Alumni Network Database

Karim Iqbal

Information Management

May 02, 2024

Agenda

- Introduction
- Database Design
- Implementation
- Conclusion

Introduction

Introduction

Raiser's Edge (RE)

- Customer Relationship Management (CRM) solution for records management and fundraising.
- Nonprofit Organizations (NPOs) and Higher Education Institutions (HEIs) commonly use RE.
- RE NXT (First released in 2015)

'Alumni Network' Database

- Focus on career service functionalities and networking opportunities for Alumni.
- Address identified gaps.
- Jobs, Events, and Mentorship opportunities

Database Design

Database Design

- Select entities to provide foundation for how data was to be inserted in the database.
- Entity sets and relationships

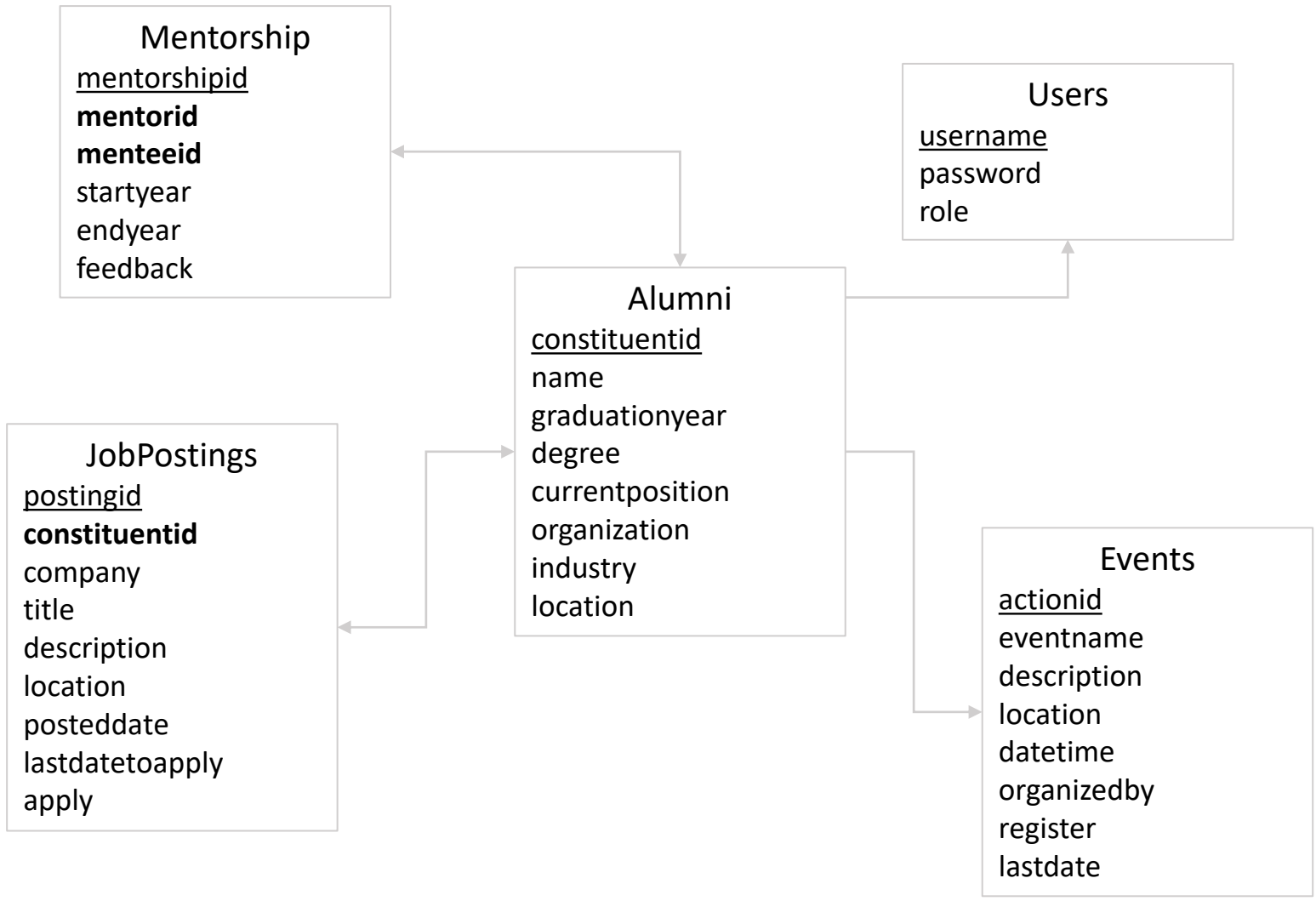
Alumni	Events	JobPostings	Mentorship	Users
Individual UTD Alumni with their personal and professional information	Detail of events that may be of interest to alumni such as reunions, galas, etc.	Information about job opportunities posted, especially those for alumni	Mentorship relations for alumni, facilitating career development	Manage user accounts, particularly for authentication and role management

Database Design

- Determine primary key or foreign key to preserve referential integrity.
- Ensuring keys are related to current RE format.
- Primary key (PK) examples
  - Alumni: ‘constituentid’, Events: ‘actionid’, JobPostings: ‘postingid’, Mentorship: ‘mentorship id’, User: username
- Foreign key (FK) examples
  - JobPostings: ‘constituentid’ references ‘alumni(constituentid)’ – Links job posting to alumni
  - Mentorship: ‘mentorid’ references ‘alumni(constituentid)’ – Links to the alumni table identifying who is the mentor
- Sequences for primary keys like ‘alumni_constituentid_seq’, ‘events_actionid_seq’, etc., are used to auto-increment these identifiers.
- Attribute datatypes

Database Design

For the determined entities, keys to be used (primary key is underlined, foreign key is bold)	
Alumni	constituentid, name, graduationyear, degree, currentposition, organization, industry, location
Events	actionid, eventname, description, location, datetime, organizedby, register, lastdate
JobPostings	postingid, **constituentid**, company, title, description, location, posteddate, lastdatetoapply, apply
Mentorship	mentorshipid, **mentorid**, **menteeid**, startyear, endyear, feedback
Users	username, password, role


Mentorship
mentorshipid
mentorid
menteeid
startyear
endyear
feedback
Users
username
password
role
Alumni
constituentid
name
graduationyear
degree
currentposition
organization
industry
location
JobPostings
postingid
constituentid
company
title
description
location
posteddate
lastdatetoapply
apply
Events
actionid
eventname
description
location
datetime
organizedby
register
lastdate

Implementation

Implementation

- Create tables

```
CREATE TABLE public.alumni (
    constituentid integer NOT NULL,
    name character varying(100),
    graduationyear integer,
    degree character varying(50),
    currentposition character varying(100),
    organization character varying(100),
    industry character varying(100),
    location character varying(100)
);
```

```
CREATE TABLE public.events (
    actionid integer NOT NULL,
    eventname character varying(1000),
    description text,
    location character varying(1000),
    datetime timestamp without time zone,
    register character varying(1000),
    organizedby character varying(1000),
    latedate timestamp without time zone
);
```

```
CREATE TABLE public.jobpostings (
    postingid integer NOT NULL,
    constituentid integer,
    company character varying(1000),
    title character varying(1000),
    description text,
    location character varying(1000),
    posteddate date,
    apply character varying(1000),
    lastdatetoapply date
);
```

```
CREATE TABLE public.mentorship (
    mentorshipid integer NOT NULL,
    mentorid integer,
    menteeid integer,
    startyear integer,
    endyear integer,
    feedback text
);
```

```
CREATE TABLE public.users (
    username character varying(50) NOT NULL,
    password character varying(50),
    role character varying(10)
);
```

Link

Implementation

- Create sequence and default values for Primary Keys

```
CREATE SEQUENCE public.alumni_constituentid_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE ONLY public.alumni ALTER COLUMN constituentid SET DEFAULT nextval('public.alumni_constituentid_seq'::regclass);
```

- Establish Foreign Keys and Additional Constraints

```
ALTER TABLE ONLY public.mentorship
    ADD CONSTRAINT mentorship_menteeid_fkey FOREIGN KEY (menteeid) REFERENCES public.alumni(constituentid);
```

- Populating tables with data

```
COPY public.alumni (constituentid, name, graduationyear, degree, currentposition, organization, industry, location) FROM stdin;
```

- Running and testing queries

```
SELECT COUNT(*) FROM public.alumni;
```

```
SELECT * FROM public.jobpostings
ORDER BY postingid ASC
```

Implementation

- **Setup environment**
- **Define UI components**
- Define server logic
- Data interaction and updates
- Deploy and manage database connection

```
# Define UI for the application
# Define UI for the application
ui <- fluidPage(
  useShinyjs(),
  navbarPage(
    theme = shinytheme("flatly"),
    "Alumni Network",
    id = "navbar_tabs",
    tabPanel("Login",
             id="login",
             fluidRow(
               column(2,
                      textInput("username", "Username"),
                      passwordInput("password", "Password"),
                      actionButton("submit_login", "Submit", class = "btn btn-primary")
               )
             )
    ),
    tabPanel("Alumni Listing",
             dataTableOutput("alumni_table")
    ),
```

Implementation

- Setup environment
- Define UI components
- **Define server logic**
- Data interaction and updates
- Deploy and manage database connection

```
# Define server logic
server <- function(input, output, session) {
  observe({
    # Hide all tabs except the login tab
    shinyjs::hide("navbar_tabs")
    runjs("$('#submit_logout').css('display','none');")
    shinyjs::show("login")
  })
  user_logged_in <- reactiveVal(FALSE)

  output$alumni_names <- renderUI({
    # Connect to the database
    con <- dbConnect(
      RPostgres::Postgres(),
      dbname = "alumni",
      host = "ep-falling-silence-a4xfratq.us-east-1.aws.neon.tech",
      port = 5432,
      user = "default",
      password =
    )

    # Query to retrieve alumni names
    alumni_names <- dbGetQuery(con, "SELECT name FROM alumni")
```

Implementation

- Setup environment
- Define UI components
- Define server logic
- **Data interaction and updates**
- Deploy and manage database connection

```
# Alumni Create
observeEvent(input$submit_alumni, {
  # Construct query to insert values into the Alumni table
  qry <- paste0("INSERT INTO Alumni (Name, GraduationYear, Degree, CurrentPosition, Organization, Industry, Location) ",
                "VALUES ('", input$name, "', ", input$graduation_year, ", '", input$degree, "', '", input$current_position, "', '",
                input$organization, "', '", input$industry, "', '", input$location, "')")

  # Query to send to the database
  dbSendQuery(conn = con, statement = qry)

  qry <- paste0("INSERT INTO users (username, password, role) ",
                "VALUES ('", input$alumini_username, "', '", input$alumini_password, "', 'user')")



  # Query to send to the database
  dbSendQuery(conn = con, statement = qry)

  # Show modal dialog to user when the update to the database table is successful
  showModal(
    modalDialog(
      title = "Alumni Data Inserted",
      br(),
      div(tags$b("You have inserted the data into the Alumni table"), style = "color: green;")
    )
  )
  output$alumni_table <- renderDataTable({
    alumni_data <- dbGetQuery(con, "SELECT Name, GraduationYear, Degree, CurrentPosition, Organization, Industry, Location FROM Alumni")
```

Implementation

- Setup environment
- Define UI components
- Define server logic
- Data interaction and updates
- **Deploy and manage database connection**

Link

```
  # Disconnect from database on session end
  session$onSessionEnded(function() {
    dbDisconnect(con)
  })
}
```

Conclusion

Conclusion

- RE Alumni Network Database
- Database design based on UTD requirements and RE features
- Implementation
- Limitations

Thank You!