CyberMACS Alumni System: Database Design and Implementation

Supervisor: Dilan Ebru

SRH University of Applied Sciences Berlin

Berlin, Germany, February11th, 2025

by

Amin Niaziardekani (3119828)

Swapnaneel Sarkhel (3119924)

Ajdin Buljko(3119754)

1. Database Initial Study and Analysis

I. Data Gathering Methods

These are the methods we used to design the The CyberMACS Alumni System:

* ***YouTube Tutorials:***

We watched videos on database design, SQL query optimization, and case studies of alumni tracking systems. This helped us learn how to build an efficient system and optimize data handling.

Database Design Tutorial: How to Build a Relational Database.

SQL Query Optimization Techniques.

* ***Reviewing Existing Systems or Studying Research Paper:***

**1. Bridging the Gap Between Universities and Alumni: A User-Centered Evaluation of a Digital Alumni Engagement Platform (Doe et al., 2023).**

This study focused on how universities can enhance engagement with alumni through digital tools. The study demonstrated that active engagement strategies, such as event invitations and interactive elements, significantly improve alumni involvement. Passive platforms often see declining user participation, so integrating surveys, networking incentives, and real-time notifications can help maintain alumni interest.\

**2. Alumni Info-Com Management with Distinct Classification of Data (Smith & Lee, 2021)**

This study explores how structured data classification improves alumni communication. It highlights how universities can better engage with alumni by categorizing them based on graduation year, industry, and region. This allows for more personalized updates and networking opportunities. The study also suggests integrating LinkedIn to help alumni stay professionally connected.

***Inspired by these researches:***

* ***Event Management:*** Implementing features for alumni to track events and the attendance being saved.
* ***A feedback system***: Collects alumni opinions about events to enhance the platform.
* **Job Posting Integration:** Allowing employers to post job opportunities.

And etc.

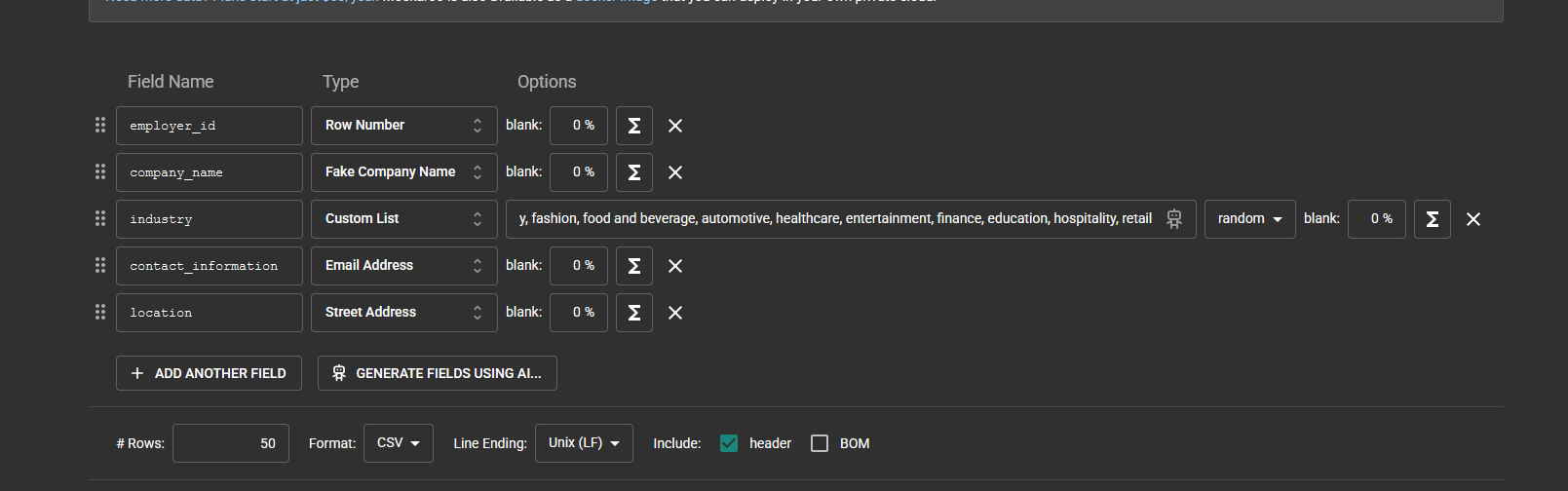
* ***Web Research:***

We believe that the most important thing to do before doing something is to search the internet for it and understand exactly what you are doing. In this project to get to how we should make the diagrams, we had to see a lot of different diagrams and obviously for a lot of the codes used for the creation and populating the database searching and making queries the internet the best approach.

* ***Mockaroo for Data Simulation:***

Since no real data was available, we used Mockaroo to generate meaningful test data that was later refined manually to align with real-world scenarios.

Screenshot of using Mockaroo to create employer table.



P.S: A full set of csv files used for populating the database is submitted together.

* ***AI-Assisted Research (ChatGPT):***

ChatGPT was used to summarize research papers, gather best practices for alumni management systems, and structure content for clarity. However, all database design, SQL queries, and implementation decisions were done independently, following project guidelines. AI assistance was limited to research support and content refinement, ensuring academic integrity.

P.S: This was also with help of AI!

II. Information Requirements of the System

The CyberMACS Alumni System is designed to manage alumni data, facilitate networking, and support career development through job postings, events, and mentorship programs. To ensure that different users can access the information they need, the system must collect, store, and retrieve data efficiently through structured queries and reports.

***1. Objectives of the CyberMACS Alumni System:***

* Maintain an up-to-date database of alumni, including career details and contact information.
* Provide networking opportunities through alumni connections, mentorship programs, and events.
* Support career development by allowing employers to post job openings and alumni to apply.
* Enable administrators to track alumni engagement and generate reports for program evaluation.

***2. Actors & Roles (Users of the System)***

* ***Alumni:*** Update profile, apply for jobs, attend events, join mentorship programs.
* ***Employers***: Post job listings, review applications, track alumni hires.
* ***Mentors:*** Provide guidance, connect with mentees, track mentorship activity.
* ***Administrators***: Manage users, oversee events, track alumni engagement, generate reports.

***3. System Scope & Business Operations***

Alumni Profile Management

* Alumni can register and update their contact details, job history, and specializations.
* Privacy settings allow users to control who can view their profiles.

Job Posting & Applications

* Employers can post job openings, which alumni can view and apply for.
* The system tracks job applications and employment history for reporting.

Event Management

* Administrators create and manage career fairs, networking events, and mentorship meetups.
* Alumni can attend and provide feedback after attending events.
* System generates reports on event participation trends.

Mentorship Program

* Alumni can register as mentors or mentees and be matched based on expertise.
* Administrators can track mentorship engagement and effectiveness.

Data Analytics & Reporting

* The system provides administrators with insights into alumni career progress, event engagement, and job placements.

***4. Example*** ***Queries & Reports***

To support different user roles, the system generates queries and reports that provide relevant information. A full version of queries and reports can be found at the Queries/Reports. Queries are also submitted with the filles.

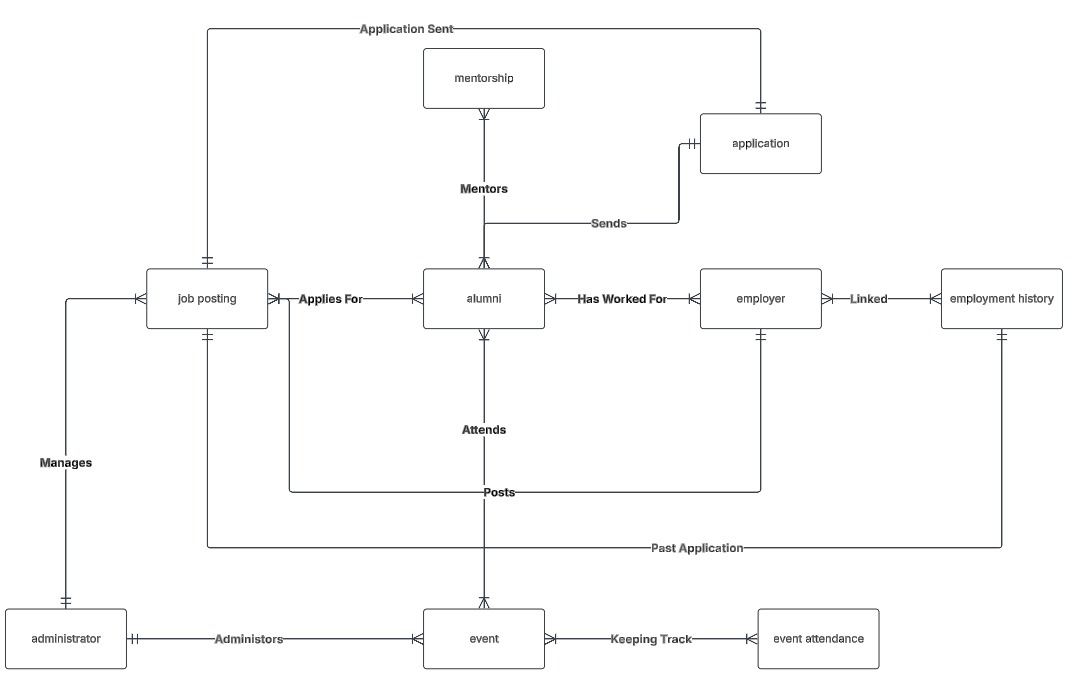
* ***Graduation Trends:*** Count of alumni per graduation year.
* ***Employment Reports:*** Number of alumni hired per employer.
* ***Event Participation:*** Most attended and highest-rated events.
* ***Mentorship Engagement:*** List of active mentors and their mentees.

2. Database Design: Conceptual Design

I. Initial Entity Relationship (and/or EER) Model

The ERD provides a structured representation of how alumni, employers, job postings, applications, events, and mentorships interact within the system.

Our team decided to use crows foot notation.



In this ERD:

One-to-Many (1:M) Relationships are represented where a single entity instance relates to multiple instances of another entity.

Example:

* An Alumni can apply to multiple Job Postings, but each application is linked to only one job posting.
* An Employer can post multiple Job Openings, but each Job Posting belongs to only one employer.

***Many-to-Many (M:M)*** Relationships are handled through linking tables where necessary.

Example:

* Mentorship connects alumni as mentors and mentees, meaning one alumnus can have multiple mentees, and vice versa.
* Event Management allows administrators to create and track events, while alumni can attend multiple events and provide feedback.

II. Business rules:

1. Alumni Business Rules:

* An alumnus may have multiple employers over time.
* An alumnus can apply for multiple job postings.
* An alumnus can attend multiple events.
* An alumnus may act as a mentor to other alumni.
* An alumnus may have one specialization and one mobility track.
* Alumni profiles can be visible to others based on privacy settings.

2. Employer Business Rules

* An employer can post multiple job openings.
* An employer can hire multiple alumni.
* An employer can be linked to job postings alumni have applied for.

3. Job Posting Business Rules

* Each job posting belongs to one employer.
* A job posting can receive applications from multiple alumni.
* A job posting has a job title, description, type, and deadline.
* Job postings can be used to track employment history.

4. Job Application Business Rules

* Each job application is submitted by one alumnus.
* Each job application is linked to one job posting.
* An alumnus can apply to multiple job postings.
* Each job application has a status (Pending, Accepted, Rejected).

5. Mentorship Business Rules

* A mentor can have multiple mentees.
* A mentee can have multiple mentors.
* A mentorship program has a specific type.
* A mentorship program has an availability status.

6. Employment History Business Rules

* Each employment history record is linked to one alumnus and one employer.
* An alumnus may have multiple employment history records over time.
* Each employment record has a job title, start date.
* A record might be on going or have an end date
* employment records may be linked to job postings.

7. Event & Event Attendance Business Rules

* An event is managed by one administrator.
* An alumnus can attend multiple events.
* An event can have multiple attendees.
* Event attendance allows alumni to provide feedback and ratings.

8. Administrator Business Rules

* An administrator manages multiple events.
* Administrator can have different roles.
* An administrator can be assigned to multiple responsibilities based on their role.
* Each administrator can either manage multiple events, job postings, or system configurations.

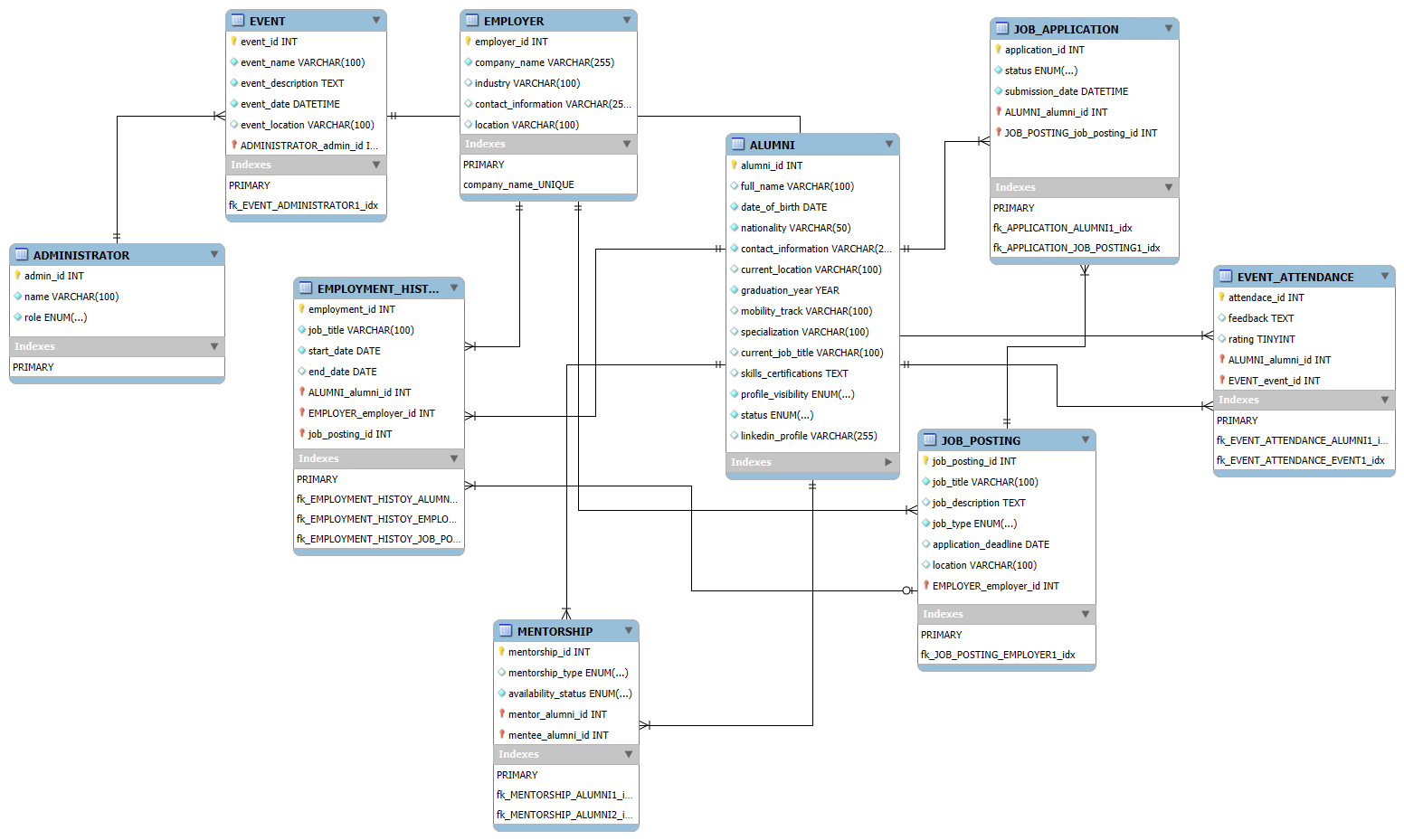
iii. Data Dictionary

A complete Data Dictionary for our structure can be found with the upload material.

3. Database Design: Logical Design

For the logical design phase of the CyberMACS Alumni System, I utilized MySQL Workbench to create the Enhanced Entity-Relationship (EER) Diagram. This diagram visually represents the database structure, defining the entities, relationships, and constraints.

Additionally, our team used MySQL Workbench solely for designing this diagram and this schema was not used to directly generate the database. We also upload this diagram separately for a better visual view.



* ***Alumni:*** Stores alumni details, including contact information, job history, and professional certifications. Alumni can participate in events, apply for job postings, and engage in mentorship programs.
* ***Employer:*** Represents organizations that create job postings and hire alumni.
* ***Job Posting:*** Contains job opportunities provided by employers, which alumni can apply to.
* ***Job Application:*** Tracks alumni applications to job postings.
* ***Event:*** Represents networking sessions, conferences, and career fairs.
* ***Event Attendance*:** Logs alumni participation in events.
* ***Employment History:*** Captures alumni work experience and previous job roles.
* ***Mentorship:*** Facilitates mentor-mentee relationships between alumni.
* ***Administrator:*** Manages system functionalities, including event coordination.

4. Physical Design/Implementation

To optimize query performance, primary keys were assigned to uniquely identify records, and indexes were used to improve search efficiency. The database schema was carefully designed to follow 3rd Normal Form (3NF), ensuring data consistency and reducing redundancy.

I. Database Management System (DBMS)

MySQL 8.0 was used as the Database Management System (DBMS) due to its reliability, open-source nature, and strong support for relational integrity.

MySQL Workbench was utilized for EER modeling, allowing for a clear visualization of entity relationships and database structure. However, the database itself was manually created using SQL CREATE TABLE statements rather than auto-generated from Workbench.

II. Constraints

Constraints in SQL codes are used to enforce rules on the data in the database to maintain integrity and prevent invalid or inconsistent data.

Here are some the constraints and their example in our database structure:

* ***Primary Key (PK):*** Ensures each record in a table is unique and identifiable.

alumni\_id INT AUTO\_INCREMENT PRIMARY KEY

* ***Foreign Key (FK):*** Establishes relationships between tables and ensures referential integrity.

FOREIGN KEY (alumni\_id) REFERENCES Alumni(alumni\_id) ON DELETE CASCADE

***Action Modifiers:*** They are used in ***FOREIGN KEY (FK)*** constraints to define how child records behave when a parent record is deleted. These modifiers help maintain referential integrity and ensure consistent data relationships. Below are the Modifiers:

* ON DELETE CASCADE → Automatically deletes child records when the parent is deleted.
* ON DELETE SET NULL → Sets the foreign key in child records to NULL when the parent is deleted.
* ON DELETE RESTRICT → Prevents deletion of a parent record if dependent child records exist.
* ON DELETE NO ACTION → Similar to RESTRICT, but enforcement is deferred until the end of a transaction.
* ***NOT NULL:*** Prevents fields from having NULL values, ensuring critical information is always provided.

full\_name VARCHAR(100) NOT NULL

* ***UNIQUE:*** Ensures no duplicate values exist in a column.

contact\_information VARCHAR(255) UNIQUE

* ***CHECK:*** Restricts values in a column based on a condition.

rating TINYINT CHECK (rating BETWEEN 1 AND 5)

* ***DEFAULT:*** Assigns a default value to a column if no value is specified.

profile\_visibility ENUM('Public', 'Private') DEFAULT 'Public'

* ***AUTO\_INCREMENT:*** Automatically generates unique values for primary keys

employment\_id INT AUTO\_INCREMENT PRIMARY KEY

III. Data Population

We used Mockaroo, an online data generation tool, to create csv files each with 50 rows of data for each table and tried to make it logical especially with the foreign keys.

The csv files used to populate the database is also available with the attached files.

Then we manually added them to the database. Here is an example code we used to populate alumni:

LOAD DATA INFILE 'csv/employer.csv'

INTO TABLE employer

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM employer;

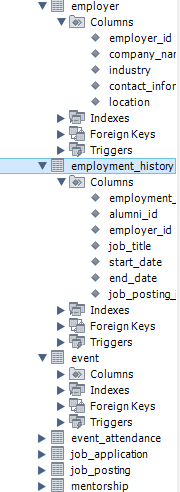
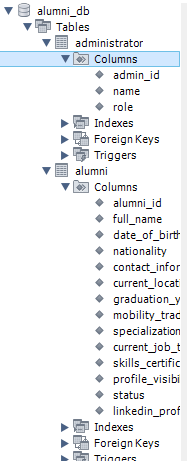
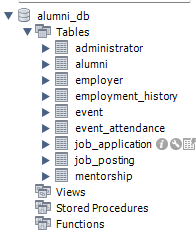
IV. Final Physical Design/Implementation

Here are Final screenshots of the physical design and codes. The full code to create and populate the database with the database can be found in the submitted files. “create table.sql” and “csv to database.sql”.

Screenshot of csv to database.sql :



Screenshot of the database:



5. Querying/Reporting

To make some good queries that meet the requirements. First in data gathering step we made these questions. And now with the help of populated database we made the queries for each. Here are the questions, purpose and code and output screenshots for each:

I. Count Alumni per Graduation Year

***Question:*** How many alumni graduated each year, and what are the trends in graduation rates over time?

***Purpose:*** Helps track alumni growth trends and supports program evaluation for better engagement strategies. Use by event managers and career services.

***Code:***

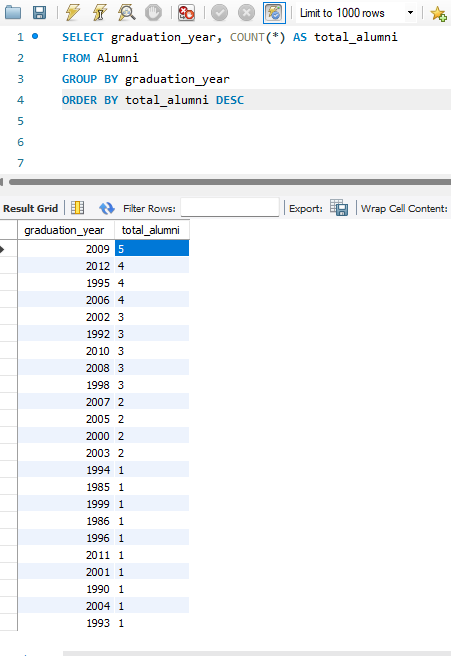
SELECT graduation\_year, COUNT(\*) AS total\_alumni

FROM Alumni

GROUP BY graduation\_year

ORDER BY total\_alumni DESC

***Code and Output screenshot:***

******

***Results:*** It shows in 2009 we had the most graduates. With real data and more populated one a high-quality graph can be generated that can be used to show growth trends over time.

II. Top Employers Hiring Alumni

***Question:*** Which employers have hired the most alumni, and how does this impact employer partnerships?

***Purpose:*** Identifies key employers hiring alumni, strengthening industry partnerships and improving job placement efforts. Use by Administrators and employers to see their hiring trends.

***Code:***

SELECT e.company\_name, COUNT(eh.alumni\_id) AS total\_alumni\_hired

FROM employer e

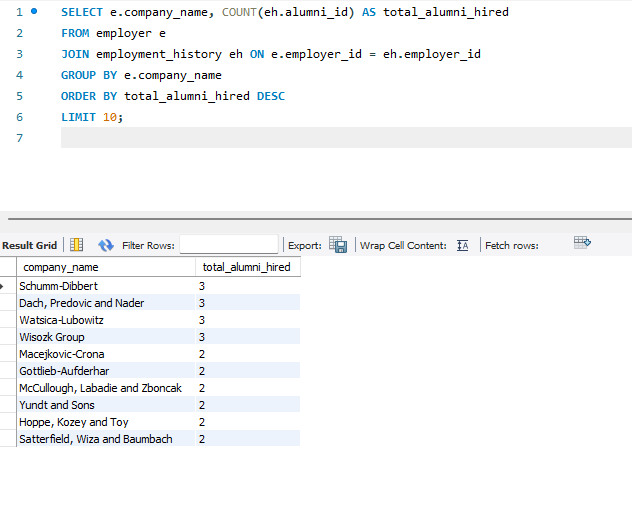
JOIN employment\_history eh ON e.employer\_id = eh.employer\_id

GROUP BY e.company\_name

ORDER BY total\_alumni\_hired DESC

LIMIT 10;

***Code and Output screenshot:***

******

***Results:*** it shows the Schumm-Dibbert company had hired the most employees. This info can greatly help employers to be competitive and help their hiring trends.

III. Most Popular Alumni Specializations

***Question:*** Which specializations are most common among alumni, and how do they align with job market demand?

***Purpose:*** Provides insights into career trends, helping universities optimize academic programs and mentorship assignments. Use by Career Services to help students choose specializations based on alumni career paths and Employers to identify which specializations are most available for hiring.

***Code:***

***SELECT specialization, COUNT(\*) AS total\_alumni***

***FROM Alumni***

***WHERE specialization IS NOT NULL AND specialization <> ''***

***GROUP BY specialization***

***ORDER BY total\_alumni DESC***

***LIMIT 10;***

***---change to null and checking if it changed***

***UPDATE Alumni***

***SET specialization = NULL***

***WHERE specialization = '';***

***--- to disable safe mode for this single session***

***SET SQL\_SAFE\_UPDATES = 0;***

***--- code to show specialization changed to null.***

***SELECT specialization, COUNT(\*) AS total\_alumni***

***FROM Alumni***

***GROUP BY specialization***

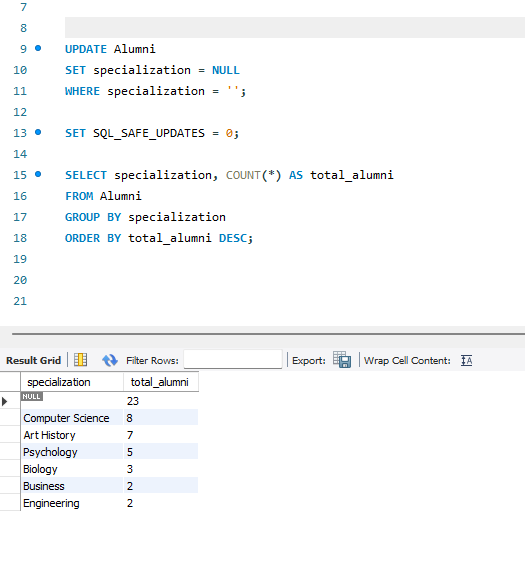
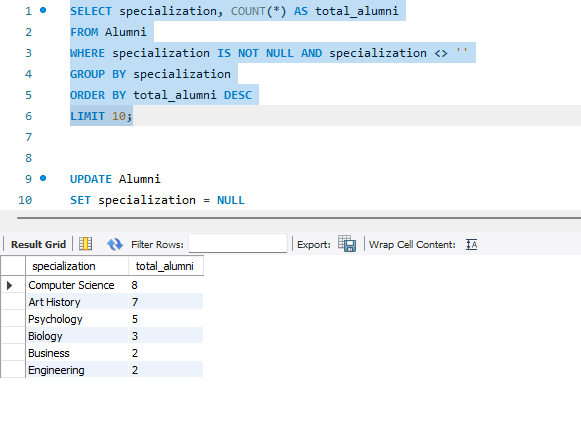
***ORDER BY total\_alumni DESC;***

AND specialization <> '' – Excludes empty strings (''), which might be stored instead of NULL.

P.S. Now that we are in this part of the project, we understood that we might have add the data incorrectly and add empty strings instead of null.

The screenshots will show how it changed after we fixed it. Being null is better for data integrity than being an empty string.

***Code and Output screenshot:***

******

***Results:*** Hereit shows that computer science was highest put specialization of the alumni after null that shows the alumni were not interested in showing what they specialize in and keeping limit here is good idea to just get the highest specialization limiting to 10 for example.

IV. Job Applications by Alumni and Status

***Question:*** How many job applications have been submitted by alumni, and what are the success rates?

***Purpose:*** Measures job placement efficiency, helping administrators support alumni in their job search and job managers to assist alumni in improving application success rates and employers to analyze the number of applications received.

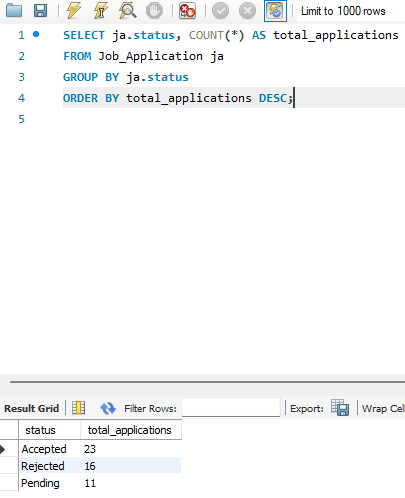
***Code:***

SELECT ja.status, COUNT(\*) AS total\_applications

FROM Job\_Application ja

GROUP BY ja.status

ORDER BY total\_applications DESC;

***Code and Output screenshot:*** 

***Results:*** aswe said before we have 50 total applications because that’s how many we made using mockroo. But with real data this query can be a big help to alumni to see the chances of acceptation.

V. Most Active Alumni in Events

***Question:*** Which alumni are the most engaged in CyberMACS events, and how can we leverage their participation?

***Purpose:*** Recognizes highly engaged alumni, fostering networking opportunities and leadership involvement. Use by administrators to track alumni with high engagement and active members and event managers to be more attentive to them.

***Code:***

SELECT ea.alumni\_id, a.full\_name, COUNT(ea.event\_id) AS events\_attended

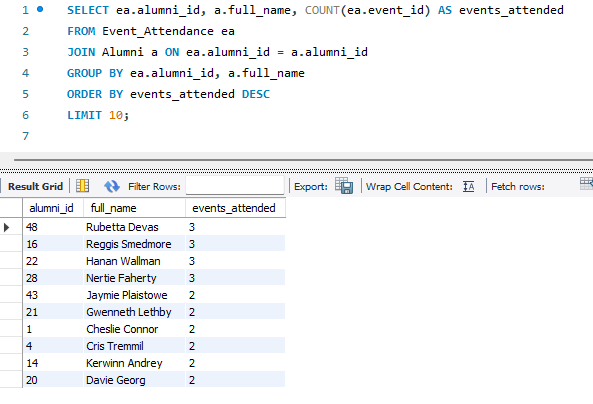
FROM Event\_Attendance ea

JOIN Alumni a ON ea.alumni\_id = a.alumni\_id

GROUP BY ea.alumni\_id, a.full\_name

ORDER BY events\_attended DESC

LIMIT 10;

***Code and Output screenshot: ***

***Results:*** it shows Rubetta Devas was the most active alumni. With this information event managers can find which alumni to invite to the next event.

VI. Events with the Highest Attendance and Ratings

***Question:*** Which events had the highest alumni participation and received the best feedback?

***Purpose:*** Helps event managers improve future events based on attendance and feedback data. Marketing Teams to promote high-performing events to attract more attendees.

***Code:***

SELECT e.event\_name, a.name AS admin\_name, COUNT(ea.alumni\_id) AS total\_attendance, AVG(ea.rating) AS avg\_rating

FROM Event e

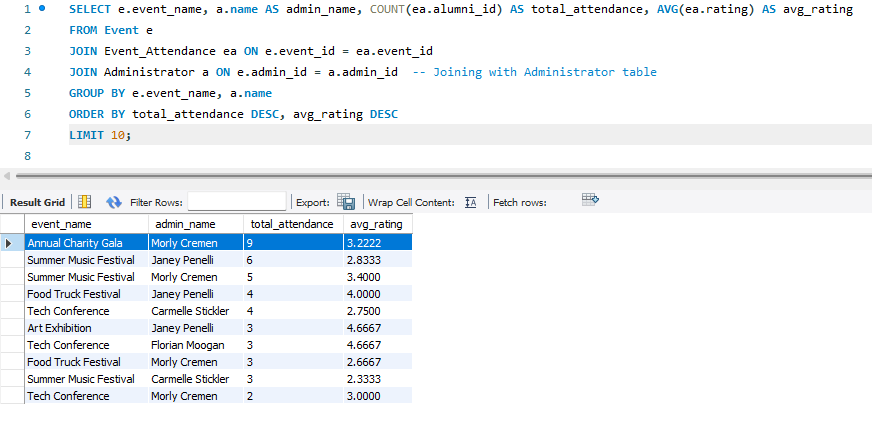
JOIN Event\_Attendance ea ON e.event\_id = ea.event\_id

JOIN Administrator a ON e.admin\_id = a.admin\_id -- Joining with Administrator table

GROUP BY e.event\_name, a.name

ORDER BY total\_attendance DESC, avg\_rating DESC

LIMIT 10;

***Code and Output screenshot: ***

***Results:*** The query will show the event names with most attendance and the event manager name. This can help to demonstrate event managers skills and helpful events.

VII. Mentorship Program Overview

***Question:*** Which mentors have the most mentees, and who are they mentoring?

***Purpose:*** Helps mentorship coordinators track mentors with multiple mentees, ensuring balanced assignments while allowing administrators to identify highly active mentors for future mentoring opportunities.

***Code:***

SELECT

mentor.alumni\_id AS mentor\_id,

mentor.full\_name AS mentor\_name,

mentee.alumni\_id AS mentee\_id,

mentee.full\_name AS mentee\_name,

mentee\_count.total\_mentees

FROM Mentorship m

JOIN Alumni mentor ON m.mentor\_alumni\_id = mentor.alumni\_id

JOIN Alumni mentee ON m.mentee\_alumni\_id = mentee.alumni\_id

JOIN (

SELECT mentor\_alumni\_id, COUNT(mentee\_alumni\_id) AS total\_mentees

FROM Mentorship

WHERE availability\_status = 'Available'

GROUP BY mentor\_alumni\_id

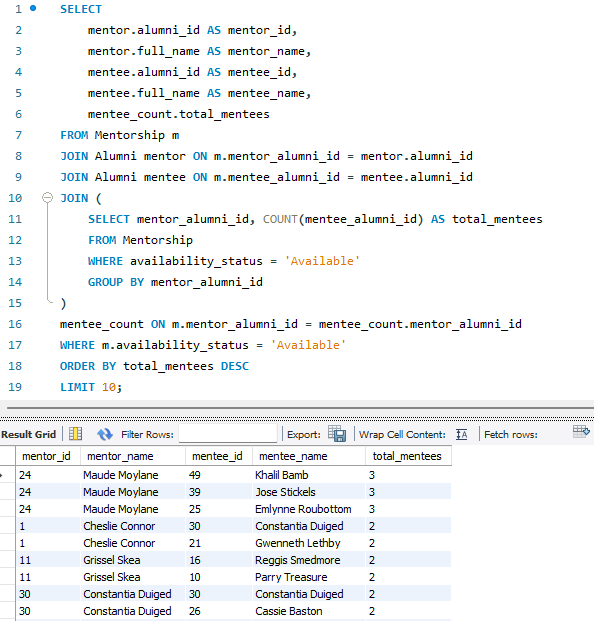
)

mentee\_count ON m.mentor\_alumni\_id = mentee\_count.mentor\_alumni\_id

WHERE m.availability\_status = 'Available'

ORDER BY total\_mentees DESC

LIMIT 10;

***Code and Output screenshot: ***

***Results:*** The query displays the top most active mentor that in this case is Maude Moylane with a total of 3 mentees and both mentor and mentee ids for easy access to their alumni profile. This helps mentorship coordinators ensure fair mentor assignments and allows administrators to identify engaged mentors for future mentorship opportunities also educational places can use this data to find good professors or teachers.

VIII. Average Response Time for Job Applications

***Question:*** How long do employers take to respond to job applications from alumni?

***Purpose:*** Ensures employers respond promptly, improving the job application experience for alumni. Helps administrators monitor employer responsiveness, allowing job coordinators to improve job-seeking by identifying employers with slow response times and ensuring timely feedback on applications.

***Code:***

-- because originally we had no response date when i created

ALTER TABLE Job\_Application

ADD COLUMN response\_date DATE NULL;

-- populate the data

UPDATE Job\_Application

SET response\_date = DATE\_ADD(submission\_date, INTERVAL FLOOR(RAND() \* 30) DAY)

WHERE status IN ('Accepted', 'Rejected');

--

SELECT

jp.job\_title,

e.company\_name,

AVG(DATEDIFF(ja.response\_date, ja.submission\_date)) AS avg\_response\_time

FROM Job\_Application ja

JOIN Job\_Posting jp ON ja.job\_posting\_id = jp.job\_posting\_id

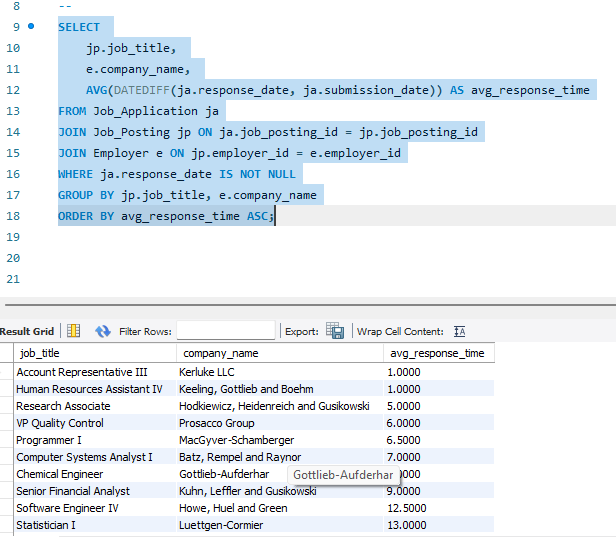
JOIN Employer e ON jp.employer\_id = e.employer\_id

WHERE ja.response\_date IS NOT NULL

GROUP BY jp.job\_title, e.company\_name

ORDER BY avg\_response\_time ASC;

***Code and Output screenshot:***



***Results:*** After randomly adding between 1 to 30 days to the submission date and putting it as a response date. The avg response time of kerluke LLC were the lowest. This helps job managers ensure that alumni receive timely feedback and allows employers to track their hiring efficiency. And alumni can use this avg response time to get an idea of when to expect a contact.

IX. Alumni Who Never Attended an Event

***Question:*** Which alumni have experienced the fastest career growth, moving into senior roles within the shortest time?

***Purpose:*** Identifies the fastest career advancements among alumni who achieved a "Senior" role. Use by employers to keep track on them and recruit them, if possible, also it Highlights graduates who may be ideal mentors.

***Code:***

SELECT

a.alumni\_id,

a.full\_name,

e.company\_name,

eh.job\_title AS latest\_job,

eh.start\_date AS latest\_start\_date,

a.graduation\_year,

FLOOR(DATEDIFF(eh.start\_date, CONCAT(a.graduation\_year, '-06-01')) / 365) AS years\_to\_senior\_role

FROM Employment\_History eh

JOIN Alumni a ON eh.alumni\_id = a.alumni\_id

JOIN Employer e ON eh.employer\_id = e.employer\_id

WHERE eh.start\_date = (

-- Subquery: Get the earliest start date for a senior-level role for each alumnus

SELECT MIN(eh2.start\_date)

FROM Employment\_History eh2

WHERE eh2.alumni\_id = eh.alumni\_id

AND eh2.job\_title LIKE '%Senior%'

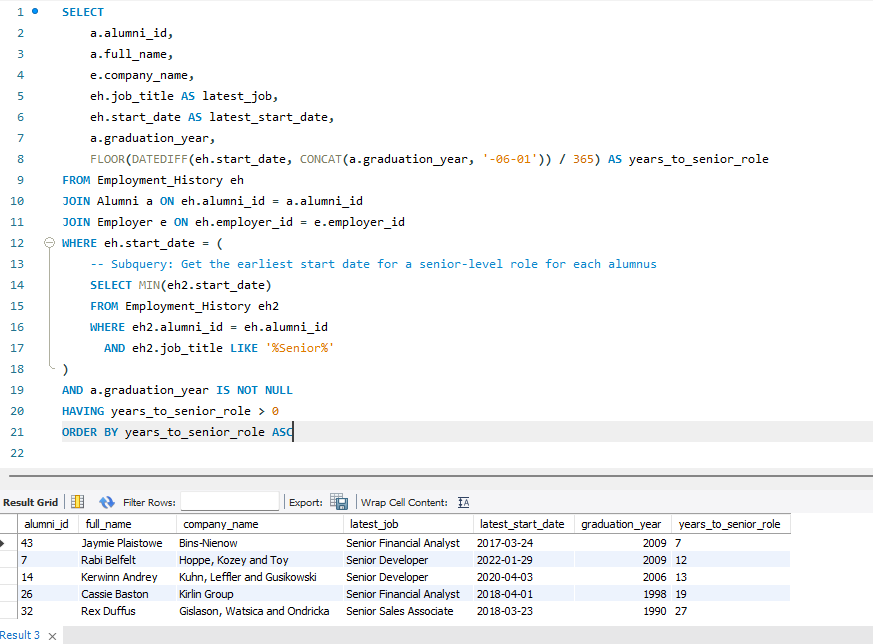
)

AND a.graduation\_year IS NOT NULL

HAVING years\_to\_senior\_role > 0

ORDER BY years\_to\_senior\_role ASC

***Code and Output screenshot:***



X. Alumni Who Applied for a Job and Later Worked There

***Question:*** How many alumni applied for a job and were later employed by the same company?

***Purpose:*** Measures job posting success and helps job recommendations for alumni. Helps administrators and career services evaluate the effectiveness of job postings by identifying alumni who successfully transitioned from applicants to employees within the same company. This provides insights into employer engagement and job posting success rates.

***Code:***

SELECT DISTINCT ja.alumni\_id, a.full\_name, jp.job\_title, e.company\_name,ja.submission\_date,eh.start\_date

FROM Job\_Application ja

JOIN Job\_Posting jp ON ja.job\_posting\_id = jp.job\_posting\_id

JOIN Employer e ON jp.employer\_id = e.employer\_id

JOIN Employment\_History eh ON ja.alumni\_id = eh.alumni\_id

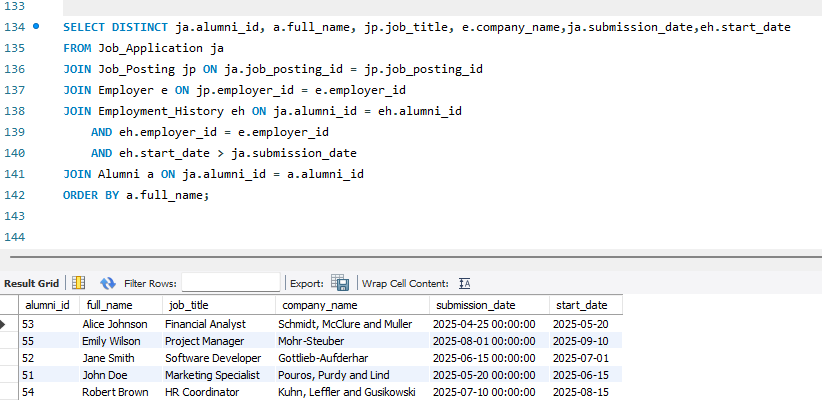
AND eh.employer\_id = e.employer\_id

AND eh.start\_date > ja.submission\_date

JOIN Alumni a ON ja.alumni\_id = a.alumni\_id

ORDER BY a.full\_name;

***Code and Output screenshot:***

******

***Results:*** This query shows a list of alumni who applied for a job at a company and later worked there. This query needed special attention to the data consistency because it involved multiple dependent relationships across different tables: alumni\_id had to be matched with 3 other tables then Job\_Application, Job\_Posting, Employment\_History ids also had to be matched in each other.

P.S: We added the data via INSERT at first to all 4 tables from row 51 and all were wrong. We have to say that we asked chatgpt to help us match these ids and create new values. And the way we used to populate and modify, INSERT and SET can be found in ‘csv to database.sql’.

P.S: This is from all of us, first of all thank you so much and I hope you will like our project. Sorry for the late submission and not showing in class regularly but we wanted to say that this course was really interesting especially this final project that helped us understand what are databases from scratch .

References:

1. Smith, R., & Lee, K. (2021). Alumni Info-Com Management with Distinct Classification of Data. ResearchGate.
2. Doe, J., Smith, A., & Brown, B. (2023). Bridging the Gap Between Universities and Alumni: A User-Centered Evaluation of a Digital Alumni Engagement Platform. ResearchGate.
3. Radhika, A., Mayraaj, S., Devisree, B., Surendra Sai, B., & Uma Ganesh, B. (2024). Alumni Management System. International Journal of Engineering Science and Advanced Technology (IJESAT).
4. IEEE Xplore. (2024). Design and Implementation of an Alumni Management System
5. Database Design Tutorial: How to Build a Relational Database. <https://www.youtube.com/watch?v=5RpUmDEsn1k&t=1s>
6. SQL Query Optimization Techniques. <https://www.youtube.com/watch?v=PPFyoXA_FC0>