ALUMNI EVENT MANAGEMENT SYSTEM

Objective: To develop an effective console-based Application designed to manage and maintain the all the Alumni Records. Aims to keep an organized and efficient Application that provides the Facility for maintaining good relationship between college and their former Students as Alumni is the backbone of any Educational Institute so the goal of this Application is to manage the database of alumni, manage Alumni meeting schedule and manage the event, also sending the notification message via SMS to alumni as notification.

Scope: The Alumni Management System will maintain comprehensive alumni records and engagement with those Students through SMS or by any other Platform. So, Educational Institute will have a Strong network with their Alumni.

Technology used: Core Java, SQL/PLSQL, JDBC, JSP, SERVLET etc.

Front-end Technology Used: HTML, CSS and JavaScript etc.

Tools required: Eclipse IDE and MYSQL, Fast2SMS etc.

Database: In this Project we will be using College Database. Means we are taking the data of the pass out Students of that College which include multiple tables and we are going to perform different Operation on that Database tables using SQL. For this we need to create different tables in our database which are as follows:

1. **BatchMaster**

Sr. No	Fields	Data Type	Size	Constraints
1.	Bid (PK)	Int	5	Primary Key Auto Increment
2.	Batch_name	Year	-	Not Null

Above table will Store Alumni's Information regarding their Pass out Year.

- **Table Fields**

a. **Bid:** This Column is for Batch Id which is a Primary Key where we can Store Batchid.
b. **Batch_name:** Here we will store Batch name means generally Alumni's Pass out Year.

2. **AlumniMaster**

Sr no.	Fields	Data Type	Size	Constraints
1.	Aid (PK)	Int	5	Primary Key Auto Increment
2.	Name	Varchar	20	Not Null
3.	Email	Varchar	20	Unique
4.	Contact	Varchar	20	Unique
5.	Age	Int	5	Not Null
6.	Company	varchar	20	Not Null

7.	Bid (FK)	Int	5	Foreign key on delete cascade on update cascade

Above table is created for storing all the Alumni Data.

- **Table Fields**

a. **Aid:** This column is for Storing Alumni Id which is a Primary Key.
b. **Name:** Here we will Store the Full Name of the Alumni.
c. **Email:** We are storing Email Id of an Alumni in this Column which will be Unique.
d. **Contact:** We are storing Contact of an Alumni in this Column which also will be Unique.
e. **Age:** To Calculate Age of the Alumni we are using Age Column in this table.
f. **Company:** Here we are storing Alumni's Company means Currently in which Company he is Working which will be helpful for Future Purpose.
g. **Bid:** Here we are storing Batch Id which is Primary Key from BatchMaster will work as Foreign Key here.

3. EventMaster

Sr no.	Fields	Data Type	Size	Constraints
1.	Eid (PK)	Int	5	Primary Key auto increment
2.	name	varchar	20	Not Null
3.	date	Date	-	Not Null
4.	Time	varchar	20	Not Null
5.	Venue	Varchar	20	Not Null

6.	Bid (FK)	Int	5	Foreign Key on delete cascade on update cascade
7.	Subject	Varchar	200	Not Null

We are creating EventMaster table for Storing all Event Details that will be Happening in that College.

- **Table Fields**

  a. **Eid:** This column will contain Unique Id of Evey Event, it will be a Primary Key.
  b. **Name:** Name Column will contain the Event Name.
  c. **date:** Here we will Store the Date of Event, i.e. when will event Occur.
  d. **StartTime:** Here we will add StartTime of the Event (Time When the event will be going to Start).
  e. **EndTime:** Here we will add EndTime of the Event (Time When the event will be going to Conclude).
  f. **Venue:** The Venue of the Event will be added in this Column.
  g. **Bid:** This is the Column for Storing Batch's Id. It is Foreign Key as it was Primary Key from BatchMaster
  h. **Aid:** This Column is for Storing Alumni Id so we can Identify which Alumni has Attended which Event.

4. Eventregister

Sr No.	Fields	Data Type	Size	Constraints
1.	Reg_id	Int	5	Primary Key auto increment
2.	Reg_date	Date	-	Not Null

3.	Eid	Int	5	Foreign Key on delete cascade on update cascade
4.	Aid	Int	5	Foreign Key on delete cascade on update cascade

Here we are creating Event Register table so that Alumni can do a Registration for any Event.

- **Table Fields**

  a. **Reg_id:** We are designing Registration Id column as every Registration has unique Id which will work as Primary Key here.
  b. **Reg_date:** Here this Column will be reserved for storing the Registration Date**.**
  c. **Eid:** Here we are storing Event Id for that Event, will be Foreign Key in this table.
  d. **Aid:** Here we are Storing Alumni's Id will work as foreign key in this table.

5. FeedbackMaster

Sr. No	Fields	Data Type	Size	Constraints
1.	details	Varchar	20	Not Null
2.	Rating	Int	5	Not Null
3.	Eid	Int	5	Foreign Key on delete cascade on update cascade
4.	Aid	Int	5	Foreign Key on delete cascade on update cascade

Purpose of Creating FeedbackMaster table is to collect Feedback from Alumni for an Event.

6. **Alumni-Batch Join**

Sr no.	Fields	Data Type	Size	Constraints
1.	Aid	Int	5	Foreign Key on delete on Update cascade
2.	Bid	Int	5	Foreign Key on delete on Update cascade

We have created above Table for

- **Table Fields**

a. **Fid:** This Column is for Storing Feedback Id which will be Unique as well as we are using it as Primary Key In this Table.
b. **Feedback_details:** Here we are going to Store details Feedback given by Alumni for that Event.
c. **Rating:** Here we are using this Column for storing Ratings given by Alumni for that Event.
d. **Eid:** This column will Store the Event Id which will be Foreign Key for that column means it will indicate that Alumni will be giving Feedback for that Event Only.
e. **Aid:** This Column will Store the Alumni's Id which will be Foreign Key in this Table and will work as Primary Key in AlumniMaster Table.

Classes Required

---

1. **ClientApplication:** ClientApplication is our Main Class which Contains main method and generally used to provide some inputs and get results. In Client Application, we generally create the Object of our Service and Model class.

2. **AlumniMasterModel:** This is the Model Class where we generally create one POJO class which is used to store the data and send it to the Service layer from ClientApplication and then sending it to the repository Layer from Service Layer. We are creating AlumniMasterModel class so that we can store the All the Alumni's data here using setter methods and retrieve that data using getter methods and display it on console through Client Application.

   Methods used in AlumniMasterModel

   In AlumniMasterModel we are creating the getter and setter methods to store and fetch the Alumni Data.

3. **AlumniRepository:** This is the class where we can write our database logics, means when we create AlumniMaster Table as per the give Constraints and if we want to perform SQL operation on that table then we can do it in this repository**.** Whatever methods we create in repository for calling that method we can create its object in Service class.

   Methods in Repository Class

   ---

   a. **boolean isAlumniadded (AlumniMasterModel):** Here we can Add new Alumni data inside database table. Here we are passing reference of AlumniMasterModel as Parameter.
   b. **List<AlumniMasterModel> getalumni ():** We can fetch all Alumni's Data inside this Method. We are using List Collection as Alumni

data involves their Id, Name, Contact, Email, Age, Pass Out Year, etc. means it contains different types of data So using collection we can achieve it.

c. **boolean isUpdateAlumni (AlumniMasterModel):** Here we are writing Logics for updating Alumni Data by passing AlumniMastermodel reference.
d. **boolean isDeleteAlumni (Alumni's Id):** Here we are deleting Alumni records or data through its id.

All the above defined Methods can be called by creating Object of repository in Service Class.

4. **AlumniService:** This will be our Service Class where generally we write our Business Logics. Also, here we create Object of our Repository Class and call the methods inside it.

Methods inside Service Class

---

a. **boolean isAlumniadded (AlumniMasterModel):** This method will check if city successful added or not return true if successfully added.
b. **List<AlumniMasterModel> getalumni ():** This method will return List as inside that List we have stored all Alumni Data.
c. **boolean isUpdateAlumni (AlumniMasterModel):** This method will check whether we have Alumni details has been successfully updated or not if updated return true otherwise false.
d. **boolean isDeleteAlumni (Alumni's Id):** This method will check whether we have Alumni details has been successfully deleted by id or not if deleted return true otherwise false.

Here all the Above methods will execute only when we create object of Service class in ClientApplication and call the methods in ClientApplication As needed.

5. **EventMasterModel:** This will be the Another POJO class like as AlumniMaster where we will store Event Details using setter method and Fetch Event data using getter method and display on console through ClientApplication.

Methods inside EventMastermodel

In the EventMasterModel we have POJO class for EventMaster which will have setter and getter methods for storing and retrieving Event details. For executing all these Methods, we need to Create Object of EventMastermodel class in ClientApplication and then calling the setter and getter method as required.

6. **EventMasterrepository:** After creating EventMaster table as per the given Constraints if we want to perform some operations on EventMaster using SQL inside this Repository.

Methods of EventMasterrepository class

---

a. **boolean isEventadded (EventMasterModel):** Through this method we are adding event details inside database table. We are passing EventMasterModel reference as parameter.

We can call the above method by creating its Object in Service Class.

7. **EventMasterService:** We will be writing business logics inside Service Layer.

Methods of EventMasterService

---

a. **boolean isEventadded (EventMasterModel):** This method will check whether the Even has been successfully added or not return true if added otherwise false.

Above method will be called when we create object of Service Class in ClientApplication and call the method as required.

8. **FeedbackMasterModel:** This is Another Model class which we are creating for maintaining the feedback of every event. This is very important part as our main goal in this Project is to collect feedback from only those Alumni who were present in that event. So, in this Model class we are creating one POJO class where we are storing all the Fields in Feedback table with the help of getter and setter methods.

9. **FeedbackRepsitory:** In this method we will be Performing Operations on FeedbackMaster table using SQL.

Methods of FeedbackRepository

---

a. **boolean isAddfeedback (FeedbackMasterModel):** Here we **are** adding Feedback details inside the database table for that we are passing FeedbackMasterModel reference as parameter.
b. **List<FeedbackMaterModel> getfeedback ():** Here we are Storing all the feedback details inside List as there is Feedback id,

Feedback details, ratings and many more means there are multiple data types so in such case we can use Collection as it stores multiple type of data.

We can execute the above methods by creating its Object in Service Class and calling those methods.

10. **FeedbackService:** Here we generally write our business logics. Object of repository gets created in FeedbackService class.

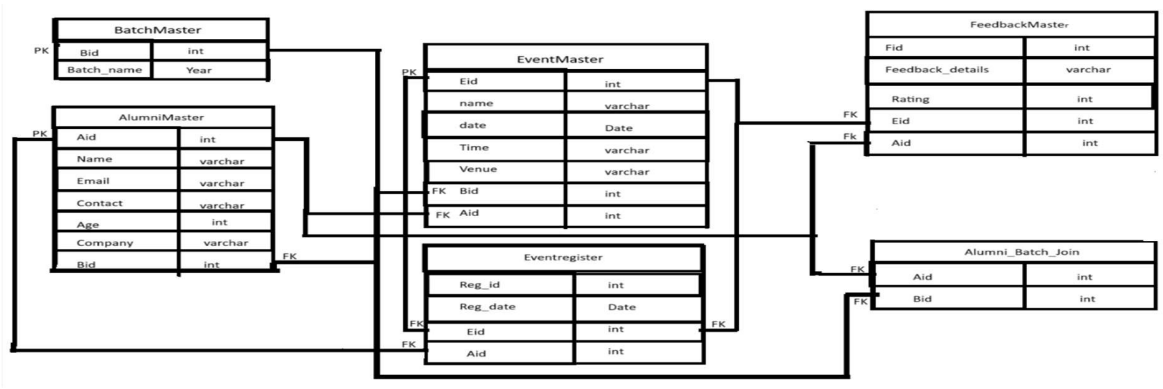
Methods of FeedbackService

---

a. **boolean isAddfeedback (FeedbackMasterModel):** Ti=his method will check whether Feedback details has been added or not if added return true else false.
b. **List<FeedbackMaterModel> getfeedback ():** Here List will return all the feedback details present in database table.

The above methods will be called when we create object of Service class in ClientApplication and call the above methods.

ER Diagram:-



FlowDiagram:-