NAME:THILAK RAAJ R A

TOPIC:CAREER HUB

CODING CHALLENGE

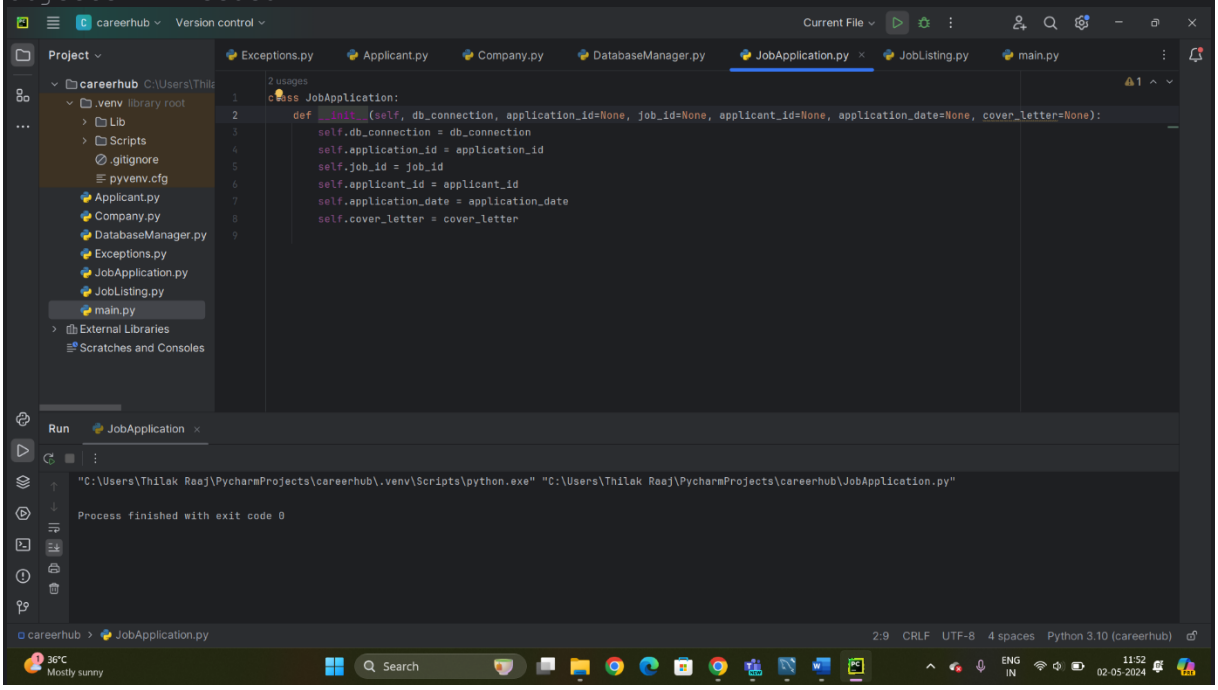
Create SQL Schema from the application, use the class attributes for table column names. 1.Create and implement the mentioned class and the structure in your application. JobListing Class: Attributes: • JobID (int): A unique identifier for each job listing. • CompanyID (int): A reference to the company 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: Retrieves a list of applicants who have applied for the job. 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: Retrieves a list of job listings posted by the company. 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. 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.

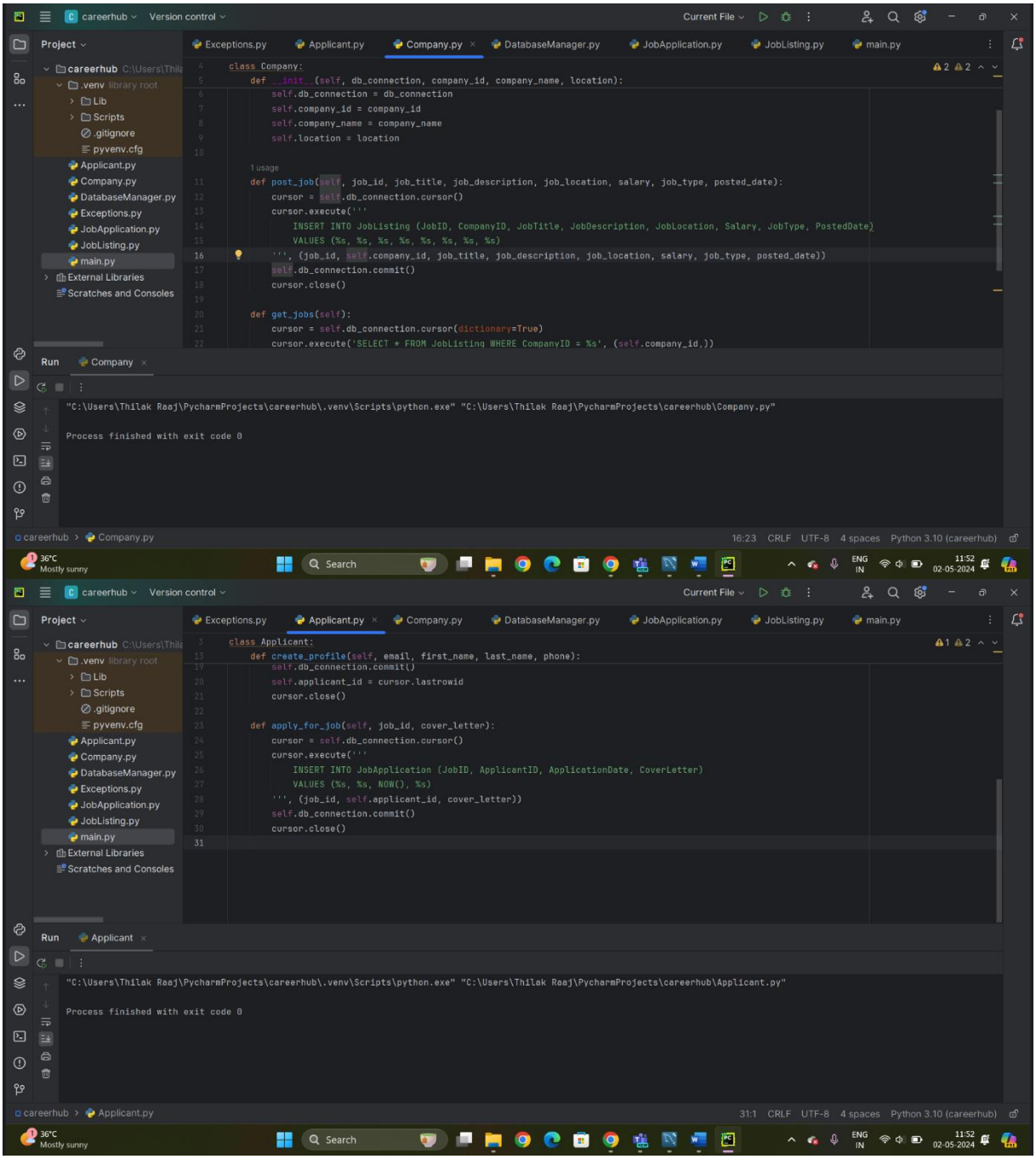
CODE:

```
import mysql.connector

class Applicant:
    def __init__(self, db_connection, applicant_id=None, first_name=None,
last_name=None, email=None, phone=None, resume=None):
        self.db_connection = db_connection
        self.applicant_id = applicant_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.resume = resume

    def create_profile(self, email, first_name, last_name, phone):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO Applicant (FirstName, LastName, Email, Phone,
Resume)
            VALUES (%s, %s, %s, %s, %s)
        ''', (first_name, last_name, email, phone, self.resume))
        self.db_connection.commit()
        self.applicant_id = cursor.lastrowid
        cursor.close()
```

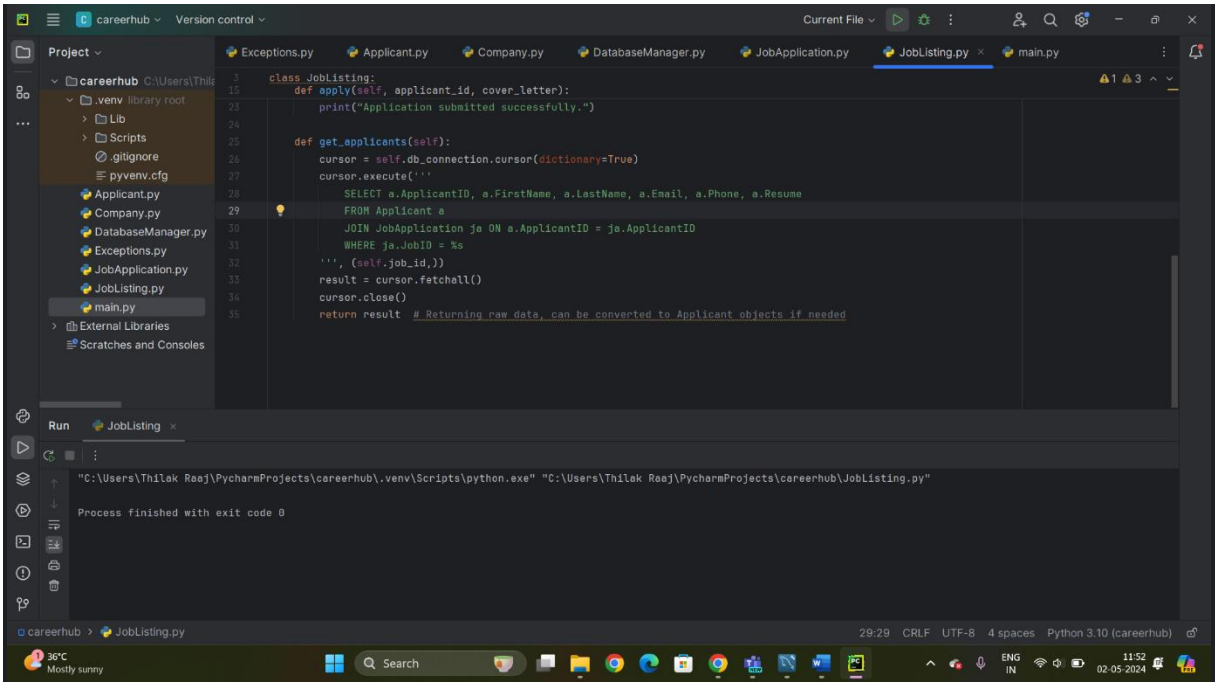
```
    def apply_for_job(self, job_id, cover_letter):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO JobApplication (JobID, ApplicantID,
ApplicationDate, CoverLetter)
            VALUES (%s, %s, NOW(), %s)
        ''', (job_id, self.applicant_id, cover_letter))
        self.db_connection.commit()
        cursor.close()
import mysql.connector
import JobListing

class Company:
    def __init__(self, db_connection, company_id, company_name, location):
        self.db_connection = db_connection
        self.company_id = company_id
        self.company_name = company_name
        self.location = location

    def post_job(self, job_id, job_title, job_description, job_location,
salary, job_type, posted_date):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO JobListing (JobID, CompanyID, JobTitle,
JobDescription, JobLocation, Salary, JobType, PostedDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        ''', (job_id, self.company_id, job_title, job_description,
job_location, salary, job_type, posted_date))
        self.db_connection.commit()
        cursor.close()

    def get_jobs(self):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('SELECT * FROM JobListing WHERE CompanyID = %s',
(self.company_id,))
        result = cursor.fetchall()
        cursor.close()
        return [JobListing(self.db_connection, **row) for row in result]
class JobApplication:
    def __init__(self, db_connection, application_id=None, job_id=None,
applicant_id=None, application_date=None, cover_letter=None):
        self.db_connection = db_connection
        self.application_id = application_id
        self.job_id = job_id
        self.applicant_id = applicant_id
        self.application_date = application_date
        self.cover_letter = cover_letter
import mysql.connector

class JobListing:
    def __init__(self, db_connection, job_id, company_id, job_title,
job_description, job_location, salary, job_type, posted_date):
        self.db_connection = db_connection
        self.job_id = job_id
        self.company_id = company_id
        self.job_title = job_title
        self.job_description = job_description
        self.job_location = job_location
        self.salary = salary
        self.job_type = job_type
```

```
        self.posted_date = posted_date

    def apply(self, applicant_id, cover_letter):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO JobApplication (JobID, ApplicantID,
ApplicationDate, CoverLetter)
            VALUES (%s, %s, NOW(), %s)
        ''', (self.job_id, applicant_id, cover_letter))
        self.db_connection.commit()
        cursor.close()
        print("Application submitted successfully.")

    def get_applicants(self):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('''
            SELECT a.ApplicantID, a.FirstName, a.LastName, a.Email,
a.Phone, a.Resume
            FROM Applicant a
            JOIN JobApplication ja ON a.ApplicantID = ja.ApplicantID
            WHERE ja.JobID = %s
        ''', (self.job_id,))
        result = cursor.fetchall()
        cursor.close()
        return result  # Returning raw data, can be converted to Applicant
objects if needed
```




careerhub
Version control
Current File
Project
Exceptions.py
Applicant.py
Company.py
DatabaseManager.py
JobApplication.py
JobListing.py
main.py
careerhub C:\Users\Thila
.venv library root
Lib
Scripts
.gitignore
pyvenv.cfg
Applicant.py
Company.py
DatabaseManager.py
Exceptions.py
JobApplication.py
JobListing.py
main.py
External Libraries
Scratches and Consoles
class Company:
def __init__(self, db_connection, company_id, company_name, location):
self.db_connection = db_connection
self.company_id = company_id
self.company_name = company_name
self.location = location
1 usage
def post_job(self, job_id, job_title, job_description, job_location, salary, job_type, posted_date):
cursor = self.db_connection.cursor()
cursor.execute('''
INSERT INTO JobListing (JobID, CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType, PostedDate)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
''', (job_id, self.company_id, job_title, job_description, job_location, salary, job_type, posted_date))
self.db_connection.commit()
cursor.close()
def get_jobs(self):
cursor = self.db_connection.cursor(dictionary=True)
cursor.execute('SELECT * FROM JobListing WHERE CompanyID = %s', (self.company_id,))
Run
Company
"C:\Users\Thilak Raaj\PycharmProjects\careerhub\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\careerhub\Company.py"
Process finished with exit code 0
careerhub > Company.py
16:23 CRLF UTF-8 4 spaces Python 3.10 (careerhub)
36°C
Mostly sunny
Search
ENG
IN
11:52
02-05-2024

careerhub
Version control
Current File
Project
Exceptions.py
Applicant.py
Company.py
DatabaseManager.py
JobApplication.py
JobListing.py
main.py
careerhub C:\Users\Thila
.venv library root
Lib
Scripts
.gitignore
pyvenv.cfg
Applicant.py
Company.py
DatabaseManager.py
Exceptions.py
JobApplication.py
JobListing.py
main.py
External Libraries
Scratches and Consoles
class Applicant:
def create_profile(self, email, first_name, last_name, phone):
self.db_connection.commit()
self.applicant_id = cursor.lastrowid
cursor.close()
def apply_for_job(self, job_id, cover_letter):
cursor = self.db_connection.cursor()
cursor.execute('''
INSERT INTO JobApplication (JobID, ApplicantID, ApplicationDate, CoverLetter)
VALUES (%s, %s, NOW(), %s)
''', (job_id, self.applicant_id, cover_letter))
self.db_connection.commit()
cursor.close()
Run
Applicant
"C:\Users\Thilak Raaj\PycharmProjects\careerhub\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\careerhub\Applicant.py"
Process finished with exit code 0
careerhub > Applicant.py
31:1 CRLF UTF-8 4 spaces Python 3.10 (careerhub)
36°C
Mostly sunny
Search
ENG
IN
11:52
02-05-2024



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" table.

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

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

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

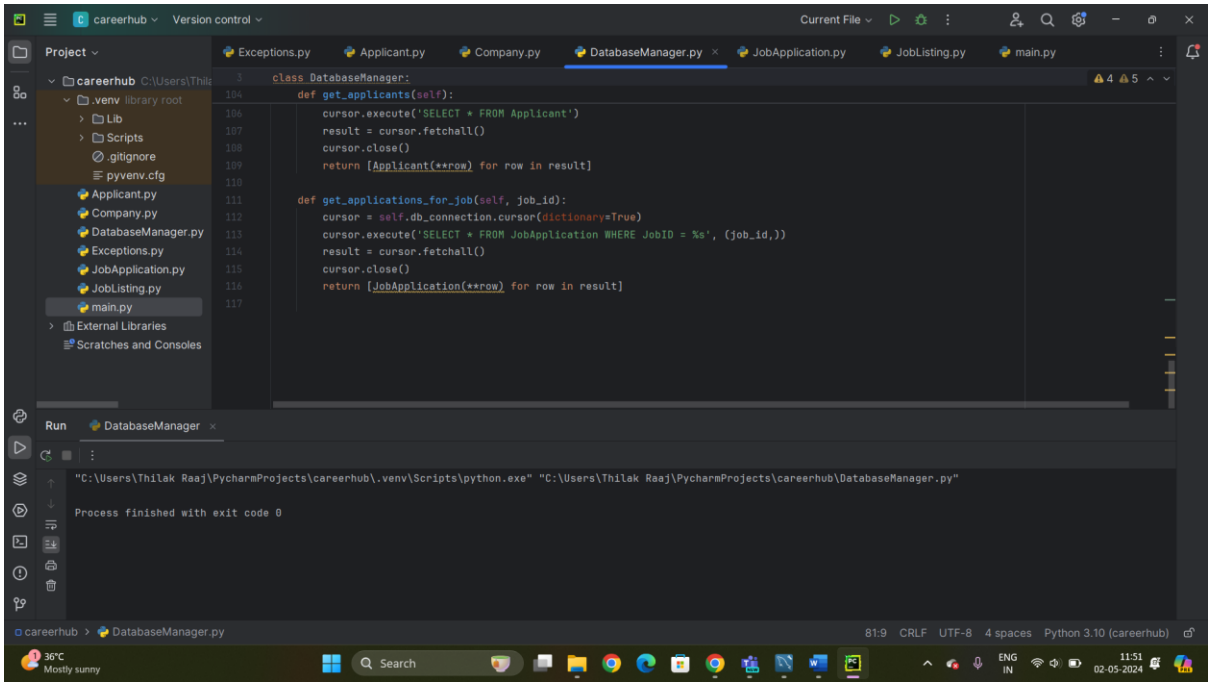
• GetApplicationsForJob(jobID: int): List: Retrieves a list of job applications for a specific job listing.

CODE:

```
import JobListing, Company, Applicant, JobApplication

class DatabaseManager:
    def __init__(self, db_connection):
        self.db_connection = db_connection
        self.initialize_database()

    def initialize_database(self):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Company (
                CompanyID INT AUTO_INCREMENT PRIMARY KEY,
                CompanyName VARCHAR(255),
                Location VARCHAR(255)
            );
        ''')
```

```
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS JobListing (
                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,
                FOREIGN KEY (CompanyID) REFERENCES Company(CompanyID)
            );
        ''')
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applicant (
                ApplicantID INT AUTO_INCREMENT PRIMARY KEY,
                FirstName VARCHAR(255),
                LastName VARCHAR(255),
                Email VARCHAR(255),
                Phone VARCHAR(50),
                Resume TEXT
            );
        ''')
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS JobApplication (
                ApplicationID INT AUTO_INCREMENT PRIMARY KEY,
                JobID INT,
                ApplicantID INT,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY (JobID) REFERENCES JobListing(JobID),
                FOREIGN KEY (ApplicantID) REFERENCES Applicant(ApplicantID)
            );
        ''')
        self.db_connection.commit()
        cursor.close()

    def insert_job_listing(self, job):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO JobListing (CompanyID, JobTitle, JobDescription,
JobLocation, Salary, JobType, PostedDate)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        ''', (job.company_id, job.job_title, job.job_description,
job.job_location, job.salary, job.job_type, job.posted_date))
        self.db_connection.commit()
        cursor.close()

    def insert_company(self, company):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO Company (CompanyName, Location)
            VALUES (%s, %s)
        ''', (company.company_name, company.location))
        self.db_connection.commit()
        cursor.close()

    def insert_applicant(self, applicant):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO Applicant (ApplicantID, FirstName, LastName, Email,
```

```
Phone, Resume)
            VALUES (%s, %s, %s, %s, %s, %s)
        ''', (applicant.applicant_id, applicant.first_name,
applicant.last_name, applicant.email, applicant.phone, applicant.resume))
        self.db_connection.commit()
        cursor.close()

    def insert_job_application(self, application):
        cursor = self.db_connection.cursor()
        cursor.execute('''
            INSERT INTO JobApplication (ApplicationID, JobID, ApplicantID,
ApplicationDate, CoverLetter)
            VALUES (%s, %s, %s, %s, %s)
        ''', (application.application_id, application.job_id,
application.applicant_id, application.application_date,
application.cover_letter))
        self.db_connection.commit()
        cursor.close()

    def get_job_listings(self):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('SELECT * FROM JobListing')
        result = cursor.fetchall()
        cursor.close()
        return [JobListing(**row) for row in result]

    def get_companies(self):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('SELECT * FROM Company')
        result = cursor.fetchall()
        cursor.close()
        return [Company(**row) for row in result]

    def get_applicants(self):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('SELECT * FROM Applicant')
        result = cursor.fetchall()
        cursor.close()
        return [Applicant(**row) for row in result]

    def get_applications_for_job(self, job_id):
        cursor = self.db_connection.cursor(dictionary=True)
        cursor.execute('SELECT * FROM JobApplication WHERE JobID = %s',
(job_id,))
        result = cursor.fetchall()
        cursor.close()
        return [JobApplication(**row) for row in result]
```

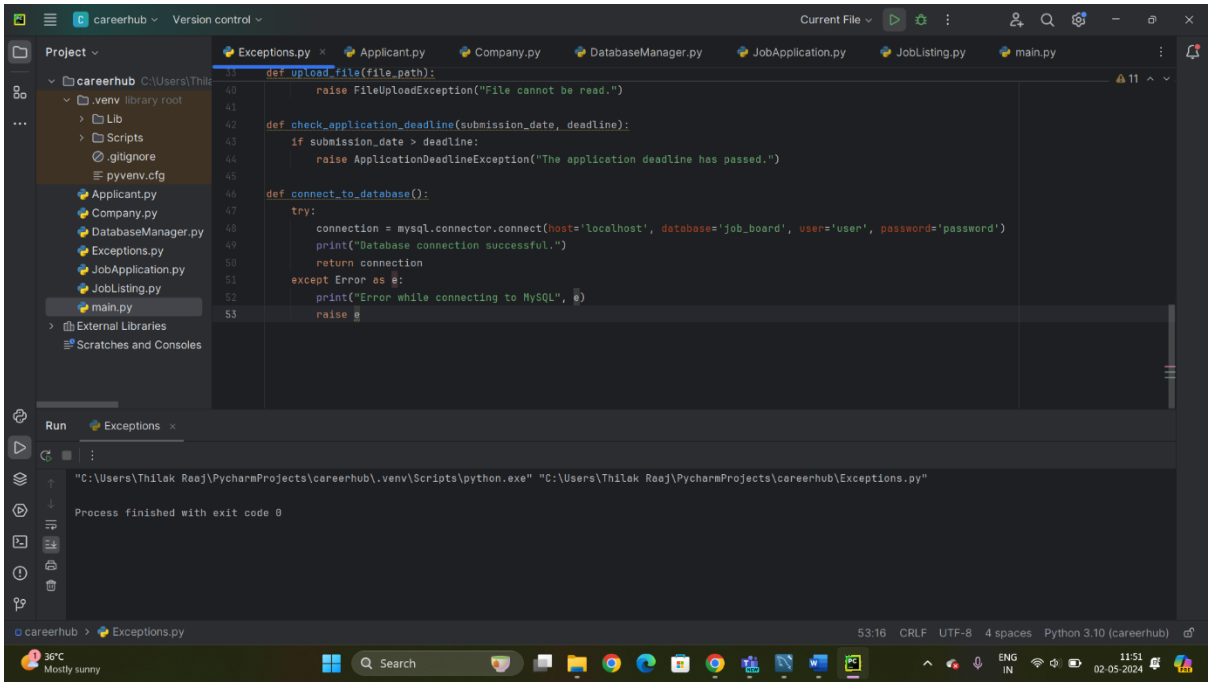


3.Exceptions handling Create and implement the following exceptions in your application. • Invalid Email Format Handling: o In the Job Board application, during the applicant registration process, users are required to enter their email addresses. Write a program that prompts the user to input an email 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. • 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. • 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. • 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. • 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.

CODE:

```
import re
import mysql.connector
from mysql.connector import Error

class InvalidEmailFormatException(Exception):
    def __init__(self, email, message="Invalid email format"):
```

```
        self.email = email
        self.message = message
        super().__init__(self.message)

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

class FileUploadException(Exception):
    pass

class ApplicationDeadlineException(Exception):
    pass

# Utility functions
def validate_email(email):
    if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
        raise InvalidEmailFormatException(email)

def calculate_average_salary(salaries):
    if any(s < 0 for s in salaries):
        raise NegativeSalaryException("Salaries must be non-negative.")
    return sum(salaries) / len(salaries) if salaries else 0

def upload_file(file_path):
    try:
        with open(file_path, 'r') as file:
            print("File uploaded successfully")
    except FileNotFoundError:
        raise FileUploadException("File not found.")
    except IOError:
        raise FileUploadException("File cannot be read.")

def check_application_deadline(submission_date, deadline):
    if submission_date > deadline:
        raise ApplicationDeadlineException("The application deadline has
passed.")

def connect_to_database():
    try:
        connection = mysql.connector.connect(host='localhost',
database='job_board', user='user', password='password')
        print("Database connection successful.")
        return connection
    except Error as e:
        print("Error while connecting to MySQL", e)
        raise 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.

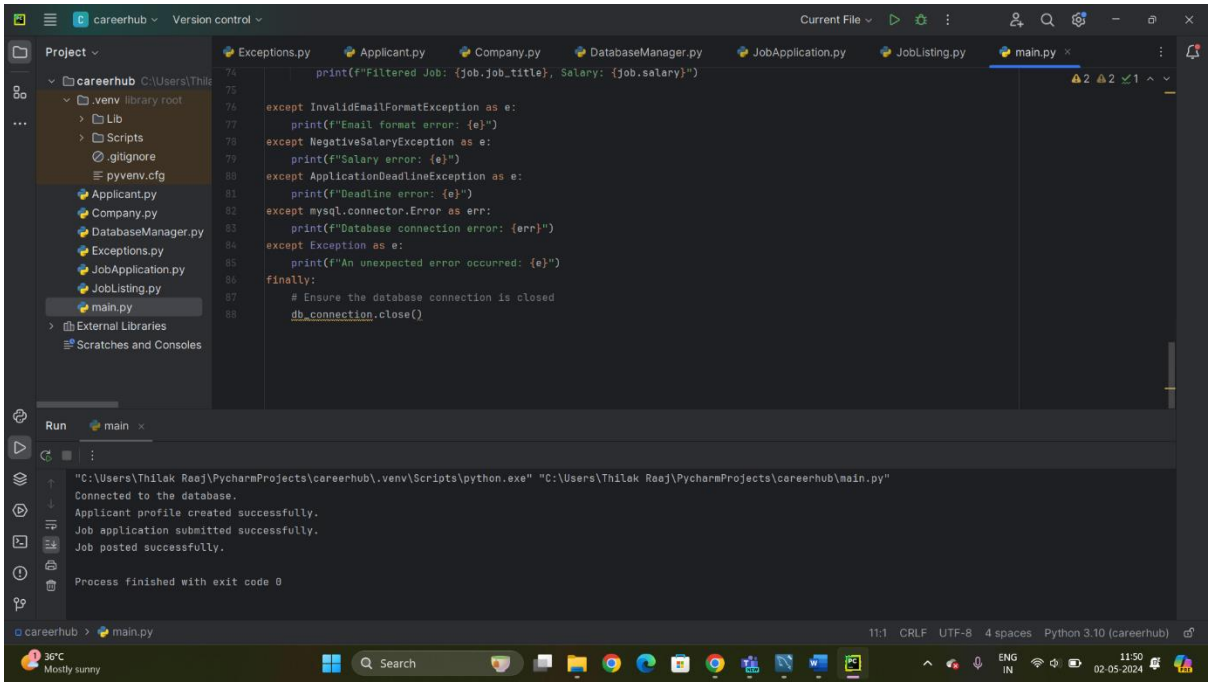
• Salary Range Query: Create a program that allows users to search for job listings within a specified salary range. Implement database connectivity to retrieve job listings that match the user's criteria, including job titles, company names, and salaries. Ensure the program handles database connectivity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

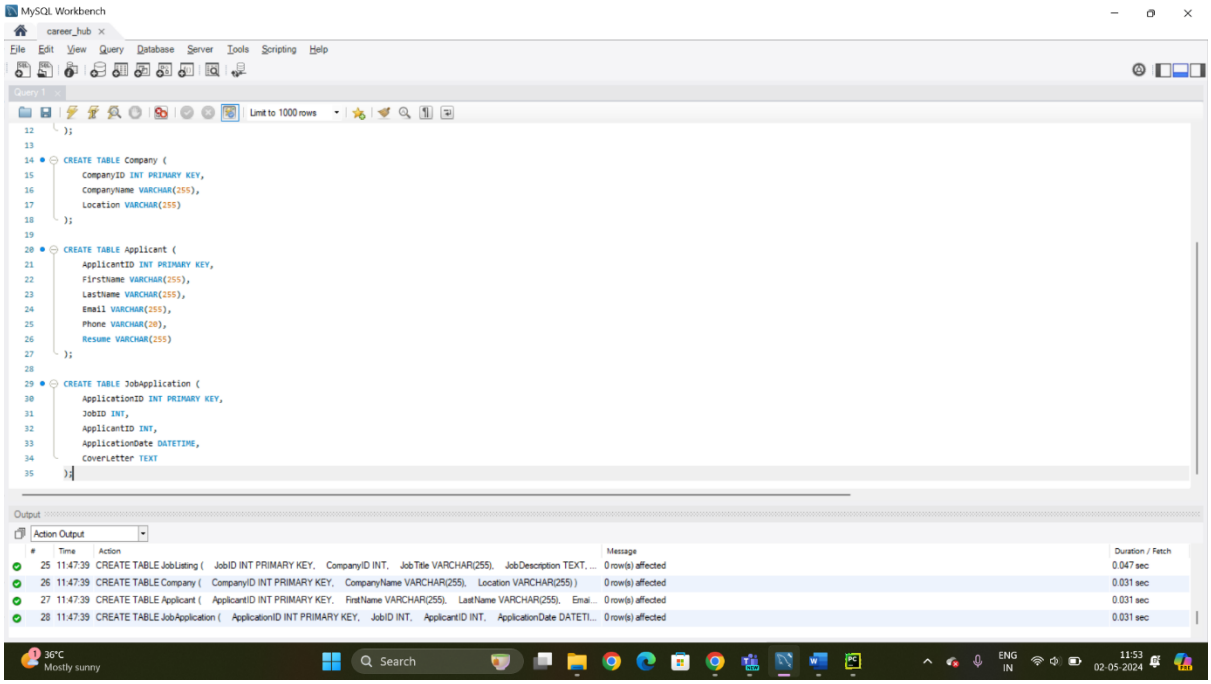
CODE:

```
from DatabaseManager import DatabaseManager
from JobListing import JobListing
from Company import Company
from Applicant import Applicant
from JobApplication import JobApplication
import mysql.connector
import re
from Exceptions import InvalidEmailFormatException
from Exceptions import NegativeSalaryException
```

```
from Exceptions import ApplicationDeadlineException

applicant_email = "jane.doe@example.com"
today_date = "2023-04-01"  # Assuming current date is April 1, 2023
application_date = "2023-05-01"
job_id = 1
applicant_id = 1
company_id = 1
company_name = "Hexaware Technologies"
company_location = "New York"
job_title = "Software Engineer"
job_description = "Develop enterprise software."
job_location = "New York"
salary = 95000
job_type = "Full-time"
posted_date = "2023-05-01"
cover_letter = "I am very interested in this position."
min_salary = 50000
max_salary = 150000

try:
    # Establishing the database connection using DatabaseManager
    db_connection = mysql.connector.connect(
        host='localhost',
        user='root',
        password='root',
        database='career_hub'
    )
    db_manager = DatabaseManager(db_connection)
    print("Connected to the database.")

    # Job Listing Retrieval
    job_listings = db_manager.get_job_listings()
    for job in job_listings:
        if job.salary < 0:
            raise NegativeSalaryException(job.salary, "Salary cannot be
negative")
        print(f"Job Title: {job.job_title}, Company ID: {job.company_id},
Salary: {job.salary}")

    # Applicant Profile Creation
    if not re.match(r"[^@]+@[^@]+\.[^@]+", applicant_email):
        raise InvalidEmailFormatException(applicant_email)
    applicant = Applicant(db_connection, applicant_id, first_name="Jane",
last_name="Doe", email=applicant_email,
                          phone="1234567890", resume="resume_link.pdf")
    db_manager.insert_applicant(applicant)
    print("Applicant profile created successfully.")

    # Job Application Submission
    if today_date > application_date:
        raise ApplicationDeadlineException("The application deadline has
passed.")
    job_application = JobApplication(db_connection, application_id=1,
job_id=job_id, applicant_id=applicant_id,
                                     application_date=application_date,
cover_letter=cover_letter)
    db_manager.insert_job_application(job_application)
    print("Job application submitted successfully.")

    # Company Job Posting
```

```
    if salary < 0:
        raise NegativeSalaryException(salary, "Salary cannot be negative.")
    company = Company(db_connection, company_id=company_id,
company_name=company_name, location=company_location)
    company.post_job(job_id, job_title, job_description, job_location,
salary, job_type, posted_date)
    print("Job posted successfully.")

    # Salary Range Query
    filtered_jobs = [job for job in job_listings if job.salary >=
min_salary and job.salary <= max_salary]
    for job in filtered_jobs:
        print(f"Filtered Job: {job.job_title}, Salary: {job.salary}")

except InvalidEmailFormatException as e:
    print(f"Email format error: {e}")
except NegativeSalaryException as e:
    print(f"Salary error: {e}")
except ApplicationDeadlineException as e:
    print(f"Deadline error: {e}")
except mysql.connector.Error as err:
    print(f"Database connection error: {err}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
finally:
    # Ensure the database connection is closed
    db_connection.close()
```




MySQL Workbench
career_hub
File Edit View Query Database Server Tools Scripting Help
Limit to 1000 rows
);
CREATE TABLE Company (
CompanyID INT PRIMARY KEY,
CompanyName VARCHAR(255),
Location VARCHAR(255)
);
CREATE TABLE Applicant (
ApplicantID INT PRIMARY KEY,
FirstName VARCHAR(255),
LastName VARCHAR(255),
Email VARCHAR(255),
Phone VARCHAR(20),
Resume VARCHAR(255)
);
CREATE TABLE JobApplication (
ApplicationID INT PRIMARY KEY,
JobID INT,
ApplicantID INT,
ApplicationDate DATETIME,
CoverLetter TEXT
);
Output
Action Output
Time Action Message Duration / Fetch
25 11:47:39 CREATE TABLE JobListing (JobID INT PRIMARY KEY, CompanyID INT, JobTitle VARCHAR(255), JobDescription TEXT, ... 0 row(s) affected 0.047 sec
26 11:47:39 CREATE TABLE Company (CompanyID INT PRIMARY KEY, CompanyName VARCHAR(255), Location VARCHAR(255)) 0 row(s) affected 0.031 sec
27 11:47:39 CREATE TABLE Applicant (ApplicantID INT PRIMARY KEY, FirstName VARCHAR(255), LastName VARCHAR(255), Emai... 0 row(s) affected 0.031 sec
28 11:47:39 CREATE TABLE JobApplication (ApplicationID INT PRIMARY KEY, JobID INT, ApplicantID INT, ApplicationDate DATETI... 0 row(s) affected 0.031 sec
36°C
Mostly sunny
Search
ENG
IN
11:53
02-05-2024