Alumni Database Management

Team Members: Giridhar Reddy Jajapuram, Je Sai Kailash Pulipati

Abstract:

This project focuses on creating a robust alumni management system that integrates a well-structured database schema with a user-friendly Java application. Using Oracle JDBC with NetBeans, we developed functionalities to manage alumni details, including personal information, education, work experiences, and skills. The system also supports event management, allowing alumni to view available events, register for them, and track their participation. The database schema comprises seven tables: Student, StudentAdd, Education, WorkExperience, StudentSkills, Events, and EventParticipants, ensuring a scalable and efficient structure for storing and retrieving alumni-related data.

To align the system with practical needs, we consulted the Director of Alumni Relations to understand existing workflows and refine the schema. The application includes panels for creating, viewing, and deleting alumni records, as well as for managing event registrations. By integrating these features, the project enhances data accessibility and usability, fostering better engagement between alumni and the institution while streamlining administrative tasks.

Introduction:

Alumni play a crucial role in fostering the reputation and growth of an institution, acting as brand ambassadors and valuable contributors to the community. Managing alumni records efficiently is vital for maintaining strong institutional ties, tracking their accomplishments, and enhancing engagement through events and professional networks. This project aims to develop a comprehensive Alumni Management System that provides seamless management of alumni information while facilitating their active participation in institutional events.

The system is built on a relational database schema with seven interconnected tables, ensuring an efficient and scalable data structure to store alumni records, including personal details, education history, work experiences, skills, and event participation. The application is developed in Java, using Oracle JDBC for database connectivity, and features an intuitive user interface designed in NetBeans.

To tailor the solution to real-world requirements, the project involved a consultation with the Director of Alumni Relations to understand the operational workflows and database structure currently in use. This interaction provided critical insights that shaped the database design and ensured that the application meets the practical needs of stakeholders.

This report outlines the design, development, and functionality of the Alumni Management System, emphasizing its role in enhancing alumni engagement, streamlining administrative processes, and fostering a stronger connection between the alumni and the institution.

System Requirements:

1. Hardware Requirements
   - Processor: Intel Core i5 or equivalent (minimum), Intel Core i7 or higher (recommended)
   - RAM: 8 GB (minimum), 16 GB or higher (recommended)
   - Storage: 500 GB HDD (minimum), 256 GB SSD or higher (recommended)
   - Display: 1366 x 768 resolution (minimum), Full HD 1920 x 1080 (recommended)
   - Peripherals: Keyboard, mouse, and optional printer for report generation

2. Software Requirements
   - Operating System: Windows 10/11, macOS, or Linux (Ubuntu 20.04 or later)
   - Database: MySQL 8.0 Community Edition (local database)
   - Development Environment:
     - Java Development Kit (JDK): JDK 11 or higher
     - IDE: NetBeans IDE 12 or later
     - JDBC Driver: Oracle JDBC Thin Driver (ojdbc8.jar or compatible)
   - Additional Tools:
     - Apache Maven for dependency management (optional)
     - Git for version control (optional)

3. Dependencies
   - Java Libraries:
     - Java Swing for user interface design
     - Oracle JDBC Thin Driver for database connectivity
     - javax.swing.table.DefaultTableModel for table manipulation
   - Database Design:
     - Local MySQL database for storing and managing alumni records and event data

4. Networking Requirements
   - Local Environment:
     - MySQL database hosted locally on the development machine
   - Optional Configuration:
     - For network-based deployment, ensure proper port configuration (default MySQL port: 3306)
     - A stable internet connection if hosting the database on a cloud server

5. User Prerequisites

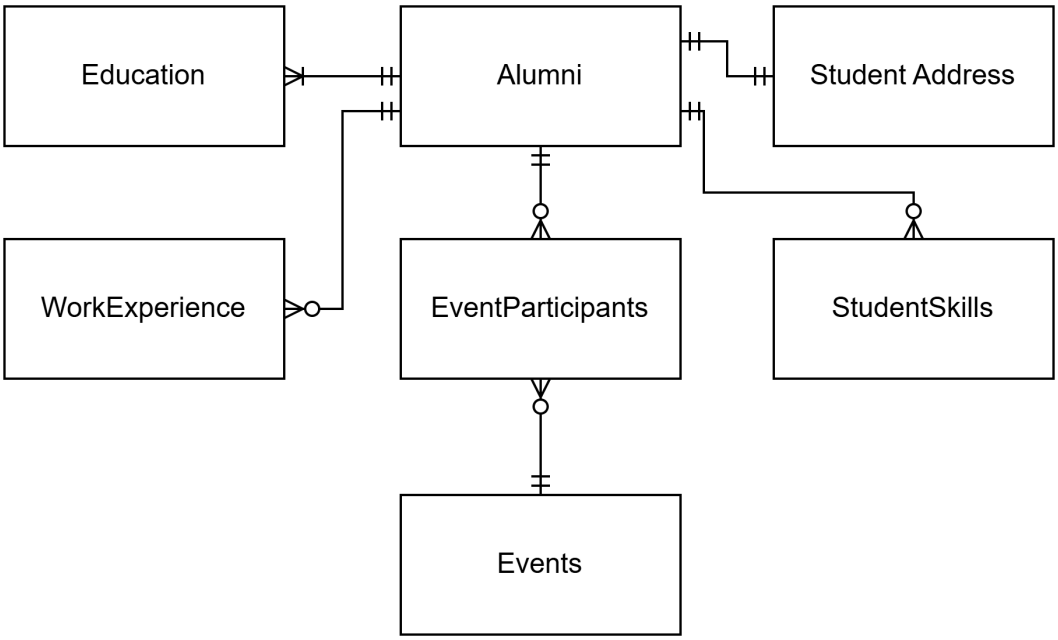
- Basic knowledge of Java and database management systems
- Familiarity with NetBeans IDE, MySQL tools and Oracle Database tools
- Access to the system's administrative privileges to set up the database and Java environment

This configuration ensures optimal performance for the Alumni Management System, supporting efficient record management, event registration, and robust data handling capabilities.

System Design:

ER Diagram:

Below is the ER diagram for the Alumni database management system:



The following explanation details the significance of each entity and relationship within the system:

Entities and Attributes

1. Alumni:
    - Represents the primary entity, storing details of alumni.
    - Key Attributes: StudentID (Primary Key), Name, ContactNo, PrimaryEmail, Gender, DateOfBirth, Nationality.

- Alumni act as the central entity connecting to education, work experience, address, skills, and event participation.

2. Education:
    - Tracks the educational background of alumni.
    - Key Attributes: EducationID (Primary Key), StudentID (Foreign Key), Institution, Degree, Specialization, StartingYear, PassingYear, GPA.
    - Relationship with Alumni: One alumni can have multiple education records (1:N).

3. WorkExperience:
    - Captures professional experience of alumni.
    - Key Attributes: WorkID (Primary Key), StudentID (Foreign Key), Company, Title, StartDate, EndDate, PlaceOfWork.
    - Relationship with Alumni: One alumni can have zero or many work experiences (0..N).

4. Student Address:
    - Stores the residential address of alumni.
    - Key Attributes: StudentID (Primary Key, Foreign Key), Add1, Add2, City, State, Postal_Code, Country.
    - Relationship with Alumni: One alumni may have zero or one address (0..1).

5. StudentSkills:
    - Represents the skills possessed by alumni.
    - Key Attributes: StudentID (Foreign Key), SkillName.
    - Relationship with Alumni: One alumni can have zero or many skills (0..N).

6. Events:
    - Represents the events organized for alumni.
    - Key Attributes: EventID (Primary Key), EventName, EventDate, Location.
    - Relationship with EventParticipants: Each event must have at least one participant (1:N).

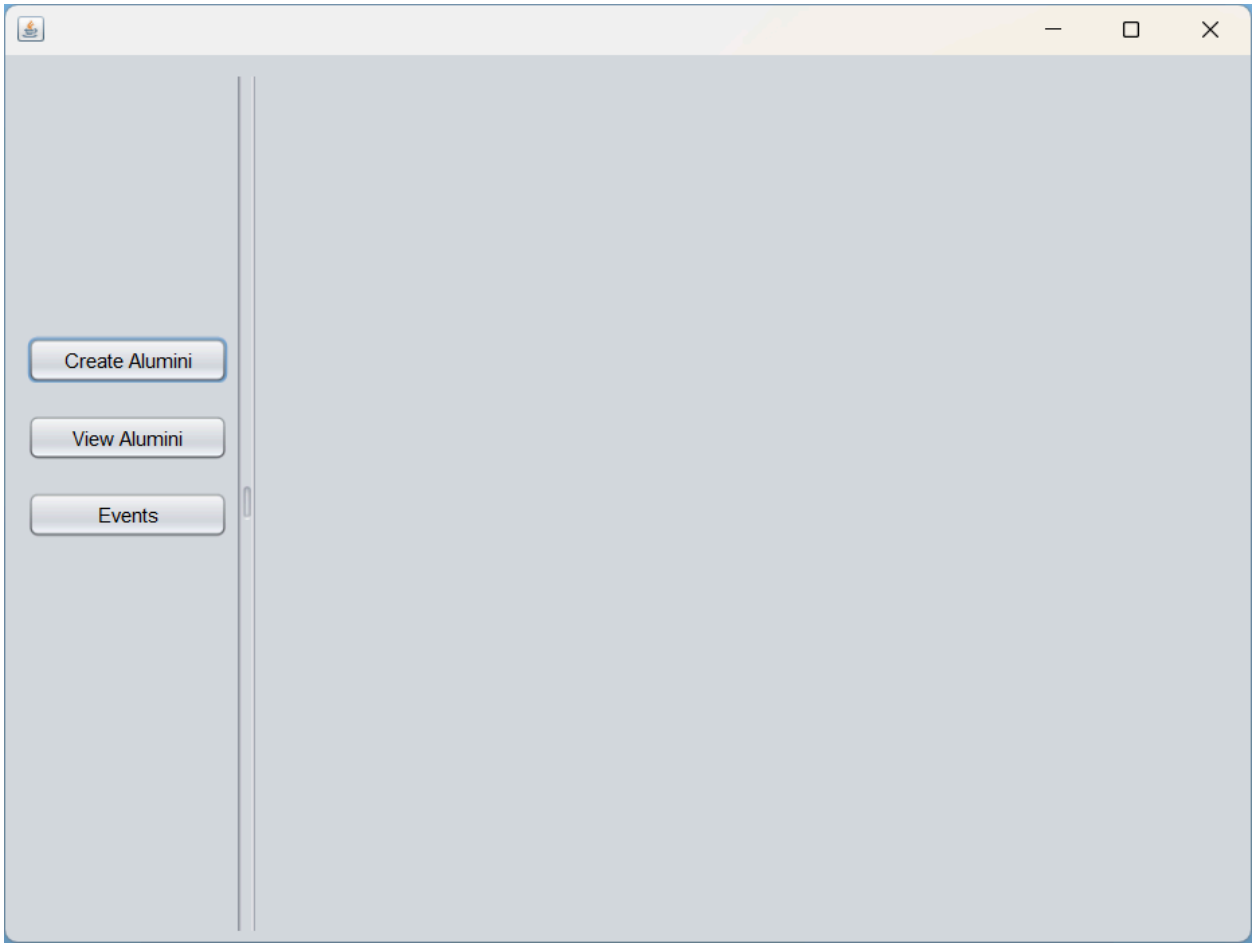
7. EventParticipants:
    - Acts as a bridge entity connecting alumni and events.
    - Key Attributes: EventID (Foreign Key), StudentID (Foreign Key).
    - Relationship with Alumni: One alumni can participate in zero or many events (0..N).
    - Relationship with Events: One event must have at least one or many participants (1:N).

Relationships:

- Alumni ↔ Education (1:N): Each alumni can have multiple education records, such as undergraduate, postgraduate, or certifications, allowing a detailed academic profile.
- Alumni ↔ WorkExperience (0..N): Alumni may have multiple professional experiences or none, depending on their career stage.
- Alumni ↔ Student Address (0..1): Address is optional, accommodating alumni who may not want to provide residential details.
- Alumni ↔ StudentSkills (0..N): Alumni can have a diverse set of skills, which can be used to map expertise within the database.
- Alumni ↔ EventParticipants (0..N): Alumni can register for multiple events or choose not to participate at all.
- EventParticipants ↔ Events (1:N): Every event must have at least one participant, ensuring events are meaningful and attended.

Implementation:

UI Design:
MainJFrame



The MainJFrame serves as the central frame or window of the application, managing the navigation and interactions between the different functional panels.

Its Main components are:

jSplitPane1:

- A split pane divides the frame into two sections:
- Left Panel (leftjPanel): Contains navigation buttons.
- Right Panel (userProcessContainer): Displays the selected panel (e.g., CreateAluminiJPanel, ViewJPanel, RegistrationJPanel).

Buttons on the Left Panel:

- btnCreate: Navigates to the "Create Alumni" panel for adding alumni records.
- btnView: Navigates to the "View Alumni" panel to display and manage existing records.
- btnCreateEvent: Navigates to the "Events" registration panel for managing alumni events.

userProcessContainer:

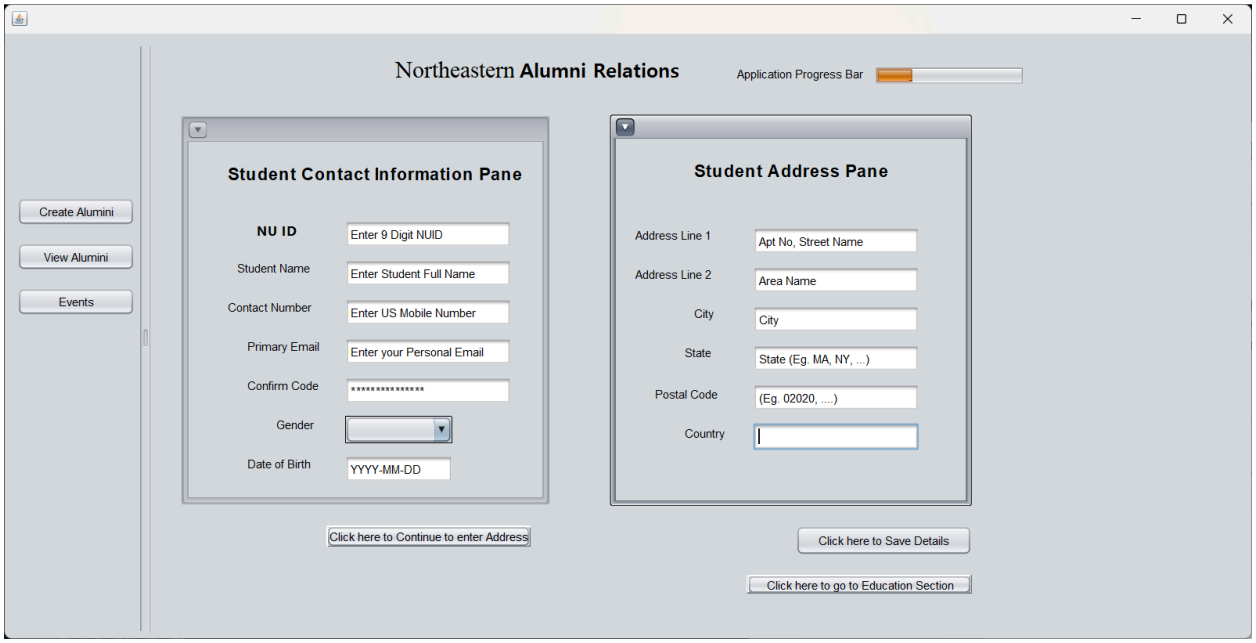
- Uses a CardLayout to dynamically switch between panels based on user interaction.

How the Flow Works:
Each button click triggers an ActionListener that creates an instance of the corresponding panel (CreateAluminiJPanel, ViewJPanel, or RegistrationJPanel).

The MainJFrame establishes the overall flow, ensuring users can:

- Add new alumni records.
- View and manage existing alumni data.
- Register and handle alumni events.

Create Alumni Panel:



The CreateAlumniJPanel class is a Swing-based Java UI component designed for creating and managing alumni data. This class is part of a multi-panel alumni management application. It provides an interface for users to input and save alumni details, including personal information and address. It also includes navigation features for moving to other sections, such as the education panel.

This Panel Incorporates two internal frames (jInternalFrame2 for personal details and InternalFrame1 for address input) to organize inputs. Employs a JProgressBar to indicate application progress visually.

Input Fields:
Contains multiple fields for alumni data entry:

- Personal Details: NUID, Name, Contact Number, Email, Gender, Date of Birth, and a confirmation code.
- Address Details: Address Line 1, Address Line 2, City, State, Postal Code, and Country.
- Dropdown (JComboBox) for gender selection with predefined options.
- Password field (JPasswordField) for secure input.

Key Components in this panel are Personal Details Pane and Address Details Pane
Personal Details Pane

- Positioned within a scrollable pane.
- Organized with labels and fields for essential alumni details.

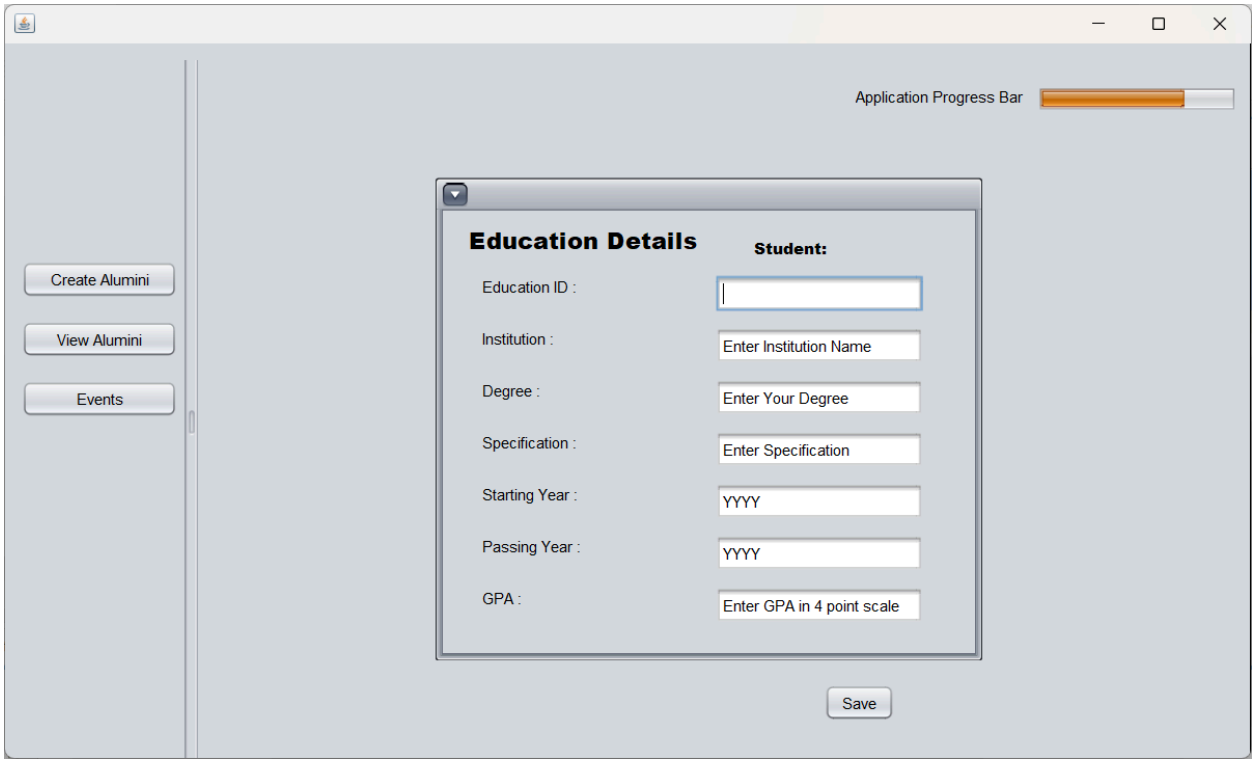
Address Pane

- Initially hidden until the user clicks btnStudentAddPane.
- Includes fields and labels for entering address details.

How the User can input data:

- Users start by filling out personal details.
- Clicking "Continue to enter Address" reveals jInternalFrame1 and updates the progress bar to 25%.
- Clicking "Save Details" commits the entered data, potentially connecting to a database.
- Then the student can go to the education input panel to input his education details

Education Details Panel:



The EducationJPanel class is a part of the ui.Education package and is responsible for displaying and collecting education-related information of a student.

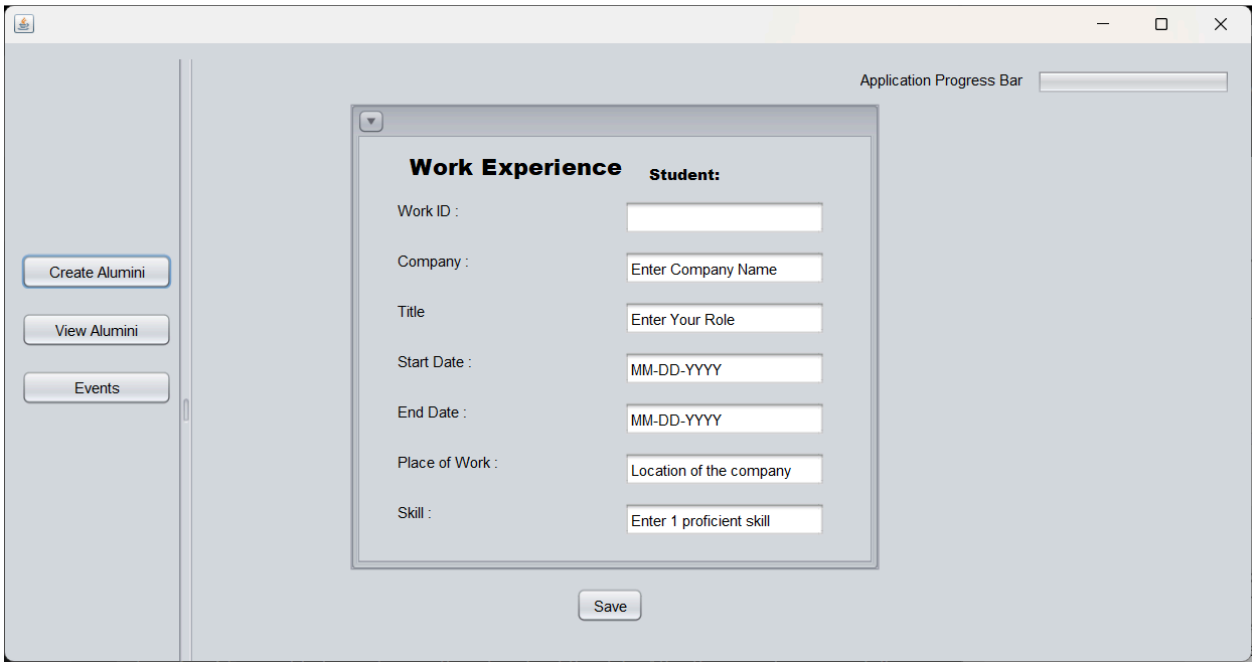
Input Fields:

- Education ID: A unique identifier for each education entry.
- Institute Name: The name of the educational institution attended by the student.
- Degree: The degree or certification the student obtained (e.g., Bachelor's, Master's).

- Field of Study: The academic discipline or specialization (e.g., Computer Science, Engineering).
- Start Date: The date when the student started their education at the institution.
- End Date: The date when the student completed their education.
- GPA: The Grade Point Average or equivalent academic score.

The Add Education Record feature allows users to input and save a new educational qualification by filling in various fields such as the institute name, degree, field of study, start and end dates, and GPA. Once the information is entered, the **Database Validation** ensures that no duplicate records for the same `education_id` are added, maintaining the integrity of the data. After the record is successfully saved, the system provides **User Feedback** in the form of confirmation messages or error alerts, notifying users of the success or failure of their action, ensuring they are always informed about the status of their input.

Work Experience Details Panel



The WorkExperienceJPanel is a Swing-based Java component designed for managing and recording the work experience of students in a user-friendly GUI. It connects to a MySQL database to store and retrieve data. This panel is a part of a larger application and provides the functionality to input, save, and manage work experience details for individual students.

Form Input Fields:

- Work ID (txtWorkID): A unique identifier for each work experience entry.
- Company Name (txtCompany): The name of the company where the student worked.

- Title (txtTitle): The role or position held by the student.
- Start Date (txtStartDate): The starting date of the work experience.
- End Date (txtEndDate): The ending date of the work experience.
- Place of Work (txtPlaceOfWork): Location of the workplace.
- Skill (txtSkill): A single skill the student gained during the experience.

Database Integration:

- Saves work experience details to the Work_Exp table.
- Saves a related skill to the skill table.
- Uses PreparedStatement for SQL queries to prevent SQL injection.

Progress Bar:

- Indicates the completion status of the work experience data entry process.
- Automatically updates its value after saving the record.

Functional Flow:

- WorkExperienceJPanel accepts the userProcessContainer (parent panel) and a student ID (sid) as parameters.
- Displays the student ID dynamically using the lblStudID label.
- Input fields provide default text that clears when clicked.
- Date fields accept dates in MM-DD-YYYY format.
- SQL Queries inserts the work experience details into the Work_Exp table.
- Adds the associated skill into the skill table.

Database Connection:

```
Connection conn = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/Admin_Database?zeroDateTimeBehavior=CONV
ERT_TO_NULL",
    "root", "password");
```

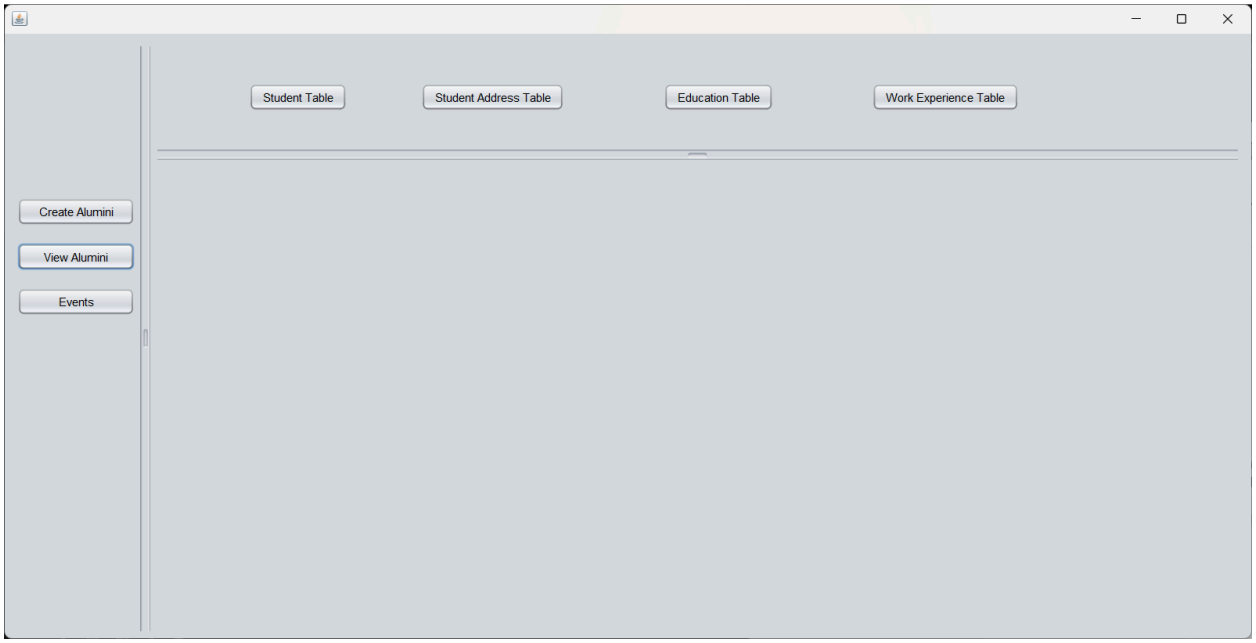
SQL Insert Statements:

```
String sql = "INSERT INTO Work_Exp(workid, company, title,
start_date, end_date, work_place) VALUES (?, ?, ?, ?, ?, ?)";
String sql2 = "INSERT INTO skill(sid, skill) VALUES (?, ?)";
```

Prepared Statement:

```
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, wid);
pstmt.setString(2, cname);
pstmt.setString(3, title);
pstmt.setString(4, strt_date);
pstmt.setString(5, end_date);
pstmt.setString(6, work_place);
```

View Student Details Panel:



The `ViewJPanel` class is a panel used to display different tables of data related to students. It features a navigation layout where users can switch between various data views (student table, address table, education table, and work experience table). The panel utilizes a `JSplitPane`, splitting the screen into two sections: a left section with buttons for each data view, and a right section that dynamically displays the corresponding table views based on user interaction.

Buttons for Data Views: The left section contains four buttons: "Student Table," "Student Address Table," "Education Table," and "Work Experience Table." Each button, when clicked, triggers the display of a corresponding table in the right section.

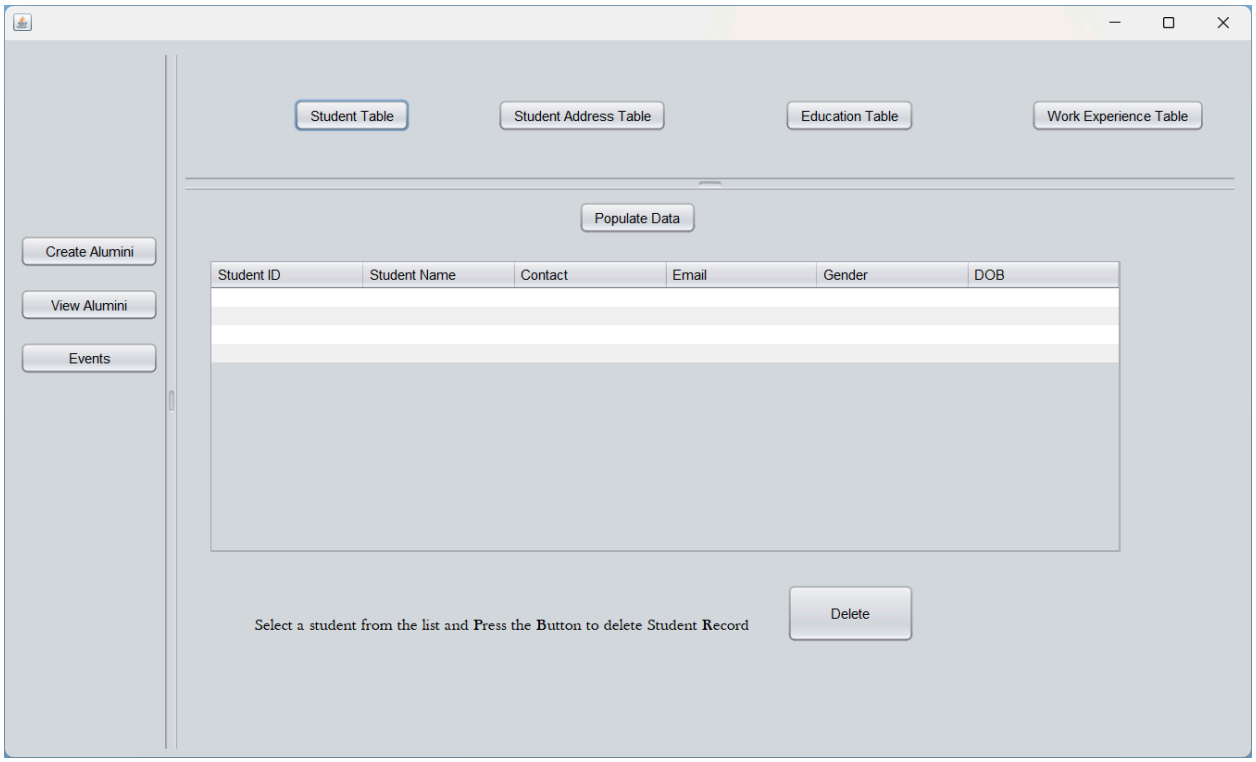
Dynamic Table Display: When a user clicks on any of the buttons, a new panel related to the selected data (e.g., `StudTableJPanel`, `StudAddressJPanel`, `EducationJPanel`,

`WorkTableJPanel`) is added to the `container` panel, which uses a `CardLayout` to allow smooth transitions between different views.

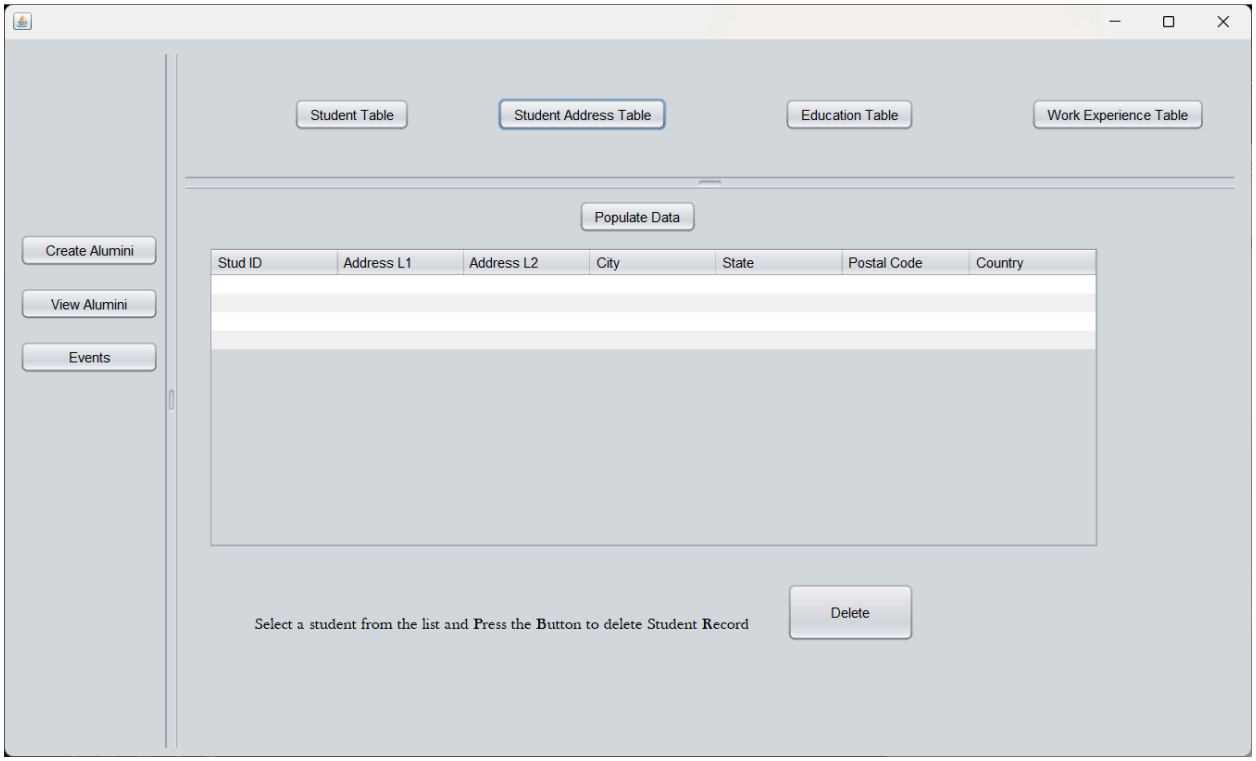
CardLayout: The `CardLayout` is used to manage the dynamic switching between different panels in the container. This layout makes it easy to navigate through various data views without reloading the entire panel.

The overall structure is designed to keep the interface simple and organized, allowing users to view and navigate between different tables related to student data.

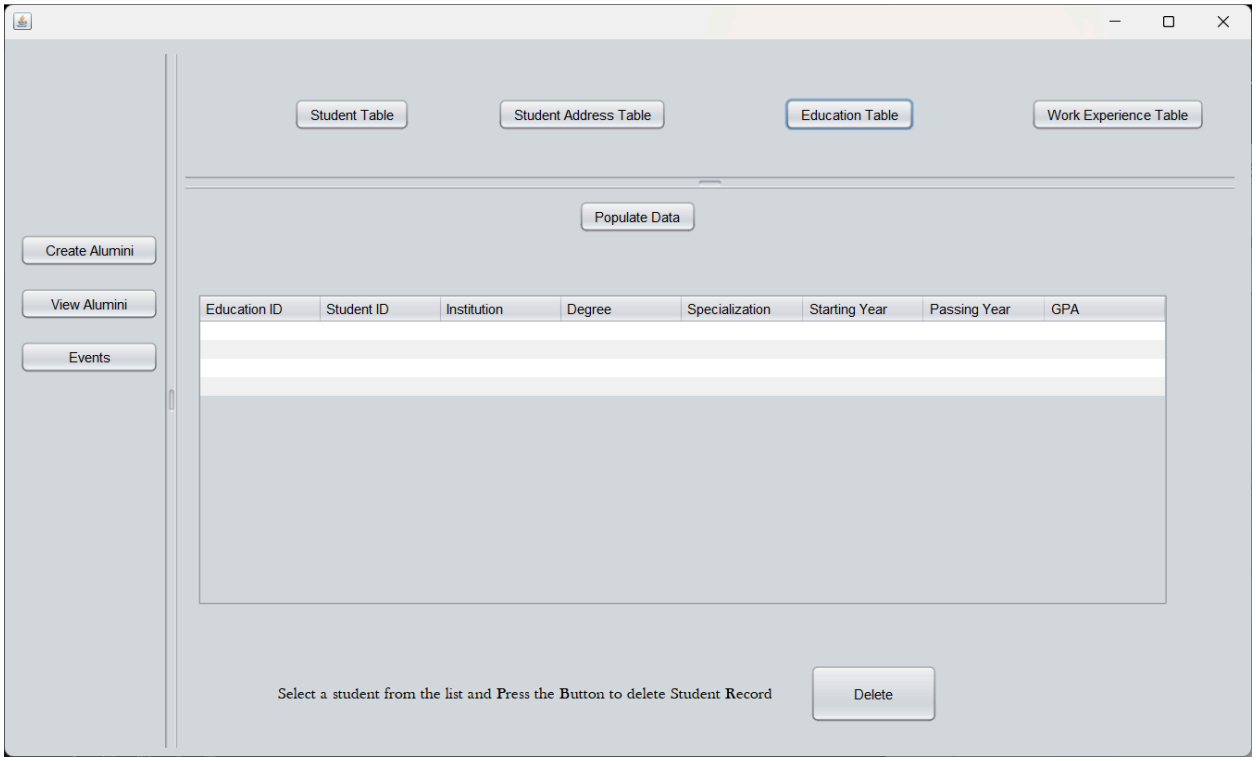
View Student Table:



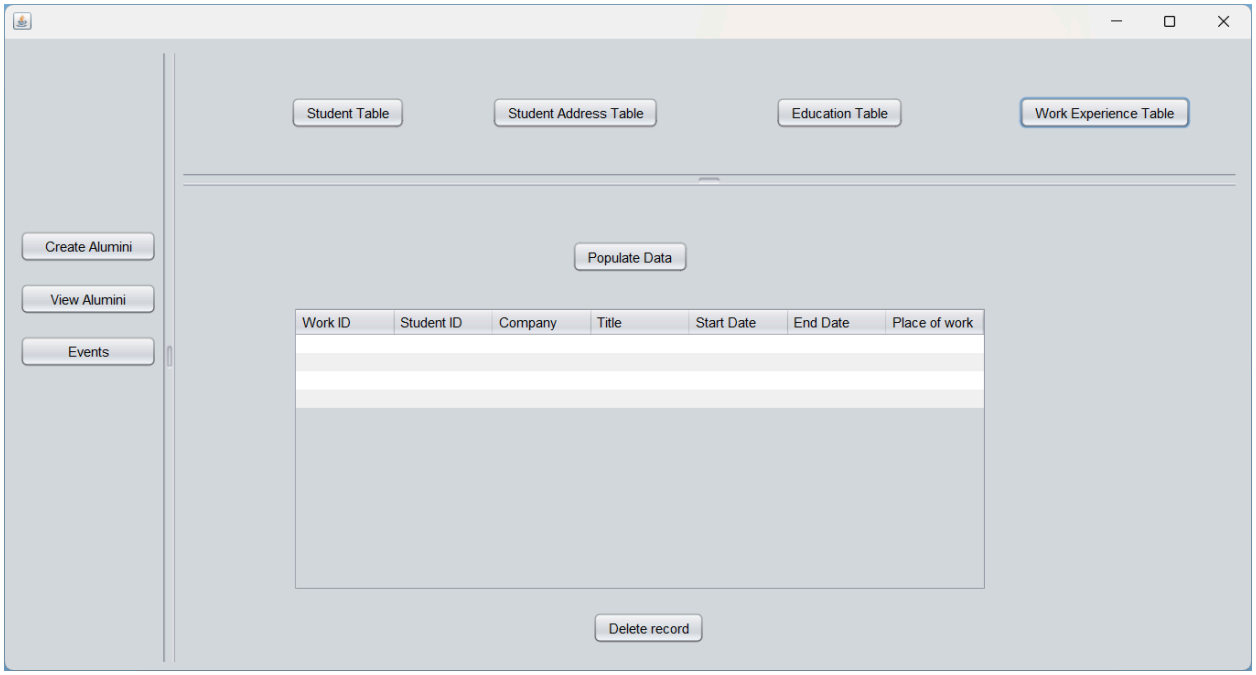
View Student Address Table:



View Education Table:

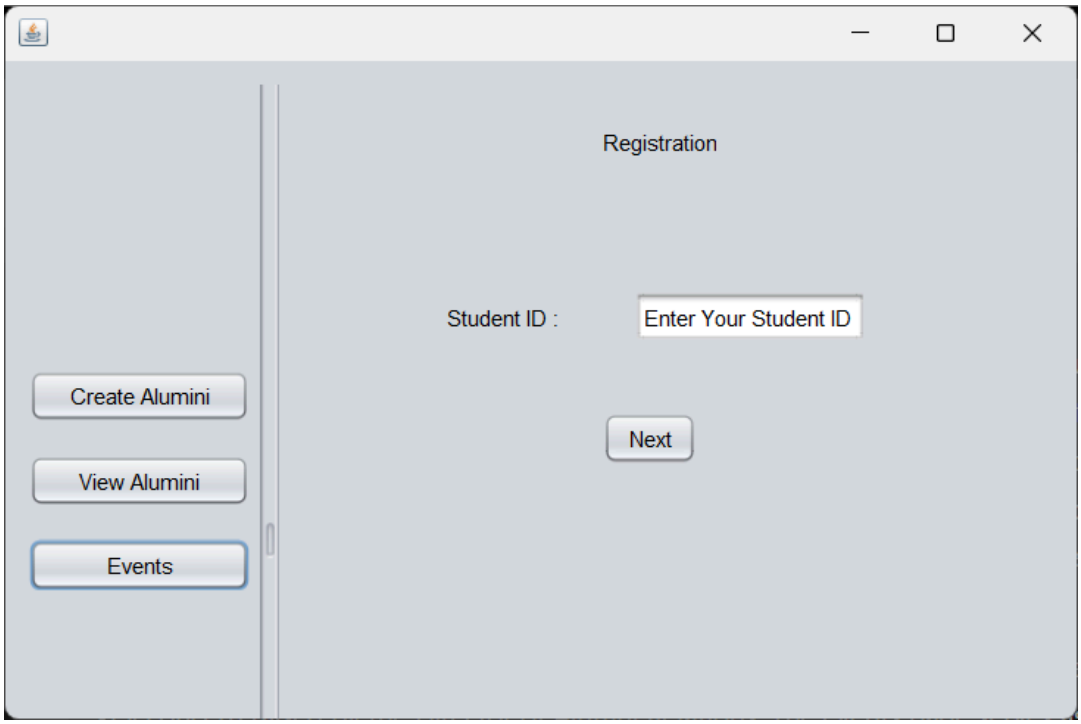


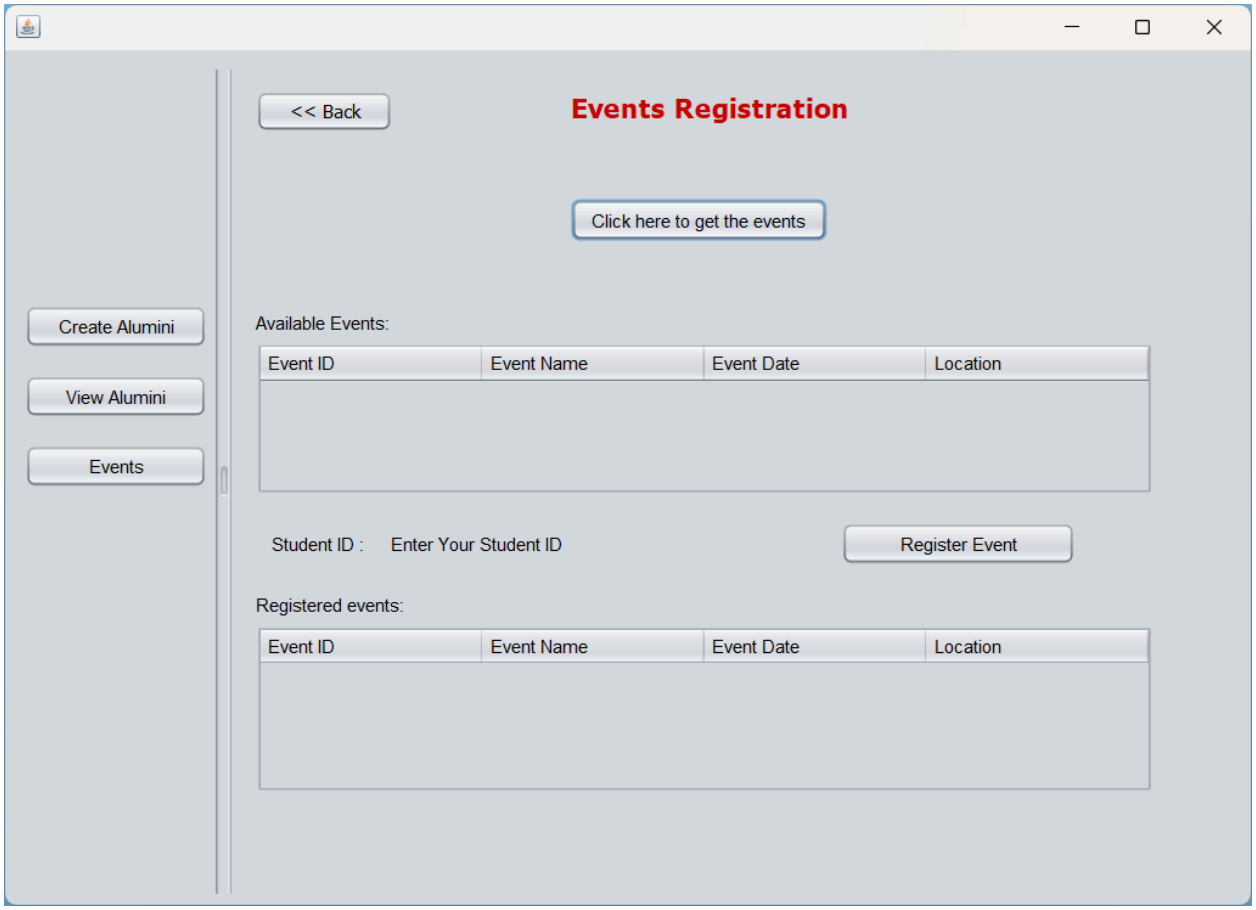
View Work Experience Table:



This overall structure is designed to keep the interface simple and organized, allowing users to view and navigate between different tables related to student data.

Events Creation Panel:





In response to a valuable suggestion from the **Director of Alumni Relations**, we have developed and integrated an innovative events page into our alumni database project. Initially, the idea of creating such a feature was not part of the original project scope. However, after a detailed conversation with the Director, we recognized the need for a platform that would not only help alumni stay connected to the university but also provide them with an avenue to contribute to the university community in a more meaningful way.

The events page allows alumni to log in securely using their student ID, ensuring that they have personalized access to events hosted by the university. Once logged in, alumni can view a list of current and upcoming events, including conferences, workshops, networking sessions, and other university activities. Alumni can easily register to attend these events, making it simpler for them to stay engaged with the university's initiatives and stay connected with their peers and faculty.

Beyond attending events, the page introduces a powerful feature that enables alumni to host events for current students. This allows alumni to share their expertise, insights, and professional experiences by organizing events such as information sessions, career development workshops, hackathons, and networking opportunities. These events not only benefit current students but also provide alumni with an opportunity to give back to the university in a way that strengthens the connection between them and the next generation of students.

The inclusion of this feature aims to streamline the process of managing alumni event registrations, a key concern raised by the Director of Alumni Relations. The events page serves as an efficient tool for the alumni team, making it easier to track alumni participation, organize events, and communicate with alumni. It also enhances the overall experience for alumni by offering them a user-friendly platform to contribute and stay involved in the university's activities.

In summary, this events page not only addresses the immediate need expressed by the Director but also creates a sustainable framework for fostering lasting relationships between alumni and the university. By facilitating alumni engagement in a more structured and meaningful way, this feature helps bridge the gap between the university's current students and its alumni network, ensuring the continued success and growth of both communities.

About the code:
The code consists of two methods: btnRegisterActionPerformed and btnDBActionPerformed. Each method is part of an event-driven Java application that manages alumni event registration.

The provided code consists of two methods: btnRegisterActionPerformed and btnDBActionPerformed. Each method is part of an event-driven Java application that manages alumni event registration.

btnRegisterActionPerformed:
This method handles the registration process for an alumni event. When the user attempts to register for an event, the method first checks if an event has been selected from the event list (tblEventList). If no event is selected, an error message is displayed to the user.

Once an event is selected, the method retrieves the EventID from the chosen row in the tblEventList table. It then checks whether the user has already registered for the event by searching the tblRegList table for the same EventID. If the user is already registered, a warning message is shown. If the user isn't registered yet, the event details (event name, date, and location) are added to the tblRegList table, which tracks the registered events.

Next, the method connects to the database and inserts a new record into the EventParticipants table, associating the selected EventID with the logged-in student's ID. After the registration is successfully added, the user is informed with a success message. If any errors occur during the process (such as issues with database connectivity or query execution), an error message is displayed.

The provided code consists of two methods: btnRegisterActionPerformed and btnDBActionPerformed. Each method is part of an event-driven Java application that manages alumni event registration.

btnRegisterActionPerformed
This method handles the registration process for an alumni event. When the user attempts to register for an event, the method first checks if an event has been selected from the event list (tblEventList). If no event is selected, an error message is displayed to the user.

Once an event is selected, the method retrieves the EventID from the chosen row in the tblEventList table. It then checks whether the user has already registered for the event by searching the tblRegList table for the same EventID. If the user is already registered, a warning message is shown. If the user isn't registered yet, the event details (event name, date, and location) are added to the tblRegList table, which tracks the registered events.

Next, the method connects to the database and inserts a new record into the EventParticipants table, associating the selected EventID with the logged-in student's ID. After the registration is successfully added, the user is informed with a success message. If any errors occur during the process (such as issues with database connectivity or query execution), an error message is displayed.

btnDBActionPerformed:
This method handles the database interaction and populates the event tables with data from the database. First, the method loads the JDBC driver and establishes a connection to the database. It then executes a SQL query to retrieve all records from the Events table. These records, which contain details about available events, are displayed in the tblEventList table. The table is first cleared of any existing rows to ensure it only shows the most up-to-date event data.

Next, the method queries the EventParticipants table to retrieve the events that the logged-in student is registered for. This information is fetched using an SQL INNER JOIN between the Events and EventParticipants tables, filtering by the student's ID. The resulting data is displayed in the tblRegList table. Similar to the previous part, the table is cleared before populating it with the student's registered events.

Once the data is successfully retrieved and displayed, the database connection is closed, and a success message is logged to the console. If any errors occur during this process, an error message is logged.

In summary, the btnRegisterActionPerformed method handles the user registration for events and updates both the UI and the database, while the btnDBActionPerformed method loads event data from the database and displays it in the event tables.

The Interaction Session with the Northeastern Alumni Board Director

The primary objective of the alumni database is to foster ongoing engagement between the university and its alumni, support fundraising initiatives, provide career services, and streamline event tracking. By maintaining a comprehensive and easily accessible alumni record, the university can facilitate meaningful connections and opportunities for both alumni and current students. Regarding datasets, the alumni team does not have private data readily available for sharing. However, public datasets related to alumni engagement and event participation are accessible, which can be utilized for the project. These datasets can be used to model engagement patterns, event participation, and similar aspects that align with the goals of the alumni database.

Current Workflow and Processes:
Currently, the process of managing alumni records involves manually updating and tracking alumni information, organizing events, and collecting donations. The alumni team is heavily reliant on various ad-hoc methods for managing these tasks, which include spreadsheets and independent software tools. The process for managing alumni data typically includes collecting and storing basic details such as contact information and graduation year. This data is used to plan and execute various engagement initiatives, such as events, reunions, and career support programs. The primary challenge with the current system is the inconsistency of alumni data—missing or outdated contact information, and a lack of automated processes to update and maintain records, creating a strain on the team and stakeholders.

Data Collection and Management:
The key details collected about alumni include their name, graduation year, degree, contact information, current job, and location. While most of this data is collected at the time of graduation, the completeness and accuracy of the data can be an issue. Missing or outdated information is a common challenge, and the current system has no automated means of keeping alumni details up to date. Data is updated through alumni self-reports, manual data entry, or integrations with external platforms like LinkedIn. When data is incomplete, the alumni team uses forms and events to collect new information, encouraging alumni to update their records during registration for events or through periodic emails.

Regarding automation, the team is open to incorporating automated processes for updating data and for streamlining event registration. The alumni database currently tracks essential data such as contact details, event participation, and donations. This data is used to support various

operations, including event management, sending newsletters, and tracking donations. The analysis of this data primarily focuses on organizing events and assessing alumni engagement through participation metrics. Reporting includes standard attributes such as event dates, alumni attendance, and donation amounts, although managing duplicate records and incomplete data remains a significant challenge.

Entities and Relationships:
The key entities involved in the alumni database include alumni, events, donations, and communication records. These entities are interconnected, with alumni attending events, donating to campaigns, and interacting with communications from the university. Managing these relationships effectively is crucial to ensuring that the database serves its purpose in engaging alumni and supporting university initiatives.

System Requirements:
The automated system should include functionalities such as alumni search, event scheduling, report generation, and email notifications. It is important to differentiate user roles—admin versus standard user access—so that the database can be used securely and effectively. Search and filter functionalities should allow users to query alumni data based on various attributes, such as graduation year, location, and career field. The system should also track alumni interaction history, including participation in events, donations, and communications, which would provide valuable insights into engagement levels. Event tracking should include attributes like event name, date, location, and participating alumni, while donation tracking should record donation amounts, dates, and campaigns.

Reports generated by the system should include metrics such as alumni engagement by region, total donations by campaign, and participation in specific events. These reports would assist the alumni team in refining their engagement strategies and understanding trends in alumni involvement.

Data Privacy and Security:
Given the sensitive nature of alumni data, there are significant legal and privacy considerations to account for. The alumni database must comply with data protection regulations, and protocols for ensuring data privacy and security must be strictly adhered to. Only authorized personnel, such as alumni relations staff, should have access to the database, and user roles and access levels will need to be clearly defined to protect sensitive data. Backup and recovery mechanisms should be in place to ensure that the data is securely stored and can be recovered in case of any system failures.

Future Needs and Integration:
The alumni database system should be designed with future expansion and integration in mind. It should be able to integrate with existing tools or platforms, such as payment systems, mailing platforms, and university databases, to ensure seamless workflows. As the needs of the alumni relations team evolve, the system should be adaptable and able to accommodate new features, such as advanced analytics or a more comprehensive event management module. Improving user experience and utility is key, and areas for future enhancement could include better reporting tools, more intuitive user interfaces, and improved data integration capabilities.

Collaboration with other departments is essential to optimizing the use of alumni data. By working closely with departments like career services, admissions, and fundraising, the alumni relations team can leverage the database to drive broader university initiatives and provide better services to alumni. The database will be a valuable tool for cross-departmental collaboration and for strengthening the connection between alumni and the university community.

SQL

Imagine you're managing a university's student data. You need tables to store different types of information about the students, such as their basic details, addresses, education, work experience, skills, and events they attend. This is exactly what these tables do. Each table is linked by a unique student ID, which makes it easy to combine data from multiple tables. The structure ensures that you can efficiently store, update, and retrieve information about students and their involvement in university activities, without creating duplicate or inconsistent records. The foreign keys act like connections that allow you to match records from different tables together, ensuring your database is organized and meaningful.

SQL code defines several tables for managing student data, including their personal information, educational background, work experience, skills, and event participation. Each table is structured with specific attributes to ensure efficient data storage and relationships.

1. Student Table
This table stores the basic information of each student. It includes the following columns:

- sid: A unique identifier for each student (primary key).
- sname: The student's name.
- contact: The student's contact number.
- semail: The student's email address, which must be unique.
- sgender: The student's gender, stored as either 'Male' or 'Female'.
- sdob: The student's date of birth.

The Student table serves as the central table for storing essential student data and provides references to other tables.

2. StudentAdd Table
This table contains the student's address details, linked to the Student table by the student's unique identifier (sid). The columns are:

- sid: The student ID (foreign key, referencing Student(sid)).
- Add1: The first line of the student's address.
- Add2: An optional second line of the address.
- City: The city where the student lives.
- State: The state of residence.
- Postal_Code: The postal or ZIP code.
- Country: The country where the student resides.

This table uses a foreign key constraint (FK_StudentID_Addr) to ensure that each address is associated with a valid student.

3. Education Table
The Education table stores information about the student's educational history. It is linked to the Student table through the student's ID (StudentID), which is a foreign key. The columns include:

- EducationID: A unique identifier for each educational record (primary key).
- StudentID: A reference to the student (sid in the Student table).
- Institution: The name of the institution where the student studied.
- Degree: The degree obtained (e.g., Bachelor's, Master's).
- Specialization: The student's major or specialization (optional).
- StartingYear: The year the education began.
- PassingYear: The year the education ended (optional).
- GPA: The student's Grade Point Average (optional).

The foreign key constraint (FK_Edu_StudentID) links each educational record to a student in the Student table.

4. WorkExperience Table
This table stores details about the student's work experience. The columns are:

- WorkID: A unique identifier for each work experience record (primary key).
- StudentID: A reference to the student (sid in the Student table).
- Company: The name of the company where the student worked.
- Title: The job title or role the student held.

- StartDate: The date the student started working at the company.
- EndDate: The date the student ended their employment (optional).
- PlaceOfWork: The location of the workplace.

This table also has a foreign key constraint (FK_Student_WorkExperience) linking each work experience record to a specific student.

5. StudentSkills Table
The StudentSkills table stores the skills associated with each student. It uses a composite primary key consisting of the StudentID and SkillName:

- StudentID: The student's unique identifier (foreign key referencing Student(sid)).
- SkillName: The name of the skill (e.g., programming languages, tools, etc.).

This table allows a many-to-many relationship between students and their skills. A student can have multiple skills, and each skill can belong to multiple students.

6. Events Table
The Events table stores information about events that are organized for students. The columns are:

- EventID: A unique identifier for each event (primary key).
- EventName: The name of the event (e.g., seminar, workshop).
- EventDate: The date the event is scheduled.
- Location: The location where the event is held.

This table keeps track of all available events, and it will be referenced by the EventParticipants table to track student participation.

7. EventParticipants Table
The EventParticipants table stores the students who have registered for each event. It links students to events using the following columns:

- EventID: The ID of the event (foreign key referencing Events(EventID)).
- StudentID: The ID of the student (foreign key referencing Student(sid)).

The combination of EventID and StudentID forms the primary key for this table, ensuring that a student can only register once for a specific event. This table allows for tracking which students are attending which events.

SQL Code Snippet:

```
CREATE TABLE Student (
    sid INT PRIMARY KEY,
    sname VARCHAR(100),
    contact VARCHAR(15),
    semail VARCHAR(255) UNIQUE,
    sgender ENUM('Male', 'Female'),
    sdob DATE
);

CREATE TABLE StudentAdd (
     sid INT PRIMARY KEY,                    -- Foreign key and primary
key
     Add1 VARCHAR(255) NOT NULL,                 -- First line of the
address
     Add2 VARCHAR(255),                         -- Second line of the
address (optional)
    City VARCHAR(100) NOT NULL,                -- City
    State VARCHAR(100) NOT NULL,               -- State
    Postal_Code VARCHAR(20) NOT NULL,          -- Postal or ZIP code
    Country VARCHAR(100) NOT NULL,             -- Country
        CONSTRAINT FK_StudentID_Addr FOREIGN KEY (sid) REFERENCES
Student(sid)
);



CREATE TABLE Education (
    EducationID INT AUTO_INCREMENT PRIMARY KEY,  -- Unique identifier
for each education entry
     StudentID INT NOT NULL,                          -- Foreign key
referencing Student table
     Institution VARCHAR(255) NOT NULL,              -- Name of the
institution
     Degree VARCHAR(100) NOT NULL,                 -- Degree obtained
(e.g., Bachelors, Masters)
     Specialization VARCHAR(100),                 -- Specialization or
major (optional)
     StartingYear YEAR NOT NULL,                    -- Year education
started
     PassingYear YEAR,                               -- Year education
completed (optional)
```

```
    GPA DECIMAL(4, 2),                                  -- Grade Point
Average (optional)
     CONSTRAINT FK_Edu_StudentID FOREIGN KEY (StudentID) REFERENCES
Student(sid)
);

CREATE TABLE WorkExperience (
    WorkID INT AUTO_INCREMENT PRIMARY KEY,      -- Unique identifier
for each work experience
     StudentID INT NOT NULL,                            -- Foreign key
referencing Student table
    Company VARCHAR(255) NOT NULL,              -- Name of the company
    Title VARCHAR(100) NOT NULL,                -- Job title or role
     StartDate DATE NOT NULL,                        -- Date when work
started
     EndDate DATE,                                    -- Date when work
ended (optional for ongoing roles)
     PlaceOfWork VARCHAR(255) NOT NULL,           -- Location of the
workplace
      CONSTRAINT FK_Student_WorkExperience FOREIGN KEY (StudentID)
REFERENCES Student(sid)
);

CREATE TABLE StudentSkills (
    StudentID INT NOT NULL,
    SkillName VARCHAR(255) NOT NULL,
    PRIMARY KEY (StudentID, SkillName),
    FOREIGN KEY (StudentID) REFERENCES Student(sid)
);

CREATE TABLE Events (
    EventID INT PRIMARY KEY,
    EventName VARCHAR(255) NOT NULL,
    EventDate DATE NOT NULL,
    Location VARCHAR(255)
);

CREATE TABLE EventParticipants (
    EventID INT NOT NULL,
    StudentID INT NOT NULL,
    PRIMARY KEY (EventID, StudentID),
    FOREIGN KEY (EventID) REFERENCES Events(EventID),
    FOREIGN KEY (StudentID) REFERENCES Student(sid)
);
```