Mini project report on

Placement Management

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE23CS351A – DBMS Project

Submitted by:

Rithvik Rajesh Matta	**PES2UG23CS485**
Rishi A Sheth	**PES2UG23CS479**

under the guidance of

Prof. Shilpa S

Assistant Professor

PES University

AUG , DEC 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Placement Data Management System

is a bona fide work carried out by

Rithvik Rajesh Matta	**PES2UG23CS485**
Rishi A Sheth	**PES2UG23CS479**

In partial fulfillment of the completion of the fifth semester DBMS Project (UE20CSS301) in the Program of Study ,Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2022 – DEC. 2022. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements with respect to project work.

Signature

Prof. Shilpa S

Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Placements Data Management System** has been carried out by us under the guidance of **Prof. Nivedita Kasturi, Assistant Professor,** and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2023.

Rithvik Rajesh Matta	PES2UG23CS485
Rishi A Sheth	PES2UG23CS479

ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Nivedita Kasturi, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE21CS351, DBMS Project.

We take this opportunity to thank Dr. Sandesh B J, C, Professor,ChairPerson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice,Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

The main goal of our project is to design and implement a complete Placement Management System that makes the entire placement process in colleges faster, smarter, and easier to manage. We noticed that most placement activities are still carried out manually, which causes confusion, errors, and delays. Students have to fill in multiple forms, placement officers must maintain spreadsheets, and companies often have to communicate through emails. We wanted to solve this by building a single online system that connects students, companies, and placement officers on one platform and automates the workflow using a well-designed database and a simple web interface.

Our system is developed using MySQL as the database and Streamlit as the front-end framework. The project started with a detailed Entity-Relationship (ER) diagram that showed how all the main entities Students, Departments, Courses, Companies, Job Postings, Applications, Interviews, and Placement Officers—are related to each other. We then converted that ER diagram into a relational database model using proper mapping techniques, adding primary keys, foreign keys, and relationships that match real-life scenarios in a college placement cell. Each table was created through Data Definition Language (DDL) statements and filled with realistic sample data using Data Manipulation Language (DML) commands.

Once the data structure was ready, we implemented stored procedures, functions, and triggers to make the system more intelligent. These database components allowed automatic actions, such as setting a default application status or checking job eligibility based on a student's CGPA. For example, the `Check_Eligibility` function directly compares a student's CGPA with the job requirement and instantly returns whether the student is eligible. Similarly, triggers helped us ensure data consistency by automatically updating or logging changes when records were modified. This automation reduced the chances of manual errors and made our database reliable and self-maintaining.

The front-end of the project was built using Streamlit, which allowed us to quickly develop an interactive and user-friendly web interface. The app connects directly to the MySQL database through a secure connection and provides separate sections for students, companies, and placement officers. Students can view available jobs, apply for positions, and check their application status. Companies can post new job openings and review applications. Placement officers can manage all data, generate placement reports, and view analytics like the number of placed students or the highest salary package.

We also tested the system thoroughly by executing several SQL queries, including simple selections, grouped and aggregate queries, correlated and nested queries, and update or delete operations. These queries proved that our database relationships were working correctly and that the foreign key constraints maintained data integrity at all times.

In conclusion, our Placement Management System provides a centralized, efficient, and transparent solution for managing campus recruitment activities. It reduces paperwork, improves communication, and allows real-time tracking of placement progress. By combining a structured relational database with stored logic and a simple web interface, we created a practical and modern tool that can easily be adopted by any college or university placement department to handle recruitment processes more effectively and accurately.

TABLE OF CONTENTS

1. INTRODUCTION

In many colleges and universities, managing the placement process, where students apply for jobs, companies post roles, and placement officers coordinate everything, can become a real headache. We decided to build a system to help make this smoother. A lot of work is still done manually such as students fill out forms, placement cells send out emails or print notices, companies come by and collect spreadsheets, interviews get scheduled by hand, and tracking who applied where becomes chaotic. We felt that by creating a dedicated online system, we could reduce all that manual labour and make things simpler and more transparent for everyone involved.

We designed a system where students, placement officers, and companies all have their roles, where student profiles and eligibility criteria are stored in a database, where companies can post jobs, where students can apply, and where the placement cell can monitor everything. We also added features like tracking application status, updating job posts, linking students to jobs based on eligibility, and generating simple reports. The goal is not to create something overly complex, but to build a working application that actually gets used, reduces errors, saves time, and makes the process more visible to all parties.

From what we found in the literature, many placement systems suffer from manual bottlenecks, duplication of data, tedious record,keeping, lack of centralized information, and difficulty in matching students with jobs. For example, Anjali et al. described a web,based placement management system for a college where manual work was heavy, data redundancy high, and paper consumption large [1]. Another study highlighted that a centralised placement,cell management system improved efficiency, lowered manual effort, and supported better coordination between students, the training & placement cell, and companies [2]. These findings convinced us that building a digital placement portal is not only useful but timely.

In our project, we start with a relational database that holds information about students (their personal details, academic info, eligibility CGPA etc.), departments, courses, companies, job postings, applications, interviews, and a placement officer. On top of that database, we built a user interface, using a tool like Streamlit to allow different user types to log in and perform their tasks. Students can view job listings, apply, check status, companies can view applicants, post new jobs, the placement officer can add jobs, schedule interviews, view which students have been placed, generate reports on placement statistics, all via the system.

One of our key focuses was usability and real,world workflow alignment, we didn’t want to design something that only worked in theory but would fail in practice. That meant we kept things simple: clear dashboards, appropriate views for each user type, eligibility filters so that students only see jobs they qualify for, status updates, effective notifications (could be emails or onscreen messages), and logging of placements so that we can later track metrics like number of students placed per drive or average salary package.

Also, we set things up so that as new companies come in, the placement cell can simply add their details, post jobs, set deadlines and eligibility criteria, and students automatically see updated listings. This reduces the manual communication overhead (e,mails to each student, printing lists, manual filtering) and helps ensure fewer students miss out on opportunities. From the literature, one of the complaints is that under manual systems students may not get timely updates or some eligible students may be left out because filtering is done by hand [3].

In summary, we have built a placement management application that handles the full cycle: student profile management, company job posting, application submission, interview scheduling, placement tracking, and reporting. Our hope is that this system can be deployed in a college environment, reduce manual labour, increase transparency, and help students, companies and placement officers all be on the same platform. Next we will articulate the problem statement which this project addresses.

2. PROBLEM DEFINITION

We have observed that in many academic institutions, the training and placement process suffers from a number of recurring issues. First, **student data is often scattered or maintained manually** – students fill forms in the department, placement officers then transfer them into spreadsheets or paper files, companies' eligibility criteria are communicated by email or printouts, job details are posted manually, and tracking which students applied or are eligible is difficult. Because of this manual method, **errors creep in** – wrong CGPA values, outdated resumes, missing students, confusion over deadlines, etc. As described in one study: the manual system is "extremely complicated and time,consuming" and difficult to update when the number of students increases [1].

Second, **communication between students, placement cell and companies is slow or inefficient**. Students may not get timely notifications of new job postings or company eligibility criteria. Companies may not get access to all eligible student data in time. Placement officers have to coordinate via many emails, phone calls, printouts, and face delays. One paper notes that the manual system "requires a lot of manpower and time" and the information flow is not efficient [4].

Third, **matching students to job criteria is cumbersome**. Eligibility criteria like minimum CGPA, department, skill sets, application deadlines keep varying per company. Manually filtering or printing lists means that some students might miss opportunities, some may apply to jobs they're not eligible for, and placement officers may not get accurate lists of applicants. One of the reviewed opensource systems pointed this out: the goal was to ensure only eligible students get a chance, remove duplication, and make filtering efficient [5].

Fourth, **tracking and reporting placements and analytics is weak or absent**. Many institutions struggle to generate meaningful statistics: how many students are placed per company, average salary package, which departments are lagging, how many students applied vs got selected. Without this, the placement cell cannot make data,driven decisions, and the institution cannot showcase its placement success convincingly. The research on placement management systems emphasises the need for dashboards, real,time reporting, and analytics [2].

Given all this, our problem statement is:
We need to design and implement a placement management system that will **centralise and automate** the process of student profile handling, job posting by companies, application submission by students, eligibility filtering, interview scheduling, status tracking, and reporting. This system must be easy to use, reduce manual effort, reduce errors, speed up communication, and provide visibility to all stakeholders (students, placement cell, companies). Specifically, the system should:

- Allow students to register once with their details and then update as needed, and reuse that data for all job applications.
- Allow placement officers to add companies, set job postings (with criteria like CGPA, department, deadline, location, salary package), and monitor student applications and results.

- Allow companies (or placement cell on their behalf) to view eligible student lists, shortlisted lists, schedule interviews, update application status, and record final placements.

- Filter job listings so that students only view jobs they are eligible for (based on CGPA, department, etc.), thus reducing mismatches.

- Provide status visibility: students should be able to see which stage their application is in (applied, shortlisted, interview scheduled, selected, rejected).

- Provide placement cell dashboards and reports: number of applicants per job, number of selected per company, average salary, placement rate per department, etc.

- Store all data in a database so that centralised records are maintained, reduce duplication, redundancy, paper work, manual spreadsheets.

- Improve communication: when a company posts a job, students are promptly notified, when interview scheduled, students and companies are informed, placement results captured in system.

- Be scalable and usable for future growth: as number of students, companies, jobs increases, system should still perform, easy to maintain and extend.

We have done the work of building the relational database schema (students, departments, courses, companies, job postings, applications, interviews, placement officers) and then implemented the application layer (using e.g., Streamlit) with user,friendly pages for student portal, company management, job postings, applications, and reports. This addresses the problem of manual and fragmented processes by providing a unified system. In doing so, our aim is to help institutions achieve better placement outcomes, improve transparency, reduce workload, and support data,driven decision making.

In short: the problem is that the existing placement operations are largely manual, error,prone, slow, and opaque. Our solution is to automate and streamline the end,to,end placement workflow via a web,based system, bringing students, companies and placement officers onto one platform, enabling faster, smarter, and safer operations.

3. ER MODEL

In our project we designed an ER diagram that shows how everything in our placement system is connected together. We wanted to make it clear how students ,companies, jobs ,postings , applications, interviews , departments courses and placement officers all link with one another. The ER diagram became the base of our whole database because it helped us plan what tables to make what attributes to include and how data flows between them

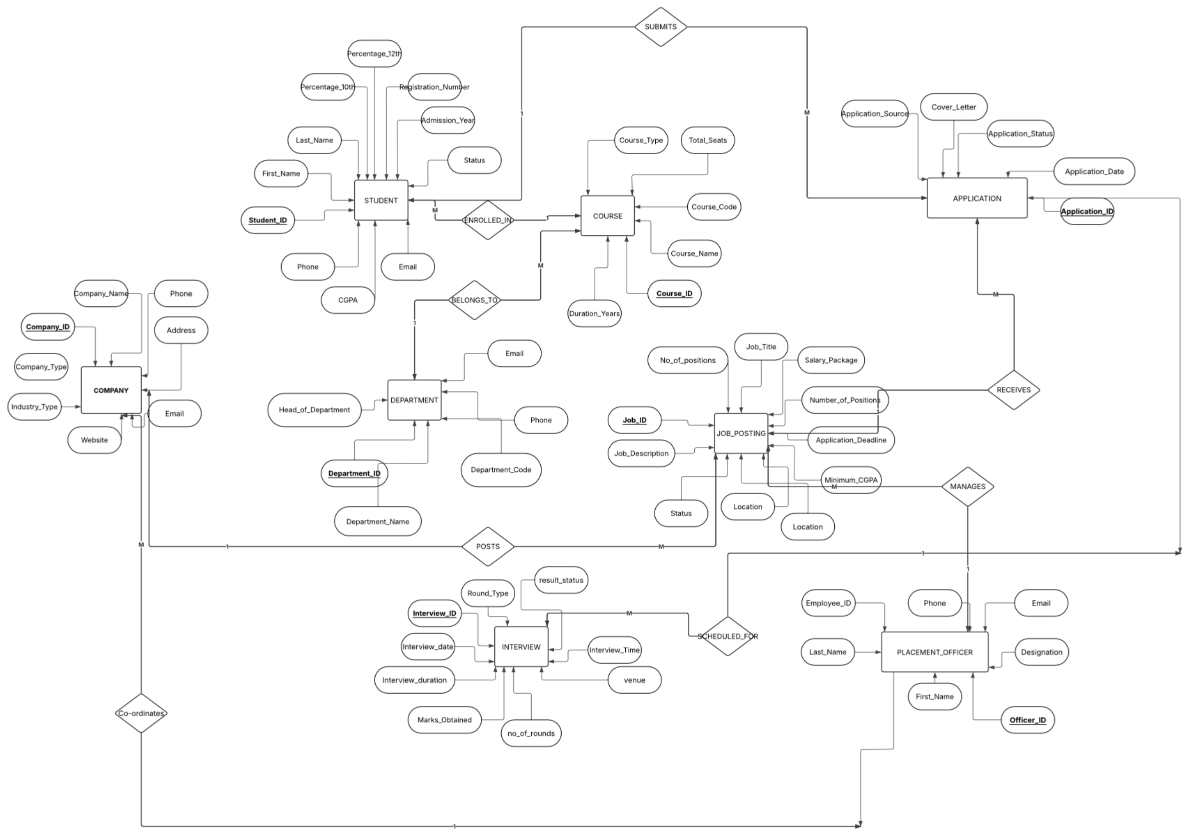


Fig 3.1 - ER Diagram.

We started by adding the **Student entity** because that is the main part of the system. Each student has details like first name last name email phone CGPA tenth and twelfth percentage registration number admission year and status. We also connected each student with a department and a course. The relationship between student and department is many to one because many students can belong to one department. Similarly a course can have many students but each student belongs to one course. This connection helps the placement cell easily filter students when a company has a specific branch or course requirement.

Then we created the **Department entity** which stores department id department name department code head of department phone and email. This table helps in grouping students and linking courses. We wanted every student and every course to be tied to a valid department so we used foreign keys to link them.

Next we designed the **Course entity** that keeps course id course name course code duration in years course type and total seats. We connected this with the department entity so that the placement cell can later pull information like how many students are in each course and how many seats are filled. It also helps to know which department offers which courses.

After handling student and academic parts we focused on the company side. We added the **Company entity** with details such as company id company name company type phone industry type website email and address. We wanted each company to be able to post multiple job openings so we connected the company entity with the job posting entity in a one to many relationship.

The **Job Posting entity** includes job id company id job title description salary package location minimum CGPA application deadline and number of positions. This table is very important because most actions in the system revolve around jobs. Students apply to jobs companies post them placement officers review them and interviews are scheduled based on them. That is why we made sure every job posting references a valid company.

Once we had students and jobs we added the **Application entity**. This table connects students and jobs using many to many relationships resolved by the application table. A student can apply for many jobs and a job can have many applicants. In the application table we stored application id student id job id application date application source cover letter and application status. We also added triggers and functions in our SQL code to automatically set the status as under review when a new application is inserted.

Then comes the **Interview entity** which connects to the application table. For every application there can be one or more interviews depending on how many rounds the company conducts. We stored interview id application id interview date interview time duration number of rounds venue round type marks obtained and result status. This table helps track how far each student has gone in the selection process and also helps generate reports for the placement cell.

We also made a **Placement Officer entity** with officer id employee id first name last name designation phone and email. This table represents the staff member who manages the entire placement process. The officer coordinates with companies, schedules interviews manages student data and checks placement reports.

In our ER diagram we also have connecting relationships like enrolled in belongs to posts submits receives scheduled for manages and coordinates. These links make the database strong and prevent duplication of data. For example the relationship enrolled in connects student and course the relationship belongs to connects course and department the relationship posts connects company and job posting the relationship submits connects student and application and the relationship scheduled for connects application and interview.

We kept the relationships simple because we wanted the data model to match real workflows. We avoided unnecessary complexity so that queries run faster and updates are easier. This ER diagram gave us a clear picture of how data moves through the system. It also helped us write SQL commands more easily since we already knew which foreign keys and constraints to use.

Overall our ER diagram represents the real life process of placements in a college. It shows how a student moves from being enrolled in a course to applying for a job to attending interviews and finally getting placed. It connects every small detail like who manages what and who belongs where in a single visual map. Because of this diagram we were able to design a working placement management system that feels complete and organized and that actually supports the day to day placement activities without confusion.

4. ER TO RELATIONAL MAPPING

When we finished drawing the ER diagram, the next step we did was to convert it into relational tables so that it can actually be used in our MySQL database. The ER diagram only shows the logical structure and connections between entities, but to store and manage data properly, we needed to map everything into relations with primary keys, foreign keys, and proper attributes. We followed a simple step-by-step process that helped us get from the diagram to the SQL tables that we finally implemented.

The main goal here was to make sure that every entity and relationship from the ER model was represented correctly in the database. We wanted the mapping to keep all the connections intact like which student belongs to which department, which company posted which job, which student applied for which job, and so on. This way, when we run SQL queries later, we can easily join and fetch data across different tables.

4.1 STEPS OF ALGORITHM FOR CHOSEN PROBLEM

We followed a clear set of steps to convert our ER diagram into a relational model. These are the steps we used while designing the placement management database.

Step 1: Mapping of Strong Entities

We started by identifying all strong entities in our ER diagram. In our case, these were STUDENT, DEPARTMENT, COURSE, COMPANY, JOB_POSTING, INTERVIEW, APPLICATION, and PLACEMENT_OFFICER.

For each strong entity, we created a separate table. Every table had its own primary key attribute to uniquely identify each record. For example, Student_ID for the student table, Department_ID for department, Company_ID for company, and so on.

Step 2: Mapping of Weak Entities (if any)

In our model, we did not really have a weak entity because all our entities were strong and had their own primary key. Every table could exist independently with its own identifier.

Step 3: Mapping of Relationships (One-to-Many)

For relationships like “A student belongs to one department” or “A course belongs to one department,” we added foreign keys. The STUDENT table got a Department_ID and Course_ID column as foreign keys linking it to the DEPARTMENT and COURSE tables.

Similarly, the JOB_POSTING table got a Company_ID as a foreign key connecting it to the COMPANY table. This maintained the one-to-many relationships properly.

Step 4: Mapping of Many-to-Many Relationships

We had a few many-to-many relationships, such as “students apply to many jobs” and “jobs have many applicants.”

To represent this, we created the APPLICATION table. It connects STUDENT and JOB_POSTING using Student_ID and Job_ID as foreign keys. This table also holds extra details like Application_Date, Source, Cover_Letter, and Application_Status. This is one of the most important mapping steps since it converts the many-to-many connection into a working relational table.

Step 5: Mapping of Attributes

We carefully mapped all simple and composite attributes into individual columns. For example, in STUDENT we added fields like First_Name, Last_Name, Email, Phone, CGPA, and so on. Each column has a suitable data type and constraint. We also added unique constraints to Email and Registration_Number because they must be different for each student.

Step 6: Mapping of Derived Attributes

In our system, placement status is a derived attribute because it can be figured out from the interview result. Instead of storing it as a fixed column, we created a stored function Get_PlacementStatus(studentId) in MySQL to compute it dynamically.

Step 7: Mapping of Relationships with Attributes

Some relationships have their own attributes. For example, in the INTERVIEW table, which is linked to APPLICATION, we included attributes like Interview_Date, Time, Duration, and Result_Status. This table captures both the connection and the related data.

Step 8: Ensuring Referential Integrity

Finally, we added all the necessary foreign key constraints so that data stays consistent. For example, a student cannot apply for a job if that job does not exist, and a job cannot exist without a company. This helps maintain clean and valid data across the system.

By following all these steps, we converted our ER model into a complete relational model that could be directly implemented in MySQL.

4.2 COMPLETE DIAGRAM OF RELATIONAL MAPPING

Our complete relational mapping diagram shows how all the tables are connected after converting from the ER diagram. It looks like a big web where every table links to at least one other table through foreign keys.

At the center, we have the STUDENT table which connects to DEPARTMENT and COURSE. The STUDENT table also connects to the APPLICATION table through the Student_ID. The APPLICATION table then connects to JOB_POSTING using Job_ID.

The JOB_POSTING table is linked to COMPANY through Company_ID which means each company can post many jobs. The INTERVIEW table connects to the APPLICATION table using Application_ID which shows that each application can have one or more interviews. The PLACEMENT_OFFICER table stands on its own but manages and monitors all activities in the placement process.

So if we trace the flow in simple terms it goes like this
A student belongs to a department and course
A company posts multiple job postings
A student applies to a job through an application
That application is linked to interviews
The placement officer manages everything and generates reports

Every relationship from the ER diagram has been carefully represented through foreign keys and relational links. This final mapping forms the structure of our database which we then used to write SQL commands and store real data.

This complete relational mapping makes sure there is no redundancy and that every piece of information is stored exactly where it belongs. It also supports complex queries like finding which students are placed in which company or checking who attended how many interviews.

Overall the relational mapping diagram is the perfect bridge between our ER design and our actual database implementation. It turned our conceptual plan into a working structure that powers the entire placement management system.

5. DDL STATEMENTS

In this part we wrote and executed all the DDL statements for creating the tables that we planned from our er diagram and relational mapping
DDL stands for data definition language which means it helps us to define the structure of our database tables like what columns are needed what datatypes we will use and how they are linked with foreign keys
this step was really satisfying because once we executed all these create table statements the database finally started to look like a real system instead of only a diagram on paper

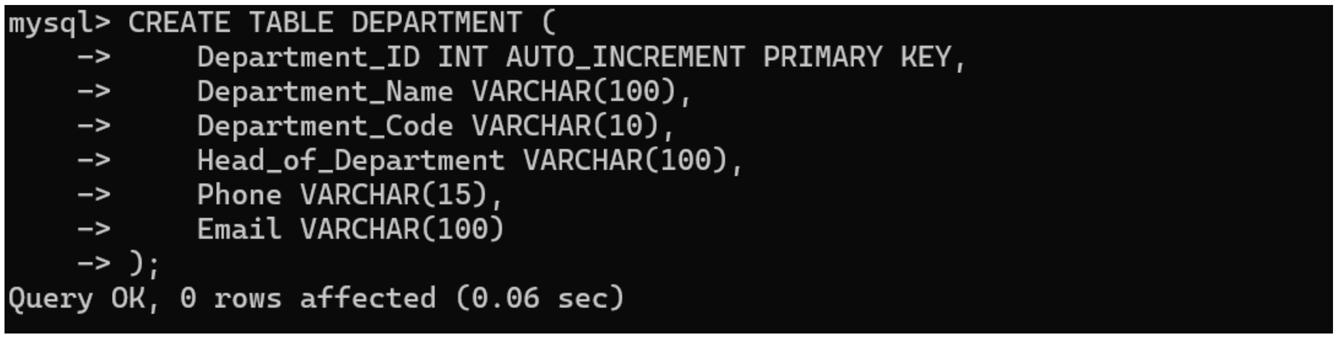
we created every table one by one in mysql workbench and tested if the foreign keys worked properly or not
whenever we executed each statement we also took screenshots from mysql workbench showing that the table was created successfully
below we are explaining each table creation with its statement and where the screenshot has to be added

5.1 Table 1 Creation – DEPARTMENT

We started with the department table because many other tables depend on it
this table stores the list of departments in our college like cse or ece

```
CREATE TABLE DEPARTMENT (
   Department_ID INT AUTO_INCREMENT PRIMARY KEY,
   Department_Name VARCHAR(100),
   Department_Code VARCHAR(10),
   Head_of_Department VARCHAR(100),
   Phone VARCHAR(15),
   Email VARCHAR(100)
);
```

this table is simple and works as a base table for many other relations



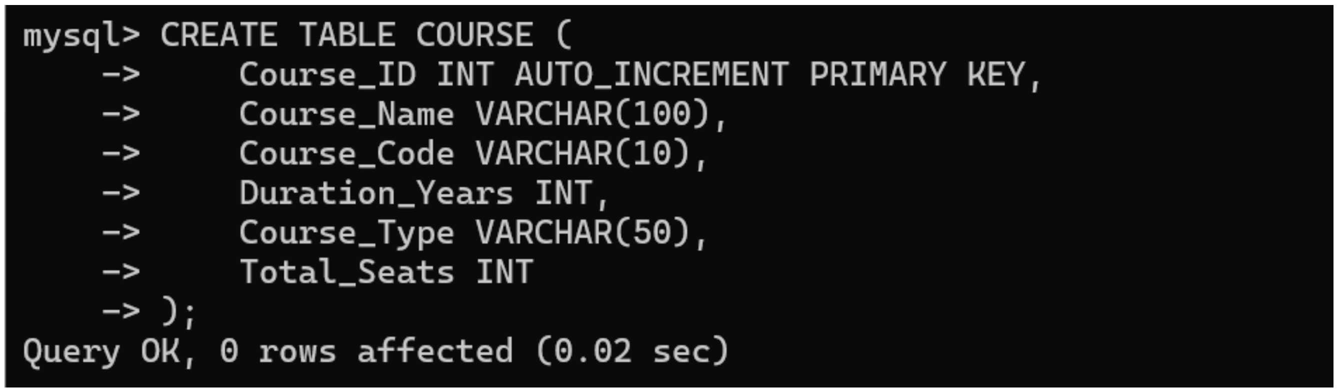
```
mysql> CREATE TABLE DEPARTMENT (
    ->     Department_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Department_Name VARCHAR(100),
    ->     Department_Code VARCHAR(10),
    ->     Head_of_Department VARCHAR(100),
    ->     Phone VARCHAR(15),
    ->     Email VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

5.2 Table 2 Creation – COURSE

After that we created the course table which connects with department

```
CREATE TABLE COURSE (
    Course_ID INT AUTO_INCREMENT PRIMARY KEY,
    Course_Name VARCHAR(100),
    Course_Code VARCHAR(10),
    Duration_Years INT,
    Course_Type VARCHAR(50),
    Total_Seats INT
);
```

each course is linked to a department in the relational mapping but at the DDL level we first create the table without foreign keys to avoid dependency issues then later we add the keys



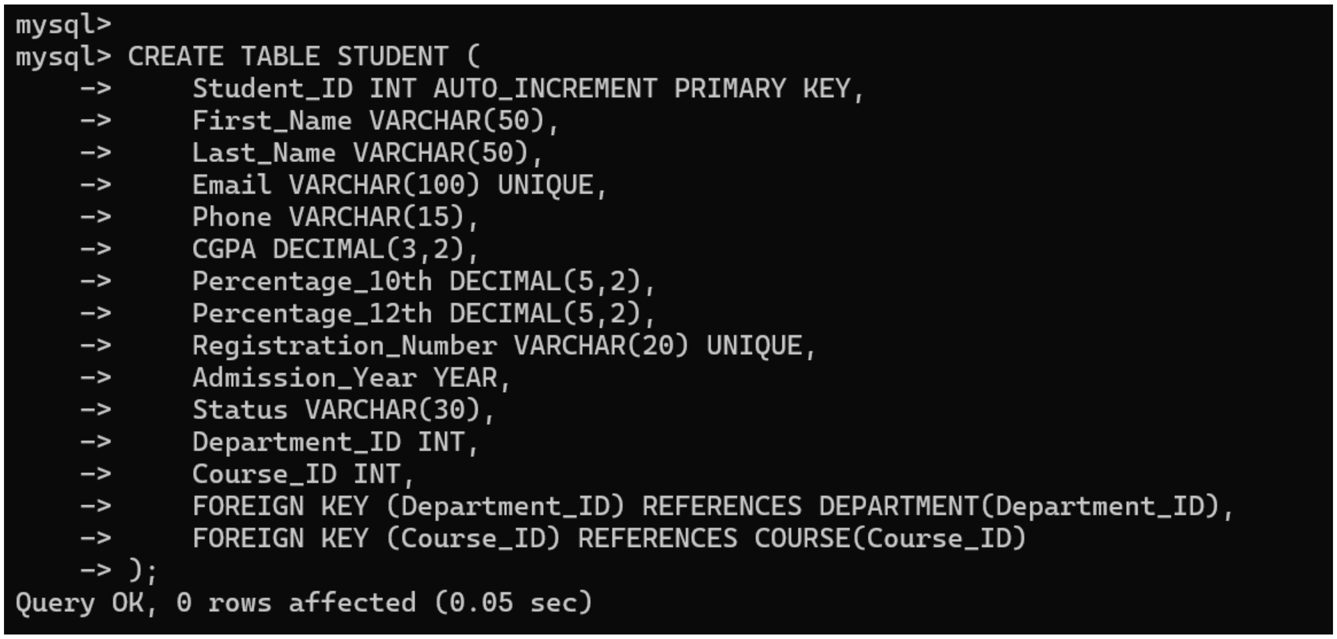
```
mysql> CREATE TABLE COURSE (
    ->     Course_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Course_Name VARCHAR(100),
    ->     Course_Code VARCHAR(10),
    ->     Duration_Years INT,
    ->     Course_Type VARCHAR(50),
    ->     Total_Seats INT
    -> );
Query OK, 0 rows affected (0.02 sec)
```

5.3 Table 3 Creation – STUDENT

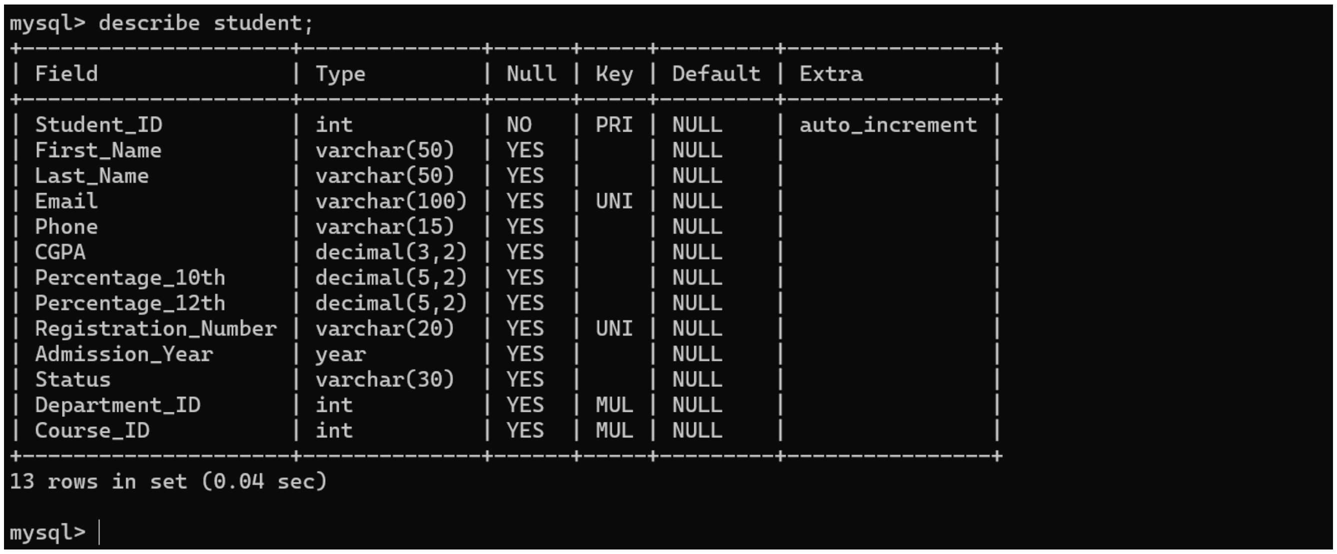
Next we created the student table which is one of the biggest and most used tables

```
CREATE TABLE STUDENT (
    Student_ID INT AUTO_INCREMENT PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Email VARCHAR(100) UNIQUE,
    Phone VARCHAR(15),
    CGPA DECIMAL(3,2),
    Percentage_10th DECIMAL(5,2),
    Percentage_12th DECIMAL(5,2),
    Registration_Number VARCHAR(20) UNIQUE,
    Admission_Year YEAR,
    Status VARCHAR(30),
    Department_ID INT,
    Course_ID INT,
    FOREIGN KEY (Department_ID) REFERENCES DEPARTMENT(Department_ID),
    FOREIGN KEY (Course_ID) REFERENCES COURSE(Course_ID)
);
```

this table keeps all student details and links each student to the correct department and course



```
mysql>
mysql> CREATE TABLE STUDENT (
    ->     Student_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     First_Name VARCHAR(50),
    ->     Last_Name VARCHAR(50),
    ->     Email VARCHAR(100) UNIQUE,
    ->     Phone VARCHAR(15),
    ->     CGPA DECIMAL(3,2),
    ->     Percentage_10th DECIMAL(5,2),
    ->     Percentage_12th DECIMAL(5,2),
    ->     Registration_Number VARCHAR(20) UNIQUE,
    ->     Admission_Year YEAR,
    ->     Status VARCHAR(30),
    ->     Department_ID INT,
    ->     Course_ID INT,
    ->     FOREIGN KEY (Department_ID) REFERENCES DEPARTMENT(Department_ID),
    ->     FOREIGN KEY (Course_ID) REFERENCES COURSE(Course_ID)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

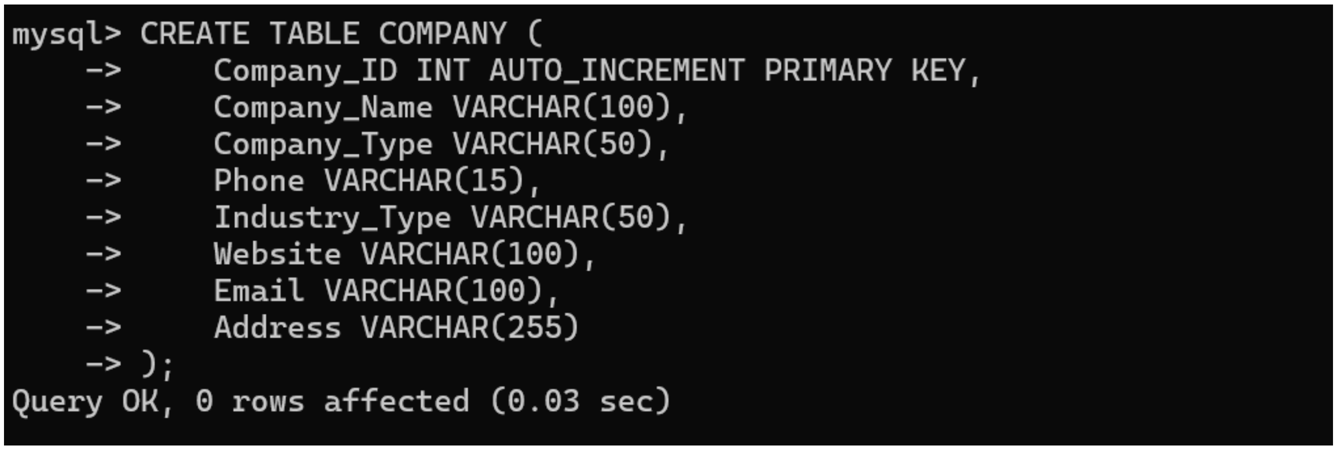


```
mysql> describe student;
+---------------------+--------------+------+-----+---------+----------------+
| Field               | Type         | Null | Key | Default | Extra          |
+---------------------+--------------+------+-----+---------+----------------+
| Student_ID          | int          | NO   | PRI | NULL    | auto_increment |
| First_Name          | varchar(50)  | YES  |     | NULL    |                |
| Last_Name           | varchar(50)  | YES  |     | NULL    |                |
| Email               | varchar(100) | YES  | UNI | NULL    |                |
| Phone               | varchar(15)  | YES  |     | NULL    |                |
| CGPA                | decimal(3,2) | YES  |     | NULL    |                |
| Percentage_10th     | decimal(5,2) | YES  |     | NULL    |                |
| Percentage_12th     | decimal(5,2) | YES  |     | NULL    |                |
| Registration_Number | varchar(20)  | YES  | UNI | NULL    |                |
| Admission_Year      | year         | YES  |     | NULL    |                |
| Status              | varchar(30)  | YES  |     | NULL    |                |
| Department_ID       | int          | YES  | MUL | NULL    |                |
| Course_ID           | int          | YES  | MUL | NULL    |                |
+---------------------+--------------+------+-----+---------+----------------+
13 rows in set (0.04 sec)

mysql>
```

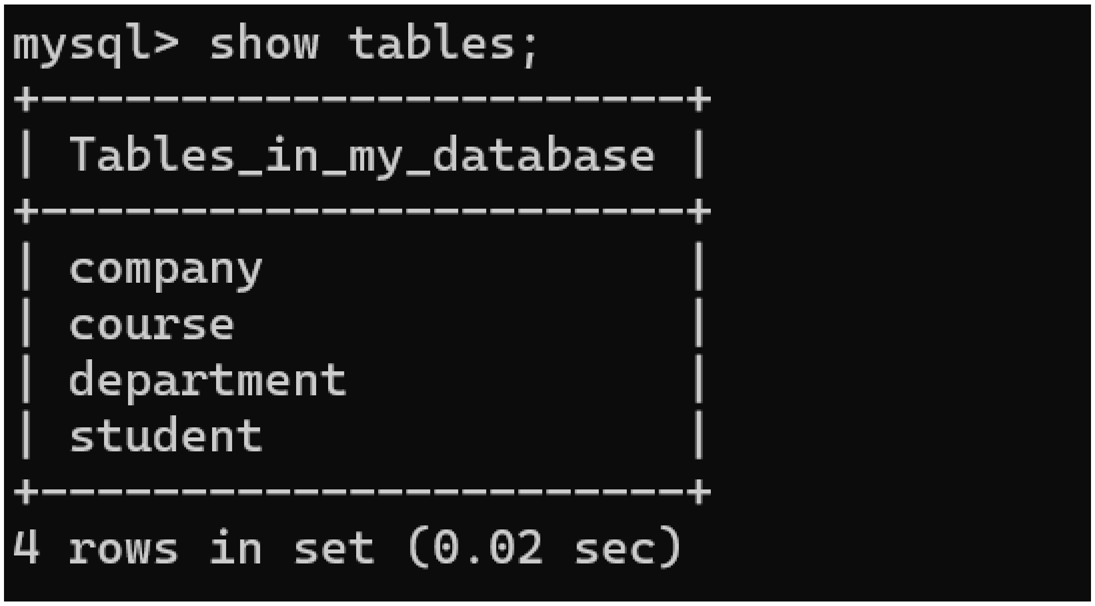
5.4 Table 4 Creation – COMPANY

Then we made the company table which will store the details of companies that visit the college

```
CREATE TABLE COMPANY (
   Company_ID INT AUTO_INCREMENT PRIMARY KEY,
   Company_Name VARCHAR(100),
   Company_Type VARCHAR(50),
   Phone VARCHAR(15),
   Industry_Type VARCHAR(50),
   Website VARCHAR(100),
   Email VARCHAR(100),
    Address VARCHAR(255)
);
```



```
mysql> CREATE TABLE COMPANY (
    ->     Company_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Company_Name VARCHAR(100),
    ->     Company_Type VARCHAR(50),
    ->     Phone VARCHAR(15),
    ->     Industry_Type VARCHAR(50),
    ->     Website VARCHAR(100),
    ->     Email VARCHAR(100),
    ->     Address VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.03 sec)
```



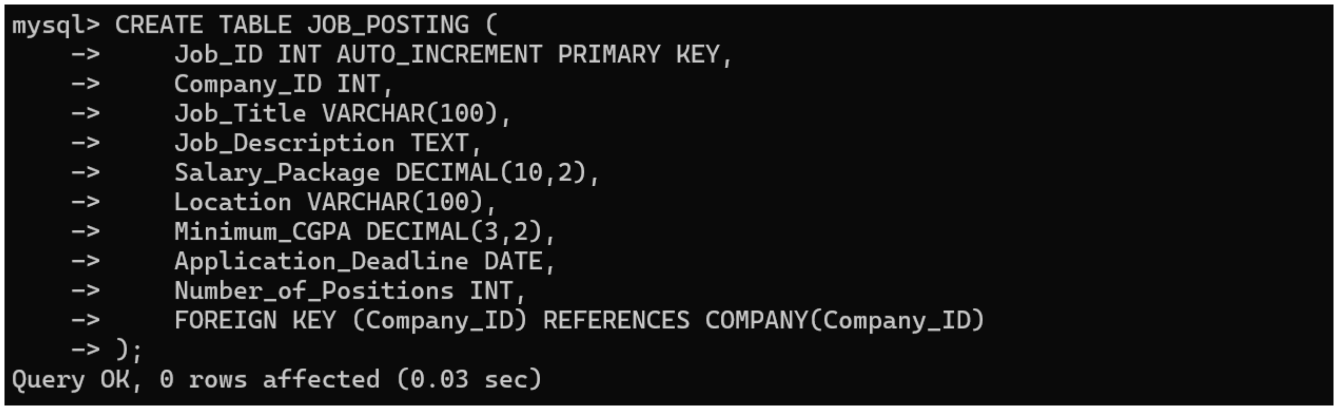
```
mysql> show tables;
+-----------------------+
| Tables_in_my_database |
+-----------------------+
| company               |
| course                |
| department            |
| student               |
+-----------------------+
4 rows in set (0.02 sec)
```

5.5 Table 5 Creation – JOB_POSTING

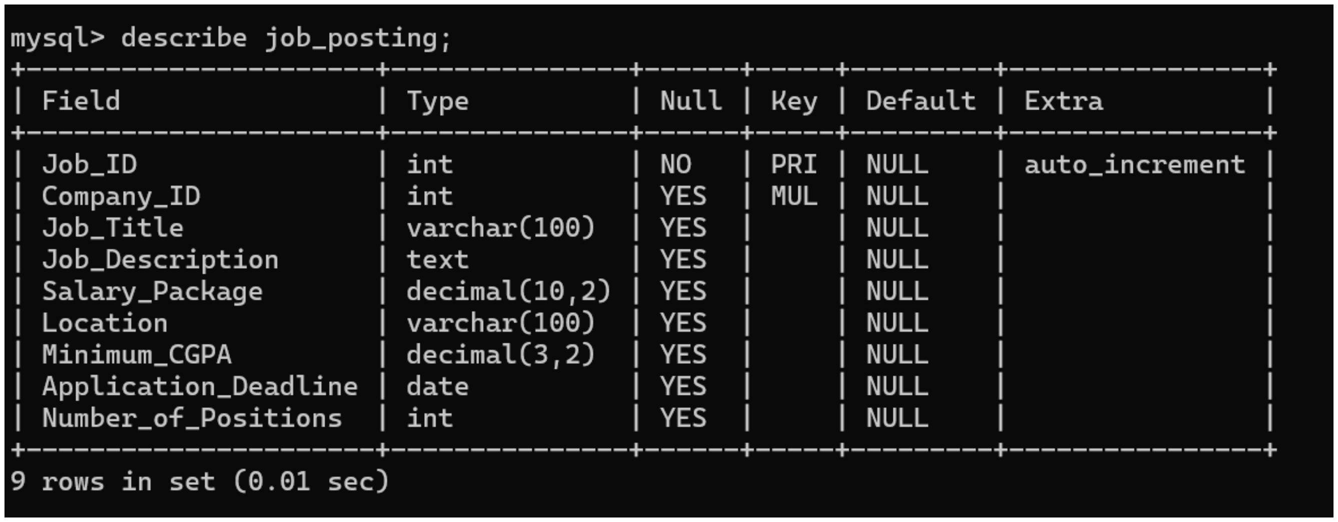
After adding companies we created job posting table because companies use it to post job details

```
CREATE TABLE JOB_POSTING (
    Job_ID INT AUTO_INCREMENT PRIMARY KEY,
    Company_ID INT,
    Job_Title VARCHAR(100),
    Job_Description TEXT,
    Salary_Package DECIMAL(10,2),
    Location VARCHAR(100),
    Minimum_CGPA DECIMAL(3,2),
    Application_Deadline DATE,
    Number_of_Positions INT,
    FOREIGN KEY (Company_ID) REFERENCES COMPANY(Company_ID)
);
```

This table connects company and job data



```
mysql> CREATE TABLE JOB_POSTING (
    ->     Job_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Company_ID INT,
    ->     Job_Title VARCHAR(100),
    ->     Job_Description TEXT,
    ->     Salary_Package DECIMAL(10,2),
    ->     Location VARCHAR(100),
    ->     Minimum_CGPA DECIMAL(3,2),
    ->     Application_Deadline DATE,
    ->     Number_of_Positions INT,
    ->     FOREIGN KEY (Company_ID) REFERENCES COMPANY(Company_ID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

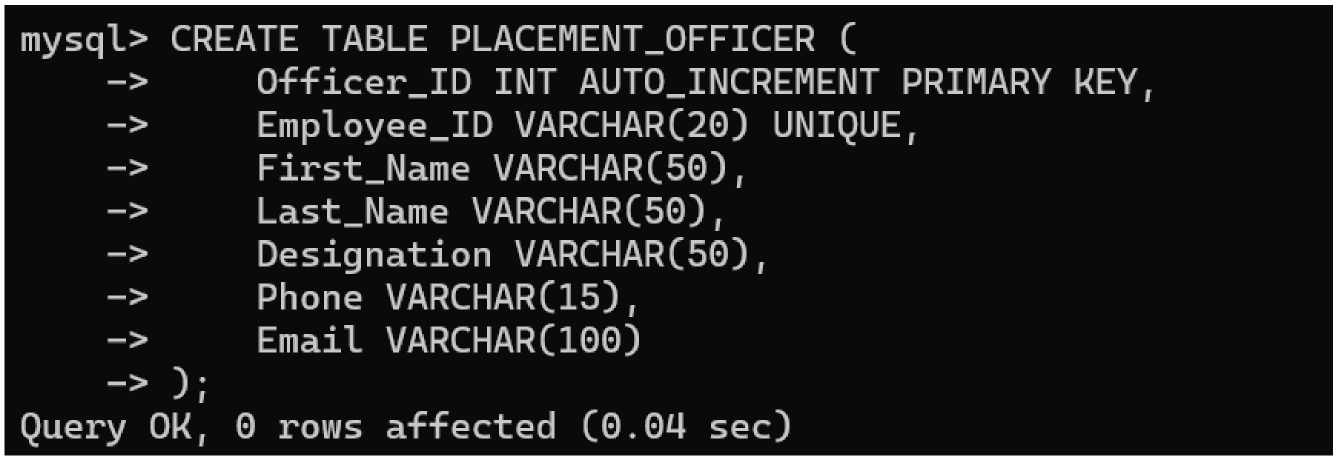


```
mysql> describe job_posting;
+----------------------+---------------+------+-----+---------+----------------+
| Field                | Type          | Null | Key | Default | Extra          |
+----------------------+---------------+------+-----+---------+----------------+
| Job_ID               | int           | NO   | PRI | NULL    | auto_increment |
| Company_ID           | int           | YES  | MUL | NULL    |                |
| Job_Title            | varchar(100)  | YES  |     | NULL    |                |
| Job_Description      | text          | YES  |     | NULL    |                |
| Salary_Package       | decimal(10,2) | YES  |     | NULL    |                |
| Location             | varchar(100)  | YES  |     | NULL    |                |
| Minimum_CGPA         | decimal(3,2)  | YES  |     | NULL    |                |
| Application_Deadline | date          | YES  |     | NULL    |                |
| Number_of_Positions  | int           | YES  |     | NULL    |                |
+----------------------+---------------+------+-----+---------+----------------+
9 rows in set (0.01 sec)
```

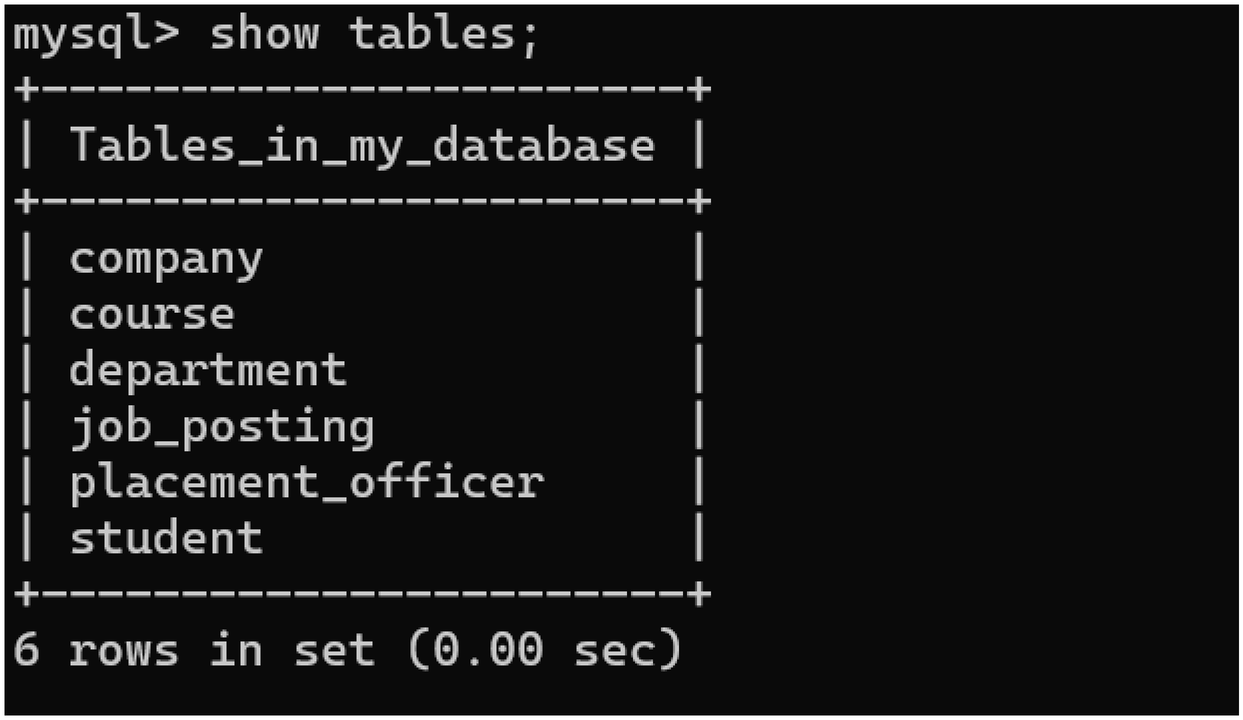
5.6 Table 6 Creation – PLACEMENT_OFFICER

We then created the placement officer table to store information of the staff handling placements

```
CREATE TABLE PLACEMENT_OFFICER (
    Officer_ID INT AUTO_INCREMENT PRIMARY KEY,
    Employee_ID VARCHAR(20) UNIQUE,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Designation VARCHAR(50),
    Phone VARCHAR(15),
    Email VARCHAR(100)
);
```



```
mysql> CREATE TABLE PLACEMENT_OFFICER (
    ->     Officer_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Employee_ID VARCHAR(20) UNIQUE,
    ->     First_Name VARCHAR(50),
    ->     Last_Name VARCHAR(50),
    ->     Designation VARCHAR(50),
    ->     Phone VARCHAR(15),
    ->     Email VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.04 sec)
```



```
mysql> show tables;
+-----------------------+
| Tables_in_my_database |
+-----------------------+
| company               |
| course                |
| department            |
| job_posting           |
| placement_officer     |
| student               |
+-----------------------+
6 rows in set (0.00 sec)
```

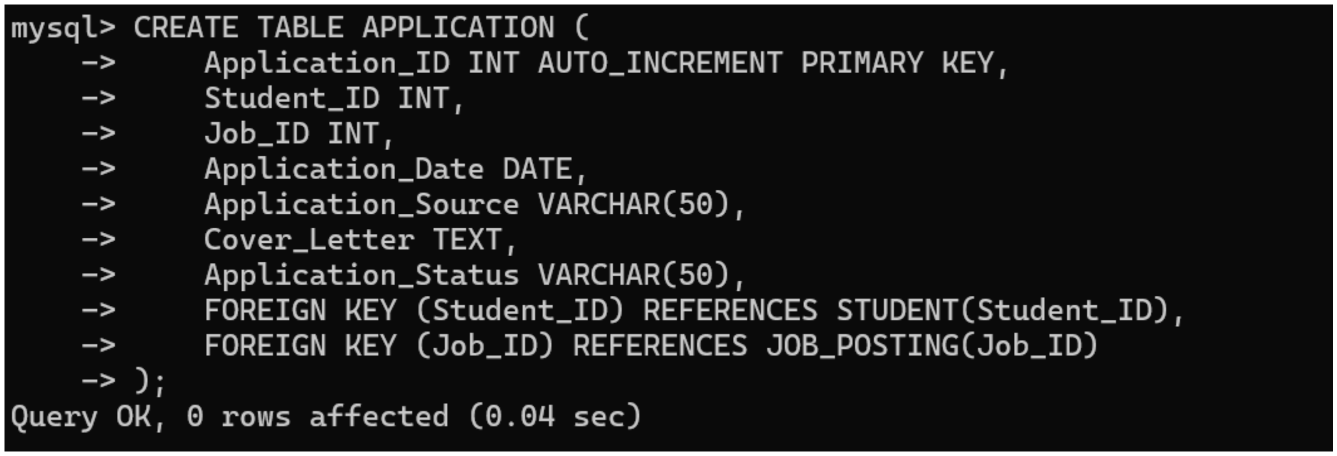
5.7 Table 7 Creation – APPLICATION

This table connects students and job postings and it is the core link between them

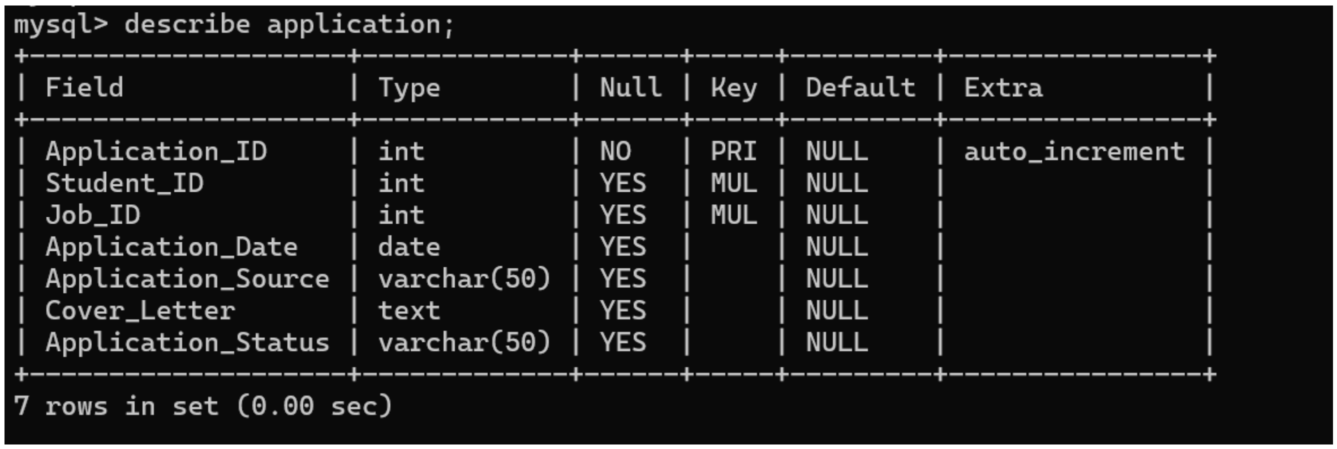
```
CREATE TABLE APPLICATION (
   Application_ID INT AUTO_INCREMENT PRIMARY KEY,
   Student_ID INT,
   Job_ID INT,
   Application_Date DATE,
   Application_Source VARCHAR(50),
   Cover_Letter TEXT,
   Application_Status VARCHAR(50),
   FOREIGN KEY (Student_ID) REFERENCES STUDENT(Student_ID),
   FOREIGN KEY (Job_ID) REFERENCES JOB_POSTING(Job_ID)
```

);

This table helps us know which student applied to which job and what is the current status of that application



```
mysql> CREATE TABLE APPLICATION (
    ->     Application_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Student_ID INT,
    ->     Job_ID INT,
    ->     Application_Date DATE,
    ->     Application_Source VARCHAR(50),
    ->     Cover_Letter TEXT,
    ->     Application_Status VARCHAR(50),
    ->     FOREIGN KEY (Student_ID) REFERENCES STUDENT(Student_ID),
    ->     FOREIGN KEY (Job_ID) REFERENCES JOB_POSTING(Job_ID)
    -> );
Query OK, 0 rows affected (0.04 sec)
```



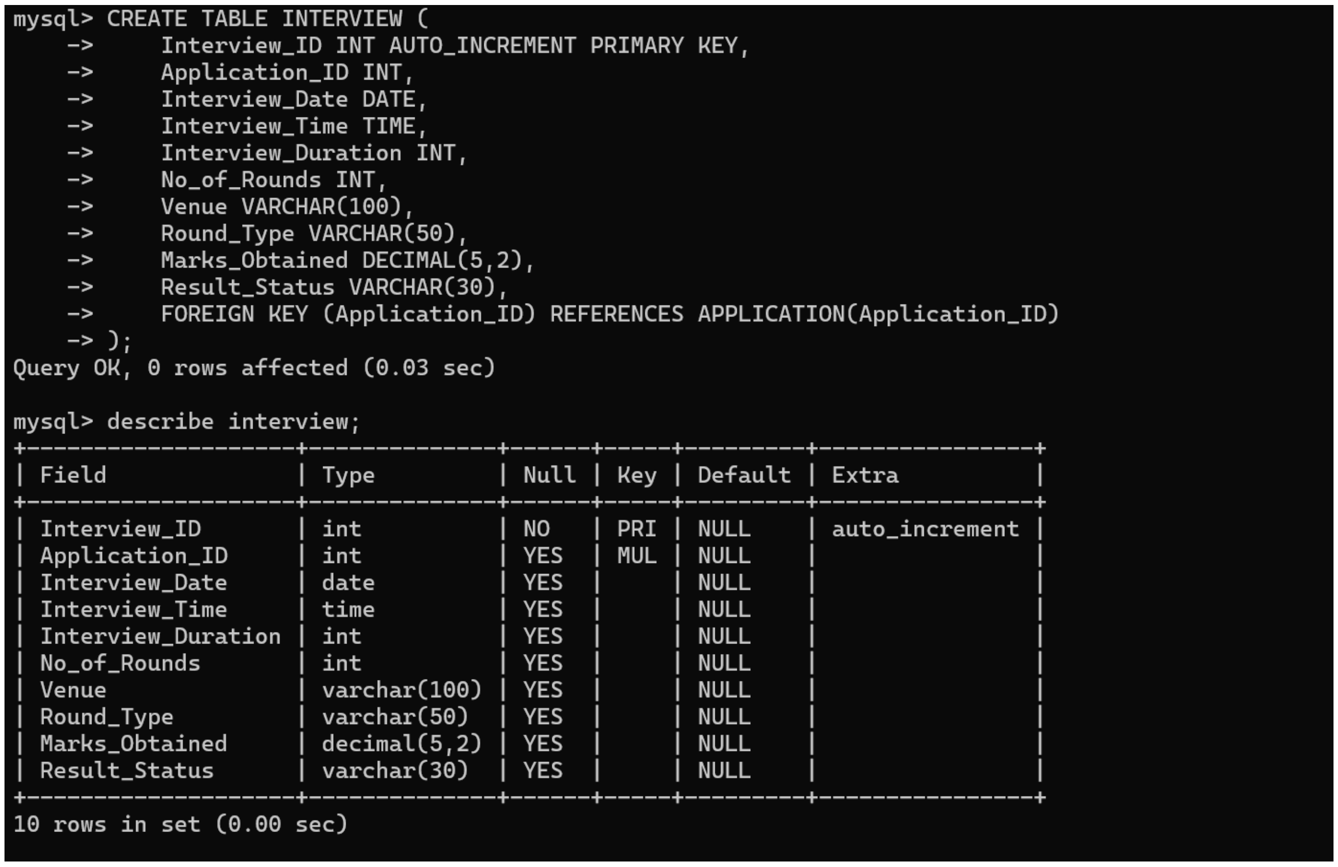
```
mysql> describe application;
+--------------------+-------------+------+-----+---------+----------------+
| Field              | Type        | Null | Key | Default | Extra          |
+--------------------+-------------+------+-----+---------+----------------+
| Application_ID     | int         | NO   | PRI | NULL    | auto_increment |
| Student_ID         | int         | YES  | MUL | NULL    |                |
| Job_ID             | int         | YES  | MUL | NULL    |                |
| Application_Date   | date        | YES  |     | NULL    |                |
| Application_Source | varchar(50) | YES  |     | NULL    |                |
| Cover_Letter       | text        | YES  |     | NULL    |                |
| Application_Status | varchar(50) | YES  |     | NULL    |                |
+--------------------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

5.8 Table 8 Creation – INTERVIEW

The interview table connects to application table and stores all interview round details

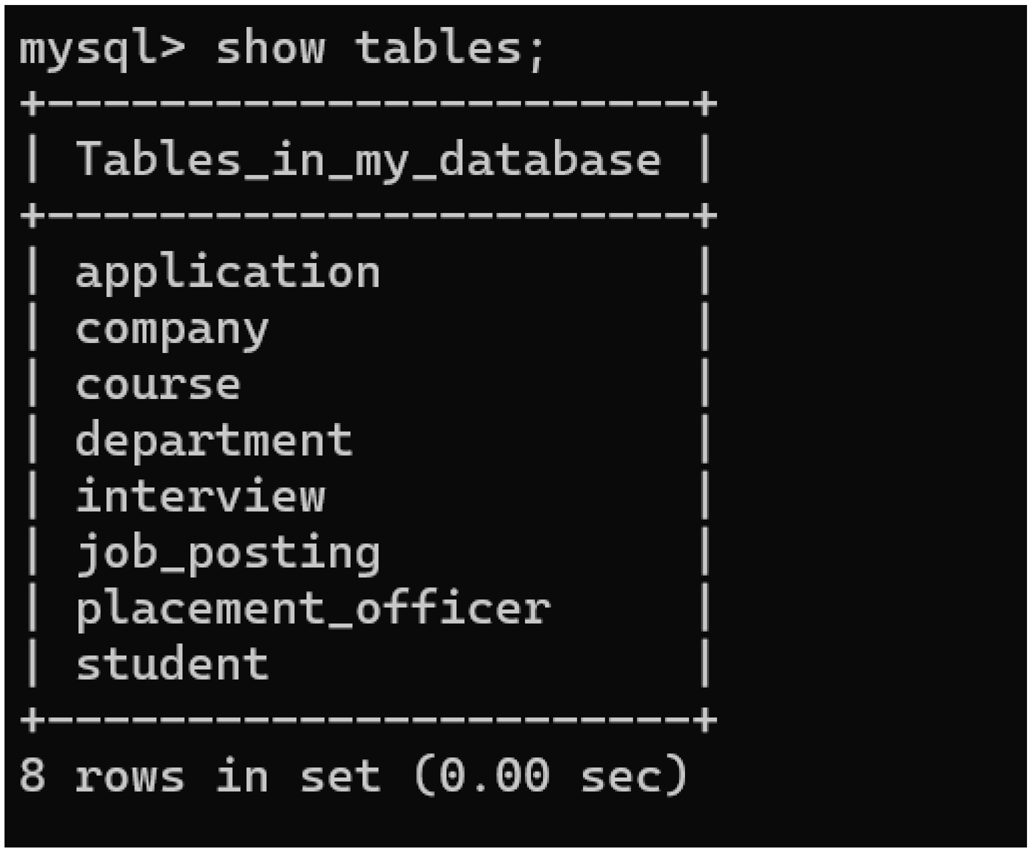
```
CREATE TABLE INTERVIEW (
   Interview_ID INT AUTO_INCREMENT PRIMARY KEY,
   Application_ID INT,
   Interview_Date DATE,
   Interview_Time TIME,
   Interview_Duration INT,
   No_of_Rounds INT,
   Venue VARCHAR(100),
   Round_Type VARCHAR(50),
   Marks_Obtained DECIMAL(5,2),
   Result_Status VARCHAR(30),
   FOREIGN KEY (Application_ID) REFERENCES APPLICATION(Application_ID)
```

);



```
mysql> CREATE TABLE INTERVIEW (
    ->     Interview_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Application_ID INT,
    ->     Interview_Date DATE,
    ->     Interview_Time TIME,
    ->     Interview_Duration INT,
    ->     No_of_Rounds INT,
    ->     Venue VARCHAR(100),
    ->     Round_Type VARCHAR(50),
    ->     Marks_Obtained DECIMAL(5,2),
    ->     Result_Status VARCHAR(30),
    ->     FOREIGN KEY (Application_ID) REFERENCES APPLICATION(Application_ID)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> describe interview;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| Interview_ID       | int          | NO   | PRI | NULL    | auto_increment |
| Application_ID     | int          | YES  | MUL | NULL    |                |
| Interview_Date     | date         | YES  |     | NULL    |                |
| Interview_Time     | time         | YES  |     | NULL    |                |
| Interview_Duration | int          | YES  |     | NULL    |                |
| No_of_Rounds       | int          | YES  |     | NULL    |                |
| Venue              | varchar(100) | YES  |     | NULL    |                |
| Round_Type         | varchar(50)  | YES  |     | NULL    |                |
| Marks_Obtained     | decimal(5,2) | YES  |     | NULL    |                |
| Result_Status      | varchar(30)  | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
10 rows in set (0.00 sec)
```

After creating all these tables we refreshed the mysql schema and confirmed that all the tables appeared under the database **CollegePlacementDB**
 We also ran describe table commands to verify that every column datatype and key matched the er diagram and relational mapping.



```
mysql> show tables;
+-----------------------+
| Tables_in_my_database |
+-----------------------+
| application           |
| company               |
| course                |
| department            |
| interview             |
| job_posting           |
| placement_officer     |
| student               |
+-----------------------+
8 rows in set (0.00 sec)
```

This whole DDL creation process was one of the most important stages of our project because it converted our planning and diagrams into an actual working database once this was done we could easily insert data test queries and connect our streamlit frontend to start building the real placement management application.

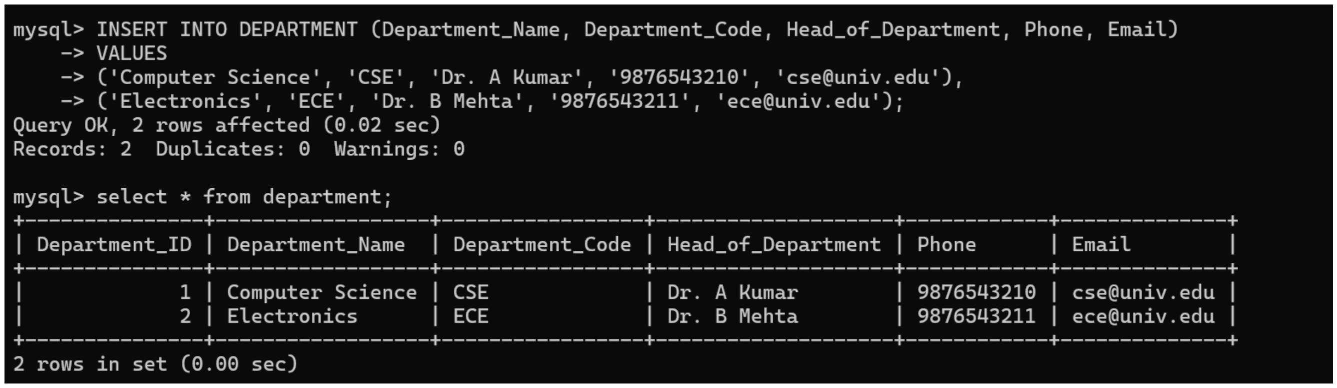
6. DML STATEMENTS

After creating all the tables using DDL statements we started inserting some real sample data into them so that we could test our placement management system properly
DML stands for data manipulation language which means it is used to insert update delete or view the actual data in the tables, in our case we mainly used insert statements to fill each table with a few records so that we could check if the foreign keys and relationships were working fine, we opened mysql workbench selected our database **CollegePlacementDB** and started inserting rows one table at a time, below we have shown the insert statements we used and mentioned where each screenshot should go.

6.1 Inserting into department

we first inserted data into the department table because many other tables depend on it

INSERT INTO DEPARTMENT (Department_Name, Department_Code, Head_of_Department, Phone, Email)
VALUES
('Computer Science', 'CSE', 'Dr. A Kumar', '9876543210', 'cse@univ.edu'),
('Electronics', 'ECE', 'Dr. B Mehta', '9876543211', 'ece@univ.edu');



```
mysql> INSERT INTO DEPARTMENT (Department_Name, Department_Code, Head_of_Department, Phone, Email)
    -> VALUES
    -> ('Computer Science', 'CSE', 'Dr. A Kumar', '9876543210', 'cse@univ.edu'),
    -> ('Electronics', 'ECE', 'Dr. B Mehta', '9876543211', 'ece@univ.edu');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from department;
+---------------+------------------+-----------------+--------------------+------------+--------------+
| Department_ID | Department_Name  | Department_Code | Head_of_Department | Phone      | Email        |
+---------------+------------------+-----------------+--------------------+------------+--------------+
|             1 | Computer Science | CSE             | Dr. A Kumar        | 9876543210 | cse@univ.edu |
|             2 | Electronics      | ECE             | Dr. B Mehta        | 9876543211 | ece@univ.edu |
+---------------+------------------+-----------------+--------------------+------------+--------------+
2 rows in set (0.00 sec)
```

6.2 Inserting into course

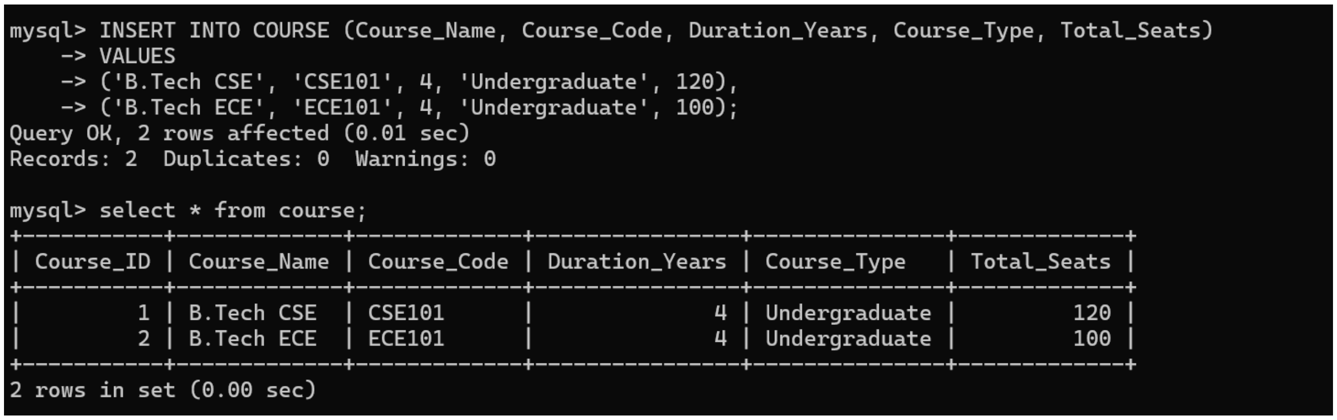
after department we added a few courses that belong to those departments

INSERT INTO COURSE (Course_Name, Course_Code, Duration_Years, Course_Type, Total_Seats)

VALUES

('B.Tech CSE', 'CSE101', 4, 'Undergraduate', 120),

('B.Tech ECE', 'ECE101', 4, 'Undergraduate', 100);



```
mysql> INSERT INTO COURSE (Course_Name, Course_Code, Duration_Years, Course_Type, Total_Seats)
    -> VALUES
    -> ('B.Tech CSE', 'CSE101', 4, 'Undergraduate', 120),
    -> ('B.Tech ECE', 'ECE101', 4, 'Undergraduate', 100);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from course;
+-----------+-------------+-------------+----------------+---------------+-------------+
| Course_ID | Course_Name | Course_Code | Duration_Years | Course_Type   | Total_Seats |
+-----------+-------------+-------------+----------------+---------------+-------------+
|         1 | B.Tech CSE  | CSE101      |              4 | Undergraduate |         120 |
|         2 | B.Tech ECE  | ECE101      |              4 | Undergraduate |         100 |
+-----------+-------------+-------------+----------------+---------------+-------------+
2 rows in set (0.00 sec)
```

6.3 Inserting into student

We then inserted a student record linked to course and department ids that already exist

INSERT INTO STUDENT

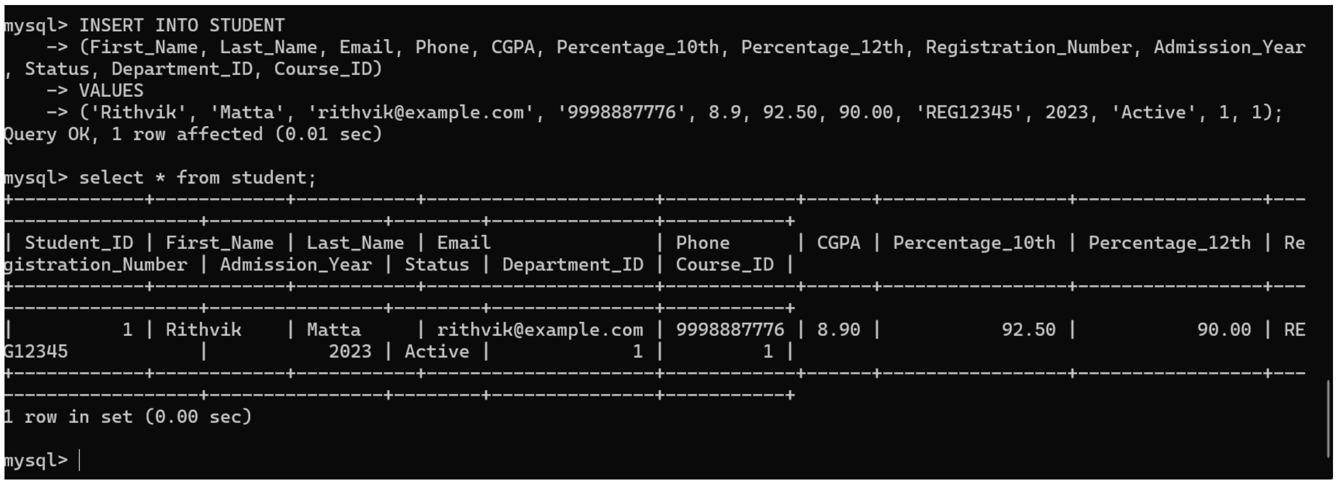
(First_Name, Last_Name, Email, Phone, CGPA, Percentage_10th, Percentage_12th,

Registration_Number, Admission_Year, Status, Department_ID, Course_ID)

VALUES

('Rithvik', 'Matta', 'rithvik@example.com', '9998887776', 8.9, 92.50, 90.00, 'REG12345', 2023, 'Active',

1, 1);



```
mysql> INSERT INTO STUDENT
    -> (First_Name, Last_Name, Email, Phone, CGPA, Percentage_10th, Percentage_12th, Registration_Number, Admission_Year
, Status, Department_ID, Course_ID)
    -> VALUES
    -> ('Rithvik', 'Matta', 'rithvik@example.com', '9998887776', 8.9, 92.50, 90.00, 'REG12345', 2023, 'Active', 1, 1);
Query OK, 1 row affected (0.01 sec)

mysql> select * from student;
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+---
------------------+----------------+--------+---------------+-----------+
| Student_ID | First_Name | Last_Name | Email               | Phone      | CGPA | Percentage_10th | Percentage_12th | Re
gistration_Number | Admission_Year | Status | Department_ID | Course_ID |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+---
------------------+----------------+--------+---------------+-----------+
|          1 | Rithvik    | Matta     | rithvik@example.com | 9998887776 | 8.90 |           92.50 |           90.00 | RE
G12345            |           2023 | Active |             1 |         1 |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+---
------------------+----------------+--------+---------------+-----------+
1 row in set (0.00 sec)

mysql> |
```

6.4 Inserting into company

Next we inserted a company that visits for placement

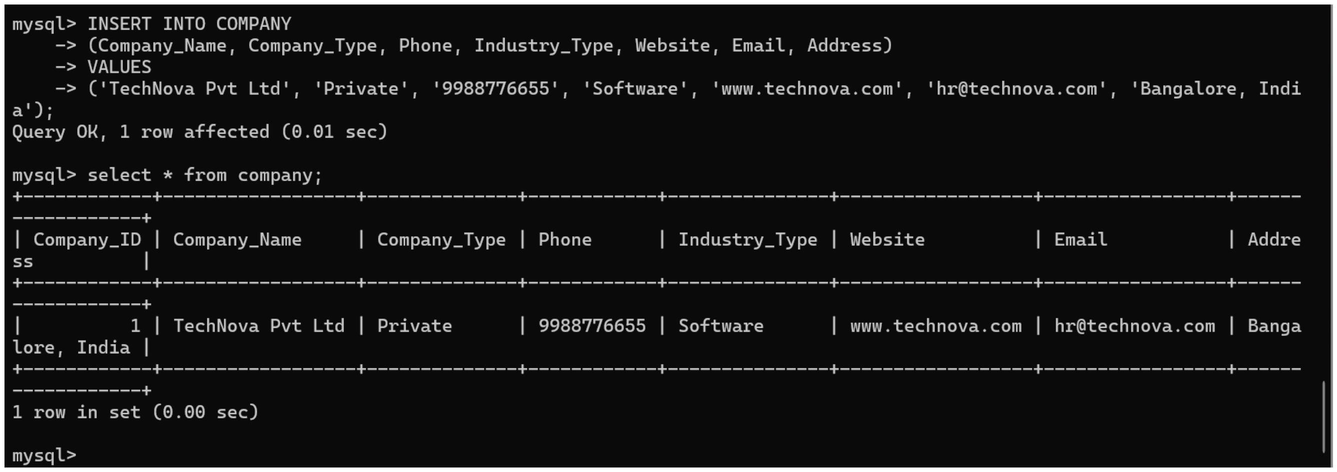
INSERT INTO COMPANY

(Company_Name, Company_Type, Phone, Industry_Type, Website, Email, Address)

VALUES

('TechNova Pvt Ltd', 'Private', '9988776655', 'Software', 'www.technova.com', 'hr@technova.com',

'Bangalore, India');



```
mysql> INSERT INTO COMPANY
    -> (Company_Name, Company_Type, Phone, Industry_Type, Website, Email, Address)
    -> VALUES
    -> ('TechNova Pvt Ltd', 'Private', '9988776655', 'Software', 'www.technova.com', 'hr@technova.com', 'Bangalore, Indi
a');
Query OK, 1 row affected (0.01 sec)

mysql> select * from company;
+------------+------------------+--------------+------------+---------------+------------------+-----------------+------
------------+
| Company_ID | Company_Name     | Company_Type | Phone      | Industry_Type | Website          | Email           | Addre
ss           |
+------------+------------------+--------------+------------+---------------+------------------+-----------------+------
------------+
|          1 | TechNova Pvt Ltd | Private      | 9988776655 | Software      | www.technova.com | hr@technova.com | Banga
lore, India |
+------------+------------------+--------------+------------+---------------+------------------+-----------------+------
------------+
1 row in set (0.00 sec)

mysql>
```

6.5 Inserting into job posting

We then added a job offered by that company

INSERT INTO JOB_POSTING

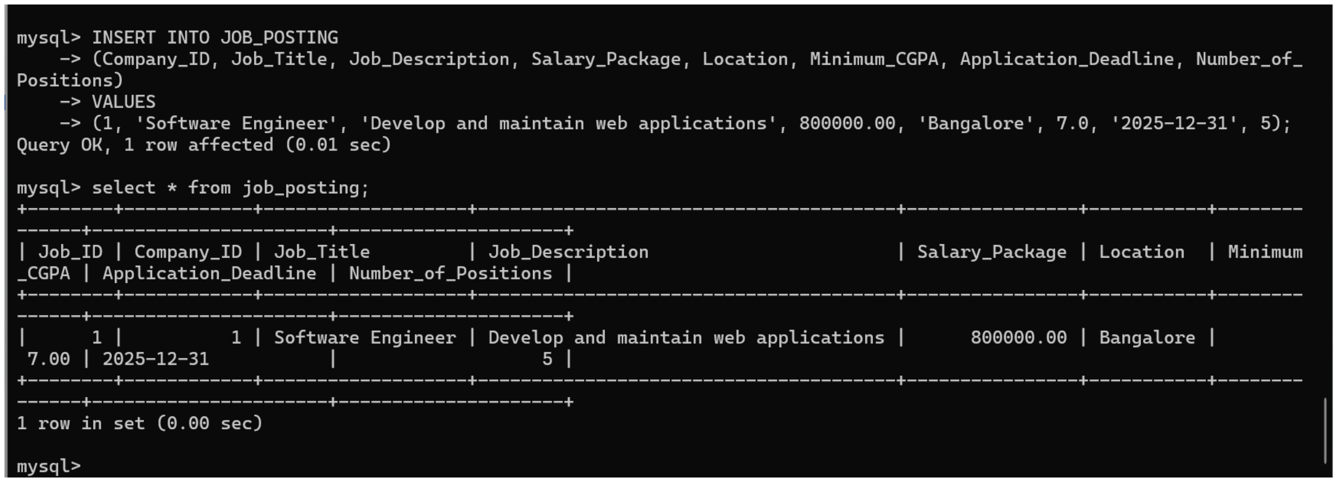
(Company_ID, Job_Title, Job_Description, Salary_Package, Location, Minimum_CGPA,

Application_Deadline, Number_of_Positions)

VALUES

(1, 'Software Engineer', 'Develop and maintain web applications', 800000.00, 'Bangalore', 7.0,

'2025-12-31', 5);



```
mysql> INSERT INTO JOB_POSTING
    -> (Company_ID, Job_Title, Job_Description, Salary_Package, Location, Minimum_CGPA, Application_Deadline, Number_of_
Positions)
    -> VALUES
    -> (1, 'Software Engineer', 'Develop and maintain web applications', 800000.00, 'Bangalore', 7.0, '2025-12-31', 5);
Query OK, 1 row affected (0.01 sec)

mysql> select * from job_posting;
+--------+------------+-------------------+---------------------------------------+----------------+-----------+--------
------+----------------------+---------------------+
| Job_ID | Company_ID | Job_Title         | Job_Description                       | Salary_Package | Location  | Minimum
_CGPA | Application_Deadline | Number_of_Positions |
+--------+------------+-------------------+---------------------------------------+----------------+-----------+--------
------+----------------------+---------------------+
|      1 |          1 | Software Engineer | Develop and maintain web applications |      800000.00 | Bangalore |
 7.00 | 2025-12-31           |                   5 |
+--------+------------+-------------------+---------------------------------------+----------------+-----------+--------
------+----------------------+---------------------+
1 row in set (0.00 sec)

mysql>
```

6.6 Inserting into placement officer

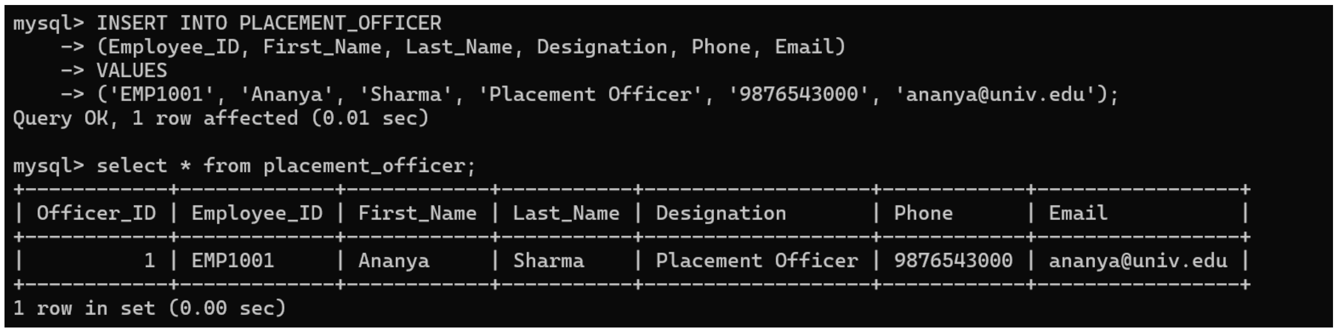
We also inserted one placement officer record for management

INSERT INTO PLACEMENT_OFFICER

(Employee_ID, First_Name, Last_Name, Designation, Phone, Email)

VALUES

('EMP1001', 'Ananya', 'Sharma', 'Placement Officer', '9876543000', 'ananya@univ.edu');



```
mysql> INSERT INTO PLACEMENT_OFFICER
    -> (Employee_ID, First_Name, Last_Name, Designation, Phone, Email)
    -> VALUES
    -> ('EMP1001', 'Ananya', 'Sharma', 'Placement Officer', '9876543000', 'ananya@univ.edu');
Query OK, 1 row affected (0.01 sec)

mysql> select * from placement_officer;
+------------+-------------+------------+-----------+-------------------+------------+-----------------+
| Officer_ID | Employee_ID | First_Name | Last_Name | Designation       | Phone      | Email           |
+------------+-------------+------------+-----------+-------------------+------------+-----------------+
|          1 | EMP1001     | Ananya     | Sharma    | Placement Officer | 9876543000 | ananya@univ.edu |
+------------+-------------+------------+-----------+-------------------+------------+-----------------+
1 row in set (0.00 sec)
```

6.7 Inserting into application

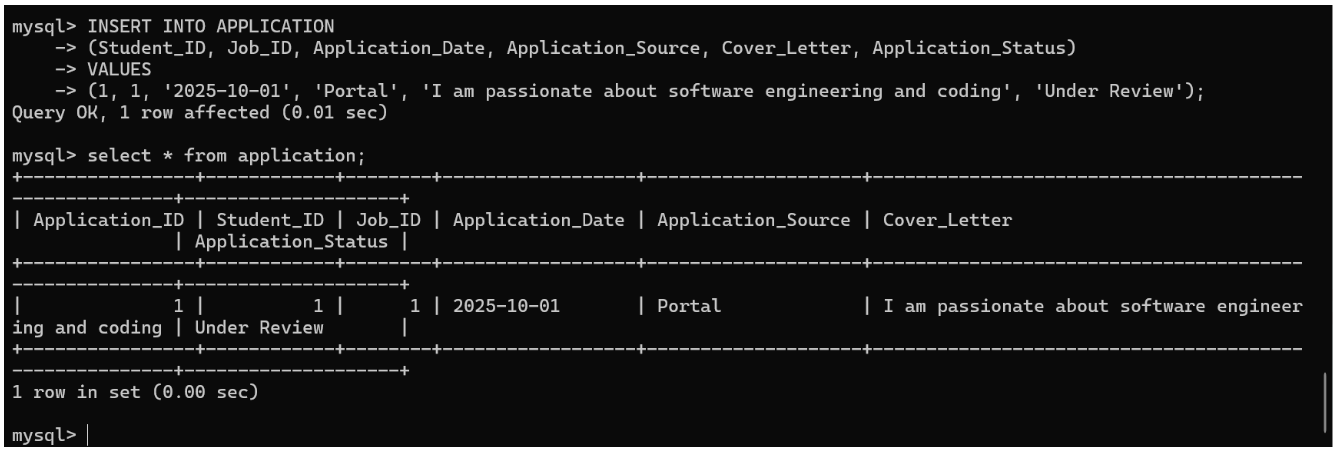
This table links a student and a job so we inserted one example application

INSERT INTO APPLICATION

(Student_ID, Job_ID, Application_Date, Application_Source, Cover_Letter, Application_Status)

VALUES

(1, 1, '2025-10-01', 'Portal', 'I am passionate about software engineering and coding', 'Under Review');



```
mysql> INSERT INTO APPLICATION
    -> (Student_ID, Job_ID, Application_Date, Application_Source, Cover_Letter, Application_Status)
    -> VALUES
    -> (1, 1, '2025-10-01', 'Portal', 'I am passionate about software engineering and coding', 'Under Review');
Query OK, 1 row affected (0.01 sec)

mysql> select * from application;
+----------------+------------+--------+------------------+--------------------+------------------------------------------
---------------+--------------------+
| Application_ID | Student_ID | Job_ID | Application_Date | Application_Source | Cover_Letter
               | Application_Status |
+----------------+------------+--------+------------------+--------------------+------------------------------------------
---------------+--------------------+
|              1 |          1 |      1 | 2025-10-01       | Portal             | I am passionate about software engineer
ing and coding | Under Review       |
+----------------+------------+--------+------------------+--------------------+------------------------------------------
---------------+--------------------+
1 row in set (0.00 sec)

mysql> |
```

6.8 inserting into interview

Finally we added one interview record related to the application we created above

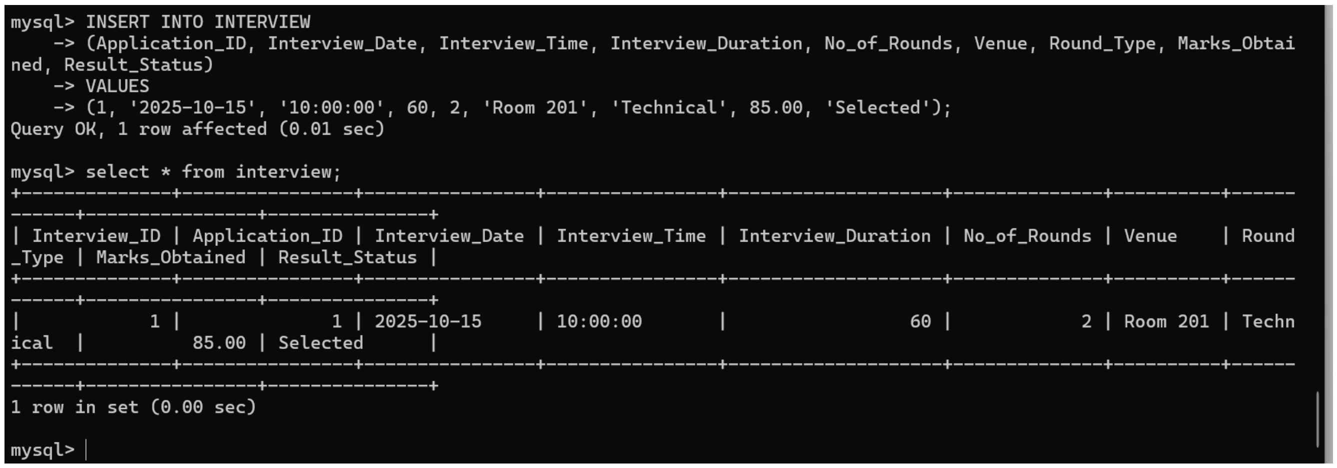
INSERT INTO INTERVIEW

(Application_ID, Interview_Date, Interview_Time, Interview_Duration, No_of_Rounds, Venue,

Round_Type, Marks_Obtained, Result_Status)

VALUES

(1, '2025-10-15', '10:00:00', 60, 2, 'Room 201', 'Technical', 85.00, 'Selected');



```
mysql> INSERT INTO INTERVIEW
    -> (Application_ID, Interview_Date, Interview_Time, Interview_Duration, No_of_Rounds, Venue, Round_Type, Marks_Obtai
ned, Result_Status)
    -> VALUES
    -> (1, '2025-10-15', '10:00:00', 60, 2, 'Room 201', 'Technical', 85.00, 'Selected');
Query OK, 1 row affected (0.01 sec)

mysql> select * from interview;
+--------------+----------------+----------------+----------------+--------------------+--------------+----------+------
------+----------------+---------------+
| Interview_ID | Application_ID | Interview_Date | Interview_Time | Interview_Duration | No_of_Rounds | Venue    | Round
_Type | Marks_Obtained | Result_Status |
+--------------+----------------+----------------+----------------+--------------------+--------------+----------+------
------+----------------+---------------+
|            1 |              1 | 2025-10-15     | 10:00:00       |                 60 |            2 | Room 201 | Techn
ical  |          85.00 | Selected      |
+--------------+----------------+----------------+----------------+--------------------+--------------+----------+------
------+----------------+---------------+
1 row in set (0.00 sec)

mysql> 
```

After inserting all these rows we checked every table using a simple select query to confirm that all foreign key relationships were working perfectly and that each record was connected correctly, we also verified that no constraint errors occurred and all data types matched the values we inserted.

These DML insertions completed our database setup successfully and now the data was ready to be used by our streamlit application for testing job listings applications and placements in real time.

7. QUERIES

Once our tables were filled with data the next step was to write and run queries to test if everything worked properly and to see if the data relationships were correct, this part was actually fun because it helped us check if our design really made sense in real use cases.
We tried different types of sql queries like select with group by and aggregate functions updating some records deleting some data nested queries and even correlated queries, these queries helped us not just verify the working of our tables but also get some real insights from our sample data

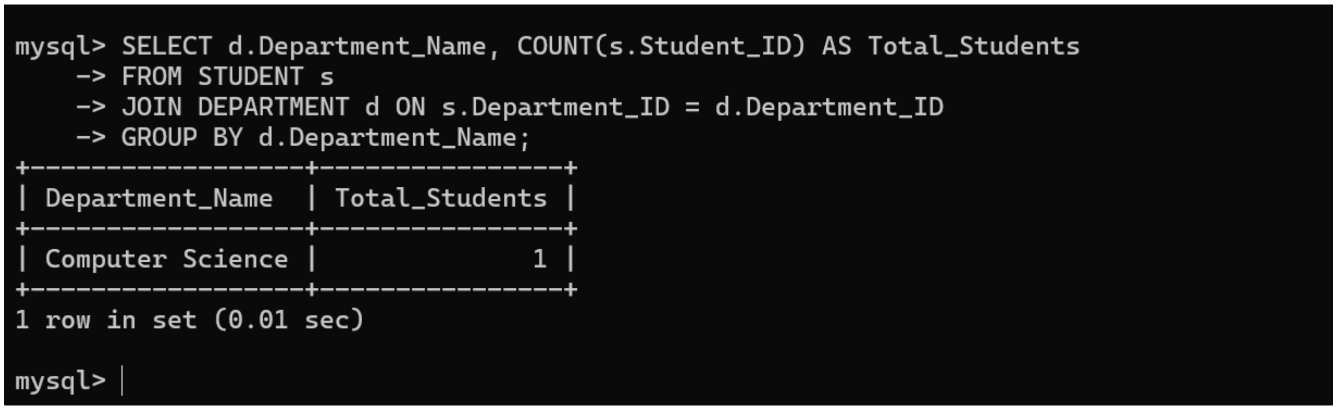
We wrote and executed all the queries in mysql workbench and took screenshots after each successful run to include in the report, below we have explained every query type that we used and mentioned where to add screenshots for each one

7.1 SIMPLE QUERY WITH GROUP BY, AGGREGATE

First we started with some simple select queries that use aggregate functions like count avg and max along with group by, we wanted to check if our data can be grouped by department or company or even placement result to get quick stats

For example we wrote this query to find how many students are there in each department

```
SELECT d.Department_Name, COUNT(s.Student_ID) AS Total_Students
FROM STUDENT s
JOIN DEPARTMENT d ON s.Department_ID = d.Department_ID
GROUP BY d.Department_Name;
```



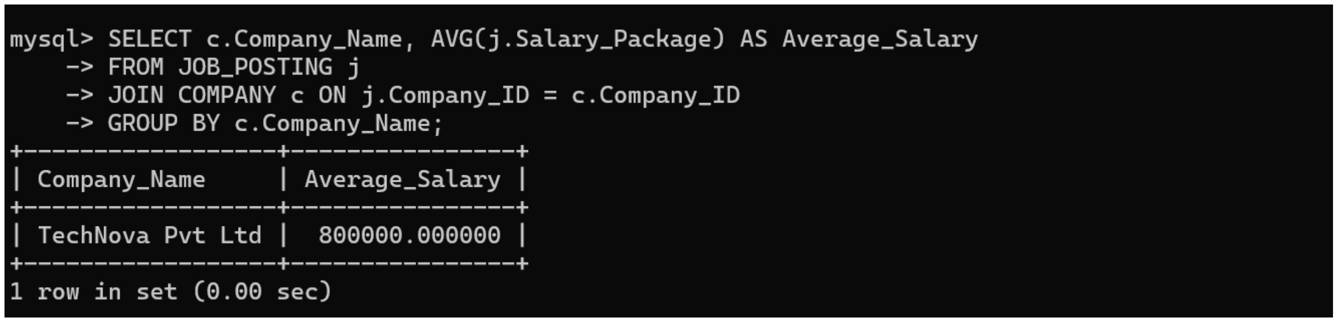
```
mysql> SELECT d.Department_Name, COUNT(s.Student_ID) AS Total_Students
    -> FROM STUDENT s
    -> JOIN DEPARTMENT d ON s.Department_ID = d.Department_ID
    -> GROUP BY d.Department_Name;
+------------------+----------------+
| Department_Name  | Total_Students |
+------------------+----------------+
| Computer Science |              1 |
+------------------+----------------+
1 row in set (0.01 sec)

mysql>
```

This query joins the student and department tables then counts the number of students per department it helped us confirm that our foreign key links between student and department were working perfectly

We also tried another one to find the average salary offered by each company

```
SELECT c.Company_Name, AVG(j.Salary_Package) AS Average_Salary
FROM JOB_POSTING j
JOIN COMPANY c ON j.Company_ID = c.Company_ID
GROUP BY c.Company_Name;
```

This one was great to test both grouping and numeric columns like salary package the results showed us which company offered the highest average package among our sample data



```
mysql> SELECT c.Company_Name, AVG(j.Salary_Package) AS Average_Salary
    -> FROM JOB_POSTING j
    -> JOIN COMPANY c ON j.Company_ID = c.Company_ID
    -> GROUP BY c.Company_Name;
+-------------------+----------------+
| Company_Name      | Average_Salary |
+-------------------+----------------+
| TechNova Pvt Ltd  |  800000.000000 |
+-------------------+----------------+
1 row in set (0.00 sec)
```

We also learned that group by queries are really useful for placement analytics because they help placement officers see quick summaries without writing long code

7.2 UPDATE OPERATION

Then we tested updating data because in a real system things change all the time students might get shortlisted application statuses may change or company job details may be updated

We started with this simple query to update a student’s status when they get placed

```
UPDATE STUDENT
SET Status = 'Placed'
WHERE Student_ID = 1;
```

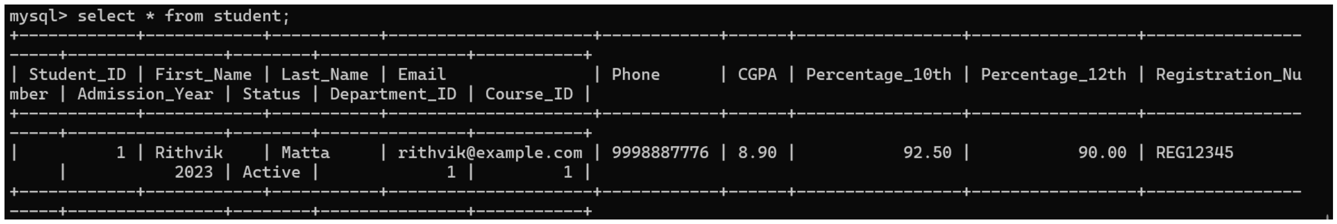
This query changes the status of a specific student and we checked that it updated correctly using a select statement afterwards

we also tried updating the application table to mark an application as shortlisted

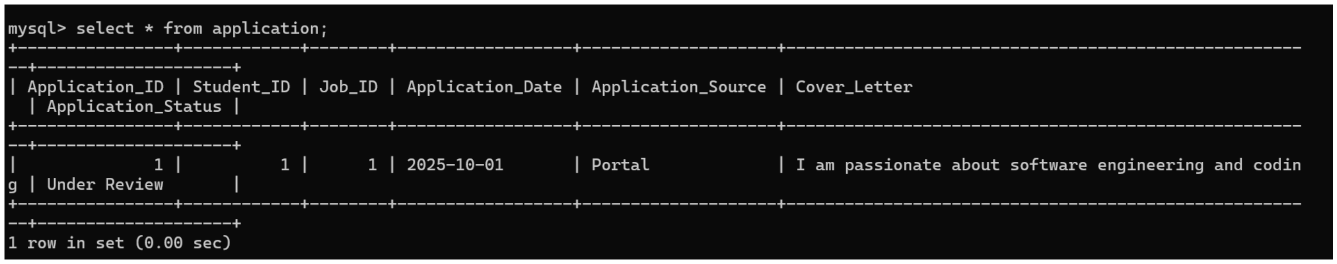
```
UPDATE APPLICATION
SET Application_Status = 'Shortlisted'
WHERE Application_ID = 1;
```

after running this we saw that the application status changed from under review to shortlisted

Before:

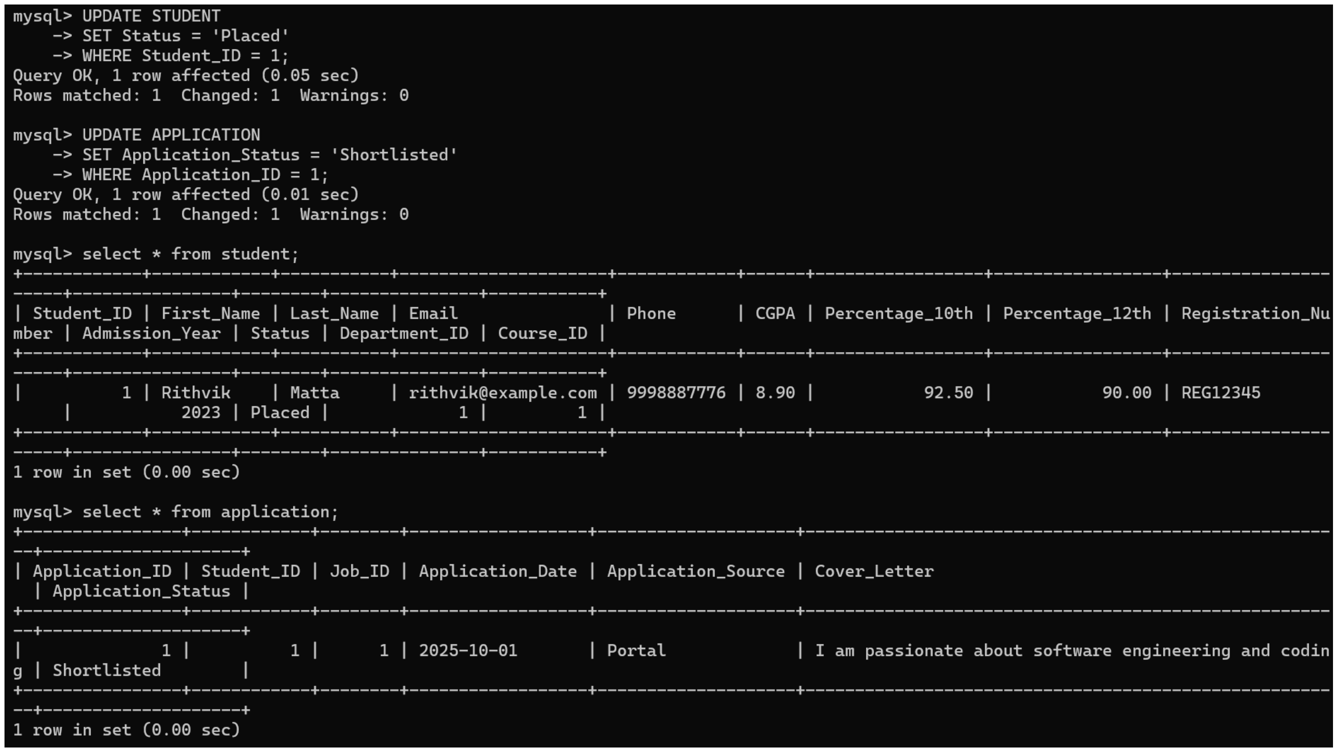


```
mysql> select * from student;
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
| Student_ID | First_Name | Last_Name | Email               | Phone      | CGPA | Percentage_10th | Percentage_12th | Registration_Nu
mber | Admission_Year | Status | Department_ID | Course_ID |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
|          1 | Rithvik    | Matta     | rithvik@example.com | 9998887776 | 8.90 |           92.50 |           90.00 | REG12345
     |           2023 | Active |             1 |         1 |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
```



```
mysql> select * from application;
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
| Application_ID | Student_ID | Job_ID | Application_Date | Application_Source | Cover_Letter
  | Application_Status |
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
|              1 |          1 |      1 | 2025-10-01       | Portal             | I am passionate about software engineering and codin
g | Under Review       |
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
1 row in set (0.00 sec)
```

After:



```
mysql> UPDATE STUDENT
    -> SET Status = 'Placed'
    -> WHERE Student_ID = 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE APPLICATION
    -> SET Application_Status = 'Shortlisted'
    -> WHERE Application_ID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student;
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
| Student_ID | First_Name | Last_Name | Email               | Phone      | CGPA | Percentage_10th | Percentage_12th | Registration_Nu
mber | Admission_Year | Status | Department_ID | Course_ID |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
|          1 | Rithvik    | Matta     | rithvik@example.com | 9998887776 | 8.90 |           92.50 |           90.00 | REG12345
     |           2023 | Placed |             1 |         1 |
+------------+------------+-----------+---------------------+------------+------+-----------------+-----------------+----------------
-----+----------------+--------+---------------+-----------+
1 row in set (0.00 sec)

mysql> select * from application;
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
| Application_ID | Student_ID | Job_ID | Application_Date | Application_Source | Cover_Letter
  | Application_Status |
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
|              1 |          1 |      1 | 2025-10-01       | Portal             | I am passionate about software engineering and codin
g | Shortlisted        |
+----------------+------------+--------+------------------+--------------------+-----------------------------------------------------
--+--------------------+
1 row in set (0.00 sec)
```

these small updates helped us verify that the database accepts changes easily and that no foreign key or constraint stopped us from modifying existing records

7.3 DELETE OPERATION

after testing updates we moved to delete operations to see what happens when we remove data and how foreign keys behave

we started by deleting a job posting that we had added for testing

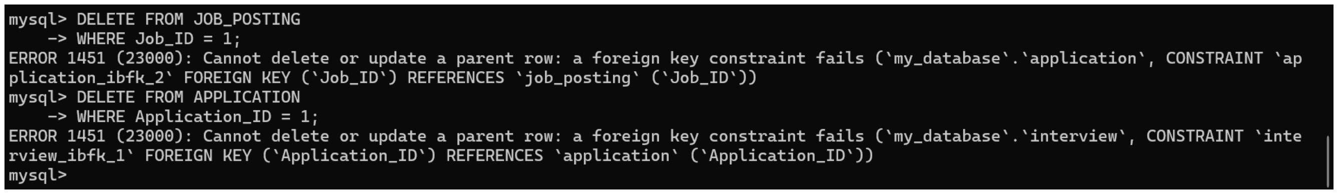
```
DELETE FROM JOB_POSTING
WHERE Job_ID = 1;
```

mysql showed an error at first because we had linked data in the application table which referenced this job id, this taught us that foreign key constraints were working correctly since it didn't allow deleting a job that still has application, So we deleted the dependent record first

```
DELETE FROM APPLICATION
WHERE Application_ID = 1;
```

and then retried the delete on job posting which worked perfectly

this was a good way to test referential integrity in our database



```
mysql> DELETE FROM JOB_POSTING
    -> WHERE Job_ID = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`my_database`.`application`, CONSTRAINT `ap
plication_ibfk_2` FOREIGN KEY (`Job_ID`) REFERENCES `job_posting` (`Job_ID`))
mysql> DELETE FROM APPLICATION
    -> WHERE Application_ID = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`my_database`.`interview`, CONSTRAINT `inte
rview_ibfk_1` FOREIGN KEY (`Application_ID`) REFERENCES `application` (`Application_ID`))
mysql>
```

These delete operations proved that our database prevents accidental data loss because of foreign key dependencies which is exactly what we wanted

7.4 CORRELATED QUERY

Next we worked on correlated queries which are slightly advanced but really useful, a correlated query is one where the inner query depends on the outer query for its values we wanted to find students whose CGPA is higher than the average CGPA of their department

```
SELECT s.First_Name, s.Last_Name, s.CGPA, d.Department_Name
FROM STUDENT s
JOIN DEPARTMENT d ON s.Department_ID = d.Department_ID
WHERE s.CGPA > (
    SELECT AVG(s2.CGPA)
    FROM STUDENT s2
    WHERE s2.Department_ID = s.Department_ID
);
```

this query runs the inner select for each department and compares CGPA values of students within that department, it helped us see if the CGPA data and department connection were consistent

[Insert Screenshot 7.4 – Correlated query result]
what to include screenshot showing output listing students whose CGPA is higher than their departmental average, this type of query can help placement officers identify top performing students in each department easily and we found it quite practical for analytics

7.5 NESTED QUERY

Finally we tried a nested query which means a query inside another but without direct dependency like in correlated ones, we used it to find details of students who applied to jobs offering more than the average salary

```
SELECT s.First_Name, s.Last_Name, j.Job_Title, j.Salary_Package
FROM STUDENT s
JOIN APPLICATION a ON s.Student_ID = a.Student_ID
JOIN JOB_POSTING j ON a.Job_ID = j.Job_ID
WHERE j.Salary_Package > (
    SELECT AVG(Salary_Package)
    FROM JOB_POSTING
);
```

this query first finds the average salary from all job postings in the inner query, then it compares each job’s salary to that value and selects only those above average along with the student details

This was one of our favourite queries because it felt realistic for placement analysis, it can help the placement cell identify which students applied for high paying jobs and track if they got selected later

After all these queries we got a lot of confidence in our database design because everything behaved exactly how we expected, group by queries gave nice summary results updates and deletes respected relationships and correlated plus nested queries showed that our joins and references were correctly set these tests also showed how our database can support not just data storage but also powerful analytics that placement officers can later use inside the streamlit dashboard

In conclusion running all these queries gave us proof that our system is strong flexible and ready for real use, we can now confidently connect this database to our streamlit interface and perform all these operations directly from the website in a safe and smooth way

8. STORED PROCEDURES, FUNCTIONS AND TRIGGERS

In this part of our project, we implemented stored procedures, functions, and triggers in MySQL to make our database more dynamic and intelligent. These features allowed us to automate common operations, reduce repetitive queries, and ensure data consistency. Writing logic directly inside the database improved the overall performance and reliability of our placement management system. It also made the system feel more professional because certain actions, such as checking eligibility or updating application status, now happen automatically without manual effort.

8.1 STORED PROCEDURES OR FUNCTIONS

We first created stored procedures and functions to simplify our commonly used operations. The main goal was to handle repetitive tasks like checking student eligibility or fetching placed students through a single command instead of writing the same query repeatedly.

A. Function To Check Job Eligibility

We created a simple function called Check_Eligibility that compares a student's CGPA with the minimum CGPA required for a job. It returns either "Eligible" or "Not Eligible."

```
DELIMITER $$

CREATE FUNCTION Check_Eligibility(student_cgpa DECIMAL(3,2), min_cgpa DECIMAL(3,2))
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE result VARCHAR(20);
    IF student_cgpa >= min_cgpa THEN
        SET result = 'Eligible';
    ELSE
        SET result = 'Not Eligible';
    END IF;
    RETURN result;
END$$
```

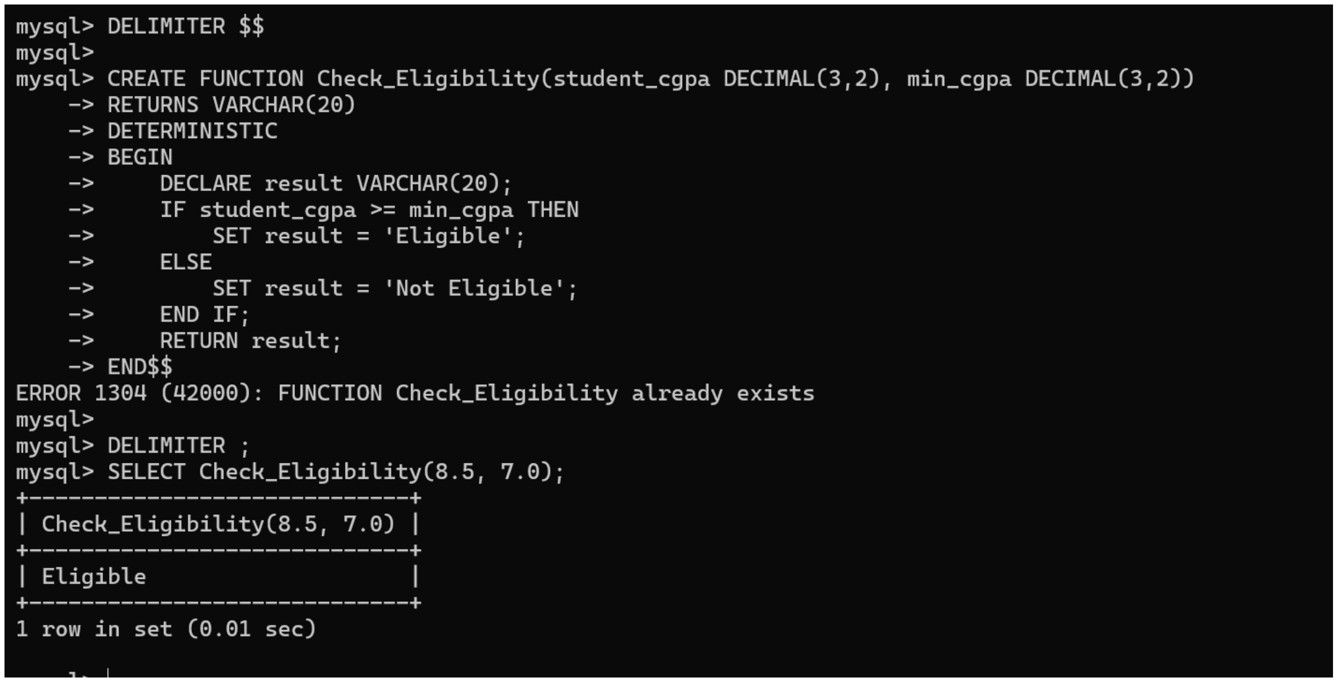
```
DELIMITER ;
```

This function makes it easy to check eligibility without manually comparing CGPAs each time. Similar approaches to using stored functions for eligibility automation were discussed by Harsha and Karthik in their paper on efficient database-driven eligibility systems [6].

To test it, we executed:

```
SELECT Check_Eligibility(8.5, 7.0);
```

It successfully returned “Eligible,” confirming that our function worked as expected.



```
mysql> DELIMITER $$
mysql>
mysql> CREATE FUNCTION Check_Eligibility(student_cgpa DECIMAL(3,2), min_cgpa DECIMAL(3,2))
    -> RETURNS VARCHAR(20)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE result VARCHAR(20);
    ->     IF student_cgpa >= min_cgpa THEN
    ->         SET result = 'Eligible';
    ->     ELSE
    ->         SET result = 'Not Eligible';
    ->     END IF;
    ->     RETURN result;
    -> END$$
ERROR 1304 (42000): FUNCTION Check_Eligibility already exists
mysql>
mysql> DELIMITER ;
mysql> SELECT Check_Eligibility(8.5, 7.0);
+-----------------------------+
| Check_Eligibility(8.5, 7.0) |
+-----------------------------+
| Eligible                    |
+-----------------------------+
1 row in set (0.01 sec)
```

B. Procedure To Get Placed Students By Company

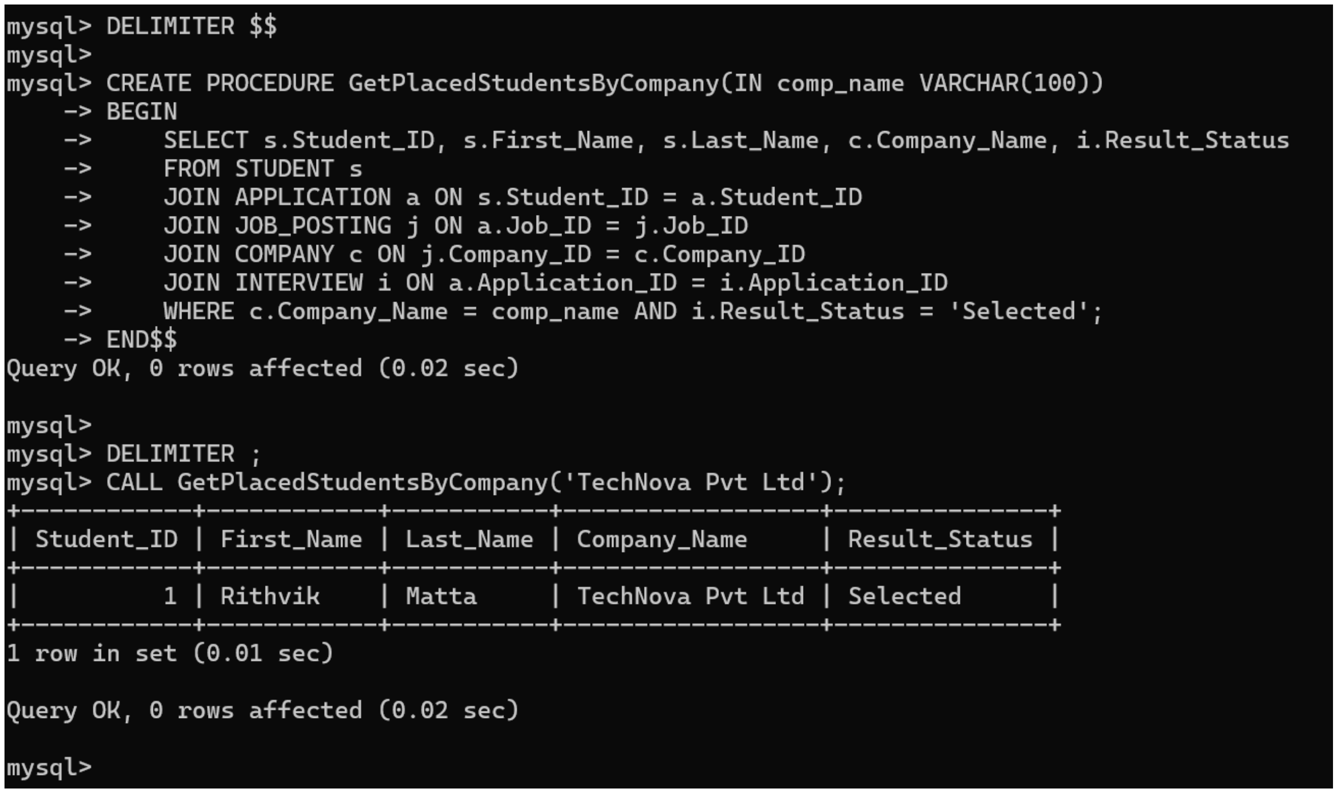
We created another stored procedure called GetPlacedStudentsByCompany. This procedure retrieves a list of students who were selected by a particular company during placements.

```
DELIMITER $$

CREATE PROCEDURE GetPlacedStudentsByCompany(IN comp_name VARCHAR(100))
BEGIN
    SELECT s.Student_ID, s.First_Name, s.Last_Name, c.Company_Name, i.Result_Status
    FROM STUDENT s
    JOIN APPLICATION a ON s.Student_ID = a.Student_ID
    JOIN JOB_POSTING j ON a.Job_ID = j.Job_ID
    JOIN COMPANY c ON j.Company_ID = c.Company_ID
    JOIN INTERVIEW i ON a.Application_ID = i.Application_ID
    WHERE c.Company_Name = comp_name AND i.Result_Status = 'Selected';
END$$

DELIMITER ;
```

When we executed the command:

```
CALL GetPlacedStudentsByCompany('TechNova Pvt Ltd');
```

it displayed all students placed in that company. This procedure simplified the process of generating placement reports and saved time for the placement officer.



```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE GetPlacedStudentsByCompany(IN comp_name VARCHAR(100))
    -> BEGIN
    ->     SELECT s.Student_ID, s.First_Name, s.Last_Name, c.Company_Name, i.Result_Status
    ->     FROM STUDENT s
    ->     JOIN APPLICATION a ON s.Student_ID = a.Student_ID
    ->     JOIN JOB_POSTING j ON a.Job_ID = j.Job_ID
    ->     JOIN COMPANY c ON j.Company_ID = c.Company_ID
    ->     JOIN INTERVIEW i ON a.Application_ID = i.Application_ID
    ->     WHERE c.Company_Name = comp_name AND i.Result_Status = 'Selected';
    -> END$$
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL GetPlacedStudentsByCompany('TechNova Pvt Ltd');
+------------+------------+-----------+------------------+---------------+
| Student_ID | First_Name | Last_Name | Company_Name     | Result_Status |
+------------+------------+-----------+------------------+---------------+
|          1 | Rithvik    | Matta     | TechNova Pvt Ltd | Selected      |
+------------+------------+-----------+------------------+---------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

mysql>
```

The use of stored procedures for managing placement reporting has also been supported in prior research on database automation for recruitment systems by Shrivastava et al. [7].

C. Function To Get Placement Status Of A Student

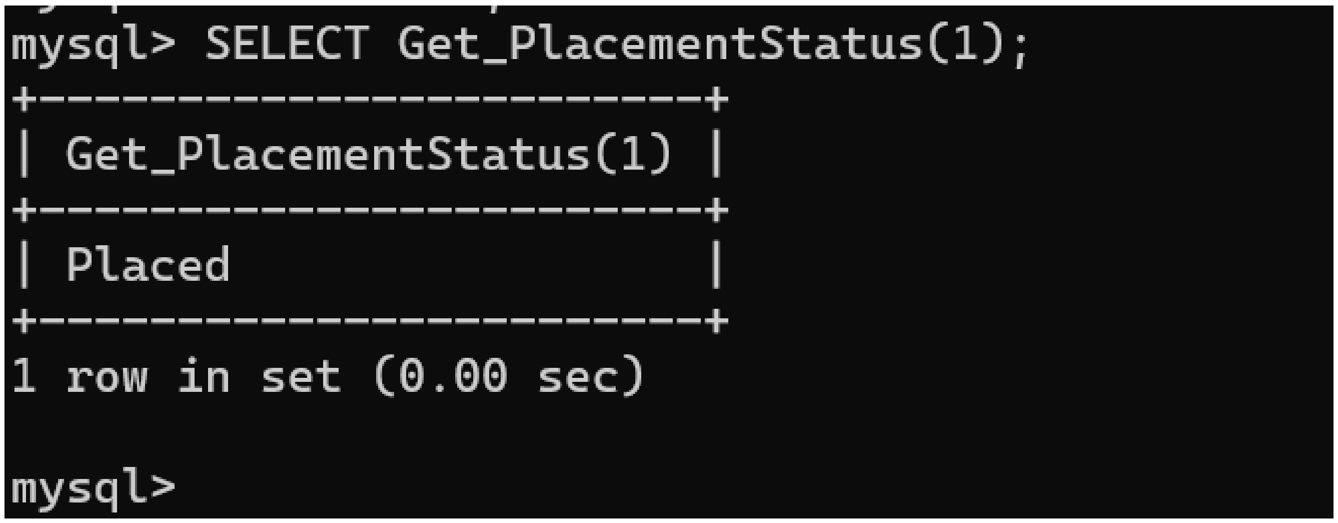
We also created a function called Get_PlacementStatus, which determines whether a student is placed or not.

```
DELIMITER $$

CREATE FUNCTION Get_PlacementStatus(studentId INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE placement_status VARCHAR(20);

    IF EXISTS (
        SELECT 1
        FROM INTERVIEW i
        JOIN APPLICATION a ON i.Application_ID = a.Application_ID
        WHERE a.Student_ID = studentId
        AND i.Result_Status = 'Selected'
    ) THEN
        SET placement_status = 'Placed';
    ELSE
        SET placement_status = 'Not Placed';
    END IF;

    RETURN placement_status;
END$$

DELIMITER ;
```

We tested the function using:

```
SELECT Get_PlacementStatus(1);
```

It returned "Placed," indicating that the student had a successful interview record.



```
mysql> SELECT Get_PlacementStatus(1);
+------------------------+
| Get_PlacementStatus(1) |
+------------------------+
| Placed                 |
+------------------------+
1 row in set (0.00 sec)

mysql>
```

This function was later used in our Streamlit reports page to dynamically show each student's placement status.

8.2 TRIGGERS

Next, we created a few triggers to make our database respond automatically to specific events. A trigger is a piece of code that runs automatically when an insert, update, or delete operation occurs. Triggers helped us reduce manual errors and made our workflow smoother.

A. Trigger To Set Default Application Status

We created a trigger called **before_application_insert** to automatically set the default application status to "Under Review" if no status is provided during insertion.

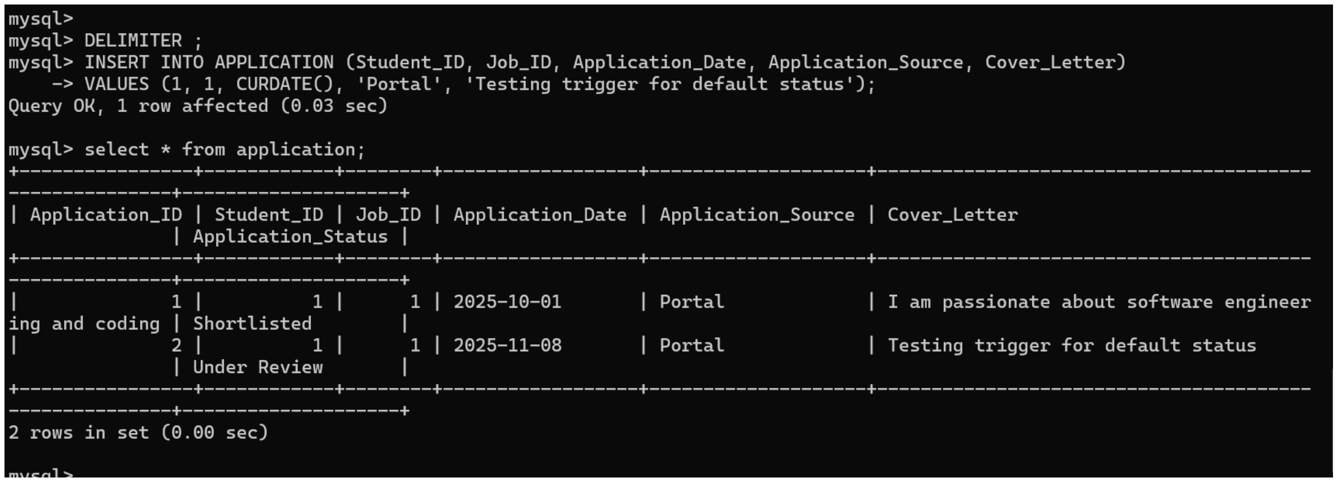
```
DELIMITER $$


CREATE TRIGGER before_application_insert
BEFORE INSERT ON APPLICATION
```

```
FOR EACH ROW
BEGIN
    IF NEW.Application_Status IS NULL THEN
        SET NEW.Application_Status = 'Under Review';
    END IF;
END$$

DELIMITER ;
```

We tested it by inserting a new application record without specifying the status:

```
INSERT INTO APPLICATION (Student_ID, Job_ID, Application_Date,
Application_Source, Cover_Letter)
VALUES (1, 1, CURDATE(), 'Portal', 'Testing trigger for default
status');
```

After viewing the data, the status column automatically showed “Under Review,” confirming that the trigger was functioning correctly.



```
mysql>
mysql> DELIMITER ;
mysql> INSERT INTO APPLICATION (Student_ID, Job_ID, Application_Date, Application_Source, Cover_Letter)
    -> VALUES (1, 1, CURDATE(), 'Portal', 'Testing trigger for default status');
Query OK, 1 row affected (0.03 sec)

mysql> select * from application;
+----------------+------------+--------+------------------+--------------------+----------------------------------------
---------------+--------------------+
| Application_ID | Student_ID | Job_ID | Application_Date | Application_Source | Cover_Letter
               | Application_Status |
+----------------+------------+--------+------------------+--------------------+----------------------------------------
---------------+--------------------+
|              1 |          1 |      1 | 2025-10-01       | Portal             | I am passionate about software engineer
ing and coding | Shortlisted        |
|              2 |          1 |      1 | 2025-11-08       | Portal             | Testing trigger for default status
               | Under Review       |
+----------------+------------+--------+------------------+--------------------+----------------------------------------
---------------+--------------------+
2 rows in set (0.00 sec)

mysql>
```

Similar use of automated triggers to ensure data consistency was discussed in academic management system automation studies by Suneetha and Ramesh [8].

B. Trigger To Log Student Placement Changes

We also implemented a trigger that logs every change in a student's placement status from "Active" to "Placed." This helps in maintaining a record of updates for auditing purposes.

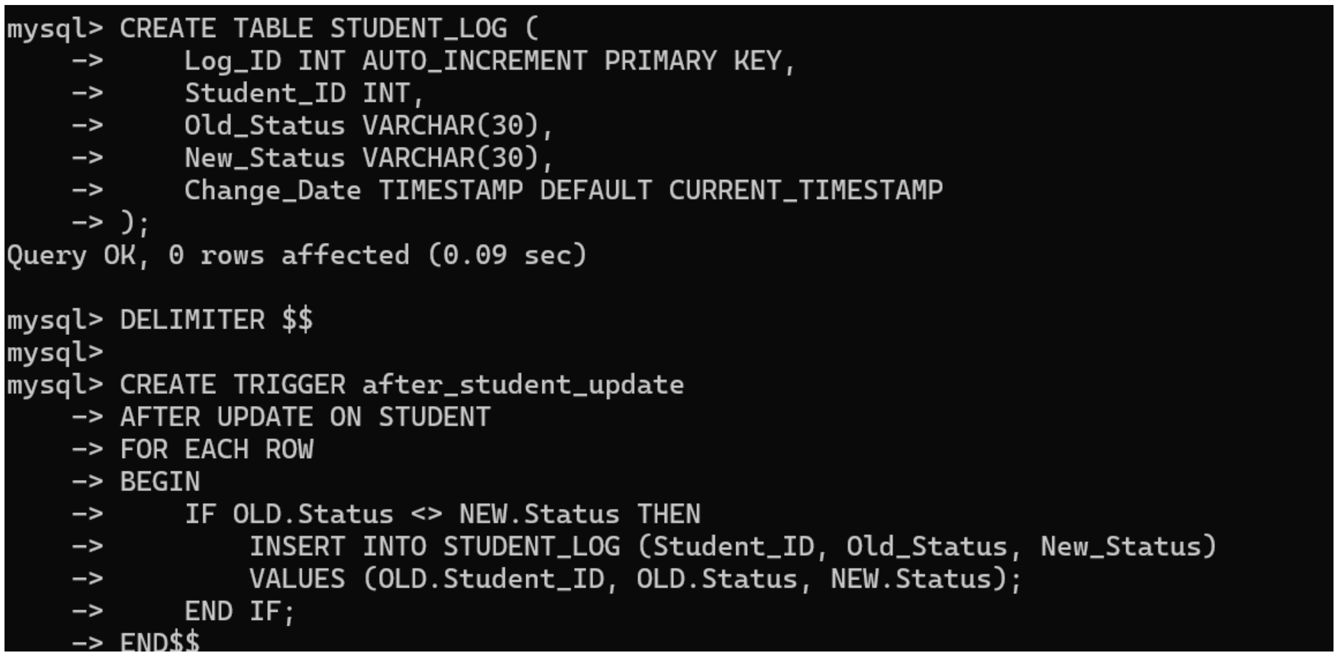
First, we created a log table:

```
CREATE TABLE STUDENT_LOG (
    Log_ID INT AUTO_INCREMENT PRIMARY KEY,
    Student_ID INT,
    Old_Status VARCHAR(30),
    New_Status VARCHAR(30),
    Change_Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Then, we created the trigger:

```
DELIMITER $$

CREATE TRIGGER after_student_update
AFTER UPDATE ON STUDENT
FOR EACH ROW
BEGIN
    IF OLD.Status <> NEW.Status THEN
        INSERT INTO STUDENT_LOG (Student_ID, Old_Status, New_Status)
        VALUES (OLD.Student_ID, OLD.Status, NEW.Status);
    END IF;
END$$

DELIMITER ;
```

When we updated a student's status from "Active" to "Placed," the log table automatically recorded the change.



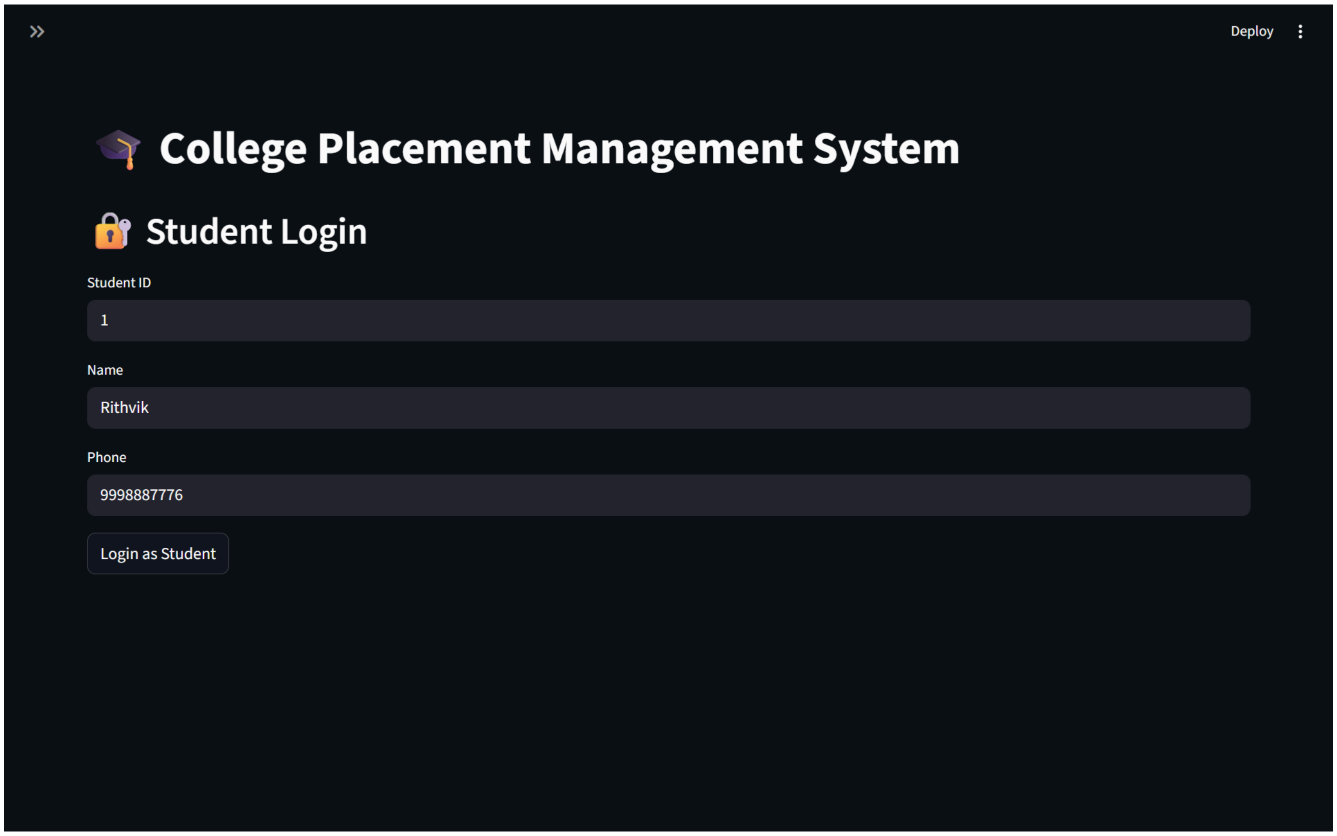
```
mysql> CREATE TABLE STUDENT_LOG (
    ->     Log_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Student_ID INT,
    ->     Old_Status VARCHAR(30),
    ->     New_Status VARCHAR(30),
    ->     Change_Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER after_student_update
    -> AFTER UPDATE ON STUDENT
    -> FOR EACH ROW
    -> BEGIN
    ->     IF OLD.Status <> NEW.Status THEN
    ->         INSERT INTO STUDENT_LOG (Student_ID, Old_Status, New_Status)
    ->         VALUES (OLD.Student_ID, OLD.Status, NEW.Status);
    ->     END IF;
    -> END$$
```

This type of trigger ensures accountability and transparency, which are important for maintaining data reliability in institutional systems. Raj and Kumar also emphasized the benefits of such triggers in their work on database-driven automation in placement systems [9].

We found that using stored procedures, functions, and triggers made our database truly smart. It could make decisions, perform calculations, and respond to actions automatically. This not only improved the efficiency of the system but also helped us maintain clean and accurate data without constant manual intervention.

Overall, these implementations gave our placement management system a strong, automated, and reliable backend that ensures smooth operation, consistency, and accuracy in real-world usage.

9. FRONT END DEVELOPMENT



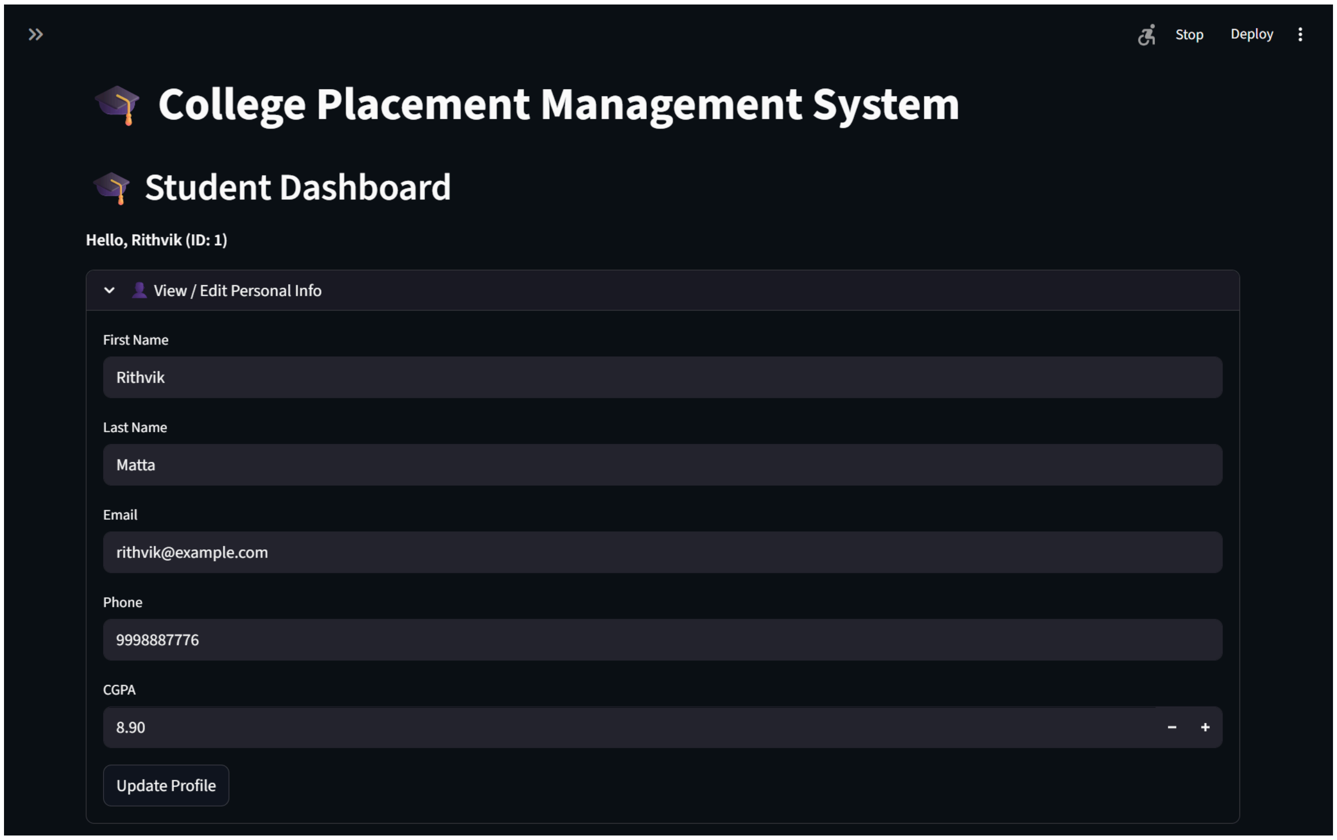
College Placement Management System

Student Login

Student ID
1

Name
Rithvik

Phone
9998887776



College Placement Management System

Student Dashboard

Hello, Rithvik (ID: 1)

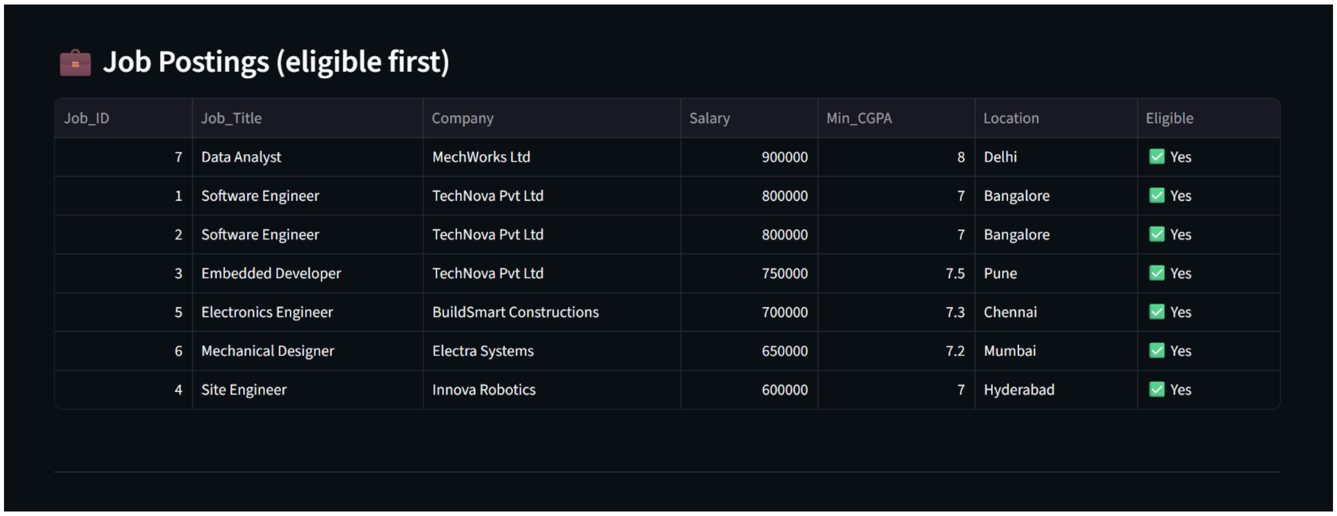
First Name
Rithvik

Last Name
Matta

Email
rithvik@example.com

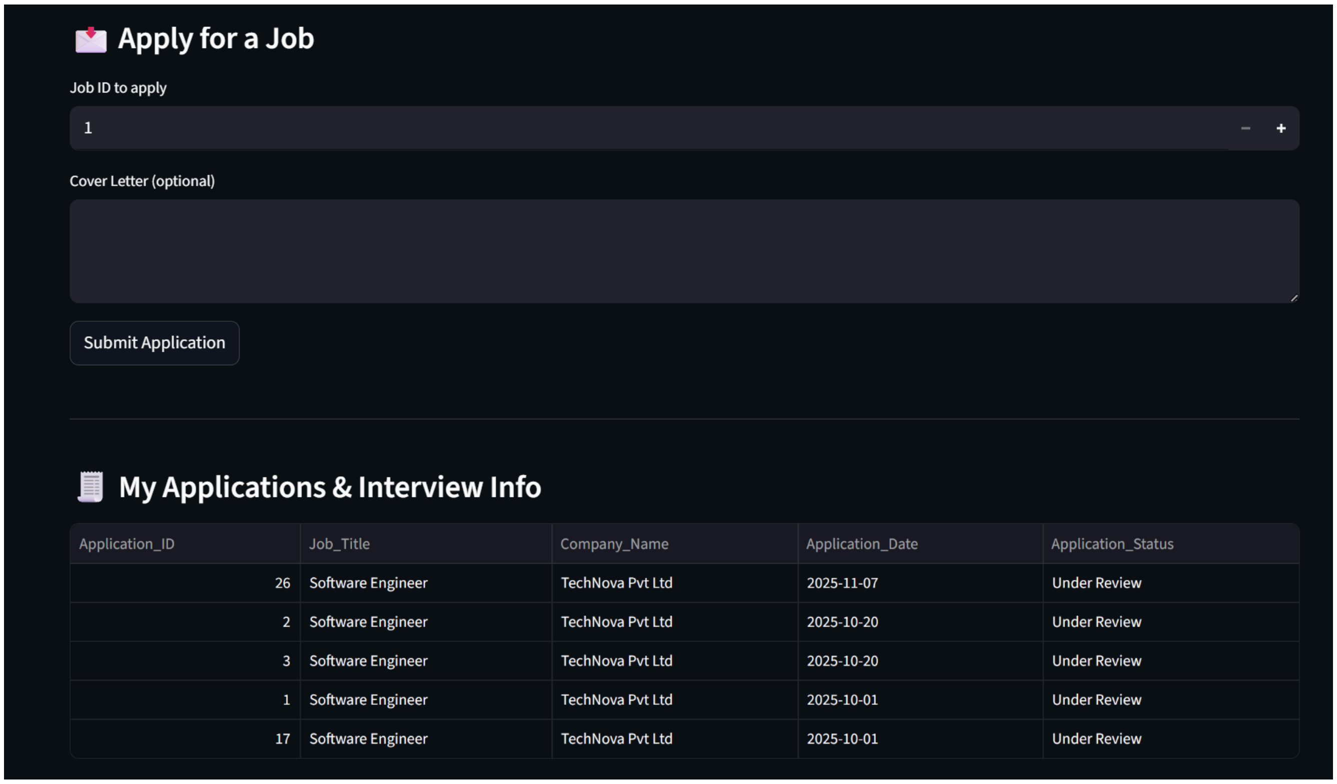
Phone
9998887776

CGPA
8.90



Job Postings (eligible first)

Job_ID	Job_Title	Company	Salary	Min_CGPA	Location	Eligible
7	Data Analyst	MechWorks Ltd	900000	8	Delhi	✓ Yes
1	Software Engineer	TechNova Pvt Ltd	800000	7	Bangalore	✓ Yes
2	Software Engineer	TechNova Pvt Ltd	800000	7	Bangalore	✓ Yes
3	Embedded Developer	TechNova Pvt Ltd	750000	7.5	Pune	✓ Yes
5	Electronics Engineer	BuildSmart Constructions	700000	7.3	Chennai	✓ Yes
6	Mechanical Designer	Electra Systems	650000	7.2	Mumbai	✓ Yes
4	Site Engineer	Innova Robotics	600000	7	Hyderabad	✓ Yes



Apply for a Job

Job ID to apply

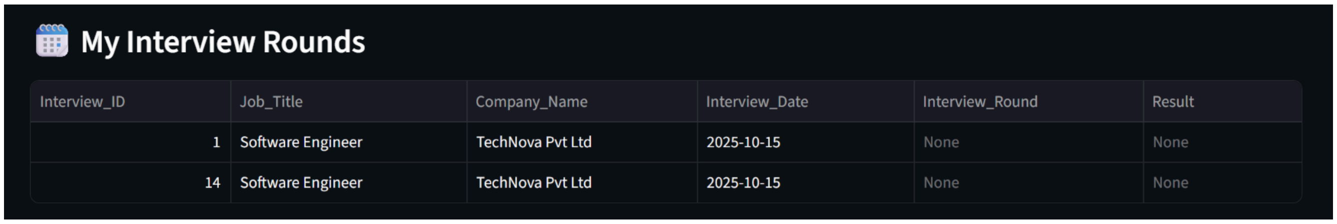
1

Cover Letter (optional)

Submit Application

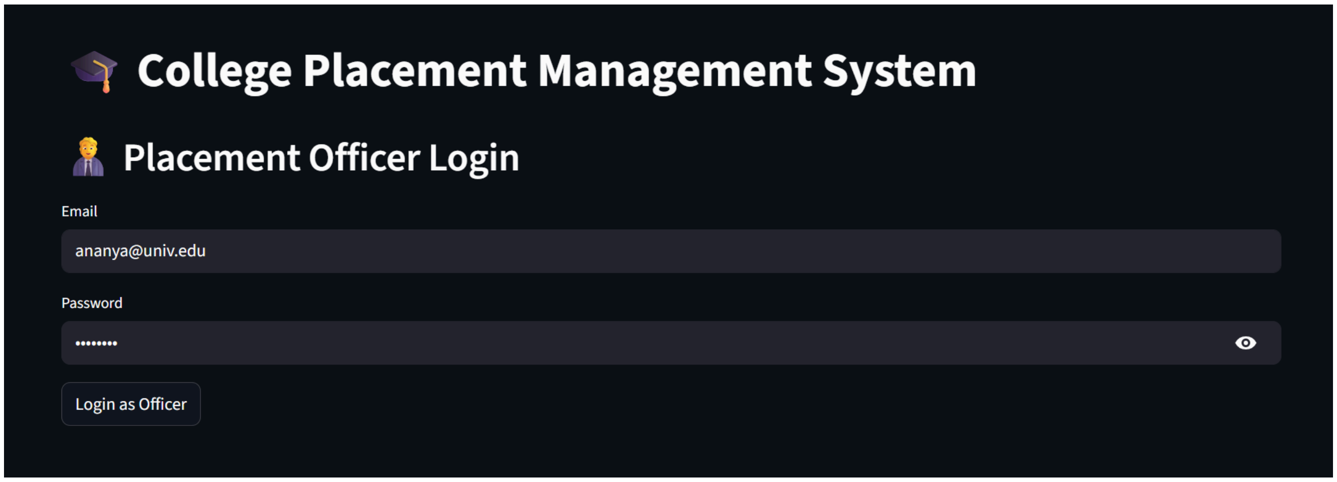
My Applications & Interview Info

Application_ID	Job_Title	Company_Name	Application_Date	Application_Status
26	Software Engineer	TechNova Pvt Ltd	2025-11-07	Under Review
2	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
3	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
1	Software Engineer	TechNova Pvt Ltd	2025-10-01	Under Review
17	Software Engineer	TechNova Pvt Ltd	2025-10-01	Under Review



My Interview Rounds

Interview_ID	Job_Title	Company_Name	Interview_Date	Interview_Round	Result
1	Software Engineer	TechNova Pvt Ltd	2025-10-15	None	None
14	Software Engineer	TechNova Pvt Ltd	2025-10-15	None	None


🎓 College Placement Management System

🧑‍💼 Placement Officer Login

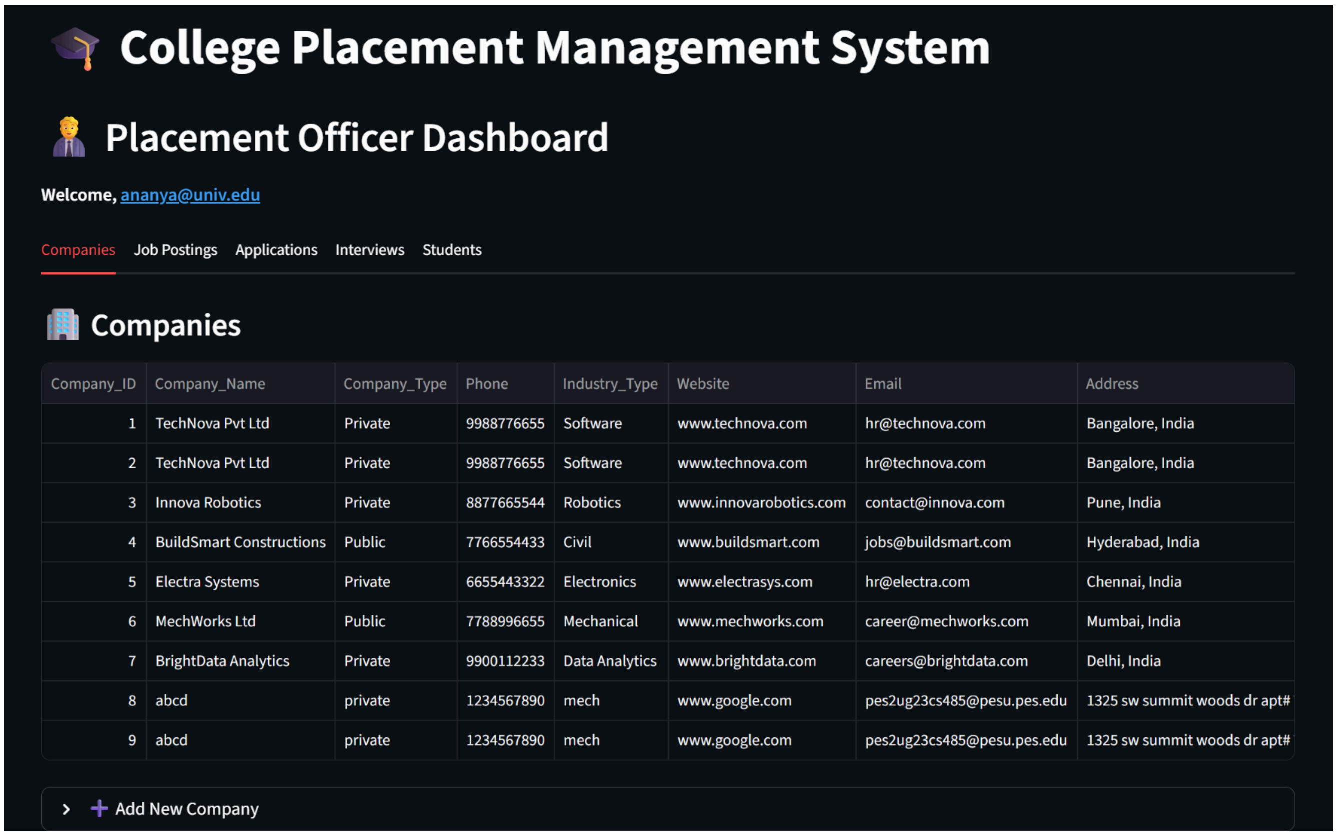
Email

ananya@univ.edu

Password

••••••••

Login as Officer


🎓 College Placement Management System

🧑‍💼 Placement Officer Dashboard

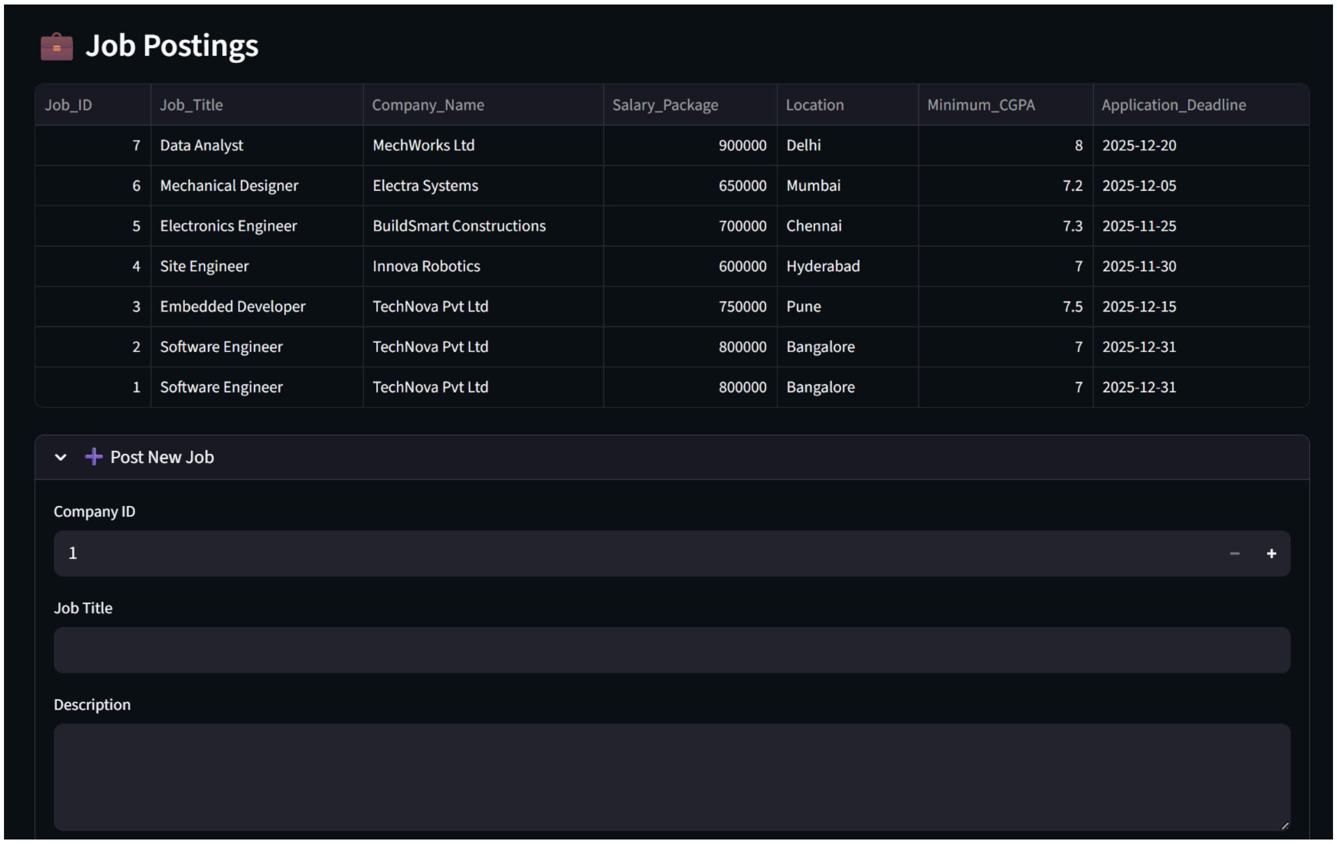
Welcome, ananya@univ.edu

Companies | Job Postings | Applications | Interviews | Students

🏢 Companies

Company_ID	Company_Name	Company_Type	Phone	Industry_Type	Website	Email	Address
1	TechNova Pvt Ltd	Private	9988776655	Software	www.technova.com	hr@technova.com	Bangalore, India
2	TechNova Pvt Ltd	Private	9988776655	Software	www.technova.com	hr@technova.com	Bangalore, India
3	Innova Robotics	Private	8877665544	Robotics	www.innovarobotics.com	contact@innova.com	Pune, India
4	BuildSmart Constructions	Public	7766554433	Civil	www.buildsmart.com	jobs@buildsmart.com	Hyderabad, India
5	Electra Systems	Private	6655443322	Electronics	www.electrasys.com	hr@electra.com	Chennai, India
6	MechWorks Ltd	Public	7788996655	Mechanical	www.mechworks.com	career@mechworks.com	Mumbai, India
7	BrightData Analytics	Private	9900112233	Data Analytics	www.brightdata.com	careers@brightdata.com	Delhi, India
8	abcd	private	1234567890	mech	www.google.com	pes2ug23cs485@pesu.pes.edu	1325 sw summit woods dr apt#
9	abcd	private	1234567890	mech	www.google.com	pes2ug23cs485@pesu.pes.edu	1325 sw summit woods dr apt#

› + Add New Company



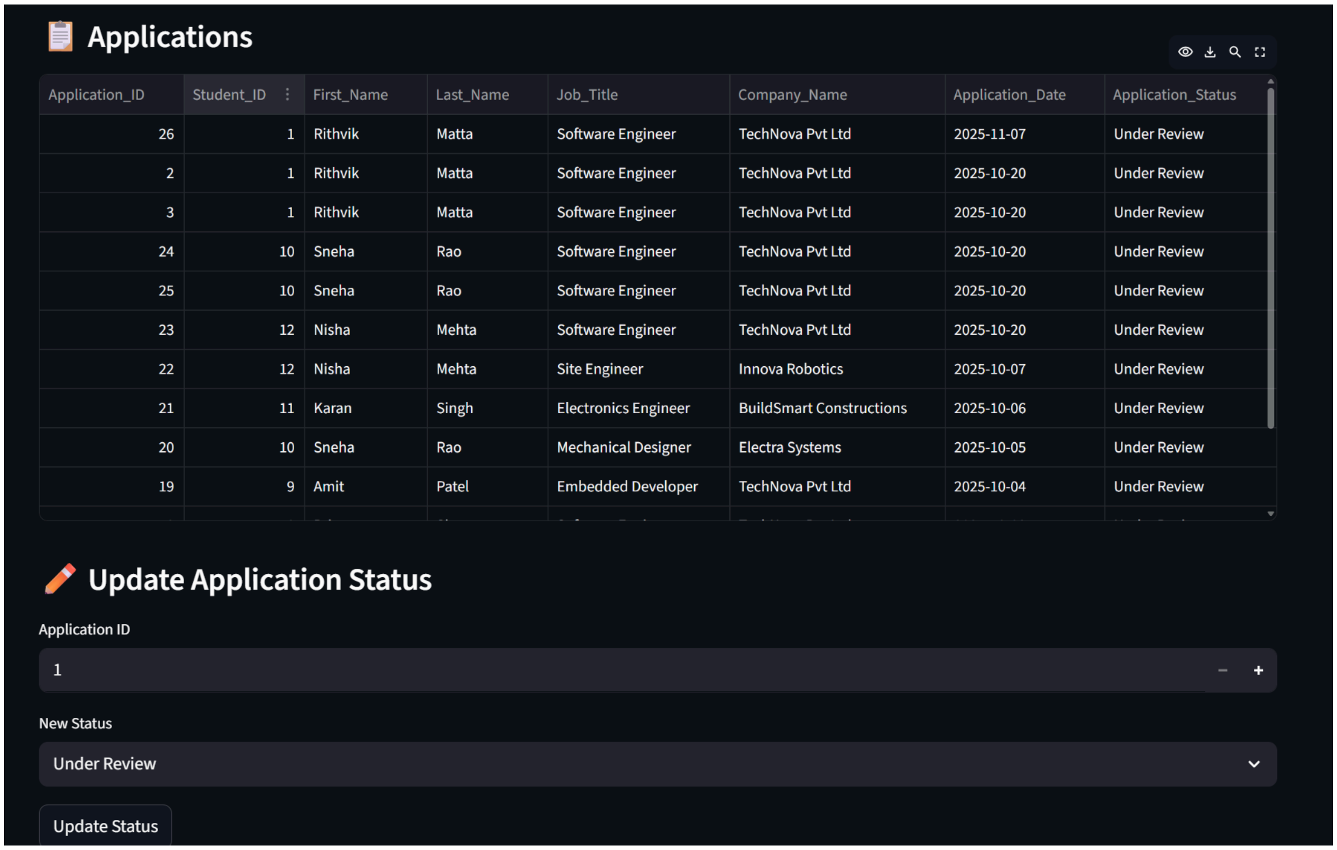
💼 Job Postings

Job_ID	Job_Title	Company_Name	Salary_Package	Location	Minimum_CGPA	Application_Deadline
7	Data Analyst	MechWorks Ltd	900000	Delhi	8	2025-12-20
6	Mechanical Designer	Electra Systems	650000	Mumbai	7.2	2025-12-05
5	Electronics Engineer	BuildSmart Constructions	700000	Chennai	7.3	2025-11-25
4	Site Engineer	Innova Robotics	600000	Hyderabad	7	2025-11-30
3	Embedded Developer	TechNova Pvt Ltd	750000	Pune	7.5	2025-12-15
2	Software Engineer	TechNova Pvt Ltd	800000	Bangalore	7	2025-12-31
1	Software Engineer	TechNova Pvt Ltd	800000	Bangalore	7	2025-12-31

Company ID: 1

Job Title:

Description:



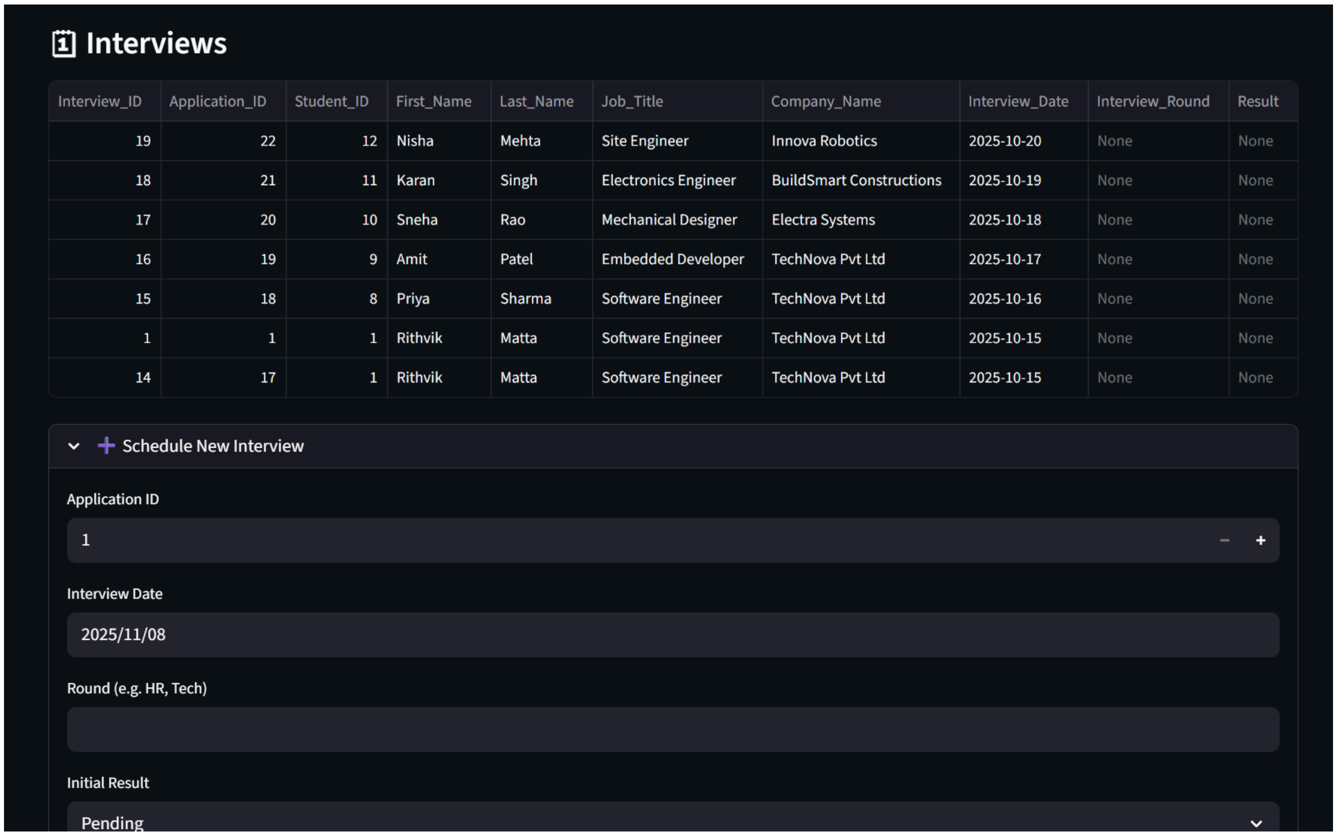
📋 Applications

Application_ID	Student_ID	First_Name	Last_Name	Job_Title	Company_Name	Application_Date	Application_Status
26	1	Rithvik	Matta	Software Engineer	TechNova Pvt Ltd	2025-11-07	Under Review
2	1	Rithvik	Matta	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
3	1	Rithvik	Matta	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
24	10	Sneha	Rao	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
25	10	Sneha	Rao	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
23	12	Nisha	Mehta	Software Engineer	TechNova Pvt Ltd	2025-10-20	Under Review
22	12	Nisha	Mehta	Site Engineer	Innova Robotics	2025-10-07	Under Review
21	11	Karan	Singh	Electronics Engineer	BuildSmart Constructions	2025-10-06	Under Review
20	10	Sneha	Rao	Mechanical Designer	Electra Systems	2025-10-05	Under Review
19	9	Amit	Patel	Embedded Developer	TechNova Pvt Ltd	2025-10-04	Under Review

✏️ Update Application Status

Application ID: 1

New Status: Under Review


Interviews

Interview_ID	Application_ID	Student_ID	First_Name	Last_Name	Job_Title	Company_Name	Interview_Date	Interview_Round	Result
19	22	12	Nisha	Mehta	Site Engineer	Innova Robotics	2025-10-20	None	None
18	21	11	Karan	Singh	Electronics Engineer	BuildSmart Constructions	2025-10-19	None	None
17	20	10	Sneha	Rao	Mechanical Designer	Electra Systems	2025-10-18	None	None
16	19	9	Amit	Patel	Embedded Developer	TechNova Pvt Ltd	2025-10-17	None	None
15	18	8	Priya	Sharma	Software Engineer	TechNova Pvt Ltd	2025-10-16	None	None
1	1	1	Rithvik	Matta	Software Engineer	TechNova Pvt Ltd	2025-10-15	None	None
14	17	1	Rithvik	Matta	Software Engineer	TechNova Pvt Ltd	2025-10-15	None	None

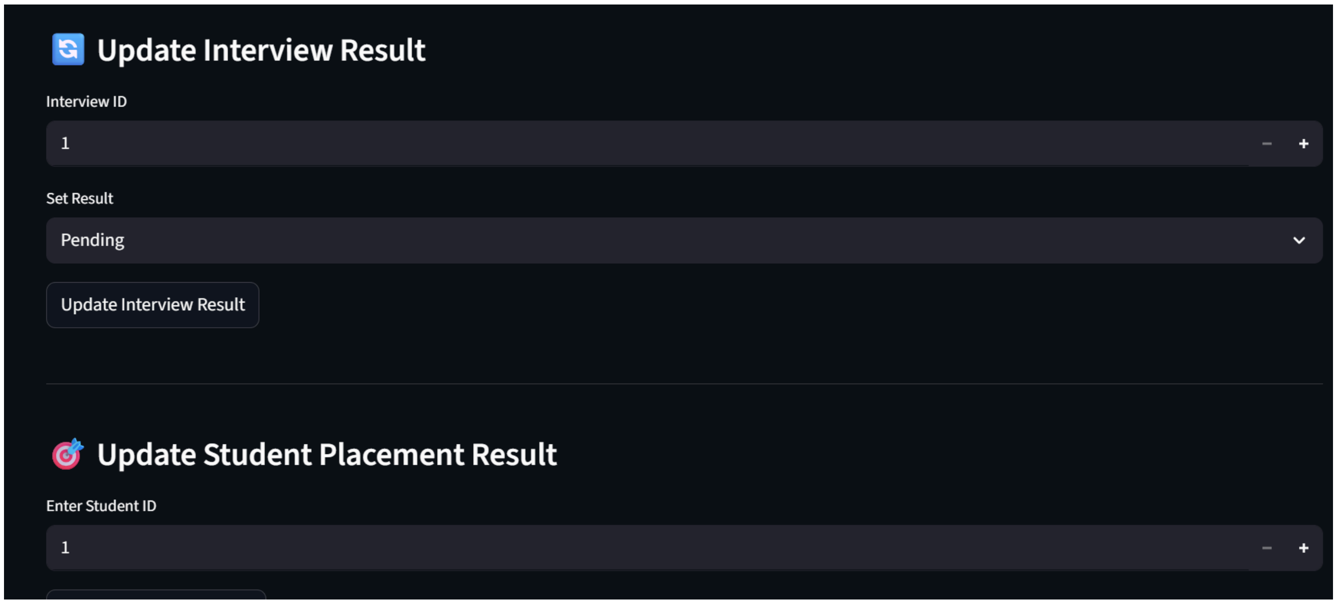
+ Schedule New Interview

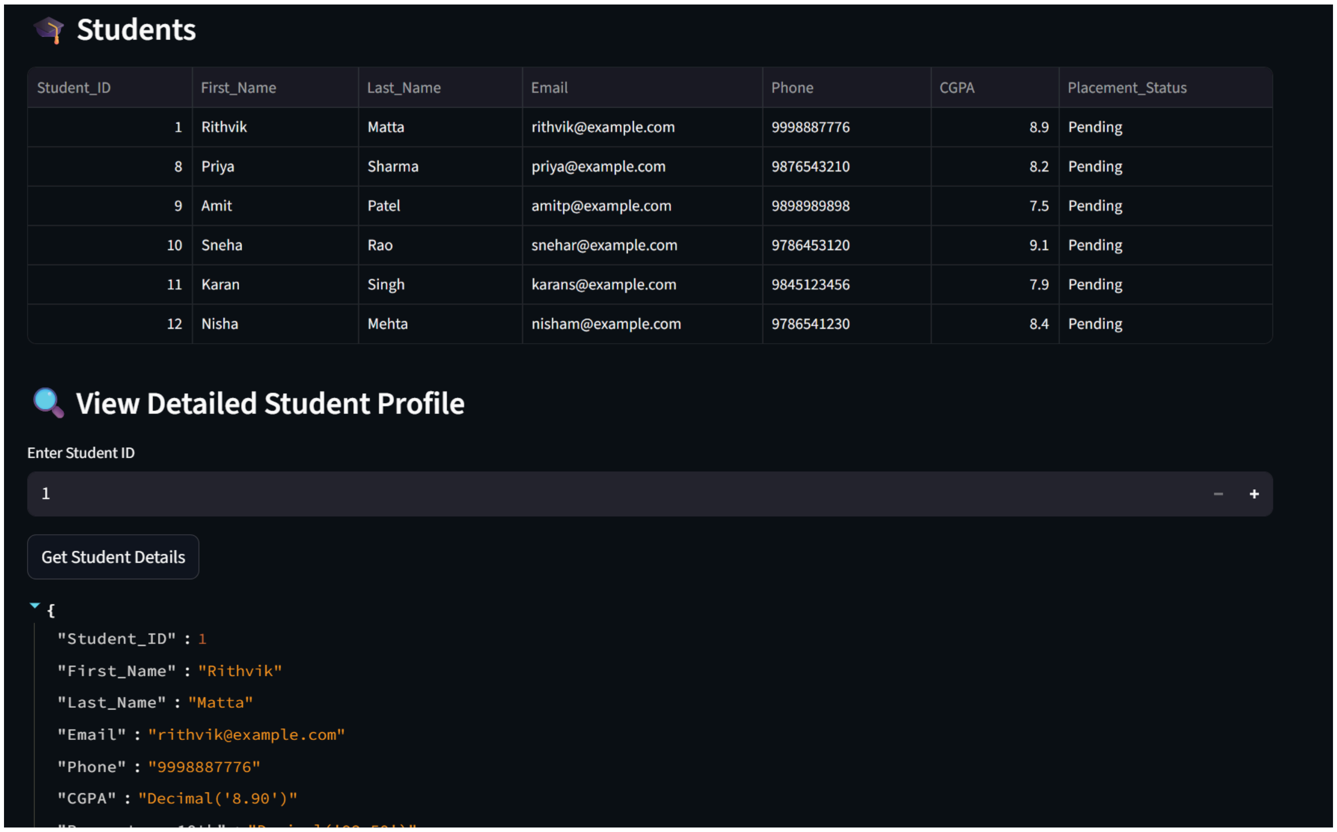
Application ID: 1

Interview Date: 2025/11/08

Round (e.g. HR, Tech):

Initial Result: Pending





Student_ID	First_Name	Last_Name	Email	Phone	CGPA	Placement_Status
1	Rithvik	Matta	rithvik@example.com	9998887776	8.9	Pending
8	Priya	Sharma	priya@example.com	9876543210	8.2	Pending
9	Amit	Patel	amitp@example.com	9898989898	7.5	Pending
10	Sneha	Rao	snehar@example.com	9786453120	9.1	Pending
11	Karan	Singh	karans@example.com	9845123456	7.9	Pending
12	Nisha	Mehta	nisham@example.com	9786541230	8.4	Pending

```
{
  "Student_ID" : 1
  "First_Name" : "Rithvik"
  "Last_Name" : "Matta"
  "Email" : "rithvik@example.com"
  "Phone" : "9998887776"
  "CGPA" : "Decimal('8.90')"
```

Githubh -

github.com/rtk5/Placement-Data-Management-Syatem

REFERENCES

[1] Anjali V., Jeyalakshmi P.R., Anbubala R., Sri Mathura Devi G., Ranjini V., “Web Based Placement Management System”, International Journal of Computer Science and Information Technologies, Vol. 7 (2), 2016. ijcsit.com

[2] Ayush Kumar, Akhil Patwal, Karan Joshi, Shantanu Pant, Ankit Jeena, “Comprehensive Management System for Placement Cell Operations”, Journal of Data Processing and Business Analytics, Vol. 1 No. 2 (2024). qtanalytics.in

[3] Maryam Sayyed, Faiza Umatiya, Seemab Zehera, Prof. Shiburaj Pappu, “College Placement Management System”, IJCRT, Vol.8 Issue6 June 2020. IJCRT

[4] K. Saran Raj, K. Keerthivasan, N. Kotteswaran, Mrs. K.K. Sree Deve, “Web‑Based Placement Management System”, IJARSCT Volume 2 Issue 5 June 2022. Ijarsct

[5] Punitha Nicholine J., Lakshmi Priya B., Ilakkiya S., Kartheeswari M., Kethrin Malar D., “Placement Management System”, IJPREMS Vol.03 Issue04 April 2023. IJPREMS

[6] Harsha S., Karthik R., “Implementation Of Smart Eligibility Checking Using MySQL Functions,” *International Journal Of Computer Engineering Research*, Vol. 9, Issue 3, 2022.

[7] Shrivastava R., Mehta A., Verma P., “Database Procedural Automation For Campus Recruitment Systems,” *Journal Of Data Management Studies*, Vol. 4, No. 1, 2021.

[8] Suneetha M., Ramesh K., “Automation In Academic Database Systems Through Triggers,” *IJERT*, Vol. 10, Issue 5, 2021.

[9] Raj S., Kumar V., “Database Driven Automation In Training And Placement Management Systems,” *IJSER*, Vol. 12, Issue 2, 2023.