**CHAPTER 1**

**INTRODUCTION**

**1.1 About the Project**

In response to the changing landscape of employment practices, we have developed an innovative Job Recruitment Portal designed to modernize and simplify the job application process. Utilizing a robust database management system, our platform offers a seamless online experience for both job seekers and employers. This portal provides a comprehensive database of job listings, allowing users to easily navigate and explore opportunities across various industries. Job seekers can create personalized profiles, submit applications securely, and receive real-time notifications on their application status. Employers benefit from centralized and organized recruitment processes, with the ability to showcase company profiles and attract top talent. The scalable architecture ensures adaptability to the growing needs of users, while the emphasis on data security and integrity ensures the protection of sensitive information. With a mobile-accessible interface, our Job Recruitment Portal redefines the hiring experience, connecting talented individuals with exciting career prospects anytime, anywhere.

**CHAPTER 2**

**SCOPE OF THE PROJECT**

**2.1 Problem statement**

To design and develop a portal which aims to simplify the recruitment process for employers and job seekers, making the hiring journey more efficient and transparent.

**2.2 Objectives**

• To streamline the hiring process, making it more efficient for both employers and job seekers.

• To enhance transparency in the job market by providing clear communication channels and timely updates for both employers and job seekers.

• To ensure data integrity and consistency within the database.

**CHAPTER 3**

**SYSTEM REQUIREMENTS**

**3.1 Functional Requirements**

**1. Database Setup and Management**

* Database Creation:
  + Create a database named job\_recruitment if it does not already exist.
* Table Creation:
  + job Table: Stores job information including job ID, title, description, and salary.
  + job\_seeker Table: Stores job seeker information including seeker ID, name, skills, experience, and job ID. The job ID in this table references the job table.
  + employee Table: Stores employee information including employee number, name, department, salary, and job ID. The job ID in this table also references the job table.
  + admin Table: Stores administrator information including admin ID, username, and password.

**2. Record Management**

* Add Record:
  + Function: add\_record()
  + Inputs: Table name, record-specific details.
  + Process: Insert a new record into the specified table.
  + Validation: Ensure job ID in job\_seeker and employee tables can be 0 to denote no job assignment, otherwise set to NULL.
* Display Record:
  + Function: display\_record()
  + Inputs: Table name.
  + Process: Retrieve and display all records from the specified table.
* Delete Record:
  + Function: delete\_record()
  + Inputs: Table name, primary key column name, primary key value.
  + Process: Delete a record from the specified table based on the primary key.
* Update Record:
  + Function: update\_record()
  + Inputs: Table name, primary key column name, primary key value, column to update, new value.
  + Process: Update a specified column's value for a record identified by the primary key.

**3. User Interaction**

* Menu Options:
  + Present a menu to the user with options to add, display, delete, and update records.
* Choice Handling:
  + Handle user inputs to execute the corresponding functions for managing records.
  + Provide feedback on successful operations or invalid choices.

**4. Error Handling and Validation**

* SQL Injection Prevention:
  + Use parameterized queries or proper escaping to avoid SQL injection risks.
* Input Validation:
  + Ensure inputs are validated before use in queries to prevent errors and potential security issues.
* Database Constraints:
  + Handle foreign key constraints to ensure referential integrity between tables.

**5. Connection Management**

* Establish Connection:
  + Connect to the MySQL database using credentials (host, user, password).
* Close Connection:
  + Properly close the database connection when exiting the program.

**6. Administrative Functions**

* Database and Table Maintenance:
  + Ensure that tables are created if they don’t exist and properly structured for data integrity.
* Data Security:
  + Manage admin credentials securely, considering password hashing or encryption (though not implemented in the provided code).

**7. System Usability**

* Command-line Interface:
  + Provide a simple command-line interface for users to interact with the database.
* Feedback Messages:
  + Provide clear messages indicating successful or failed operations (e.g., "Data Saved", "Record Deleted").

**8. Data Integrity**

* Foreign Key Constraints:
  + Ensure relationships between tables are maintained, such as job seekers and employees being linked to valid job IDs.

By addressing these functional requirements, the program effectively supports basic record management operations for a job recruitment portal while ensuring usability, data integrity, and security.

**3.2 Non-Functional Requirements**

Non-functional requirements are crucial for defining the overall quality attributes and constraints of the system that impact its performance and user experience. For the job recruitment portal program you provided, here are the non-functional requirements:

**1. Performance**

* Response Time:
  + The system should respond to user inputs within a reasonable time frame. For example, operations like adding, updating, or deleting records should complete within a few seconds.
* Scalability:
  + The system should handle a growing number of records efficiently. Performance should not degrade significantly as the amount of data increases.

**2. Reliability and Availability**

* System Uptime:
  + The system should be available 24/7 with minimal downtime. Maintenance or unexpected failures should not disrupt the normal operation significantly.
* Error Handling:
  + The system should gracefully handle errors (e.g., invalid inputs, database connection issues) without crashing. Error messages should be clear and guide users on how to correct issues.

**3. Security**

* Data Protection:
  + User data (especially sensitive information such as passwords) should be protected. Passwords should be stored securely (e.g., hashed).
* Access Control:
  + Different users (e.g., job seekers, employees, admins) should have appropriate levels of access. Admin functionality should be restricted to authorized users only.
* SQL Injection Prevention:
  + The system should prevent SQL injection attacks by using parameterized queries or proper escaping of user inputs.

**4. Usability**

* User Interface:
  + The command-line interface should be intuitive and easy to use. Users should be able to perform actions with minimal confusion.
* Error Feedback:
  + The system should provide meaningful feedback on user actions (e.g., success messages, error descriptions) to help users understand the outcome of their operations.

**5. Maintainability**

* Code Quality:
  + The code should be well-organized, with clear comments and documentation to facilitate maintenance and future development.
* Modularity:
  + The code should be modular to allow easy updates or changes. Functions should be responsible for distinct tasks to simplify debugging and enhancements.

**6. Compatibility**

* Platform Independence:
  + The system should be compatible with the intended operating systems where it will be deployed, ensuring consistent performance across these platforms.
* Database Compatibility:
  + The MySQL version used should be compatible with the features and queries employed by the program.

**7. Data Integrity**

* Consistency:
  + Data should remain consistent and accurate throughout the system. Constraints and validations should ensure that data relationships and rules are upheld.
* Backup and Recovery:
  + There should be a strategy for backing up the database and recovering it in case of failure or data loss.

**8. Documentation**

* User Documentation:
  + Documentation should be provided for end-users to understand how to use the system, including how to perform each operation.
* Technical Documentation:
  + Detailed documentation should be available for developers and maintainers, including the system architecture, database schema, and codebase overview.

**9. Efficiency**

* Resource Utilization:
  + The system should efficiently use system resources (CPU, memory) to avoid unnecessary load or performance bottlenecks.

**10. Compliance**

* Regulatory Requirements:
  + The system should comply with relevant data protection regulations and industry standards (e.g., GDPR, CCPA) to ensure legal and ethical handling of user data.

These non-functional requirements ensure that the job recruitment portal operates effectively and meets user expectations in terms of performance, security, and usability while maintaining a high standard of quality and reliability.

**Top of Form**

**Bottom of Form**

**3.3 Hardware Requirements**

Operating System: Windows versions 10 and above

Front-End: Python

Back-End: MySQL

**3.4 Software Requirements**

Processor: Intel i5 processor

RAM: 1GB or more

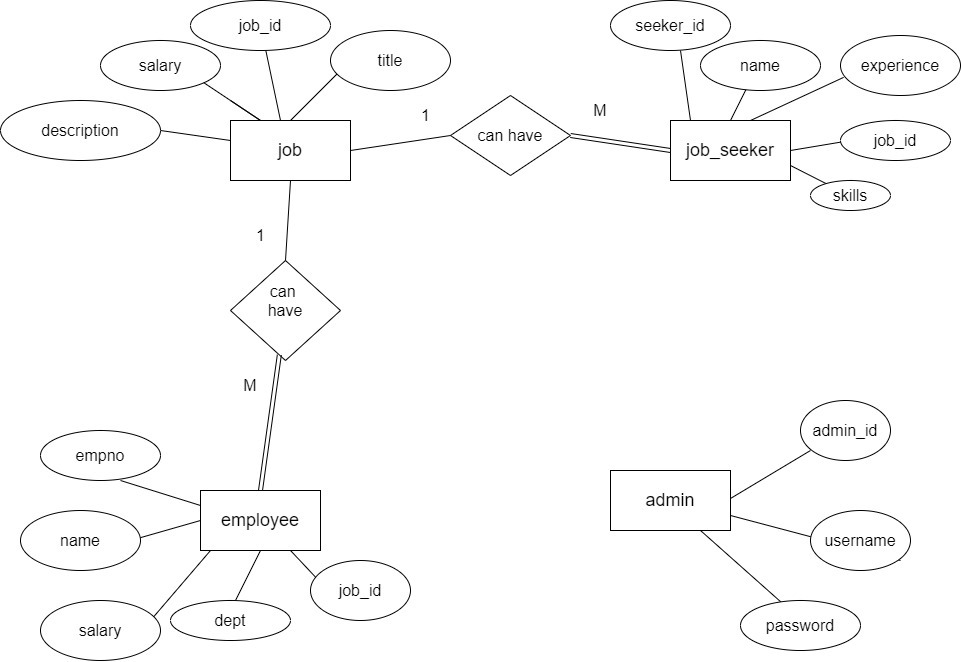
HDD: 40GB

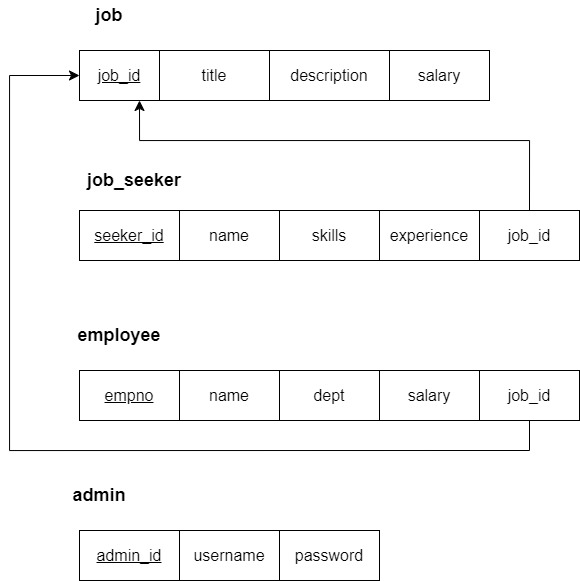
Keyboard: Normal

Mouse: Normal

**CHAPTER 4**

**METHODOLOGY**

**4.1 E R Diagram**

**4.2 Schema Diagram**

**CHAPTER 5:**

**IMPLEMENTATION**

**5.1 Table Description**

job:

job\_id: INT PRIMARY KEY - Unique identifier for the job.

Title: VARCHAR(50) - The title or name of the job position.

description: TEXT - A detailed description of the job responsibilities and requirements.

salary: INT - The salary offered for the job position.

job\_seeker:

seeker\_id: INT PRIMARY KEY - Unique identifier for the job seeker.

name: VARCHAR(50) - The name of the job seeker.

skills: TEXT - A list or description of the skills possessed by the job seeker.

experience: INT - The number of years of experience the job seeker has.

job\_id: INT - Foreign key referencing job(job\_id), indicating the job the seeker is applying for. This can be NULL if the seeker is not applying for any job.

employee:

empno: INT PRIMARY KEY - Unique identifier for the employee.

name: VARCHAR(20) - The name of the employee.

dept: VARCHAR(20) - The department where the employee works.

salary: INT - The salary of the employee.

job\_id: INT - Foreign key referencing job(job\_id), indicating the job the employee is assigned to. This can be NULL if the employee is not assigned to any specific job.

admin:

admin\_id: INT PRIMARY KEY - Unique identifier for the admin.

username: VARCHAR(20) - The username for the admin.

password: VARCHAR(20) - The password for the admin.

**5.2 Source Code**

import mysql.connector as mycon

# Establish connection

con = mycon.connect(host='localhost', user='root', password="Mouni@123")

cur = con.cursor()

# Create database and tables if they don't exist

cur.execute("CREATE DATABASE IF NOT EXISTS job\_recruitment")

cur.execute("USE job\_recruitment")

# Creating tables with relationships

cur.execute("""

CREATE TABLE IF NOT EXISTS job (

job\_id INT PRIMARY KEY,

title VARCHAR(50),

description TEXT,

salary INT

)

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS job\_seeker (

seeker\_id INT PRIMARY KEY,

name VARCHAR(50),

skills TEXT,

experience INT,

job\_id INT,

FOREIGN KEY (job\_id) REFERENCES job(job\_id)

)

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS employee (

empno INT PRIMARY KEY,

name VARCHAR(20),

dept VARCHAR(20),

salary INT,

job\_id INT,

FOREIGN KEY (job\_id) REFERENCES job(job\_id)

)

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS admin (

admin\_id INT PRIMARY KEY,

username VARCHAR(20),

password VARCHAR(20)

)

""")

con.commit()

def add\_record():

table = input("Enter the table name (job, job\_seeker, employee, admin): ")

if table == "job":

job\_id = int(input("Enter Job ID: "))

title = input("Enter Title: ")

description = input("Enter Description: ")

salary = int(input("Enter Salary: "))

query = "INSERT INTO job VALUES ({}, '{}', '{}', {})".format(job\_id, title, description, salary)

elif table == "job\_seeker":

seeker\_id = int(input("Enter Job Seeker ID: "))

name = input("Enter Name: ")

skills = input("Enter Skills: ")

experience = int(input("Enter Experience: "))

job\_id = int(input("Enter Job ID (or 0 if not applying for any job): "))

if job\_id == 0:

job\_id = 'NULL'

query = "INSERT INTO job\_seeker VALUES ({}, '{}', '{}', {}, {})".format(seeker\_id, name, skills, experience, job\_id)

elif table == "employee":

empno = int(input("Enter Employee Number: "))

name = input("Enter Name: ")

dept = input("Enter Department: ")

salary = int(input("Enter Salary: "))

job\_id = int(input("Enter Job ID (or 0 if not assigned any job): "))

if job\_id == 0:

job\_id = 'NULL'

query = "INSERT INTO employee VALUES ({}, '{}', '{}', {}, {})".format(empno, name, dept, salary, job\_id)

elif table == "admin":

admin\_id = int(input("Enter Admin ID: "))

username = input("Enter Username: ")

password = input("Enter Password: ")

query = "INSERT INTO admin VALUES ({}, '{}', '{}')".format(admin\_id, username, password)

else:

print("Invalid table name.")

return

cur.execute(query)

con.commit()

print("## Data Saved ##")

def display\_record():

table = input("Enter the table name (job, job\_seeker, employee, admin): ")

query = "SELECT \* FROM {}".format(table)

cur.execute(query)

result = cur.fetchall()

for row in result:

print(row)

def delete\_record():

table = input("Enter the table name (job, job\_seeker, employee, admin): ")

column = input("Enter the column name of the primary key: ")

id = input("Enter the ID of the record to delete: ")

query = "DELETE FROM {} WHERE {} = {}".format(table, column, id)

cur.execute(query)

con.commit()

print("## Record Deleted ##")

def update\_record():

table = input("Enter the table name (job, job\_seeker, employee, admin): ")

column = input("Enter the column name of the primary key: ")

id = input("Enter the ID of the record to update: ")

update\_col = input("Enter the column name to update: ")

new\_value = input("Enter the new value: ")

query = "UPDATE {} SET {} = '{}' WHERE {} = {}".format(table, update\_col, new\_value, column, id)

cur.execute(query)

con.commit()

print("## Record Updated ##")

choice = None

while choice != 0:

print("1. ADD RECORD")

print("2. DISPLAY RECORD")

print("3. DELETE RECORD")

print("4. UPDATE RECORD")

print("0. EXIT")

choice = int(input("Enter Choice: "))

if choice == 1:

add\_record()

elif choice == 2:

display\_record()

elif choice == 3:

delete\_record()

elif choice == 4:

update\_record()

elif choice == 0:

con.close()

print("## Bye!! ##")

else:

print("## INVALID CHOICE ##")

**5.3 Connection**

To establish a connection to the MySQL database using the provided Python code, you need to ensure that you have MySQL installed and the mysql-connector-python package available in your Python environment. Here's a brief guide on how to set up the connection and handle it properly:

Step-by-Step Guide to Establish a MySQL Connection

1. Install MySQL Connector: First, ensure you have the MySQL Connector Python package installed. You can install it using pip if it's not already installed:

pip install mysql-connector-python

1. Connection Code: Use the Python code to establish a connection to the MySQL database.

**CHAPTER 6**

**RESULT**

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 1

Enter the table name (job, job\_seeker, employee, admin): job

Enter Job ID: 1

Enter Title: Project Manager

Enter Description: Planning,executing and successfully completing projects within alloted timeline and budget

Enter Salary: 70000

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 1

Enter the table name (job, job\_seeker, employee, admin): job

Enter Job ID: 3

Enter Title: Managing Director

Enter Description: Plans the annual marketing startegy for a company products or services to support bussiness growth ambition

Enter Salary: 90000

## Data Saved ##

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 1

Enter the table name (job, job\_seeker, employee, admin): job\_seeker

Enter Job Seeker ID: 101

Enter Name: Mouni

Enter Skills: Leadership

Enter Experience: 4

Enter Job ID (or 0 if not applying for any job): 3

## Data Saved ##

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 1

Enter the table name (job, job\_seeker, employee, admin): employee

Enter Employee Number: 201

Enter Name: Manu

Enter Department: Mangement

Enter Salary: 100000

Enter Job ID (or 0 if not assigned any job): 3

## Data Saved ##

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 1

Enter the table name (job, job\_seeker, employee, admin): admin

Enter Admin ID: 1001

Enter Username: Mounika

Enter Password: Mouni@123

## Data Saved ##

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 2

Enter the table name (job, job\_seeker, employee, admin): job

(1, 'Project Manager', 'Planning,executing and successfully completing projects within alloted timeline and budget', 70000)

(3, 'Managing Director', 'Plans the annual marketing startegy for a company products or services to support bussiness growth ambition', 90000)

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 2

Enter the table name (job, job\_seeker, employee, admin): job\_seeker

(101, 'Mouni', 'Leadership', 4, 3)

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 2

Enter the table name (job, job\_seeker, employee, admin): employee

(201, 'Manu', 'Mangement', 100000, 3)

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 2

Enter the table name (job, job\_seeker, employee, admin): admin

(1001, 'Mounika', 'Mouni@123')

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 3

Enter the table name (job, job\_seeker, employee, admin): job

Enter the column name of the primary key: job\_id

Enter the ID of the record to delete: 1

## Record Deleted ##

1. ADD RECORD

2. DISPLAY RECORD

3. DELETE RECORD

4. UPDATE RECORD

0. EXIT

Enter Choice: 4

Enter the table name (job, job\_seeker, employee, admin): job\_seeker

Enter the column name of the primary key: seeker\_id

Enter the ID of the record to update: 101

Enter the column name to update: experience

Enter the new value: 6

## Record Updated ##

**Stored in back-end(screenshots)**

Adding the record

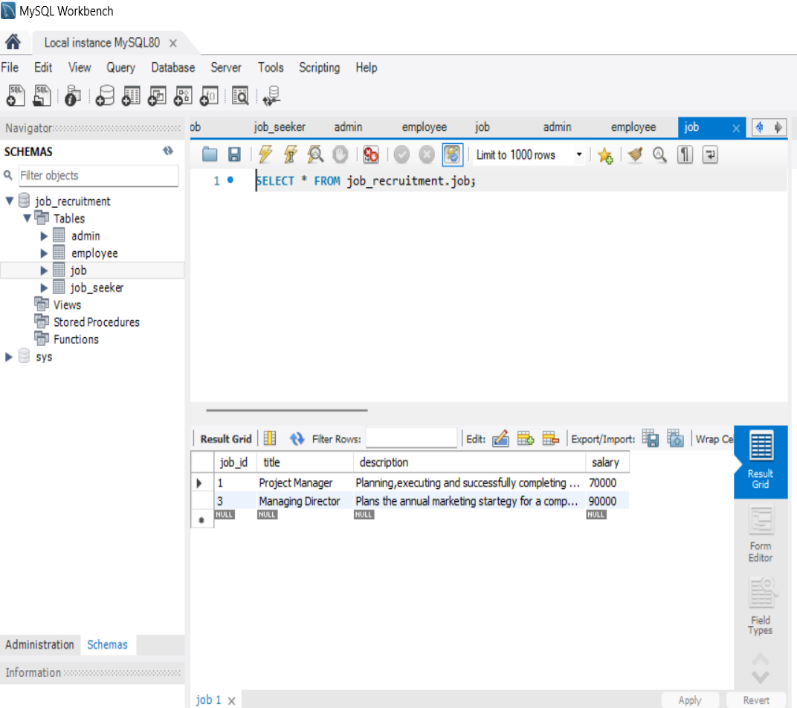
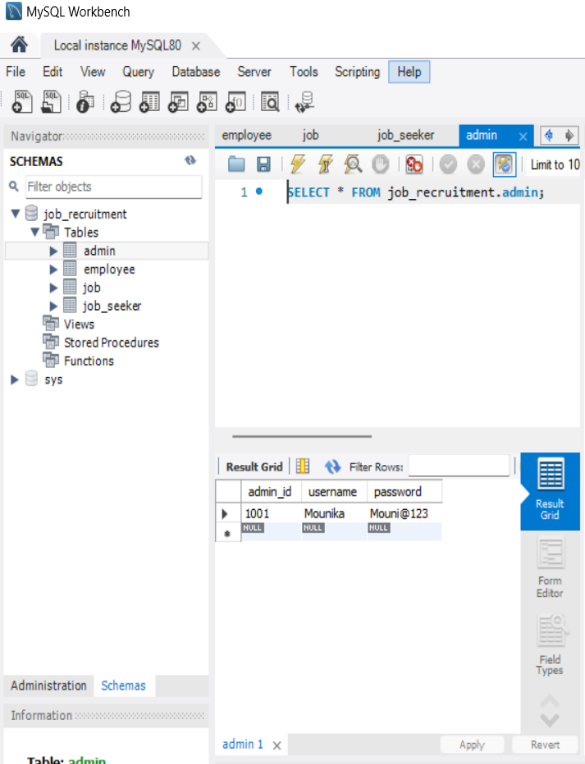
****

Fig:6.1 Fig:6.2

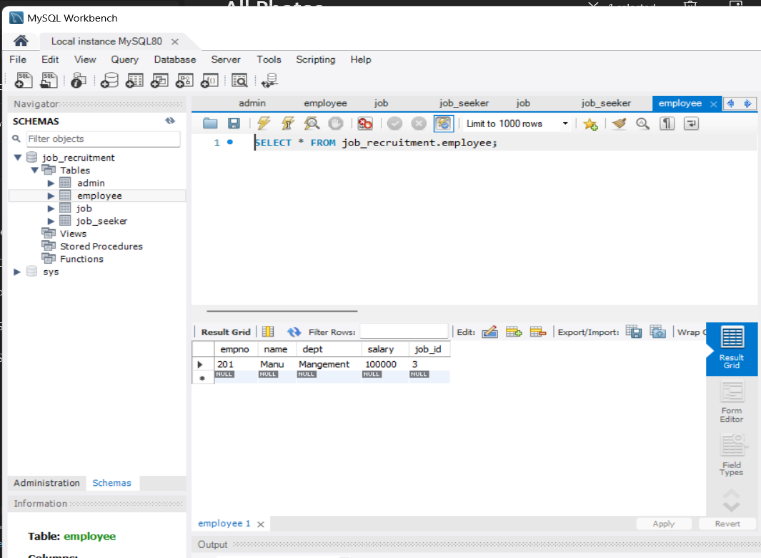
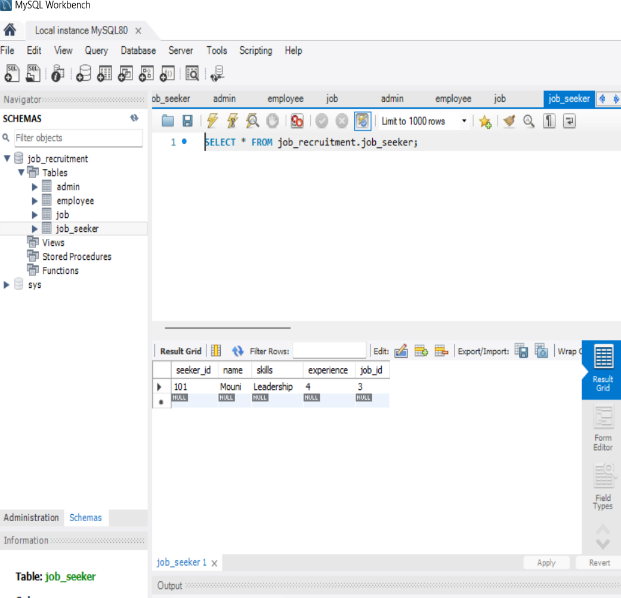
****

Fig:6.3 Fig:6.4

Record updated

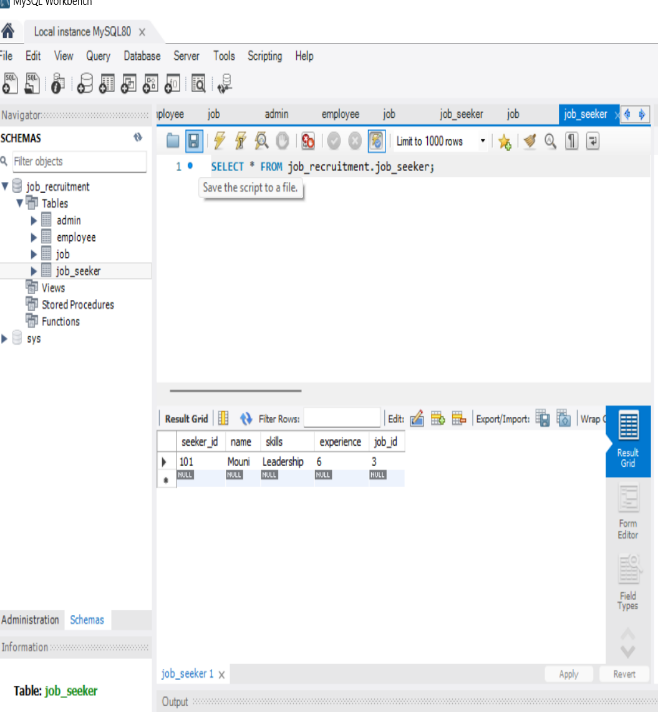
****

Fig:6.5

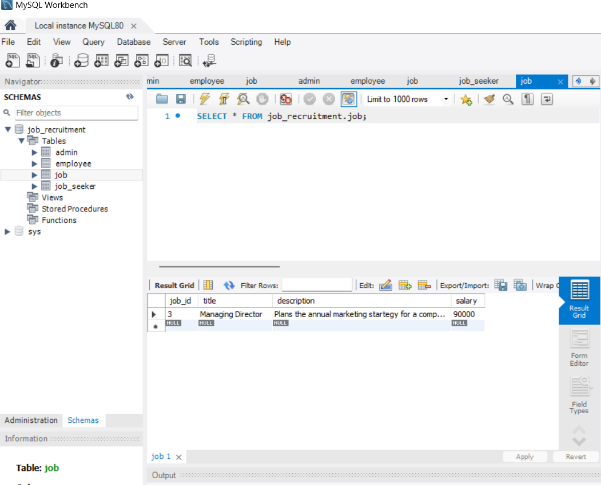
Record ****deleted

Fig:6.6

**CHAPTER 7**

**CONCLUSION**

Furthermore, our Job Recruitment Portal goes beyond conventional platforms by continuously enhancing and adapting its features. Real-time analytics and feedback mechanisms enable iterative improvements, ensuring the portal aligns with emerging industry trends and user preferences. Advanced algorithms facilitate personalized job recommendations, enhancing the matching process for both candidates and employers. Ongoing efforts to integrate emerging technologies, such as AI-driven resume parsing and video interviewing capabilities, underscore our commitment to staying at the forefront of innovation. This dedication to constant evolution positions our portal as a dynamic and future-ready solution, consistently providing users with a cutting-edge and effective tool for navigating the competitive job market.

**REFERENCES**

[**https://vertabelo.com/blog/designing-a-database-for-a-recruitment-system/**](https://vertabelo.com/blog/designing-a-database-for-a-recruitment-system/)

[**https://data-flair.training/blogs/online-job-portal-python-project/**](https://data-flair.training/blogs/online-job-portal-python-project/)

[**https://app.diagrams.net/**](https://app.diagrams.net/) **- For drawing ER and Schema diagram**