

# College Event Database

COP 4710 Summer 2016

Group 8: Joshua Casserino, Lindsay Hofer, Richard Pazda

# Table of Contents

[Project Description](#)

[GUI](#)

[ER Model](#)

[Relational Data Model](#)

[Populating Tables](#)

[SQL Examples and Results](#)

[Software Installation](#)

[Observations](#)

## Project Description

The purpose of this project was to create a web application to manage and view events for universities. The application provides the tools to create and manage Users, RSO's, Universities, and Events.

## GUI

The graphical user interface (GUI) for the application was built using modern web components for a standard website. The layout, style, and many components were created using HTML, CSS, and Bootstrap, a popular CSS and JavaScript framework. Connecting the UI elements to the database to send and receive data was done using PHP, a popular server-side scripting language. The database itself was an instance of either MySQL or MariaDB (different members used different configurations) administered through PHPMysqlAdmin.

Additional UI features and components were added using FontAwesome and ListJS.

FontAwesome is an icon library that we used to add a logo to the header and for any possible icons we might have needed. ListJS is a JavaScript library that adds the ability to sort and filter tables using simple classes and a simple JavaScript function. This was used to make the events lists easier to navigate and get relevant information from.

# Event Details Page

University Events

User: Test Student10@knights.ucf.edu  
Log Out

Event Details


Event Name

TestEvent01

Event Location

UCF Main Campus

Map



Event Description

Event Category

Academic

Hosting RSO

TestRSO01

Event Contact Information

Contact Name

Test Student02

Contact Phone

555-555-0001

Contact Email

Test.Student2@knights.ucf.edu

Rating

1

2

3

4

5

Comments

Comment


Event Comments

Average Rating 0.0

User	Comment	Created On
Test.Student2@knights.ucf.edu	This is a test comment!	2016-07-27 22:04:54
Test Student10@knights.ucf.edu	This is a test comment!	2016-07-27 22:04:54
Test Student13@knights.ucf.edu	This is a test comment!	2016-07-27 22:04:54

Back to Events

# Create RSO

 University Events

User: Test.Student10@knights.ucf.edu  
Log Out

Join RSOs

There are no pending RSO requests!

Create RSO

**RSO Name**

Name

**RSO University**

**RSO Description**

Description

RSO Initial Members

Creating a new RSO requires six (6) initial members, one of which must be assigned as the RSO Admin. The RSO Admin is responsible for creating events on behalf of their RSO and are the only members allowed to do so.

**RSO Admin Email**

Admin Email

**RSO Member Emails**

Member Email

Member Email

Member Email


Member Email

Member Email

Create RSO

Back to Events

## Create Event

 University Events

User: Test.Student2@knights.ucf.edu  
Log Out

Create New Event

**Event Type**  
☐ Public Event  
☐ Private Event  
☐ RSO Event

**Event Name**

**RSO you are Admin of :**

**Event Date and Time :**

**Event Location**  
Select a location from the dropdown or select a new location on the map below  
  
OR  

**New Event Location Name**

**Latitude**

**Longitude**

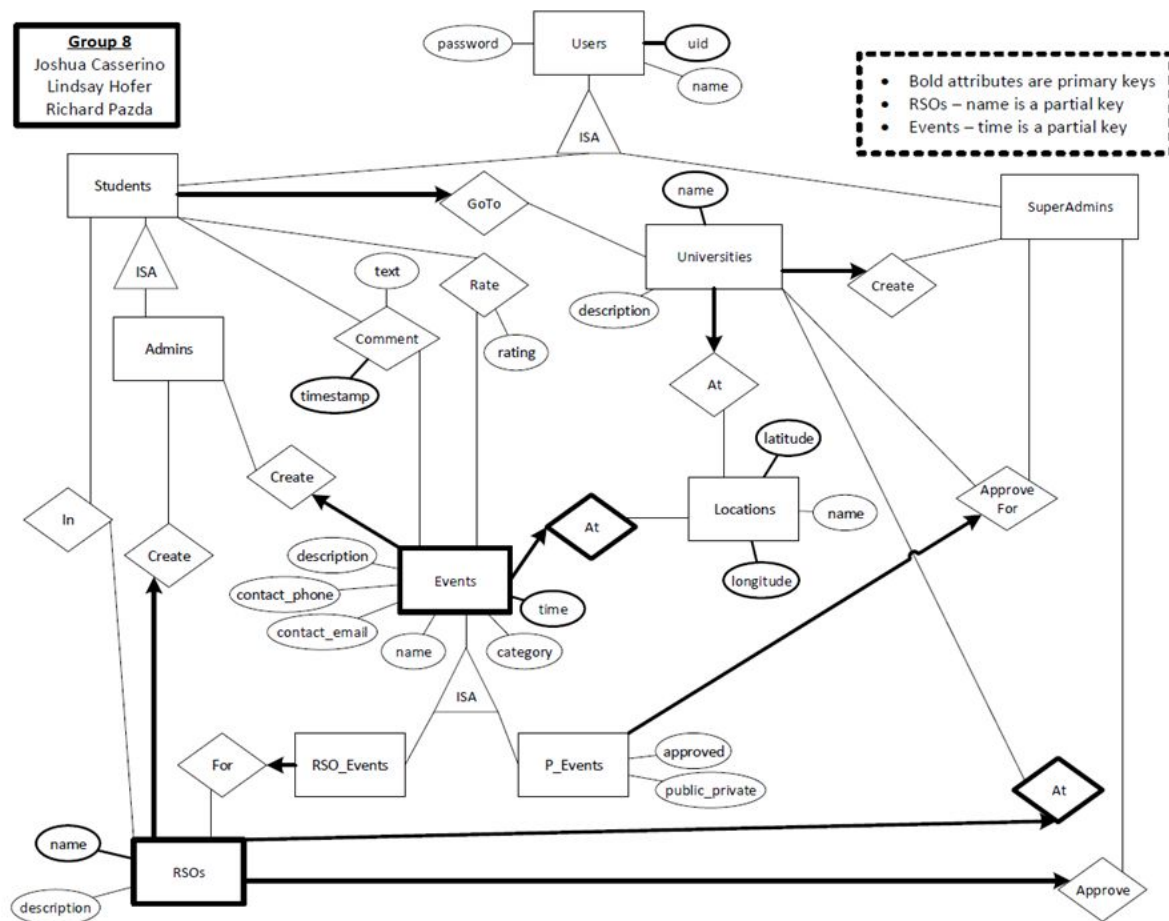
**Event Description**

**Event Category**

**Event Contact Information**

Create Event

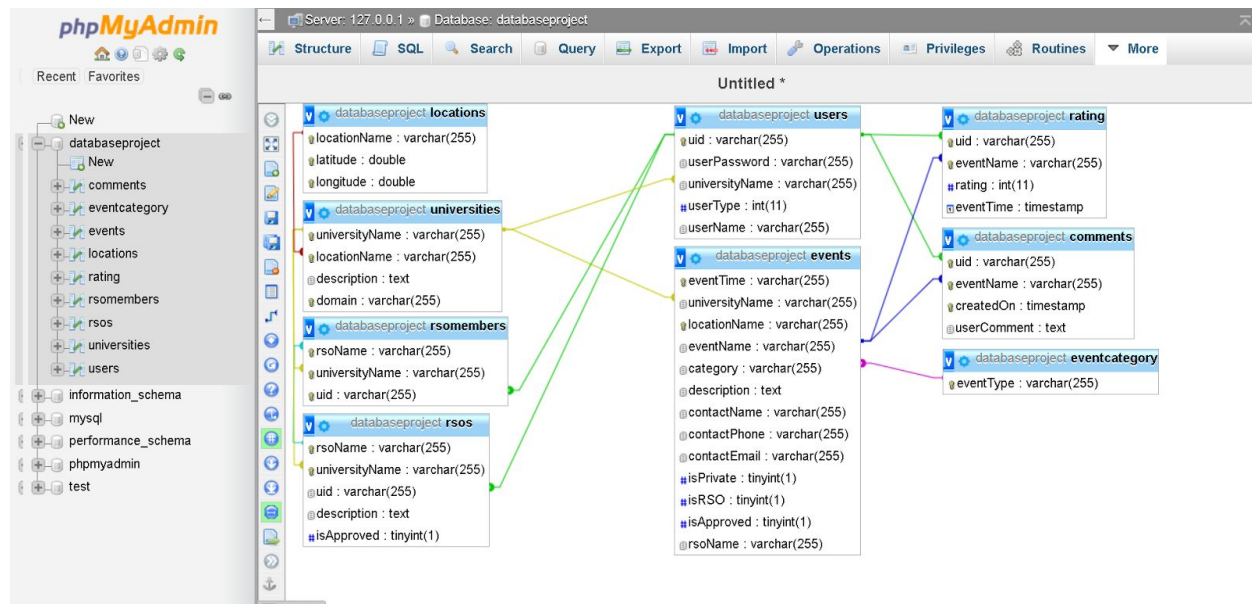
## ER Model



We wanted to make our database as compact as possible, so we started with a simple ER diagram. Students and SuperAdmins both connect to Users, Admins connects to Students, and RSO\_Events and P\_Events connect to Events, all using ISA relationships. On the tables in the database, these ISA relationships are modeled by trinary attribute, such as 'userType' in the Users table. We also made Events weak to Locations (making a Location a portion of the primary key of an Event, so that there can be multiple Events at a time or location, but not both) and RSOs were made weak to Universities (making a University a portion of the primary key of an RSO, so that there can be RSOs with the same name as long as they are at different Universities). Most of the relationships on the ER diagram are 1-to-many, so we

combined them with the entities in the database. Also, we enforced the 5 extra students constraint on RSO creation in the Create\_RSO form, so we didn't need to create an assertion to check it.

## Relational Data Model



## Populating Tables

A file called “databaseproject.sql” file is included in the total zip file for the website. This file is an exact breakdown and copy of our whole database and, from it, the entire database can be recreated and tested. The database is currently only filled with a limited amount of test data. Currently, there are 21 users (20 students/1 superAdmin), 4 Universities(3 for users/1 for superAdmins), 3 RSOs (With 5 members and 1 admin each), 3 events and 3 commits. When the database is recreated, and the website is running, all insertions should be done through the site and not using the SQL option in myAdmin. The website aids in data integrity as it limits the way a user can interact with the database directly.



## SQL Examples and Results

### SQL statement to insert a new RSO (part of the processing of the ‘Create RSO’ form)

```
INSERT INTO `rsos` (`rsoName`, `universityName`, `uid`, `description`)
VALUES ($rsoName, $universityName, $rsoAdmin, $rsoDescription);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoMember1);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoMember2);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoMember3);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoMember4);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoMember5);

INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($rsoName, $universityName, $rsoAdmin);
```

### SQL statement to insert a new student to an existing RSO (part of the processing of the ‘Join RSO’ form)

```
INSERT INTO `rsomembers` (`rsoName`, `universityName`, `uid`)
VALUES ($optionArray[$i], $userUniversity, $inputUser);
```

### SQL statement to insert a new event (part of the processing of the ‘Create Event’ form)

```
INSERT INTO `events` (`eventName`, `eventTime`, `universityName`, `locationName`,
`category`, `description`, `contactName`, `contactPhone`, `contactEmail`, `isPrivate`)
```

```
VALUES ('$eventName', '$eventDate', '$currentUniversity',
'$location', '$eventCategory', '$eventDescription', '$eventContactName', '$eventContactPhone',
'$eventContactEmail', 1);
```

**SQL statement to insert/update a (new) comment (part of the processing of the ‘Create/Add/Modify Comment’ form)**

```
INSERT INTO `comments` (`uid`, `eventName`, `createdOn`, `userComment`)
VALUES ($userID, '$eventInfo['eventName']',
CURRENT_TIMESTAMP, '$comment');
```

**Several SQL queries to display events—public, private, and RSO-- (part of the processing of the ‘View Event’ request by a user with a specific role)**

Public Event Query

```
SELECT `eventName`, `locationName`, `eventTime`, `category`
FROM `events`
WHERE `isPrivate` = 0
AND `isRSO` = 0
AND `isApproved` = 1;
```

Private Event Query

```
SELECT `eventName`, `locationName`, `eventTime`, `category`
FROM `events`
WHERE `isPrivate` = 1
AND `isRSO` = 0
AND `isApproved` = 1";
```

RSO Event Query

```
SELECT                                events.eventName, events.rsoName, events.category,
events.eventTime, events.locationName

FROM                                events, rsomembers

WHERE                                (events.isRSO = 1)

AND                                (events.rsoName = rsomembers.rsoName)

AND                                (rsomembers.uid = '$userID')";
```

**SQL statements of interest (optional), e.g., advanced SQL queries**

**Available RSOs users at that university can join**

First find users university

```
SELECT                                universityName

FROM                                `users`

WHERE                                `uid` = '$inputUser';
```

Then with that information find any RSO the user can join, plus only show RSOs the user isn't already a member of

```
SELECT                                rsos.rsoName

FROM                                rsos, users

WHERE                                (users.uid = '$inputUser')

AND                                (users.universityName = rsos.universityName)

AND                                (rsos.isApproved = 1)

AND                                rsos.rsoName

NOT IN (

                                SELECT                                rsoName
                                FROM                                rsomembers
```

```
WHERE      uid = '$inputUser'  
  
);
```

## Software Installation

The source code should be placed in the main folder of a web server that is running PHP and preferably be configured with a compatible database such as MySQL (ours were configured with WAMP and XAMPP localhost instances). The database should have a localhost account with account name “root” and password “”. This was configured for the sake of simplicity and would be modified to use a configuration file before being deployed for real-world use. A database should be created with the name “databaseproject”. From there, the populate scripts file contents (databaseproject.sql file) should be copied and ran through the myAdmin SQL option to create the tables within the database.

## Observations

During testing the database performed well. The response time for queries, while not formally recorded and averaged, was deemed satisfactory for the scope of the application. Real-world performance would be difficult to estimate due to the development team’s use of local databases rather than a deployed production database and lack of experience with such systems. Despite this, the team is confident the database was designed efficiently and would be satisfactory for a deployed application. The project makes use of one index, one on event name in the events table, which was used to order the data and treat it as a foreign key without designating it as a primary key. Due to the minimal nature on the schema the development does not have many recommendations for indexes because most data can be ordered by primary key. One possible beneficial index would be on Location name since presenting the data alphabetized would be a huge aid if there was more data and performance improvements would be good for the UI.

There were several additional features that would be desirable expansions to this project. The most obvious aspect of the site's design that could be improved upon is security. Certain assumptions were made for the purpose of simplifying the scope of the project such as assuming certain pieces of input would be correct, assuming no malicious intent, and assuming the database contents would not be compromised. In expectation of deployment and real world use the application's security would need to be strengthened, such as sanitizing all input before being passed to queries and ensuring all pages and functions are limited to users with required access. Social media integration was also not fully configured for the application as well as the ability to pull events from an RSS feed.

The development of this application was not without problems, the most serious of these being the result of serious issues with source control. During a crucial merge of very large components the branches refused to commit and the merge could not be accomplished. This meant a large portion of the code written could not be added to the application. To continue working the team accepted the copy with the largest changes made as the newest version and rewrote the missing UI components and features on top of it. The time constraint for the project also made it challenging. This difficulty was further compounded by the variety of technologies the team needed to become competent with in such a short time.