

Share Contribute Comment (SCC) project report

Course COMP 353 Database

Professor Bipin C. Desai

Team Members

Ivan Garzon, 27006284

Ragith Sabapathipillai, 26854486

Nicolas Brodeur-Champagne, 27043651

Samie Siamak, 27502303

Jesse Desmarais, 40035761

Table of Contents

Project description	3
Assumptions	4
Applications supported	6
The limitations	7
Architectural design	8
ER diagrams	9
Relational DB design	12
3NF Solution	13
Member responsibility	16
Gantt chart	19
Detailed analysis of code	21
UI design rationale	23
Queries	37
User Manual	27
General information	31
Authorized use permissions	32
System summary	33
Overview of the user interface	34
Gitlog	47
Database	51
Script to create and populate DB	52
References	72

Project description

A social media website that allows users to register for events and subsequently event groups. Users use this interactive website to promote discussions between event group member by means of; sending emails, group wall posts, instant messaging between members. For access to the website, a person must be registered for the site and create their event, becoming the event manager. The event manager will then invite group members (potential user's) by means of a csv file, and these intended persons will receive an email, with their username and password to log into the website.

Once logged in, user's are in a default group, they may they create and add join other groups. In these user's discuss with posts, and messages. Event Manager will be able to handle the location, resources, and payment for their event.

The project includes creating a website to complete all requirements described above, designing the database that will be used to persist website data, and a lot of overhead in over-engineering the database to adhere to the courses project specifications.

Assumptions

During the first stage of the Database structure development we had to discuss several points mentioned in the documented project requirements. The project description does not explicitly states whether this platform is developed for the purpose to be used directly by clients or if there is a specific organization who uses it to manage the event on behalf of their clients. In other words, we were not sure if the intended use of this system is employed by an event management team working for a client or if it is the client himself using the platform (assuming each of the SCC roles). This was an important part of the system development, since we were not sure about the following questions:

1. What is the role of the system admin? If he's in charge of installing, deploying and maintaining the server isn't that a qualified technical support role?
2. Why is the system admin doing both system maintenance and adding events?
3. Are the storage and bandwidth limits managed by the Server Hosting provider or is the SCC hiring a tech support team to keep track of server data usage?
4. When the system admin adds a new event, who provides him the details of the event and by which means (email, excel file, google docs, using the SCC)?
5. Who provides the charge rate data of the events to the Controller? If the event manager can extend the period of time charging an extra fee/rate, isn't he also having the role of Controller?
6. When the event manager sets up a home page, who provides him the template? Is it a built-in template or he creates his own?
7. Who is the account owner of the banking details: the event manager or the client?
8. Are the banking details specific to each event or are they global to the SCC system?
9. If the event manager is the only one who can change the status of a participant, does it mean that the group manager cannot change the status of a group participant (rejected, approved, etc.) ?
10. What does it mean that the email messages have to be simulated? Does it mean the SCC has to mimic email services functionality? What exactly is an internal email message system?
11. What is meant by Report of groups/members by a category? Is it a search filter? A dashboard with chart statistics?

Hence, we built the SCC system having in mind a specialized event management company, which will use this platform as a service to manage the event for their clients.

Our team had to make the following assumptions:

1. An event can be managed by one and only one event manager. Hence, the event table will have a column for event manager, and since it's a many-to-one relationship, we don't need to create a relationship table to link both entities Users and Events.
2. In the project specification, we have two role descriptions: event manager and event admin. Due to the similarity of the role's responsibilities, we assume that this description is referring to the same role a person has in the SCC system.
3. The system admin will have the flexibility to create an empty event (no data or minimal information) which will create an empty archived event in the Database. As soon as the event manager fills in all the relevant information he has to Finalize the event, which will make it an active event at this point.
4. The system admin does not have the role of tech support, the company will hire these services from a dedicated virtual private hosting company which will manage and deploy the system. However, the system admin has all the global privileges on all the platform.
5. The storage and bandwidth limits are static fixed rates applied to all events (not tracked by a system).
6. For the first version (beta version 1.0) of the SCC system, there will be only one home page built-in template.
7. The account owner of the banking details is always the client (he pays for the event). Hence, he does not need to be a registered user in the Database.
8. The banking details are specific to each event, and for each new event a new banking record will be created in the Database.
9. Both the event manager and the group manager can approve/reject participants, therefore they can change the status of a participant.
10. There is a DB table to simulate email (inbox, sent, date, sender, subject, body)
11. There is a DB table to simulate instant messages between participants
12. The group report will be done as a php page which will display statistics/charts based on metadata

Applications supported behaviours

Creating Events: An event manager assigned to an event may create, edit, archive an event.

User import: An event manager of an event may import users from a CSV file

Adding/editing Members to Event: An Event Manager can add/edit/remove members to/from his events.

Adding Event Resources: An Event Manager can specify event resources of his events.

Adding Event Location: An Event Manager can specify location of his events.

Event Posting: An event member can create a post in his events.

Event Payment: An Event Manager can pay their monthly dues via payment system.

Group Management: An Event member can create/edit/archive a group.

Group Members: A group admin can add/reject a member to his group.

Group Invite: A group member can invite other event members to join his groups.

Group Join: An event member can accept or reject group invitations.

Group Posting: A group member can add posts to his group walls.

Instant Messaging: Event/Group members can instant message other event/Group members.

Emails: All Event members have an inbox where they can read and send emails.

Reports: An Event Manager can view the post activity of the users and sort the groups/users by tags.

The limitations

For this project there were several limitations set forth from the requirements document and project description. The software limitations were set in the initial project documentation. Our project was limited to using PHP as the server side language with the client side languages being only HTML, CSS and JavaScript. There was also the limitation which did not allow us to use any frameworks. For this reason Laravel, a PHP framework was not used.

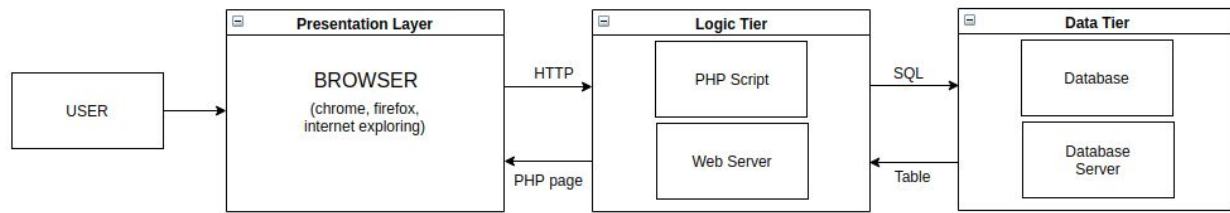
Considering that this is a database course with a focus on relational databases. The project was required to a relational database with a relational database management system. We were instructed to use MySQL database management system.

Further, we were limited in where the server could be stored. We were required to use the AITS system and thus our server has a limited space capacity and speed. Our server is limited to only being accessible through concordia university IP addresses or through a Concordia University Virtual Private Network. Having this limitation meant all development done locally outside of the school would later have to be ported to the university infrastructure.

The AITS system limits our ability to send outgoing emails. One of the requirements of this project is to build an email system into our web application. The AITS limits the ability of this system since outgoing emails are restricted. For this reason we are limited to simulating an email system but cannot integrate a fully functioning system into our application.

The last limitation that will be discussed here is the limitation of using no frameworks. For this reason we were required to use the programming languages above without the ability to use libraries or frameworks to increase the speed and efficiency of our work.

Architectural design

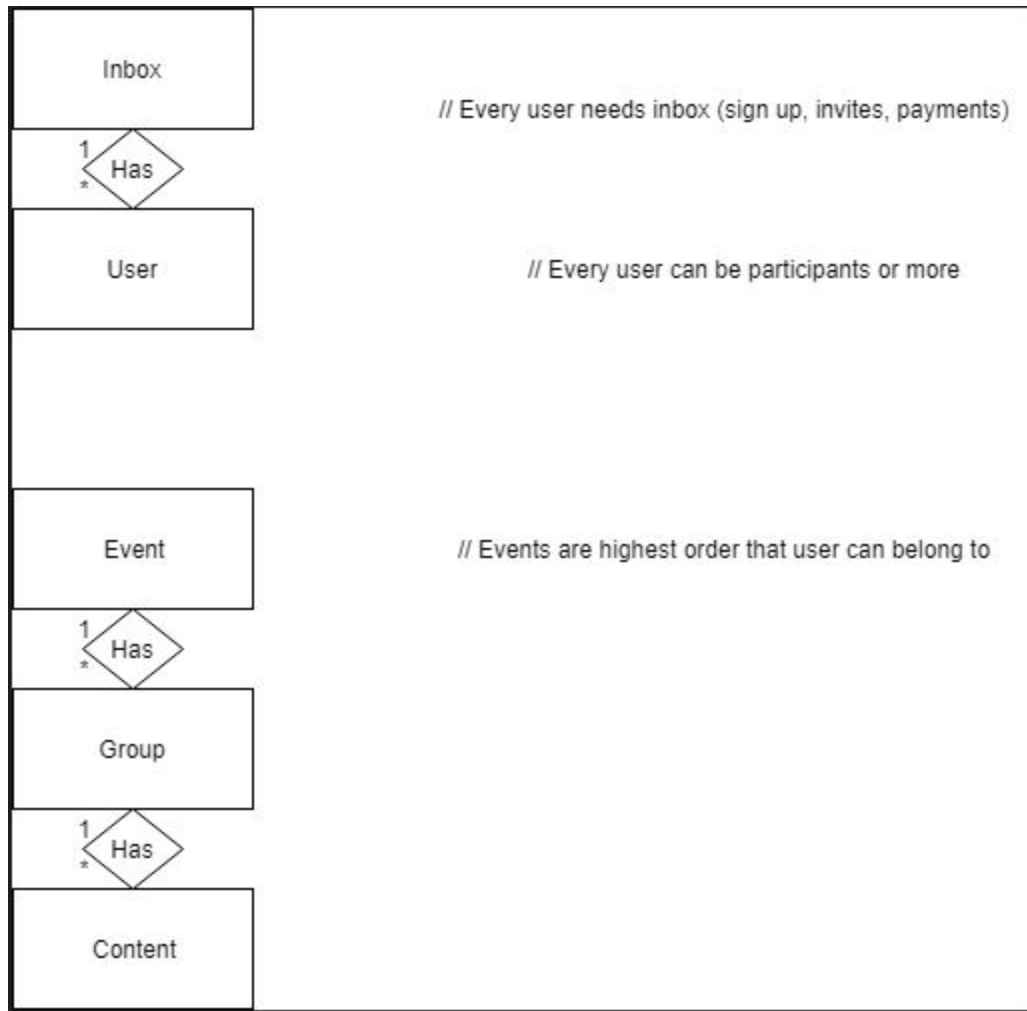


Layered Architecture Diagram

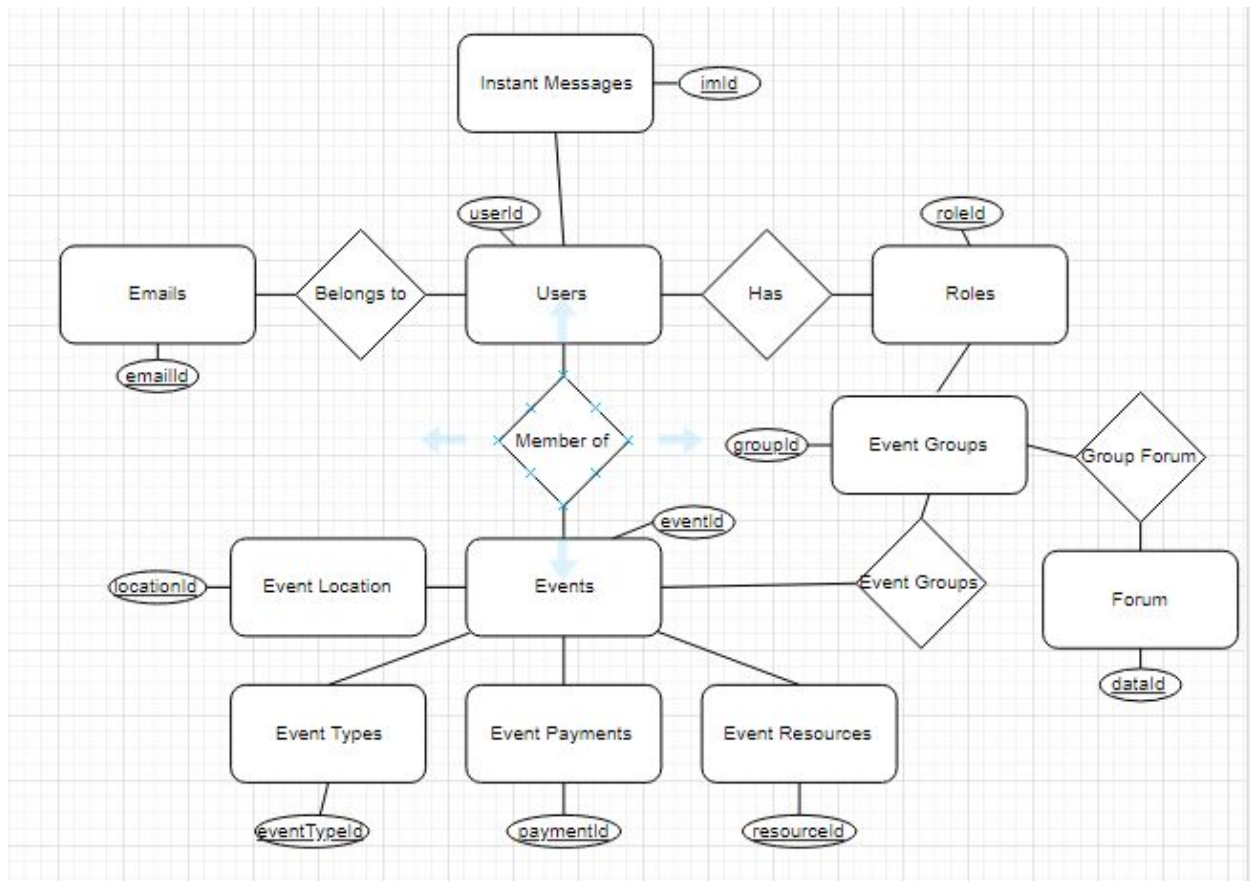
The layered architecture diagrams show the different layers present in the software application. The user interacts with the presentation layer. The presentation layer is accessible via the browser. The browser makes a HTTP request to the logic tier. The browser receives a PHP page from the logic tier. The Logic tier has PHP scripts and the web server. The scripts are used to query the Data Tier. The scripts send SQL to the data tier. The web server receives tables and data from the data tier. In the data tier, the Database is queried when the SQL is received. The Database Server responds and sends out table and data.

ER diagrams

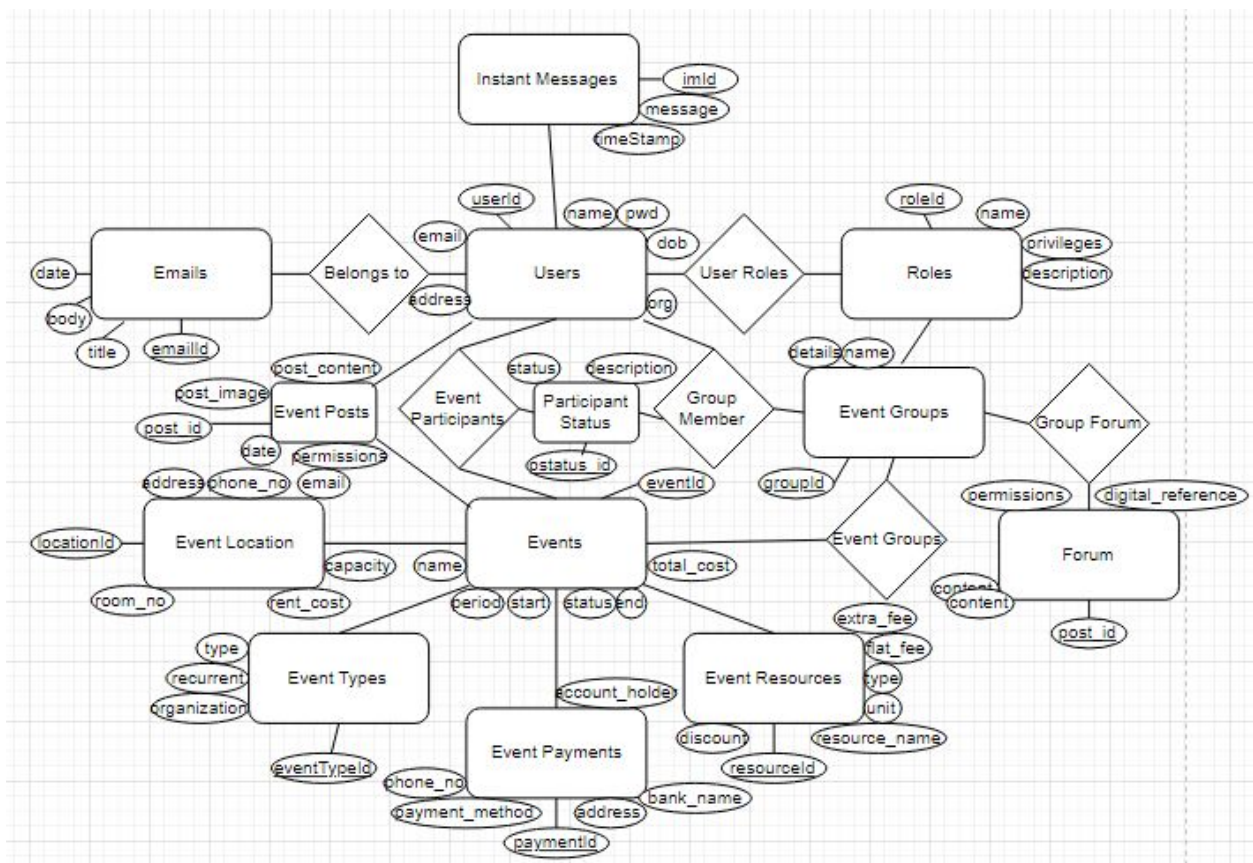
First sketch to separate important entities after first reading all requirements:



Original ER design:

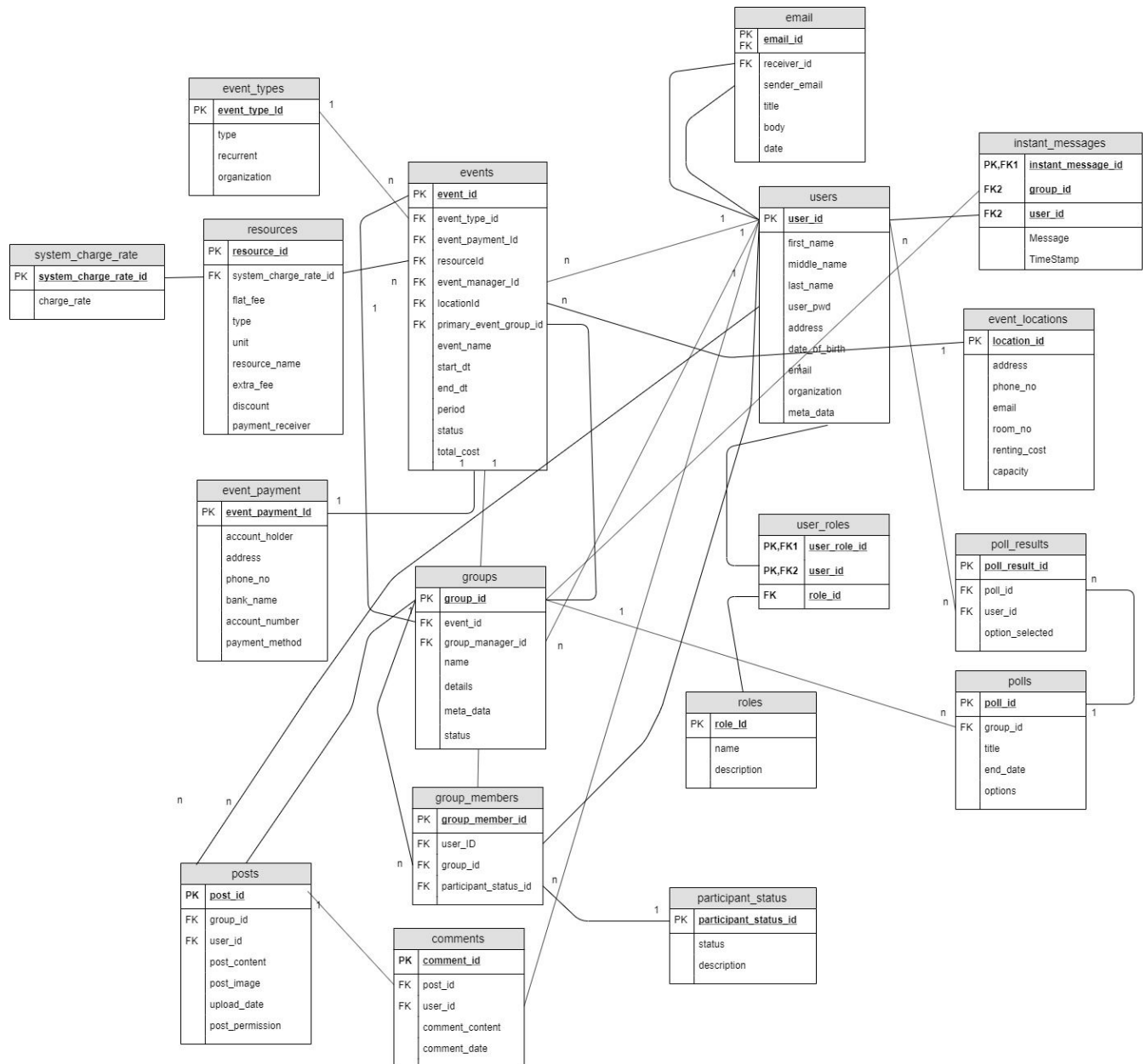


Final ER Design:



Relational DB design

Final product of relation db schema creation



3NF Solution

1st normal form:

- ☐ Each table has a primary key
- ☐ All attributes must be atomic
- ☐ No multi valued or composite attributes

2nd normal form:

- ☐ 1NF must be satisfied
- ☐ All non key attributes must be dependent on the full primary key

3rd normal form:

- ☐ 2NF must be satisfied
- ☐ No attributes must depend on other attributes that are not primary keys

Originally the design was not 3NF!

It does not satisfy the rule that no attributes must depend on other attributes that are not primary key

Problem:

The attribute “total_cost” is dependant on “event_type_id” and “resource_id” which are both non-key attribute.

The attribute “privilege” is dependant on “name” which is a non-key attribute.

Solution:

To resolve the problem we can simply remove total_cost since we can compile to total cost at runtime. No need to persist that information especially since it is subject to change.

Final Table Design, 3NF

After applying solution, the system is in 3NF. Here are the functional dependencies(_):

Event Type Table:

Event_type_id -> type, recurrent, organisation

Event Resource Table:

Resource_id -> basic_fee, type, unit, name, extra_fee, discount

Event Table:

Event_id -> event_type_id, event_payment_id, resource_id, event_manager_id, location_id, event_name, start_dt, end_dt, period, status, total_cost

Event Payment Table:

Event_payment_id -> name, address, phone_no, bank_name, account_number, payment_method

Event Location Table:

location_id -> address, phone_no, email, room_no, renting_cost, capacity

Post Table:

post_id -> group_id, user_id, data_type, content, permissions, digital_reference

Group Members Table:

group_members -> user_id, group_id, participant_status

Role Table

roleId -> name, privileges, description

User Roles Table:

userId, roleId -> group_id

Group Table:

group_id -> event_id, name, details

Group Forum Table:

post_id -> group_id, user_id, date_type, content, permissions, digital_refernence

Event Participants Table:

event_id, user_id -> participant_status_id

Participant Status Table:

participant_status_id -> status, description

Instant Messages Table:

sender_id, receiver_id -> message, timestamp

Email Table:

Email_receiver_id -> email_id, email_sender_id, title, body, date

Member responsibility

From the beginning of the development phase we discussed each of the tasks that needed to be developed and we gave responsibility to each team member. We used the application Slack, as a communication platform to share important information about the project development, raise important questions, share documents, assist a team member with code issues, collaborate continuously, assign responsibilities, and keep a log of our meetings. Below are some screenshots of the logs created and each weeks tasks assigned to each team member:



jesse D 15:26

11/06/2019 First Day of production

Each teammate brought in ER-Diagram , and we compiled them together to create a Final ER Diagram

Finalized ER Diagram was created into a DB called scc_db

Set up project github with this database Each member will have a task to do

Jesse: Create users + login

Ragith: Sys Admin Create Event

Ivan: Assigning event manager + info

Cmak : Normalizing 3NF

Nico : Relation Diagram From ER (edited)



Ivanovski 19:03

11/13/2019 Managing events, users and normalizing ER

Each teammate discussed their progress on the project , and we merged the code in the master branch

Login, authentication and uploading users into DB has been developed by Jesse. Also, Jesse created an AWS remote DB.

Ragith created the interface to save an event to DB, and a page to finalize the event attributes

Ivan created the interface to view events assigned to an Event Manager and a page to view/update event details

Samie analyzed the DB schema and normalize it

Nico analyzed simulated email interface



Ivanovski 17:05

11/20/2019 Managing events, users dashboard, emails and Remote DB

Each teammate discussed their progress on the project , and we split some tasks

Event creation has been improved by Ivan by adding new tables and an SQL trigger.

Ragith created a user dashboard to view navigation bar options and current user events based on privileges

Nico created a skeleton of simulated email functionality for users

Assigned tasks for following days:

Jesse: create an interface for Controller to manage resources. Test our DB on ENCS server

Ragith:

Ivan: Finalize event management functionality (update, finalize). create interface for Event manager to view/update/add participants

Samie:

Nico: continue development and integration of email functionality (edited)

<https://docs.google.com/spreadsheets/d/1AbX0hShdtF0M6jEbyUjIxsKYzEfkv-nlDJT3l8WOhSE/edit#gid=342039554>

Our progress sheet by the end of the project

SCC development and integration tasks				
Task	Assigned to	Status	Deadline	Notes
Create/Delete/Edit/Display participants of an event	Ivan	Done	1 Dec	
Create/Archive/Edit/Display an event	Ivan	Done	27 Nov	
Create/Delete/Edit/Display group members	Ivan	Done		
Create/Delete/Edit/Display a group	Ivan	Done		
Develop user home page template	Siamak	Done		
Controller interface: set the charge rate for usage of the system	siamak	Done		
Email integration: email send when a user has been added, and when new post added to an event	Nico	Done		
Email inbox, email table, email page, email viewing	Nico	Done	Nov 30	
Email sending functionality	Nico	Done	Nov 30	
An event's participant request to join a group	Ragith	Done		
Member's ability to withdraw from a group	Ragith	Done		
Member's ability to post: text, images, videos (Events and Groups)	Ragith	Done	29 Nov	Will have to modify ER
Member's ability to view and comment another post (Events and Groups)	Ragith	Done	29 Nov	
Content permission: each post has 3 permissions: view only, comment, add (Events and Groups)	Ragith	Done	29 Nov	
Group privacy: only members belongin to a group can view its contents	Ragith	Done		
Member's main page: display latest posts from his groups (similar to fbook) (Events and Groups)	Ragith	Done		
Private Instant Messages between members of an event and group	Siamak	Done		
Report of groups or members by category: filter by interests, age, profession, region, etc.	Jesse	Done		key:value ; keyValue
Event polls: ability to create a poll to vote on event's date, time and place	Jesse	Done		
Create a report of project documentation				
Support Paypal	Jesse	Done		
User redirection	Jesse	Done		

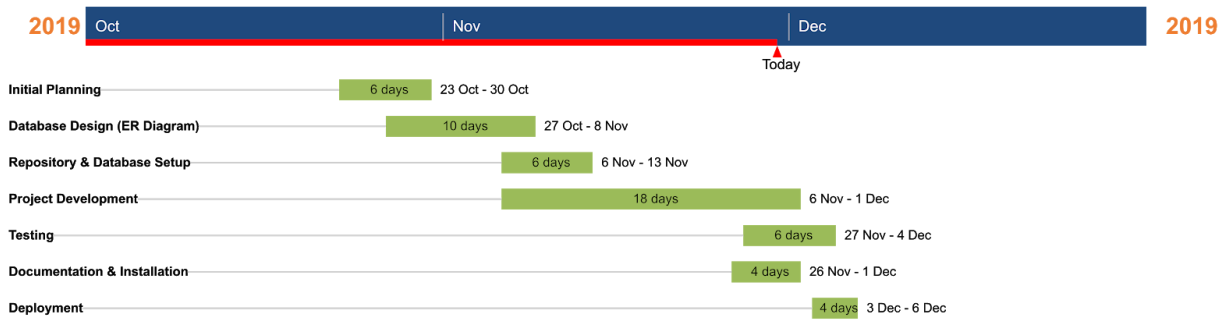
SCC Report Table of Contents				
Topic	Assigned to	Status	Notes	
Project description	Nico	Done	Will constantly add to my parts daily, to have more content	
The assumptions	Ivan	Done	Feel free to add your own assumptions	
The limitations	Ragith	Done		
Applications supported	Nico	Done		
Architectural design	Ragith	Done		
ER diagrams	Nico	Done		
Relational DB design	Ivan + Nico	Done	Ivan - updated diagram as of dec 3rd 12:30 pm	
3NF	Siamak + Nico	Done	If anybody altered the ER after this time, let me know and I will update the image	
			Added to doc	
Member responsibility	Ivan + Nico	Done	Nico - added diagram as of dec 1st, Ivan added more content and updated diagram as of 3rd December	
Gantt chart	Ragith	Done	Need to elongate text	
Detailed analysis of code			this isn't needed for the first draft	
UI design rationale	Nico	Done		
Queries	Ivan	Done	Add screenshots of your queries guys, with some simple explanations	
User Manual	Ivan	Done		
General info				
Authorized use permissions	Ragith		I guess System Admin, Event admin, Controller? Yeah I will also talk about we're handling use permissions with the user-role relation	
System summary	Nico	Done		
Overview of the user interface				
Home Page	Jesse	Done		
Visitors	Jesse	Done		
Members	Jesse	Done		
Administrator	Jesse	Done		
Gitlog	Ivan	Done		
Database				
Script to create and populate DB	Jesse	Done		

For latest update please via link above.

Gantt chart



The project has 5 main phases. These phases are planning, requirement elicitation, development, testing and documentation. The planning phase started on 2nd of October. For about 1 week. On October 13, the requirement elicitation phase began. In this phase reading the provided documents and discussion with the stakeholders were done to gather details about the project. These requirements were written down in a document. These requirements later were taken into consideration during the development cycle. The development cycle lasted about a month. After which testing was performed for a week. After testing documentation was performed for about a week.



Software Development Schedule

The software development schedule is shown above. We separated the software development schedule into 7 main phases. Each phase lasting about a week except the development phase which lasted about 3 weeks. We were able to follow this schedule. Our team members consult with each other about changes due to unforeseen circumstances and the schedule was updated in such situations.

Detailed analysis of code

For the draft we included an example of our code analysis for one of our earliest parts. We will have all code analyses in the final report.

Instant Messaging:

- Front end

In the front-end we created post container to hold all messages and a form to write a new message.

When the page loads, we immediately call a function called that makes an axios request to the backend (passing the group_id) that retrieves json of all the associated messages along with the user who posted it and the time they posted it. We then iterate over the array and append an html block that displays the information inside the post container. From inside the method, we recursively call itself after 1 second as to consistently see new posts without having to refresh the page.

When posting a new message using the form, we call a function that creates and axios request to the backend where we pass in the message and the group_id.

- Back end

Get Messages:

We have a php called “get_messages.php” page that generated a json of objects containing the following information:

- ❑ First_name
- ❑ Message
- ❑ Time

To get this information, we joined the tables instant messages and users using the ‘user_id’ in each table. And the selected only the rows that matches the group_id with the group_id provided. (sql below)

```
$sql = "SELECT * FROM instant_messages m LEFT JOIN users u on m.user_id =  
u.user_id WHERE group_id = $group_id";
```

Using the result of this query, we made an associative array with only first_name, message, and time. And then printed it as JSON

Add messages

We have a php called “add_messages.php” page that inserts the group_id, user_id, and the message that the user inputted into the instant_messages table.

Fairly straight forward, the only logic we had to include was a line to decode the POST from axios since we receive it as a json.

To insert into the table we used the following query:

```
$sql = "INSERT INTO instant_messages (group_id, user_id, message) VALUES  
($group_id, $user_id, '$message')";
```

Events

The system admin is the first user who creates the event.

backend/eventMaker.php

This script is called to insert new records for the event functional dependencies, getting the new inserted id of each record and using them as foreign keys in the new event. As well, it creates a new group, which is the default or primary group linked to the event itself. It also inserts the event manager as the group admin and group member of the primary group.

```
$sql = "INSERT INTO `groups` (event_id, group_manager_id, name, details) VALUES ($event_id, $event_manager, $event_name, 'Main event group');"
```

```
if($conn->query($sql) === TRUE) {  
    $group_id = $conn->insert_id;  
}
```

```
$insert_event_admin = "INSERT INTO group_members (user_id, group_id, participant_status_id)  
VALUES ($event_manager, $group_id, 1)";
```

```
$sql = "UPDATE events SET primary_event_group_id = $group_id WHERE event_id = $event_id";
```

backend/update_event.php

This script is called from two distinct stages. It is called when the event is still archived and the event manager is setting up all the details and finalizes the event. And once the event is finalized this script also handles any updates to the event itself, updating all the records to the data related to the event. It updates the event main information, the event location, the payment details, the resources.

backend/event_details.php

This file is used to fetch from the DB all the event details for a specific event ID. It loads all the columns into arrays and creates another array to hold all the labels and the column ids. This way it is easy to prepopulate the front end forms with the values.

backend/event_manager.php

This file is used to fetch all the relevant event info that is to be seen in the dashboard.php file.

frontend/event_details_page.php

This script displays a form with all the event information. Each section is related to the different table that has a relationship with the Foreign Keys of the event. If a user who is not the event manager of the corresponding event is viewing this page, then he can only view the data in

disabled fields and the participants. If the event manager is viewing the event, the form displays all the action buttons related to the event.

Participants

Participants are users who are linked to a group. Several actions are used throughout the platform. When a new event is created and users are imported from a CSV file, they become participants of the group linked to the event. An event manager or a group manager can perform the regular operations on each participant record. A participant status can be rejected, approved, or pending.

frontend/import_users.php

This script has a button to import a csv file with the new users that will be added to an event or users table.

frontend/view_participants.php

This script displays all the group participants and their information in a tabular form. If the user is the group manager or the event manager then an edit link is displayed to edit any participant, approve him, reject him, put his status to pending, etc.

frontend/edit_participant.php

This script displays a form to update an existing group/event participant. The user information retrieved from the DB is pre-populated automatically in the fields

frontend/new_participant.php

This script a search box to search for a new participant. The user id should exist in the users table. As well, if the user with the id already is part of the event (regardless of his status) you cannot add it here. Instead, you can go directly to users page and edit his status there. When the search returns a user it populates the fields with javascript and displays the Add Participant button.

backend/users/participant.php

This file gets all the group participant data and saves the data as an array

backend/users/search_user.php

This file searches a specific user by id and populates the table if it exists and not part of the group or event.

backend/users/update_participant.php

This file updates the participant's data or adds it to an existing group

Groups

A group is generated for each event at the creation stage. This way, every event has a primary default group. In the event home page there is a link in the menu to create a new group. As well the first option is a button to view group details or leave a group. Following is the list of current existing groups of the event. Viewing group participants, editing participant info and adding a new participant to the group has exactly the same functionality as for the events. To remove a group the group manager needs to go to group details and change the status to archived, and click update. It will automatically be removed from the event home page. A user who is a participant of the event but not a participant of the group won't be able to access the groups contents. A request to join button will appear, giving the chance to the participant to request the group manager to be added to the group. Next, the user will be added to the group with a pending status. The group admin will receive an email with the new request. The user will only be able to access the group once the group admin changes the participant status to approved.

backend/groupMaker.php

This script inserts a new group linked to the current event. It inserts all the members that were selected by the user into the group members table. Thus these users become group members at the creation of a group. The user can also not adding any member yet, however he becomes the group manager and the only group member of this group.

backend/get_event_groups.php

This script gets all the event groups and saves an array of them.

backend/group_details.php

This script gets all the group details and saves it into an array

backend/update_group.php

This script updates the group information

Authorization

Authorization uses a hierarchical series of php scripts to determine access

backend/authorize.php

The first layer determines if the user is logged in, this must be established first before any other authorization can happen. It's a simple session check. No active session, we redirect you to the login.

```
<?php
```

```
if(!isset($_SESSION)) {
```

```

        session_start();
    }

    if(!isset($_SESSION['active_user'])) {

        $_SESSION['error'] = "no_login";
        header('Location: ../frontend/index.php');
    }
?>

```

Backend/authorize_event.php

This is used for server side functions only allowed to an event manager of a specific event. This module first calls the root authorization and assuming nothing was redirected we begin to search for an event id. If none is found we consider it a failed call and redirect back to dashboard. A silent authorization flag can be sent if we want to check permissions without redirection (For example nav bar uses this to determine which options to display it cannot redirect)

```

if(isset($_GET["event_id"])) {
    $event_id = $_GET["event_id"];
}
else if(isset($_POST["event_id"])) {
    $event_id = $_POST["event_id"];
}
else {
    if(!isset($silent_auth)) {
        echo "No event id was provided";
        exit();
    }
}

```

If an event_id is found and the current user is found to not be an event manager or an admin we either redirect or ensure that no permission flag is set.

```

if($permission == 0) {
    if(!isset($silent_auth))
        header("Location: ../frontend/dashboard.php");
    if(isset($is_event_manager))
        unset($is_event_manager);
}
else
    $is_event_manager = $event_id;

```

backend/authroize_group_report.php

Similar in function to the above the following script looks for a group id and if one is found it will search to see if the current user is an event member for the event associated with the group. If they are they have permission to view that events group information, otherwise we redirect

them.

```
if(isset($_GET["group_id"])) {
    $group_id = $_GET["group_id"];
}
else if(isset($_POST["group_id"])) {
    $group_id = $_POST["group_id"];
}
else {
    echo "No Group id was provided";
    header("Location: ../frontend/dashboard.php");
}

if ($permission == 0) {
    $sql = "SELECT COUNT(*) AS 'is_admin' FROM user_roles WHERE user_roles.user_id =
$user_id";
    $is_admin = $conn->query($sql)->fetch_assoc()['is_admin'];
    if($is_admin == 0) {
        header("Location: ../frontend/dashboard.php");
    }
}
```

Backend/Authorize_group_member.php

Once again similar to the previous two examples we search for a group id and validate if the current user is a member of the group they wish to access. If they are not or if they supplied an invalid group we redirect them back to their dashboard

```
if(isset($_GET["group_id"])) {
    $group_id = $_GET["group_id"];
}
else if(isset($_POST["group_id"])) {
    $group_id = $_POST["group_id"];
}
else {
    echo "No Group id was provided";
    exit();
}

$permission = $conn->query($sql)->fetch_assoc()['permission'];

if ($permission == 0) {
    header("Location: ../frontend/dashboard.php");
}
```

Backend/ authorize_admin.php, authorize_controller.php, authorize_god.php

The authorization for admin is the same call checking for the level of roll, before we call the actual admin check these 3 scripts will set the respective admin level we are looking for

Authorize_admin.php

```
<?php
$admin_type = 0;
require("authorize_admin_type.php");
?>
```

Authorize_controller.php

```
<?php
$admin_type = 1;
require("authorize_admin_type.php");
?>
```

Authorize_god.php

```
<?php
$admin_type = 2;
require("authorize_admin_type.php");
?>
```

backend/authorize_admin_type.php

This script will validate the role previously set, if we find the current user has a user_roll row that has a role greater than what we're looking for we know they have sufficient permissions. If we set the auth to silent we will set the permission flag with the role level otherwise we redirect the user to their dashboard.

```
if ($permission == 0) {
    if(!isset($silent_auth))
        header("Location: ../frontend/dashboard.php");
}
else
    $is_admin = $admin_type;
?>
```

Login

Frontend/index.php

The actual login page is pretty simple and mostly html and styling. The only server logic is a handle that detects if a user sessions already exists, if it does we forward them to their dashboard.

```
session_start();
if(isset($_SESSION['active_user']))
{
    header("Location: dashboard.php");
}
```

```
}
backend/authenticate.php
```

We find a user matching the login and if 1 and only 1 is found (More than one we assume some error must exist with this account.) if the user is found we use the PHP password_verify function to validate that the passwords match. Password_verify automatically checks the bcrypt hash which is the hashing method we use for passwords. if the password matches we store the user information in the session for quick access, and forward them to the dashboard page.

```
if ($result->num_rows == 1) {

    while ($row = $result->fetch_assoc()) {

        if(password_verify($pass,$row['user_pwd']))
        {
            session_start();
            $_SESSION['active_user'] = $row;
            header('Location: ../frontend/dashboard.php');
        }
    }
}
```

backend/logout.php

The session is destroyed and the user is sent to the login page

```
<?php
session_start();
if(isset($_SESSION['active_user']))
{
    unset($_SESSION['active_user']);
}
header("Location: ../frontend/index.php"); ?>
```

Polls

frontend/Poll.php

When we retrieve the poll for the current group we first explode the options string. The options string is a “,” delimited string that contains all the selectable options of the poll. We next must check if the poll is still active by calculate it's end dates epoch time value and comparing it to current epoch time value.

```
while ($poll_data = $polls->fetch_assoc()) {

    $options = explode(";", $poll_data["options"]);
    $optionCount = 1;
    $today = strtotime("now");
```

```

$end = $poll_data["end_date"];
$poll_id = $poll_data["poll_id"];

if ($today < strtotime($end)) {
    $stillAvailable = true;
} else {
    $stillAvailable = false;
}

```

Next we attempt to retrieve any existing selections made by the current user. If the user has any records in the poll results table corresponding to the current poll (via poll_id) or if the poll has expired we will opt to show the options as text with the current results of the poll. Then if the current user has a selection we will display their previous selection in bold. If the poll is not yet ended a button will be placed to allow the user to actually remove their option allowing them to select a new one.

```

if ($result->num_rows > 0 || !$stillAvailable) {
    echo "<h6 class='font-weight-bold'>Poll results:</h6>";
    $our_selection = $result->fetch_assoc()["option_selected"];
    foreach ($options as $option) {

        if ($optionCount == $our_selection) {
            echo "<p class = 'bg-light p-2' rounded><b>" . $option . "</b>";
        } else {
            echo "<p class = 'bg-light p-2 rounded'>" . $option;
        }

        $sql = "SELECT count(option_selected) as votes FROM `poll_results` WHERE
option_selected = " . $optionCount++;
        $vote_count = $conn->query($sql)->fetch_assoc()["votes"];
        echo "<span class='badge badge-secondary p-2 float-right'>" . $vote_count .
"</span></p>";
    }

    if ($stillAvailable) {
        echo "<div class='alert alert-warning' role='alert'>Ends in : " . $end . "</div>";

        echo "<form action='../backend/poll_change.php' method='POST'>
            <input type='hidden' name='id' value ='$poll_id' />
            <br> <input class='btn btn-primary' type='submit' value='Change
Vote'></form></div></div>
            ";
        }
    }
}

```

If no options exist, we display each option as an input radio button. Without displaying the result values for the options. Note polls must have more than 1 option (this is also enforced at creation)

```

else {
    if (count($options) > 1) {
        echo "<form action='../backend/poll_submit.php' method='POST'><ul
class='list-group'>";
        foreach ($options as $option) {
            echo "<li class='list-group-item m-0'><input class='mr-1' type='radio'
name='vote' value='" . $optionCount++ . "'/> $option </li> ";
        }
        echo "</ul></div>

        <div class='card-footer text-xs-center'>
            <button type='submit' value='Submit' class='btn btn-primary btn-block
btn-sm'>Vote</button>
        </div>
        <input type='hidden' name='id' value ='$poll_id' />
        </form></div>";
    }
}

```

Finally we apply a silent authorization check, if the user is the event manager they can actually remove the poll and a button will be placed allowing them to do so.

```

$silent_auth = true;
include "../backend/authorize_event.php";
if (isset($is_event_manager)) {
    echo "<form class='text-center' action='../backend/poll_remove.php' method='POST'>";
    echo "<input type='hidden' name='id' value ='$poll_id' />
        <input class='btn btn-secondary' type='submit' value='Remove Poll'>
    </form> ";
}

```

frontend/Create_poll.php

Since we have certain requirements for polls on the creation page we need to have a little bit of fancy javascript to help handle this. The first functions we need allow us to dynamically add and remove input fields for the list of options that will be added. Each input field has a required attribute which in html5 means that a field cannot submit without it having a value. The removeOption will never allow there to be less than 2 option fields (polls must have more than 1 option) and since these fields are all required a poll cannot be created without at least 2 options.

```

<script>
function addOption()
{
    var optionCount = document.querySelectorAll(".option").length;
    var newOption = document.createElement("input");
    newOption.type = "input";
    newOption.classList.add("option");
    newOption.name = "option" + optionCount;
}

```

```

        newOption.placeholder = "option" + (optionCount + 1);
        newOption.required = true;
        document.querySelector(".option-list").appendChild(newOption);

    }

    function removeOption()
    {
        var optionList = document.querySelectorAll(".option");
        if(optionList.length > 2)
        {
            var option = optionList[optionList.length - 1];
            option.parentNode.removeChild(option);
        }
    }
}

```

Once the dom is loaded we place a listener to capture the submit event of the form. When the form is submitted we take each option input and concatenate it into a single “,” delimited string. We then store that string in a hidden input field to allow it to be pushed to the POST.

```

function onSubmit(e)
{
    var optionString = "";
    document.querySelectorAll(".option").forEach( (item) => {
        optionString += item.value + ",";
    });

    optionString = optionString.substr(0,optionString.length - 1);
    document.querySelector("#option_list").value = optionString;
    //e.preventDefault();

}

document.addEventListener("DOMContentLoaded",function(){

    document.querySelector("form").addEventListener('submit', onSubmit);

```

Reports

frontend/Event report.php

Since an event is just a collection of groups the event_report just serves as a collection of group reports with an option to search through groups with the use of iframe to call the group search

```

<?php
include "navbar.php";
require "../backend/get_event_report_info.php";

include "group_report.php";

?>

```



```
<iframe id="searchView" frameborder="0" width="800" height="300"
src="../backend/group_search.php?event=<?php echo $event_id ?>&primary=<?php echo
$event_primary_group_id ?>"></iframe>
```

frontend/Group_report.php

Similarly a group report just echos the group data retrieved and has a user search placed as well with an iframe.

```
<?php
require "../backend/get_group_report_info.php";
?>
<iframe id="search_view_user" frameborder="0" width="800" height="200"
src="../backend/user_search.php?group_id=<?php echo $group_id ?>"></iframe> <br>
```

The group data displays some statistics for the group like the first and last posts dates as well as the date(s) with the most activate if there is more than one it displays each

```
<ul class="list-group">
  <li class="list-group-item active">Post Statitics</li>
  <li class="list-group-item">First Post posted on : <?php echo $first_post_date ?></li>
  <li class="list-group-item">Latest Post posted on : <?php echo $latest_post_date ?></li>
<?php
if (count($most_active_dates) > 1) {
  echo "<li class='list-group-item active'>Most active dates</li>";
} else if (count($most_active_dates) == 1) {
  echo "<li class='list-group-item active'>Most active date</li>";
}
?>
<?php
foreach ($most_active_dates as $date) {
  echo "<li class='list-group-item'>$date</li>";
}
?>
```

backend/get_group_report_info.php

Once we retrieve the highest frequency post dates we store them all in an array (to be filtered through as required by whoever need sit

```
$highest_frequency = $row['frequency'];
do {
  if ($row['frequency'] == $highest_frequency) {
    array_push($most_active_dates, $row['most_frequent']);
  } else {
    break;
  }
} while ($row = $result->fetch_assoc());
```

backend/user_search.php

We iterate through each user that matches our search parameters and display their post and comment activity count.

```
while($row = $group_members->fetch_assoc()) {
    $user_id = $row["user_id"];
    $sql = "SELECT upload_date FROM posts where user_id = $user_id ORDER BY `upload_date`
DESC limit 1";
    $last_post_date = $conn->query($sql)->fetch_assoc()["upload_date"];
    $sql = "SELECT count(*) AS 'posts' FROM posts where user_id = $user_id AND group_id =
$group_id";
    $post_count = $conn->query($sql)->fetch_assoc()["posts"];
    $sql = "SELECT Count(*) AS 'comments' FROM comments JOIN posts ON posts.post_id =
comments.post_id WHERE comments.user_id = $user_id AND posts.group_id = $group_id";
    $comment_count = $conn->query($sql)->fetch_assoc()['comments'];
    echo $row["first_name"] . " " . $row["middle_name"] . " " . $row["last_name"] . "<br>";
    echo "**last posted on : $last_post_date <br>";
    echo "**Has a total of $post_count posts<br>";
    echo "**Has a total of $comment_count comments<br>";
}
```

backend/group_search.php

Similar to user for each group we find we create a link that takes the user to that specific group report.

```
while($row = $event_groups->fetch_assoc()) {
    echo "<a target='_blank' href='../frontend/group_report.php?group_id=" .
$row["group_id"]."'> Get report for : ". $row["name"]. "</a><br>";
}
```

Admin

frontend/add_admin.php

When the user arrives at this page they will first be prompted to search for a user. Once they enter a user and submit the form the page will reload with the user information stored to the pages GET variable. If this user data is found on this page it will display the users admin status (0 for regular user, 1 for admin , 2 for controller, 3 for system admin)

```
<?php
if(isset($searched_user_id)) {
    echo " <form action='../backend/add_admin.php' method='POST'>";
    echo "Update Admin status for user: $user_name <hr>
        Enter new Admin status:<div class='form-box'>";
```

```

        echo "<input type='number' value= '$role' name='role' min='0' max='3' required/>";
        echo "<input type='hidden' value= '$searched_user_id' name='user_id' required/> ";
        echo "<input class='btn btn-primary' type='submit' value='Update'/></div>";
    }
    else {
        echo "<form action='#' method='get'>";
        echo "Please enter the User's ID for new privileges :<br><div class='form-box'>";
        echo "<input type='text' name='user_id' required/> ";
        echo "<input class='btn btn-primary' type='submit' value='Search'/></div>";
    }
?>

```

If we find a user in the admin table we retrieve their name information and the current admin roll

```

        if($result->num_rows > 0) {
            $row = $result->fetch_assoc();
            $user_name = $row["first_name"] . " " . $row["middle_name"] . " " .
$row["last_name"];
            $role = $row["role_id"];
        }

```

Otherwise we retrieve the same information and set the role to 0

```

        $row = $result->fetch_assoc();
        $user_name = $row["first_name"] . " " . $row["middle_name"] . " " .
$row["last_name"];
        $role = 0;

```

backend/add_admin.php

We retrieve the list of admins for the user whose status we want to change. If the user exists and the role is 0 we delete them from the user_role table, otherwise we update the roll assigned to them. If they do not exist the table and the roll change was greater than 0 we add this user to the roll table with the information posted by add_admin in the front end.

```

if($result->num_rows > 0)
{
    $row = $result->fetch_assoc();
    if($role == 0) {
        $sql = "DELETE FROM user_roles WHERE user_id = $user_id";
    }
    else {
        $sql = "UPDATE user_roles SET role_id = $role WHERE user_id = $user_id";
    }
}

```

```

    }
}
else if ($role > 0) {
    $sql = "INSERT INTO `user_roles` (`user_role_id`, `user_id`, `role_id`) VALUES (NULL,
$user_id, $role)";
}

```

Tags

frontend/tags.php

Whenever we call tags we include it as iframe. To keep tags modular we add a type param to let the tags script know if it updating a group or a users tags.

```

if($type == "group")
{
    if(!isset($_GET["group"]))
        exit();

    $group_id = $_GET["group"];
    $sql = "SELECT meta_data AS 'tags' FROM `groups` WHERE group_id = $group_id";
    $params = "type=group&group=".$group_id;
}
else if ($type == "user")
{
    if(!isset($_GET["user"]))
        exit();

    $user_id = $_GET["user"];
    $sql = "SELECT meta_data AS 'tags' FROM users where user_id = $user_id";
    $params = "type=user&user=".$user_id;
}

```

Tags is another file which requires some fancy javascript to work. We list each tag as a button and onclick a user can remove a tag, or type a tag in a form and submit to add a tag to the list Either way we must parse through the list of all tags on the page and create a “,” delimited list push that to the add tag

```

function removeTag(name)
{
    var tagString = "";
    document.querySelectorAll(".remove-tag-btn").forEach( (tag) => {
        if(tag.name != name)
            tagString += tag.value + ",";
    });
}

```

```

        tagString = tagString.substring(0, tagString.length - 1);
        window.location.replace("../backend/tag_add.php?tag="+tagString+"&<?php echo
$params?>");
    }

    function addTag()
    {
        var tagString = "";
        document.querySelectorAll(".remove-tag-btn").forEach( (tag) => {
            tagString += tag.value + ";";
        });

        tagString += document.querySelector("#new-tag").value;
        window.location.replace("../backend/tag_add.php?tag="+tagString+"&<?php echo
$params?>");
    }

```

UI design rationale

Minimalistic, simple design that allows users to access all levels of the website once logged in. This is achieved through a navigation bar. Separation of user concerns is achieved through pages. User actions applied to event occur on the event page, group actions occur of the group page, email actions occur of the emails page, etc. Since groups are children of events access to them will be hierarchical. In order to see group posts you must select a group from the events post page. To see a groups reports, you must do so from selecting the group in the event reports page and so on.

Simple styling is achieved through html tags and css 3. Optionally some pages include bootstrap tags, which are only simple css styles that are applied through class names.

Queries

Since the system basic functionality is centered around events and users, the team begin developing the queries and functionality starting from events and users table. From the start, we encountered a challenge. We created the tables: events_locations, event_payment, event_types, resources. All of these tables were linked to the events table with foreign keys in the events table. Therefore, there were many functional dependencies linked to the events table, and we couldn't create a new event record without the foreign keys already existing in all the dependent tables (otherwise there would be a violation of Foreign Key child does not exist in dependent table). Hence, we had to think of a way to insert a new record in the events table before creating records for location, payment and resources. We came up with the idea of adding the following trigger:

```
/* This trigger creates a new row for each of the tables that have a functional dependency on Event.
Before inserting a new record in Events table, we need the foreign keys of event payment, event type, resources and event location
In order to be able to create a new row without violating foreign key dependency, this trigger creates a set of empty rows and gets the new ids
and then inserts these as foreign keys.
The new rows are empty for now, but will be used later, when event manager updates the event info*/

DELIMITER $$
CREATE TRIGGER `CreateEventForeignKeys` BEFORE INSERT ON `events`
FOR EACH ROW BEGIN
    INSERT INTO `event_payment` (`account_holder`, `address`, `phone_no`, `bank_name`, `account_number`, `payment_method`) VALUES (NULL, NULL, NULL, NULL, NULL, NULL);
    SET NEW.event_payment_id = LAST_INSERT_ID();
    INSERT INTO `resources` (`flat_fee`, `type`, `unit`, `resource_name`, `extra_fee`, `discount`) VALUES (NULL, NULL, NULL, NULL, NULL, NULL);
    SET NEW.resource_id = LAST_INSERT_ID();
    INSERT INTO `event_locations` (`address`, `phone_no`, `email`, `room_no`, `renting_cost`, `capacity`) VALUES (NULL, NULL, NULL, NULL, NULL, NULL);
    SET NEW.location_id = LAST_INSERT_ID();
END
$$
DELIMITER ;
```

This trigger worked very well. Right after the system admin clicks the button 'Create event' and the insert query would execute, this trigger would be invoked and generate new empty records on the tables, getting the inserted ID of the record, and setting the foreign key in the corresponding column before inserting an event. However, when we tried to test the trigger in the AITS Database, it was not working. The Database does not have any permissions and privileges to create and use triggers. We

tried to solve this issue, we contacted Stan to see if he could grant us permissions to execute triggers. Unfortunately, he was not able to help us and we had to remove this trigger. We decided to recreate the trigger functionality using a PHP script as following:

```
$sql = "INSERT INTO event_payment (account_holder, address, phone_no, bank_name, account_number, payment_method) VALUES (NULL, NULL, NULL, NULL, NULL, NULL)";  
  
if($conn->query($sql) === TRUE) {  
    $event_payment_id = $conn->insert_id;  
}  
  
$sql = "INSERT INTO resources (flat_fee, type, unit, resource_name, extra_fee, discount) VALUES (NULL, NULL, NULL, NULL, NULL, NULL)";  
  
if($conn->query($sql) === TRUE) {  
    $resource_id = $conn->insert_id;  
}  
  
$sql = "INSERT INTO event_locations (address, phone_no, email, room_no, renting_cost, capacity) VALUES (NULL, NULL, NULL, NULL, NULL, NULL)";  
  
if($conn->query($sql) === TRUE) {  
    $location_id = $conn->insert_id;  
}  
  
$sql = "INSERT INTO events (event_payment_id, event_type_id, resource_id, event_manager_id, location_id, event_name, status, total_cost)  
VALUES ($event_payment_id, $event_type_id, $resource_id, $event_manager, $location_id, $event_name, 2, $event_fee)";
```

This PHP script achieved the same functionality that the trigger had. The same issue came into discussion when we modified our ER diagrams to link a default group when creating an event. Hence, we insert the event record first, and then we create a group and use the inserted ID to update primary_event_group_id in the events table:

```
$sql = "INSERT INTO groups (event_id, name, details) VALUES ($event_id, $event_name, 'Main event group')";  
  
if($conn->query($sql) === TRUE) {  
    $group_id = $conn->insert_id;  
}  
  
$sql = "UPDATE events SET primary_event_group_id = $group_id WHERE event_id = $event_id";
```


A notable SQL query we wrote had to do with getting the total payment. This is because we needed some values across 5 tables. The payment process needed to know what type of event was, whether or not the event was recurring, the flat fee, the extra fee, the discount, and the system charge rate. To achieve this we had to:

- Left join groups and events on the event id. Call it table T1
- Left join events and event_types on the event_type_id. Call it table T2
- Then we left join T1 and T2 on the event ID
- We left join resources with system_charge_rate on the system_charge_rate_id. We call it table T3
- Finally we left join t1 and t3 on the resource_id
- We select only the parameters we want which are type, recurrent, flat_fee, extra_fee, discount, charge_rate

SQL query below:

```
SELECT type,
       recurrent,
       flat_fee,
       extra_fee,
       discount,
       charge_rate
FROM   ( (SELECT e.event_id,
                e.resource_id,
                g.event_id AS event_id2
        FROM   `events` e
              LEFT JOIN `groups` g
                    ON e.event_id = g.event_id
        WHERE  group_id = $group_id) t1
      LEFT JOIN (SELECT e2.event_id,
                        e2.event_type_id,
                        e2.resource_id,
                        et.event_type_id AS event_types2,
                        et.type,
                        et.recurrent
        FROM   `events` e2
              LEFT JOIN `event_types` et
                    ON e2.event_type_id =
et.event_type_id) t2
      ON ( t1.event_id = t2.event_id ) )
LEFT JOIN (SELECT r.system_charge_rate_id,
                  r.resource_id,
                  r.flat_fee,
```

```

        r.extra_fee,
        r.discount,
        s.system_charge_rate_id AS
system_charge_rate_id2,
        s.charge_rate
FROM    `resources` r
        LEFT JOIN `system_charge_rate` s
            ON r.system_charge_rate_id =
                s.system_charge_rate_id)
        t3
ON ( t1.resource_id = t3.resource_id )

```

Another instance where we use the join keyword is with instant messaging. The query is much more simple:

```

SELECT *
FROM    instant_messages m
        LEFT JOIN users u
            ON m.user_id = u.user_id
WHERE   group_id = $group_id

```

What we are doing is join instant messages and users on the user_id where the group id matches the group id we passed in. The reason to join these two tables is to get the user name and display it with their corresponding post

Authorization

background/authorize_event.php

We count the number of rows where the event manager id is equal to current user id and where the event id matches the event id for the current event they are attempting to access.

```

SELECT count(*) as 'permission' FROM events WHERE event_manager_id = $user_id AND event_id =
$event_id

```

If we find they are not an event manager we check if the current user exists in the user_roll table if they are we know they are admin and also are granted access.

```
SELECT count(*) as 'permission' FROM user_roles WHERE user_roles.user_id = $user_id
```

background/authorize_group_reports.php

We join groups with the event they belong to. And check to see if the user is the event manager from the current group they wish to access information from.

```
SELECT COUNT(*) AS 'permission' FROM groups JOIN events ON events.event_id = groups.event_id  
WHERE groups.group_id = $group_id AND events.event_manager_id = $user_id
```

We see if user exists in user_roll tables, if they are we know they are an admin and have access.

```
SELECT COUNT(*) AS 'is_admin' FROM user_roles WHERE user_roles.user_id = $user_id
```

background/authorize_group_member.php

We look to see if the user exists in the group member table for the current group and if their participant status is equal to 1 (active member of the group)

```
SELECT * FROM group_members WHERE group_id = $group_id AND user_id = $user_id And  
participant_status_id = 1
```

background/authorize_admin_type.php

We count the number of times the user is found in the user_roles table where the role_id is greater than the role provided by the caller functions

```
SELECT COUNT(*) AS 'permission' FROM user_roles WHERE user_id = $user_id AND role_id >  
$admin_type
```

Polls

backend/poll_add.php

A very simple query that adds the POST of the created poll into the database.

```
INSERT INTO `poll` (`poll_id`, `group_id`, `title`, `end_date`, `options`) VALUES (NULL,  
$group_id, '$title', '$date', '$options');
```

backend/poll_remove.php

A very basic query to remove the poll with matching poll_id which we submit with the delete poll button in frontend/poll.php

```
DELETE FROM `poll` WHERE `poll`.`poll_id` = $poll_id
```

backend/poll_submit.php

When a user selects and submits their choice for a poll we retrieve POST values for their submission from poll.php and then store it in the database.

```
INSERT INTO `poll_results` (`poll_result_id`, `poll_id`, `user_id`, `option_selected`) VALUES (NULL, '$poll_id', '$user_id', '$selection');
```

backend/poll_change.php

When a user request to change their selection in a poll we must first validate they can do so. We select only end_date from the ID of the poll requested as it is the only data we care for.

```
SELECT end_date FROM `poll` WHERE poll_id = $poll_id;
```

If the end_date has not yet been reached we then use the same idea and search for a poll result row from the current user with the poll_id we retrieved from the post

```
Delete FROM poll_results WHERE poll_id = $poll_id AND user_id = $user_id
```

Report

backend/get_event_report_info.php

We retrieve info from events ensuring that the current user is the event manager.

```
SELECT * FROM events WHERE event_manager_id = $user_id AND event_id = $event_id
```

backend/get_group_report_info.php

We join group members with users to find the list of group members that belong to this group. We use the group_id we retrieve from the group_report page and select all users who participation status is 1 (active)

```
SELECT first_name , middle_name, last_name FROM `users` JOIN group_members ON users.user_id = group_members.user_id WHERE group_members.group_id = $group_id AND group_members.participant_status_id = 1
```

We join the groups and posts table to find the posts from the current group to retrieve the user and the date uploaded. We sort them by date and limit it to one because we just want the first post posted.

```
SELECT `posts`.`user_id`, `posts`.`upload_date` FROM `groups` JOIN `posts` on `groups`.`group_id` = `posts`.`group_id` WHERE `posts`.`group_id` = $group_id ORDER BY `posts`.`upload_date` LIMIT 1
```

We repeat this process only reversing the order we sort the dates by to get the latest post

```
SELECT `posts`.`user_id`, `posts`.`upload_date` FROM `groups` JOIN `posts` on `groups`.`group_id` = `posts`.`group_id` WHERE `posts`.`group_id` = $group_id ORDER BY `posts`.`upload_date` DESC LIMIT 1
```

We now want the dates (not date time) with the most activity so we group and count the upload date from posts in decreasing order to have the higher count on top. This lets us see the number of posts per day starting with the date with the most activity.

```
SELECT CAST(upload_date AS DATE) AS 'most_frequent' , COUNT(upload_date) AS 'frequency' FROM posts WHERE group_id = $group_id GROUP BY CAST(upload_date AS DATE) ORDER BY 'frequency' DESC
```

backend/user_search.php

We join users to group member and select the name information of the user who is an active participant in the group (participant status = 1) and where the users meta tags contains the substring we are looking for (Default is empty string).

```
SELECT first_name , middle_name, last_name, users.user_id FROM `users` JOIN group_members ON users.user_id = group_members.user_id WHERE group_members.group_id = $group_id AND group_members.participant_status_id = 1 AND users.meta_data LIKE '%$search_query%'
```

We order the posts from a specific user and order their posts by date. We retrieve the date of the latest post.

```
SELECT upload_date FROM posts where user_id = $user_id ORDER BY `upload_date` DESC limit 1
```

We count the number of posts made by the user for this specific group, and we count the number of comments. In order to count the comments we join comments with group looking ro comments on posts where the post is part of the current group and made by the specific user.

```
SELECT count(*) AS 'posts' FROM posts where user_id = $user_id AND group_id = $group_id  
SELECT Count(*) AS 'comments' FROM comments JOIN posts ON posts.post_id = comments.post_id  
WHERE comments.user_id = $user_id AND posts.group_id = $group_id
```

Admin

frontend/add_admin.php

We join users on user_roles where the user_id is the one we're looking for. This tells us if the user is currently and admin

```
SELECT * FROM users JOIN user_roles ON user_roles.user_id = users.user_id WHERE users.user_id = $searched_user_id
```

backend/add_admin.php

We look for the user i the user_role table

```
SELECT * FROM user_roles WHERE user_id = $user_id
```

We delete the user from the user_role table

```
DELETE FROM user_roles WHERE user_id = $user_id
```

We update the user roll for the user we want to update in the user_role table

```
UPDATE user_roles SET role_id = $role WHERE user_id = $user_id
```

We add a new user to the user_roll table with the roll we wish to give them.

```
INSERT INTO `user_roles` (`user_role_id`, `user_id`, `role_id`) VALUES (NULL, $user_id, $role)
```

User Manual

Logging in

Users will arrive at our index page and be prompted to login or to register. They login with their userId that was provided to them in their registration email and proceed to our SCC. When you login the landing page is our dashboard. There you will be able to see the list of events you belong to, and navbar with options to create an event, view your emails or logout. This navbar will persist throughout the website.

Registration

Users may register to the website by filling out the registration form, once registered an alert will give them their initial access information from there they can login and await to be added to an event by someone, otherwise they only have access to the email client.

Event

Event Creation:

A user with System Admin privileges is the only user who can create an event. Assuming you are logged as the System admin, when you click on Create Event from the nav bar you are taken to the event creation page there you will be required to enter the required information : name, event manager, the type of event(profit, or nonprofit and recurring or not) and a fee per user. You must assign an event manager to the event, otherwise it will not be created. When the event is created, it is set as archived, since it is still not ready to appear for members to register to it. After this step, when the event manager logs in to his account, he will see the new event as not finalized. After clicking on the link, he will have the possibility to enter all the details for the event. When the event manager enters this information he can finalize at his convenience. Once the

information is finalized, the event status changes to active and then users can become members of the event.

Event Management:

The event manager has full privileges to the event he manages. He has the possibility to update the event main details, event location, event payment details, resources, event status. As well, he has the feature to import users to the system with a CSV file. Once he does that, the users will automatically become the event members of the respective event. The event manager can view all the event members, and edit any personal information related to the member. He can change the member's status to approved, rejected or pending. An approved member can view the contents and participate in the event. A rejected member does not have access to the event. A pending member is someone who requested to join an event and is waiting for approval from the event manager. The event manager has also the possibility to add individually an existing user, by searching his id.

Event Posts:

When you are on the event page itself, you can post a comment on other posts, you can upload an image and make a post yourself. on the left side polls will be displayed that are created by the event manager. You can vote on these polls by selecting an option and clicking submit. So long as the poll is still running you free to change your selection anytime.

Group

Group Creation:

When you are inside an event you will have an additional option in the navbar to create a group.

When you click it on you are taken to a group creation page, there you can choose which event members to add to this group, what to call this group, a description of the group and any tags you wish to add. Once you are finished the group will be made and you will be added to the postfeed of the group.

Group Posts:

On the group page you will see a feed of all posts ordered by date just like in the event page. Only these posts are only visible to group members.

Instant Messaging:

A user will be able to read all messages from the group instant message chat in real-time (updating every 1 second) without having to refresh the page. To participate in the chat, the user can type inside the text-area and click submit. Their will appear in the chat box in real-time as well.

Emails

Users have access to an email client on our site, where they can send and receive emails from the system or other users. The system has automated email to alert members when they have been added to a group.

Event Payment:

Event managers will have the cost of their system usage and flat fee calculated, the sys admin can request payment and payment is processed through paypal.

Even Details / Reports

The event manager can look at the report for the event. The report lists basic information about the event as well as the last post/comment date and post/comment

count for each participant. They can also see this information for each group within the event. Events and users can be sorted by tags.

Account

Users can click on the account option in the navbar in order to review and update their personal information. They may also add tags they want the system to identify them with.

Permission Granting

The sys admin can update any users admin status by going to the add admin option on the navbar. There they can first find a user by id, and then change his status from

0 normal user

1 regular admin

2 Controller

3 Sys admin

General information

This website is for educational use only, it is not meant for the general public.
Please use and consume responsibly.

Authorized use permissions

This website was not meant for commercial use. However in it open-source and can be accessed via https://github.com/ivangar/COMP_353. Since it was created for educational use only, it is not meant for the general public. Please use and consume responsibly.

The system is build on different user roles and privileges. Authorizations are as follows:

System Admin: Has the highest level of authorization. Can create events, groups, edit any member's information, archive events, add or reject users. He can post in any group or event.

Event Manager: This user is authorized to finalize events previously created by the System Admin. He has the full privileges to update event information, import users, reject members, edit member's information and create groups. Event manager can participate on his events, add/edit/remove contents.

Controller: Has authorized permission to add charge rates for the event resources.

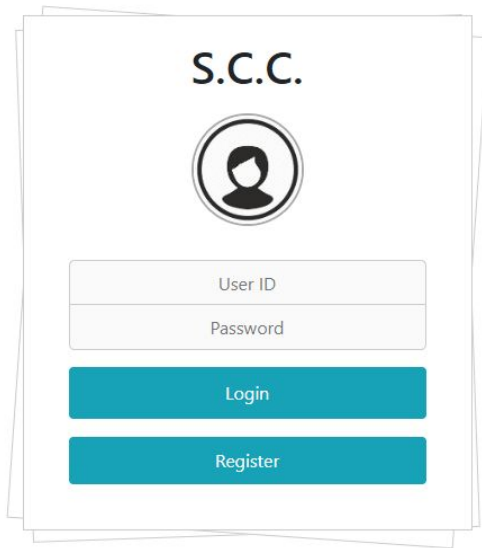
Participant: Has lower level authorizations. He can only add content to the events/groups where he has been added. He can request to join a group, but cannot access the group contents if he is not an active member.

System summary

Internal social media, used to organize events and event members. System is rudimentary and can only be used by people who have prior knowledge of the system. System contains login, however account creation is only possible through csv files. System events can hold location, resources, payment, and posts. System can create and users can view, internal emails. Event user's can create event groups, posts, and have access to emails and instant messaging.

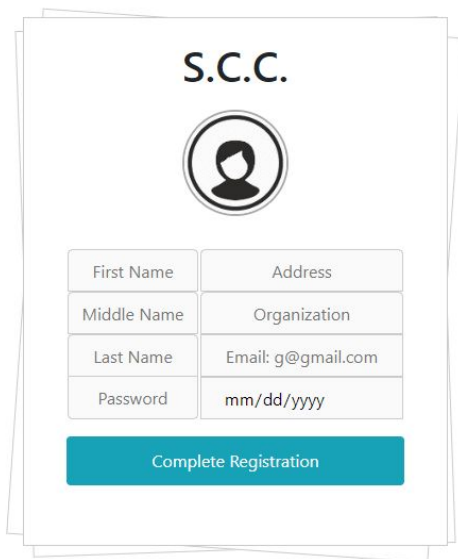
Overview of the user interface

Login And Registration:



The login page mockup features the S.C.C. logo at the top, followed by a circular profile icon. Below these are two input fields: 'User ID' and 'Password'. At the bottom are two teal buttons: 'Login' and 'Register'.

Login page is a simple centered square, where the use may enter or register.



The registration page mockup features the S.C.C. logo at the top, followed by a circular profile icon. Below these are six input fields arranged in a 3x2 grid: 'First Name', 'Address', 'Middle Name', 'Organization', 'Last Name', 'Email: g@gmail.com', 'Password', and 'mm/dd/yyyy'. At the bottom is a teal button labeled 'Complete Registration'.

The registration page will allow a user to login without access to anything except their their emails until they are added to an event by someone. It follows the same styling conventions as the login

Dashboard:

SCC Home My Account Create Event Add Admin Controller Dashboard Emails Logout

Welcome Sandras!

You're the Event Manager of the following events:

Event Id	Event Name	Start Date	End Date	Status		
7	Weddings celebration	2019-11-19	2020-02-26	active	view details	Make Payment
35	new event 2			active	view details	Make Payment
36	party at my ohuse			active	view details	Make Payment
48	test addin event manager			archived	Finalize Event	Make Payment
56	safddsafdsafsf			active	view details	Make Payment
57	create test event	2020-01-10	2019-12-17	active	view details	Make Payment

The dashboard is the landing page for any user. Up top we see the nav bar with the list of options available to sandra (a sys admin) and below that a list of events Sandra has access to. Since Sandra is a system admin she has access to all the events on the application.

This is the dashboard view of a regular user. Since Theo is not a sys admin, he cannot create events and only has access to events he is participating in

SCC Home My Account Emails Logout

Welcome Theo!

You're the Event Manager of the following events:

Event Id	Event Name	Start Date	End Date	Status	
57	Wedding	2020-01-10	2019-12-17	active	view details

Events:

Admin View

Enter event name

My new event

Select an event manager

Kalpdum Passi

Select event type

non-profit recurrent

Enter user fee

123

Create new Event

The form used by system admin to create new events.

Event Details

Main details

Event name

Weddings celebration

Start Date

11/19/2019

End Date

02/26/2020

Period (expires)

12/31/2020

Status

☒ active ☐ archived

Event type

non-profit ▼

Recurrent

no ▼

Total cost

5000

Location details

Event address

The event details page, when viewed by a system admin the details are input boxes whos changes can be committed by clicking the update button. All option buttons at the bottom are visible only to the event manager of the respective event. It is also visible to the system admins. Any other user without these privileges will only see a disabled form without any buttons except View Participants.

Payment amount

Update

Import users

View participants

Add new participant

View Event Report

The event manager has a list of options at the bottom of the event page where they can add users one by one or from a csv, view participants update the details or view the event report.

test group FK event participants

Participant Name	Address	Date of birth	Email	Organization	Status
Sandra Deamo	123 street street	0000-00-00	email	org	approved edit
Theo Harder	123 street street	0000-00-00	email	org	approved edit
Kalpdru Passaretti	123 street street	0000-00-00	email	org	approved edit
Shriss Ojhawe	123 street street	2019-12-11	email	org	approved edit

Any event/group member can view participants in the group and their information. If the user who is viewing the participants is the system admin or the group admin, the edit link is displayed next to each participant. He can click on the link to edit the participant or remove it from the group.

Add a new participant to your event

Please type the user id of the person
you want to add to your event

Search User

First Name	<input type="text"/>
Last Name	<input type="text"/>
Middle Name	<input type="text"/>
Address	<input type="text"/>
Date of birth	<input type="text" value="yyyy-mm-dd"/>
Email	<input type="text"/>
Organization	<input type="text"/>

The event manager can add manually a user. In order to add the user, he must search the user id, and if the user exists in the Database, the page will populate the fields automatically and the button 'Add participant' will display. If the user is already a member of your event, but his status is not active, then you will not be able to add the participant (he is already a participant of your event). In order to change his status, the

event manager has to go view all the participants, and edit the participant status to make it active.

Leave Group

Current Group Details

Groups List

create text event

Post to Wall

Write a post

what is on your mind?

Upload image

Select image/video to upload

Browse

Set Permission

view only

Post

Wall

Sandras Deamo 2019-12-10 16:56:51

post accepting comments

comment

Polls

Chat Box

Sandras 2019-12-10 16:51:17

I'm going to go buy a new dress

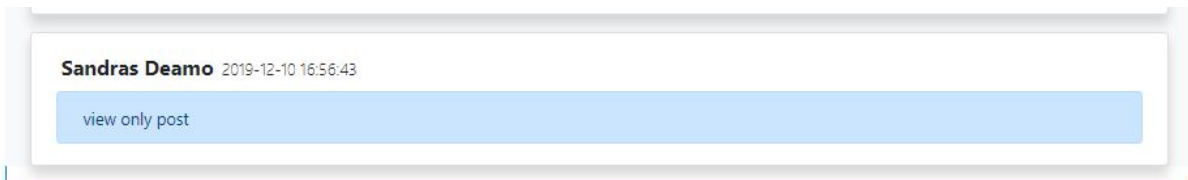
Sandras 2019-12-10 16:51:04

omg so excited for the wedding like AAAAAA

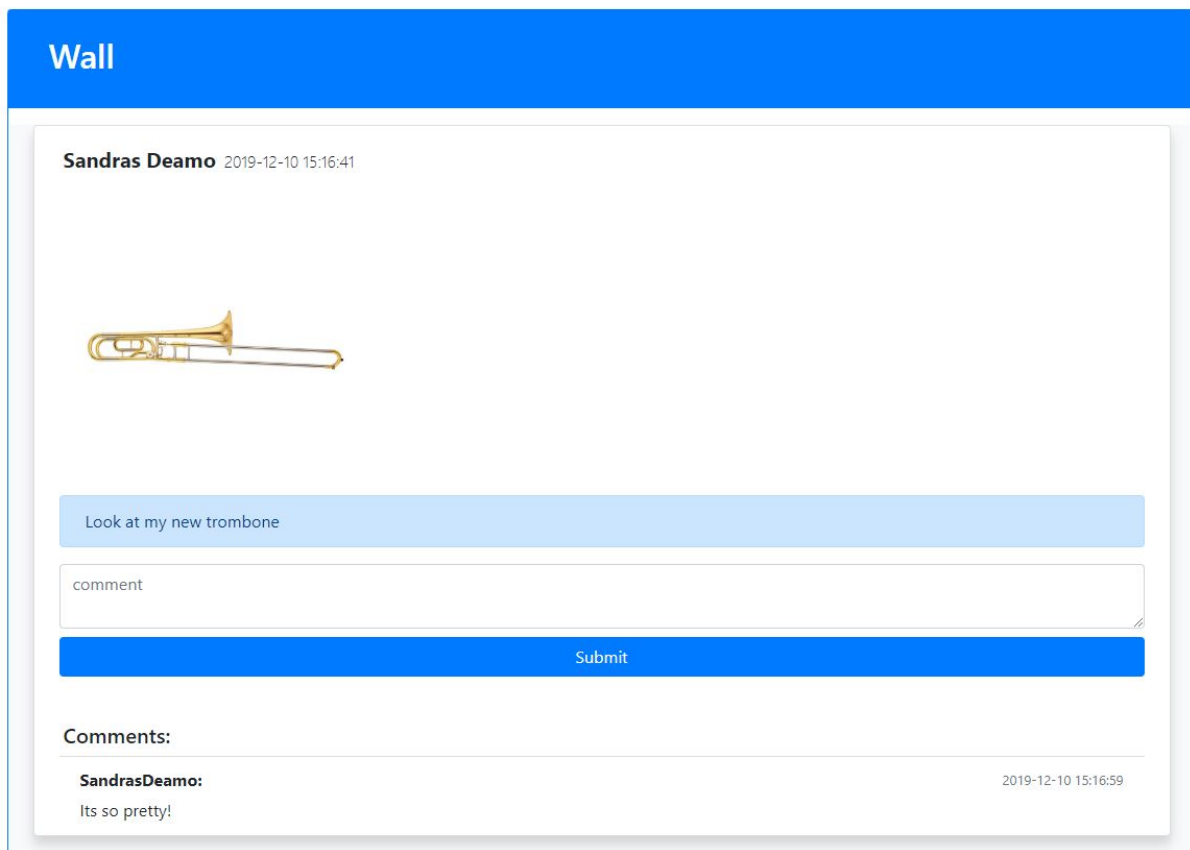
message

Send

The event page will list your current group by default it will be the event wide group. At the top you will see a box prompting you to post something, you can post text add an image or even video. You can set your post to view only, or enable comments. Towards the bottom we find polls and the chatbox for that group.



This is a post with a view only permissions. There is no option to leave a comment.



A post with comments enabled. The comment input box is visible and the comments section is shown with a comment from sandra.

Polls

Music ideas?

☐ Mary had a little lamb

☒ Monster Mash

Vote

Remove Poll

Polls appear on the page below posts each option is a radio button that can be selected and the user can cast a vote. If the user is an event manager they can see a button to remove the poll.

Music ideas?

Poll results:

Mary had a little lamb 2

Monster Mash 3

Ends in : 2019-12-19 00:00:00

Change Vote

After voting users can see their selection in bold and the time remaining. They will also be able to change their vote

Music ideas?

Poll results:

Mary had a little lamb 2

Monster Mash 2

Remove Poll

Once the poll is finished only the results will be displayed with no option to vote or change vote.

Groups:

Create Group

Group details

Group Name

Group Description

Select members of your new group

Name	Address	Date of birth	Email
<input type="checkbox"/> Sandras Deamo	123 street street	0000-00-00	sandra@email.com
<input type="checkbox"/> Hannes Voigt	123 street street	0000-00-00	email
<input type="checkbox"/> Theo Harder	123 street street	0000-00-00	email

Create group form -- Only accessible from events page, shows only names of members inside of event (user name not shown but is automatically added to new group)

Group details

Group Name

Group Details

Status ☒ active ☐ archived

manager_id

Once you create a group you can view and modify the groups details as group manager, you can also add tags to the group

Reports:

Posts information

Post Statistics
First Post posted on : 2019-12-04 20:19:57
Latest Post posted on : 2019-12-05 13:42:56
Most active date
2019-12-04

Search for a user

Users with tags containing : health
Number of users 1
Sandra Deamo
* last posted on : 2019-12-10 15:16:41
* Has a total of 3 posts
* Has a total of 0 comments

Reports show a basic event information at the top. The cost of the current events operation
The run time of the event and it's id in the system.

We can see basic statistics on the report, the first post date , the most recent post date and the date on which most posts were posted.

We have a list of users with their specific post information, which can be searched through by a tag

And below a list of the groups in the event that we can also get a report on.

Event info

Name	Value
Event ID	35
Event Name	new event 2
Event Cost	56
Start Date	
End Date	
Primary Group ID	37

Search for a group

Number of groups 3
[Get report for _party.at/Concordia](#)
[Get report for _disband](#)
[Get report for _santa gift](#)

Emails:

Inbox

Sender	Subject	Date	See email
sandra@email.com	hello	2019-12-04 19:24:45	Show Email
sandra@email.com	test	2019-12-05 13:34:30	Show Email
sandra@email.com	'fsfdh'	2019-12-05 13:41:49	Show Email
sandra@email.com	'sgsag'	2019-12-05 13:46:07	Show Email
sandra@email.com	'sag'	2019-12-05 14:01:22	Show Email
sandra@email.com	'title'	2019-12-05 14:08:39	Show Email
sandra@email.com	title	2019-12-05 14:11:26	Show Email

In the email tab users will have access to their inbox if they. The inbox contains a list of all emails showing the sender's email address, the subject the date sent and a button to see its contents, if you choose to explore its contents a modal is added revealing the email in its entirety.

Email

×

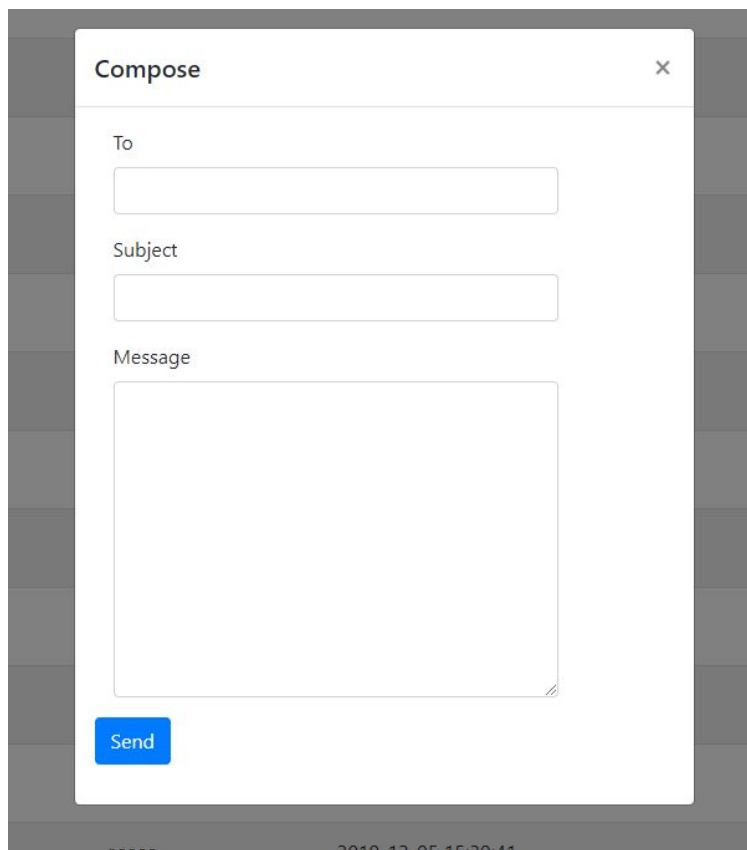
From: sandra@email.com

Date: 2019-12-04 19:24:45

Subject: hello

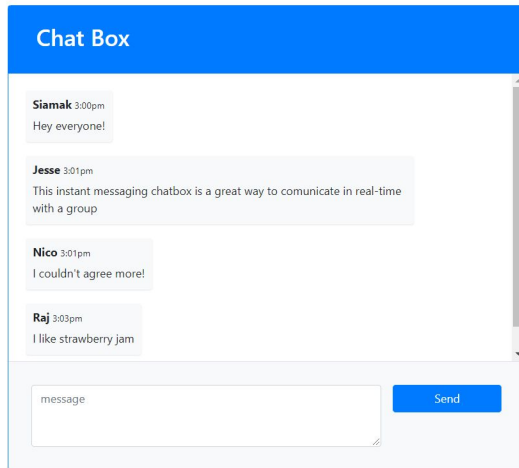
Email: its me

If the users clicks on the compose button a different modal appears to allow them to fill out an email form and send it to whoever they wish.



The image shows a 'Compose' modal window. It has a title bar with the word 'Compose' and a close button (X). Inside the modal, there are three input fields: 'To' (a single-line text box), 'Subject' (a single-line text box), and 'Message' (a multi-line text area). At the bottom left of the modal is a blue 'Send' button. The modal is set against a dark gray background.

Instant Messaging



The screenshot shows a chat box titled "Chat Box" with a blue header. It contains a list of messages from four users: Siamak (3:00pm) saying "Hey everyone!", Jesse (3:01pm) saying "This instant messaging chatbox is a great way to communicate in real-time with a group", Nico (3:01pm) saying "I couldn't agree more!", and Raj (3:03pm) saying "I like strawberry jam". At the bottom, there is a text input field with the placeholder "message" and a blue "Send" button.

The chat box will sit in a locked position on the posts page, so the chat can be quickly and easily accessed at all times while in the event. All messages are served instantly (with upto 1 second of delay).

Permission Handling

Please enter the User's ID for new privileges :



The form consists of a text input field containing the number "781264" and a blue "Search" button.

On the add admin page users can select a user by submitting an ID

Update Admin status for user: Kalpdrum Passi

Enter new Admin status:



The form consists of a text input field containing the number "2" and a blue "Update" button.

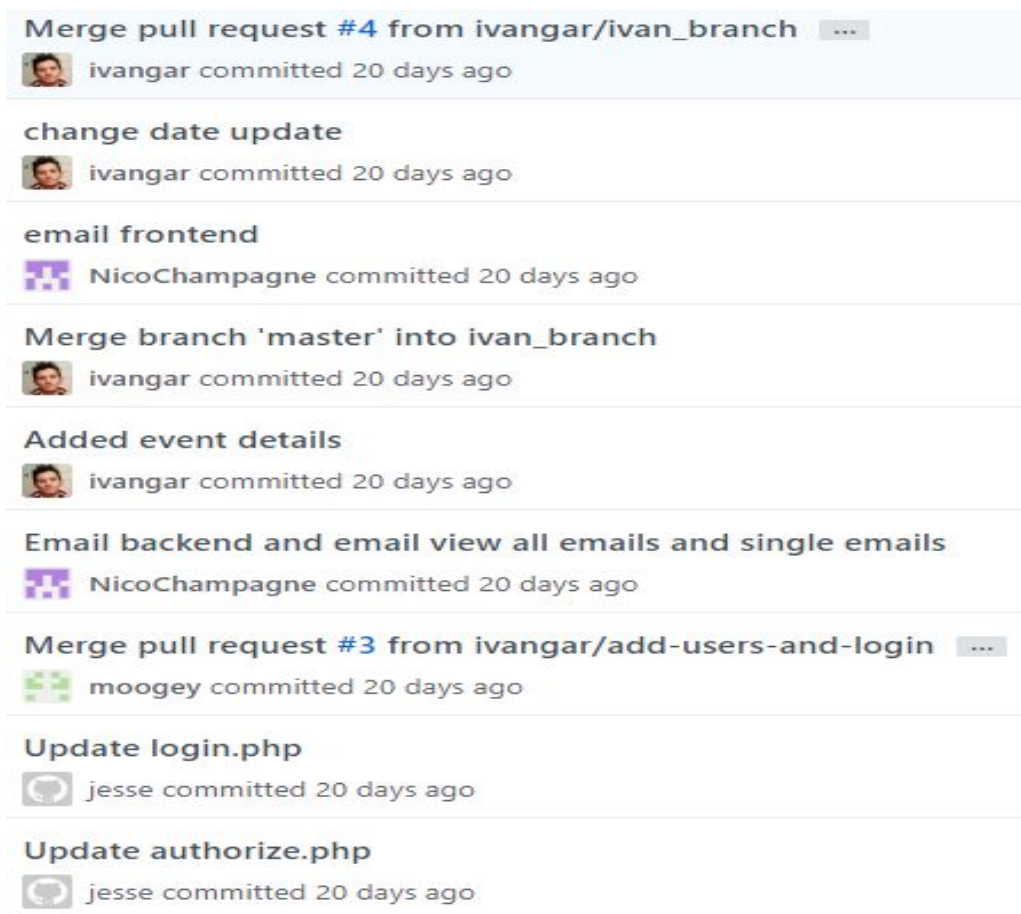
Once they find a user they can update the admin status of that user 1 for regular admin, 2 for controller and 3 for system admin

Gitlog

We created a master branch called COMP_353 in github and commit all our working source code, and files to this branch. Every team member creates a local repository on his laptop and creates a branch to work on his features. Once he's ready to push some code, he pushes to the server, creates a Pull Request and if it needs to be reviewed he asks another member to review and approve his work. If there are no conflicts he can merge into master branch and we all synch the changes from there. Locally everybody installed an XAMP environment to develop and test the code.

Github access: https://github.com/ivangar/COMP_353

Here are some screenshots of the github commit history from all the team members.



Merge branch 'master' of github.com:ivangar/COMP_353



r-saba committed 15 days ago

add dashboard files



r-saba committed 15 days ago

Merge pull request #9 from ivangar/Ivan/event-manager-edit ...



ivangar committed 15 days ago

Added Logout function ...



ivangar committed 15 days ago

Commits on Nov 17, 2019

Merge pull request #8 from ivangar/Ivan/event-manager-edit ...



ivangar committed 16 days ago

Changes to event manager and event details ...



ivangar committed 16 days ago

renamed login.php to index.php



r-saba committed 16 days ago

changed login.php to index.php



r-saba committed 16 days ago

Merge branch 'master' of github.com:ivangar/COMP_353 into ragith/user... [...](#)



r-saba committed 2 days ago

add post files



r-saba committed 2 days ago

Merge pull request [#20](#) from ivangar/lvan/event-participants [...](#)



ivangar committed 2 days ago

Event participants functionality [...](#)



ivangar committed 2 days ago

Merge pull request [#19](#) from ivangar/nico/emails [...](#)



NicoChampagne committed 2 days ago

added emails table to import and fixed table to 3NF



NicoChampagne committed 2 days ago

Merge pull request [#18](#) from ivangar/nico/emails [...](#)



NicoChampagne committed 2 days ago

Working inbox and send emails



NicoChampagne committed 2 days ago

Merge pull request [#27](#) from ivangar/lvan/event-participants ...

 ivangar committed 2 days ago

Fixed importing users

 ivangar committed 2 days ago

fixed untracked files

 r-saba committed 2 days ago

Merge pull request [#26](#) from ivangar/ragith/posts-users-2 ...

 r-saba committed 2 days ago

add connection.php

 r-saba committed 2 days ago

Merge branch 'ragith/user-post' of github.com:ivangar/COMP_353 into r... ...

 r-saba committed 2 days ago

Merge pull request [#25](#) from ivangar/siamak/instant_messaging ...

 SiamakSamie committed 2 days ago

resolve merge conflict

 r-saba committed 2 days ago

working with tables

 SiamakSamie committed 2 days ago

Database

For early testing we hosted the contents of our application on our local machines using XAMP, shortly after used an AWS server to have a shared database between us all. As of December 3rd we now have our application sitting on the AITS server and use the database provided to us from AITS to use in our application.

Access information for this database is :

Host: grc353.encs.concordia.ca

User: 353_2

Pass: kMT6E6

Server Version: 8.0.18

PHP Version 7.2.11

Script to create and populate DB

```
-- phpMyAdmin SQL Dump
-- version 4.9.0.1
-- https://www.phpmyadmin.net/
--
-- Host: grc353.encs.concordia.ca
-- Generation Time: Dec 10, 2019 at 03:20 PM
-- Server version: 8.0.18
-- PHP Version: 7.2.11

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `grc353_2`
--

--
-- -----
--
-- Table structure for table `comments`
--

CREATE TABLE `comments` (
  `comment_id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `comment` varchar(255) NOT NULL,
  `user_id` int(11) NOT NULL,
  `comment_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
--
-- Dumping data for table `comments`
--

INSERT INTO `comments` (`comment_id`, `post_id`, `comment`, `user_id`) VALUES
(1, 2, 'err', 1751053),
(2, 3, 'this wallet is too expensive ', 1751053),
(3, 3, 'real talk tho', 1751053),
(4, 7, 'oops', 1751053),
(5, 8, 'ytey', 1751053),
(6, 3, 'another coment\r\n', 1751053),
(7, 15, '22', 1751053),
(8, 15, 'comment randojm', 781264),
(9, 17, 'asdasfdsfedd', 781264),
(10, 18, 'comment', 1751053),
(11, 19, 'Its so pretty!', 1751053);

-----

--
-- Table structure for table `emails`
--

CREATE TABLE `emails` (
  `email_id` int(11) NOT NULL,
  `receiver_id` int(11) NOT NULL,
  `sender_email` varchar(120) NOT NULL,
  `title` tinytext NOT NULL,
  `body` text NOT NULL,
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `emails`
--

INSERT INTO `emails` (`email_id`, `receiver_id`, `sender_email`, `title`, `body`) VALUES
(1, 781264, 'bemail@ca.ca', 'Title', 'Body of the email'),
(2, 781264, 'bemail@ca.ca', 'A beautiful day', 'The biggest email in the db by a long shot
hahahahahha, long email sir'),
(3, 781264, 'bemail@ca.ca', 'Major Tom', 'Please dont crash'),
(4, 1043082, 'aemail@ca.ca', 'Hello', 'Can we go to the zoo?'),
(5, 1043082, 'aemail@ca.ca', 'Pew', 'Pew Pew you dead'),
```


(6, 1043082, 'aemail@ca.ca', 'You', 'Iz a mad lad'),
(7, 1751053, 'sandra@email.com', 'hello', 'its me'),
(8, 1751053, 'sandra@email.com', 'test', 'gszgag'),
(9, 1751053, 'sandra@email.com', '\fsfdh\',"'\sffhhh\""),
(10, 1751053, 'sandra@email.com', '\sgsag\',"'\sfdhshsh\""),
(11, 1751053, 'sandra@email.com', '\sag\',"'\sgf\""),
(12, 1751053, 'sandra@email.com', '\title\',"'\body\""),
(13, 1751053, 'sandra@email.com', 'title', 'body'),
(14, 1751053, 'sandra@email.com', 'TEST BOY', 'TESTS'),
(15, 1751053, 'sandra@email.com', 'TEST BOY', 'TESTS'),
(16, 1751053, 'sandra@email.com', 'title', 'body'),
(17, 1751053, 'sandra@email.com', 'title', 'body'),
(18, 1751053, 'sandra@email.com', 'title', 'body'),
(19, 1751053, 'sandra@email.com', 'test', 'test'),
(20, 1751053, 'sandra@email.com', 'test', 'test'),
(21, 1751053, 'sandra@email.com', 'test', 'test'),
(22, 1751053, 'sandra@email.com', 'newest', 'sandra@email.com'),
(23, 1751053, 'sandra@email.com', 'newest', 'sandra@email.com'),
(24, 1751053, 'sandra@email.com', 'texting', 'bodying'),
(25, 9981465, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981465'),
(26, 9981474, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981474'),
(27, 9981475, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981475'),
(28, 9981476, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981476'),
(29, 9981477, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981477'),
(30, 9981477, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981477'),
(31, 1751053, 'sandra@email.com', 'newest', 'sandra@email.com'),
(32, 1751053, 'sandra@email.com', 'newest', 'sandra@email.com'),
(33, 9981479, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981479'),
(34, 1751053, 'sandra@email.com', 'test', 'sandra@email.com'),
(35, 1751053, 'sandra@email.com', 'test', 'sandra@email.com'),
(36, 9981480, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981480'),
(37, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(38, 9981481, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your user id to log in is :9981481'),
(39, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),

```

(40, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(41, 9981481, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your
user id to log in is :9981481'),
(42, 9981481, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your
user id to log in is :9981481'),
(43, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(44, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(45, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(46, 1751053, 'sandra@email.com', 'sss', 'sandra@email.com'),
(47, 1751053, 'sandra@email.com', 'sdf', 'sandra@email.com'),
(48, 1751053, 'sandra@email.com', 'testgg', 'sandra@email.com'),
(49, 1751053, 'sandra@email.com', 'testgg', 'sandra@email.com'),
(50, 1751053, 'sandra@email.com', 'testgg', 'sandra@email.com'),
(51, 1751053, 'sandra@email.com', 'testgg', 'sandra@email.com'),
(52, 1751053, 'sandra@email.com', 'testgg', 'sandra@email.com'),
(53, 1751053, 'sandra@email.com', 'ttt', 'sandra@email.com'),
(54, 1751053, 'sandra@email.com', 'ttt', 'sandra@email.com'),
(55, 1751053, 'sandra@email.com', 'aaaaa', 'sandra@email.com'),
(56, 1751053, 'sandra@email.com', 'aaaaa', 'sandra@email.com'),
(57, 1751053, 'sandra@email.com', 'ads', 'sandra@email.com'),
(58, 2135714, 'sandra@email.com', 'aaaaaaaaaaaa', 'sandra@email.com'),
(59, 2135714, 'muigi.rulock@gmail.com', 'aaaa', 'muigi.rulock@gmail.com'),
(60, 2135714, 'muigi.rulock@gmail.com', 'aaaa', 'muigi.rulock@gmail.com'),
(61, 2135714, 'muigi.rulock@gmail.com', 'asasaaaa', 'muigi.rulock@gmail.com'),
(62, 2135714, 'muigi.rulock@gmail.com', 'zsdgdg', 'muigi.rulock@gmail.com'),
(63, 9981484, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your
user id to log in is :9981484'),
(64, 9981485, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your
UserId for logging in is: 9981485'),
(65, 9981486, 'sandra@email.com', 'New Registration', 'Congrats on creating an account, your
user id to log in is :9981486'),
(66, 9981486, 'g@ggg.com', 'Bye', 'g@ggg.com');

```

```

-- -----

```

```

--
-- Table structure for table `events`
--

```

```

CREATE TABLE `events` (
  `event_id` int(11) NOT NULL,
  `event_payment_id` int(11) DEFAULT NULL,
  `event_type_id` int(11) DEFAULT NULL,

```

```

`resource_id` int(11) DEFAULT NULL,
`event_manager_id` int(11) NOT NULL,
`location_id` int(11) DEFAULT NULL,
`primary_event_group_id` int(11) DEFAULT NULL,
`event_name` varchar(250) DEFAULT NULL,
`start_date` date DEFAULT NULL,
`end_date` date DEFAULT NULL,
`period` date DEFAULT NULL,
`status` tinyint(1) DEFAULT NULL,
`total_cost` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `events`
--

INSERT INTO `events` (`event_id`, `event_payment_id`, `event_type_id`, `resource_id`,
`event_manager_id`, `location_id`, `primary_event_group_id`, `event_name`, `start_date`,
`end_date`, `period`, `status`, `total_cost`) VALUES
(7, 1, 2, 4, 4480757, 1, 1, 'Weddings celebration', '2019-11-19', '2020-02-26', '2020-12-31', 1,
5000),
(8, 2, 3, 2, 4480757, 3, 6, 'Halloween Party', '2019-11-19', '2019-11-29', '2019-12-27', 1, 500),
(9, 3, 4, 2, 4480757, 4, 7, 'Bachelor party', '2020-01-07', '2020-01-11', '2022-01-11', 2, 800),
(10, NULL, 2, 3, 6797613, 5, 8, 'John's birthday', '2019-12-16', '2019-12-16', '2021-12-16', 2,
250),
(11, 5, 1, 1, 6797613, 6, 9, 'Toronto Conference', '2020-01-10', '2020-01-18', '2020-01-18', 1,
250),
(12, 6, 3, 5, 5526601, 7, 10, 'siamak event', NULL, NULL, NULL, 2, 90),
(13, 7, 4, 6, 781264, 8, 11, 'create test', NULL, NULL, NULL, 2, 123),
(14, 8, 4, 7, 1751053, 9, 5, 'proper test', '2019-11-20', '2019-05-30', '2019-12-19', 1, 4),
(35, 29, 4, 28, 1751053, 32, 37, 'new event 2', NULL, NULL, NULL, 1, 56),
(36, 30, 4, 29, 1751053, 33, 43, 'party at my ohuse', NULL, NULL, NULL, 1, NULL),
(37, 31, 4, 30, 1157581, 34, 47, 'soccer game', NULL, NULL, NULL, 2, 2),
(38, 32, 3, 31, 2135714, 35, 48, 'baseketball game', NULL, NULL, NULL, 2, 123),
(39, 33, 1, 32, 1751053, 36, 50, 'movie watching', NULL, NULL, NULL, 2, 100),
(40, 34, 3, 33, 1751053, 37, 51, 'rock concet', NULL, NULL, NULL, 2, 123),
(41, 35, 2, 34, 1751053, 38, 52, 'rock concet2', NULL, NULL, NULL, 2, 123),
(42, 36, 2, 35, 1751053, 39, 53, 'rock concet2', NULL, NULL, NULL, 2, 123),
(43, 37, 2, 36, 1751053, 40, 54, 'rock concet3', NULL, NULL, NULL, 2, 123),
(44, 38, 2, 37, 1751053, 41, 55, 'rock concet3', NULL, NULL, NULL, 2, 123),
(45, 39, 2, 38, 1751053, 42, 56, 'rock concet3', NULL, NULL, NULL, 2, 123),
(46, 40, 2, 39, 1751053, 43, 57, 'rock concet4 and ifnal', NULL, NULL, NULL, 2, 123),
(47, 41, 4, 40, 1751053, 44, 58, 'test Ivan event', NULL, NULL, NULL, 2, 34),

```

```
(48, 42, 4, 41, 1751053, 45, 59, 'test addin event manager', NULL, NULL, NULL, 2, 34),
(55, 50, 2, 43, 1043082, 53, 70, 'dfsgfdsg', NULL, NULL, NULL, 2, NULL),
(56, 51, 2, 44, 1751053, 54, 71, 'safddsafdsafsf', NULL, NULL, NULL, 1, NULL),
(57, 52, 3, 45, 1751053, 55, 72, 'create test event', '2020-01-10', '2019-12-17', '2019-12-27', 1,
34);
```

```
-- -----
```

```
--
-- Table structure for table `event_locations`
--
```

```
CREATE TABLE `event_locations` (
  `location_id` int(11) NOT NULL,
  `address` varchar(500) DEFAULT NULL,
  `phone_no` varchar(15) DEFAULT NULL,
  `email` varchar(50) DEFAULT NULL,
  `room_no` int(5) DEFAULT NULL,
  `renting_cost` int(12) DEFAULT NULL,
  `capacity` int(12) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
-- Dumping data for table `event_locations`
--
```

```
INSERT INTO `event_locations` (`location_id`, `address`, `phone_no`, `email`, `room_no`,
`renting_cost`, `capacity`) VALUES
(1, NULL, NULL, NULL, NULL, NULL, NULL),
(2, '8273 Street Green, H3R 5L3, Montreal, Quebec', '514-999-2345',
'rgc_receptions@gmail.com', 66, 150, 200),
(3, NULL, NULL, NULL, NULL, NULL, NULL),
(4, NULL, NULL, NULL, NULL, NULL, NULL),
(5, NULL, NULL, NULL, NULL, NULL, NULL),
(6, '7830 Avenue Mountain Sights, app 1', '5147071023', 'garzon861@gmail.com', 43, 89, 600),
(7, NULL, NULL, NULL, NULL, NULL, NULL),
(8, NULL, NULL, NULL, NULL, NULL, NULL),
(9, '1234 street Newman', NULL, NULL, NULL, NULL, NULL),
(10, NULL, NULL, NULL, NULL, NULL, NULL),
(11, NULL, NULL, NULL, NULL, NULL, NULL),
(12, NULL, NULL, NULL, NULL, NULL, NULL),
(13, NULL, NULL, NULL, NULL, NULL, NULL),
(14, NULL, NULL, NULL, NULL, NULL, NULL),
```

(15, NULL, NULL, NULL, NULL, NULL, NULL),
(16, NULL, NULL, NULL, NULL, NULL, NULL),
(17, NULL, NULL, NULL, NULL, NULL, NULL),
(18, NULL, NULL, NULL, NULL, NULL, NULL),
(19, NULL, NULL, NULL, NULL, NULL, NULL),
(20, NULL, NULL, NULL, NULL, NULL, NULL),
(21, NULL, NULL, NULL, NULL, NULL, NULL),
(22, NULL, NULL, NULL, NULL, NULL, NULL),
(23, NULL, NULL, NULL, NULL, NULL, NULL),
(24, NULL, NULL, NULL, NULL, NULL, NULL),
(27, NULL, NULL, NULL, NULL, NULL, NULL),
(28, NULL, NULL, NULL, NULL, NULL, NULL),
(29, NULL, NULL, NULL, NULL, NULL, NULL),
(30, NULL, NULL, NULL, NULL, NULL, NULL),
(31, NULL, NULL, NULL, NULL, NULL, NULL),
(32, NULL, NULL, NULL, NULL, NULL, NULL),
(33, NULL, NULL, NULL, NULL, NULL, NULL),
(34, NULL, NULL, NULL, NULL, NULL, NULL),
(35, NULL, NULL, NULL, NULL, NULL, NULL),
(36, NULL, NULL, NULL, NULL, NULL, NULL),
(37, NULL, NULL, NULL, NULL, NULL, NULL),
(38, NULL, NULL, NULL, NULL, NULL, NULL),
(39, NULL, NULL, NULL, NULL, NULL, NULL),
(40, NULL, NULL, NULL, NULL, NULL, NULL),
(41, NULL, NULL, NULL, NULL, NULL, NULL),
(42, NULL, NULL, NULL, NULL, NULL, NULL),
(43, NULL, NULL, NULL, NULL, NULL, NULL),
(44, NULL, NULL, NULL, NULL, NULL, NULL),
(45, NULL, NULL, NULL, NULL, NULL, NULL),
(46, NULL, NULL, NULL, NULL, NULL, NULL),
(47, NULL, NULL, NULL, NULL, NULL, NULL),
(48, NULL, NULL, NULL, NULL, NULL, NULL),
(49, NULL, NULL, NULL, NULL, NULL, NULL),
(50, NULL, NULL, NULL, NULL, NULL, NULL),
(51, NULL, NULL, NULL, NULL, NULL, NULL),
(52, NULL, NULL, NULL, NULL, NULL, NULL),
(53, NULL, NULL, NULL, NULL, NULL, NULL),
(54, NULL, NULL, NULL, NULL, NULL, NULL),
(55, '234324', NULL, NULL, NULL, NULL, NULL);

-- -----

--

-- Table structure for table `event_participants`

--

```
CREATE TABLE `event_participants` (  
  `event_id` int(11) NOT NULL,  
  `user_id` int(11) NOT NULL,  
  `participant_status_id` int(11) NOT NULL DEFAULT '1'  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

--

-- Table structure for table `event_payment`

--

```
CREATE TABLE `event_payment` (  
  `event_payment_id` int(11) NOT NULL,  
  `account_holder` varchar(250) DEFAULT NULL,  
  `address` varchar(500) DEFAULT NULL,  
  `phone_no` varchar(15) DEFAULT NULL,  
  `bank_name` varchar(100) DEFAULT NULL,  
  `account_number` varchar(250) DEFAULT NULL,  
  `payment_method` varchar(50) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

--

-- Dumping data for table `event_payment`

--

```
INSERT INTO `event_payment` (`event_payment_id`, `account_holder`, `address`, `phone_no`,  
`bank_name`, `account_number`, `payment_method`) VALUES  
(1, NULL, NULL, NULL, NULL, NULL, NULL),  
(2, NULL, NULL, NULL, NULL, NULL, NULL),  
(3, 'Jesse Deschamps', '1234 rue Saint Antoine, H2J 3K4, Montreal', '514-234-1234', 'TD',  
'122-9274-1234', 'Visa'),  
(5, 'Ivan Garzon', '7830 Avenue Mountain Sights, app 1', '15147071023', 'BMO',  
'122-1234-1234', 'Visa, Master Card, Paypal'),  
(6, NULL, NULL, NULL, NULL, NULL, NULL),  
(7, NULL, NULL, NULL, NULL, NULL, NULL),  
(8, 'guy', NULL, NULL, NULL, NULL, NULL),  
(9, NULL, NULL, NULL, NULL, NULL, NULL),  
(10, NULL, NULL, NULL, NULL, NULL, NULL),  
(11, NULL, NULL, NULL, NULL, NULL, NULL),
```

```
--  
-- Table structure for table `event_types`  
--
```

```
CREATE TABLE `event_types` (  
  `event_type_id` int(11) NOT NULL,  
  `type` varchar(250) DEFAULT NULL,  
  `recurrent` tinyint(4) DEFAULT NULL,  
  `organization` varchar(250) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `event_types`  
--
```

```
INSERT INTO `event_types` (`event_type_id`, `type`, `recurrent`, `organization`) VALUES  
(1, 'private', 0, NULL),  
(2, 'non-profit', 0, NULL),  
(3, 'private', 1, NULL),  
(4, 'non-profit', 1, NULL);
```

```
-- -----
```

```
--  
-- Table structure for table `groups`  
--
```

```
CREATE TABLE `groups` (  
  `group_id` int(11) NOT NULL,  
  `event_id` int(11) NOT NULL,  
  `group_manager_id` int(11) DEFAULT NULL,  
  `name` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci DEFAULT NULL,  
  `details` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci DEFAULT NULL,  
  `meta_data` varchar(1000) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL  
  DEFAULT "",  
  `status` int(11) NOT NULL DEFAULT '1'  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `groups`  
--
```



```

INSERT INTO `groups` (`group_id`, `event_id`, `group_manager_id`, `name`, `details`,
`meta_data`, `status`) VALUES
(1, 7, NULL, 'eventname', 'details', '', 1),
(2, 8, NULL, 'eventname22', 'details22', '', 1),
(3, 9, NULL, 'eventname33', 'details33', '', 1),
(4, 10, NULL, 'eventname44', 'details44', '', 1),
(5, 14, NULL, 'proper test', 'Main event group', '', 1),
(6, 8, NULL, 'Halloween Party', 'Main event group', '', 1),
(7, 9, NULL, 'Bachelor party', 'Main event group', '', 1),
(8, 10, NULL, 'John birthday', 'Main event group', '', 1),
(9, 11, NULL, 'Toronto Conference', 'Main event group', '', 1),
(10, 12, NULL, 'siamak event', 'Main event group', '', 1),
(11, 13, NULL, 'create test', 'Main event group', '', 1),
(37, 35, 1751053, 'change name', 'change details', '', 2),
(43, 36, NULL, 'party at my ohuse', 'Main event group', '', 1),
(46, 35, 1751053, 'party at Concordia', 'party with our friends', '', 2),
(47, 37, NULL, 'soccer game', 'Main event group', '', 1),
(48, 38, NULL, 'baseketball game', 'Main event group', '', 1),
(49, 35, 6461563, 'dsafsadf', 'asdsad', '', 1),
(50, 39, NULL, 'movie watching', 'Main event group', '', 1),
(51, 40, NULL, 'rock concet', 'Main event group', '', 1),
(52, 41, NULL, 'rock concet2', 'Main event group', '', 1),
(53, 42, NULL, 'rock concet2', 'Main event group', '', 1),
(54, 43, NULL, 'rock concet3', 'Main event group', '', 1),
(55, 44, NULL, 'rock concet3', 'Main event group', '', 1),
(56, 45, NULL, 'rock concet3', 'Main event group', '', 1),
(57, 46, NULL, 'rock concet4 and ifnal', 'Main event group', '', 1),
(58, 47, 1751053, 'test Ivan event', 'Main event group', '', 1),
(59, 48, 1751053, 'test addin event manager', 'Main event group', '', 1),
(63, 35, 781264, 'santa gift', 'shhhh', '', 1),
(70, 55, 1043082, 'dfsgfdsg', 'Main event group', '', 1),
(71, 56, 1751053, 'safddsafdsafsf', 'Main event group', '', 1),
(72, 57, 1751053, 'create test event', 'Main event group', '', 1),
(73, 7, 1751053, 'Wedding band', 'We are the wedding band', 'music', 1);

```

```

--
-- Table structure for table `group_members`
--

```

```

CREATE TABLE `group_members` (
  `group_members_id` int(11) NOT NULL,

```

```
`user_id` int(11) NOT NULL,  
`group_id` int(11) NOT NULL,  
`participant_status_id` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
--  
-- Dumping data for table `group_members`  
--
```

```
INSERT INTO `group_members` (`group_members_id`, `user_id`, `group_id`,  
`participant_status_id`) VALUES  
(1, 1751053, 1, 1),  
(3, 781264, 37, 1),  
(4, 3715673, 37, 1),  
(5, 5677623, 37, 1),  
(6, 4433784, 37, 1),  
(7, 3143297, 37, 1),  
(8, 9818575, 37, 1),  
(9, 8263266, 37, 1),  
(11, 9693449, 37, 1),  
(12, 6461563, 37, 1),  
(13, 5528650, 37, 1),  
(14, 6890285, 37, 1),  
(15, 1157581, 37, 1),  
(16, 9981456, 37, 1),  
(17, 9547285, 37, 1),  
(18, 5526601, 37, 1),  
(19, 1940295, 37, 3),  
(20, 7126196, 37, 1),  
(21, 7935081, 37, 1),  
(22, 6630784, 37, 1),  
(23, 4569161, 37, 1),  
(24, 7137011, 37, 1),  
(25, 4480757, 37, 1),  
(26, 2135714, 37, 1),  
(27, 6797613, 37, 2),  
(28, 1043082, 37, 1),  
(29, 8640039, 37, 1),  
(30, 7034113, 37, 1),  
(31, 9293949, 37, 1),  
(35, 1751053, 37, 1),  
(36, 3060862, 2, 1),  
(54, 1751053, 43, 1),
```

(56, 3060862, 46, 1),
(57, 4433784, 46, 1),
(58, 9818575, 46, 1),
(59, 6461563, 46, 1),
(60, 3060862, 49, 1),
(61, 6461563, 49, 1),
(62, 1751053, 59, 1),
(71, 1751053, 7, 2),
(72, 1751053, 2, 2),
(73, 1751053, 46, 2),
(74, 1751053, 2, 2),
(75, 1751053, 2, 2),
(76, 1751053, 2, 2),
(77, 1751053, 2, 2),
(78, 1751053, 2, 2),
(79, 1751053, 2, 2),
(80, 1751053, 2, 2),
(81, 1751053, 2, 2),
(82, 1751053, 2, 2),
(83, 781264, 63, 1),
(84, 3715673, 63, 1),
(85, 781264, 63, 1),
(86, 8634886, 11, 1),
(87, 8634886, 37, 1),
(94, 1043082, 70, 1),
(95, 1751053, 71, 1),
(96, 4433784, 43, 1),
(97, 3060862, 43, 1),
(98, 781264, 43, 1),
(99, 3715673, 43, 1),
(100, 5677623, 43, 1),
(101, 3143297, 43, 1),
(102, 9818575, 43, 1),
(103, 8263266, 43, 1),
(104, 8634886, 43, 1),
(105, 9693449, 43, 1),
(106, 6461563, 43, 1),
(107, 5528650, 43, 1),
(108, 6890285, 43, 1),
(109, 1157581, 43, 1),
(110, 9981456, 43, 1),
(111, 9547285, 43, 1),
(112, 5526601, 43, 1),

```
(113, 1940295, 43, 1),
(114, 7126196, 43, 1),
(115, 7935081, 43, 1),
(116, 6630784, 43, 1),
(117, 4569161, 43, 1),
(118, 7137011, 43, 1),
(119, 4480757, 43, 1),
(120, 2135714, 43, 1),
(121, 6797613, 43, 1),
(122, 1043082, 43, 1),
(123, 8640039, 43, 1),
(124, 7034113, 43, 1),
(125, 9293949, 43, 1),
(126, 1751053, 72, 1),
(127, 1043082, 72, 2),
(128, 1751053, 73, 1),
(129, 1751053, 73, 1);
```

```
-- -----
```

```
--
```

```
-- Table structure for table `instant_messages`
```

```
--
```

```
CREATE TABLE `instant_messages` (
  `instant_message_id` int(11) NOT NULL,
  `group_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `message` varchar(1000) NOT NULL,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
--
```

```
-- Dumping data for table `instant_messages`
```

```
--
```

```
INSERT INTO `instant_messages` (`instant_message_id`, `group_id`, `user_id`, `message`)
VALUES
(1, 1, 2, 'hallo frand!!'),
(2, 1, 1751053, 'haaalllooo baddy'),
(3, 1, 1751053, 'hardcoded test'),
(4, 1, 1751053, 'this is a not hardcoded test'),
(5, 1, 1751053, 'ok boomer'),
```

```

(6, 1, 1751053, ''),
(7, 1, 1751053, ''),
(8, 1, 1751053, 'Clear'),
(9, 1, 1751053, 'testin'),
(10, 1, 1751053, 'Sandra talks too much'),
(11, 1, 1751053, ''),
(12, 1, 1751053, 'test\n'),
(13, 1, 1751053, 'test'),
(14, 1, 1751053, 'NEAT'),
(15, 7, 1751053, 'tes'),
(16, 7, 1751053, 'testing'),
(17, 7, 1751053, 'e-yes\n'),
(18, 37, 2135714, 'hallooo\n'),
(19, 1, 1751053, 'hi\n'),
(20, 1, 1751053, 'aefhsghtrhe'),
(21, 1, 1751053, 'Hello fellow'),
(22, 1, 1751053, 'I want to dance all night long on the beach'),
(23, 73, 1751053, 'Hey guys any idea what to do for the music?\n');

```

```

-- -----

```

```

--
-- Table structure for table `participant_status`
--

```

```

CREATE TABLE `participant_status` (
  `participant_status_id` int(11) NOT NULL,
  `status` varchar(100) NOT NULL,
  `description` varchar(250) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

```

--
-- Dumping data for table `participant_status`
--

```

```

INSERT INTO `participant_status` (`participant_status_id`, `status`, `description`) VALUES
(1, 'approved', 'The participant has been approved by the admin'),
(2, 'rejected', 'The participant has been rejected by the admin'),
(3, 'pending', 'A user requested to join a group/event but has not been approved by the admin');

```

```

-- -----

```

```

--

```

```
-- Table structure for table `poll`
```

```
--
```

```
CREATE TABLE `poll` (  
  `poll_id` int(11) NOT NULL,  
  `group_id` int(11) NOT NULL,  
  `title` varchar(100) NOT NULL,  
  `end_date` datetime NOT NULL,  
  `options` varchar(1000) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `poll`
```

```
--
```

```
INSERT INTO `poll` (`poll_id`, `group_id`, `title`, `end_date`, `options`) VALUES  
(1, 73, 'Music ideas?', '2019-12-09 00:00:00', 'Mary had a little lamb;Monster Mash'),  
(2, 7, 'another poll', '2019-12-05 00:00:00', 'burger;fries;steak');
```

```
-- -----
```

```
--
```

```
-- Table structure for table `poll_results`
```

```
--
```

```
CREATE TABLE `poll_results` (  
  `poll_result_id` int(11) NOT NULL,  
  `poll_id` int(11) NOT NULL,  
  `user_id` int(11) NOT NULL,  
  `option_selected` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `poll_results`
```

```
--
```

```
INSERT INTO `poll_results` (`poll_result_id`, `poll_id`, `user_id`, `option_selected`) VALUES  
(1, 1, 9547285, 1),  
(2, 1, 6461563, 2),  
(4, 1, 6630784, 2),  
(5, 1, 9547285, 1);
```

```
-- -----
```

```

--
-- Table structure for table `posts`
--

CREATE TABLE `posts` (
  `post_id` int(11) NOT NULL,
  `group_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `post_content` varchar(255) NOT NULL,
  `post_image` varchar(255) NOT NULL,
  `upload_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `post_permission` int(11) NOT NULL COMMENT '0:view only 1: comment 2:add'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

--
-- Dumping data for table `posts`
--

INSERT INTO `posts` (`post_id`, `group_id`, `user_id`, `post_content`, `post_image`,
`post_permission`) VALUES
(1, 37, 1751053, 'ey', '', 0),
(2, 7, 1751053, 'er', '', 1),
(3, 7, 1751053, '', '89icyG3ZeiAv.jpg', 1),
(4, 7, 1751053, 'this is a belt', '596mxu3wThHe.jpg', 2),
(5, 37, 1751053, '', '27Y5tnpeWm9i.jpg', 1),
(6, 37, 1751053, '', '68djYvWZnyiz.mp4', 1),
(7, 7, 1751053, 'hello', '', 0),
(8, 46, 1751053, 'ytsrdhh', '405g1ikV8zNA.jpg', 0),
(9, 7, 1751053, 'post', '', 0),
(10, 46, 1751053, 'video cat', '25XNxnFUXK6b.mp4', 0),
(11, 1, 1751053, 'cats', '68twljBL2czK.mp4', 0),
(12, 7, 1751053, '', '97fhz8d9Wy4q.mp4', 0),
(13, 37, 3060862, 'hey everybody', '', 0),
(14, 43, 1751053, 'test', '', 0),
(15, 43, 1751053, '123131', '', 1),
(16, 43, 781264, 'hello woelrd', '', 0),
(17, 43, 781264, 'this is a tree', '51Nn2tzmtNWY.jpg', 1),
(18, 73, 1751053, 'hey geys', '', 1),
(19, 73, 1751053, 'Look at my new trombone', '84tmzrKiLCkR.jpg', 1);

```

```

--
-- Table structure for table `resources`
--

CREATE TABLE `resources` (
  `resource_id` int(11) NOT NULL,
  `system_charge_rate_id` int(11) NOT NULL DEFAULT '1',
  `flat_fee` int(12) DEFAULT NULL,
  `type` varchar(100) DEFAULT NULL,
  `unit` varchar(50) DEFAULT NULL,
  `resource_name` varchar(250) DEFAULT NULL,
  `extra_fee` int(12) DEFAULT NULL,
  `discount` int(12) DEFAULT NULL,
  `payment_receiver` varchar(100) CHARACTER SET latin1 COLLATE latin1_swedish_ci
  DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `resources`
--

INSERT INTO `resources` (`resource_id`, `system_charge_rate_id`, `flat_fee`, `type`, `unit`,
`resource_name`, `extra_fee`, `discount`, `payment_receiver`) VALUES
(1, 1, 140, 'Bandwidth', 'Kbps/s', 'Event bandwidth', 50, 30, ''),
(2, 1, 80, 'Bandwidth', 'kbps/s', 'Halloween bandwidth', 50, 25, ''),
(3, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(4, 1, 80, 'Bandwidth', 'app 1', 'Halloween bandwidth', 33, 9, 'muigi.rulock@gmail.com'),
(5, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(6, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(7, 1, 234, NULL, NULL, NULL, NULL, NULL, ''),
(8, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(9, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(10, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(11, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(12, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(13, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(14, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(15, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(16, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(17, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(18, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(19, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),
(20, 1, NULL, NULL, NULL, NULL, NULL, NULL, ''),

```



```
(21, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(22, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(23, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(24, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(25, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(26, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(27, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(28, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(29, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(30, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(31, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(32, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(33, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(34, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(35, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(36, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(37, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(38, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(39, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(40, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(41, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(42, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, "),
(43, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL),
(44, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL),
(45, 1, 34, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

—

— —

```
CREATE TABLE `roles` (  
  `roleId` int(11) NOT NULL,  
  `name` text NOT NULL,  
  `description` text NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- Dumping data for table `roles`
```

```
(1, 'admin', 'admin account'),  
(2, 'Controller', 'Admin + can set price of system'),  
(3, 'god', 'all access in the system can create admins');
```

```
-- -----
```

```
--  
-- Table structure for table `system_charge_rate`  
--
```

```
CREATE TABLE `system_charge_rate` (  
  `system_charge_rate_id` int(11) NOT NULL,  
  `charge_rate` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `system_charge_rate`  
--
```

```
INSERT INTO `system_charge_rate` (`system_charge_rate_id`, `charge_rate`) VALUES  
(1, 34);
```

```
-- -----
```

```
--  
-- Table structure for table `users`  
--
```

```
CREATE TABLE `users` (  
  `user_id` int(11) NOT NULL,  
  `first_name` varchar(250) NOT NULL,  
  `middle_name` varchar(250) DEFAULT NULL,  
  `last_name` varchar(250) NOT NULL,  
  `user_pwd` varchar(1000) NOT NULL,  
  `address` varchar(500) DEFAULT NULL,  
  `date_of_birth` date NOT NULL,  
  `email` varchar(120) NOT NULL,  
  `organization` varchar(250) DEFAULT NULL,  
  `meta_data` varchar(1000) DEFAULT ""  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `users`
```

--

```
INSERT INTO `users` (`user_id`, `first_name`, `middle_name`, `last_name`, `user_pwd`,
`address`, `date_of_birth`, `email`, `organization`, `meta_data`) VALUES
(781264, 'Kalpdrum', NULL, 'Passi',
'$2y$10$vPrz6huWkTewkBd72qObve0aV2NZoQvpRqi8j0rsHFPATwgZX.Zn2', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(1043082, 'Hannes', NULL, 'Voigt',
'$2y$10$xDrnkEuKIBWns/OIF.4zFOzvG9mFHRDrIAwTM7GdkVvwEj3rWQp7e', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(1157581, 'Wookey', NULL, 'Lee',
'$2y$10$f5mRnQSEGjTOjqayw8daw.H3Oy5VOyFYDK7O0zY7eBjLaAajU7JwO', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(1751053, 'Sandra', NULL, 'Deamo',
'$2y$10$zmaHslvJqvUv25OIEF25iOKhkLc1NmLfQXcNyUF6lhywGO0gfzpXC', '123 street
street', '0000-00-00', 'sandra@email.com', ' org ', 'admin;health nut'),
(1940295, 'Normandsss', 'asdsad', 'Seguinsss',
'$2y$10$3WeoNLLtOe7lshSzEEb1ceFNFU7XT.oYHfvgck8.tW9JTl1TiUaUy', '23345 new
address', '2019-12-10', 'email@mail.com', ' org ', NULL),
(2135714, 'Di', NULL, 'Wang',
'$2y$10$j4QfK0GvwdUI1aMJtA.Ezuc1rKD3wBzrrx/7DnbL4PepY5gRnMZBi', '123 street street',
'0000-00-00', 'muigi.rulock@gmail.com', ' org ', NULL),
(3060862, 'Theo', NULL, 'Harder',
'$2y$10$9164hWi4IAEJDfWwCMxXi.hPOgdwFcJTT6bsJfW4ijQn/BSgo2cVu', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(3143297, 'Dominique', NULL, 'Laurent',
'$2y$10$y5HdM1CEqb144BVoBJp/..ONoauFBbcDu4neJws7bNda6SeYrbBe', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(3715673, 'Shri', 'Kant', 'Ojha',
'$2y$10$YHZMotzdDw2ScjsJ4K2jR.QbphdhVxetDdveuftxJRTmgCFGLh4Ye', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(4433784, 'Ozgur', NULL, 'Ulusoy',
'$2y$10$Btb.gK/Oq033qh4XDThyF.OoVxAV1pE4W4ZtduApssqGUxuXB5NGW', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(4480757, 'Carlos', 'Jose', 'Lucena',
'$2y$10$IQA/.VhLX7bXE8SlwaLVle.GT4CC5o3JzWW3yG6BJtYD0C1AufF8S', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(4569161, 'Leong', NULL, 'Lee',
'$2y$10$yljLTicPLTcC1U4BFbNy.OMw.NqbQfshFTM6HTmTbfuCitiz5csPeW', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(5526601, 'Sanja', NULL, 'Candrlic',
'$2y$10$76UglwrZAbuDDUDn/.lu1.6GkemUqpFSxOcKfWMLqHHNz2Awo6Rna', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
```

(5528650, 'John', NULL, 'Plaice',
'\$2y\$10\$bQA1r1nYs..CORdq0Zol8O6D88pHdvJRWhJhfKoksuKKQd0fNuYjC', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(5677623, 'Raisow', 'K.', 'Agrawal',
'\$2y\$10\$G9sx4Zn0l3eCv.PpVhtab.QGKx5VxLd4oRgCNYJICQ9hqFWRD0c4G', '1234 street
Newman', '1997-07-18', 'raisow@gmail.com', NULL, NULL),
(6461563, 'Cagatay', NULL, 'Catal',
'\$2y\$10\$mNddUeC0h15vXiekCbXGfe4EEhKsmALeNG3YBhm.5NZcUVnluy53G', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(6630784, 'Jennifer', 'L', 'Leopold',
'\$2y\$10\$pUbKtXAemRs8CS9TnpzJu8oCRceGWSsrq7z4PoCVF2uZ9OwS6mZq', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(6797613, 'Michael', 'middle', 'Jordan',
'\$2y\$10\$Cdse.CWnQPFwSjehHxgRTe5RevB2kld55DyaZesUm/XjZWYabAVC2', '2345 rue
berri', '2019-12-11', 'email@mail.com', 'org Inc', NULL),
(6890285, 'Jeevaratnam', NULL, 'Kolins',
'\$2y\$10\$sku72/AYPwiDvL9hLZW4/U.fJvU3c1MmTqKdLrRPJbYvoVjQfyAXcO', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(7034113, 'Kyoko', NULL, 'Nagano',
'\$2y\$10\$M7.4lvXtafOVDEepg0z6LeI/Qk/h3aWp9ER2.SJ450pl3C1FI4LvO', '123 street street',
'0000-00-00', 'email', ' ', NULL),
(7126196, 'Udai', NULL, 'Shanker',
'\$2y\$10\$7PVsAEvPXL3Q3wPzO2T6weJBFSPcCCUpY1HSNX8qMGOEu1w1AMZlu', '123
street street', '0000-00-00', 'email', ' org ', NULL),
(7137011, 'Olivier', NULL, 'Savarybelanger',
'\$2y\$10\$wwL/RbIJCb0saEjBm5zf4uO9w08dxCBBWi5nU.Sx0ETCbqW2OG.6e', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(7935081, 'Guenter', NULL, 'Hackl',
'\$2y\$10\$q02MEhp7SA0lfJ5lwMPZheE91rsbqXYpP1LP/nQyaCbTWBXLBmMTu', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(8263266, 'Rainer', NULL, 'Unland',
'\$2y\$10\$frER/.Qyp0l73mslrLld5.iCrA3Ofg/9ctu7/k7SxEIsBCoRfyChe', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(8634886, 'Alfredo', NULL, 'Cuzzocre',
'\$2y\$10\$SoGs.g4iMz5gPy6u1SvD3C.AeCW3zYD1OkoKBeC6xPOT9OQX9I3zy', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(8640039, 'Wolfgang', NULL, 'Lehner',
'\$2y\$10\$7SO.yBq0eDI7yoXP53kQf.zPTggDPpgqQVNJLQ00W.NWQV.4JrcEy', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(9293949, 'Garzon', NULL, 'Ivan',
'\$2y\$10\$jMec/ljyt9j0aQDI.UFZbebaTX6D.5DbF5JfP7p/BMaCpzzjvRwwK', 'random street',
'1989-05-10', 'ivang@mail.com', ' org ', NULL),

```
(9547285, 'Alen', NULL, 'Jakupovic',
'$2y$10$zZqz4QU5p4E5tDb7wdQEQ.z6XK0cqdzEPjs8xljkWWeXS1ltHtvEO', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(9693449, 'Christine', NULL, 'Collet',
'$2y$10$6MRDNri4mWu8zw6EaWAOx.wy7vW2FeOaPUj.7ke1PMpC7Luaz6Lfm', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(9818575, 'Ratvinder', 'S', 'Grewal',
'$2y$10$YCsWCM.fzo2Ot6sJYU.dN.mjHuzmDiZpggBNYq1nw3.kqxaHH8992', '123 street
street', '0000-00-00', 'email', ' org ', NULL),
(9981456, 'Roger', 'Castillo', 'Espinola',
'$2y$10$OjcfF4vldz2naoOTMIHcWe30qblpL.3R4Wd/YjNEE9won0VurxstS', '123 street street',
'0000-00-00', 'email', ' org ', NULL),
(9981468, 'Wookey', NULL, 'Lee',
'$2y$10$f5mRnQSEGjTOjqayw8daw.H3Oy5VOyFYDK7O0zY7eBjLaAajU7JwO', '123 street
street', '0000-00-00', 'email', ' org ', ''),
(9981484, 'Nico', '', 'Nico',
'$2y$10$bskIOiAhNPICIWzDquj7U.m11LLZB4Kkcq9v9DJM.P0..C2J1h39S', 'maple',
'1999-11-22', 'g@gg.com', 'cdk', ''),
(9981486, 'Nico', '', 'Nico',
'$2y$10$ItTeaFRCa8FPstTJ5oap8.IM6sR86yGYV8ycZy7SUGFRodf9bE8/y', 'fds', '1999-11-11',
'g@ggg.com', 'fds', '');
```

```
-- -----
```

```
--
-- Table structure for table `user_roles`
--
```

```
CREATE TABLE `user_roles` (
  `user_role_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `role_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
-- Dumping data for table `user_roles`
--
```

```
INSERT INTO `user_roles` (`user_role_id`, `user_id`, `role_id`) VALUES
(6, 1751053, 3),
(7, 2135714, 3),
(8, 781264, 2);
```

```

--
-- Indexes for dumped tables
--

--
-- Indexes for table `comments`
--
ALTER TABLE `comments`
  ADD PRIMARY KEY (`comment_id`),
  ADD KEY `post_id` (`post_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `emails`
--
ALTER TABLE `emails`
  ADD PRIMARY KEY (`email_id`),
  ADD KEY `receiver_id` (`receiver_id`);

--
-- Indexes for table `events`
--
ALTER TABLE `events`
  ADD PRIMARY KEY (`event_id`),
  ADD KEY `fk_event_payment` (`event_payment_id`),
  ADD KEY `fk_event_resource` (`resource_id`),
  ADD KEY `fk_event_location` (`location_id`),
  ADD KEY `fk_event_manager` (`event_manager_id`),
  ADD KEY `fk_event_type` (`event_type_id`),
  ADD KEY `fk_primary_group_id` (`primary_event_group_id`);

--
-- Indexes for table `event_locations`
--
ALTER TABLE `event_locations`
  ADD PRIMARY KEY (`location_id`);

--
-- Indexes for table `event_participants`
--
ALTER TABLE `event_participants`
  ADD PRIMARY KEY (`event_id`,`user_id`),
  ADD KEY `fk_user_id` (`user_id`),

```

```

ADD KEY `fk_event_id` (`event_id`),
ADD KEY `fk_participant_status` (`participant_status_id`);

--
-- Indexes for table `event_payment`
--
ALTER TABLE `event_payment`
  ADD PRIMARY KEY (`event_payment_id`);

--
-- Indexes for table `event_types`
--
ALTER TABLE `event_types`
  ADD PRIMARY KEY (`event_type_id`);

--
-- Indexes for table `groups`
--
ALTER TABLE `groups`
  ADD PRIMARY KEY (`group_id`),
  ADD KEY `fk_group_to_event_id` (`event_id`);

--
-- Indexes for table `group_members`
--
ALTER TABLE `group_members`
  ADD PRIMARY KEY (`group_members_id`),
  ADD KEY `fk_group_member_to_user` (`user_id`),
  ADD KEY `fk_group_member_to_participant_status` (`participant_status_id`),
  ADD KEY `fk_group_member_to_group` (`group_id`);

--
-- Indexes for table `instant_messages`
--
ALTER TABLE `instant_messages`
  ADD PRIMARY KEY (`instant_message_id`);

--
-- Indexes for table `participant_status`
--
ALTER TABLE `participant_status`
  ADD PRIMARY KEY (`participant_status_id`);

```

```

--
-- Indexes for table `poll`
--
ALTER TABLE `poll`
  ADD PRIMARY KEY (`poll_id`),
  ADD KEY `fk_poll_to_group` (`group_id`);

--
-- Indexes for table `poll_results`
--
ALTER TABLE `poll_results`
  ADD PRIMARY KEY (`poll_result_id`),
  ADD KEY `fk_poll_result_to_user` (`user_id`),
  ADD KEY `fk_poll_results_to_poll` (`poll_id`);

--
-- Indexes for table `posts`
--
ALTER TABLE `posts`
  ADD PRIMARY KEY (`post_id`),
  ADD KEY `group_id` (`group_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `resources`
--
ALTER TABLE `resources`
  ADD PRIMARY KEY (`resource_id`);

--
-- Indexes for table `roles`
--
ALTER TABLE `roles`
  ADD PRIMARY KEY (`roleid`),
  ADD UNIQUE KEY `roleid` (`roleid`);

--
-- Indexes for table `system_charge_rate`
--
ALTER TABLE `system_charge_rate`
  ADD PRIMARY KEY (`system_charge_rate_id`);

--

```



```

-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`user_id`);

--
-- Indexes for table `user_roles`
--
ALTER TABLE `user_roles`
  ADD PRIMARY KEY (`user_role_id`),
  ADD KEY `fk_user_roles_to_users` (`user_id`),
  ADD KEY `fk_user_roles_to_roles` (`role_id`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `comments`
--
ALTER TABLE `comments`
  MODIFY `comment_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=12;

--
-- AUTO_INCREMENT for table `emails`
--
ALTER TABLE `emails`
  MODIFY `email_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=67;

--
-- AUTO_INCREMENT for table `events`
--
ALTER TABLE `events`
  MODIFY `event_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=58;

--
-- AUTO_INCREMENT for table `event_locations`
--
ALTER TABLE `event_locations`
  MODIFY `location_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=56;

--
-- AUTO_INCREMENT for table `event_payment`

```

```

--
ALTER TABLE `event_payment`
  MODIFY `event_payment_id` int(11) NOT NULL AUTO_INCREMENT,
  AUTO_INCREMENT=53;

--
-- AUTO_INCREMENT for table `event_types`
--
ALTER TABLE `event_types`
  MODIFY `event_type_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;

--
-- AUTO_INCREMENT for table `groups`
--
ALTER TABLE `groups`
  MODIFY `group_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=74;

--
-- AUTO_INCREMENT for table `group_members`
--
ALTER TABLE `group_members`
  MODIFY `group_members_id` int(11) NOT NULL AUTO_INCREMENT,
  AUTO_INCREMENT=130;

--
-- AUTO_INCREMENT for table `instant_messages`
--
ALTER TABLE `instant_messages`
  MODIFY `instant_message_id` int(11) NOT NULL AUTO_INCREMENT,
  AUTO_INCREMENT=24;

--
-- AUTO_INCREMENT for table `poll`
--
ALTER TABLE `poll`
  MODIFY `poll_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;

--
-- AUTO_INCREMENT for table `poll_results`
--
ALTER TABLE `poll_results`
  MODIFY `poll_result_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=16;

```

```

--
-- AUTO_INCREMENT for table `posts`
--
ALTER TABLE `posts`
  MODIFY `post_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=20;

--
-- AUTO_INCREMENT for table `resources`
--
ALTER TABLE `resources`
  MODIFY `resource_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=46;

--
-- AUTO_INCREMENT for table `system_charge_rate`
--
ALTER TABLE `system_charge_rate`
  MODIFY `system_charge_rate_id` int(11) NOT NULL AUTO_INCREMENT,
  AUTO_INCREMENT=9;

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `user_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9981487;

--
-- AUTO_INCREMENT for table `user_roles`
--
ALTER TABLE `user_roles`
  MODIFY `user_role_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `comments`
--
ALTER TABLE `comments`
  ADD CONSTRAINT `comments_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts`
  (`post_id`),
  ADD CONSTRAINT `comments_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users`
  (`user_id`),

```

```

    ADD CONSTRAINT `comments_ibfk_3` FOREIGN KEY (`post_id`) REFERENCES `posts`
(`post_id`),
    ADD CONSTRAINT `comments_ibfk_4` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`);

--
-- Constraints for table `emails`
--
ALTER TABLE `emails`
    ADD CONSTRAINT `emails_ibfk_1` FOREIGN KEY (`receiver_id`) REFERENCES `users`
(`user_id`);

--
-- Constraints for table `events`
--
ALTER TABLE `events`
    ADD CONSTRAINT `fk_event_location` FOREIGN KEY (`location_id`) REFERENCES
`event_locations` (`location_id`) ON DELETE SET NULL ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_event_manager` FOREIGN KEY (`event_manager_id`)
REFERENCES `users` (`user_id`) ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_event_payment` FOREIGN KEY (`event_payment_id`) REFERENCES
`event_payment` (`event_payment_id`) ON DELETE SET NULL ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_event_resource` FOREIGN KEY (`resource_id`) REFERENCES
`resources` (`resource_id`) ON DELETE SET NULL ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_event_type` FOREIGN KEY (`event_type_id`) REFERENCES
`event_types` (`event_type_id`) ON DELETE SET NULL ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_primary_group_id` FOREIGN KEY (`primary_event_group_id`)
REFERENCES `groups` (`group_id`) ON DELETE SET NULL ON UPDATE CASCADE;

--
-- Constraints for table `event_participants`
--
ALTER TABLE `event_participants`
    ADD CONSTRAINT `fk_event_id` FOREIGN KEY (`event_id`) REFERENCES `events`
(`event_id`) ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_participant_status` FOREIGN KEY (`participant_status_id`)
REFERENCES `participant_status` (`participant_status_id`) ON UPDATE CASCADE,
    ADD CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
ON UPDATE CASCADE;

--
-- Constraints for table `groups`
--

```

```

ALTER TABLE `groups`
  ADD CONSTRAINT `fk_group_to_event_id` FOREIGN KEY (`event_id`) REFERENCES
`events` (`event_id`);

--
-- Constraints for table `group_members`
--
ALTER TABLE `group_members`
  ADD CONSTRAINT `fk_group_member_to_group` FOREIGN KEY (`group_id`)
REFERENCES `groups` (`group_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `fk_group_member_to_participant_status` FOREIGN KEY
(`participant_status_id`) REFERENCES `participant_status` (`participant_status_id`) ON
DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `fk_group_member_to_user` FOREIGN KEY (`user_id`) REFERENCES
`users` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `poll`
--
ALTER TABLE `poll`
  ADD CONSTRAINT `fk_poll_to_group` FOREIGN KEY (`group_id`) REFERENCES `groups`
(`group_id`);

--
-- Constraints for table `poll_results`
--
ALTER TABLE `poll_results`
  ADD CONSTRAINT `fk_poll_result_to_user` FOREIGN KEY (`user_id`) REFERENCES
`users` (`user_id`),
  ADD CONSTRAINT `fk_poll_results_to_poll` FOREIGN KEY (`poll_id`) REFERENCES `poll`
(`poll_id`);

--
-- Constraints for table `posts`
--
ALTER TABLE `posts`
  ADD CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`group_id`) REFERENCES `groups`
(`group_id`),
  ADD CONSTRAINT `posts_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`);

--
-- Constraints for table `user_roles`

```

```
--  
ALTER TABLE `user_roles`  
  ADD CONSTRAINT `fk_user_roles_to_roles` FOREIGN KEY (`role_id`) REFERENCES `roles`  
  (`roleId`) ON DELETE CASCADE ON UPDATE CASCADE,  
  ADD CONSTRAINT `fk_user_roles_to_users` FOREIGN KEY (`user_id`) REFERENCES  
  `users` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE;  
COMMIT;  
  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

References

Pdf documentation for db project 2 [db19s-P2.pdf](#) From
https://confsys.encs.concordia.ca/CrsMgr/crs_student/course_student_default.php?

Comp353 powerpoint slides From
https://confsys.encs.concordia.ca/CrsMgr/crs_student/course_student_default.php?

