IT-214

Hackathon Management System

(Final Submission)

Team ID: 5.12

202001239 Patel Tirth Nishitkumar
202001244 Masalawala Parth Nileshkumar

Index

Software Requirement Specifications

1. Description/Case Study of the Problem Domain

1.1 Purpose

The purpose of the Hackathon Management System is to build a smooth and untroublesome environment among Hackathon organizers and participants during the process.
The objective of this documentation is to brief the system functionalities and to show an abstract view of the system, which is to be implemented further. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate, and how the system will react to external stimuli. The system's primary purpose will be to simplify the process by setting communication between the host and participants.

1.2 Intended Audience and Reading Suggestions

This documentation is for people such as software developers, programmers, students, hackathon organizers, judges, etc. who will use this system.

Organizers: They can Organize different Hackathons to reach their intended participants and manage the whole data during the hackathon process.

Software developers/ Programmers/ Students: They can find the Hackathons of their interests by using such systems and can get more privileges towards the usage.

1.3 Product Scope

This product can help to resolve ongoing real-time problems during the Hackathon for organizers (Companies, Universities, Governments, etc.) and participants (Programmers, Developers, Students, etc.). The whole system will maximize the productivity and efficiency of the process of the hackathon for both sides with a compatible connection between host and guest. Organizers will be able to gather the intended participants and manage the users' data to fulfill their hackathon purpose; on the other side, the participants will be able to select the competition compatible with their interests. The system will categorize the data for both types of users, hosts, and participants.
The system will help the users to maintain the data of their interests under the defined constraints. The system will act as a mediator for the organizers and participants.

1.4 Description

The system will mediate between two entities(Organizers and Participants). This platform will allow the different organizers to announce various Hackathon events to reach their intended participants. The system will store the data about the hackathon so that people can use it to categorize various competitions by their interests. The participants will register for their interested hackathons, and the organizers will use the input data.
The primary goal is to make it simple for companies and developers to communicate.

Functionalities:

Sign up: The new users (Organizer or Participants) can fill in their asked information which will be stored in the system's database.

Login: The existing users(Organizer or Participants) can enter into their work environment after the information asked into the login section.

For Organizers:

- Accept registrations from participants
- Reject registrations from participants
- Updation
- Regulating details of hackathons
- Access all the details about participants/ users of the system.

For Participants:

- Search tool
- Update profile
- Registration for hackathons
- Submissions
- FAQs

For Judges:

- Search
- Comments and feedback on submissions.
- access submissions

2. Document the Requirements Collection/ Fact-Finding phase

2.1 Reading and Description

By going through the following references and performing background reading on them, we took a brief understanding of how the basic hackathon management system looks like and what functionalities they provide to the different entities(organizers, participants, judges, etc.).

1) **Devfolio Docs: Why Devfolio?**

   Reference: https://guide.devfolio.co/

2) **Mercer | mettl : Online hackathon**

   Reference: https://mettl.com/online-hackathons/

The above considerations have helped us to examine an already existing Hackathon management system. We referred to the views and functionalities that the database system should offer and maintain. We also explored the functions and entities we need to design and maintain and examine the overall relationship between them. We understood the general structure of the database. From these references, we understood the overall requirements of the Hackathon management system and the requirements to be followed for the smooth and untroublesome functioning of the system and to overcome the problems that are faced by the various type of users during the hackathons.

3) Database System Concepts, by Abraham Silberschatz, Henry F. Korth, S. Sudarshan

Reference: https://db-book.com/

From this article, we got familiar with the various DBMS concepts like E-R model, E-R diagrams, Relational model, Mapping of E-R model to Relational model, Database designing and Database development, etc.

4) Survey techniques:

References:
https://www.diva-portal.org/smash/get/diva2:1512603/FULLTEXT01.pdf

Requirements gathered from the Background readings:

- ➔ A well-functioning Database management system is required to maintain and update all the information about athletes, sports, events, leaderboards, etc.
- ➔ The different user interfaces are required for the different user Entities(Organizers, Participants, Judges) so that the particular user can access only the corresponding data and functionalities.
- ➔ System structure and functions should be designed in such a way that it ensures the efficient performance of the system.
- ➔ The interface should be clean and without any redundant data.

Description (Devfolio):

- ➔ For an organization to organize a hackathon, it must register on the website.
- ➔ Participants can find information about the rules of the hackathon, criteria, registration fees, the prize pool, etc.
- ➔ Participants can register based on their interest if the hackathon is live or upcoming. Participants who fulfill eligibility criteria and agree to all conditions can register after paying registration fees (if any).
- ➔ Organizers must select the theme, price pool, mode, criteria, sponsors, and all other information regarding the hackathon.
- ➔ Participants have to submit their work (i.e. Powerpoint, Video demonstration, Code File, Output Snippets, etc.) during all the phases to fulfill the requirement of the hackathon.
- ➔ Students can view the result of the hackathon on the website itself.
- ➔ Moreover, after organizing a hackathon, they will learn about the number of participants, the event's rating, the leaderboard, etc.

2.2 Interview(s)

1) Participants

Interviewer:

- ★ **Tirth Patel** (Co-Founder)
- ★ **Parth Masalawala** (Co-Founder)

Interviewee:

→ **Gangaraju Bopparam**

Date: 01/10/2022 **Time: 17:00**
Duration: 30 mins **Place: DA-IICT**

1) Why are you interested in participating in the hackathon?
2) How often do you participate in hackathons?
3) Which mode do you like the most? Online or offline.
4) What types of problems do you encounter during the whole process?

Summary of Results and Requirements:

- ➔ Current submission status in different phases.
- ➔ There should be accessible communication between the participants and the organizers.
- ➔ Comments regarding the judge's judgment (feedback), so one can enhance their performance.
- ➔ Details about the hackathons, events, and winners of various events.
- ➔ Project details of previous top submissions.
- ➔ Efficient, optimal, and reliable services.

2) Organizers

Interviewer:

- ★ **Tirth Patel** (Co-Founder)
- ★ **Parth Masalawala** (Co-Founder)

Interviewee:

→ **Devansh Patel**

Date: 01/10/2022 **Time: 19:00**
Duration: 40 mins **Place: DA-IICT**

1) How many participants would you expect?
2) Which type of platforms have been used by you?
3) What are your main expectations toward such systems?
4) What types of problems have you faced so far?
5) What are the general Eligibility criteria for the participants?
6) What new features are you willing to have in your system?

Summary of Results and Requirements:

- ➔ Efficient, optimal, and reliable services.
- ➔ Security, consistency, and integrity of the data stored in the database.
- ➔ Analysis of the social and economic impact on the cities or countries hosting the Olympics events.
- ➔ Details about the viewership so that they can negotiate for better sponsorship and broadcasting rights.
- ➔ Efficient statistical analysis of the performance of each nation in order to get an idea about how favorable it could be for the nation hosting the Olympics over other nations.

3) Judges

Interviewer:

- ★ **Tirth Patel** (Co-Founder)
- ★ **Parth Masalawala** (Co-Founder)

Interviewee:

→ **Jaydeep Mulherkar**

Date: 02/10/2022 **Time: 15:00**
Duration: 45 mins **Place: Faculty Block-1, DAIICT.**

1) What characteristics do you observe in the submissions of participants?
2) How do you categorize the submissions?
3) What will be your criteria for the judgments?
4) Would you like to give us any suggestions which can help you to declare the results?

Summary of Results and Requirements:

- ➔ Submissions should be easy to access.
- ➔ Plagiarized work should be directly disqualified.
- ➔ The incomplete work and the complete work should be distinguished under provided constraints to make the judgment process faster.

4) Sponsor

Interviewer:

- ★ **Tirth Patel** (Co-Founder)
- ★ **Parth Masalawala** (Co-Founder)

Interviewee:

→ **Devansh Patel**

Date: 02/10/2022 **Time: 14:00**
Duration: 15 mins **Place: DA-IICT**

1) What is the theme for the hackathon?
2) How do you distinguish between good hackathons?
3) What is your primary goal for sponsoring the hackathon?
4) What primary information do you need about
5) What types of services, prizes or goodies do you sponsor?

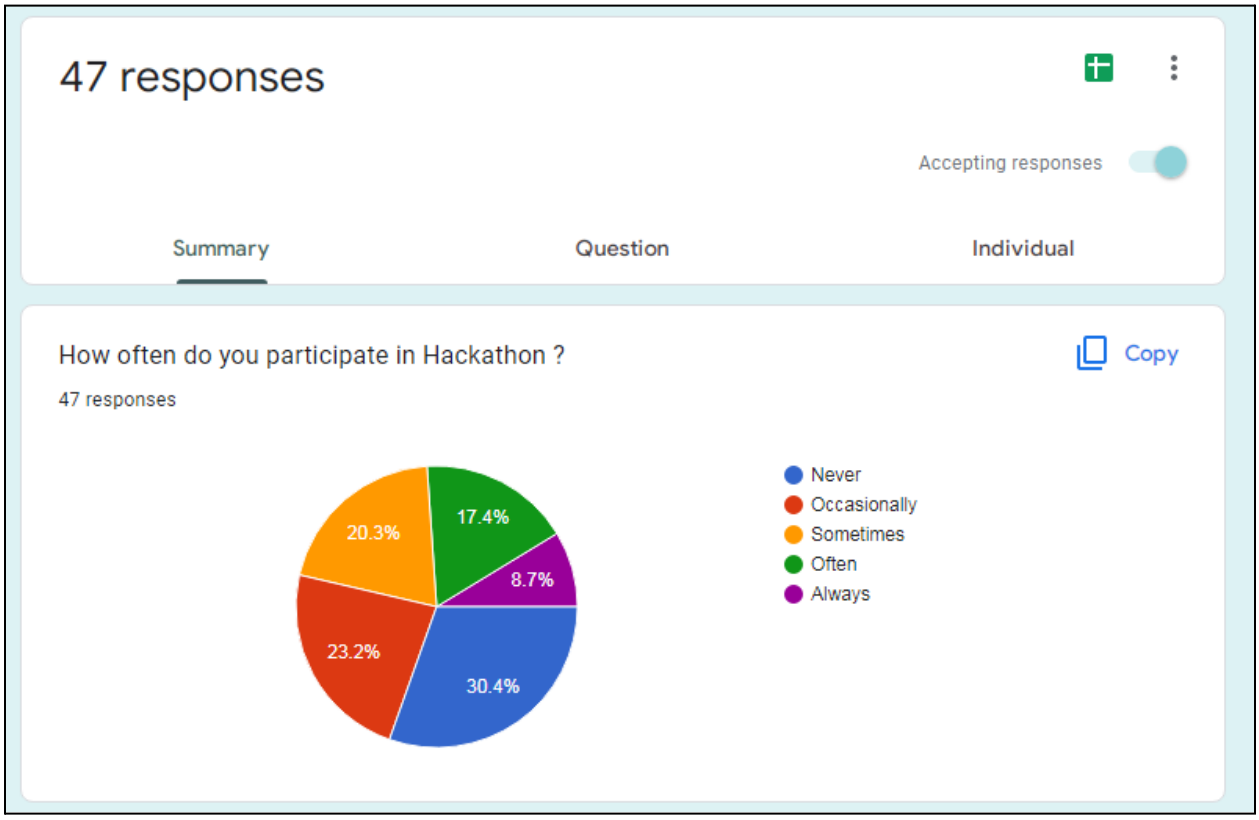
Summary of Results and Requirements:

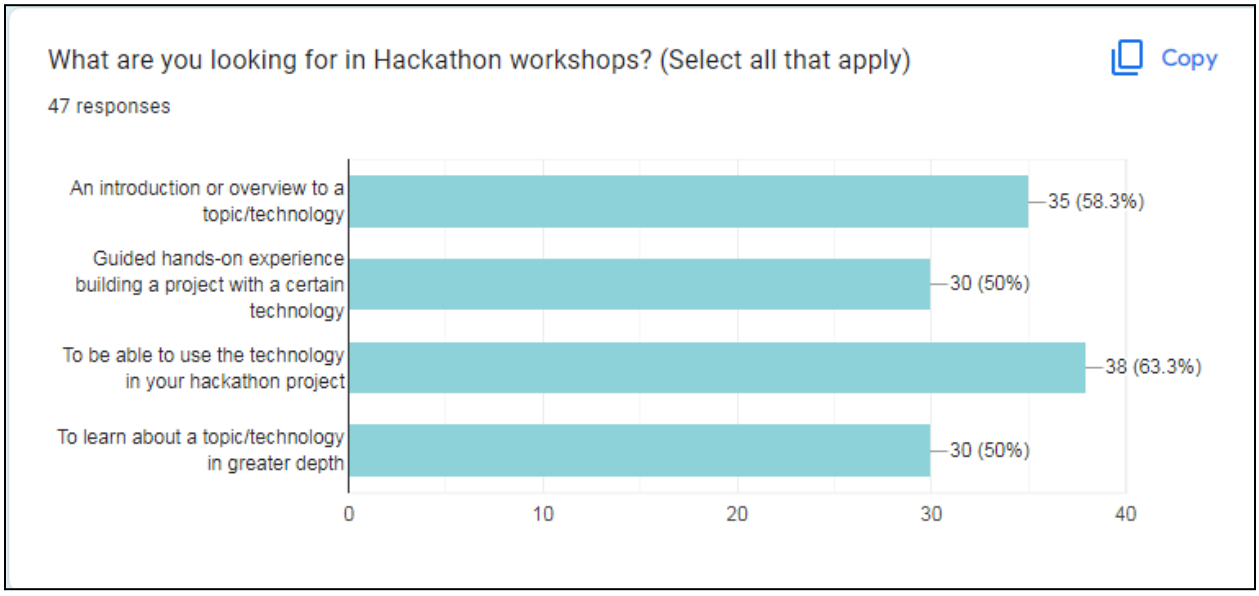
➔ Sponsors get some requests to sponsor travel expenses for participants during the offline events up to some extent.

Requirements gathered from the Interviews:

→ A well-functioning Database management system is required to maintain and update all the information about athletes, sports, events, leaderboards, etc.
→ The different user interfaces are required for the different user Entities(Organizers, Participants, Judges) so that the particular user can access only the corresponding data and functionalities.
→ System structure and functions should be designed in such a way that it ensures the efficient performance of the system.
→ The interface should be clean and without any redundant data.

2.3 Questionnaires

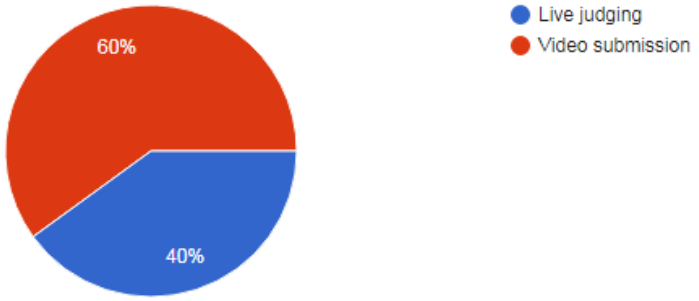




How would you prefer to have your project be judged at a hackathon?

Copy

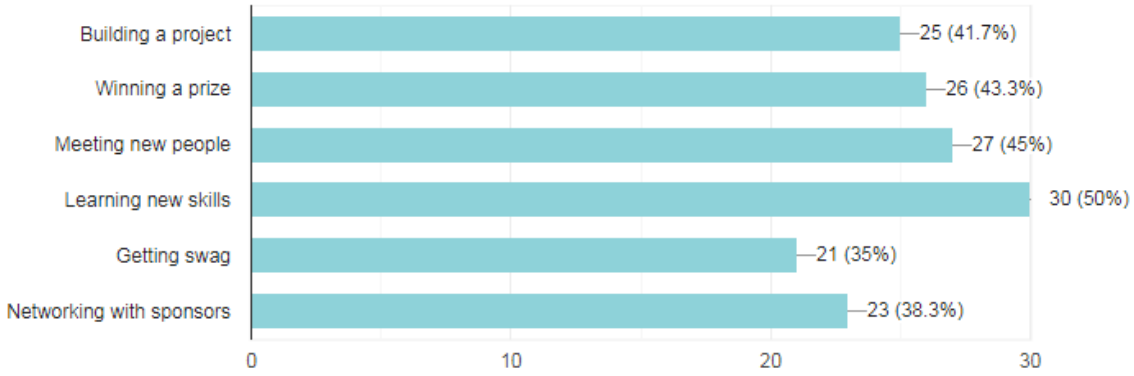
47 responses



What are the most valuable aspects of a hackathon to you? (Select all that apply)

Copy

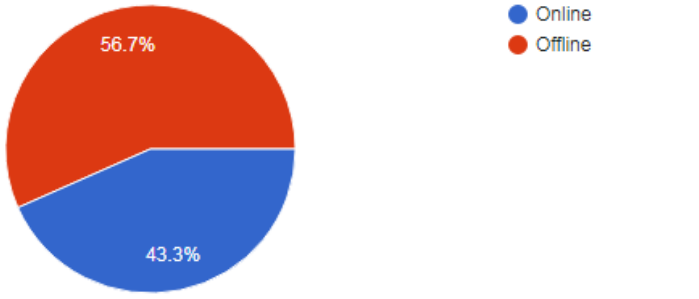
47 responses

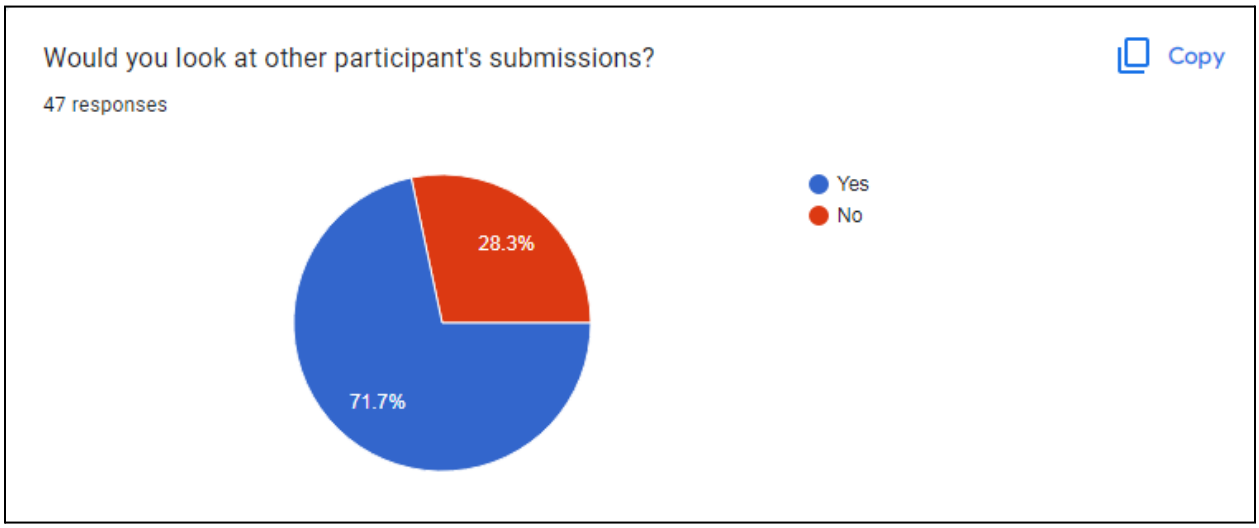


What would you prefer online or offline Hackathon?

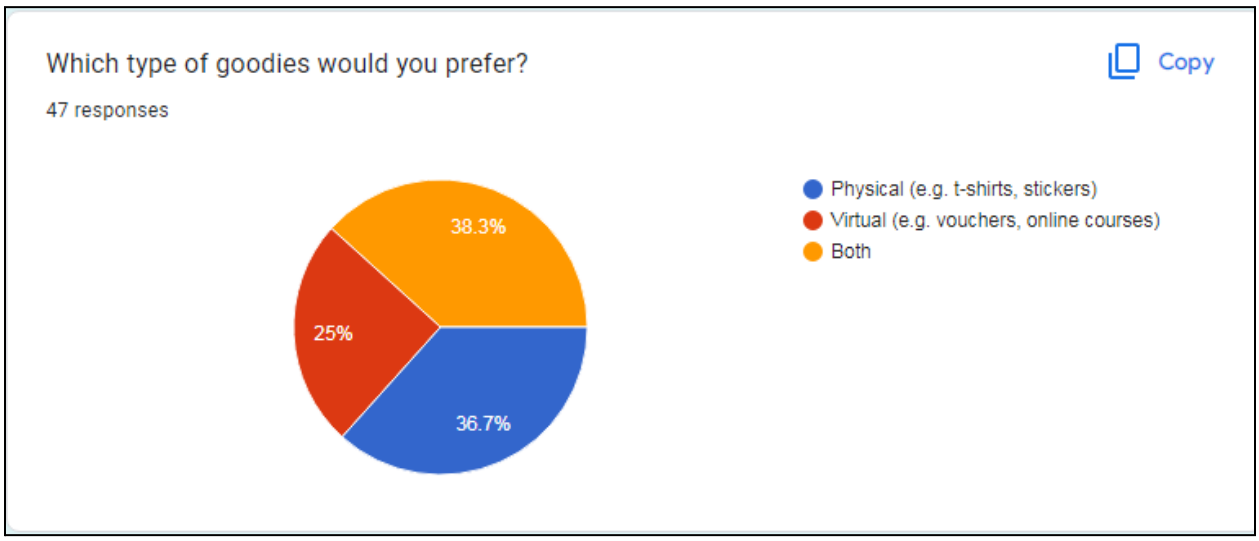
Copy

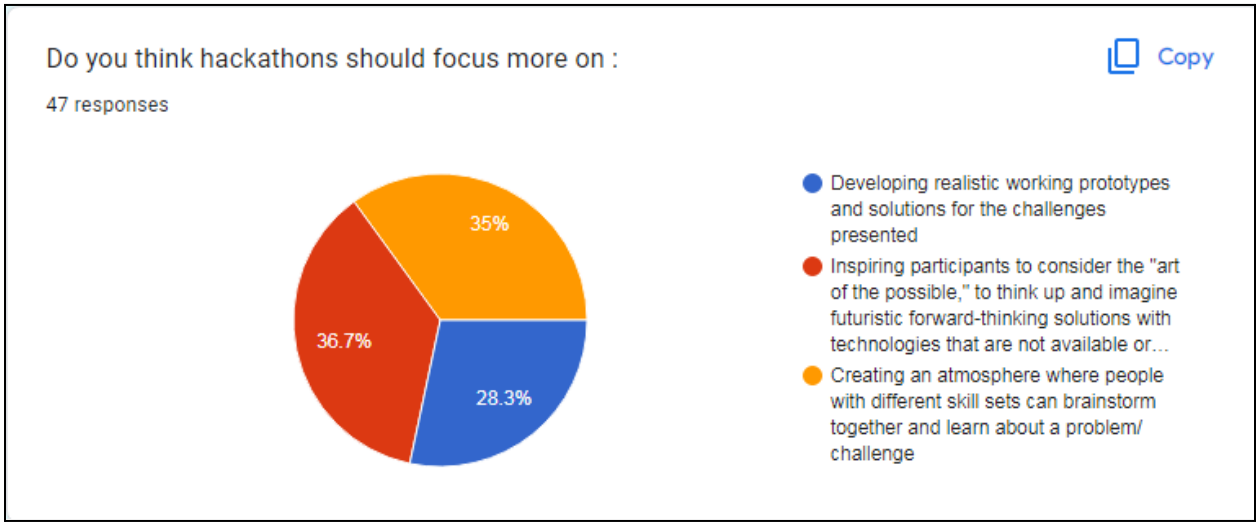
47 responses





→ A participant should be able to see other's project and their status.





Combined requirements gathered from the Questionnaire:

- ➔ The database system to be developed should have a good UI and should provide efficient services.
- ➔ The multiple teammates should be able to submit the submissions.
- ➔ Regular updates of the data stored in the database system.
- ➔ User feedback on a regular basis.
- ➔ Incorporation of the additional requirements proposed by the users during the feedback.
- ➔ In the online hackathon there must be compatible communication between the participants and the organizers.

2.4 Observations

- ➔ The participants were not able to communicate with the organizers during the hackathon.
- ➔ The judgment was time-consuming as some participants' projects didn’t follow the instructions, such submissions should be discarded automatically.
- ➔ The participants were not able to get comments and feedback from judges regarding their submissions to enhance their skills.
- ➔ The entire data wasn’t accessible to everyone; privacy and security of the data were given priority.
- ➔ On the home page of each participant, different suggestions for hackathons could be seen based on the activity (previous participation, profile, and search history) of the participant.
- ➔ Real-time updating of the leaderboard was done in the system to ensure that everyone gets information about the current scenario.
- ➔ All the information about a particular hackathon (like themes, prizes, rules, teams, time, deadlines, etc) should be shown on the description page of that hackathon.
- ➔ The system should be optimized and fast enough to handle multiple accesses to data. Concurrency should be taken care of.
- ➔ Efficient services related to the registration, search and upload operations should be there.

3) Fact Finding Chart:

Objective	Technique	Subject(s)	Time Commitment
To get the background of the Hackathon Management System and format of the SRS	Background Reading	Website, E-Book	1 Day
To gain an understanding of the roles of Sponsors	Interview	Organizers	45 mins
To find out the roles of organizer	Interview	Hackout Organizers	1 hour
To gain an understanding of the roles of Judge	Interview	Judge	30 mins
To determine the difficulty faced by the participant's	Questionnaire	Hackout User	2 hour
To gain an understanding of the real world Hackathon Management database	Background Readings, Observation	Devfolio Website	1 hour

4) List Requirements

- A well-functioning Database is required to maintain and update all the information about stored data.
- The database should be consistent and without any redundant data.
- The different user interfaces are required for the other user entities (Organizers, Participants, Judges) so that the particular user can access only the corresponding data and functionalities.
- The general viewer should be able to see the dashboard presenting upcoming and past events.
- The Organizers should be able to register for their upcoming hackathons easily.
- The Organizers should be able to access all the details of the participants.
- The participants should be able to see the corresponding details of upcoming and past events.
- The participants should be able to see the status of their submissions.
- Participants should be able to see the comments and feedback on their submissions given by judges.
- The participants should be able to give feedback and suggestions at the end of the hackathon event.
- There should be accessible communication between the participants and the hackathon organizer to resolve their queries during the hackathons.
- Judges should be able to access all the information about participants. (Including submissions)

- Participants should be able to see others' performances and leaderboards.
- The interface should be clean and without any redundant data.
- Real-time update of the information stored in the database system.
- The integrity of different system parts should be maintained in real-time.
- The database should be updated, and integrity should be maintained in real-time. Also, a backup should be provided for our database in case of inconsistency/crash.
- System structure and functions should be designed in such a way that it ensures the fast performance of the system.
- There should be FAQs and a discussion section.

5) User Categories and Privileges

→ List of user categories and their roles:

1) Participants:
The role of the Participants is to explore the upcoming Hackathon events and to find the hackathons of their interests.

2) Organizers:
The role of the organizers is to organize various Hackathon events providing all the details about the event on the platform.

3) Judges:
The role of the Judges is to analyze each submission associated with the hackathons they are judging and announce the results of the event.

4) General Visitor:
All the users will be considered general visitors on the first visit. The general visitors will be able to see the upcoming and past events. They won't have any major privileges.

5) Database system manager:
The role of the database system manager is to look after the technical issues faced by the users, control the traffic over the database system and regularly update the data stored in the database system, maintaining the consistency and integrity of the database.

List of privileges/functions that can be accessed by different user classes

1] General visitor:

- ➔ All the users will be considered general visitors at the first visit.
- ➔ They can see the upcoming and past events on their interface.
- ➔ They will be categorized under three different user categories: organizers, **participants** and **judges**. Each of these users will have **Sign up/Login** options to proceed further.
- ➔ One can select a category corresponding to his/her usage and can proceed further providing the requested information under the **Sign-up/Log-in** section.

2] Organizer:

- ➔ The User Interface for the organizer will guide them to organize the hackathon.
- ➔ Organizers can log in to their existing account by providing asked information and password.
- ➔ Organizers can announce the hackathons by providing its details, themes, etc.
- ➔ They can access all the information about the participants participating in their hackathons.
- ➔ They can **accept** or **reject** the application of the candidates.
- ➔ They can set the name of sponsors and their details and details about the judges.
- ➔ They will be able to communicate with the participants.
- ➔ They can search, update, delete, insert, etc. to regulate the data.

3] Participants:

- ➔ The User Interface for the participants will navigate them toward the **lists of upcoming events** and events that have been finished.

- ➔ They will be provided with their profile and they can update the details in the profile.
- ➔ Participants will be able to keep the data of their past participation.
- ➔ Participants will be given a **search** tool to find the hackathon of their interest.
- ➔ Check out submissions made to past hackathons.
- ➔ Participants will be able to get **feedback and comments** on their submissions given by the judges.
- ➔ Participants will be able to put their queries to the organizers during the hackathon to resolve their problems.
- ➔ There will also be some **general functionalities** useful to the users.

4] Judges:

- ➔ The User Interface for the judges will navigate them toward the submissions for the corresponding hackathons.
- ➔ They will be given access to the activities of participants.
- ➔ The system will provide the functionality to distinguish between valid or invalid submissions to make the process of judgment easy under given conditions.
- ➔ Judges will be given the privilege to go through the submissions of the participants. They can make appropriate comments on them as well as give feedback.
- ➔ They will be able to announce results to the participants.

6) Assumptions:

- ➔ The users of this database system are presumed to have all the requisite hardware and software to operate this application.
- ➔ Furthermore, it is presumed that the database's data is updated in real-time.
- ➔ The database system's users are presumed to have reliable internet connections.
- ➔ The database is consistent.
- ➔ The database maintains the data's integrity all the time.
- ➔ In rare circumstances, the users will have access to substitute resources.
- ➔ The internet connection at the server is strong enough to handle all the queries from users without crashing and no loss of data occurs due to any technical failures.

7) Business Constraints:

- ➔ The users' data can be stored upto at most 5-6 times.
- ➔ The computing power available for the database system is also limited.
- ➔ The capacity of the database is limited.
- ➔ The number of users accessing the database simultaneously is also limited.
- ➔ The amount of software and hardware required for the database system is limited.

Noun Analysis, ERD

Description:

1) **USERS:**

The hackathon management system provides functionalities for organizing hackathons and participating in the hackathons. It also provides features to make the whole process smooth and untroublesome.
The Hackathon management system provides a compatible connection between the companies who organize hackathons and the participants.
A general user can visit the system UI/UX where they can see upcoming and past hackathons upto some limit. They will get two options,

1. **Sign-up**
2. **Log-in**

The new users will be able to create their account using sign-up options providing required information.
The existing users will be able to enter their work environment using the log-in option and providing requested information consisting of a unique key attribute which will distinguish them from the other existing users.
In the work environment of existing users they will be given three options to proceed further,

1. **Organize Hackathon**
2. **Participate Hackathon**

3. **Evaluate Hackathon**

As per the users role they will select an option.

2) **ORGANIZERS:**

If an existing user selects the option to organize a hackathon. he/she will be asked to fill up details about the hackathon.

➔ **Mode of hackathon (offline/online)**
➔ **Location (if offline)**
➔ **Date & time**
➔ **Criteria**
➔ **Participant limit**
➔ **Duration of the hackathon**
➔ **Prizes (if any)**
➔ **Details about sponsors**
➔ **Details about judges**
➔ **Theme of hackathon**
➔ **Submission format**

- The User Interface for the organizer will guide them to organize the hackathon.
- Organizers can log in to their existing account by providing asked information and password.
- Organizers can announce the hackathons by providing its details, themes, etc.
- Also they will add criteria(if any) while organizing the hackathon
- They will add all the information about the sponsors and also of judges so judes can log in easily.
- They can access all the information about the participants participating in their hackathons.

- They can **accept** or **reject** the application of the candidates.
- They will be able to communicate with the participants so they can solve the query of participants.
- They can search, update, delete, insert, etc. to regulate the data.
- Also organizers will make a unique key so judges can login through that key.

3) PARTICIPANTS:

If an existing user selects the option to participate in a hackathon he will be navigated to the page where he/she will be able to see upcoming events and past events.

❖ **Upcoming events:**

➔ **Details about the hackathon**
➔ **Details about the sponsors**
➔ **Details about the judges**
➔ **Registration**
- **Email id**
- **Password**
- **User name for hackathon**
- **Team size (if any)**
- **Team name (if any)**
- **Details about the team**

❖ **Past events:**

➔ **Details about the hackathon**
➔ **Details about the sponsors**
➔ **Details about the judges**
➔ **Registration**
- **Email id**
- **Password**
- **User name for hackathon**
- **Team size (if any)**
- **Team name (if any)**
- **Details about the team**

- The User Interface for the participants will navigate them toward the **lists of upcoming events** and events that have been finished.
- They will be provided with their profile and they can update the details in the profile.
- Participants will be able to keep the data of their past participation.
- Participants will be given a **search** tool to find the hackathon of their interest.
- Check out submissions made to past hackathons.
- Participants will be able to get **feedback and comments** on their submissions given by the judges.
- Participants will be able to put their queries to the organizers during the hackathon to resolve their problems.
- There will also be some **general functionalities** useful to the users.

After the registration:

- After registration, the participant's team will get a regular update of the hackathon and get a reminder of the hackathon.
- Participants can see all details of the hackathon in which they have registered.
- Their past hackathon standing/point in which they had registered.

4) Judges:

If the user selects the option to evaluate the submissions.
They have to fill in the details about the hackathon for which they are evaluating.

❖ **Judges Interface:**

➔ **Login through key**
➔ **Details about the participants**
➔ **All the participant's submissions**
➔ **Only valid submissions**
➔ **Feedback to the participants submissions**
➔ **Result announcement**

- First they will log in through a unique ID that is given by the organizer.
- Then User Interface for the judges will navigate them toward the submissions for the corresponding hackathons.
- They will be given access to the activities of participants.
- The system will provide the functionality to distinguish between valid or invalid submissions to make the process of judgment easy under given conditions.
- Judges will be given the privilege to go through the submissions of the participants. They can make appropriate comments on them as well as give feedback.
- They will be able to announce results to the participants.

2) **Noun (& Verb) Analysis**

Nouns	**Verbs**
You	designing (Hackathon Management system)
Participants	Participate in hackathon
Judges	Evaluates the work
system	Provides the system
location	Provides location about participants
connection	Provides connections
companies	Companies with problem statement
Sponsors	Sponsor the hackathon
Organizers	Manages the hackathon
backup	Provides Backup of data
account	Account info users
Information	Provides data about participants
feedback	Feedback from participants
price	Available price money for winners
Code	Provided Code by participants
FAQs	Provides frequently asked questions
structure	constructs
Details	Provides user info
section	differentiates
data	Provides data of users

cash	Indicates cash payment
UI/UX	User interface
conditions	restricts
mode	Online / offline
Snippets	Pieces of code
Video	Video streaming
Duration	Defines time length
password	Password of users
environment	Provides work area
work	
mode	Offline / online
Theme	Defines subject
Submission	Provides work
format	Defines the format of submission
options	Provides choices
users	Uses system
account	Keeps user info
Powerpoint	Presents final work
requirement	identifies constraints
information	Provides info
Log-in	Provides a way to log-in
criteria	Provides constraints
options	Indicates various choices

demonstration	Demonstrates the work
queries	Inquires
time	Provides time

Rejected Nouns

Noun	**Reject Reason**
price	Irrelevant
connection	vague
options	redundancy
mode	duplicate
UI/UX	Irrelevant
Organizers	Irrelevant
backup	vague
feedback	redundancy
Code	duplicate
FAQs	General
data	redundancy
cash	duplicate
section	redundancy
conditions	Irrelevant
structure	vague

mode	duplicate
Snippets	irrelevant
Video	vague
Duration	redundancy
environment	redundancy
work	Irrelevant
Theme	vague
Submission	vague
format	redundancy
options	duplicate
users	redundancy
account	vague
Details	redundancy
Powerpoint	duplicate
requirement	General
information	redundancy
Log-in	vague
criteria	redundancy
options	duplicate
demonstration	irrelevant
queries	redundancy

Selected Nouns

Candidate entity set	Candidate attribute set	Candidate relationship set
team	team_id, team_name, no_member, leader, hackathon_id, submission	hackathon_management
sponsors	Name , sponsor id , hackathon id, amount	sponser_hackathon
participant	name, p_id, domain, phone_number, company_id, email_id	hackathon_company
hackathon	Start_time, name , end_time, hackathon id	Judge_hackathon , sponser_hackathon , hackathon_company , Hackathon_management , team_hackathon
judges	judge_id, name, hackathon_id	judge_hackathon
company	company_id, name, type_of_company	team_participant
management_team	team_id, team_name, no_of_member, contact_no	hackathon_management

1) Identify Entity types.

- ☐ In the relationship between hackathon and sponsors, this weak entity relation sponsor is the weak entity, and the name is the partial key.
- ☐ Hackathon is a strong entity in this weak relation, and Hack_id is identifying.
- ☐ Also, about hackathons, Judge is a weak relation, and Judge is a weak entity. Also, the name is a partial key to this weak relation.
- ☐ In the above weak relation, Hack_id is the identifying key, whereas Hackathon is the strong entity.

2) Identify Relationship types.

Entity vs. Attribute:

Entity	Attribute
User	Age, User_Id, DOB, Password, Contact_Info, Email_Id, User_name
Organizer	Company_Id, Name
Team	Leader, Team_Id, Current_stat, Team_Size
Participant	User_id, Hack_id, Domain
Judge	Hack_id, Judge_id, Name
Submission	Hack_id, Team_id, Time, curr_state
Sponsors	Sponsor_id, Name
Hackathon	Hack_id, date, start_time, end_time, duration, theme

Evaluation	Team_id, Hack_id, Time, Judge_id, Comment

Binary vs. Ternary Relationships:

All relations between all the entities are binary in the ER diagram.

Team_mates(relationship between Team and Participant)
- ☐ This is a one-to-many relationship.

Judes(relationship between Judge and Hackathon)
- ☐ This is a many-to-many relationship.

Organizes(relationship between Organizers and Hackathon)
- ☐ This is a one-to-many relationship.

Sponsors(relationship between Sponsor and Hackathon)
- ☐ This is a many-to-many relationship.

Evaluate(relationship between Judge and Submission)
- ☐ This is a one-to-many relationship.

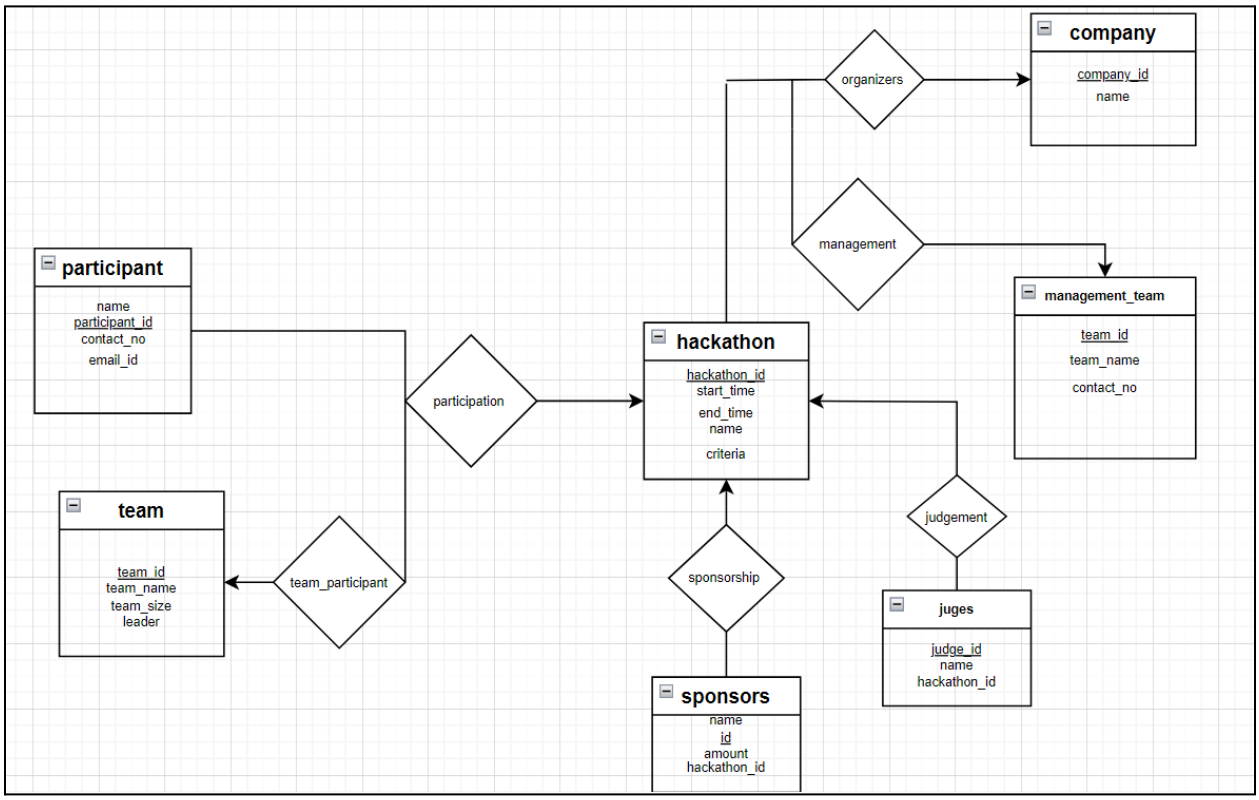
Submit(relationship between Team and submission)
- ☐ This is a one-to-many relationship.

3) Analyze ERD for any other missing information.

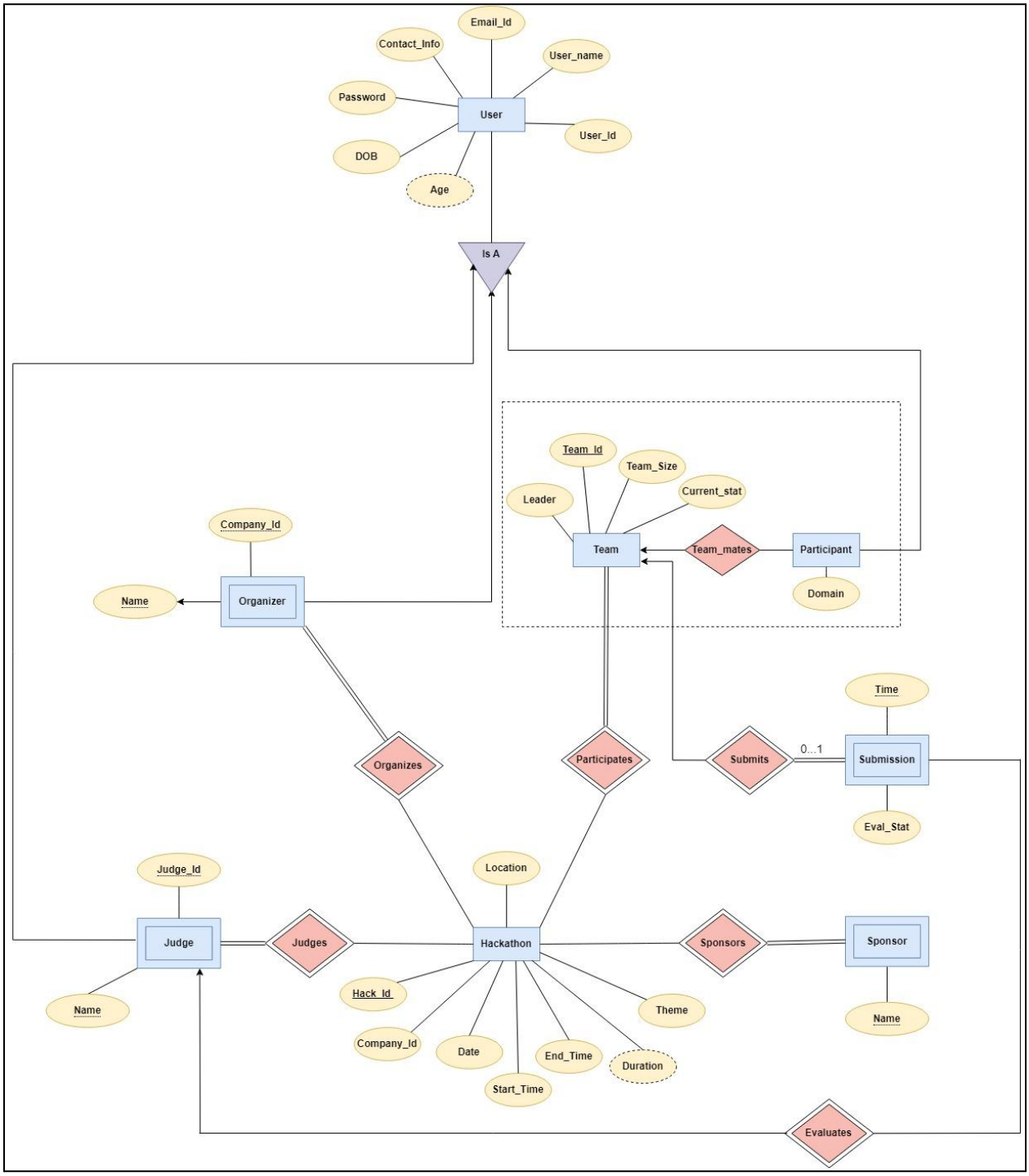
- ☐ All entities are the same as before in the latest ER diagram.
- ☐ We added a **derived** attribute(Duration).
- ☐ We have added some attributes like team_size and removed some redundant attributes like email, contact, etc.
- ☐ Instead of extra attributes, we generalized that attribute and removed redundancy.
- ☐ We added a weak entity relationship between two relations and defined partial and identifying keys for this relation.
- ☐ Also, we add ISA **specialization** from regular users to participants, organizers, judges, and sponsors.
- ☐ We make Team entity **aggregation** as all other entities are related to the Team entity.
- ☐ The judge entity has two extra attributes from the user entity, such as judge_id and name.
- ☐ Similarly, the organizer entity has more company_id and Name attributes than the user entity.
- ☐ The Hackathon entity holds various information related to the ongoing hackathon in our system (i.e., hack_id, date, start time, end time, duration, etc.).
- ☐ ER diagram also includes **weak entities** such as organizer, judge, participant, submission, and sponsor.

- ☐ The team and participants combined create a separate entity about a hackathon entity. The concept used in this procedure is aggregation.

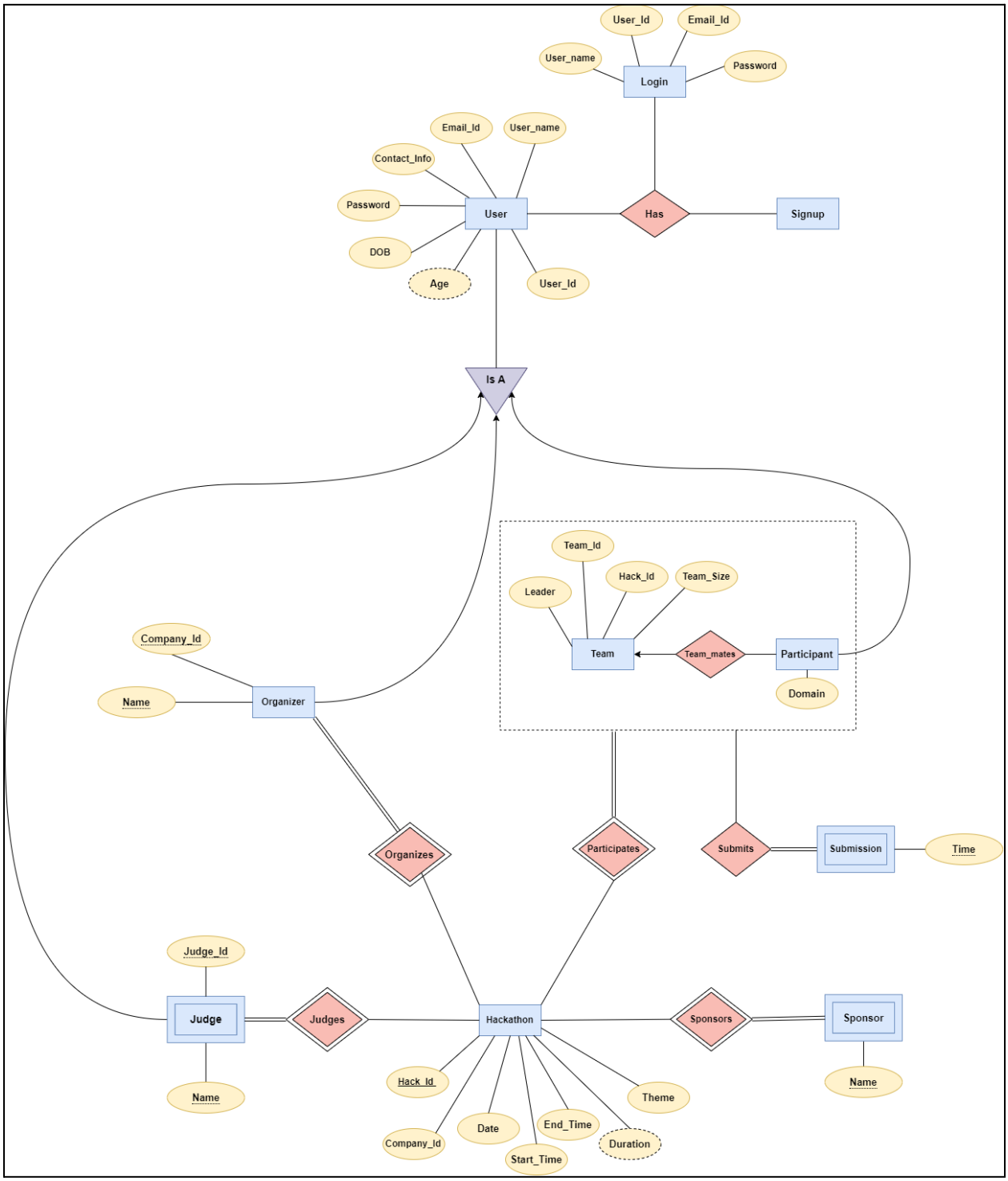
First version of ER diagram:



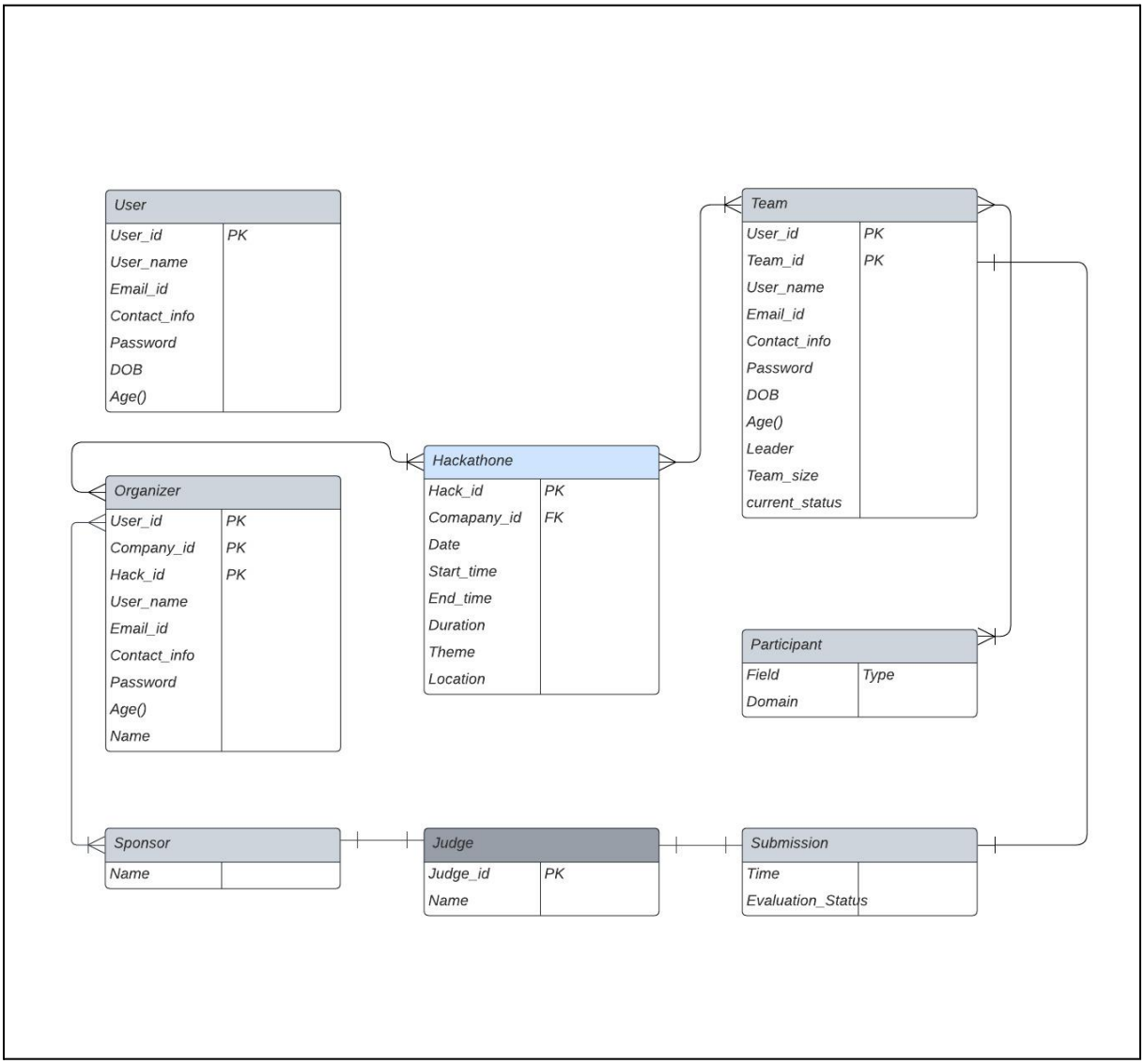
Second version of ER diagram:



Final version of ER diagram:



ERD to Relational model



→ Schema Before Refinement:

- **User**(User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age)
- **Organizer**(Hack_id , Company_id,User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age, Name)
- **Team**(Hack_id , Team_Id , Team_size , Leader , Current_Status)
    - FK Team_id references to **Participant**
- **Participant**(Tean_id, User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age , Domain)
- **Submission**(Hack_id, Time , Evaluation_Status)
- **Sponsor**(Hack_id , Name)
- **Judge**(Hack_id , Judge_Id ,User_I d , User_name , Email_Id , Contact_info , Password , DOB , Age, Name)
- **Hackathon**(Hack_id , Comapny_id , date , start_time , end_time , duration , theme , location)
    - FK Company_id references to **Organizer**

- **User:**
  - **User_id** → User_id, User_name , Email_Id , Contact_info , Password , DOB , Age
  - **DOB** → Age

- **Organizer**
  - **Hack_id , Company_id,User_Id** → User_name , Email_Id , Contact_info , Password , DOB , Age, Name
  - **DOB** → Age

- **Team**
  - **Hack_id , Team_Id** → Team_size , Leader , Current_Status
  - FK Team_id references to **Participant**

- **Participant**
  - **Team_id, User_Id** → User_name , Email_Id , Contact_info , Password , DOB , Age , Domain
  - **DOB** → Age

- **Submission**
  - **Hack_id, Team_id, Time** → Evaluation_Status

- **Sponsor**
  - **Hack_id , Name ->** Hack_id , Name

- **Hackathon**
  - **Hack_id** → Comapny_id , date , start_time , end_time , duration , theme , location
  - FK Company_id references to **Organizer**

- **Judge**
  - **Hack_id , Judge_Id ,User_Id** → User_name , Email_Id , Contact_info , Password , DOB , Age, Name
  - **DOB** → Age

Normalization and Schema Refinement

Dependencies and Normal forms

1. User (User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age)

PK dependency:
User_id → User_name, Email_Id , Contact_info , Password , DOB , Age

Functional Dependencies:
User_id→ User_id
User_id→ User_name
User_id→ Email_id
User_id→ mobile
User_id→ password
User_id→ DOB
User_id→ Age
DOB→ Age

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.

Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary

key is being accessed as a foreign key in other tables.

In this schema, every attribute is single-valued (scalar) making it already in 1NF. There is no partial dependency here, so it is in 2NF as well.

2NF Redundancies: None

Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.
For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

2. Organizer (Hack_id , Company_id, User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age, Name)

PK dependency:

Hack_id, Company_id, User_Id → User_name, Email_Id, Contact_info , Password , DOB , Age, Name

Functional Dependencies:

User_id→ User_name
User_id→ Email_id
User_id→ Contact_info
User_id→ password
User_id→ DOB
User_id→ Age
User_id→ User_id
DOB→ Age

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.

There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.
For a relation to be in BCNF,
a. It should be in the third normal form (3NF).

3. Participants(Team_id, User_Id, User_name , Email_Id , Contact_info , Password , DOB , Age , Domain)

PK dependency:

Team_id,User_Id → User_name, Email_Id, Contact_info , Password , DOB , Age, Name

Functional Dependencies:

User_id→ User_name
User_id→ Email_id
User_id→ Contact_info
User_id→ password
User_id→ DOB
User_id→ Age

User_id→ User_id
DOB→ Age

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.
For a relation to be in BCNF,
a. It should be in the third normal form (3NF).

4. Hackathon (Hack_id , Comapny_id , date , start_time , end_time , duration , theme , location)

PK dependency:
Hack_id → Comapny_id , date , start_time , end_time , duration , theme , location

Functional Dependencies:

Hack_id→ Hack_id
Hack_id→ Comapny_id
Hack_id→ date
Hack_id→ start_time

Hack_id→ end_time
Hack_id→ duration
Hack_id→ theme
Hack_id→ location

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.
For a relation to be in BCNF,
a. It should be in the third normal form (3NF)

5. Submission (Hack_id, Team_id, Time, Evaluation_Status)

PK dependency:
Hack_id, Team_id, Time → Evaluation_Status

Functional Dependencies:
Hack_id, Team_id, Time → Hack_id, Team_id, Time, Evaluation_Status

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

6. Judge (Hack_id , Judge_Id ,User_Id , User_name , Email_Id , Contact_info , Password , DOB , Age, Name)

PK dependency:
Hack_id , Judge_Id ,User_Id → User_name , Email_Id , Contact_info , Password , DOB , Age, Name

Functional Dependencies:
Hack_id , Judge_Id ,User_Id → Hack_id , Judge_Id ,User_Id **,** User_name , Email_Id , Contact_info , Password , DOB , Age, Name

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

7. Team (Hack_id , Team_Id , Team_size , Leader , Current_Status)

PK dependency:
Hack_id , Team_id→ Team_size , Leader , Current_Status

Functional Dependencies:
Hack_id , Team_id → Hack_id , Team_id , Team_size , Leader , Current_Status

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.

Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

8. Sponsor(Hack_id ,Sponser_id, Name)

PK dependency:
Hack_id , Name → Hack_id , Name

Functional Dependencies:
Hack_id , Team_id → Hack_id , Name

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.

2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

9. Evaluate(hack_id , team_id , judge_id , comment)

PK dependency:
Hack_id , team_id → judge_id , comment

Functional Dependencies:
Hack_id , Team_id → Judge_id , comment

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

10. Team_info(Hack_id ,user_id, Team_id)

PK dependency:
None

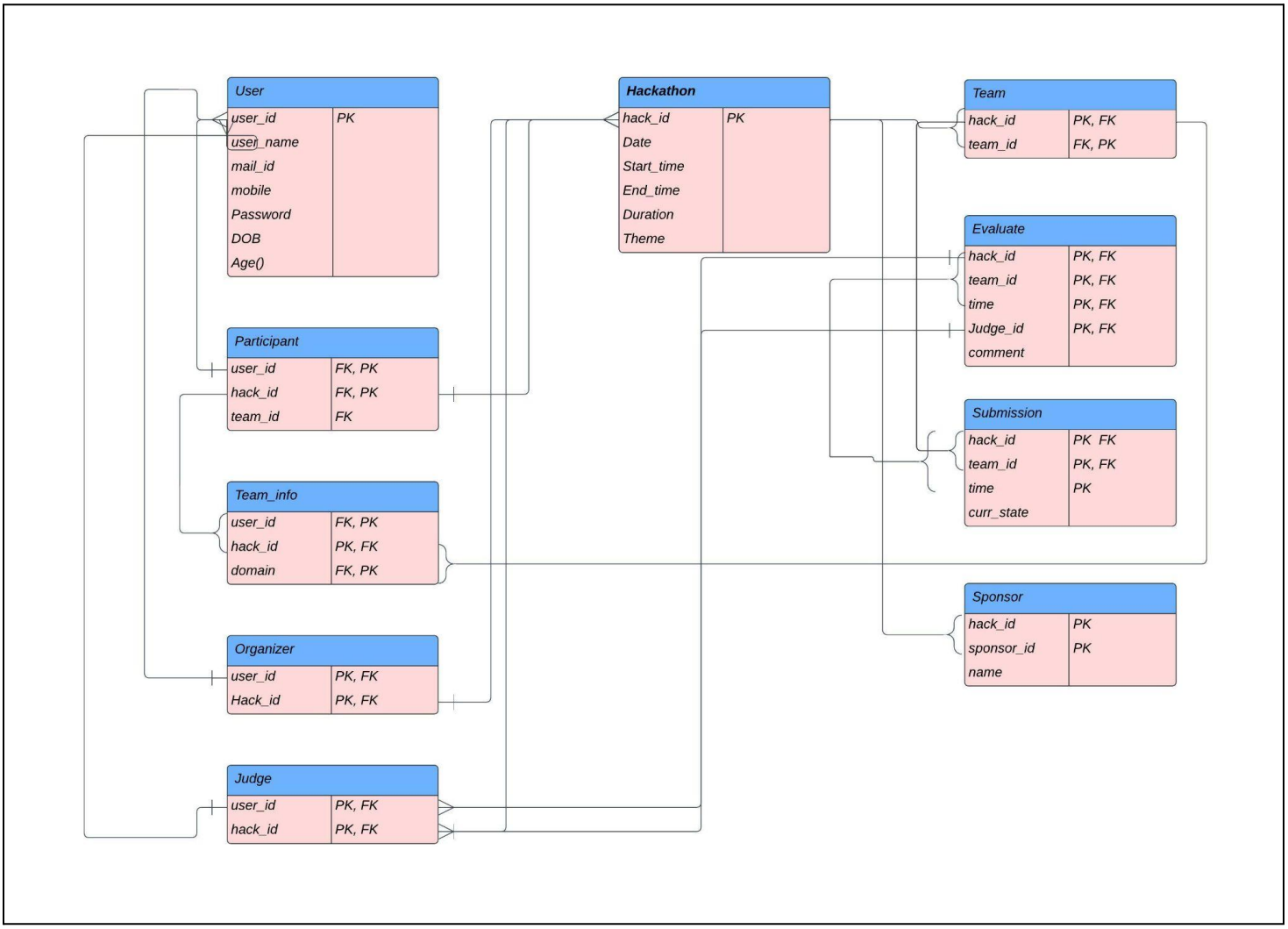
Functional Dependencies:
Hack_id ,user_id, Team_id -> Hack_id ,user_id, Team_id

Partial Key Dependency: None
Transitive Dependency: None
Redundancies: None

Anomalies:
Insert – None.
Update – Updating this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
Delete – Deleting from this table will create problems sometimes as its primary key is being accessed as a foreign key in other tables.
In this schema, every attribute is single-valued (scalar) making it already in 1NF.
There is no partial dependency here, so it is in 2NF as well.
2NF Redundancies: None
Since there is no transitive dependency, it is also in 3NF.
Thus, our final schema remains the same.

For a relation to be in BCNF,
a. It should be in the third normal form (3NF).
b. For any dependency A → B, A must be a super key.
Hence, this relation is in BCNF as well.

Final Relational model



❖ Final Schema:

- **User**(user_Id , Email_Id , name, Password , DOB , Age , mobile)

- **Organizer**(user_id , hack_id)
  - FK hack_id references to **Hackathon**
  - FK user_id references to **User**

- **Team**(Hack_id , Team_Id)
  - FK Hack_id references to **Hackathon**

- **Team_info**(user_id , Hack_id , Team_id)
  - FK user_id references to **User**
  - FK Hack_id references to **Hackathon**
  - FK Team_id references to **Team**

- **Participant**(User_Id, Hack_id , domain)
  - FK Hack_id references to **Hackathon**
  - FK user_id references to **User**

- **Submission**(Hack_id, team_id , Time , Curr_Status)
  - FK Hack_id references to **Hackathon**
  - FK team_id references to **Team**

- **Sponsor**(Hack_id , Sponsor_id , Name)
  - FK Hack_id references to **Hackathon**

- **Hackathon**(Hack_id , date , start_time , end_time , duration , theme)

- **Evaluate**(hack_id , team_id , judge_id , comment)
  - FK Hack_id references to **Hackathon**
  - FK team_id references to **Team**

- **Judge**(Hack_id , Judge_Id)
  - FK Hack_id references to **Hackathon**
  - FK Judge_id references to **User**

Modified DDL Script:

User:

```
-- Table: hm.User

-- DROP TABLE IF EXISTS hm."User";

CREATE TABLE IF NOT EXISTS hm."User"
(
    user_id character varying COLLATE pg_catalog."default" NOT NULL,
    email_id character varying COLLATE pg_catalog."default" NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    password character varying COLLATE pg_catalog."default" NOT NULL,
    dob date NOT NULL,
    age integer NOT NULL,
    CONSTRAINT "User_pkey" PRIMARY KEY (user_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."User"
    OWNER to postgres;
```

Hackathon:

```
-- Table: hm.Hackathon

-- DROP TABLE IF EXISTS hm."Hackathon";

CREATE TABLE IF NOT EXISTS hm."Hackathon"
(
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    date date NOT NULL,
    s_time time without time zone NOT NULL,
    e_time time without time zone NOT NULL,
    duration character varying COLLATE pg_catalog."default" NOT NULL,
    theme character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Hackathon_pkey" PRIMARY KEY (hack_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Hackathon"
    OWNER to postgres;
```

Participant:

```
-- Table: hm.Participant

-- DROP TABLE IF EXISTS hm."Participant";

CREATE TABLE IF NOT EXISTS hm."Participant"
(
    user_id character varying COLLATE pg_catalog."default" NOT NULL,
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    domain character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Participant_pkey" PRIMARY KEY (user_id, hack_id),
    CONSTRAINT fk_hack FOREIGN KEY (hack_id)
        REFERENCES hm."Hackathon" (hack_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user FOREIGN KEY (user_id)
        REFERENCES hm."User" (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Participant"
    OWNER to postgres;
```

Judge:

```
-- Table: hm.Judge

-- DROP TABLE IF EXISTS hm."Judge";

CREATE TABLE IF NOT EXISTS hm."Judge"
(
    user_id character varying COLLATE pg_catalog."default" NOT NULL,
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Judge_pkey" PRIMARY KEY (user_id, hack_id),
    CONSTRAINT fk_hack FOREIGN KEY (hack_id)
        REFERENCES hm."Hackathon" (hack_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user FOREIGN KEY (user_id)
        REFERENCES hm."User" (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Judge"
    OWNER to postgres;
```

Organizer:

```
-- Table: hm.Organizer

-- DROP TABLE IF EXISTS hm."Organizer";

CREATE TABLE IF NOT EXISTS hm."Organizer"
(
    user_id character varying COLLATE pg_catalog."default" NOT NULL,
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Organizer_pkey" PRIMARY KEY (user_id, hack_id),
    CONSTRAINT fk_hack FOREIGN KEY (hack_id)
        REFERENCES hm."Hackathon" (hack_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user FOREIGN KEY (user_id)
        REFERENCES hm."User" (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Organizer"
    OWNER to postgres;
```

Sponsor:

```
-- Table: hm.Sponsor

-- DROP TABLE IF EXISTS hm."Sponsor";

CREATE TABLE IF NOT EXISTS hm."Sponsor"
(
    sponsor_id character varying COLLATE pg_catalog."default" NOT NULL,
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Sponser_pkey" PRIMARY KEY (sponsor_id, hack_id),
    CONSTRAINT fk_hack FOREIGN KEY (hack_id)
        REFERENCES hm."Hackathon" (hack_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Sponsor"
    OWNER to postgres;
```

Team:

```
-- Table: hm_db.Team

-- DROP TABLE hm_db."Team";

CREATE TABLE hm_db."Team"
(
    "Curr_status" character varying COLLATE pg_catalog."default" NOT NULL,
    "Leader" character varying COLLATE pg_catalog."default" NOT NULL,
    "Team_id" character varying COLLATE pg_catalog."default" NOT NULL,
    "Hack_id" character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Team_pkey" PRIMARY KEY ("Team_id", "Hack_id"),
    CONSTRAINT fk_team1 FOREIGN KEY ("Hack_id")
        REFERENCES hm_db."Hackathon" ("Hack_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT fk_team2 FOREIGN KEY ("Leader")
        REFERENCES hm_db."Participant" ("User_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE hm_db."Team"
    OWNER to postgres;
```

Submission:

```
-- Table: hm.Submission

-- DROP TABLE IF EXISTS hm."Submission";

CREATE TABLE IF NOT EXISTS hm."Submission"
(
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    team_id character varying COLLATE pg_catalog."default" NOT NULL,
    "time" time without time zone NOT NULL,
    curr_state character varying COLLATE pg_catalog."default",
    CONSTRAINT "Submission_pkey" PRIMARY KEY (hack_id, team_id, "time"),
    CONSTRAINT fk_team_hack FOREIGN KEY (hack_id, team_id)
        REFERENCES hm."Team" (hack_id, team_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Submission"
    OWNER to postgres;

-- Trigger: curr
-- DROP TRIGGER IF EXISTS curr ON hm."Submission";
CREATE TRIGGER curr
    AFTER INSERT
    ON hm."Submission"
    FOR EACH ROW
    EXECUTE FUNCTION hm.func_state();

-- Trigger: present
-- DROP TRIGGER IF EXISTS present ON hm."Submission";

CREATE TRIGGER present
    AFTER INSERT
    ON hm."Submission"
    FOR EACH ROW
    EXECUTE FUNCTION hm.func_state();
```

Team_info:

```
-- Table: hm.Team_info

-- DROP TABLE IF EXISTS hm."Team_info";

CREATE TABLE IF NOT EXISTS hm."Team_info"
(
    user_id character varying COLLATE pg_catalog."default" NOT NULL,
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    team_id character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Team_info_pkey" PRIMARY KEY (user_id, hack_id,
team_id),
    CONSTRAINT fk_team_hack FOREIGN KEY (hack_id, team_id)
        REFERENCES hm."Team" (hack_id, team_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user_hack FOREIGN KEY (hack_id, user_id)
        REFERENCES hm."Participant" (hack_id, user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Team_info"
    OWNER to postgres;
```

Evaluate:

```
-- Table: hm.Evaluate

-- DROP TABLE IF EXISTS hm."Evaluate";

CREATE TABLE IF NOT EXISTS hm."Evaluate"
(
    hack_id character varying COLLATE pg_catalog."default" NOT NULL,
    team_id character varying COLLATE pg_catalog."default" NOT NULL,
    judge_id character varying COLLATE pg_catalog."default" NOT NULL,
    "time" time without time zone NOT NULL,
    comment character varying COLLATE pg_catalog."default",
    CONSTRAINT "Evaluate_pkey" PRIMARY KEY (hack_id, team_id, judge_id,
"time"),
    CONSTRAINT fk_judge_hack FOREIGN KEY (hack_id, judge_id)
        REFERENCES hm."Judge" (hack_id, user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_team_hack_time FOREIGN KEY ("time", team_id, hack_id)
        REFERENCES hm."Submission" ("time", team_id, hack_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS hm."Evaluate"
    OWNER to postgres;
```

Data generation

User Table:

Query | Query History

```
1  select * from hm."User"
2
```

Data output | Messages | Notifications

	user_id [PK] character varying	email_id character varying	name character varying	password character varying	dob date	age integer
1	1	iluto0@de.vu	Immanuel	Zn1rcR4ymKec	2001-08-25	10
2	2	towthwaite1@blo...	Terencio	UymLJLb	1999-04-26	32
3	3	emarston2@taob...	Edmon	KODsazTIIZ9u	1999-09-04	18
4	4	dhandley3@tech...	Danica	KLrQCcaooj	1995-09-16	11
5	5	kkingshott4@geo...	Korie	9DJVmoEKRY	1986-12-05	35
6	6	cmanoch5@tam...	Chancey	nbjmIzpbP2	2006-09-13	27
7	7	ojoderli6@paypal...	Osbert	Wsnkvd	1978-06-15	50
8	8	lmaciaszek7@ca...	Lois	v5B5TvihHz	1975-04-27	30
9	9	adaly8@japanpo...	Alec	tWiamzn7eh21	1990-11-11	32

Evaluate Table:

```
1  select *
2  from hm."Evaluate"
```

Data output | Messages | Notifications

	hack_id [PK] character varying	team_id [PK] character varying	judge_id [PK] character varying	time [PK] time without time zone	comment character varying
187	50	3	193	17:16:58	FALSE
188	50	12	193	05:19:00	TRUE
189	50	15	193	17:01:47	TRUE
190	50	31	193	16:05:40	TRUE
191	50	33	193	18:17:13	TRUE

Hackathon Table:

Query Query History

```
1  select *
2  from hm."Hackathon"
```

Data output Messages Notifications

	hack_id [PK] character varying	date date	s_time time without time zone	e_time time without time zone	duration character varying	theme character varying
47	47	2022-08-18	16:56:00	21:40:00	23:01	Brokerage
48	48	2022-09-15	12:43:00	17:19:00	19:11	NCP
49	49	2022-10-02	16:44:00	22:49:00	20:18	Formation Evalua...
50	50	2022-08-08	00:52:00	09:29:00	05:28	Slackware
51	2000	2020-05-13	[null]	[null]	[null]	[null]

Judge Table:

Query Query History

```
1  select *
2  from hm."Judge"
```

Data output Messages Notifications

	user_id [PK] character varying	hack_id [PK] character varying
46	199	46
47	197	47
48	199	48
49	197	49
50	193	50

Total rows: 50 of 50 Query complete 00:00:00.109

Organizer Table:

Query Query History

```
1  select *
2  from hm."Organizer"
```

Data output Messages Notifications

	user_id [PK] character varying	hack_id [PK] character varying
46	186	46
47	181	47
48	188	48
49	183	49
50	185	50

Total rows: 50 of 50 Query complete 00:00:00.072

Participant Table:

Query Query History

```
1  select *
2  from hm."Participant"
```

Data output Messages Notifications

	user_id [PK] character varying	hack_id [PK] character varying	domain character varying
196	172	23	Human Resources
197	173	10	Legal
198	174	47	Accounting
199	175	18	Human Resources
200	178	28	Engineering

Total rows: 200 of 200 Query complete 00:00:00.067

Sponsor Table:

Query Query History

```
1  select *
2  from hm."Sponsor"
```

Data output Messages Notifications

	sponsor_id [PK] character varying	hack_id [PK] character varying	name character varying
46	18	50	Ronnie
47	19	48	Jessy
48	20	35	Ivette
49	20	42	Ivette
50	20	49	Ivette

Total rows: 50 of 50 Query complete 00:00:00.074

Submission Table:

Query Query History

```
1  select *
2  from hm."Submission"
```

Data output Messages Notifications

	hack_id [PK] character varying	team_id [PK] character varying	time [PK] time without time zone	curr_state character varying
187	50	3	17:16:58	FALSE
188	50	12	05:19:00	TRUE
189	50	15	17:01:47	TRUE
190	50	31	16:05:40	TRUE
191	50	33	18:17:13	TRUE

Total rows: 191 of 191 Query complete 00:00:00.089

Team Table:

Query Query History

```
1  select *
2  from hm."Team"
```

Data output Messages Notifications

	hack_id [PK] character varying	team_id [PK] character varying
186	50	3
187	50	12
188	50	15
189	50	31
190	50	33

Total rows: 190 of 190 Query complete 00:00:00.088

Team_info Table:

Query Query History

```
1  select *
2  from hm."Team_info"
```

Data output Messages Notifications

	user_id [PK] character varying	hack_id [PK] character varying	team_id [PK] character varying
196	172	23	34
197	173	10	55
198	174	47	24
199	175	18	14
200	178	28	10

Total rows: 200 of 200 Query complete 00:00:00.122

Queries

1) Find details of participants whose team id is 5 of every Hackathon.

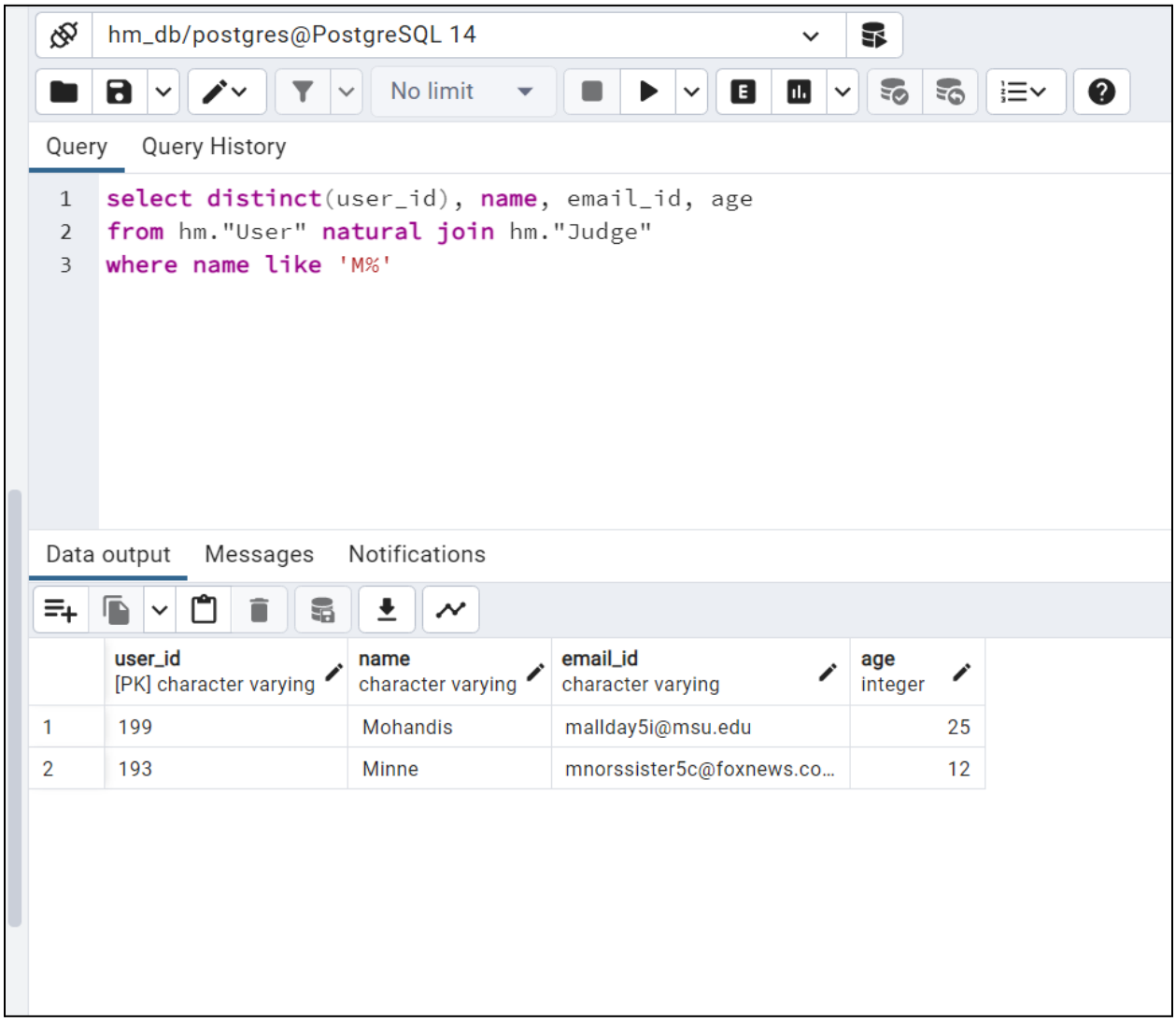
Query | Query History

```
1  select user_id, name
2  from hm."Participant" natural join
3      hm."Team_info" natural join
4      hm."Team" natural join
5      hm."User"
6  where team_id='5'
7
8
9
```

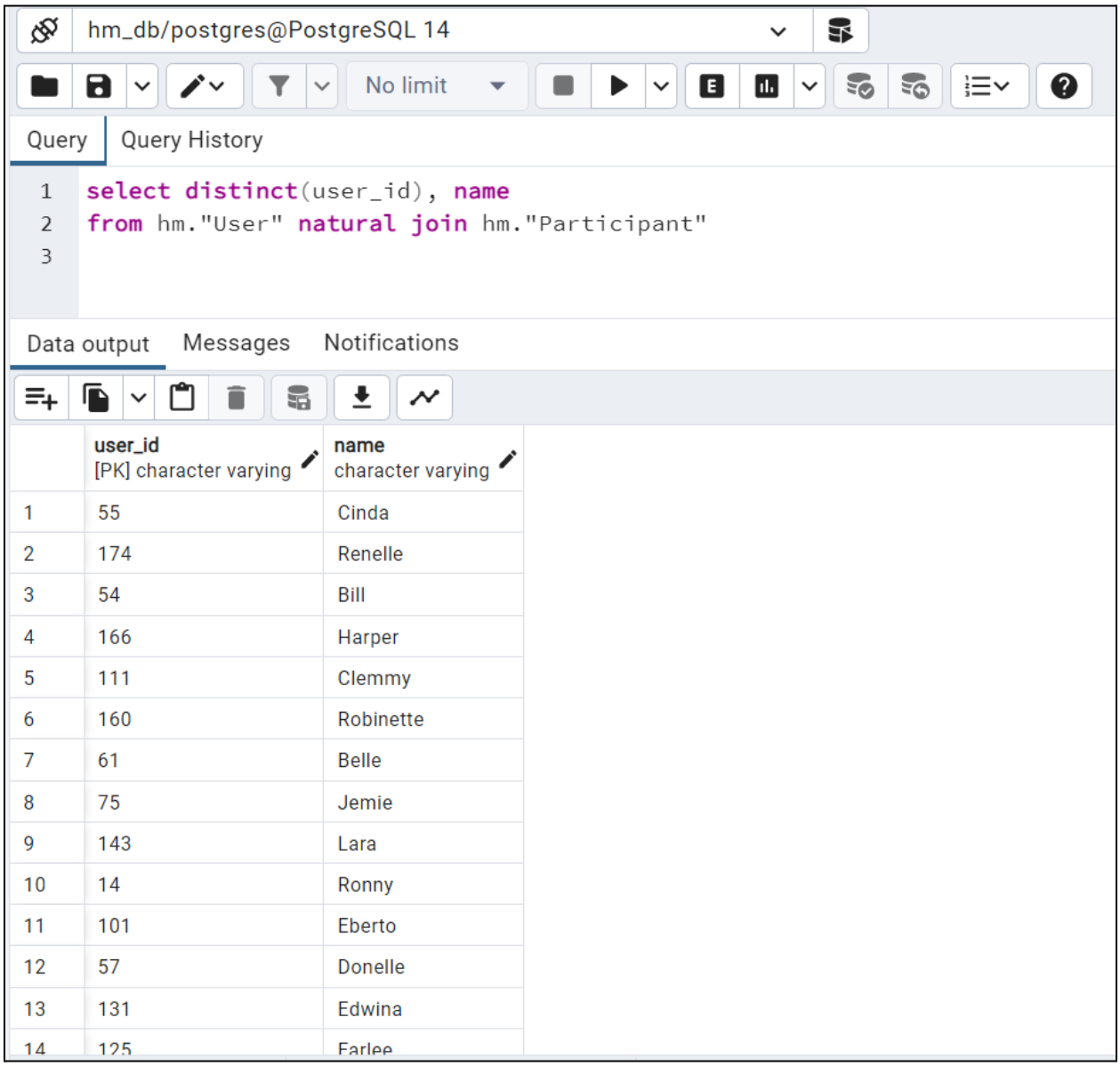
Data output | Messages | Notifications

	user_id character varying	name character varying
1	32	Petronia
2	47	Cletis
3	57	Donelle
4	58	Eleen

2) Find details of Judges whose name starts with letter M.

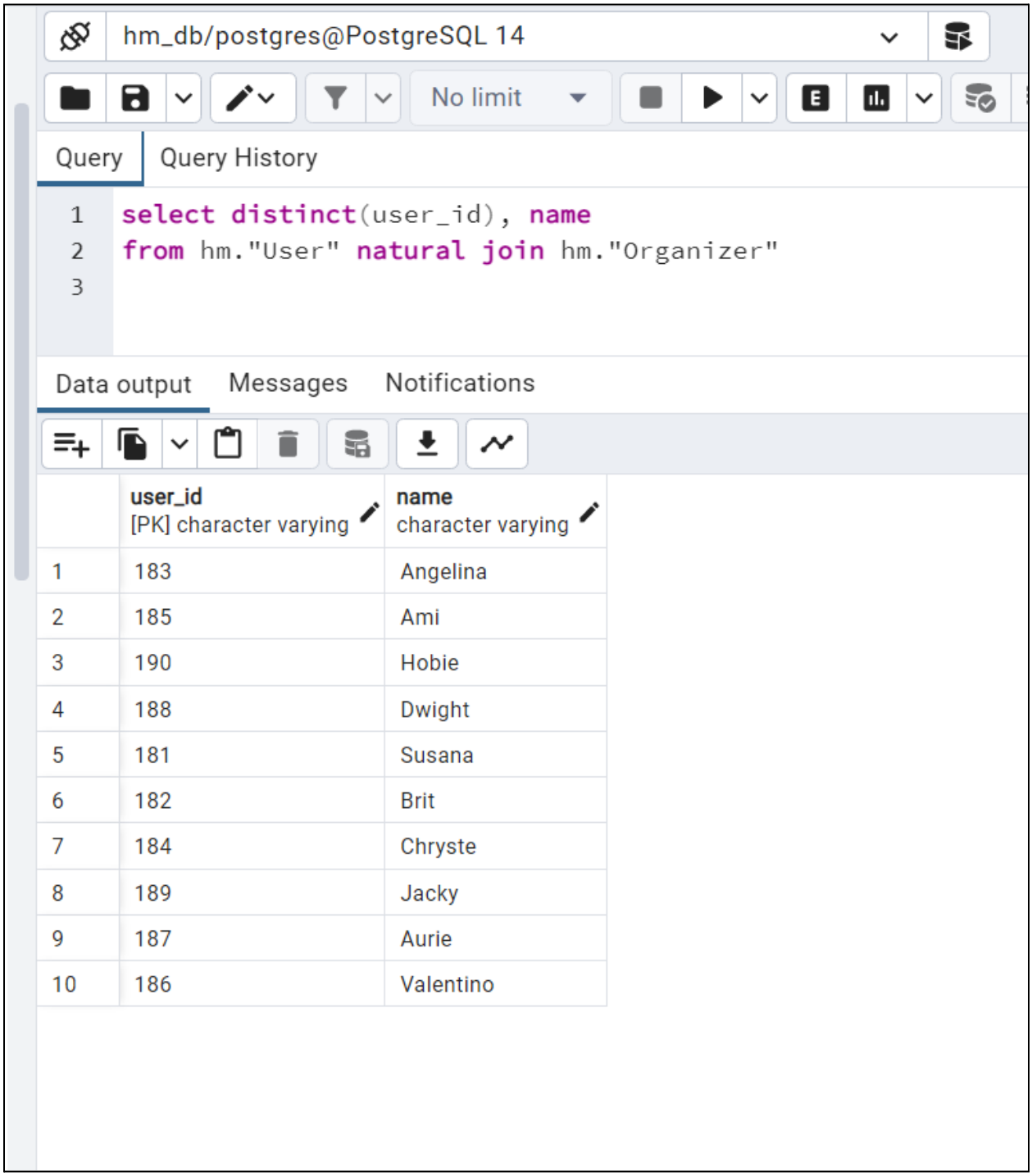


3) Show name and user_id of the users who have participated in Hackathon.]



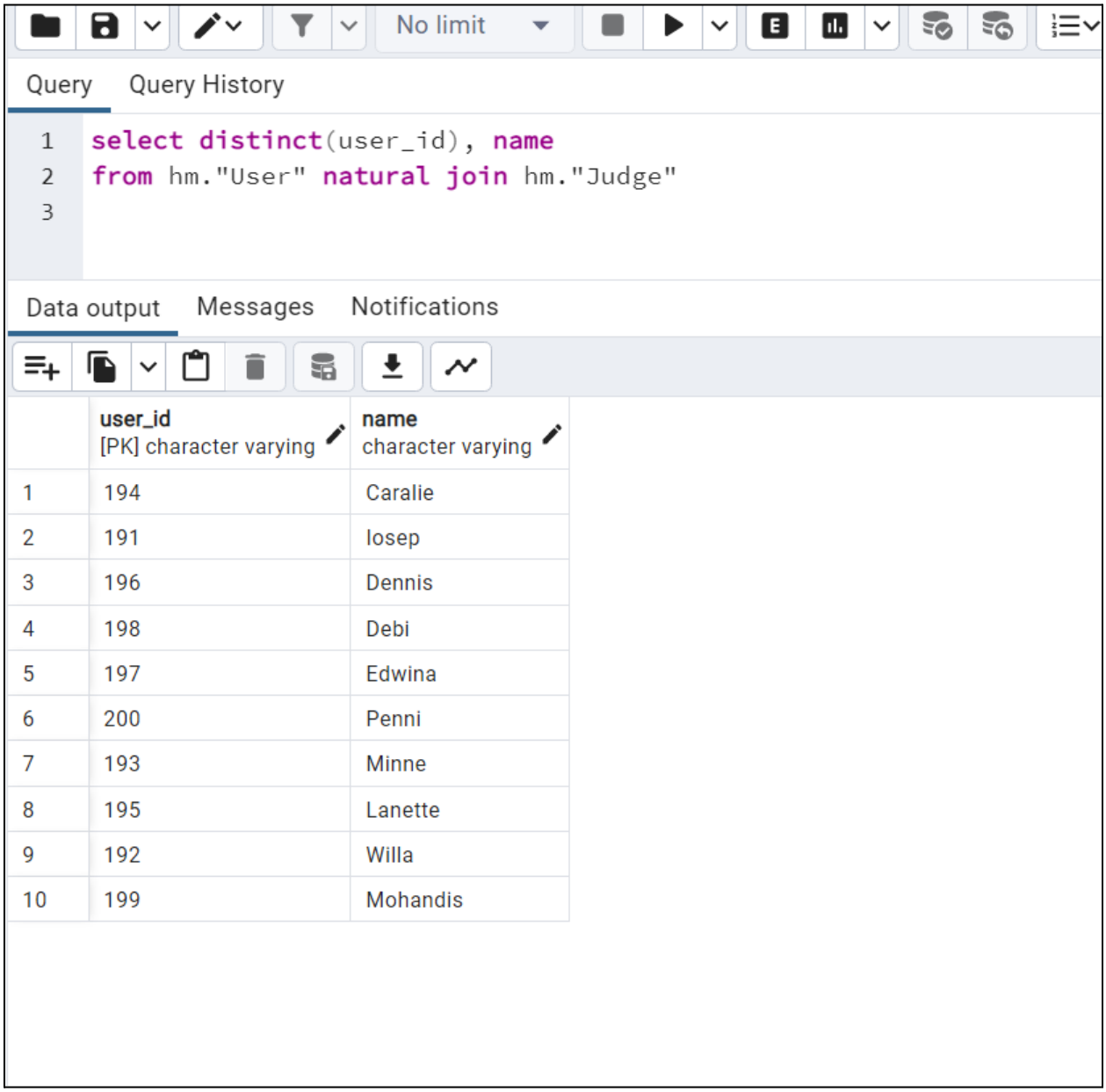
	user_id [PK] character varying	name character varying
1	55	Cinda
2	174	Renelle
3	54	Bill
4	166	Harper
5	111	Clemmy
6	160	Robinette
7	61	Belle
8	75	Jemie
9	143	Lara
10	14	Ronny
11	101	Eberto
12	57	Donelle
13	131	Edwina
14	125	Farlee

4) Show name and user_id of the users who have Organized the Hackathon.



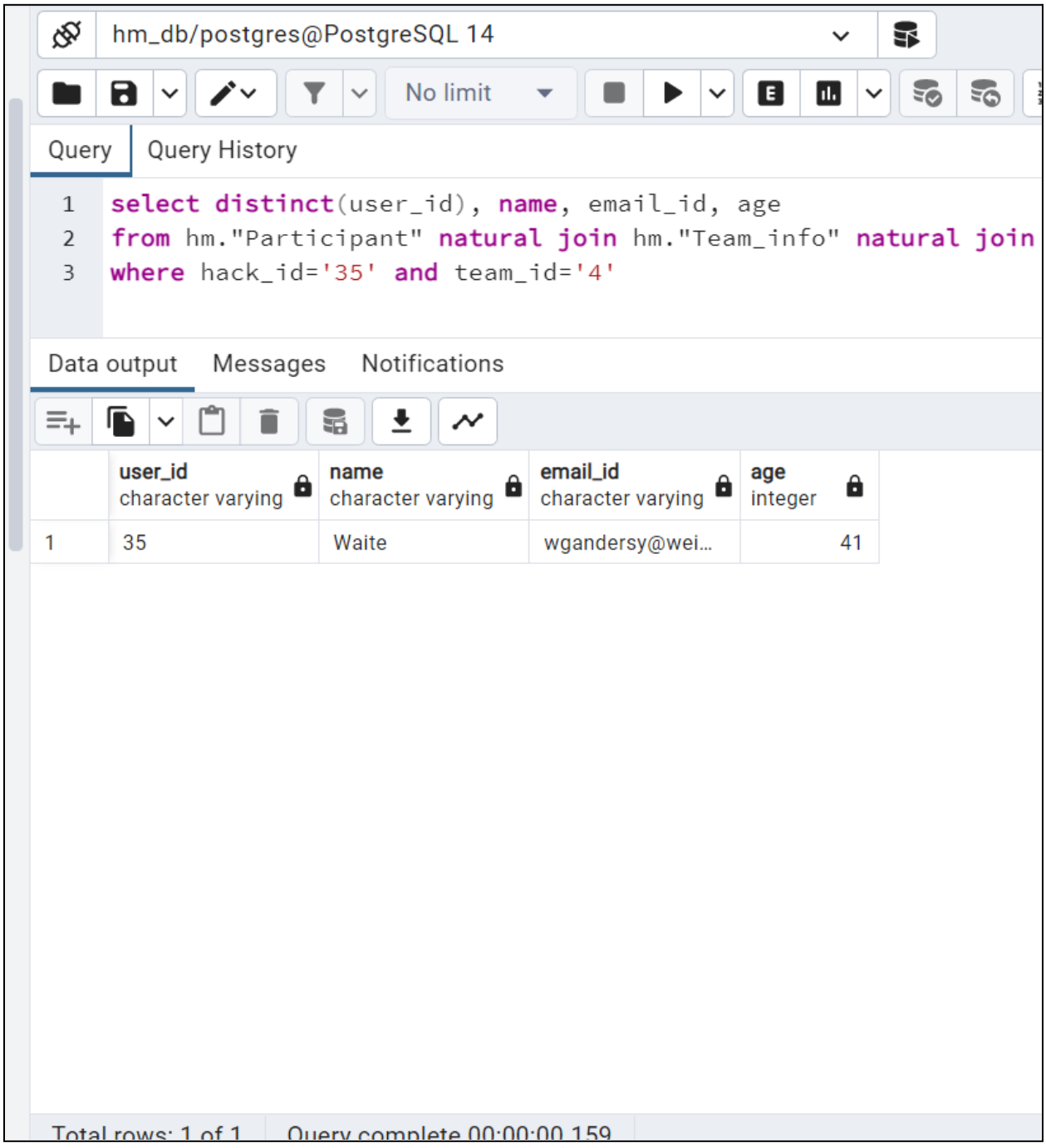
	user_id [PK] character varying	name character varying
1	183	Angelina
2	185	Ami
3	190	Hobie
4	188	Dwight
5	181	Susana
6	182	Brit
7	184	Chryste
8	189	Jacky
9	187	Aurie
10	186	Valentino

5) Show name and user_id of the users who have Evaluated the submissions in Hackathon.

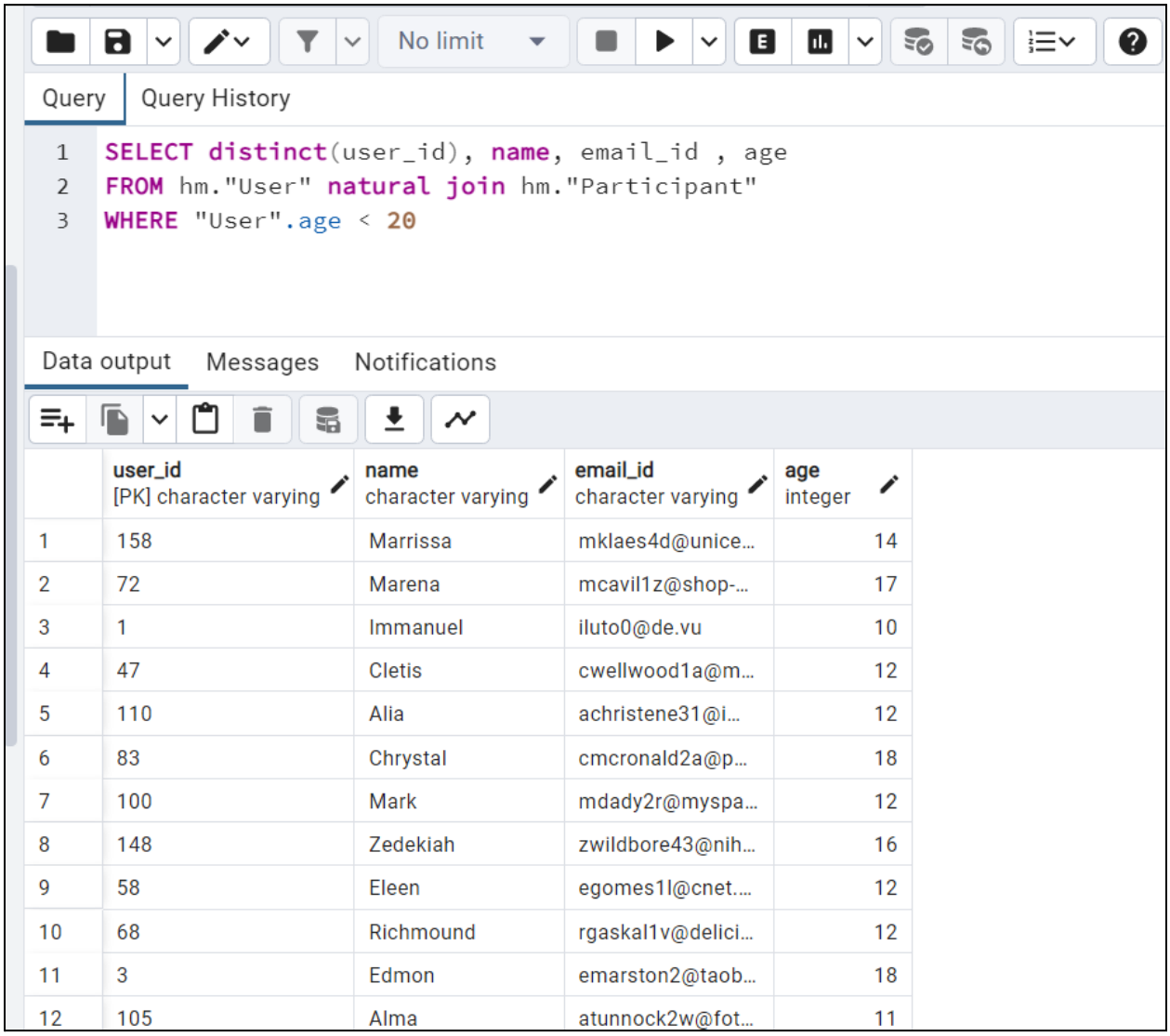


	user_id [PK] character varying	name character varying
1	194	Caralie
2	191	Iosep
3	196	Dennis
4	198	Debi
5	197	Edwina
6	200	Penni
7	193	Minne
8	195	Lanette
9	192	Willa
10	199	Mohandis

6) Show the details of Participants in Hackathon with hack_id 35 and team_id 4.



7) Show the details of participants younger than 20.



	user_id [PK] character varying	name character varying	email_id character varying	age integer
1	158	Marrissa	mklaes4d@unice...	14
2	72	Marena	mcavil1z@shop-...	17
3	1	Immanuel	iluto0@de.vu	10
4	47	Cletis	cwellwood1a@m...	12
5	110	Alia	achristene31@i...	12
6	83	Chrystal	cmcronald2a@p...	18
7	100	Mark	mdady2r@myspa...	12
8	148	Zedekiah	zwildbore43@nih...	16
9	58	Eleen	egomes1l@cnet....	12
10	68	Richmound	rgaskal1v@delici...	12
11	3	Edmon	emarston2@taob...	18
12	105	Alma	atunnock2w@fot...	11

8) Show all hackathons sponsored by sponsor id = 5.

Query Query History

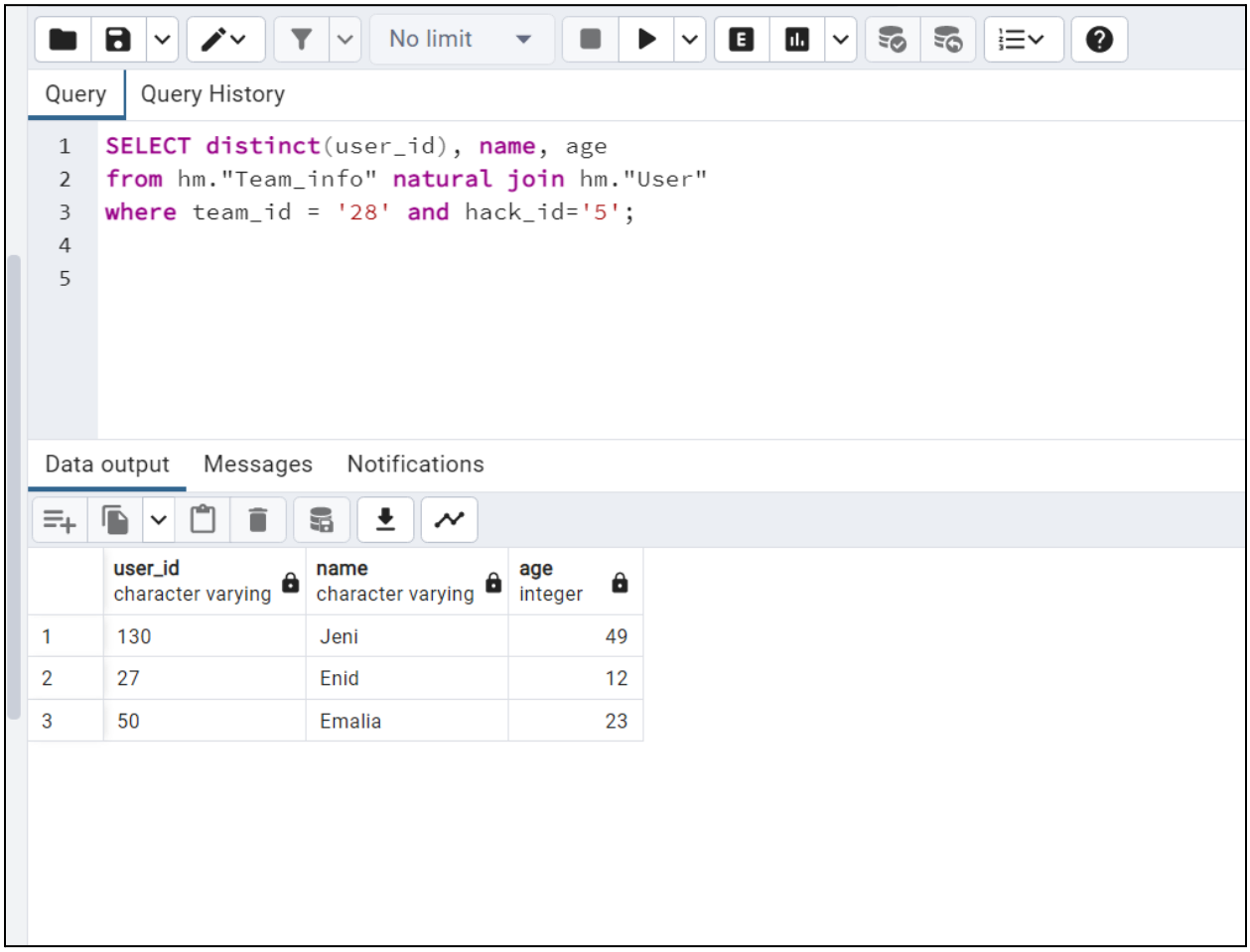
```
1  SELECT hm."Sponsor".sponsor_id , hm."Sponsor".hack_id , hm."Sponsor".name
2  FROM hm."Hackathon" natural join hm."Sponsor"
3  where hm."Sponsor".sponsor_id = '5'
```

Data output Messages Notifications

	sponsor_id [PK] character varying	hack_id [PK] character varying	name character varying
1	5	13	Carma
2	5	25	Carma
3	5	43	Carma

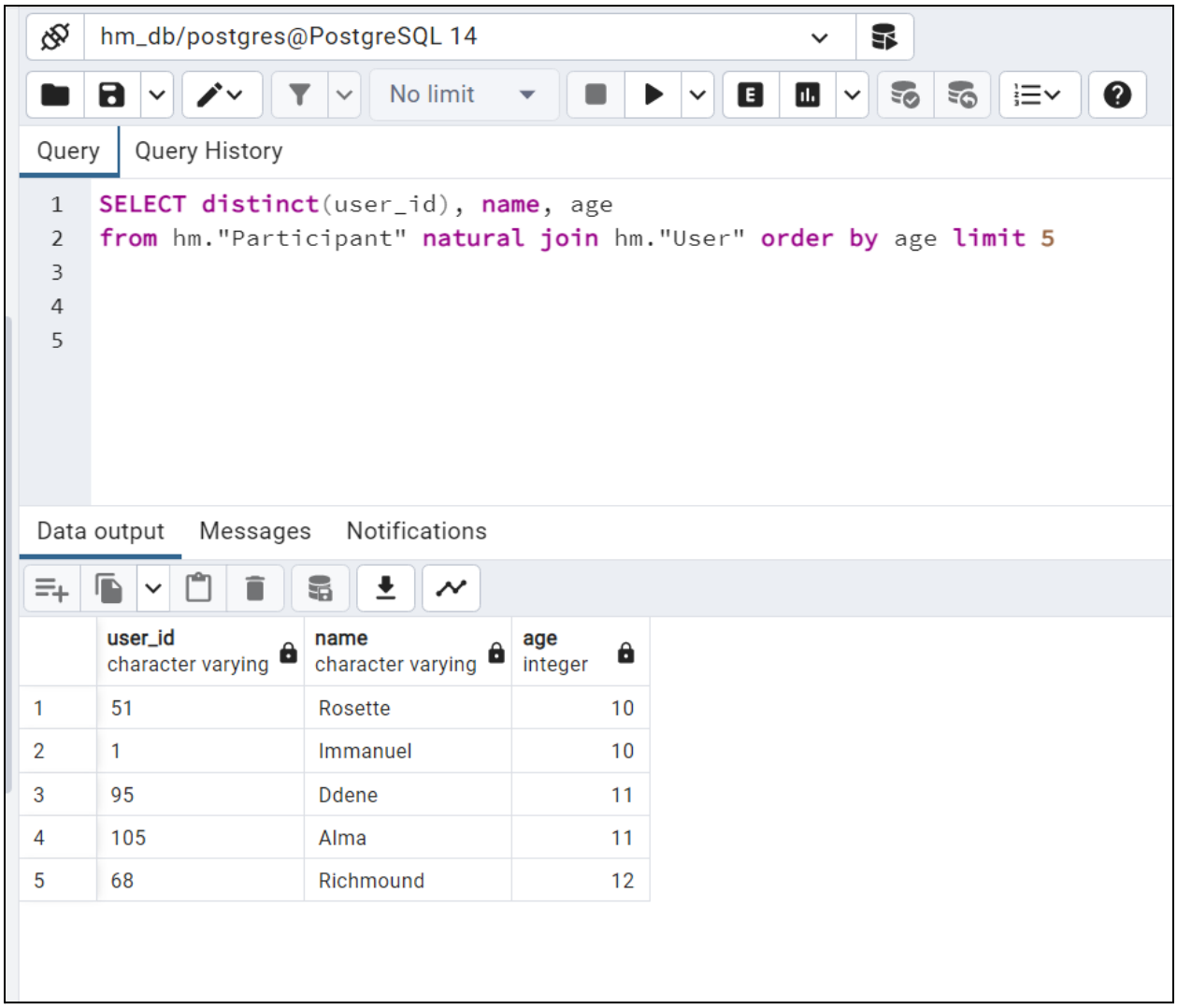
Total rows: 3 of 3 Query complete 00:00:01.658

9) Show details of participants of hack_id 5 with team_id 28.



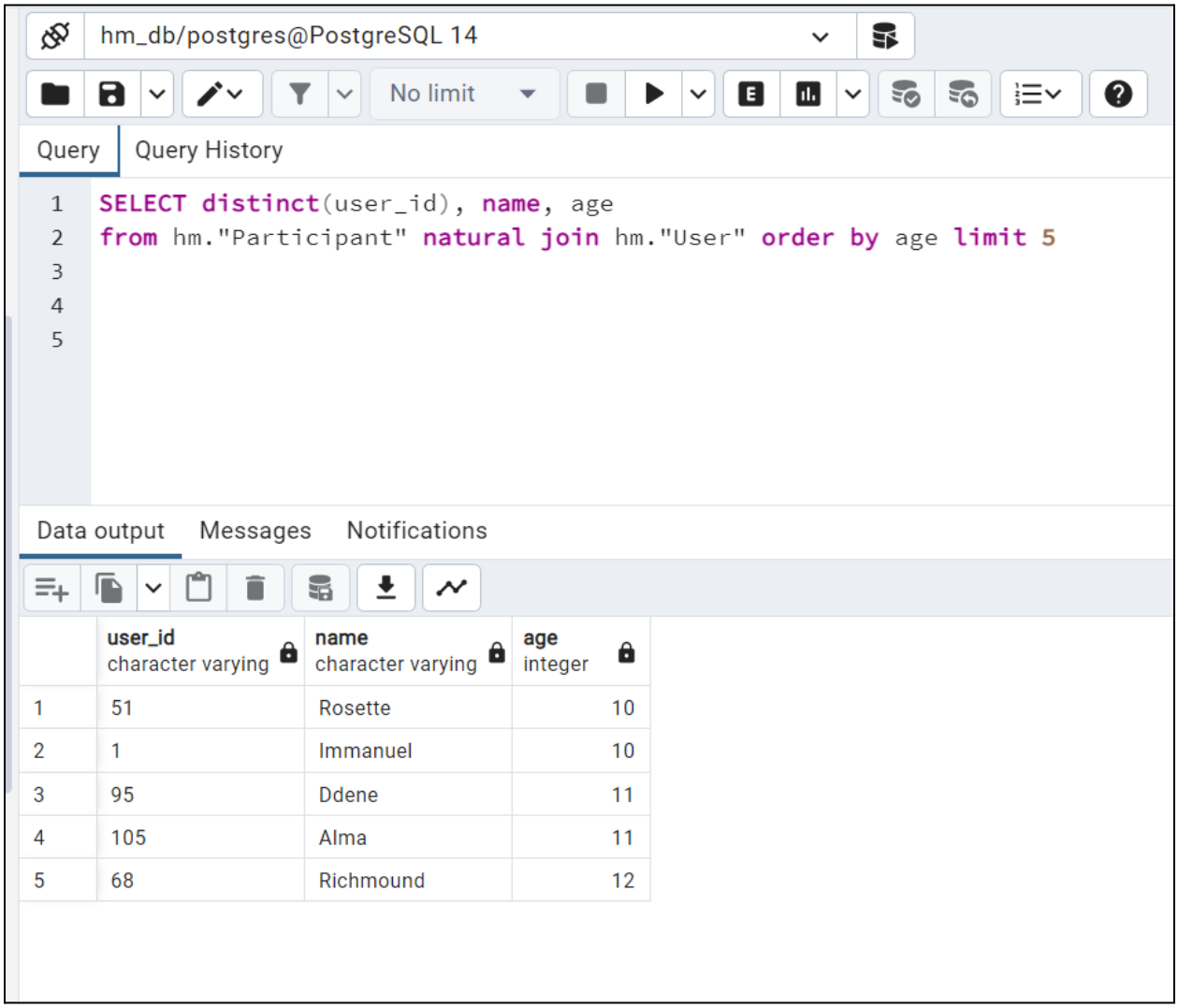
	user_id character varying	name character varying	age integer
1	130	Jeni	49
2	27	Enid	12
3	50	Emalia	23

10) Show details of the first 5 youngest participants.



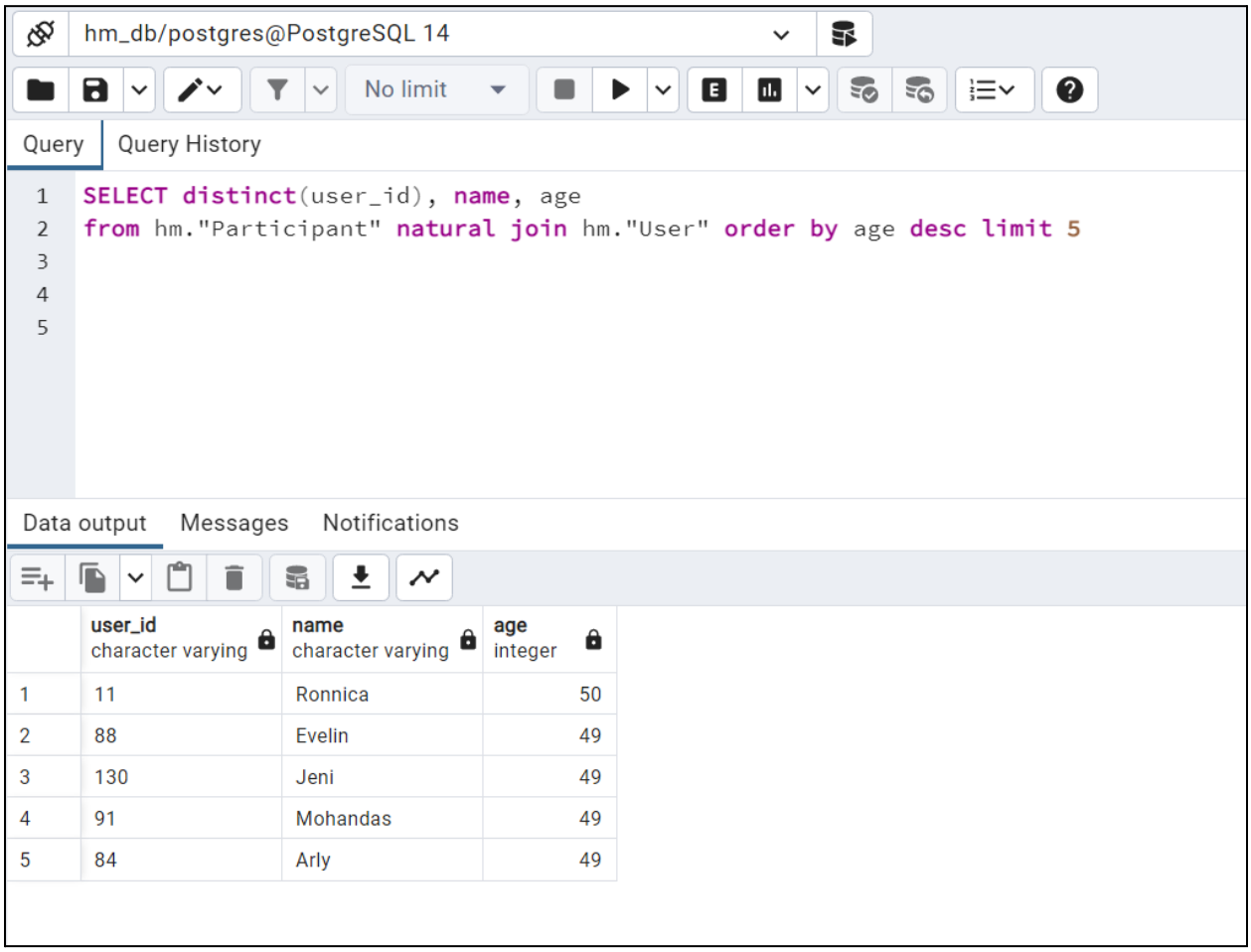
	user_id character varying	name character varying	age integer
1	51	Rosette	10
2	1	Immanuel	10
3	95	Ddene	11
4	105	Alma	11
5	68	Richmound	12

11) Show details of the first 5 youngest participants.



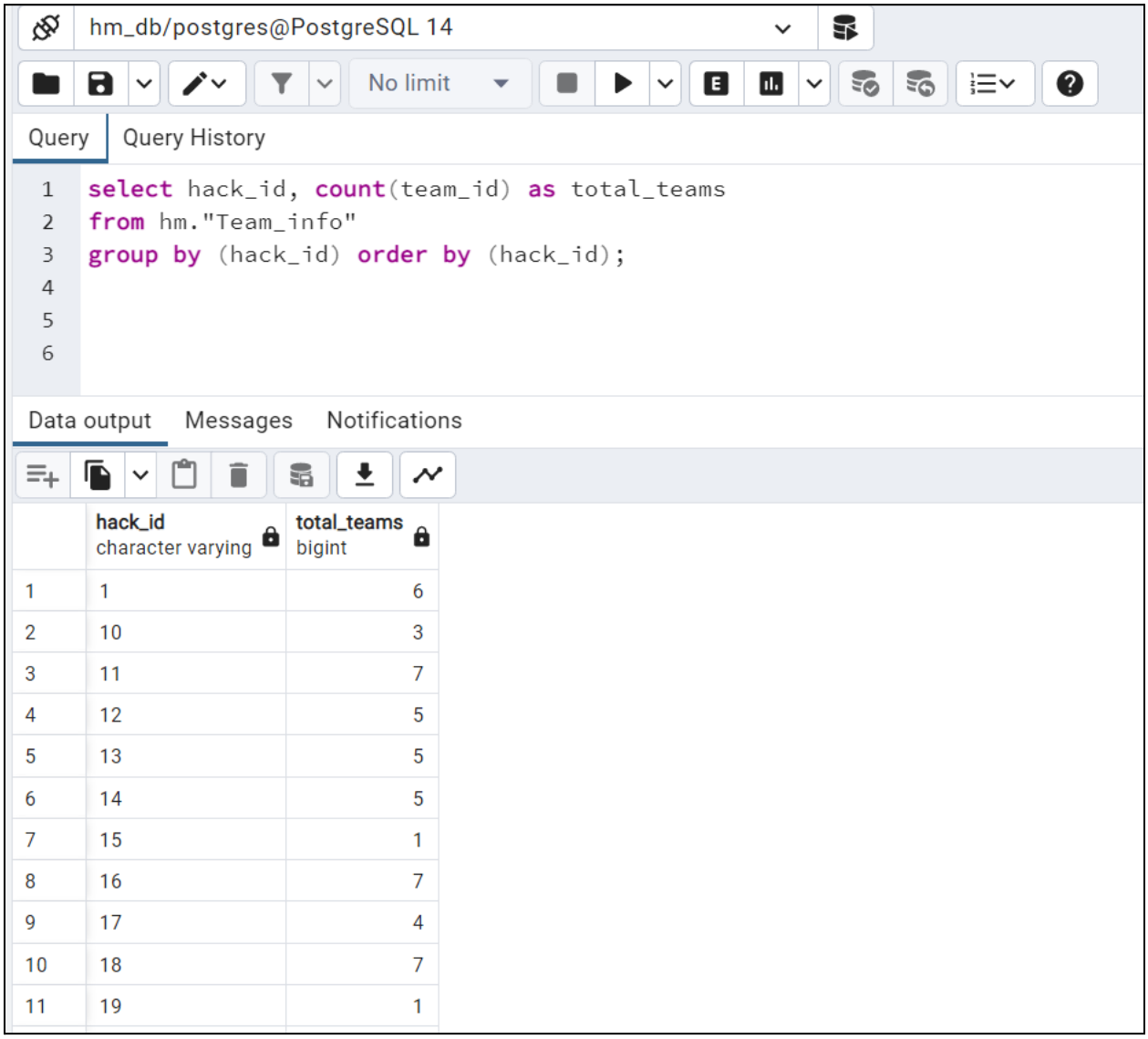
	user_id character varying	name character varying	age integer
1	51	Rosette	10
2	1	Immanuel	10
3	95	Ddene	11
4	105	Alma	11
5	68	Richmound	12

12) Show details of the first 5 oldest participants.



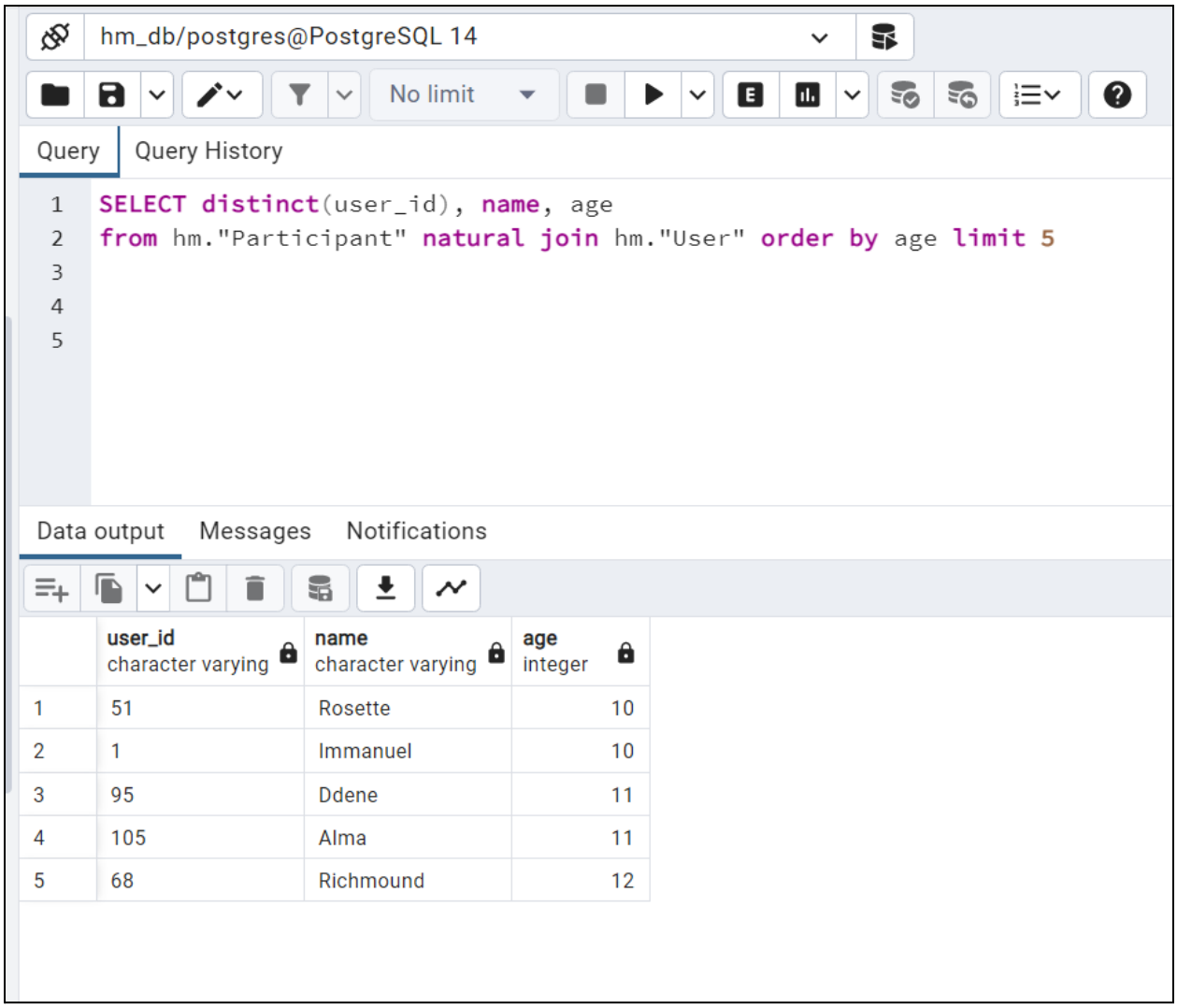
	user_id character varying	name character varying	age integer
1	11	Ronnica	50
2	88	Evelin	49
3	130	Jeni	49
4	91	Mohandas	49
5	84	Arly	49

13) Show the number of the teams participating in each hackathon.



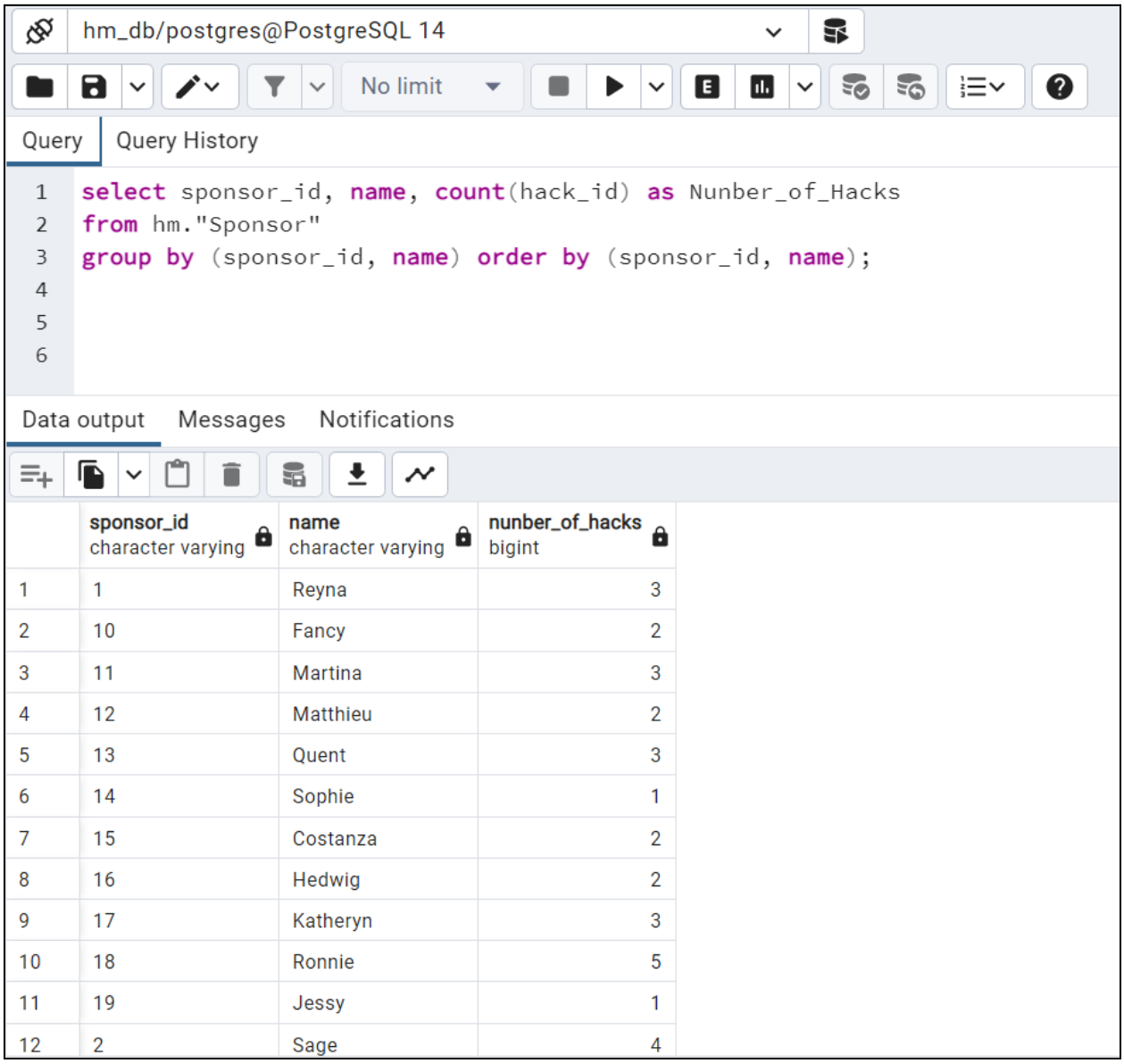
	hack_id character varying	total_teams bigint
1	1	6
2	10	3
3	11	7
4	12	5
5	13	5
6	14	5
7	15	1
8	16	7
9	17	4
10	18	7
11	19	1

14) Show details of the first 5 youngest participants.



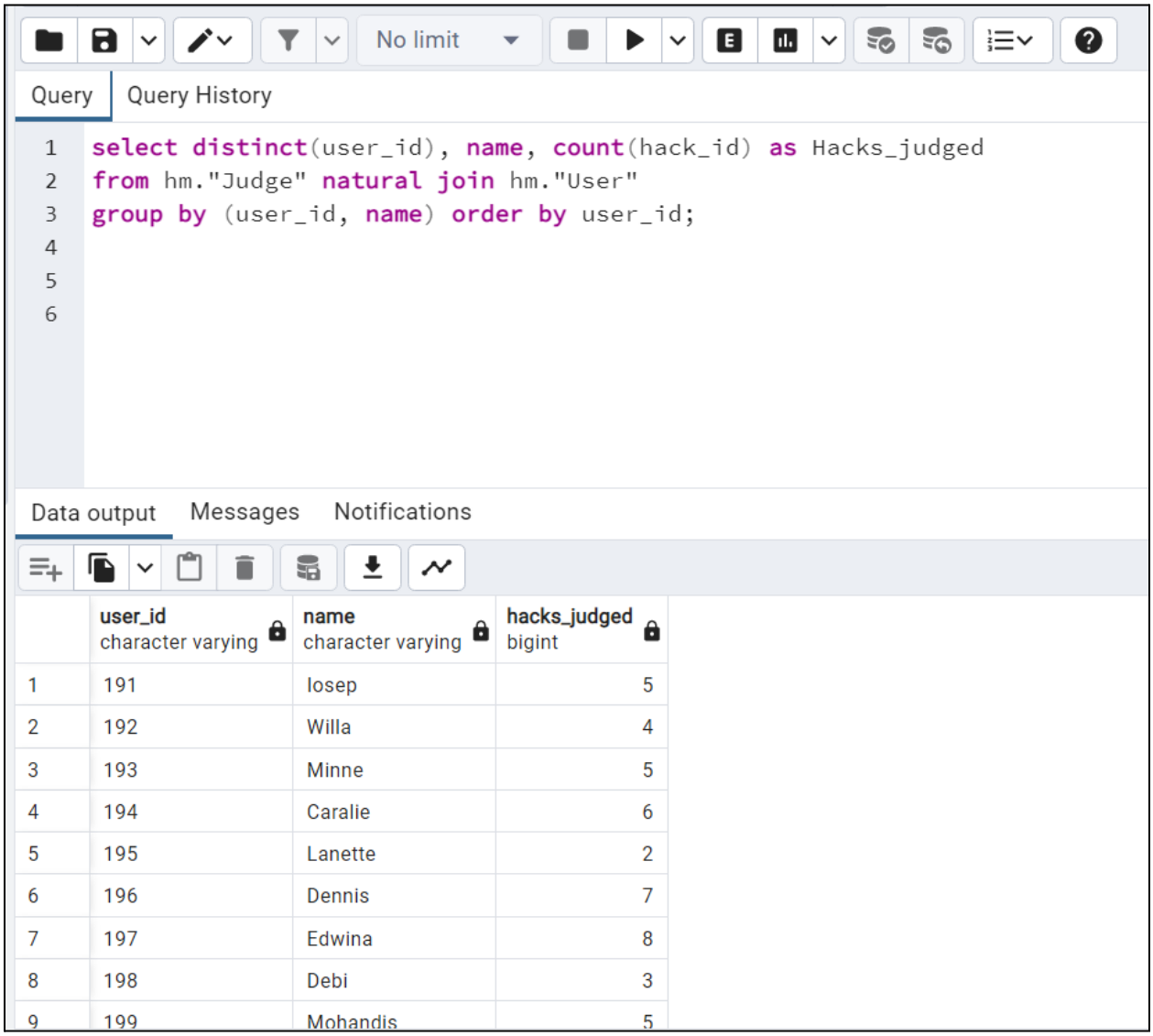
	user_id character varying	name character varying	age integer
1	51	Rosette	10
2	1	Immanuel	10
3	95	Ddene	11
4	105	Alma	11
5	68	Richmound	12

15) Show number of hackathons sponsored by each sponsor with their ids and name.



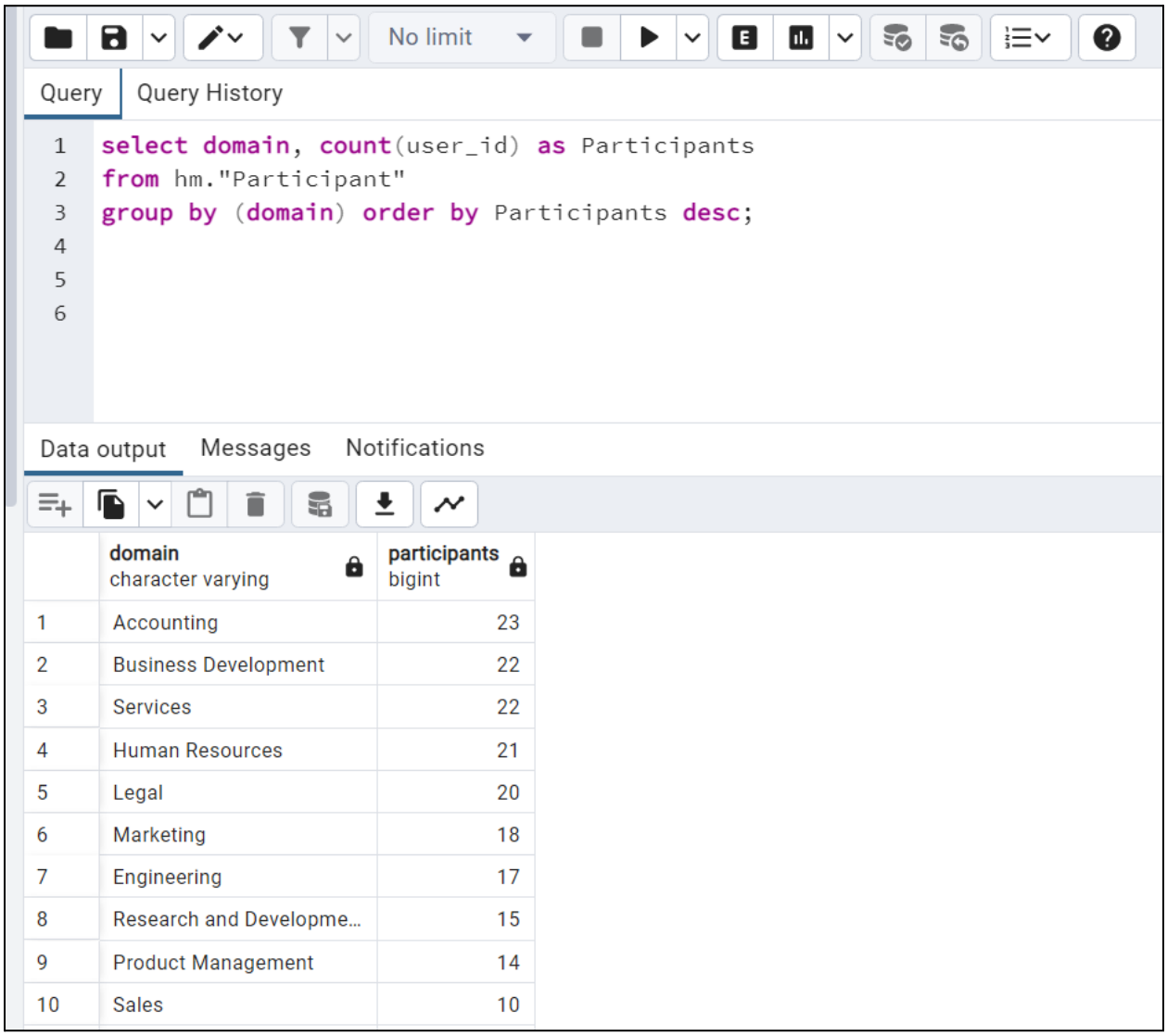
	sponsor_id character varying	name character varying	nunber_of_hacks bigint
1	1	Reyna	3
2	10	Fancy	2
3	11	Martina	3
4	12	Matthieu	2
5	13	Quent	3
6	14	Sophie	1
7	15	Costanza	2
8	16	Hedwig	2
9	17	Katheryn	3
10	18	Ronnie	5
11	19	Jessy	1
12	2	Sage	4

16) Show user_id and name of the users who have evaluated hackathons along with the number of hackathons they have evaluated.



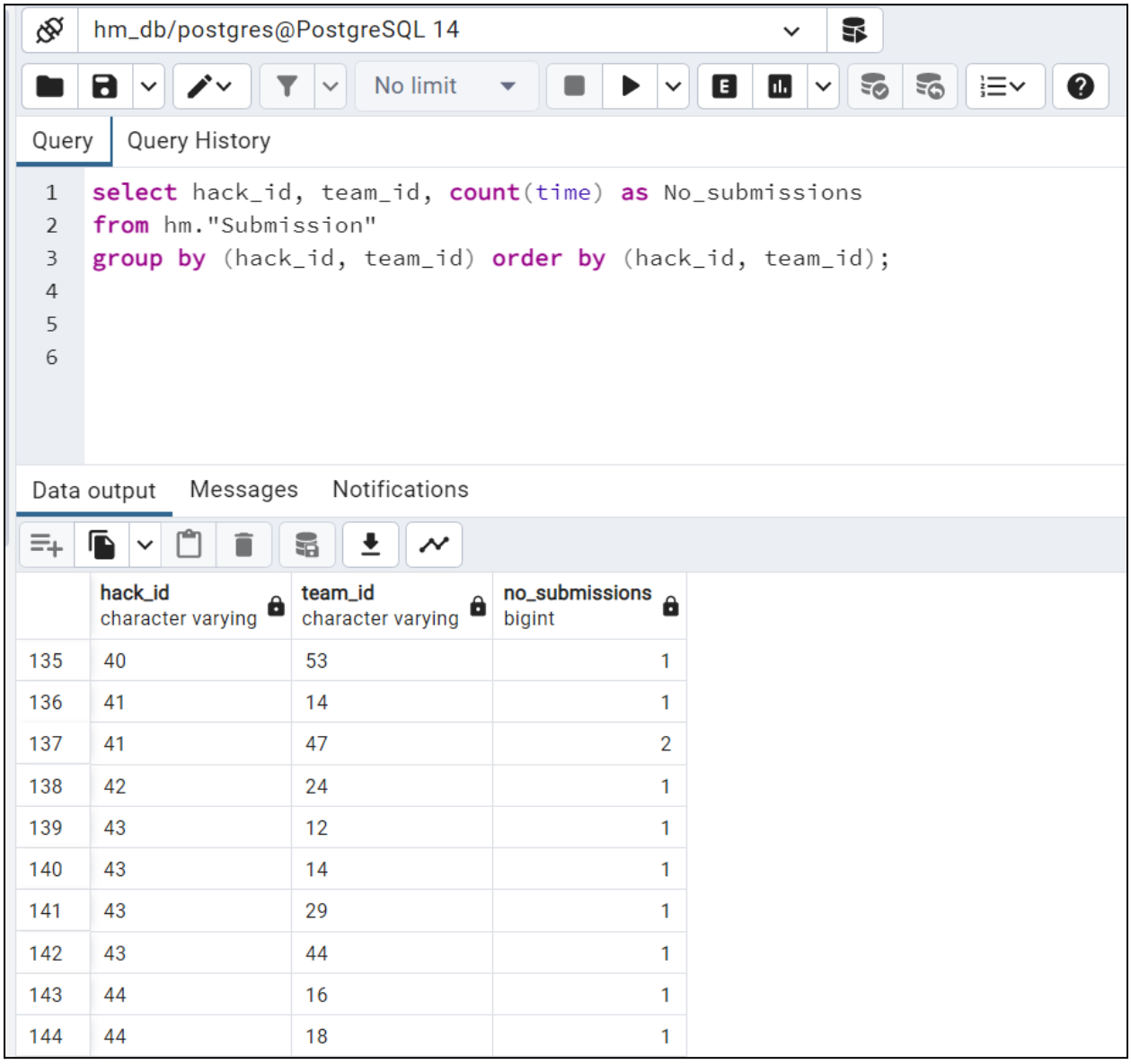
	user_id character varying	name character varying	hacks_judged bigint
1	191	Iosep	5
2	192	Willa	4
3	193	Minne	5
4	194	Caralie	6
5	195	Lanette	2
6	196	Dennis	7
7	197	Edwina	8
8	198	Debi	3
9	199	Mohandis	5

17) Show the number of participants corresponding to different domains. The domain with a higher number of participants should be first.



	domain character varying	participants bigint
1	Accounting	23
2	Business Development	22
3	Services	22
4	Human Resources	21
5	Legal	20
6	Marketing	18
7	Engineering	17
8	Research and Developme...	15
9	Product Management	14
10	Sales	10

18) Show numbers of submissions made by each team corresponding to each hackathon.



	hack_id character varying	team_id character varying	no_submissions bigint
135	40	53	1
136	41	14	1
137	41	47	2
138	42	24	1
139	43	12	1
140	43	14	1
141	43	29	1
142	43	44	1
143	44	16	1
144	44	18	1

19) Trigger:

Query Query History

```
1  Trigger
2  set search_path to hack
3
4  create or replace trigger check_id
5  before insert
6  on "hack".User
7  for each row execute function check_new_id();
8
9  create or replace function check_new_id()
10 returns trigger
11 language 'plpgsql'
12 as $body$
13 begin
14     if new.hack_id in (select hack_id from "hack")
15     then raise notice 'hack_id already exists';
16     RETURN NULL;
17     else
18     raise notice 'inserted successfully';
19     RETURN NEW;
20     END IF;
21 end;
22 $body$;
```

```
create or replace function check_new_hack_id()
returns trigger
language 'plpgsql'
as $body$
begin
   if new.hack_id in (select hack_id from hm."Hackathon")
   then raise notice 'hack_id already exists';
   RETURN NULL;
   else
   raise notice 'inserted successfully';
   RETURN NEW;
   END IF;
end;
$body$;
```

```
create or replace trigger check_hack_id
before insert
on hm."Hackathon"
for each row execute function check_new_hack_id();
```

```
insert into "hm"."v1" VALUES (2000 , date '13-05-2020')
```

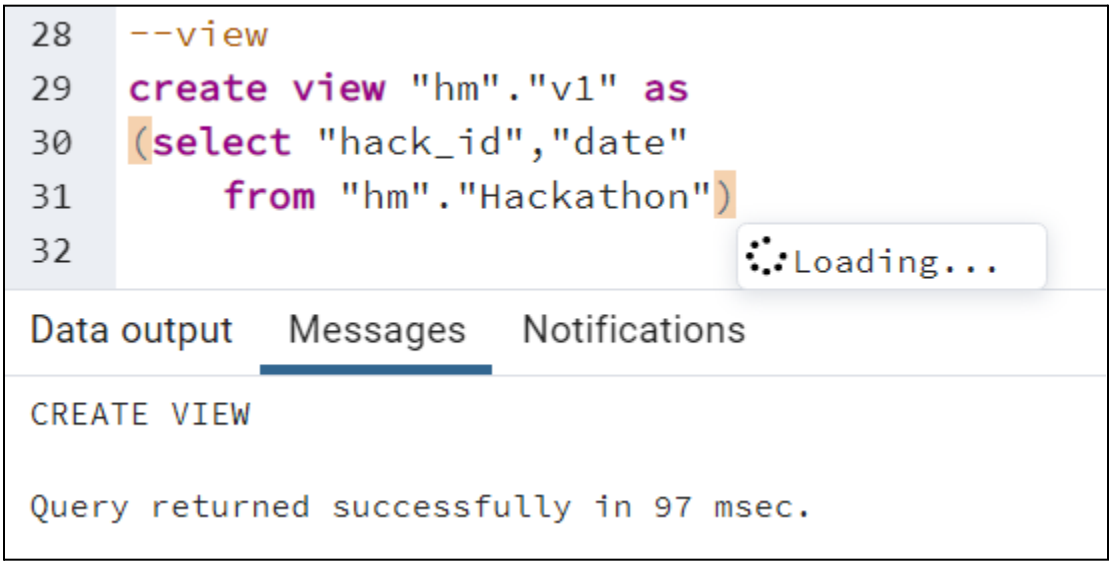
```
14 begin
15     if new.hack_id in (select hack_id from hm."Hackathon")
16     then raise notice 'hack_id already exists';
17     RETURN NULL;
18     else
19     raise notice 'inserted successfully';
20     RETURN NEW;
21     END IF;
22 end;
23 $body$;
24
25     insert into "hm"."v1" VALUES (2000 , date '13-05-2020')
26
27
```

Data output | Messages | Notifications

```
NOTICE:  hack_id already exists
INSERT 0 0

Query returned successfully in 54 msec.
```

20) View

```
--view
create view "hm"."v1" as
(select "hack_id","date"
	from "hm"."Hackathon")

	insert into "hm"."v1" VALUES (2000 , date '13-05-2020')

	select * from hm."Hackathon"
```



```
28  --view
29  create view "hm"."v1" as
30  (select "hack_id","date"
31      from "hm"."Hackathon")
32
33      insert into "hm"."v1" VALUES (2000 , date '13-05-2020')
34
35      select * from hm."v1"
36
```

Data output | Messages | Notifications

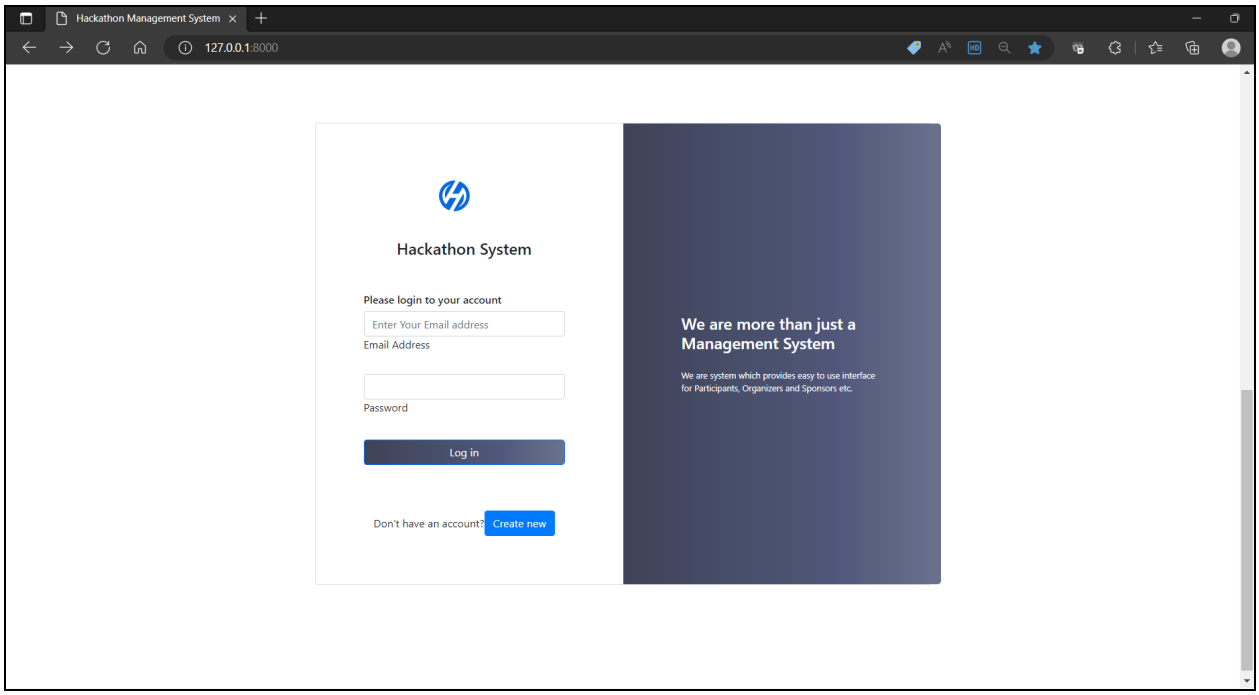
	hack_id character varying	date date
47	47	2022-08-18
48	48	2022-09-15
49	49	2022-10-02
50	50	2022-08-08
51	2000	2020-05-13

```
28  --view
29  create view "hm"."v1" as
30  (select "hack_id","date"
31      from "hm"."Hackathon")
32
33      insert into "hm"."v1" VALUES (2000 , date '13-05-2020')
34
35      select * from hm."Hackathon"
36
```

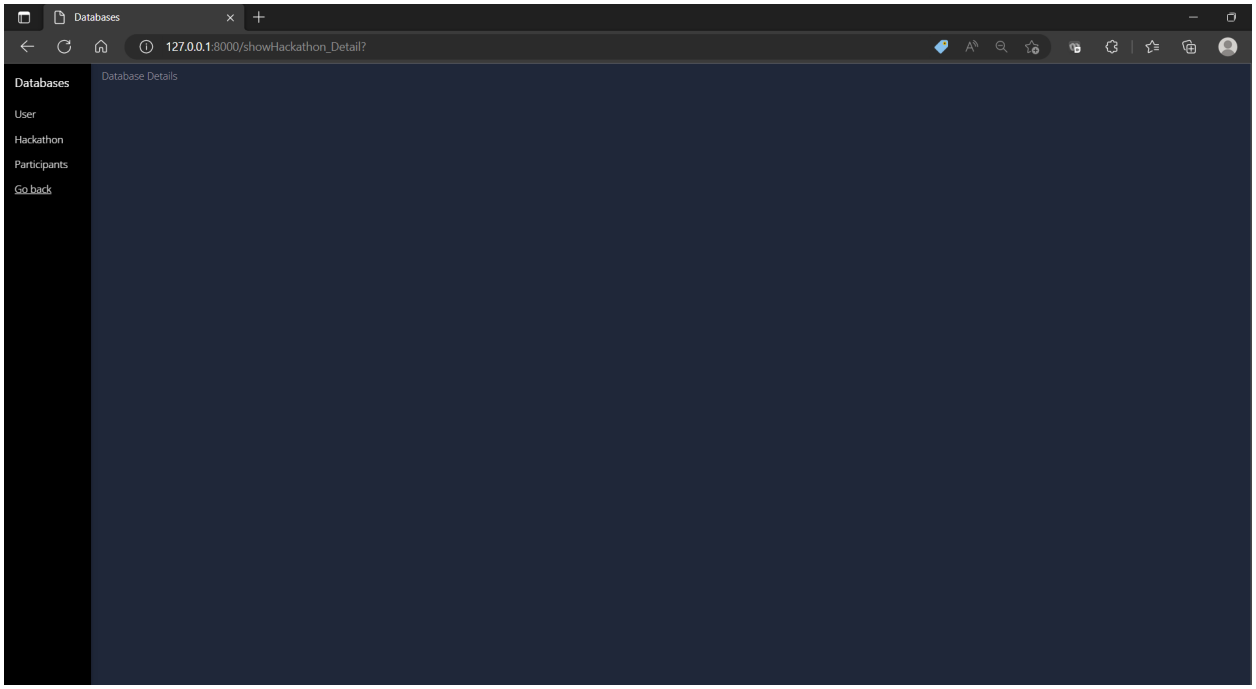
Data output | Messages | Notifications

	hack_id [PK] character varying	date date	s_time time without time zone	e_time time without time zone	duration character varying	theme character varying
47	47	2022-08-18	16:56:00	21:40:00	23:01	Brokerage
48	48	2022-09-15	12:43:00	17:19:00	19:11	NCP
49	49	2022-10-02	16:44:00	22:49:00	20:18	Formation Evalua...
50	50	2022-08-08	00:52:00	09:29:00	05:28	Slackware
51	2000	2020-05-13	[null]	[null]	[null]	[null]

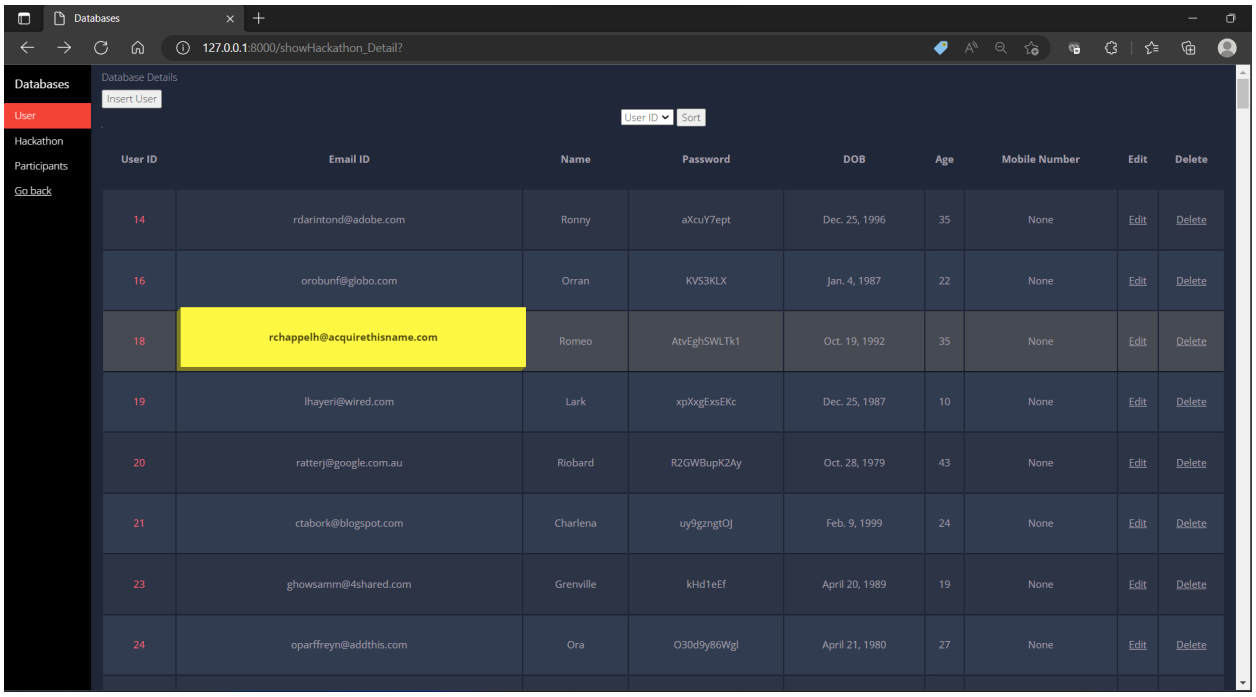
Frontend-Backend Work:



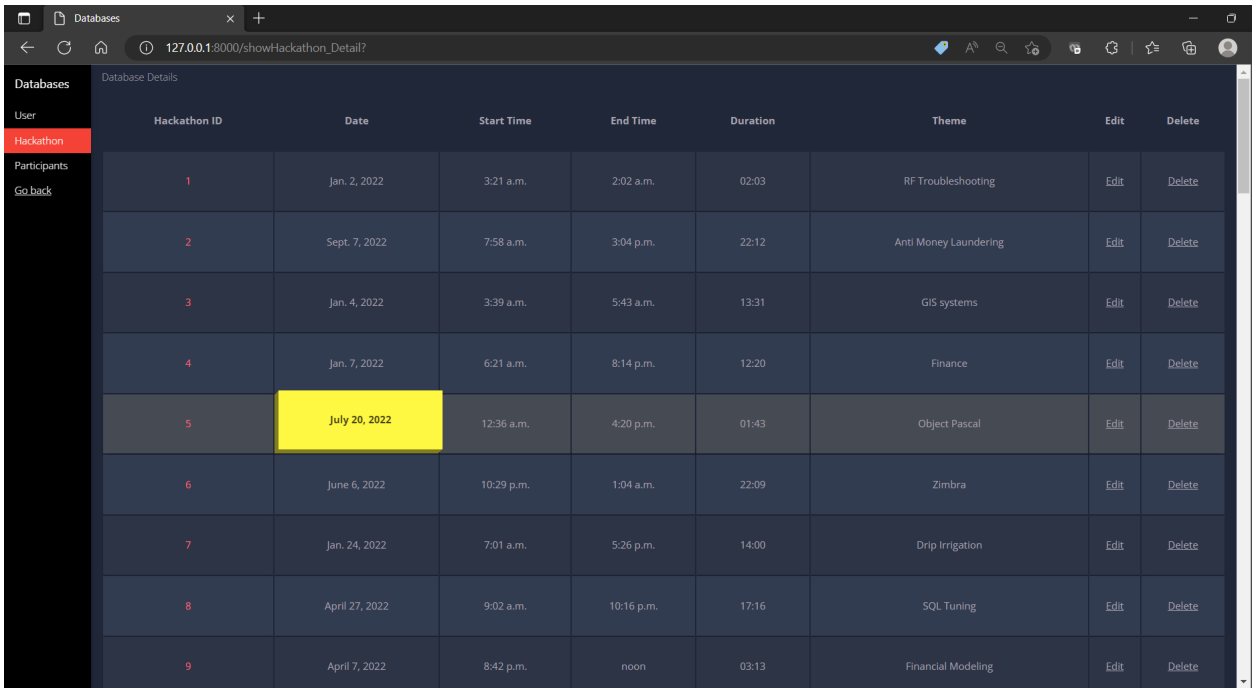
Home page for the management system with the navigation bar.



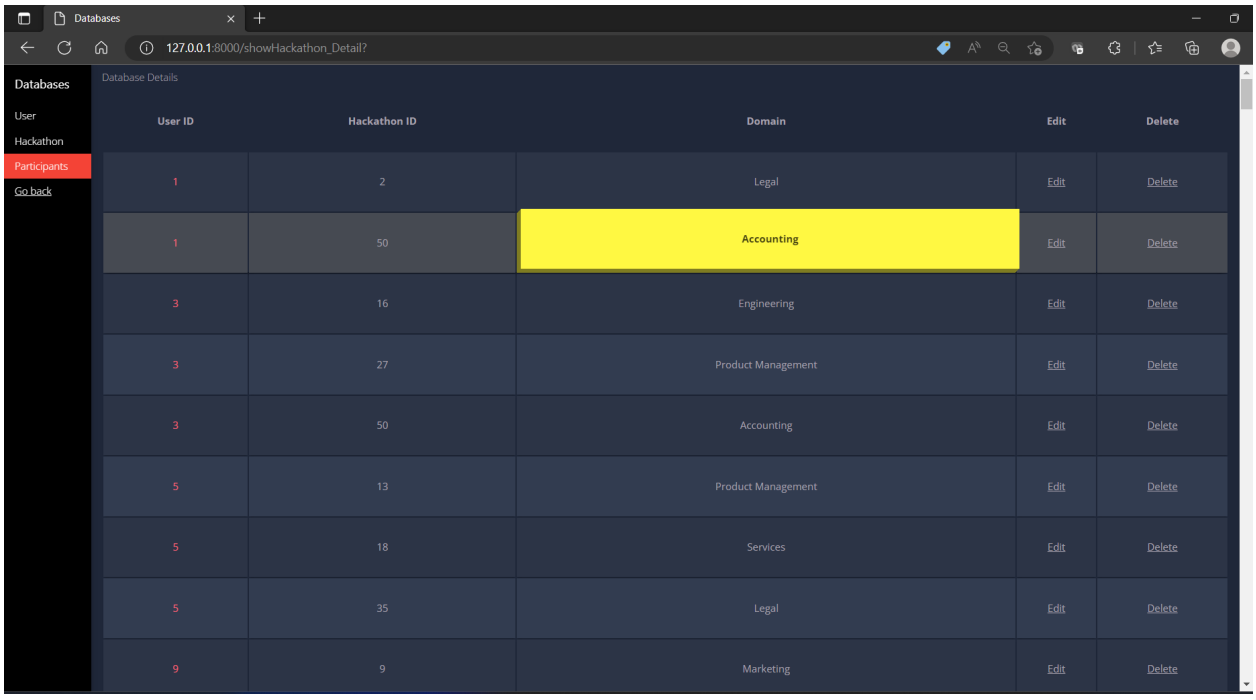
After selecting the option Database details.



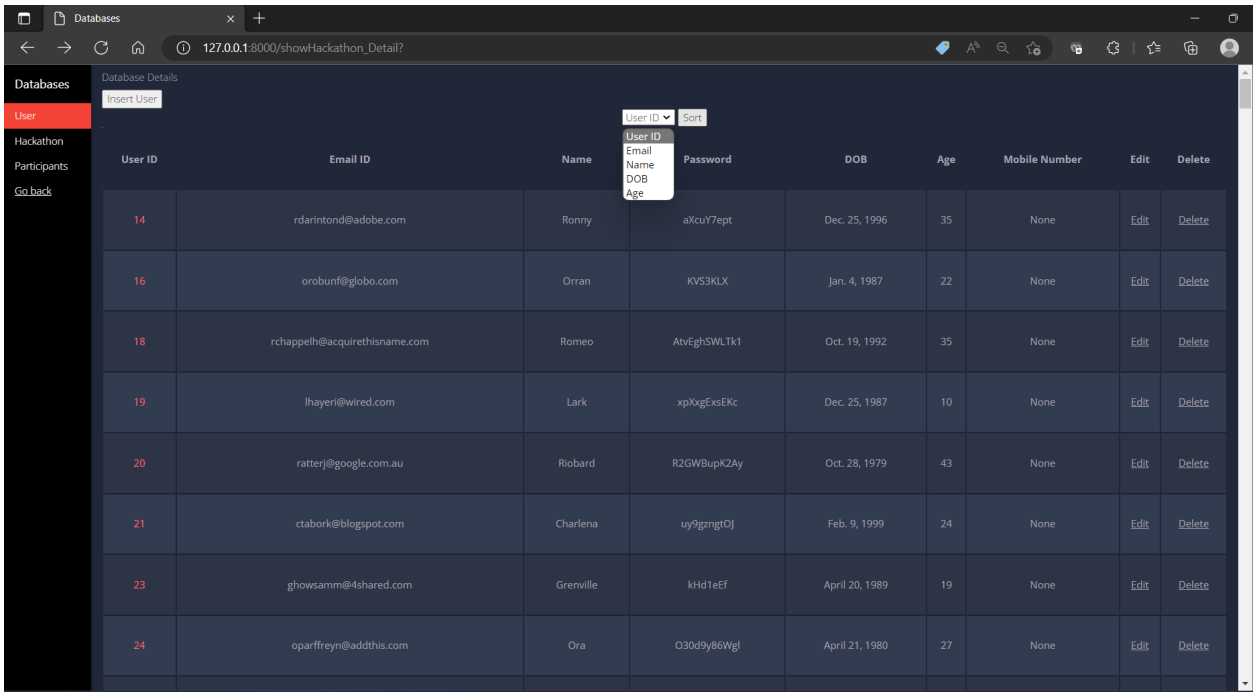
-User-database- with functionality to sort using different parameters along with functionalities to -edit/update- and -delete- particular data.



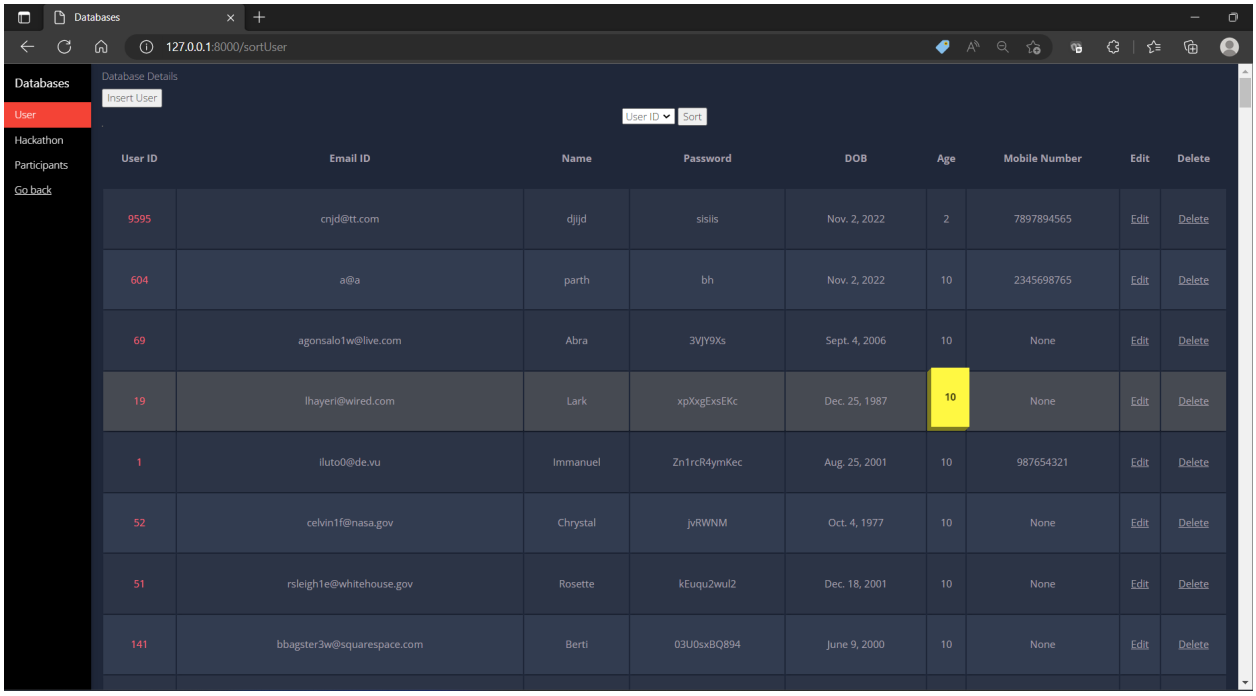
- Hackathon-database - with functionalities to - edit/update - and - delete - particular data.



- Participant-database - with functionalities to - edit/update - and - delete - the particular data.

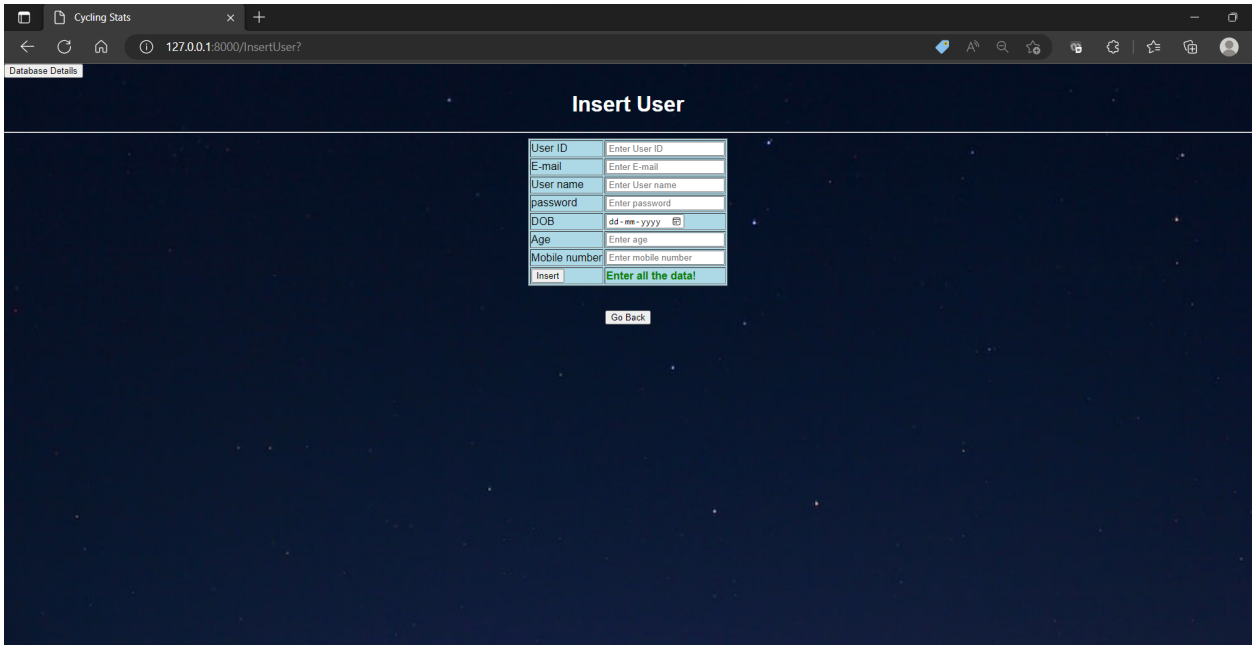


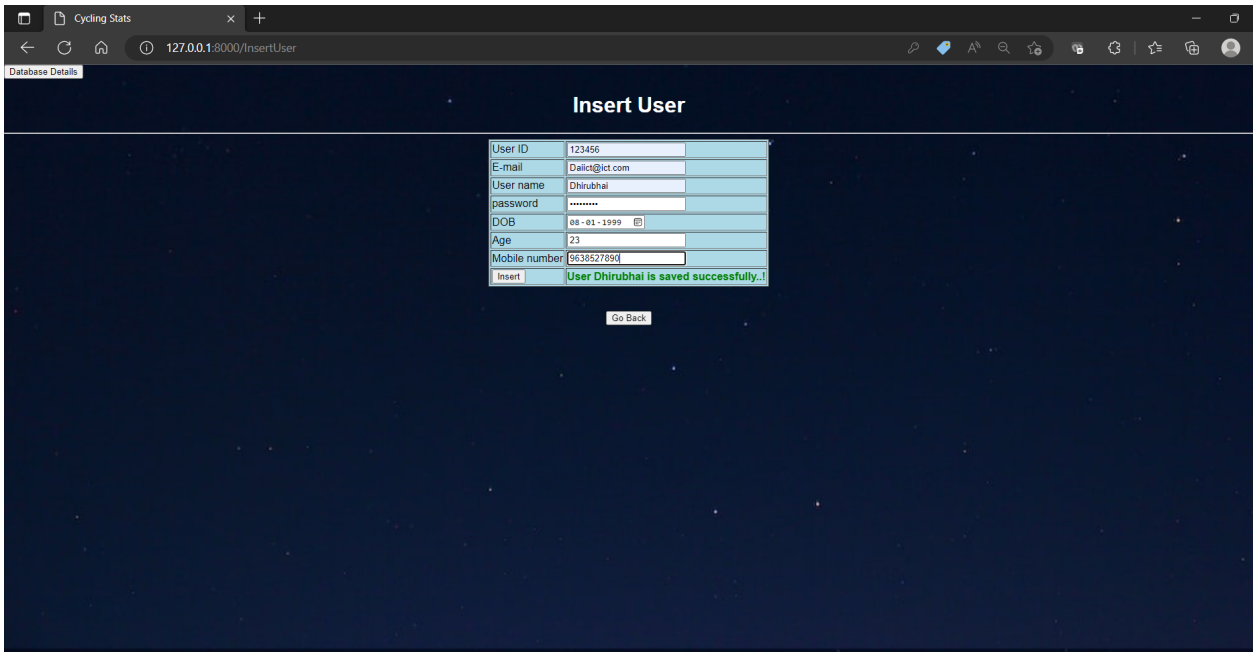
Sorting the user database according to the age of the users.



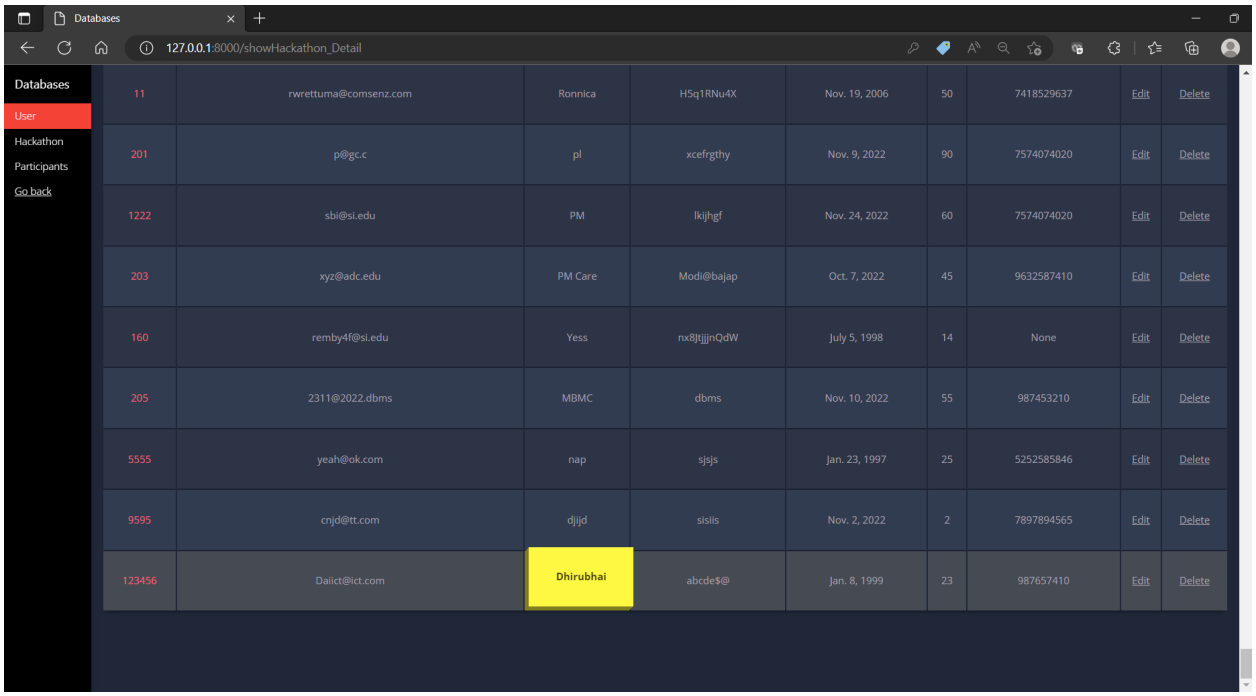
Sorted the user database according to the age of the users.

After selecting the option Sign up.

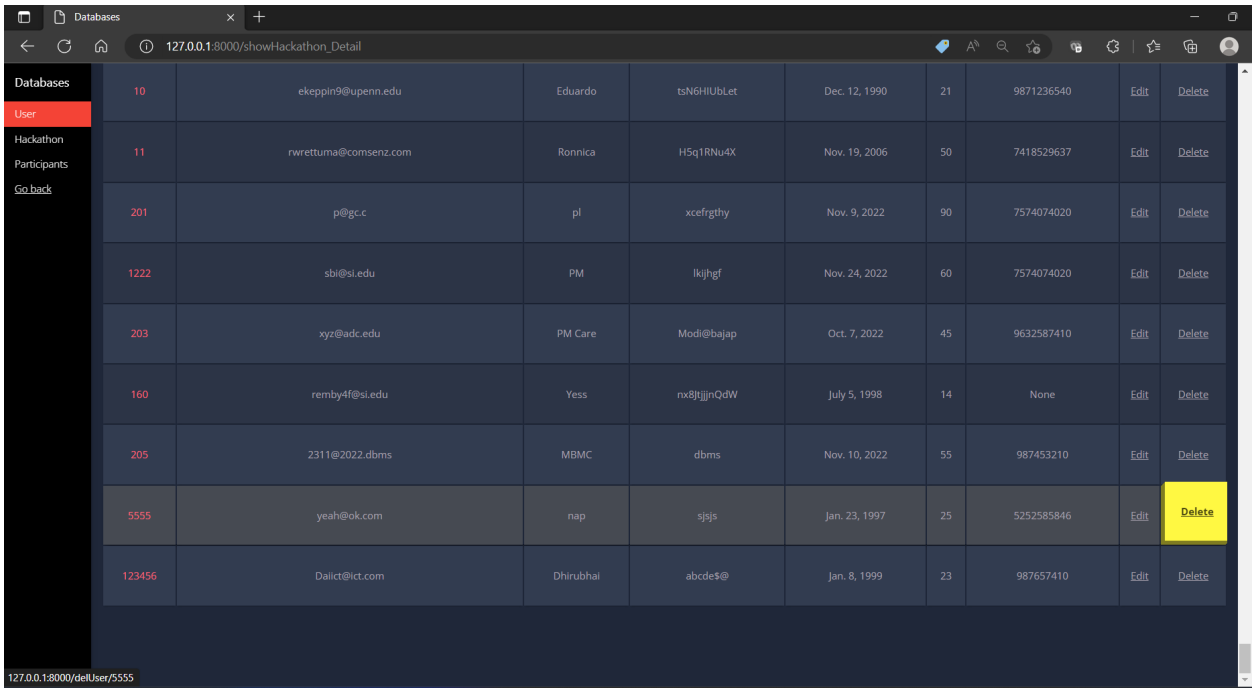




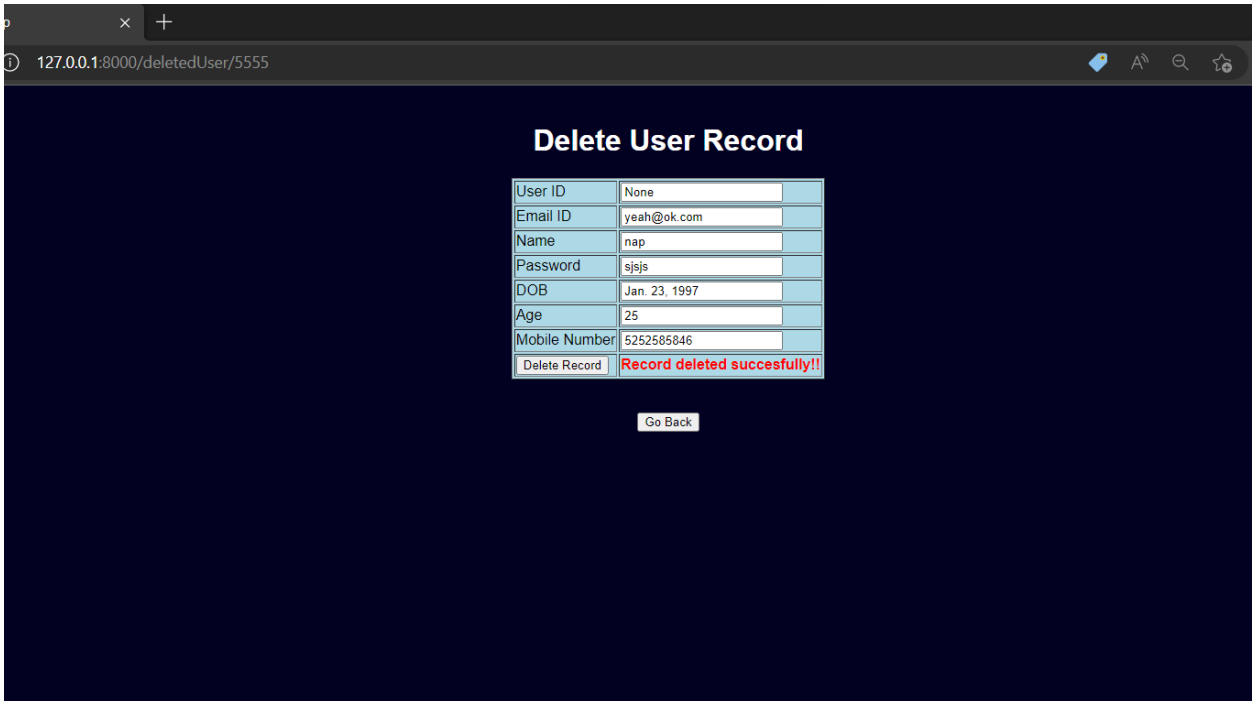
Successfully added the user named ‘Dhirubhai’ along with the details.



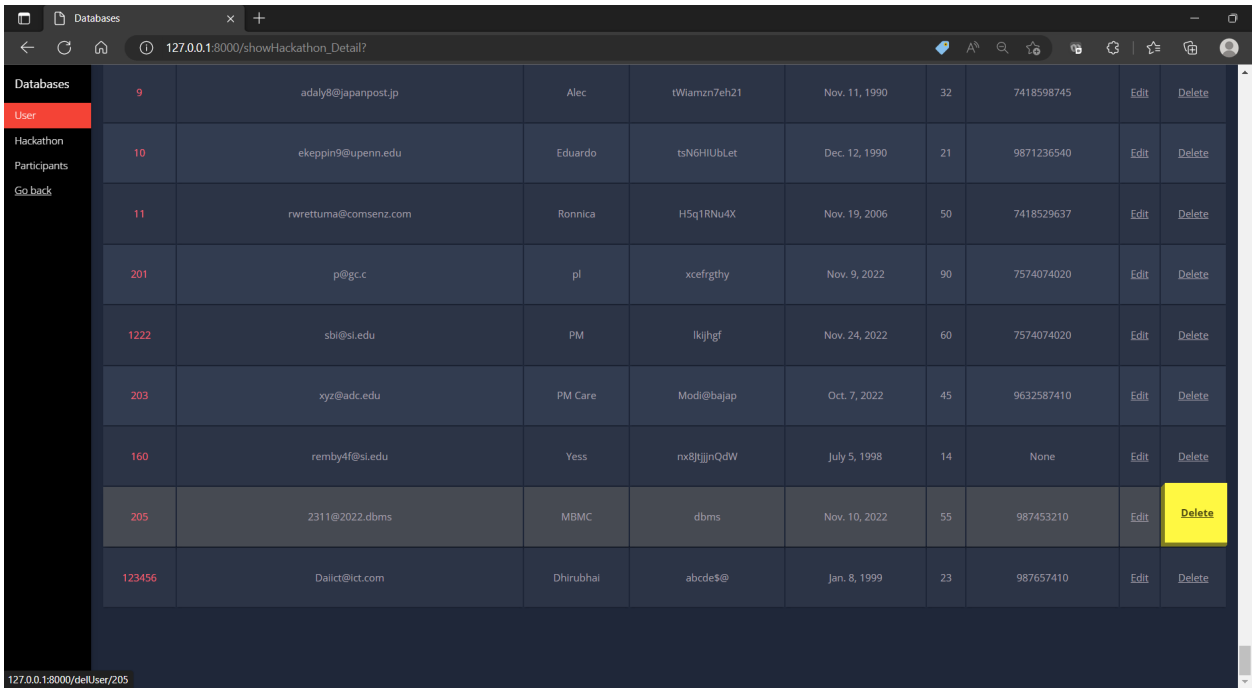
Successfully inserted the user.



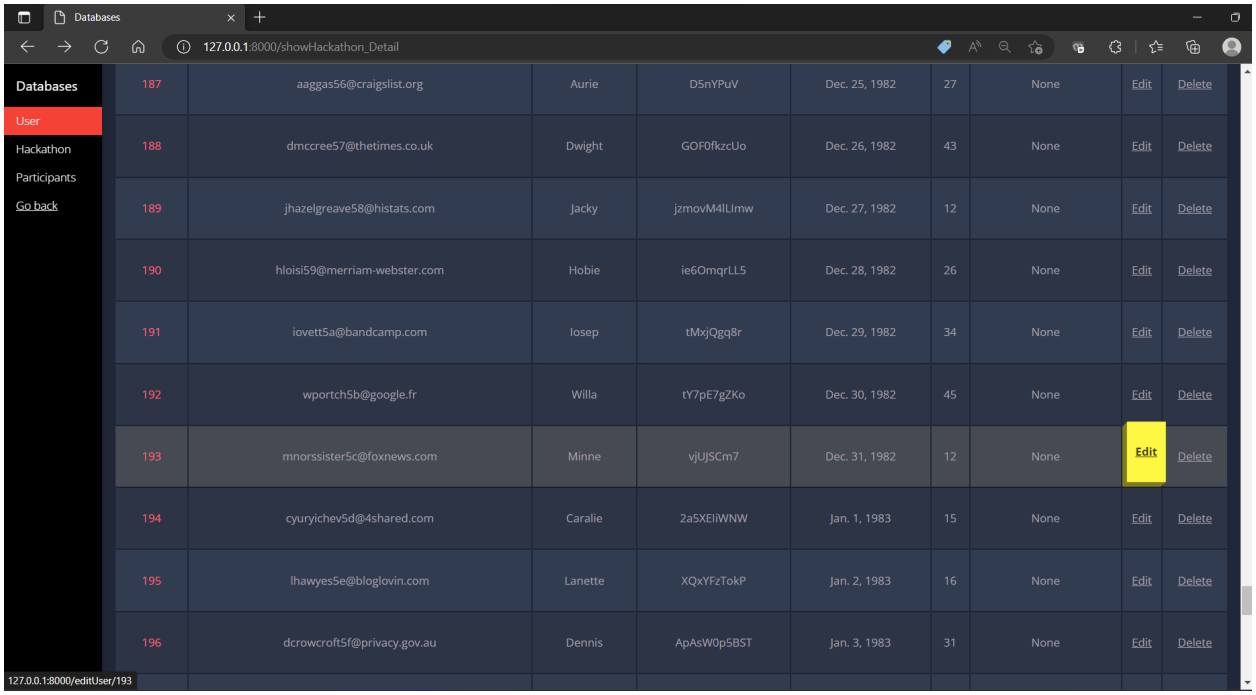
Now we use the delete option to remove a user.



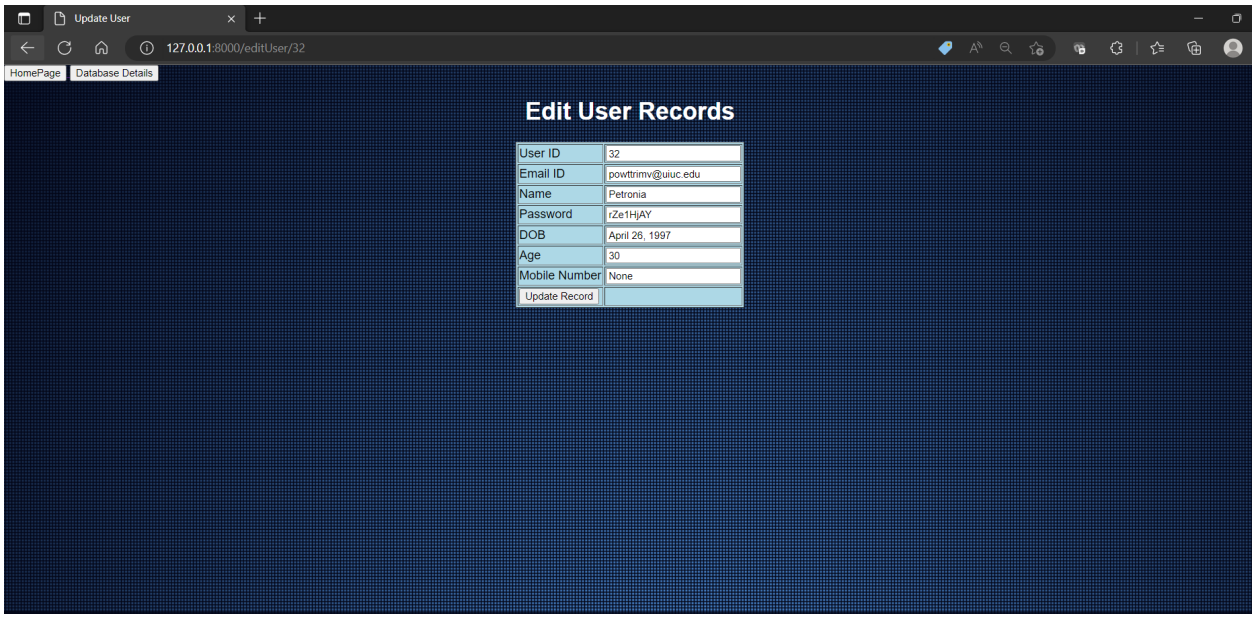
Add the details of the users you want to remove.



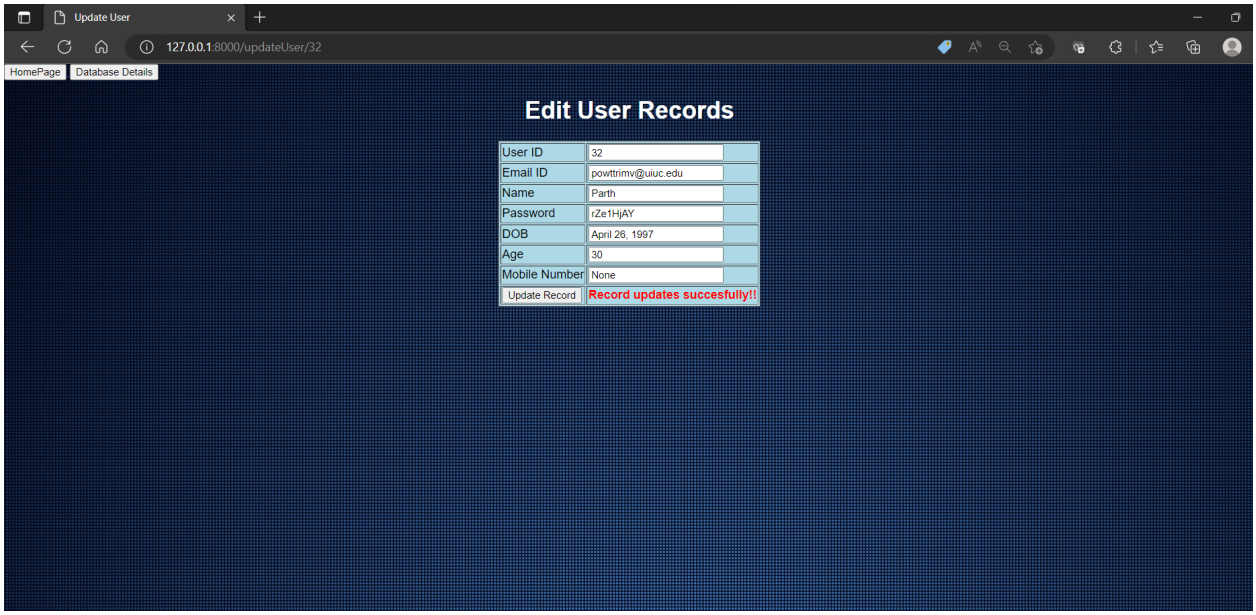
Successfully deleted the user with the given details.



Now we use the Edit option to update a user's details.



Add the updated details of the users you want to edit.

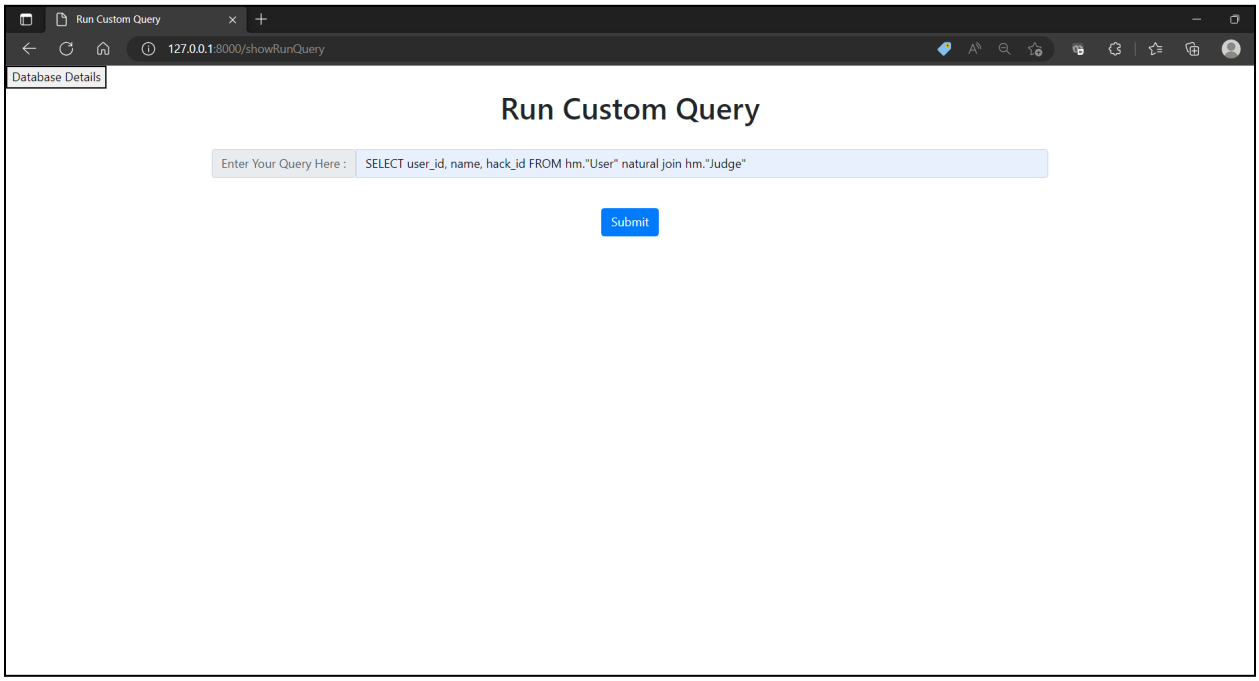


Successfully updated the use data (here name) with the given details.

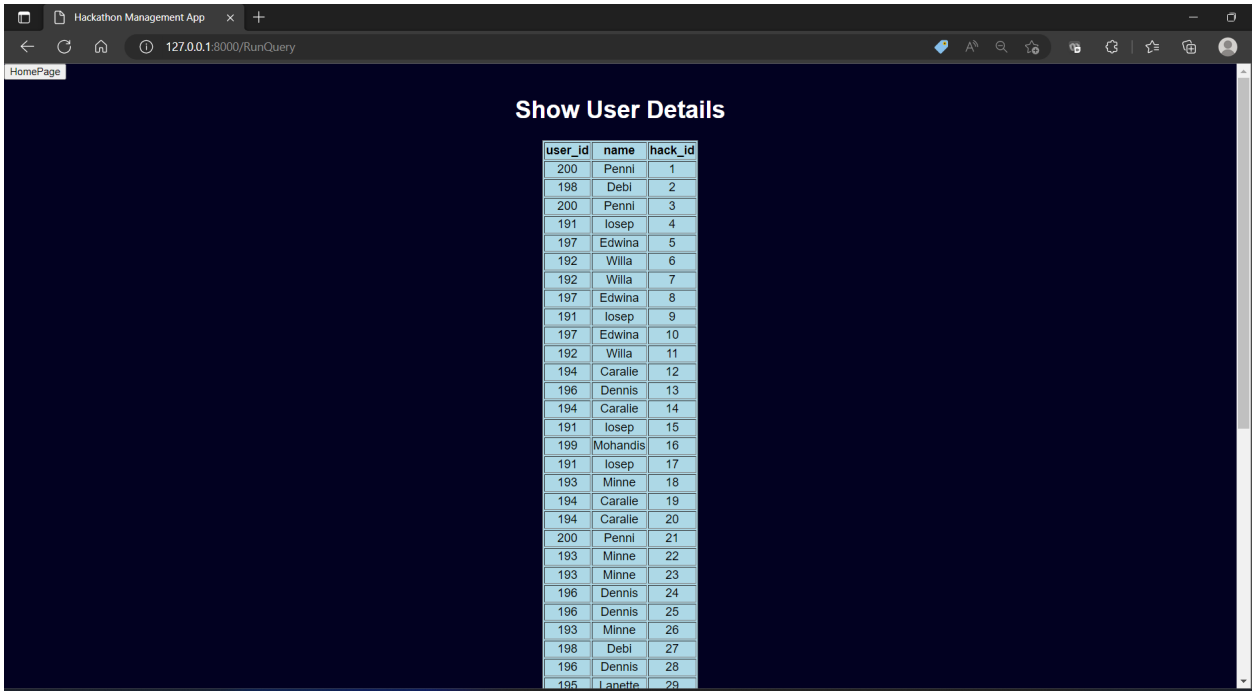
➔ After selecting the option Run Query. Here we can give any valid query and can get the output.

Example-1

Run query to get the user ids and names of the judges along with the hackathon ids of the hackathon they are judging for.



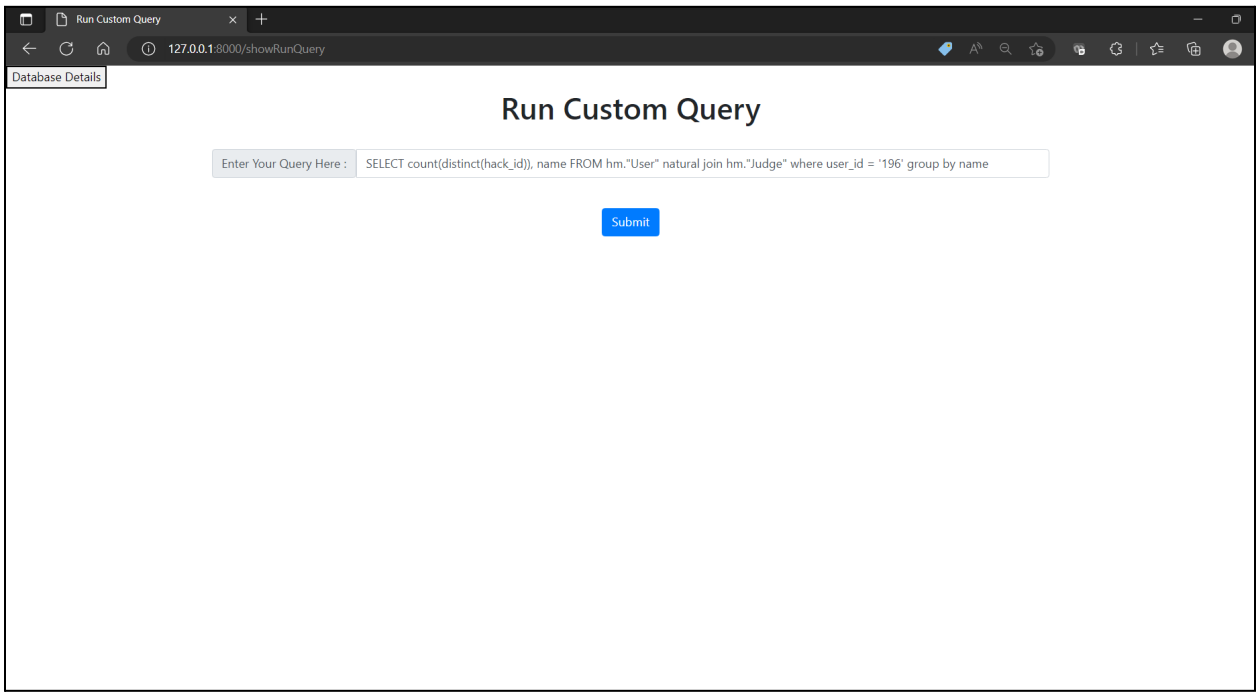
Output:



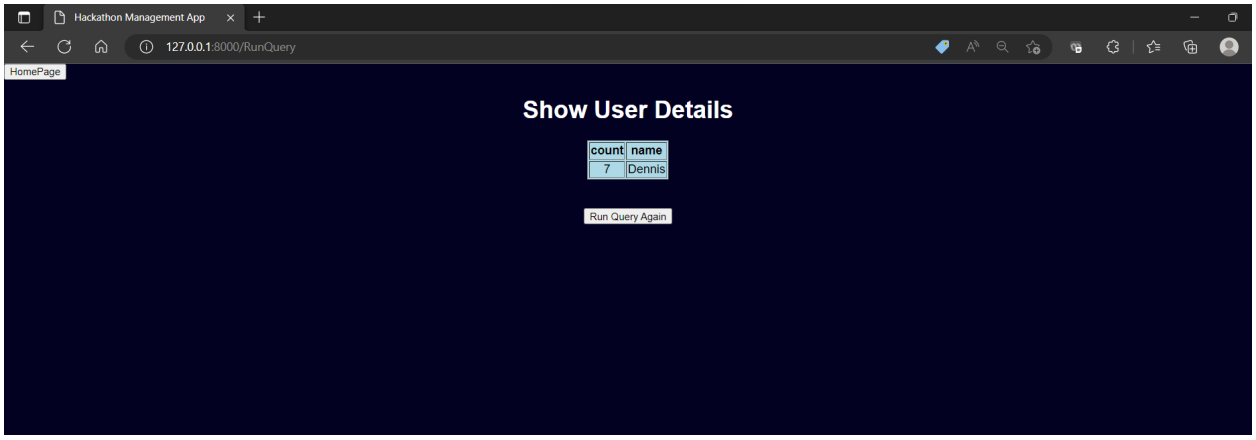
user_id	name	hack_id
200	Penni	1
198	Debi	2
200	Penni	3
191	Iosep	4
197	Edwina	5
192	Willa	6
192	Willa	7
197	Edwina	8
191	Iosep	9
197	Edwina	10
192	Willa	11
194	Caralie	12
196	Dennis	13
194	Caralie	14
191	Iosep	15
199	Mohandis	16
191	Iosep	17
193	Minne	18
194	Caralie	19
194	Caralie	20
200	Penni	21
193	Minne	22
193	Minne	23
196	Dennis	24
196	Dennis	25
193	Minne	26
198	Debi	27
196	Dennis	28
195	Lanette	29

Example-2

Run query to get the count of the total hackathons along with the name of the judge with the user_id = ‘196’.



Output:



GitHub Repository link:

https://github.com/Parthmasala/Hackathon_Management_System.git