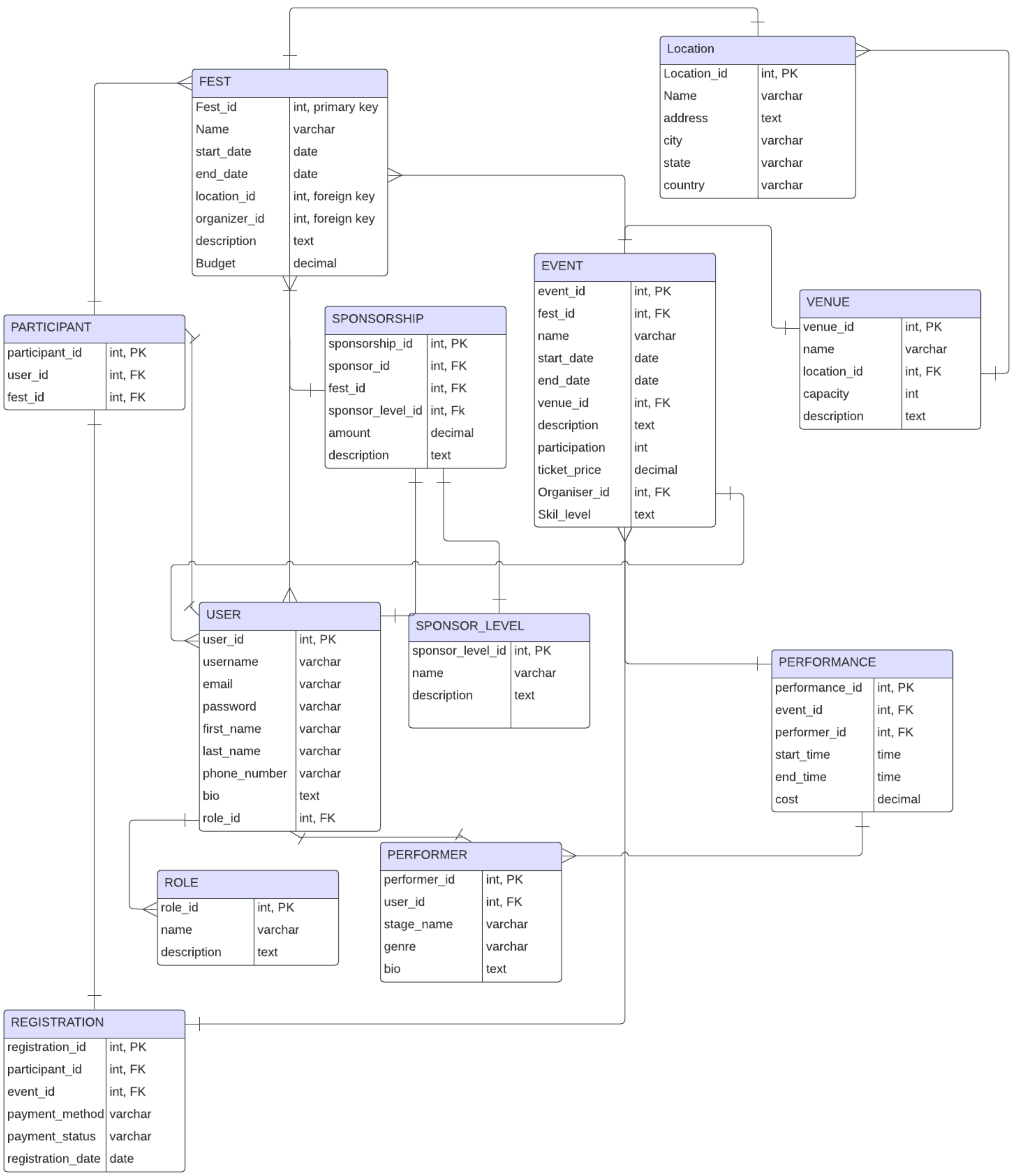
Fest Management System

Yash Aher (142201035) Devansh(142201017) Vaibhav(142201015)



FEST	
Fest_id	int, primary key
Name	varchar
start_date	date
end_date	date
location_id	int, foreign key
organizer_id	int, foreign key
description	text
Budget	decimal

Location	
Location_id	int, PK
Name	varchar
address	text
city	varchar
state	varchar
country	varchar

PARTICIPANT	
participant_id	int, PK
user_id	int, FK
fest_id	int, FK

SPONSORSHIP	
sponsorship_id	int, PK
sponsor_id	int, FK
fest_id	int, FK
sponsor_level_id	int, Fk
amount	decimal
description	text

EVENT	
event_id	int, PK
fest_id	int, FK
name	varchar
start_date	date
end_date	date
venue_id	int, FK
description	text
participation	int
ticket_price	decimal
Organiser_id	int, FK
Skil_level	text

VENUE	
venue_id	int, PK
name	varchar
location_id	int, FK
capacity	int
description	text

USER	
user_id	int, PK
username	varchar
email	varchar
password	varchar
first_name	varchar
last_name	varchar
phone_number	varchar
bio	text
role_id	int, FK

SPONSOR_LEVEL	
sponsor_level_id	int, PK
name	varchar
description	text

PERFORMANCE	
performance_id	int, PK
event_id	int, FK
performer_id	int, FK
start_time	time
end_time	time
cost	decimal

ROLE	
role_id	int, PK
name	varchar
description	text

PERFORMER	
performer_id	int, PK
user_id	int, FK
stage_name	varchar
genre	varchar
bio	text

REGISTRATION	
registration_id	int, PK
participant_id	int, FK
event_id	int, FK
payment_method	varchar
payment_status	varchar
registration_date	date

1. **FEST**:
    - **Fest_id**: Primary key (int)
    - **Name**: Name of the festival (varchar)
    - **Start date**: Date when the fest begins (date)
    - **End date**: Date when the fest ends (date)
    - **Location_id**: Foreign key referencing the location where the fest takes place (int)
    - **Organizer_id**: Foreign key referencing the organizer responsible for the fest (int)
    - **Description**: Textual description of the fest (text)
    - **Budget**: Budget allocated for the fest (decimal)
2. **PARTICIPANT**:
    - **Participant_id**: Primary key (int)
    - **User_id**: Foreign key referencing the user associated with the participant (int)
    - **Fest_id**: Foreign key referencing the fest in which the participant is registered (int)
3. **VENUE**:
    - **Venue_id**: Primary key (int)
    - **Name**: Name of the venue (varchar)
    - **Location_id**: Foreign key referencing the location of the venue (int)
    - **Capacity**: Maximum capacity of the venue (int)
    - **Description**: Description of the venue (text)
4. **USER**:
    - **User_id**: Primary key (int)
    - **Username**: User’s username (varchar)
    - **Email**: User’s email address (varchar)
    - **Password**: User’s password (varchar)
    - **First name**: User’s first name (varchar)
    - **Last name**: User’s last name (varchar)
    - **Phone number**: User’s phone number (varchar)
    - **Bio**: User’s biography (text)
    - **Role_id**: Foreign key referencing the user’s role (int)

5. **SPONSORSHIP**:
   Sponsorship_id: Primary key (int)
   Fest_id: Foreign key referencing the fest associated with the sponsorship (int)
   Sponsor_id: Foreign key referencing the sponsor providing the sponsorship (int)
   Sponsor level_id: Foreign key referencing the sponsorship level (int)
   Amount: Monetary amount of the sponsorship (decimal)
   Description: Description of the sponsorship (text)
6. **SPONSOR LEVEL**:
   Sponsor level_id: Primary key (int)
   Name: Name of the sponsorship level (varchar) ○
   Description: Description of the sponsorship level (text)
7. **PERFORMANCE**:
   Event_id: Foreign key referencing the event associated with the performance (int)
   Performance_id: Primary key (int)
   Performer_id: Foreign key referencing the performer involved in the performance (int)
   Start time: Start time of the performance (timestamp)
   End time: End time of the performance (timestamp)
   Cost : Amount taken by the performer to perform in an event.
8. **PERFORMER**:
   Performer_id: Primary key (int)
   User_id: Foreign key referencing the user associated with the performer (int)
   Stage name: Performer's stage name (varchar)
   Genre: Genre of the performer (varchar)
   Bio: Performer's biography (text)
9. **ROLE**:
   Role_id: Primary key (int)
   Name: Name of the role (varchar)
   Description: Description of the role (text)

10. **REGISTRATION**:
    Registration_id: Primary key (int)
    Participant_id: Foreign key referencing the participant registered for an event (int)
    Event_id: Foreign key referencing the event for which registration is done (int)
    Payment method: Method of payment (varchar)
    Payment status: Status of payment (varchar)
    Registration_date : date of the registration(date)
11. **LOCATION**:
    Location_id: Primary key (int)
    Name: Name of the location (varchar)
    Address: Address of the location (text)
    City: City where the location is situated (varchar)
    State: State where the location is situated (varchar)
    Country: Country where the location is situated (varchar)
12. **EVENT**:
    Event_id: Primary key (int)
    Fest_id: Foreign key referencing the fest associated with the event (int)
    Name: Name of the event (varchar)
    Start date: Date when the event begins (date)
    End date: Date when the event ends (date)
    Venue_id: Foreign Key referencing to Venue of the event (int)
    Description: Description about the event (text)
    Participation: Number of current participants in an event (int)
    Ticket_price: price of the ticket (decimal)
    Organizer_id: Foreign Key referencing to organizer organizing the event
    Skill_level: Skill required to participate in event (text)

FUNCTION :

1. total_profit_from_fest(fest_id int) :
These function takes fest_id as argument and returns the total profit from that fest.

2. total_amount_from_ticket(fest_id int) :
These function takes fest_id as input and returns the total amount obtained from selling ticket for that fest.

3. locations_performance(performerid int):
These function takes performer_id as argument and returns the locations at which the performer has performed.

4. total_amount_from_sponsors(fest_id int):
These function takes fest_id as input and returns the total amount obtained from sponsors of that fest.

5. total_participant_of_fest(fest_id int):
These function takes fest_id as input and returns the total number of participants which participated in that fest.

6. perct_ticket_sponsors(fest_id int):
These function takes fest_id as input and returns the percentage of amount obtained from sponsors and tickets for that fest.

7. total_cost_performance(fest_id int):
These function takes fest_id as input and returns the total cost required for all the performances in that event.

8. top_sponsors(fest_id):

These function takes fest_id as input and returns top sponsor for that particular fest.

9. top_event(fest_id int):
These function takes fest_id as argument and returns the most popular event of the fest.

TRIGGERS :

1. participant_insert : These trigger is called after participant is inserted into the participant table. These trigger creates the views corresponding to relation participant and registration and then grants the required permissions on these views to that participant so that current participant can't access the data of other participants.

2. sponsorship_insert : These trigger is called after sponsor is inserted into the sponsorship table. These trigger creates the view corresponding to relation sponsorship and grants the required permission on these view to sponsor user so that current sponsor can't access and change the data of other sponsors.

3. event_insert : These trigger is called after event is inserted into the event table. Trigger creates the views corresponding to relations event,registration and performance and grants the necessary privilege on them to event organizer user so that current event organizer can’t access and change data of events other than his events.

4. fest_insert : After a new fest is inserted into the fest table, these trigger is called. These trigger generates views corresponding to fest , event and sponsorship relations and grants the specific privileges on these views to the fest organizer so that fest organizer can’t access and change data of fests other than his.

5. performer_insert : These trigger is triggered when a performer is inserted into the performer table. These trigger generates views corresponding to performer and performance relation and grants required privileges on these views to performer user so that it can’t access or change the data of other performer.

6. user_insert : These trigger is triggered when any user of the database is inserted into the users relation. These trigger creates the user role with user’s username as role name and user’s password as role password.

7. upd_budget : These trigger is called after the insert in the sponsor table. It updates the budget of the fest after the new sponsor is inserted.

ROLES :

Group Roles :

1. Super_participant : These is the group role which contains all the privileges which will be granted to all the participants.

Privileges :

1. select(location_id,name,address,city,state,country) on location
2. select(fest_id,name,start_date,end_date,location_id,description) on fest
3. select(event_id,fest_id,name,start_date,end_date,venue_id,description,ticket_price) on event
4. select(venue_id,name,location_id) on venue
5. select(performance_id,event_id,performer_id,start_time,end_time) on performance
6. select(performer_id,stage_name,genre,bio) on performer
7. select(user_id,username,first_name,last_name) on users

2. Super_sponsor : These is a group role which contains all the privileges which will be granted to all the sponsors.

Privileges :

1. select on sponsor_level
2. select(location_id,name,address,city,state,country) on location
3. select(fest_id,name,start_date,end_date,location_id,description) on fest
4. select(event_id,fest_id,name,start_date,end_date,venue_id,description,ticket_price) on event
5. select(venue_id,name,location_id) on venue
6. select(performance_id,event_id,performer_id,start_time,end_time) on performance
7. select(performer_id,stage_name,genre,bio) on performer
8. select(user_id,username,first_name,last_name) on users

3. Super_event_organizers : These is a group role which contains all the privileges which will be granted to all the orgianzers of the event.

Privileges :

1. select,update on venue
2. select(fest_id,name,start_date,end_date,location_id,description) on fest
3. select(location_id,name,address,city,state,country) on location
4. select(performer_id,stage_name,genre,bio) on performer
5. select(user_id,username,first_name,last_name) on users

4. Super_fest_organizers : These is a group role which contains all the privileges which will be granted to all the organizers of the fest.

Privileges :

1. select on location
2. select,update,insert on venue
3. select on role
4. select,update,insert on sponsor_level
5. select on participant
6. select on performance
7. select on registration
8. select,insert on performer

5. Super_performer : These is a group role which contains all the privileges which will be granted to all the performers of the event.

Privileges :

1. select(location_id,name,address,city,state,country) on location
2. select(fest_id,name,start_date,end_date,location_id,description) on fest
3. select(event_id,fest_id,name,start_date,end_date,venue_id,description,ticket_price) on event
4. select(venue_id,name,location_id) on venue
5. select(performance_id,event_id,performer_id,start_time, end_time) on performance.
6. select(user_id,username,first_name,last_name) on users

Roles :

User role : These is the login role using which the users will login in postgres. We will grant privileges to these role based on which type of user it is. The name of the role will be the username using which the user will login.

NOTE : username suffix used below will be different for each user

1. If user is of type participant :

Privileges :

1. select on participant_<username>
2. select,insert on registration_<username>
3. grant super_participant

2. If user is of type sponsor :

Privileges :

1. select,update on sponsorship_<username>
2. grant super_sponsor

3. If user is of type event_organizer :

Privileges :

1. select,update on event_<username>

2. select on eventregistration_<username>
3. select,insert,update,delete on eventperformance_<username>
4. grant super_event_organizer

4. If user is of type fest_organizer :

Privileges :

1. select,update on fest_<username>
2. select,update,insert on festevent_<username>
3. select,delete,update,insert on festsponsors_<username>
4. grant super_fest_organizer

5. If user is of type performer :

Privileges :

1. select,update on performer_<username>
2. select on performance_<username>
3. grant super_performer

VIEWS :

NOTE : These views are created as soon as we insert record in the corresponding relation using triggers.

1. Views for participant :

1. participant_<username> : These is the view using which participant will be able to see only it's participation.

2. registration_<username> : These is the view using which participant will be able to see only it's registration in the events.

2. Views for Sponsors :

1. sponsorship_<username> : These is the view using which the sponsor will be able to select and update only it's sponsorships and not others.

3. Views for event_organizers :

1. event_<username> : These is the view using which the event_organizer will be able to select and update only it's events.

2. eventregistration_<username> : These is the view using which the event_organizer will be able to see registrations of only his events.

3. eventperformance_<username> : These is the view using which the event_organizer will be able to select,insert,update,delete only the performances of his events.

4. Views for fest_organizers :

1. fest_<username> : These is the view using which the fest_organizer will be able to see and update only that fests of which he/she is a organizer.

2. festevent_<username> : These is the view using which the fest organizer will be able to see, update and insert events under his fests.

3. festsponsors_<username> : These is the view using which the fest organizer will be able to see, delete, insert and update sponsors of only his/her fests.

5. Views for performer:

1. performer_<username> : These is the view using which the performer will be able to see and update only it's information from the performer relation.

2. performance_<username> : These is the view using which the performer will be able to see only his performance records from the performance relation.

INDICES :

CREATE INDEX idx_users_user_id_hash ON users USING hash (user_id);

Indexing user_id column using a hash index can be beneficial for certain queries that involve equality checks on the user_id. Hash indexes are particularly efficient for exact matches but might not be as effective for range queries.

CREATE INDEX idx_role_role_id_hash ON role USING hash (role_id);

Similar to the previous index, indexing the role_id column using a hash index can improve query performance, especially for equality checks on the role_id.

CREATE INDEX idx_event_fest_id_event_id ON event USING hash(fest_id, event_id);

This composite hash index on (fest_id,event_id) may be useful for queries that involve filtering or joining based on both fest_id and event_id. Hash indexes are effective for equality checks, so this index could speed up such queries.

CREATE INDEX idx_registration_event_id ON registration USING hash (event_id);

Indexing the event_id column can improve the performance of queries that filter or join based on the event ID. This index can help speed up searches for registrations related to a specific event.

CREATE INDEX idx_sponsorship_fest_id ON sponsorship USING hash (fest_id);

Indexing the fest_id column can facilitate queries related to sponsorships for specific fest. This index can help optimize searches for sponsorships associated with a particular fest.

CREATE INDEX idx_participant_fest_id ON participant USING hash (fest_id);

Indexing the fest_id column can improve query performance for operations related to participants of specific fest. This index can help speed up searches for participants registered for a particular fest.

```
CREATE INDEX idx_fest_fest_id ON fest USING hash (fest_id);
```

Indexing the fest_id column can be beneficial for queries that need to quickly locate fests records by their unique ID. This index can help optimize searches for fests based on their ID.

```
CREATE INDEX idx_performance_event_id ON performance USING
BTREE(event_id);
```

Indexing the event_id column using a B-tree index can improve the performance of queries that filter or join based on the event ID. B-tree indexes are well-suited for range queries and sorting operations, making them suitable for this purpose.