PYTHON CODING ASSESSMENT-CAREER HUB

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.

CREATING TABLES-

```
create database careerhub;

use careerhub;

create table if not exists companies (

CompanyID int primary key,

CompanyName varchar(100),

Location varchar(100)

);

create table if not exists jobs (

JobID int primary key,

CompanyID int,

JobTitle varchar(100),

JobDescription text,

JobLocation varchar(100),

Salary decimal(10,2),

JobType varchar(50),

PostedDate datetime,

foreign key (CompanyID) references companies(CompanyID)

);

create table if not exists applicants (

ApplicantID int primary key,

FirstName varchar(100),
LastName varchar(100),
Email varchar(100),
Phone varchar(20),
Resume text
);
create table if not exists applications (
ApplicationID int primary key,
JobID int,
ApplicantID int,
ApplicationDate datetime,
CoverLetter text,
foreign key (JobID) references jobs(JobID),
foreign key (ApplicantID) references applicants(ApplicantID)
);
```

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

<u>JobListing Class:</u>

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.

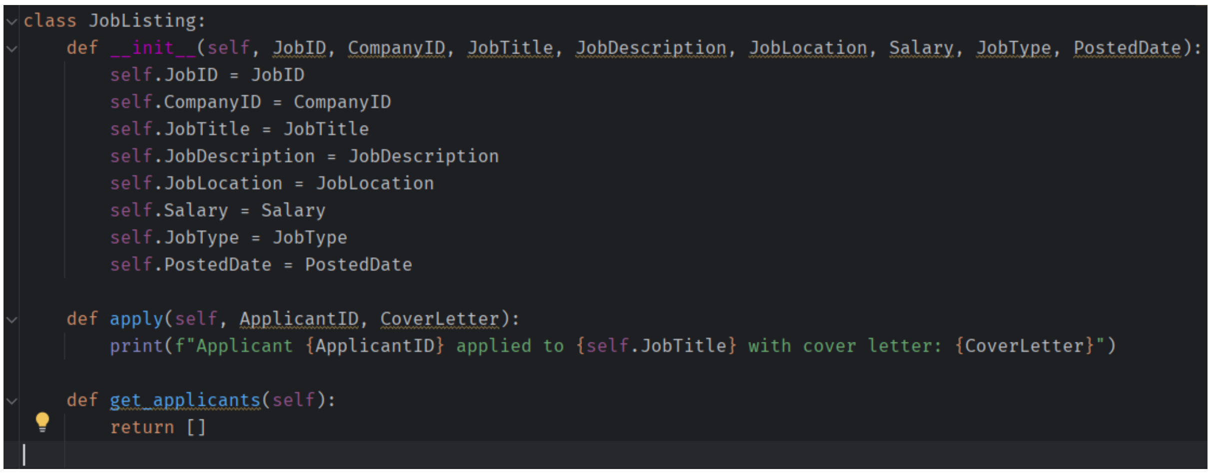
• Salary (decimal): The salary offered for the job.

• 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: Retrieves a list of applicants who have applied for the job.

```
class JobListing:
    def __init__(self, JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType,
PostedDate):
        self.JobID = JobID
        self.CompanyID = CompanyID
        self.JobTitle = JobTitle
        self.JobDescription = JobDescription
        self.JobLocation = JobLocation
        self.Salary = Salary
        self.JobType = JobType
        self.PostedDate = PostedDate

    def apply(self, ApplicantID, CoverLetter):
        print(f"Applicant {ApplicantID} applied to {self.JobTitle} with cover letter: {CoverLetter}")

    def get_applicants(self):
        return []
```



```
class JobListing:
    def __init__(self, JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType, PostedDate):
        self.JobID = JobID
        self.CompanyID = CompanyID
        self.JobTitle = JobTitle
        self.JobDescription = JobDescription
        self.JobLocation = JobLocation
        self.Salary = Salary
        self.JobType = JobType
        self.PostedDate = PostedDate

    def apply(self, ApplicantID, CoverLetter):
        print(f"Applicant {ApplicantID} applied to {self.JobTitle} with cover letter: {CoverLetter}")

    def get_applicants(self):
        return []
```

<u>Company Class:</u>

Attributes:

• CompanyID (int): A unique identifier for each company.

• CompanyName (string): The name of the hiring company.

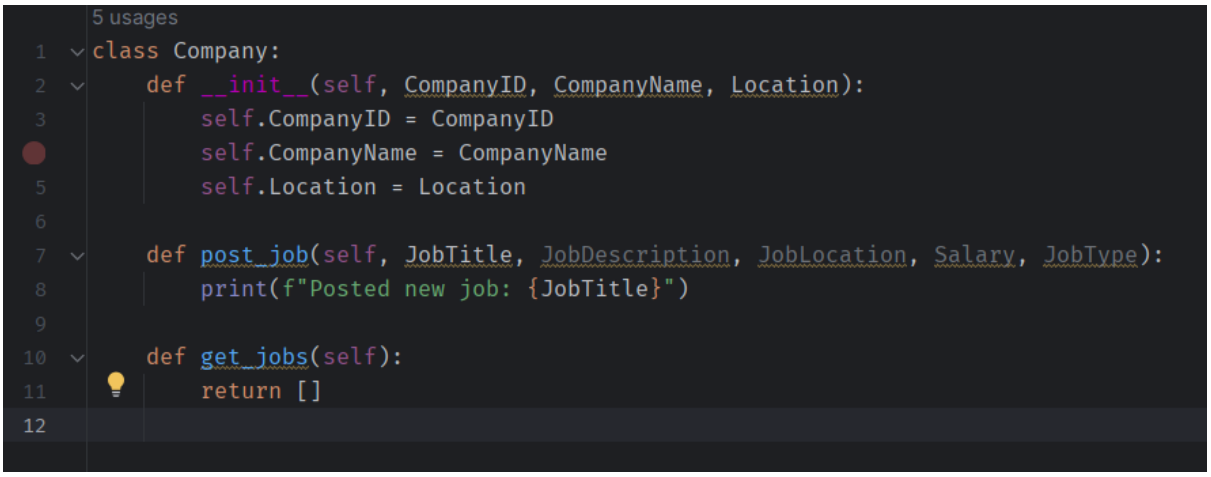
• Location (string): The location of the company.

Methods:

• PostJob(jobTitle: string, jobDescription: string, jobLocation: string, salary: decimal, jobType: string): Allows a company to post a new job listing.

• GetJobs(): List: Retrieves a list of job listings posted by the company.

```
class Company:
    def __init__(self, CompanyID, CompanyName, Location):
        self.CompanyID = CompanyID
        self.CompanyName = CompanyName
        self.Location = Location

    def post_job(self, JobTitle, JobDescription, JobLocation, Salary, JobType):
        print(f"Posted new job: {JobTitle}")

    def get_jobs(self):
        return []
```



```
5 usages
class Company:
    def __init__(self, CompanyID, CompanyName, Location):
        self.CompanyID = CompanyID
        self.CompanyName = CompanyName
        self.Location = Location

    def post_job(self, JobTitle, JobDescription, JobLocation, Salary, JobType):
        print(f"Posted new job: {JobTitle}")

    def get_jobs(self):
        return []

```

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.

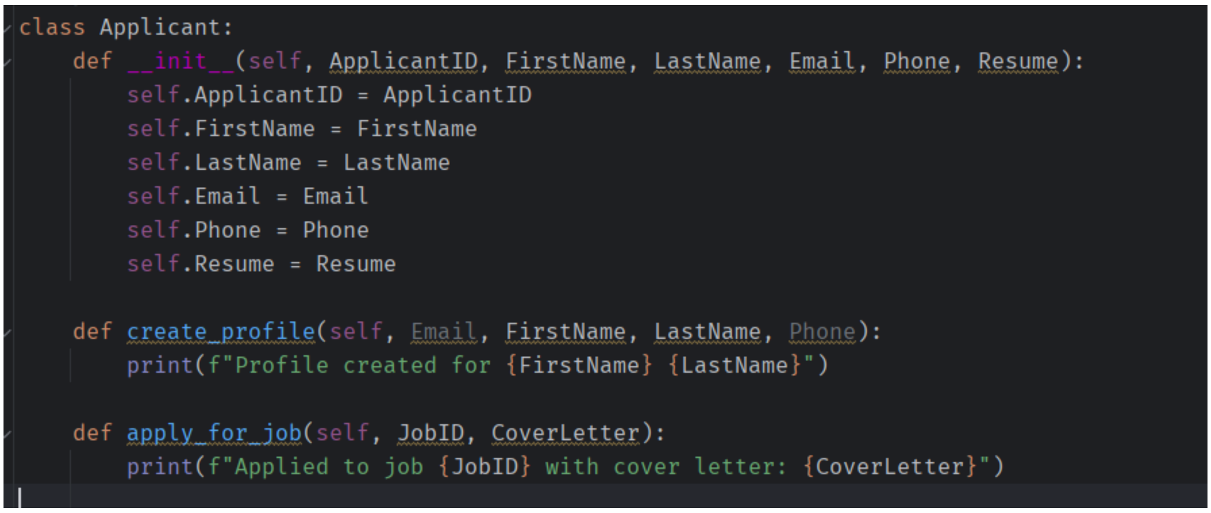
• Email (string): The email address of the applicant.

• 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. • ApplyForJob(jobID: int, coverLetter: string): Enables applicants to apply for a specific job listing.

```
class Applicant:
    def __init__(self, ApplicantID, FirstName, LastName, Email, Phone, Resume):
        self.ApplicantID = ApplicantID
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.Phone = Phone
        self.Resume = Resume

    def create_profile(self, Email, FirstName, LastName, Phone):
        print(f"Profile created for {FirstName} {LastName}")

    def apply_for_job(self, JobID, CoverLetter):
        print(f"Applied to job {JobID} with cover letter: {CoverLetter}")
```



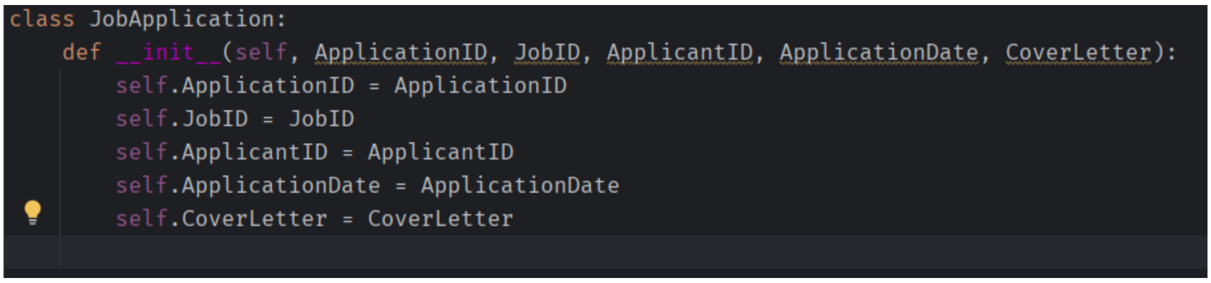
```
class Applicant:
    def __init__(self, ApplicantID, FirstName, LastName, Email, Phone, Resume):
        self.ApplicantID = ApplicantID
        self.FirstName = FirstName
        self.LastName = LastName
        self.Email = Email
        self.Phone = Phone
        self.Resume = Resume

    def create_profile(self, Email, FirstName, LastName, Phone):
        print(f"Profile created for {FirstName} {LastName}")

    def apply_for_job(self, JobID, CoverLetter):
        print(f"Applied to job {JobID} with cover letter: {CoverLetter}")
```

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.**

```
class JobApplication:
    def __init__(self, ApplicationID, JobID, ApplicantID, ApplicationDate, CoverLetter):
        self.ApplicationID = ApplicationID
        self.JobID = JobID
        self.ApplicantID = ApplicantID
        self.ApplicationDate = ApplicationDate
        self.CoverLetter = CoverLetter
```



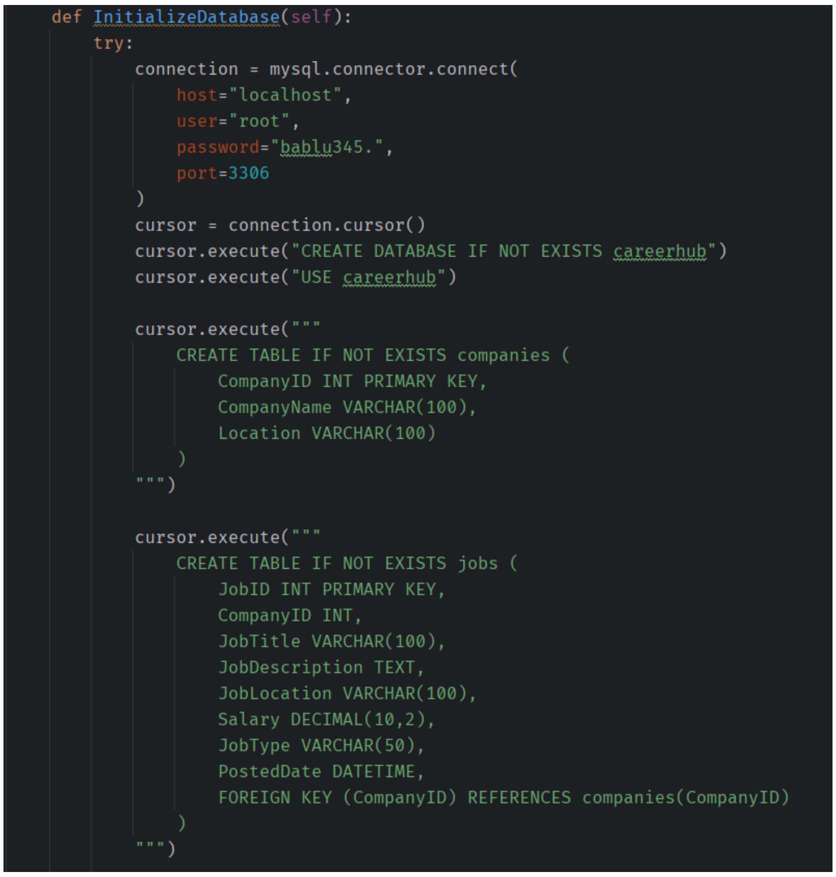
```
class JobApplication:
    def __init__(self, ApplicationID, JobID, ApplicantID, ApplicationDate, CoverLetter):
        self.ApplicationID = ApplicationID
        self.JobID = JobID
        self.ApplicantID = ApplicantID
        self.ApplicationDate = ApplicationDate
        self.CoverLetter = CoverLetter
```

2.DatabaseManager Class:

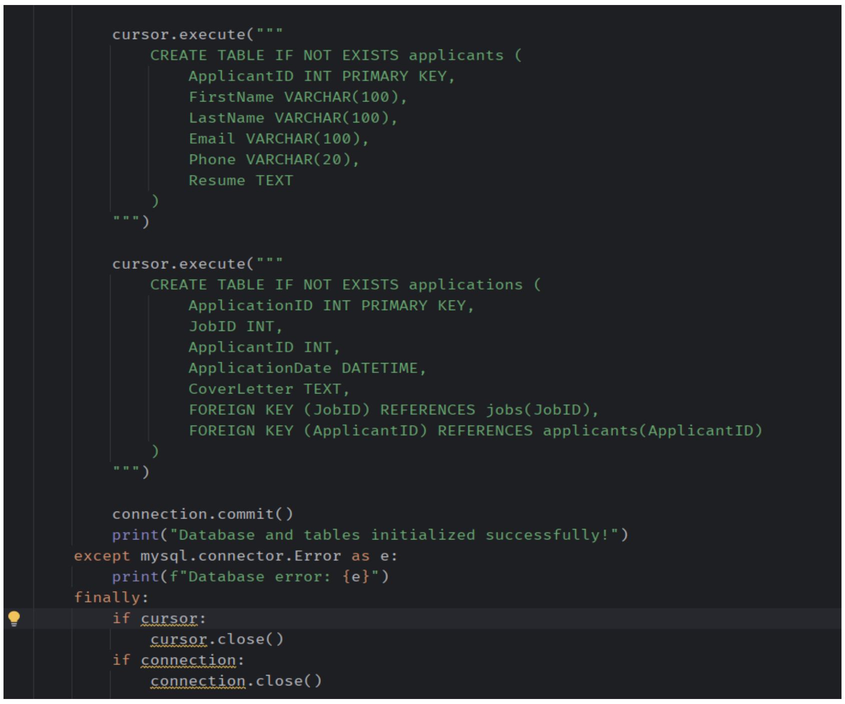
Methods:

- **InitializeDatabase(): Initializes the database schema and tables.**

```
def InitializeDatabase(self):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            port=3306
        )
        cursor = connection.cursor()
        cursor.execute("CREATE DATABASE IF NOT EXISTS careerhub")
        cursor.execute("USE careerhub")

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS companies (
                CompanyID INT PRIMARY KEY,
                CompanyName VARCHAR(100),
                Location VARCHAR(100)
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS jobs (
                JobID INT PRIMARY KEY,
                CompanyID INT,
                JobTitle VARCHAR(100),
                JobDescription TEXT,
                JobLocation VARCHAR(100),
                Salary DECIMAL(10,2),
                JobType VARCHAR(50),
                PostedDate DATETIME,
                FOREIGN KEY (CompanyID) REFERENCES companies(CompanyID)
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS applicants (
                ApplicantID INT PRIMARY KEY,
                FirstName VARCHAR(100),
                LastName VARCHAR(100),
                Email VARCHAR(100),
                Phone VARCHAR(20),
                Resume TEXT
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS applications (
                ApplicationID INT PRIMARY KEY,
                JobID INT,
                ApplicantID INT,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY (JobID) REFERENCES jobs(JobID),
                FOREIGN KEY (ApplicantID) REFERENCES applicants(ApplicantID)
            )
        """)

        connection.commit()
        print("Database and tables initialized successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```



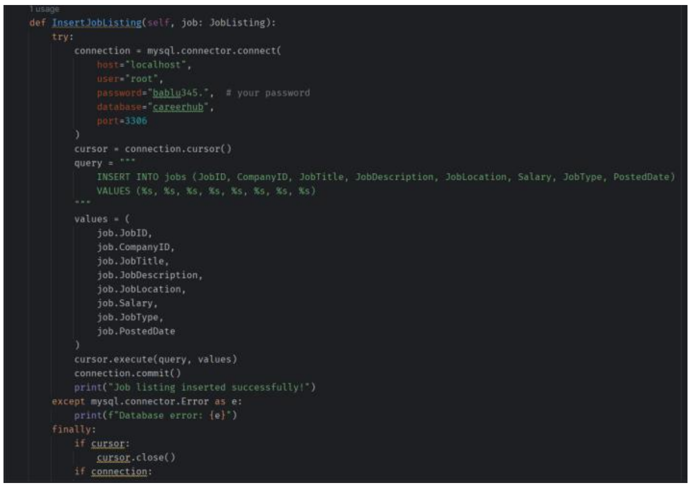
```python
def InitializeDatabase(self):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            port=3306
        )
        cursor = connection.cursor()
        cursor.execute("CREATE DATABASE IF NOT EXISTS careerhub")
        cursor.execute("USE careerhub")

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS companies (
                CompanyID INT PRIMARY KEY,
                CompanyName VARCHAR(100),
                Location VARCHAR(100)
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS jobs (
                JobID INT PRIMARY KEY,
                CompanyID INT,
                JobTitle VARCHAR(100),
                JobDescription TEXT,
                JobLocation VARCHAR(100),
                Salary DECIMAL(10,2),
                JobType VARCHAR(50),
                PostedDate DATETIME,
                FOREIGN KEY (CompanyID) REFERENCES companies(CompanyID)
            )
        """)
```



```python
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS applicants (
                ApplicantID INT PRIMARY KEY,
                FirstName VARCHAR(100),
                LastName VARCHAR(100),
                Email VARCHAR(100),
                Phone VARCHAR(20),
                Resume TEXT
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS applications (
                ApplicationID INT PRIMARY KEY,
                JobID INT,
                ApplicantID INT,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY (JobID) REFERENCES jobs(JobID),
                FOREIGN KEY (ApplicantID) REFERENCES applicants(ApplicantID)
            )
        """)

        connection.commit()
        print("Database and tables initialized successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

• InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.

```python
def InsertJobListing(self, job: JobListing):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",  # your password
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO jobs (JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary,
JobType, PostedDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            job.JobID,
            job.CompanyID,
            job.JobTitle,
            job.JobDescription,
            job.JobLocation,
            job.Salary,
            job.JobType,
            job.PostedDate
        )
        cursor.execute(query, values)
        connection.commit()
        print("Job listing inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```



```
1 usage
def InsertJobListing(self, job: JobListing):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",  # your password
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO jobs (JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType, PostedDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            job.JobID,
            job.CompanyID,
            job.JobTitle,
            job.JobDescription,
            job.JobLocation,
            job.Salary,
            job.JobType,
            job.PostedDate
        )
        cursor.execute(query, values)
        connection.commit()
        print("Job listing inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
```

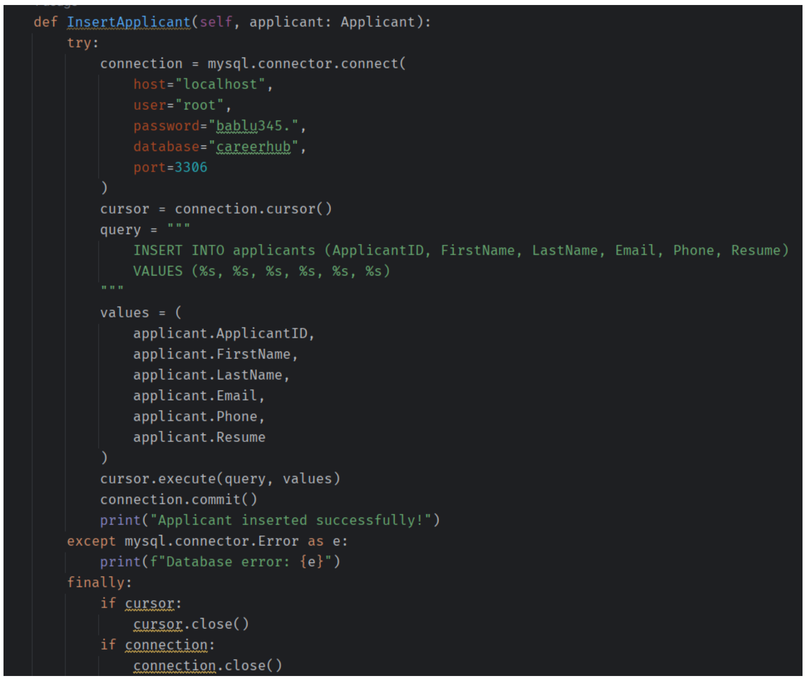
• InsertCompany(company: Company): Inserts a new company into the "Companies" table.

```
def InsertCompany(self, company: Company):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO companies (CompanyID, CompanyName, Location)
            VALUES (%s, %s, %s)
        """
        values = (company.CompanyID, company.CompanyName, company.Location)
        cursor.execute(query, values)
        connection.commit()
        print("Company inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

```
def InsertCompany(self, company: Company):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO companies (CompanyID, CompanyName, Location)
            VALUES (%s, %s, %s)
        """
        values = (company.CompanyID, company.CompanyName, company.Location)
        cursor.execute(query, values)
        connection.commit()
        print("Company inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

• InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.

```
def InsertApplicant(self, applicant: Applicant):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO applicants (ApplicantID, FirstName, LastName, Email, Phone, Resume)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        values = (
            applicant.ApplicantID,
            applicant.FirstName,
            applicant.LastName,
            applicant.Email,
            applicant.Phone,
            applicant.Resume
        )
        cursor.execute(query, values)
        connection.commit()
        print("Applicant inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

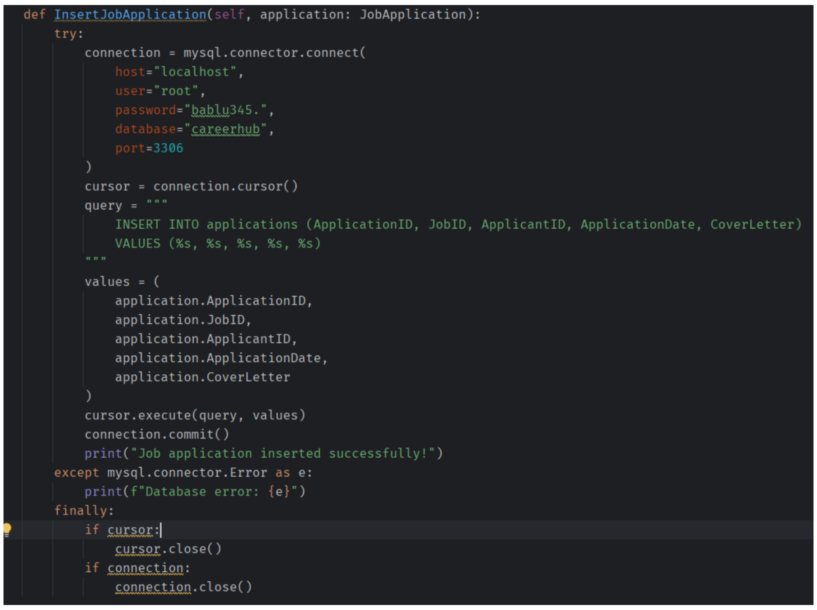


```
def InsertApplicant(self, applicant: Applicant):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO applicants (ApplicantID, FirstName, LastName, Email, Phone, Resume)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        values = (
            applicant.ApplicantID,
            applicant.FirstName,
            applicant.LastName,
            applicant.Email,
            applicant.Phone,
            applicant.Resume
        )
        cursor.execute(query, values)
        connection.commit()
        print("Applicant inserted successfully!")
    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

• InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications" table.

```
def InsertJobApplication(self, application: JobApplication):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            INSERT INTO applications (ApplicationID, JobID, ApplicantID, ApplicationDate,
CoverLetter)
            VALUES (%s, %s, %s, %s, %s)
        """
        values = (
            application.ApplicationID,
            application.JobID,
            application.ApplicantID,
            application.ApplicationDate,
            application.CoverLetter
        )
        cursor.execute(query, values)
        connection.commit()
```

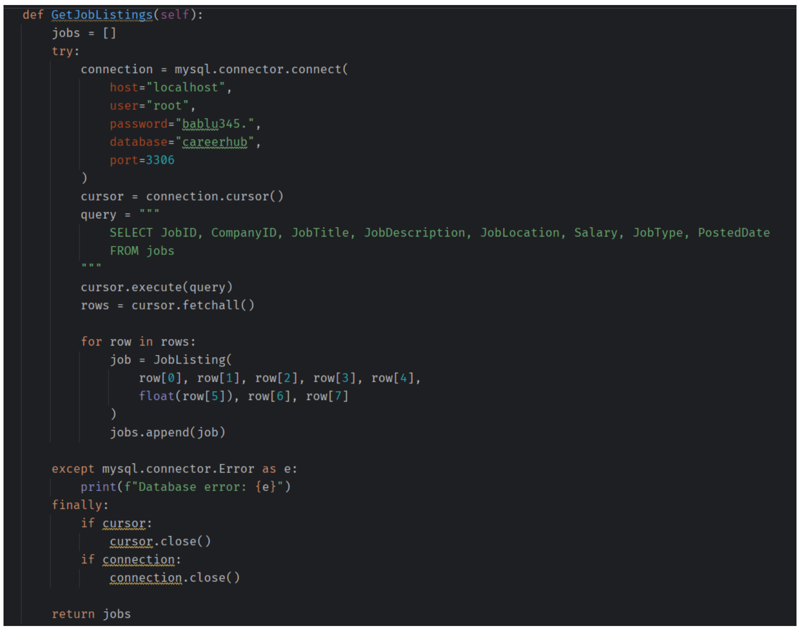
```
            print("Job application inserted successfully!")
        except mysql.connector.Error as e:
            print(f"Database error: {e}")
        finally:
            if cursor:
                cursor.close()
            if connection:
                connection.close()
```



```
    def InsertJobApplication(self, application: JobApplication):
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="bablu345.",
                database="careerhub",
                port=3306
            )
            cursor = connection.cursor()
            query = """
                INSERT INTO applications (ApplicationID, JobID, ApplicantID, ApplicationDate, CoverLetter)
                VALUES (%s, %s, %s, %s, %s)
            """
            values = (
                application.ApplicationID,
                application.JobID,
                application.ApplicantID,
                application.ApplicationDate,
                application.CoverLetter
            )
            cursor.execute(query, values)
            connection.commit()
            print("Job application inserted successfully!")
        except mysql.connector.Error as e:
            print(f"Database error: {e}")
        finally:
            if cursor:
                cursor.close()
            if connection:
                connection.close()
```

• **GetJobListings(): List: Retrieves a list of all job listings.**

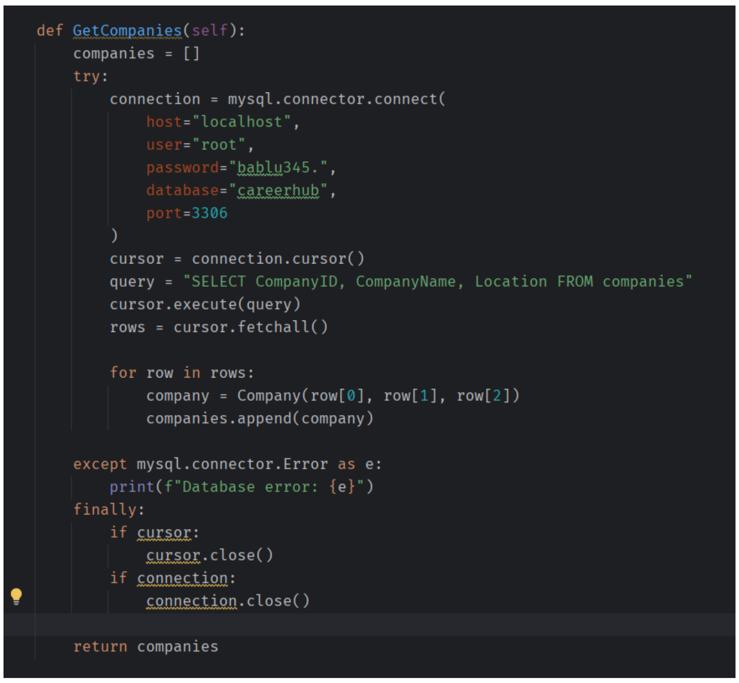
```
def GetJobListings(self):
    jobs = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            SELECT JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType,
PostedDate
            FROM jobs
        """
        cursor.execute(query)
        rows = cursor.fetchall()
        for row in rows:
            job = JobListing(
                row[0], row[1], row[2], row[3], row[4],
                float(row[5]), row[6], row[7]
            )
            jobs.append(job)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return jobs
```



```
def GetJobListings(self):
    jobs = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            SELECT JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType, PostedDate
            FROM jobs
        """
        cursor.execute(query)
        rows = cursor.fetchall()

        for row in rows:
            job = JobListing(
                row[0], row[1], row[2], row[3], row[4],
                float(row[5]), row[6], row[7]
            )
            jobs.append(job)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return jobs
```

- **GetCompanies(): List: Retrieves a list of all companies.**

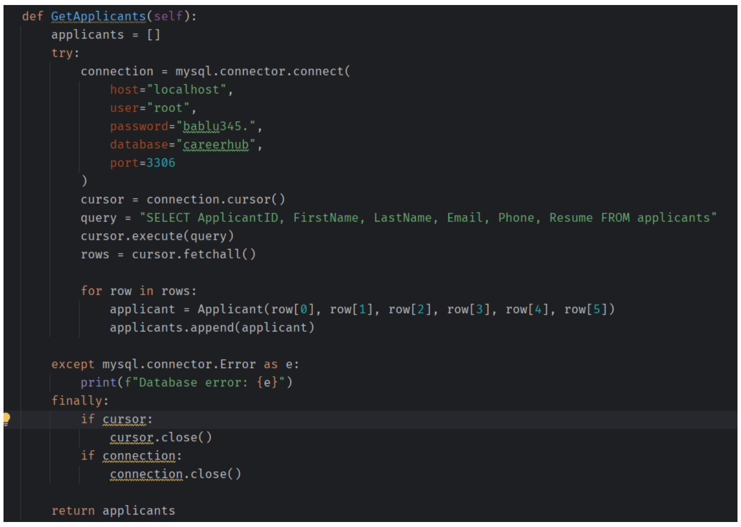
```
def GetCompanies(self):
    companies = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = "SELECT CompanyID, CompanyName, Location FROM companies"
        cursor.execute(query)
        rows = cursor.fetchall()

        for row in rows:
            company = Company(row[0], row[1], row[2])
            companies.append(company)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return companies
```



```
def GetCompanies(self):
    companies = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = "SELECT CompanyID, CompanyName, Location FROM companies"
        cursor.execute(query)
        rows = cursor.fetchall()

        for row in rows:
            company = Company(row[0], row[1], row[2])
            companies.append(company)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return companies
```

- **GetApplicants(): List: Retrieves a list of all applicants.**

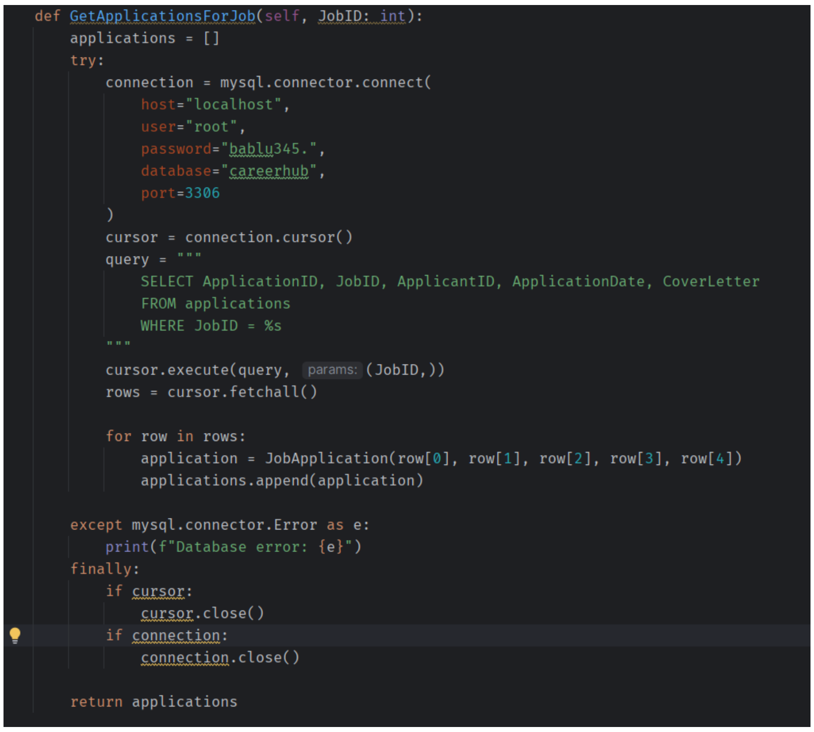
```
def GetApplicants(self):
    applicants = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = "SELECT ApplicantID, FirstName, LastName, Email, Phone, Resume FROM
applicants"
        cursor.execute(query)
        rows = cursor.fetchall()

        for row in rows:
            applicant = Applicant(row[0], row[1], row[2], row[3], row[4], row[5])
            applicants.append(applicant)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return applicants
```



```
def GetApplicants(self):
    applicants = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = "SELECT ApplicantID, FirstName, LastName, Email, Phone, Resume FROM applicants"
        cursor.execute(query)
        rows = cursor.fetchall()

        for row in rows:
            applicant = Applicant(row[0], row[1], row[2], row[3], row[4], row[5])
            applicants.append(applicant)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return applicants
```

• GetApplicationsForJob(jobID: int): List: Retrieves a list of job applications for a specific job listing.

```
def GetApplicationsForJob(self, JobID: int):
    applications = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            SELECT ApplicationID, JobID, ApplicantID, ApplicationDate, CoverLetter
            FROM applications
            WHERE JobID = %s
        """
        cursor.execute(query, (JobID,))
        rows = cursor.fetchall()

        for row in rows:
            application = JobApplication(row[0], row[1], row[2], row[3], row[4])
            applications.append(application)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return applications
```

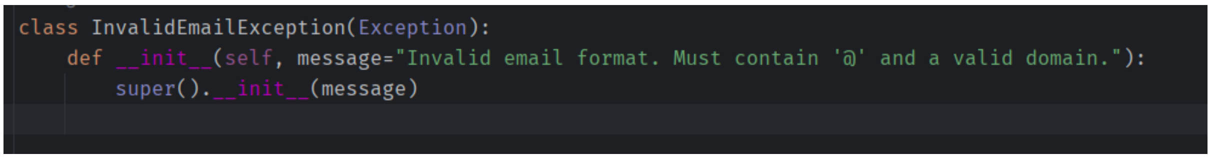


```
def GetApplicationsForJob(self, JobID: int):
    applications = []
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="bablu345.",
            database="careerhub",
            port=3306
        )
        cursor = connection.cursor()
        query = """
            SELECT ApplicationID, JobID, ApplicantID, ApplicationDate, CoverLetter
            FROM applications
            WHERE JobID = %s
        """
        cursor.execute(query, params: (JobID,))
        rows = cursor.fetchall()

        for row in rows:
            application = JobApplication(row[0], row[1], row[2], row[3], row[4])
            applications.append(application)

    except mysql.connector.Error as e:
        print(f"Database error: {e}")
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()

    return applications
```

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.

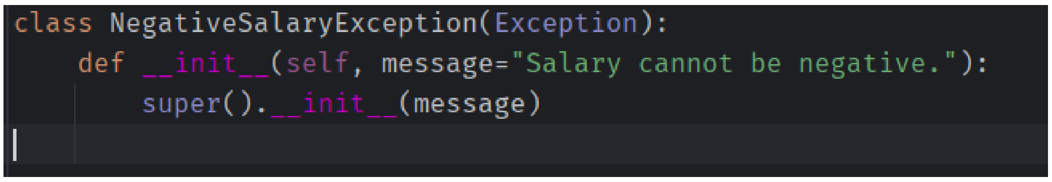
```
class InvalidEmailException(Exception):
    def __init__(self, message="Invalid email format. Must contain '@' and a valid domain."):
        super().__init__(message)
```



```
class InvalidEmailException(Exception):
    def __init__(self, message="Invalid email format. Must contain '@' and a valid domain."):
        super().__init__(message)
```

• 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 display an error message, indicating the problematic job listings.

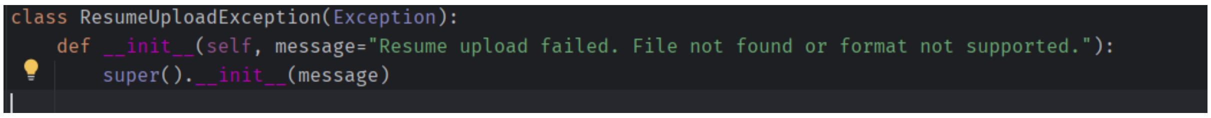
```
class NegativeSalaryException(Exception):
    def __init__(self, message="Salary cannot be negative."):
        super().__init__(message)
```



```
class NegativeSalaryException(Exception):
    def __init__(self, message="Salary cannot be negative."):
        super().__init__(message)
```

• 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.

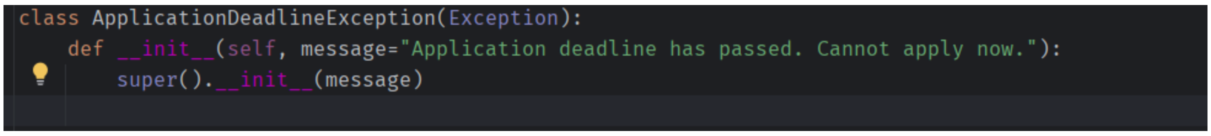
```
class ResumeUploadException(Exception):
    def __init__(self, message="Resume upload failed. File not found or format not
supported."):
        super().__init__(message)
```



```
class ResumeUploadException(Exception):
    def __init__(self, message="Resume upload failed. File not found or format not supported."):
        super().__init__(message)
```

• 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.

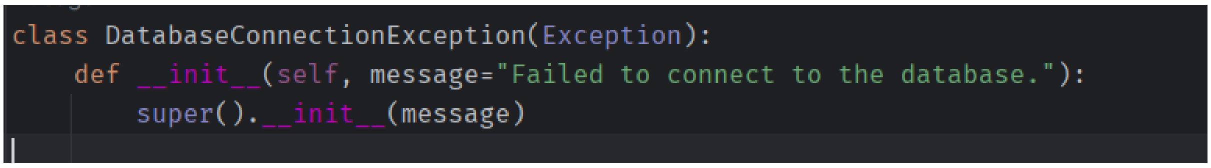
```
class ApplicationDeadlineException(Exception):
  def __init__(self, message="Application deadline has passed. Cannot apply now."):
    super().__init__(message)
```



```
class ApplicationDeadlineException(Exception):
    def __init__(self, message="Application deadline has passed. Cannot apply now."):
        super().__init__(message)
```

• 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.

```
class DatabaseConnectionException(Exception):
  def __init__(self, message="Failed to connect to the database."):
    super().__init__(message)
```



```
class DatabaseConnectionException(Exception):
    def __init__(self, message="Failed to connect to the database."):
        super().__init__(message)
```

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.**

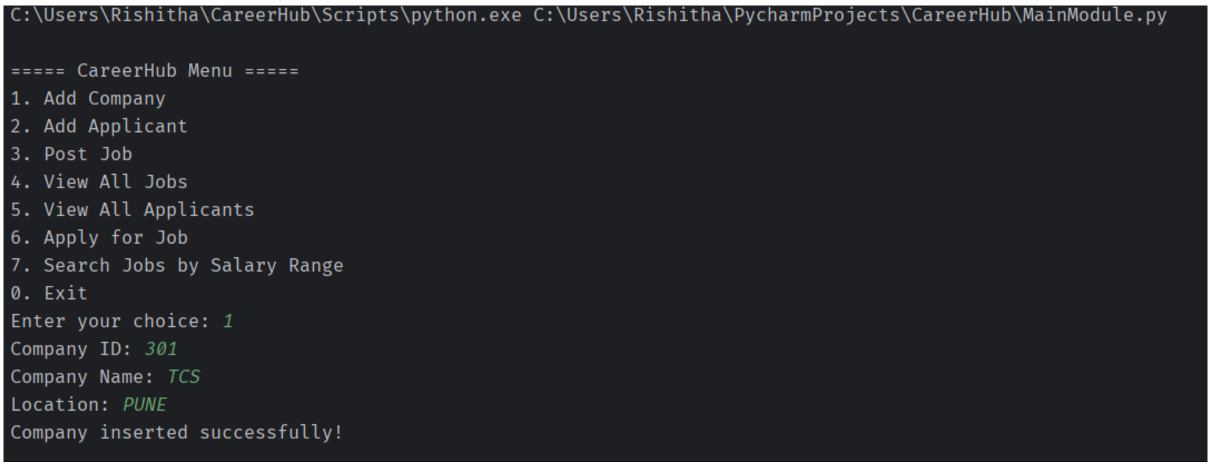
```
from service.DatabaseManager import DatabaseManager
from entity.Company import Company
from entity.Applicant import Applicant
from entity.JobListing import JobListing
from entity.JobApplication import JobApplication

from exception.InvalidEmailException import InvalidEmailException
from exception.NegativeSalaryException import NegativeSalaryException
from exception.ResumeUploadException import ResumeUploadException
from exception.ApplicationDeadlineException import ApplicationDeadlineException
from exception.DatabaseConnectionException import DatabaseConnectionException

from datetime import datetime
import os

def validate_email(email):
    if "@" not in email or "." not in email.split("@")[-1]:
        raise InvalidEmailException()

def check_resume(file_path):
    if not os.path.exists(file_path):
        raise ResumeUploadException("Resume file not found.")
    if not file_path.endswith((".pdf", ".docx", ".txt")):
        raise ResumeUploadException("Unsupported resume file format.")

def main():
    db = DatabaseManager()

    while True:
        print("\n===== CareerHub Menu =====")
        print("1. Add Company")
        print("2. Add Applicant")
        print("3. Post Job")
        print("4. View All Jobs")
        print("5. View All Applicants")
        print("6. Apply for Job")
        print("7. Search Jobs by Salary Range")
        print("0. Exit")
        choice = input("Enter your choice: ")
```

```
if choice == '1':
    try:
        cid = int(input("Company ID: "))
        name = input("Company Name: ")
        loc = input("Location: ")
        company = Company(cid, name, loc)
        db.InsertCompany(company)
    except Exception as e:
        print(e)

elif choice == '2':
    try:
        aid = int(input("Applicant ID: "))
        fname = input("First Name: ")
        lname = input("Last Name: ")
        email = input("Email: ")
        validate_email(email)
        phone = input("Phone: ")
        resume = input("Resume text or path: ")
        # check_resume(resume)  # Enable if you want to validate file
        applicant = Applicant(aid, fname, lname, email, phone, resume)
        db.InsertApplicant(applicant)
    except (InvalidEmailException, ResumeUploadException) as e:
        print(f"Validation Error: {e}")
    except Exception as e:
        print(e)

elif choice == '3':
    try:
        jid = int(input("Job ID: "))
        cid = int(input("Company ID: "))
        title = input("Job Title: ")
        desc = input("Description: ")
        loc = input("Location: ")
        salary = float(input("Salary: "))
        if salary < 0:
            raise NegativeSalaryException()
        jtype = input("Job Type: ")
        date_str = input("Posted Date (YYYY-MM-DD HH:MM:SS): ")
        posted_date = datetime.strptime(date_str, "%Y-%m-%d %H:%M:%S")
        job = JobListing(jid, cid, title, desc, loc, salary, jtype, posted_date)
        db.InsertJobListing(job)
    except NegativeSalaryException as e:
        print(f"Salary Error: {e}")
    except ValueError:
        print("Invalid date format!")
    except Exception as e:
        print(e)

elif choice == '4':
    jobs = db.GetJobListings()
    print("\n--- Job Listings ---")
    for j in jobs:
        print(f"{j.JobID} - {j.JobTitle} | {j.JobLocation} | ₹{j.Salary}")

elif choice == '5':
    applicants = db.GetApplicants()
    print("\n--- Applicants ---")
    for a in applicants:
        print(f"{a.ApplicantID} - {a.FirstName} {a.LastName} | {a.Email}")

elif choice == '6':
    try:
        app_id = int(input("Application ID: "))
        job_id = int(input("Job ID: "))
        applicant_id = int(input("Applicant ID: "))
        cover = input("Cover Letter: ")
        date_now = datetime.now()
        # Optional: check_deadline(date_now) — implement if using real deadlines
        application = JobApplication(app_id, job_id, applicant_id, date_now, cover)
        db.InsertJobApplication(application)
    except Exception as e:
        print(e)

elif choice == '7':
    try:
        min_salary = float(input("Enter minimum salary: "))
        max_salary = float(input("Enter maximum salary: "))
        results = db.GetJobsBySalaryRange(min_salary, max_salary)
        print("\n--- Jobs in Salary Range ---")
        for job in results:
            print(f"{job.JobTitle} | ₹{job.Salary} | {job.JobLocation}")
    except Exception as e:
        print(f"Error: {e}")


elif choice == '0':
```

```
            print("Exiting CareerHub. Goodbye!")
            break

        else:
            print("Invalid choice. Please try again.")

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

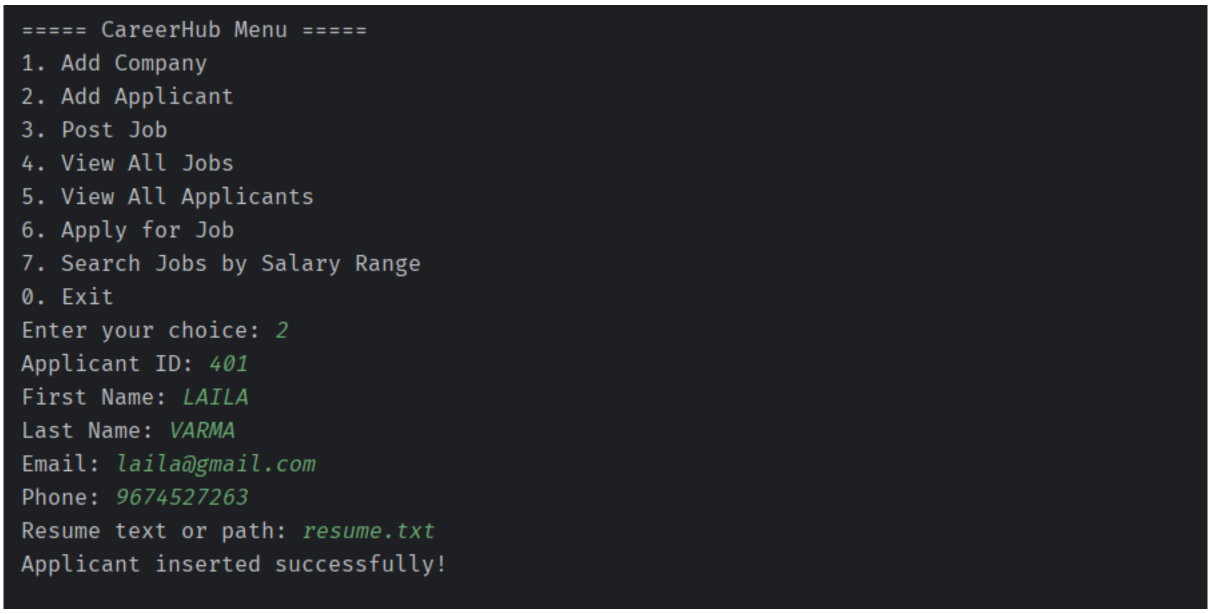
OUTPUT SCREENSHOTS

- **ADDING COMPANY**



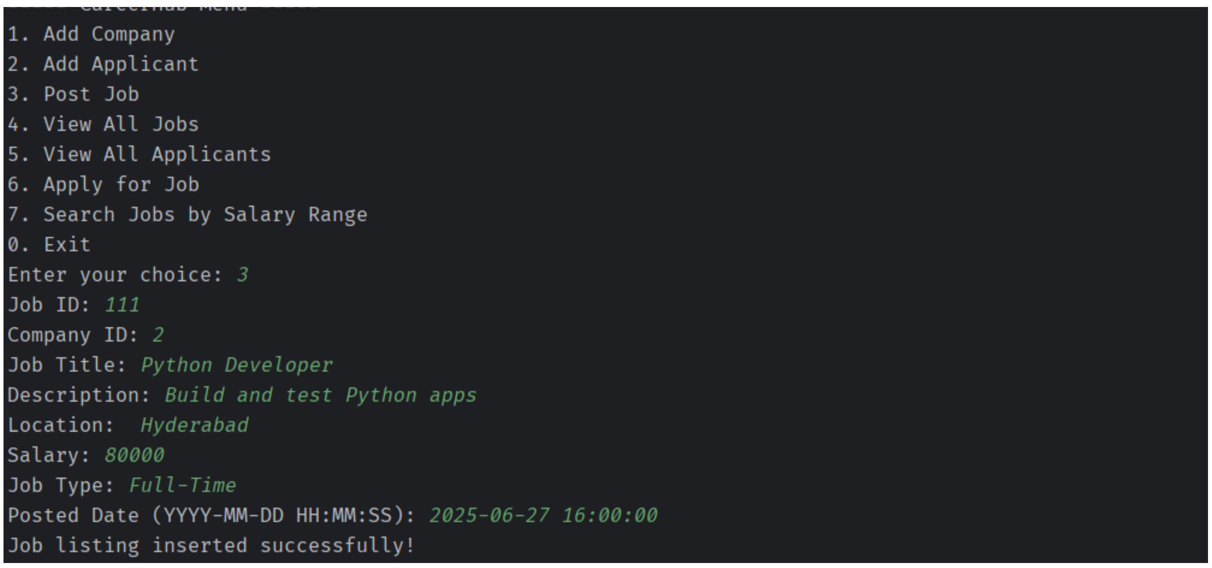
```
C:\Users\Rishitha\CareerHub\Scripts\python.exe C:\Users\Rishitha\PycharmProjects\CareerHub\MainModule.py

===== CareerHub Menu =====
1. Add Company
2. Add Applicant
3. Post Job
4. View All Jobs
5. View All Applicants
6. Apply for Job
7. Search Jobs by Salary Range
0. Exit
Enter your choice: 1
Company ID: 301
Company Name: TCS
Location: PUNE
Company inserted successfully!
```

- **ADDING APPILICANT**



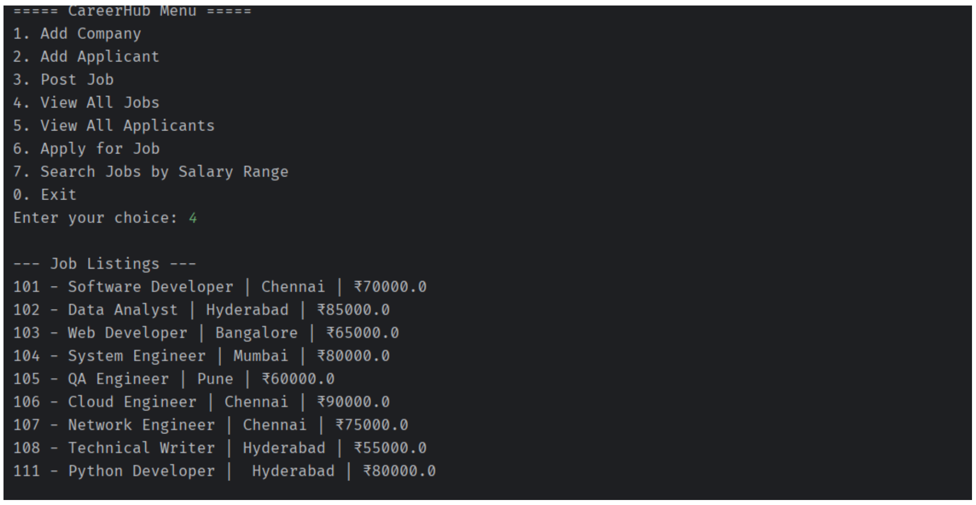
```
===== CareerHub Menu =====
1. Add Company
2. Add Applicant
3. Post Job
4. View All Jobs
5. View All Applicants
6. Apply for Job
7. Search Jobs by Salary Range
0. Exit
Enter your choice: 2
Applicant ID: 401
First Name: LAILA
Last Name: VARMA
Email: laila@gmail.com
Phone: 9674527263
Resume text or path: resume.txt
Applicant inserted successfully!
```

- **POST JOB**



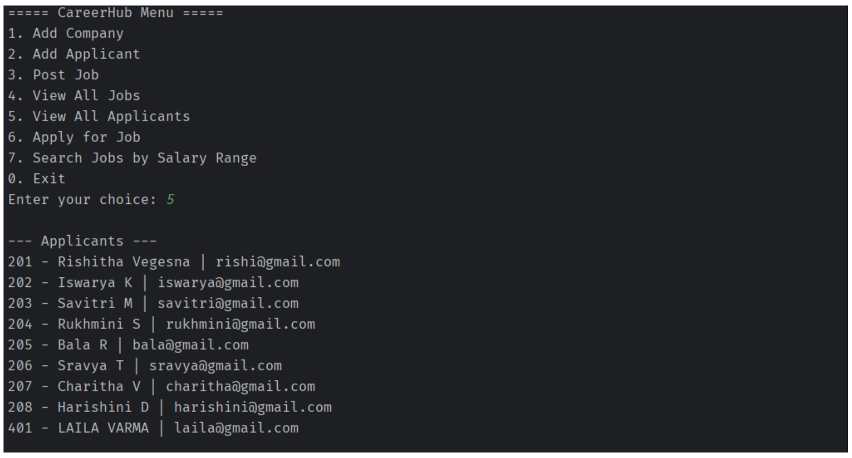
```
1. Add Company
2. Add Applicant
3. Post Job
4. View All Jobs
5. View All Applicants
6. Apply for Job
7. Search Jobs by Salary Range
0. Exit
Enter your choice: 3
Job ID: 111
Company ID: 2
Job Title: Python Developer
Description: Build and test Python apps
Location:  Hyderabad
Salary: 80000
Job Type: Full-Time
Posted Date (YYYY-MM-DD HH:MM:SS): 2025-06-27 16:00:00
Job listing inserted successfully!
```

- **VIEW ALL JOBS**



```
===== CareerHub Menu =====
1. Add Company
2. Add Applicant
3. Post Job
4. View All Jobs
5. View All Applicants
6. Apply for Job
7. Search Jobs by Salary Range
0. Exit
Enter your choice: 4

--- Job Listings ---
101 - Software Developer | Chennai | ₹70000.0
102 - Data Analyst | Hyderabad | ₹85000.0
103 - Web Developer | Bangalore | ₹65000.0
104 - System Engineer | Mumbai | ₹80000.0
105 - QA Engineer | Pune | ₹60000.0
106 - Cloud Engineer | Chennai | ₹90000.0
107 - Network Engineer | Chennai | ₹75000.0
108 - Technical Writer | Hyderabad | ₹55000.0
111 - Python Developer |  Hyderabad | ₹80000.0
```

- **APPLY FOR JOB**



```
===== CareerHub Menu =====
1. Add Company
2. Add Applicant
3. Post Job
4. View All Jobs
5. View All Applicants
6. Apply for Job
7. Search Jobs by Salary Range
0. Exit
Enter your choice: 5

--- Applicants ---
201 - Rishitha Vegesna | rishi@gmail.com
202 - Iswarya K | iswarya@gmail.com
203 - Savitri M | savitri@gmail.com
204 - Rukhmini S | rukhmini@gmail.com
205 - Bala R | bala@gmail.com
206 - Sravya T | sravya@gmail.com
207 - Charitha V | charitha@gmail.com
208 - Harishini D | harishini@gmail.com
401 - LAILA VARMA | laila@gmail.com
```