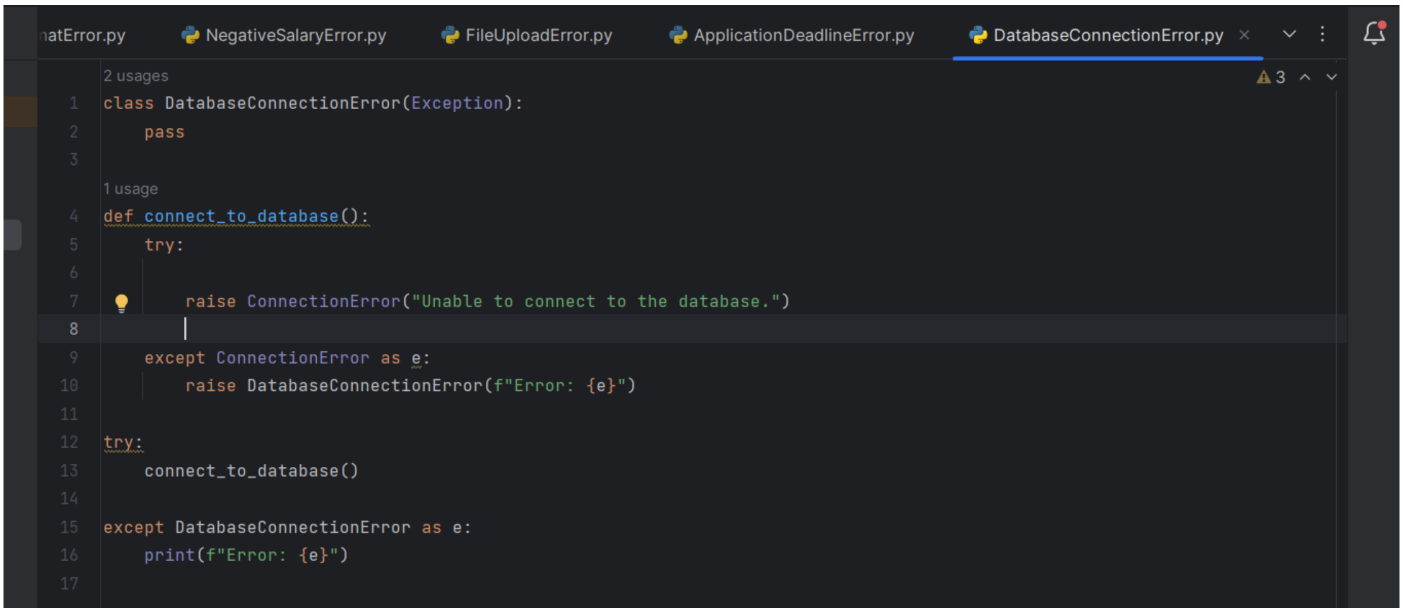
o Develop a program that checks whether a job application is submitted before the application deadline. Implement exception handling to catch situations where an applicant tries to submit an application after the deadline has passed. Display a message indicating that the application is no longer accepted.



```python
from datetime import datetime
class ApplicationDeadlineError(Exception):
    pass

def submit_application(application_date, deadline):
    if application_date > deadline:
        raise ApplicationDeadlineError("Application deadline has passed. Applications are no longer accepted.")

try:
    user_application_date = datetime.now()
    job_deadline = datetime(2024, 2, 10)
    submit_application(user_application_date, job_deadline)

except ApplicationDeadlineError as e:
    print(f"Error: {e}")
```

• Database Connection Handling:

o In the Job Board application, database connectivity is crucial. Create a program that establishes a connection to the database to retrieve job listings. Implement exception handling to catch database-related exceptions, such as connection errors or SQL query errors. Display appropriate error messages and ensure graceful handling of these exceptions.



```python
class DatabaseConnectionError(Exception):
    pass

def connect_to_database():
    try:

        raise ConnectionError("Unable to connect to the database.")

    except ConnectionError as e:
        raise DatabaseConnectionError(f"Error: {e}")

try:
    connect_to_database()

except DatabaseConnectionError as e:
    print(f"Error: {e}")
```

4.Database Connectivity

Create and implement the following tasks in your application.

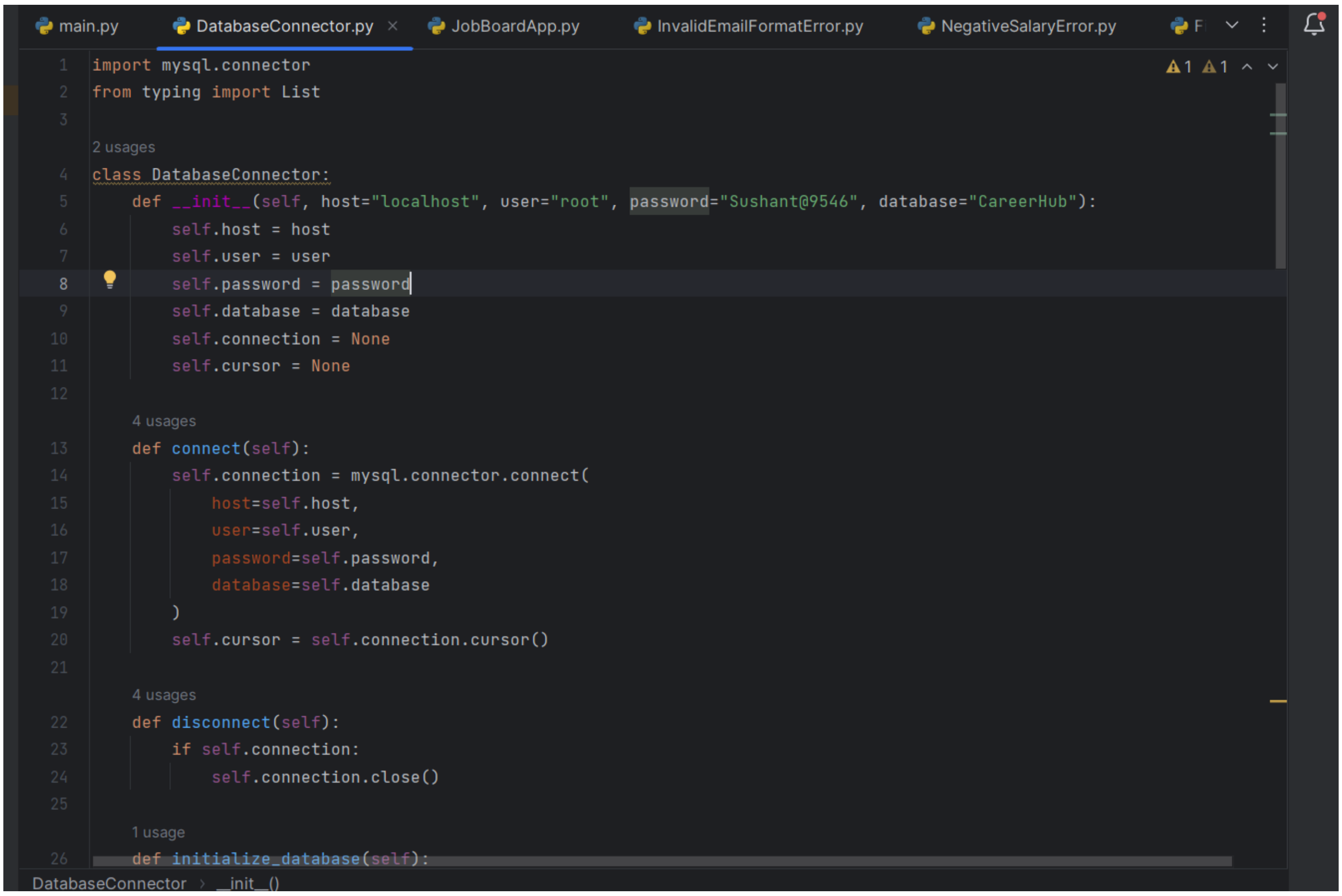
Job Listing Retrieval: Write a program that connects to the database and retrieves all job listings from the "Jobs" table. Implement database connectivity using Entity Framework and display the job titles, company names, and salaries.

Applicant Profile Creation: Create a program that allows applicants to create a profile by entering their information. Implement database connectivity to insert the applicant's data into the "Applicants" table. Handle potential database-related exceptions.

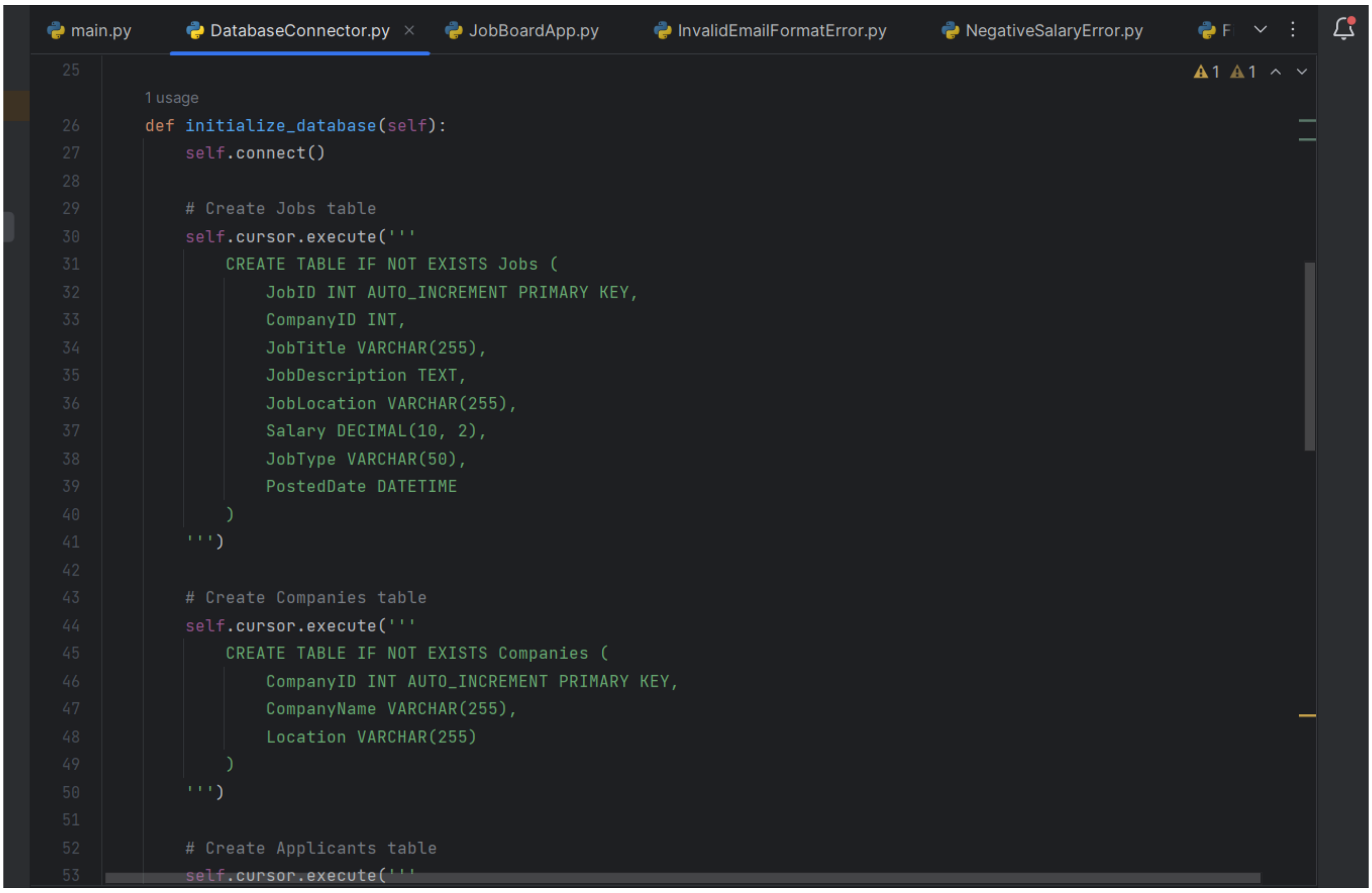
Job Application Submission: Develop a program that allows applicants to apply for a specific job listing. Implement database connectivity to insert the job application details into the "Applications" table, including the applicant's ID and the job ID. Ensure that the program handles database connectivity and insertion exceptions.

Company Job Posting: Write a program that enables companies to post new job listings. Implement database connectivity to insert job listings into the "Jobs" table, including the company's ID. Handle database-related exceptions and ensure the job posting is successful.

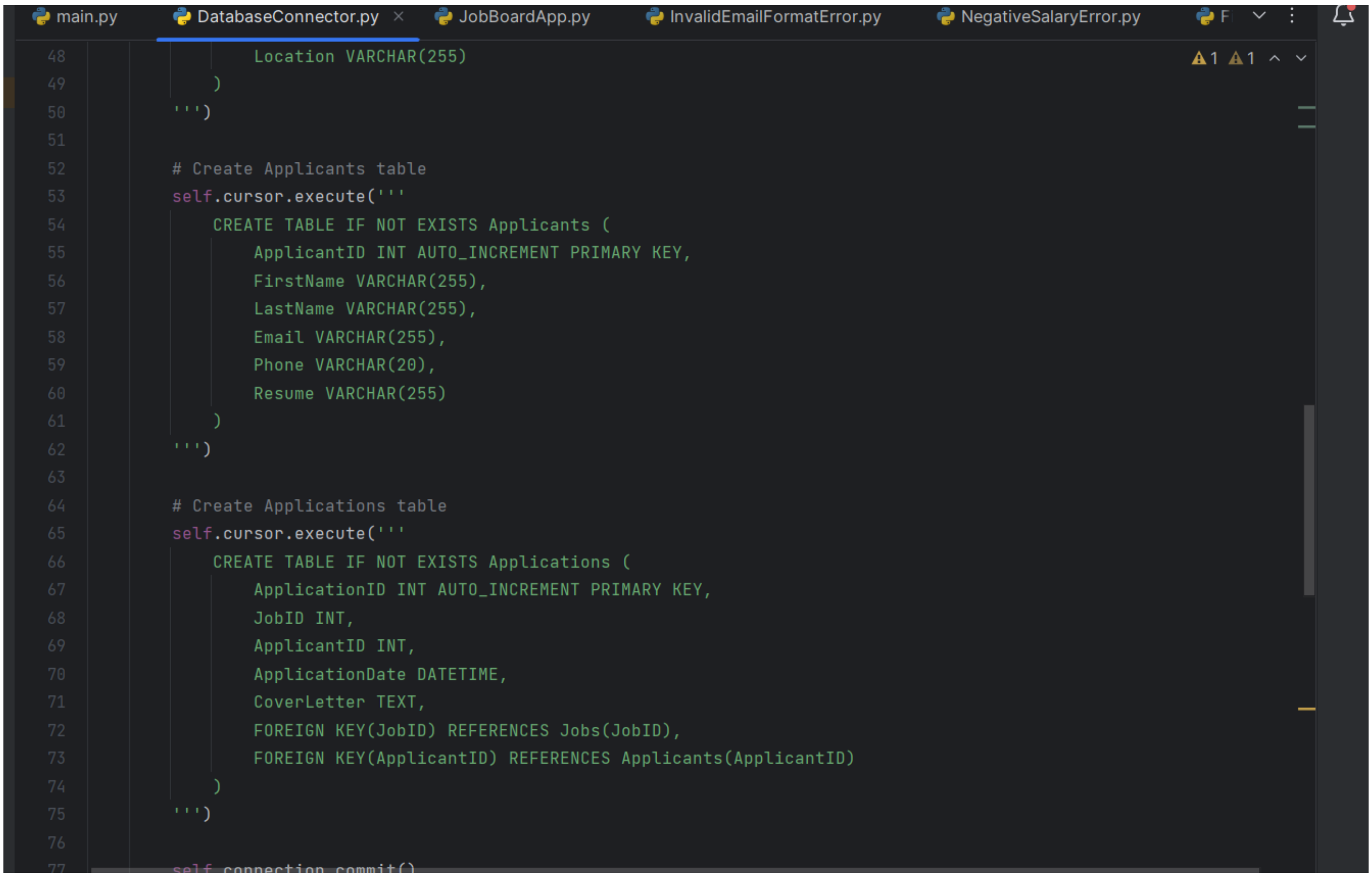
Salary Range Query: Create a program that allows users to search for job listings within a specified salary range. Implement database connectivity to retrieve job listings that match the user's criteria, including job titles, company names, and salaries. Ensure the program handles database connectivity and query exceptions.



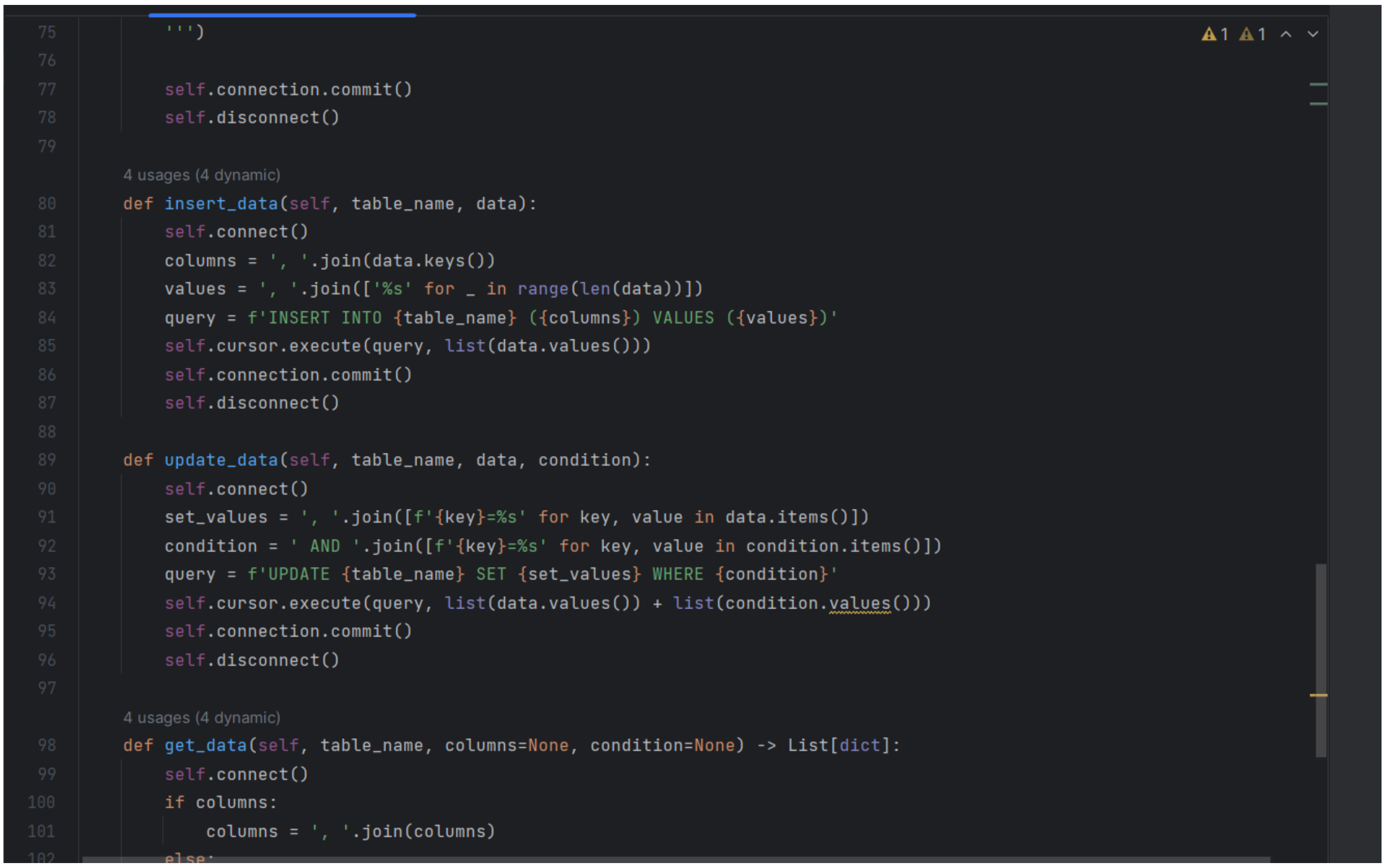
```python
import mysql.connector
from typing import List

2 usages
class DatabaseConnector:
    def __init__(self, host="localhost", user="root", password="Sushant@9546", database="CareerHub"):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None

    4 usages
    def connect(self):
        self.connection = mysql.connector.connect(
            host=self.host,
            user=self.user,
            password=self.password,
            database=self.database
        )
        self.cursor = self.connection.cursor()

    4 usages
    def disconnect(self):
        if self.connection:
            self.connection.close()

    1 usage
    def initialize_database(self):
```



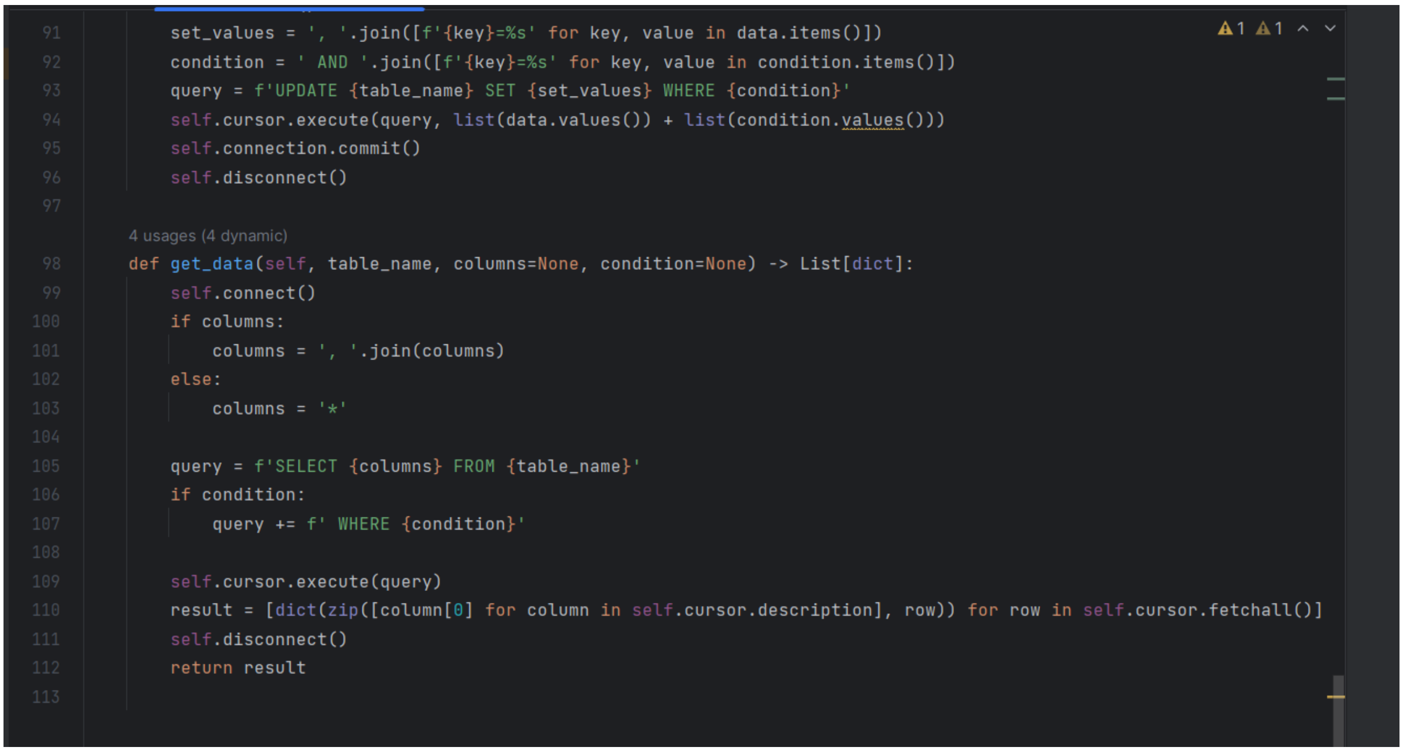
```
    def initialize_database(self):
        self.connect()

        # Create Jobs table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Jobs (
                JobID INT AUTO_INCREMENT PRIMARY KEY,
                CompanyID INT,
                JobTitle VARCHAR(255),
                JobDescription TEXT,
                JobLocation VARCHAR(255),
                Salary DECIMAL(10, 2),
                JobType VARCHAR(50),
                PostedDate DATETIME
            )
        ''')

        # Create Companies table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Companies (
                CompanyID INT AUTO_INCREMENT PRIMARY KEY,
                CompanyName VARCHAR(255),
                Location VARCHAR(255)
            )
        ''')

        # Create Applicants table
        self.cursor.execute('''
```



```
                Location VARCHAR(255)
            )
        ''')

        # Create Applicants table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applicants (
                ApplicantID INT AUTO_INCREMENT PRIMARY KEY,
                FirstName VARCHAR(255),
                LastName VARCHAR(255),
                Email VARCHAR(255),
                Phone VARCHAR(20),
                Resume VARCHAR(255)
            )
        ''')

        # Create Applications table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applications (
                ApplicationID INT AUTO_INCREMENT PRIMARY KEY,
                JobID INT,
                ApplicantID INT,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY(JobID) REFERENCES Jobs(JobID),
                FOREIGN KEY(ApplicantID) REFERENCES Applicants(ApplicantID)
            )
        ''')

        self.connection.commit()
```

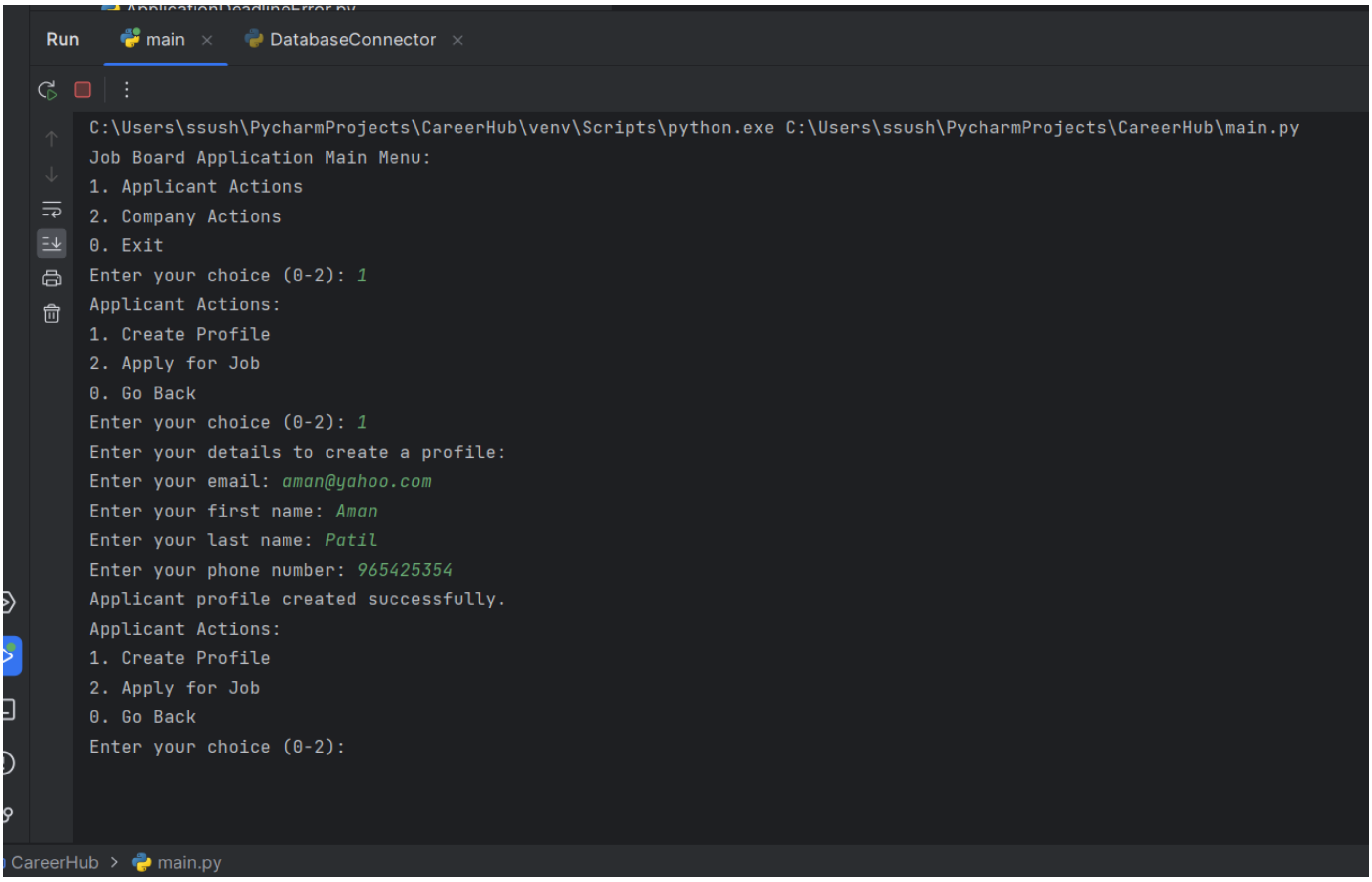


```
    ''')

    self.connection.commit()
    self.disconnect()

4 usages (4 dynamic)
def insert_data(self, table_name, data):
    self.connect()
    columns = ', '.join(data.keys())
    values = ', '.join(['%s' for _ in range(len(data))])
    query = f'INSERT INTO {table_name} ({columns}) VALUES ({values})'
    self.cursor.execute(query, list(data.values()))
    self.connection.commit()
    self.disconnect()

def update_data(self, table_name, data, condition):
    self.connect()
    set_values = ', '.join([f'{key}=%s' for key, value in data.items()])
    condition = ' AND '.join([f'{key}=%s' for key, value in condition.items()])
    query = f'UPDATE {table_name} SET {set_values} WHERE {condition}'
    self.cursor.execute(query, list(data.values()) + list(condition.values()))
    self.connection.commit()
    self.disconnect()

4 usages (4 dynamic)
def get_data(self, table_name, columns=None, condition=None) -> List[dict]:
    self.connect()
    if columns:
        columns = ', '.join(columns)
```



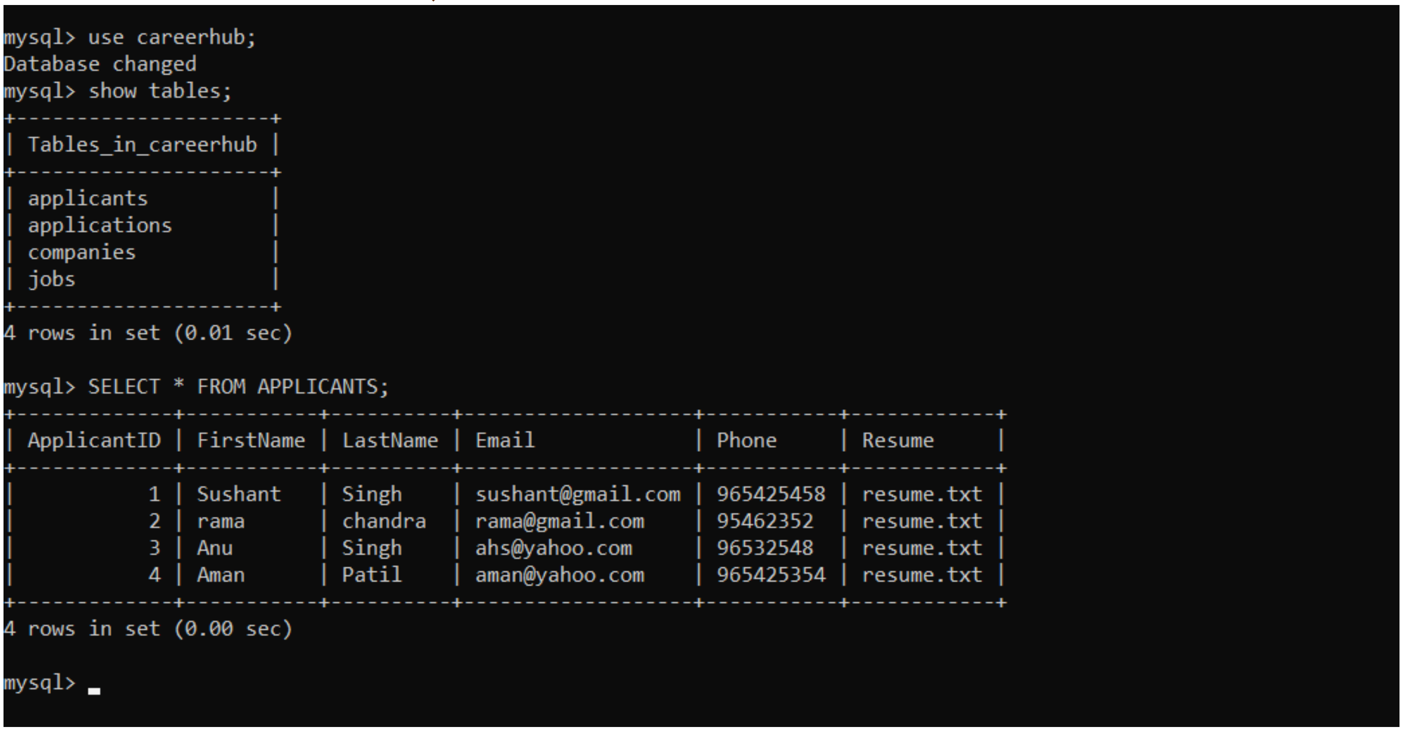
```
    set_values = ', '.join([f'{key}=%s' for key, value in data.items()])
    condition = ' AND '.join([f'{key}=%s' for key, value in condition.items()])
    query = f'UPDATE {table_name} SET {set_values} WHERE {condition}'
    self.cursor.execute(query, list(data.values()) + list(condition.values()))
    self.connection.commit()
    self.disconnect()

4 usages (4 dynamic)
def get_data(self, table_name, columns=None, condition=None) -> List[dict]:
    self.connect()
    if columns:
        columns = ', '.join(columns)
    else:
        columns = '*'

    query = f'SELECT {columns} FROM {table_name}'
    if condition:
        query += f' WHERE {condition}'

    self.cursor.execute(query)
    result = [dict(zip([column[0] for column in self.cursor.description], row)) for row in self.cursor.fetchall()]
    self.disconnect()
    return result
```

PROGRAM INPUT/OUTPUT



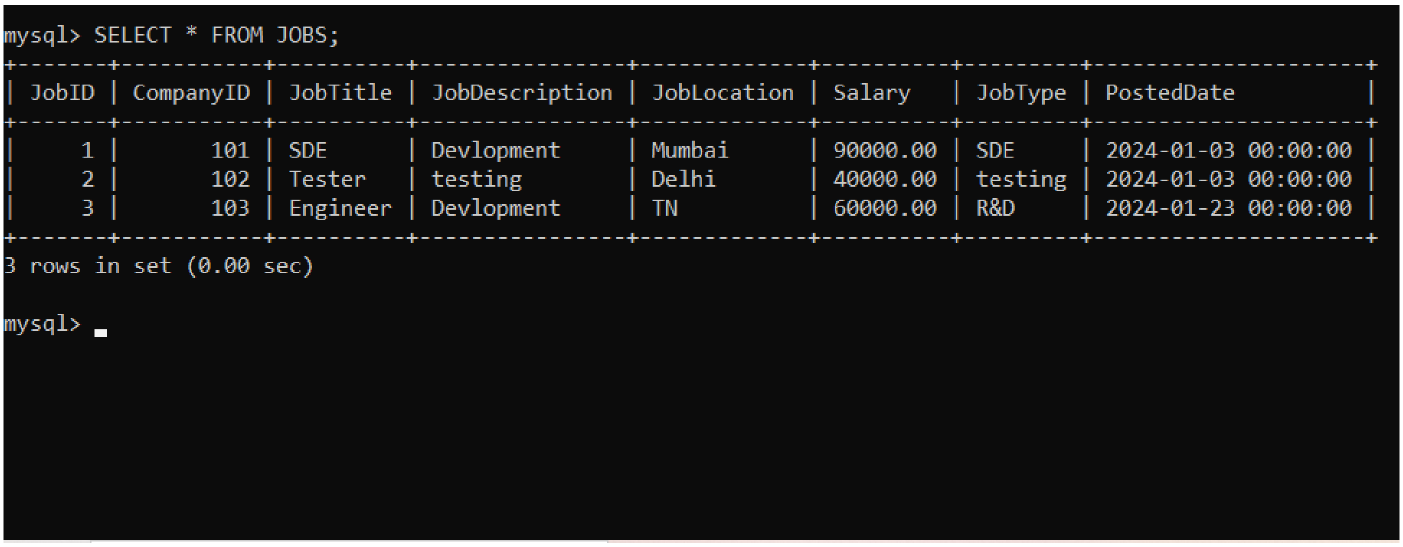
```
C:\Users\ssush\PycharmProjects\CareerHub\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\CareerHub\main.py
Job Board Application Main Menu:
1. Applicant Actions
2. Company Actions
0. Exit
Enter your choice (0-2): 1
Applicant Actions:
1. Create Profile
2. Apply for Job
0. Go Back
Enter your choice (0-2): 1
Enter your details to create a profile:
Enter your email: aman@yahoo.com
Enter your first name: Aman
Enter your last name: Patil
Enter your phone number: 965425354
Applicant profile created successfully.
Applicant Actions:
1. Create Profile
2. Apply for Job
0. Go Back
Enter your choice (0-2):
```

OUR GIVEN INPUT SUCCESSEFULL SAVEED IN OUR DATABASE

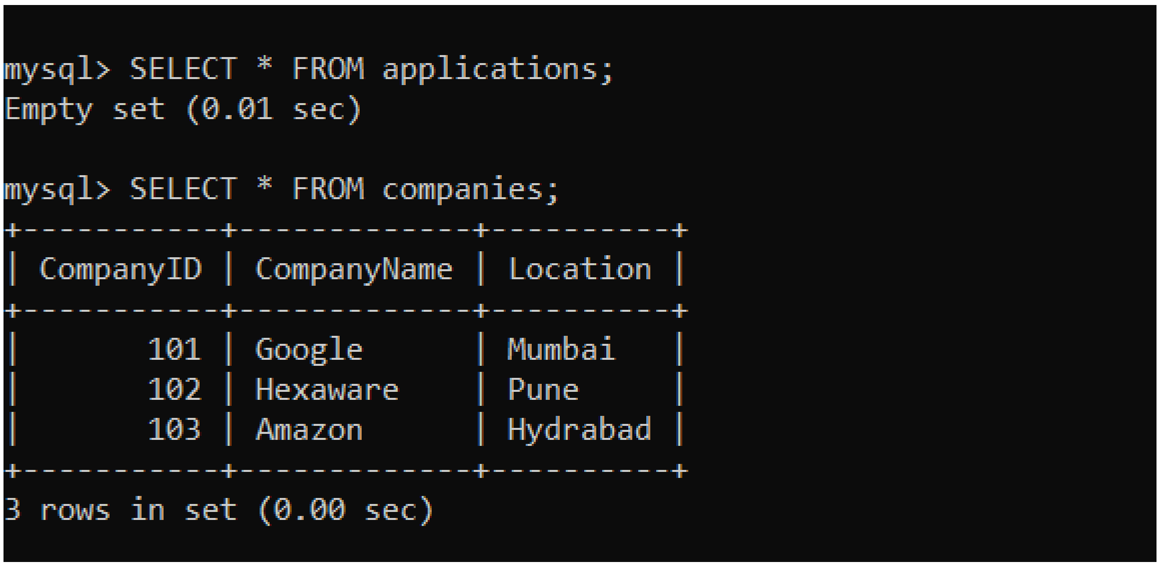


```
mysql> use careerhub;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_careerhub |
+---------------------+
| applicants          |
| applications        |
| companies           |
| jobs                |
+---------------------+
4 rows in set (0.01 sec)

mysql> SELECT * FROM APPLICANTS;
+-------------+-----------+----------+-------------------+-----------+------------+
| ApplicantID | FirstName | LastName | Email             | Phone     | Resume     |
+-------------+-----------+----------+-------------------+-----------+------------+
|           1 | Sushant   | Singh    | sushant@gmail.com | 965425458 | resume.txt |
|           2 | rama      | chandra  | rama@gmail.com    | 95462352  | resume.txt |
|           3 | Anu       | Singh    | ahs@yahoo.com     | 96532548  | resume.txt |
|           4 | Aman      | Patil    | aman@yahoo.com    | 965425354 | resume.txt |
+-------------+-----------+----------+-------------------+-----------+------------+
4 rows in set (0.00 sec)

mysql> _
```

OTHER TABLES SAVED IN DATABASE



```
mysql> SELECT * FROM JOBS;
+-------+-----------+----------+---------------+------------+----------+---------+---------------------+
| JobID | CompanyID | JobTitle | JobDescription | JobLocation | Salary   | JobType | PostedDate          |
+-------+-----------+----------+---------------+------------+----------+---------+---------------------+
|     1 |       101 | SDE      | Devlopment    | Mumbai     | 90000.00 | SDE     | 2024-01-03 00:00:00 |
|     2 |       102 | Tester   | testing       | Delhi      | 40000.00 | testing | 2024-01-03 00:00:00 |
|     3 |       103 | Engineer | Devlopment    | TN         | 60000.00 | R&D     | 2024-01-23 00:00:00 |
+-------+-----------+----------+---------------+------------+----------+---------+---------------------+
3 rows in set (0.00 sec)

mysql> 
```



```
mysql> SELECT * FROM applications;
Empty set (0.01 sec)

mysql> SELECT * FROM companies;
+-----------+-------------+----------+
| CompanyID | CompanyName | Location |
+-----------+-------------+----------+
|       101 | Google      | Mumbai   |
|       102 | Hexaware    | Pune     |
|       103 | Amazon      | Hydrabad |
+-----------+-------------+----------+
3 rows in set (0.00 sec)
```

***************THANKYOU************