DBMS Assignment 4
Topic: College Festival Management Web Application

Group name: DeBurgers

March 2024

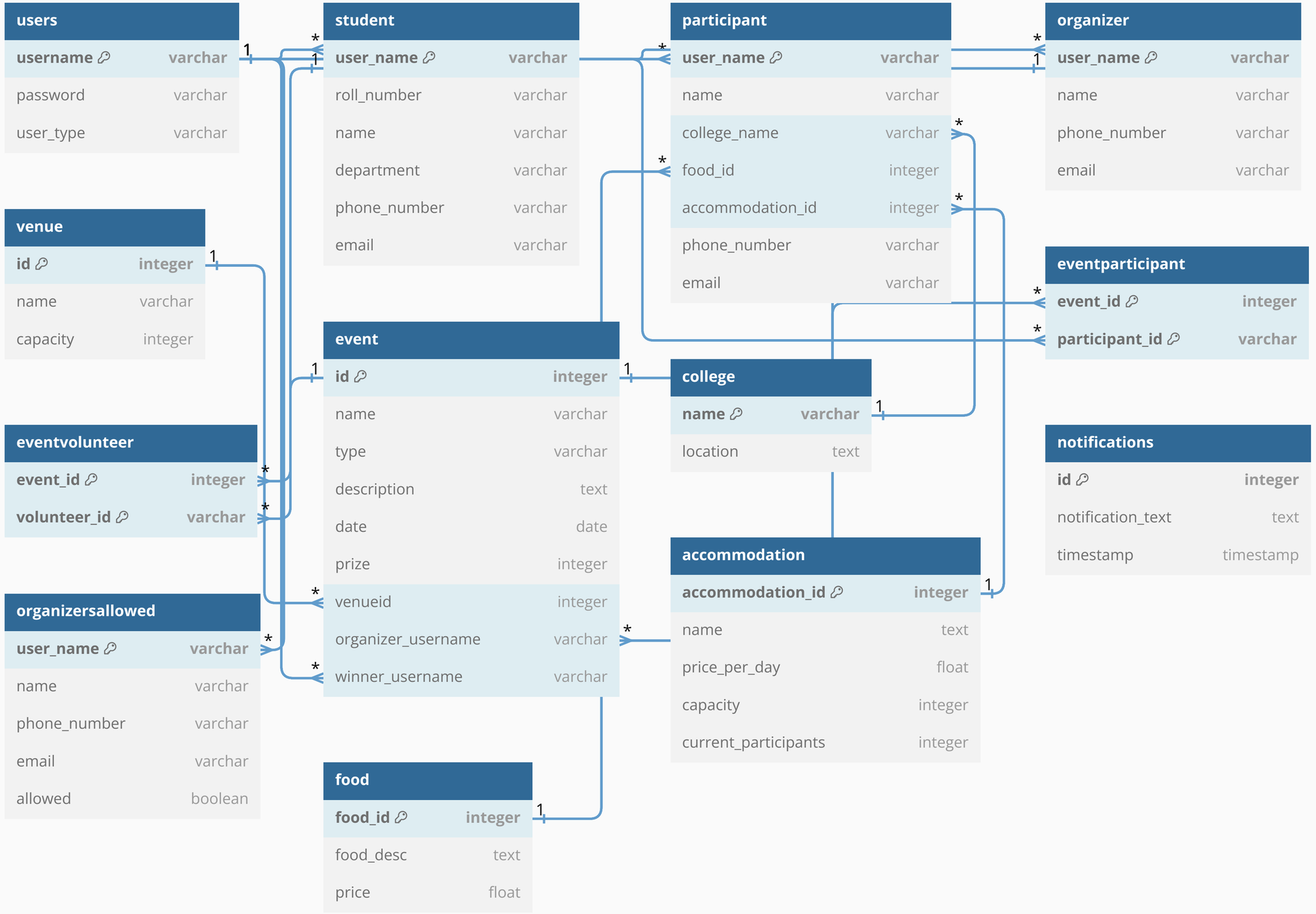
1. Introduction

The College Festival Management Web Application is designed to streamline the organization and management of a university cultural festival. In response to the growing complexity of event coordination and participant engagement, a web-based system has been developed to facilitate seamless interactions among external participants, students, volunteers, organizers, and database administrators. The application utilizes the Flask framework and PostgreSQL along with HTML and CSS, providing a comprehensive solution to meet the diverse needs of the festival.

The primary goal of this web application is to enhance user experience and ensure efficient handling of festival-related activities. The intended users include external participant registrants, students, volunteers, event organizers, and database administrators. Each user group has distinct roles and functionalities tailored to their specific requirements within the festival management ecosystem.

Functional Requirements

1. **External Participants:**
    - Create an account to access festival information.
    - Browse and search for events and schedules.
    - Register for specific events seamlessly.
    - Receive real-time updates on event winners.
    - Access logistics information, including accommodation and food options.
2. **Students:**
    - Browse and search event schedules.
    - Register for desired events.
    - Register as Volunteers.
    - Receive real time updates on event winners.
3. **Organizers:**
    - Create an account for detailed event management.
    - Browse comprehensive event details.
    - View and search volunteer and logistics information.
    - Can create events.
4. **Database Administrators:**
    - Delete existing user accounts.
    - Can allow a new Organizer to organize an event.
    - Can view details of all the users and events.



2. Schema

a. Users Schema:

- **Use:** The Users table is designed to store information about different types of users in the system. It includes external participants, students, organizers, and database administrators.
- **Primary Key:** `username` is the primary key, ensuring unique identification for each user.
- **Functions:** The `set_password` and `check_password` methods provide secure password handling using bcrypt password hashing.

b. Student Schema:

- **Use:** The Student table represents information specific to student users, such as roll number, name, department, phone number, and email.
- **Primary Key:** `user_name` serves as the primary key, linking each student to a unique user in the Users table.
- **Foreign Key:** `user_name` references the Users table, establishing a relationship between student-specific details and the general user information.

c. Participant Schema:

- **Use:** The Participant table stores details about external participants, including their name, college name, contact information, and choices for food and accommodation.
- **Primary Key:** `user_name` is the primary key, linking each participant to a unique user in the Users table.
- **Foreign Keys:** `college_name`, `user_name`, `food_id`, and `accommodation_id` establish relationships with other tables for additional information.

d. Organizer Schema:

- **Use:** The Organizer table contains information about event organizers, including their name, phone number, and email.
- **Primary Key:** `user_name` serves as the primary key, ensuring a unique identifier for each organizer.
- **Foreign Key:** `user_name` references the Users table, establishing a relationship between organizer-specific details and general user information.

e. Venue Schema:

- **Use:** The Venue table represents different event locations, storing details like venue name and capacity.
- **Primary Key:** `id` is the primary key, ensuring unique identification for each venue.

f. Event Schema:

- **Use:** The Event table manages information about various festival events, including details like event name, type, description, date, prizes, and venue.
- **Primary Key:** `id` serves as the primary key, uniquely identifying each festival event.
- **Foreign Keys:** `venueid`, `organizer_username`, and `winner_username` establish relationships with other tables for venue information, organizer details, and winner information.

g. College Schema:

- **Use:** The College table contains information about different colleges, including college name and location.
- **Primary Key:** `name` is the primary key, ensuring unique identification for each college.

h. EventParticipant Schema:

- **Use:** The EventParticipant table maintains the association between events and participants, indicating which participants are registered for specific events.
- **Primary Keys:** `event_id` and `participant_id` form a composite primary key, uniquely identifying each participation record.
- **Foreign Keys:** `event_id` and `participant_id` are foreign keys referencing their respective tables.

i. EventVolunteer Schema:

- **Use:** The EventVolunteer table records the association between events and student volunteers, signifying which students are volunteering for specific events.
- **Primary Keys:** `event_id` and `volunteer_id` create a composite primary key, ensuring a unique identifier for each volunteering record.
- **Foreign Keys:** `event_id` and `volunteer_id` are foreign keys referencing their respective tables.

j. Food Schema:

- **Use:** The Food table manages details about available food options, including a unique identifier, food description, and price.
- **Primary Key:** `food_id` is the primary key, ensuring unique identification for each food item.

k. Accommodation Schema:

- **Use:** The Accommodation table stores information about available accommodation options, including a unique identifier, accommodation name, and price per day.
- **Primary Key:** `accommodation_id` is the primary key, ensuring unique identification for each accommodation option.

l. Notifications Schema:

- **Use:** The Notifications table is designed to record system notifications, including text and timestamp details.
- **Primary Key:** `id` is the primary key, ensuring a unique identifier for each notification.

m. OrganizersAllowed Schema:

- **Use:** The OrganizersAllowed table manages a list of organizers allowed to perform specific actions, including their contact details.
- **Primary Key:** `user_name` is the primary key, ensuring a unique identifier for each allowed organizer.
- **Foreign Key:** `user_name` is the foreign key that references the users table establishing a relationship between organizer- specific details and general user information

3. Triggers

a. Trigger: winner_updated_trigger

- **Functionality:**
  - **Use:** Generates notifications when the winner is updated for an event.
  - **Event:** Triggered after an update of `winner_username` on the `event` table.
- **Function:** `winner_updated_func()`
  - **Use:** Inserts a notification into the `notifications` table.
  - **Event:** Invoked by the `winner_updated_trigger` trigger.

b. Trigger: insert_into_organizer_trigger

- **Functionality:**
  - **Use:** Inserts a record into the `Organizer` table when an organizer is allowed by Admin.
  - **Event:** Triggered after an update of `allowed` on the `organizers_allowed` table.
- **Function:** `insert_into_organizer()`
  - **Use:** Adds a record to the `Organizer` table if the `allowed` column is set to `TRUE`.
  - **Event:** Invoked by the `insert_into_organizer_trigger` trigger.

c. Trigger: event_insert_notification_trigger

- **Functionality:**
  - **Use:** Generates notifications when a new event is created.
  - **Event:** Triggered after an insert into the `event` table.
- **Function:** `create_event_notification()`
  - **Use:** Inserts a notification into the `notifications` table.
  - **Event:** Invoked by the `event_insert_notification_trigger` trigger.

d. Trigger: update_current_participants_trigger

- **Functionality:**
  - **Use:** Updates the `current_participants` count in the `accommodation` table when a new participant is added.
  - **Event:** Triggered after an insert into the `participant` table.
- **Function:** `update_current_participants()`
  - **Use:** Increments the `current_participants` count if the `accommodation_id` is not `NULL`.
  - **Event:** Invoked by the `update_current_participants_trigger` trigger.

e. Trigger: decrement_current_participants_trigger

- **Functionality:**
  - **Use:** Decrements the `current_participants` count in the `accommodation` table when a participant is deleted.
  - **Event:** Triggered after a delete from the `participant` table.
- **Function:** `decrement_current_participants()`
  - **Use:** Decrements the `current_participants` count if the `accommodation_id` is not `NULL`.
  - **Event:** Invoked by the `decrement_current_participants_trigger` trigger.

Trigger Initialization on Connection

The `create_triggers_on_connect` function is executed upon establishing a database connection to ensure that triggers are properly initialized. It checks if each required function and trigger already exist in the database. If not, it proceeds to create them using the provided SQL statements.

4. Queries

Database Functions and Queries

a. User-related Operations

- `is_user_exists(username)`: Checks if a user with the given username exists.
- `get_user(username)`: Retrieves user details for a given username.
- `create_user_student(username, password, roll_no, stud_name, dept, phone, email)`: Creates a new student user.
- `create_user_participant(username, password, participant_name, college, phone, email, food_id, accommodation_id)`: Creates a new participant user.
- `create_user_organizer(username, password, organizer_name, phone, email)`: Creates a new organizer user.
- `get_organizer(username)`: Retrieves organizer details for a given username.
- `get_students()`: Retrieves the username and name of all students.
- `get_participants_for_event(event_id)`: Retrieves participants for a specific event.

b. Event-related Operations

- `get_events_not_registered(username)`: Retrieves events not registered by a student.
- `get_events_not_participated(username)`: Retrieves events not participated in by a user.
- `register_as_participant(username, event_id)`: Registers a student as a participant for an event.
- `register_as_volunteer(username, event_id)`: Registers a student as a volunteer for an event.
- `get_participant_events(username)`: Retrieves events a user is registered in as a participant.
- `get_volunteer_events(username)`: Retrieves events a user is registered in as a volunteer.
- `get_all_events(username)`: Retrieves all events organized by a specific organizer.
- `get_all_events2()`: Retrieves details of all events with venue names.
- `get_event_volunteers(event_id)`: Retrieves volunteers for a specific event.
- `create_new_event(name, type, date, organizer_username, venueid, prize, description)`: Creates a new event.
- `get_event_details(event_id)`: Retrieves details of a specific event.
- `get_events_and_winners(username)`: Retrieves events and winners for a participant.
- `update_winner(event_id, winner_username)`: Updates the winner of an event.

c. Other Operations

- `get_college_names()`: Retrieves names of all colleges.
- `get_food_options()`: Retrieves all food options.
- `get_accommodation_options()`: Retrieves available accommodation options.
- `get_venue()`: Retrieves all venue options.
- `get_venue_from_id(venueid)`: Retrieves venue details based on the venue ID.
- `get_all_participants()`: Retrieves the name and username of all participants.
- `get_organizers()`: Retrieves the name and username of all organizers.

d. Notification and Organizer Approval Operations

- `get_all_notifications()`: Retrieves all notifications.
- `get_organizers_to_allow()`: Retrieves organizers who need approval.
- `update_organizers_allowed_status(organizers_to_allow)`: Updates the approval status of organizers.

e. User Deletion Operations

- `get_users_to_delete()`: Retrieves users (excluding Admin) for potential deletion.
- `delete_users(users_to_delete)`: Deletes selected users from the system.

f. Default System Initialization

- `default_initialization()`: Performs default initialization for the system.

g. Queries for Additional User Details

- `get_participant_details(username)`: Retrieves detailed information about a participant.
- `get_organizer_details(username)`: Retrieves detailed information about an organizer.

h. Event Details and Venue Availability Check

- `get_event_details(event_id)`: Retrieves details of a specific event.
- `check_venue_date(venueid, date)`: Checks if a venue is available on a specific date.

i. Username to Name Mapping Query

- `get_name(username)`: Retrieves the name associated with a username using a view (`username_name_view`).

5. Views

a. username_name_view

- **Purpose:** This view is designed to map usernames to corresponding names, consolidating information from the Student, Participant, Organizer, and Organizers_allowed tables. It provides a unified view for obtaining user details, regardless of their role.
- **Fields:**
  - `username`: The username of the user.
  - `name`: The name associated with the username, extracted from the appropriate table based on the user's role.

b. event_participants_view

- **Purpose:** This view combines information from the Users, Event_participant, Student, and Participant tables. It provides a convenient way to retrieve details about participants in various events, including their usernames and names.
- **Fields:**
  - `username`: The username of the participant.
  - `event_id`: The ID of the event in which the participant is involved.
  - `participant_name`: The name of the participant, extracted from the Student or Participant table.

c. events_with_winners_view

- **Purpose:** This view is designed to capture information about events along with their winners. It joins data from the Event, Event_participant, Student, and Participant tables, providing details about event names, winner usernames, and winner names.
- **Fields:**
  - `event_name`: The name of the event.
  - `winner_username`: The username of the winner.
  - `winner_name`: The name of the winner, extracted from the Student or Participant table.
  - `participant_id`: The ID of the participant associated with the event.

View Initialization on Connection

The `create_views_on_connect` function is executed on database connection to ensure that the views exist. It checks if each view exists, and if not, it creates the view using the provided SQL statements. The `metadata` object is used to reflect the views and facilitate interaction with the database.

6. Forms

- Upon accessing the login page, users are prompted to provide their respective username and password credentials.
- During student signup, students must provide their roll number, name, department, username, password, phone number, and email.
- During participant signup, participants must provide name, college name, username, password, phone number, email along with food and accomodation choices
- During organizer signup organizer must provide name, username, password, phone number, email credentials
- Participants can register for events by selecting from event details displayed alongside checkboxes for each event, facilitating registration as a participant.
- Students can register for events as either participants or volunteers by selecting the respective checkboxes displayed beneath each event.
- Administrator can delete a particular user by selecting the checkbox of that user
- Administrator can approve an organizer by selecting the checkbox of that organizer
- Organizer can create a new event by filling event details like event name, event type, date, venue and description of that event in a form

7. Roles and Access Control

The implemented role-based access control system ensures that users are granted appropriate access privileges based on their roles within the system. Upon login, each user's role is determined and stored in the session data. Subsequently, when accessing different parts of the website, the system verifies if the user's role matches the required role for that particular page. If the user's role is not aligned with the required role, they are redirected to the index page, thus preventing unauthorized access to restricted areas. This approach effectively safeguards sensitive functionality and data, ensuring that users only have access to features and information relevant to their designated roles.