IT214 - DBMS

Project - Cycling Stats

Group-4 S1 Team 1.13

Name - Varshil Nayak	ID - 202001180
Name - Hiten Mistry	ID - 202001182

Index

Section 1: Final SRS

1. Introduction
   a) Purpose
   b) Intended Audience and Reading Suggestions
   c) Product Scope
   d) Description
2. Fact-finding phase
   a) Background Readings
   b) Interviews
   c) Questionnaires
   d) Observations
3. Fact finding Chart
4. List of Requirements
5. Users Categories and Privileges
6. Assumptions
7. Business Constraints

1. Introduction

a) Purpose:

➢ The purpose of our system is to create software for managing the details about cyclists all over the world along with their journey path, competitions which they took part in and whole information about the history of a particular cyclist.

➢ System should be capable of providing details about the types of cyclists,types of cycles and list of upcoming events.

➢ This document serves as the guide for the developers of this software system.

b) Intended Audience and Reading Suggestions:

- **Intended audience for our software are:**

❖ **Developers:**
➢ They will understand the technical requirements of this case study and use this information to design a database from the given information.

❖ **Testers:**
➢ They will test the database under proper constraints to verify if it is working properly or not.

❖ **Users:**
➢ The cyclists around the world who want to participate in upcoming events.

➢ The organization which is willing to organize cycling events.

➢ The companies which want to sponsor upcoming events based on the popularity of a particular event.

➢ The general people who love to watch cycling events.

- ➢ The analysts who analyze the data about a particular cyclist.
- ➢ The general online user who wants to know the outcome of an event.

- **Reading Suggestions:**

- ➢ SRS basics Purpose and Scope
- ➢ Target Audience
- ➢ Types of cyclists

c) Product Scope:

- ➢ The software intends to make the process of managing cyclist statistics simple.
- ➢ It is useful for cyclists to choose different cycles so as to increase performance in events. Useful for organization to track the best performing cyclists and make a team of cyclists country-wise so as to organize international events.
- ➢ Describes the list of upcoming events city-wise making it easy for cyclists to choose any event which is going to happen in nearby cities.
- ➢ Keeps track of all competitions the cyclist has taken part in, his past performances and ongoing competitions so one can determine about the quality of a cyclist.
- ➢ The goal is to provide a very intuitive and simple interface to the user and the administrator, so that the user can easily navigate through the system.

d) Description:

- ➢ Our system is for cyclists and analysts all around the globe to get relevant data about cycling events and cyclist performances in the events.

- ➢ System should be able to avoid redundancy and inconsistency of data which affects the database greatly.

- ➢ The system will ensure that data accessibility is only to the people with privileges.

- **The relations we are planning to implement are:**

1) Cyclists:

- ➢ This relation consists of cyclist information who registers which includes Cyclist_name, DOB, Age, Gender, Country, Email_id along with an unique identifier Cyclist_id which identifies a particular cyclist who participates in an event.

2) Events:

- ➢ This relation consists of event information which includes Event_name, Location, Date, Time, Event_type along with an unique identifier Event_id which uniquely identifies a particular event from a group of events.
- ➢ This relation will keep track of all events which the cyclist organizers have organized for the cyclists all over the world.

3) Organizers:

- ➢ This relation consists of organizer information which includes Organizer_name, Contact, Address along with an unique identifier

Organizer_id which uniquely identifies a particular organizer who organizes an event.

➢ This relation will keep track of all the cyclist event organizers which have organized cyclists events in the past.

4) **Participants**:

➢ This relation consists of participant information for an event which includes Event_id, Event_name, Participant_id, Cycle_type, Cyclist_type. It helps in determining the ones who participates in the event.

5) **Sponsors:**

➢ This relation consists of sponsor information which includes Sponsor_name, Sponsor_type, Location, Contact of sponsor who sponsors an event.

➢ This relation will keep track of all the sponsors of cyclists events around the world.

6) **Performance** :

➢ This relation consists of cyclist performance information which includes Event_id, Event_name, Rank,Ranker_id (Winner, RunnerUp, 2ndRunnerUp), Ranker_country (Winner, RunnerUp, 2ndRunnerUp).

➢ We want to design a way for analysts to check the performances of cyclists to plan future events based on the history of performances. This can be done by performance relation which stores the data about the individual cyclist such as their performances in the recently conducted events.

7) Event Status:

- ➢ This relation consists of event outcome information which includes Status,Date.
- ➢ This relation is helpful in case of any rescheduling or cancellation of any event.

8) Admin:

- ➢ This relation consists of admin information which includes admin_id and admin_name.

● Real-World Workflow:

- ➢ The system has many operations related to inserting, updating or deleting data.

- ➢ An existing cyclist can login and check about upcoming cycling events. He/she can participate in the event if they wish to.

- ➢ A new cyclist can register into the system and be assigned a cyclist ID along with access a cyclist has.

- ➢ An existing organizer can login and can update about an event that it organized or can insert a new event that it is going to organize.

- ➢ A new organizer can register and can have the access to organize a new event.
- ➢ A sponsor can check the listed events and may sponsor any event it wishes to sponsor.

- ➢ After completion of the event, the organizer can update the outcome of the event and mark completion of the event.

2. Fact Finding Phase

a) Background Readings:

I. Cycling Stats from https://www.procyclingstats.com/

➢ From this reading, we referred to what kind of statistics we need to maintain and what kind of functions and relations we need to implement for the system.

II. Cyclist Rankings from https://www.cyclingranking.com/

➢ From this reading, we referred to how the rankings of cyclists are maintained based on the performances of the cyclist in cycling events. Also we obtained information about maintaining country-wise statistics.

III. Events Organization from https://cyclingfederationofindia.org/

➢ From this reading, we got basic information about how cycling events are organized (particularly in India) along with getting sponsors for the events. Also, the data is maintained city-wise or state-wise.

- **Combined Requirements gathered from Background Readings:**

➢ A proper database management system is needed for maintaining all the information about cyclists, their stats and events.

➢ System should be capable of maintaining all the basic functionality that is related to a cycling event. E.g. Keeping record of Cyclists and their performance in events along with maintaining country-wise or city-wise stats etc.

➢ System should have a user friendly interface so that the users can easily access the data required without having to know about the actual implementation of the system.

➢ System should be designed in such a way that it ensures fast access to users and avoids redundancy.

b) Interviews

1. Interview plan:

System: Pro Cycling

Interviewee:

1) Jaimin Rathwa **(Role play)** **Designation:** Professional Cyclist

Interviewer:

1) Varshil Nayak **Designation:** Business Development Executive

2) Hiten Mistry **Designation:** Developer at Pro Cycling

Date: 29/9/2002 **Time:** 18:00

Duration: 30 minutes **Place:** Google Meet

Purpose of interview:

To identify the problems and requirements of cyclists regarding the cyclist statistics system.

Agenda:

- To know which problems cyclists face for participating in cycling events.
- To know which system cyclists currently use for participating and overall performance of the current system which is used.
- To get an idea about which things to include in the system which can help cyclists.

- **Interview 1 Summary:**

  System: Pro Cycling

  Interviewee:

  1) Jaimin Rathwa **(Role play)** **Designation:** Professional Cyclist

  Interviewer:

  1) Varshil Nayak **Designation:** Business Development Executive

  2) Hiten Mistry **Designation:** Developer at Pro Cycling

  Date: 29/9/2002 **Time:** 18:00

  Duration: 30 minutes **Place:** Google Meet

  Purpose of interview:

  To identify the problems and requirements of cyclists regarding the cyclist statistics system.

  Result of interview:

  - There is no proper listing of events which causes an event to be missed by a cyclist.
  - The current system lacks management of events and as a result cyclists find it difficult to find an event nearby which they can participate in.
  - To include proper event management which eases the task of registration. Also to store information of cyclists uniquely so that they can directly participate in events without having to fill all information again.

2. Interview plan:

System: Pro Cycling

Interviewee:

1) Harsh Metkel (**Role play**) **Designation :** Event organizer

Interviewer:

1) Varshil Nayak **Designation:** Business Development Executive

2) Hiten Mistry **Designation:** Developer at Pro Cycling

Date: 29/9/2002 **Time:** 17:00

Duration: 30 minutes **Place:** Google Meet

Purpose of interview:

To identify the problems and requirements of event organizers regarding organization of cyclist events.

Agenda:

- To know which problems event organizers face for organizing cycling events.
- To get an idea about how they manage the event based on the location and what they do to increase participation.
- To know their perspective of a better system and to get an idea about which things to include for handling an event.

- **Interview 2 summary:**

System: Pro Cycling

Interviewee:

1) Harsh Metkel (**Role play**) **Designation :** Event organizer

Interviewer:

1) Varshil Nayak **Designation:** Business Development Executive

2) Hiten Mistry **Designation:** Developer at Pro Cycling

Date: 29/9/2002 **Time:** 17:00

Duration: 30 minutes **Place:** Google Meet

Purpose of interview:

To identify the problems and requirements of event organizers regarding organization of cyclist events.

Result of interview:

- There is a lack of sponsors for organizing an event which reduces the chances of organizing upcoming events.
- They do advertising of event through the sponsors which increases participation and based on the amount of participants they select an appropriate place to organize the event.
- To include accurate event information and timings at a place which eases the task for participants and also sponsors to track the upcoming events.

- **Combined Requirements gathered from Interviews:**

➢ System should have a proper listing of events along with proper timings and location such that cyclists get a clear idea about upcoming events which they may participate in.

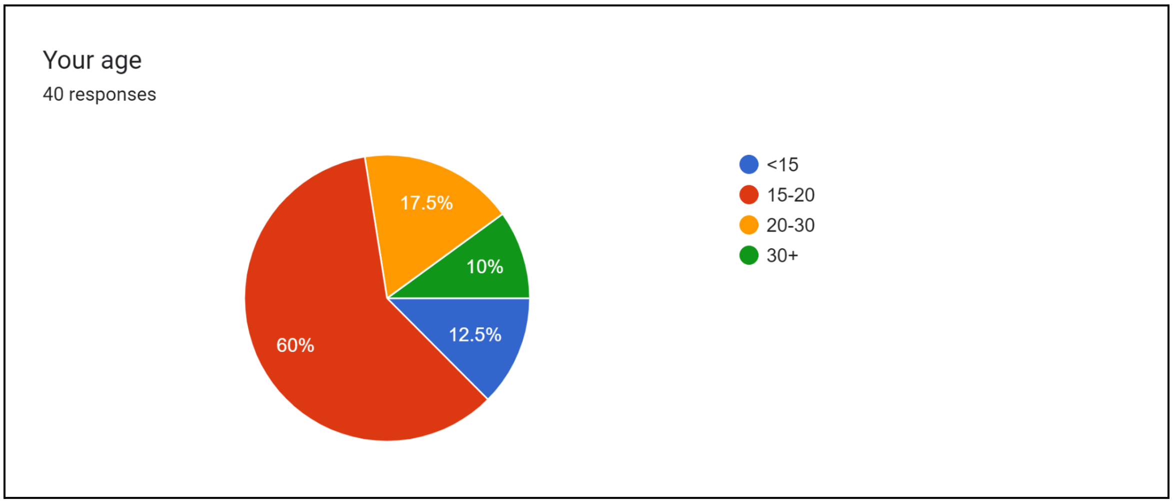
➢ System should be capable of maintaining proper updates about the event. Eg: Cyclists should be able to get an update if any event is canceled or if there is a change in timings.

➢ System should store cyclists information uniquely which enables them to participate in upcoming events directly using stored information.

➢ The events should be shown properly which attracts sponsors.

c) Questionnaires:

1.



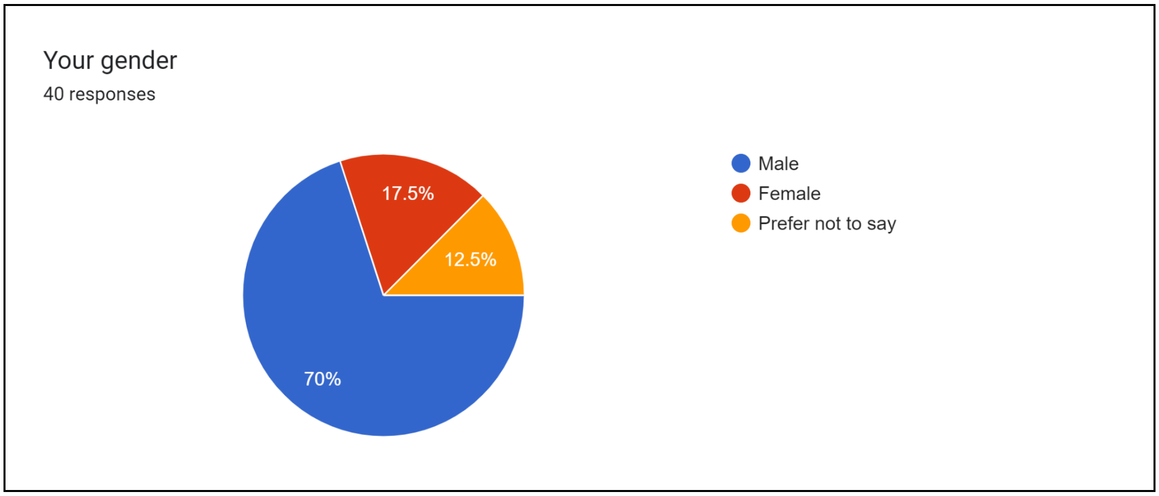
- **Intent of the question:**

➢ To get a basic idea about what age group users belong to, so as to design the interface accordingly.

- **Observation from responses:**

➢ Majority of users lie in the age group 15-30 so software needs to be designed accordingly.

2.



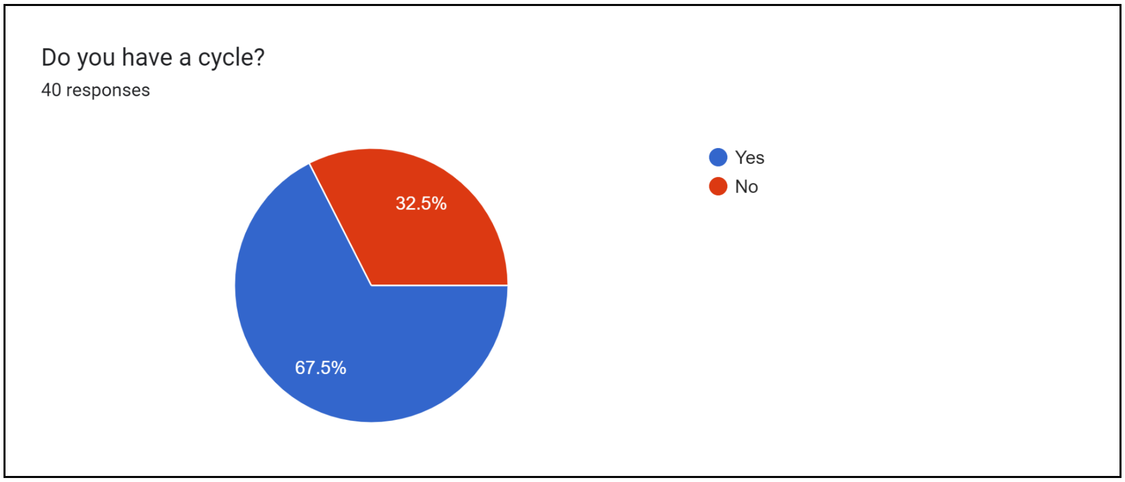
- **Intent of the question:**

➢ To know about the proportion of users according to their gender.

- **Observation from responses:**

➢ Majority of users are male so we need to come up with something which encourages females to participate in cycling events.

3.



- **Intent of the question:**

➢ To get a basic idea about the proportion of users who own a cycle.

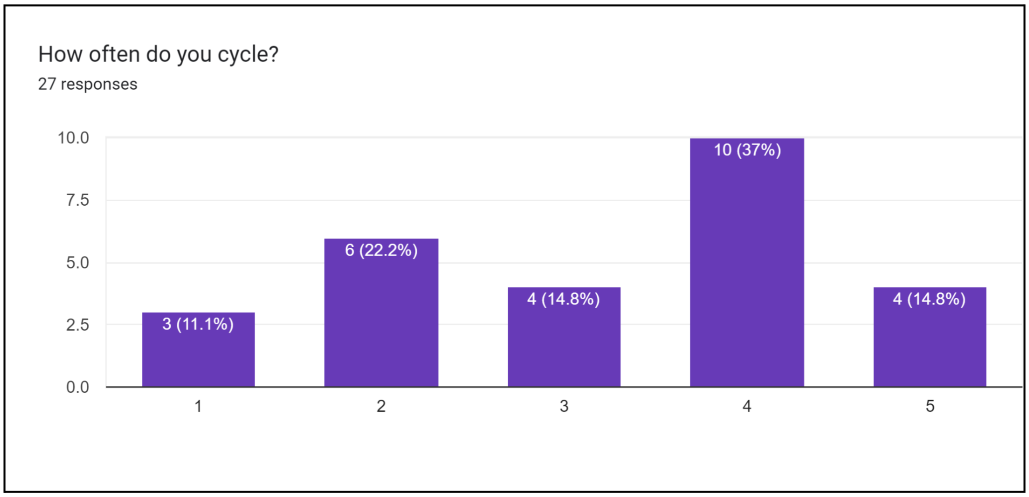
- **Observation from responses:**

➢ About 2/3rd of users have a cycle and 1/3rd don't have one.

➢ **We further classified users based on their response to the previous question. Users who own a cycle as cyclists and others as audience.**

❖ Cyclist information:

4.


How often do you cycle?

27 responses

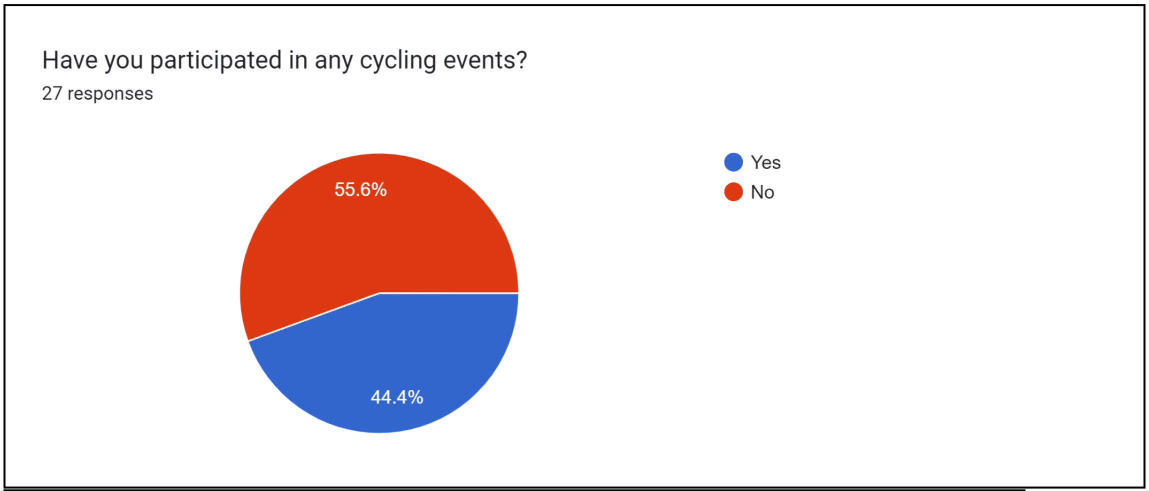
- **Intent of the question:**

➢ To know about how frequent users cycle.

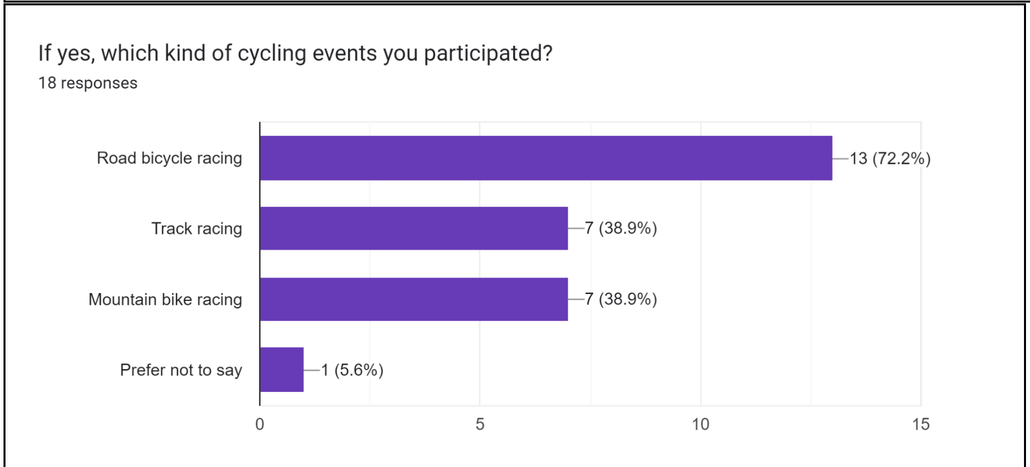
- **Observation from responses:**

➢ Majority of users cycle frequently. So we need them to participate in events.

5.


Have you participated in any cycling events?

27 responses


If yes, which kind of cycling events you participated?

18 responses

- **Intent of the question:**

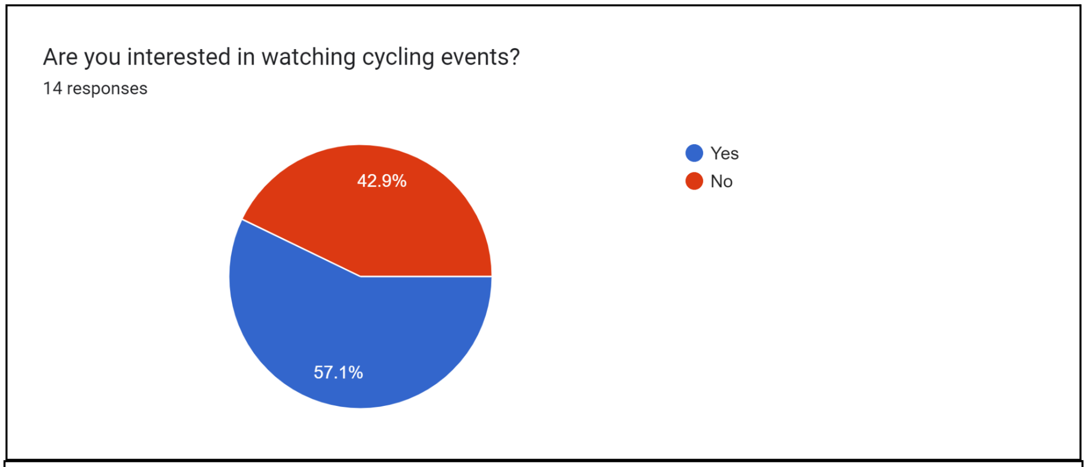
➢ To know about the user's participation in any cycling events.

- **Observation from responses:**

➢ Majority of users haven’t participated in any event. From those who participated, the majority took part in road bicycle racing. So we need to organize events accordingly.

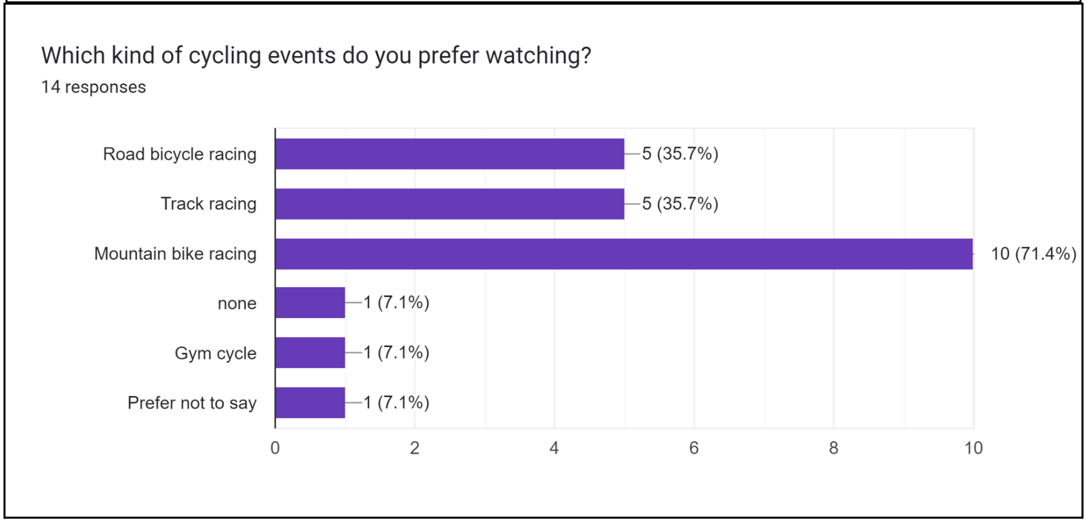
❖ **Audience information:**

6.



Are you interested in watching cycling events?

14 responses



Which kind of cycling events do you prefer watching?

14 responses

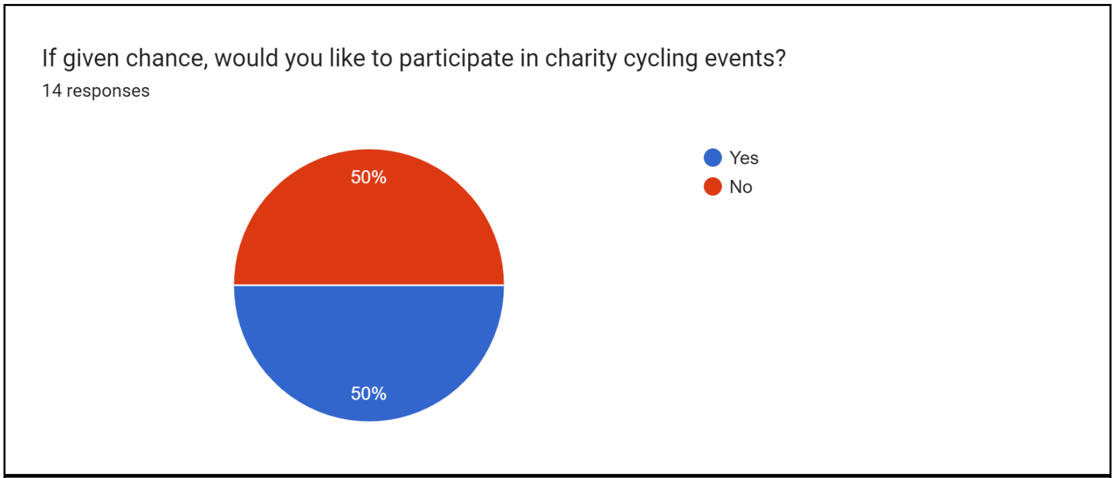
- **Intent of the question:**

➢ To know about the interest in watching events from users who don’t have a cycle and also type of event users want to watch.

- **Observation from responses:**

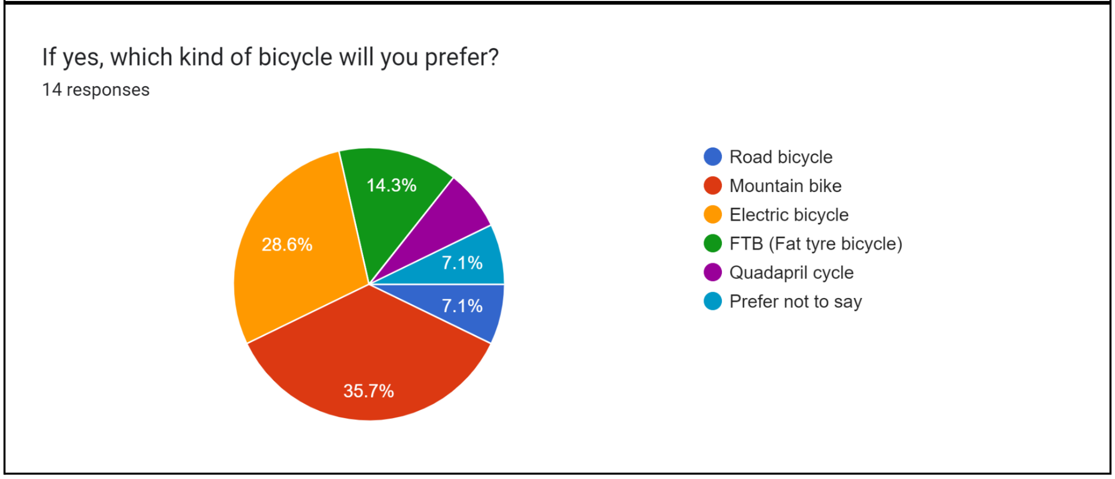
➢ Majority of users are interested in watching cycling events and particularly they prefer watching events like mountain bike racing,track racing and road racing more.

7.



If given chance, would you like to participate in charity cycling events?

14 responses



If yes, which kind of bicycle will you prefer?

14 responses

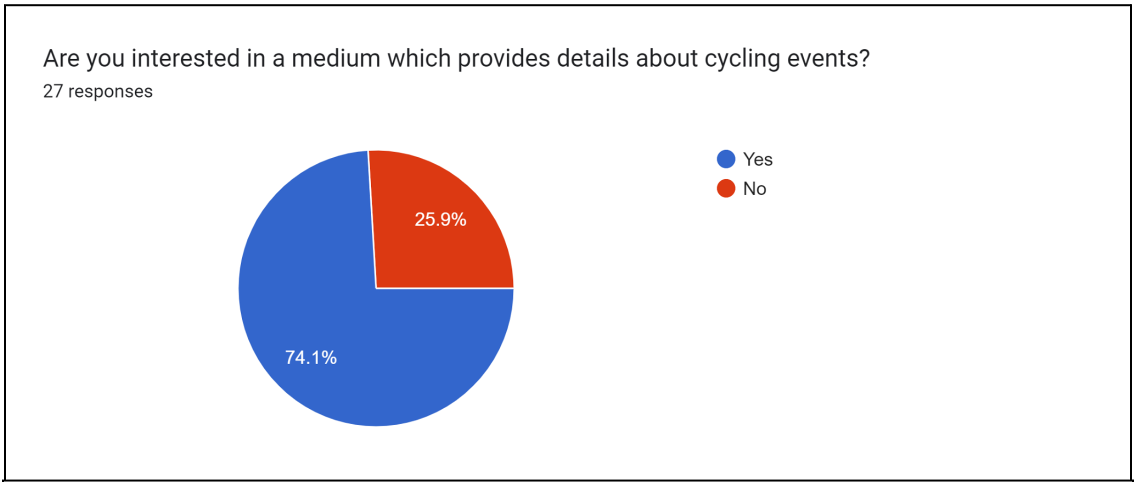
- **Intent of the question:**

➢ To get preference of the audience if they want to participate in charity events or not.

- **Observation from responses:**

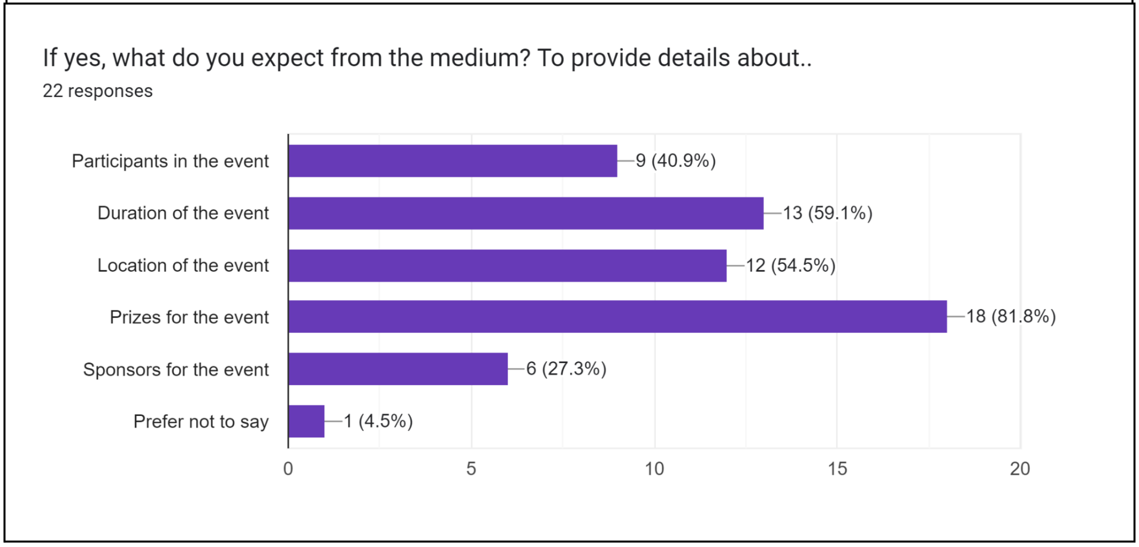
➢ Based on the responses, we do not get clear preference of the audience but those interested in participating preferred mountain bike and electric bike more.

❖ Question based on the system:



Are you interested in a medium which provides details about cycling events?

27 responses



If yes, what do you expect from the medium? To provide details about..

22 responses

- **Intent of the question:**

➢ To get basic ideas from users if they want a system which provides maintaining data about cycling events.

- **Observation from responses:**

➢ Majority of users want a system like that which suggests the need to make such a system capable of handling a list of information which users preferred the system should include (from above responses).

- **Combined Requirements gathered from Questionnaires:**

➢ The system environment should be designed such that it suits people from the age group 15-30.

➢ There should be events for females only so that their participation in cyclist events increases.

➢ Most cyclists cycle frequently so the system should be capable of encouraging them in taking part in cycling events.

➢ Cyclists prefer road bicycle racing and mountain bike racing so the system should be capable of organizing more of such events to increase participation.

➢ Also most of the audience prefer watching the above mentioned events so it is appropriate for the system to organize such events for increasing audience.

➢ Most users prefer mountain bike and electric bike so we can keep the record for the purpose of sales and maintanence.

➢ Most users wanted to have a system for cycling events which suggests the need to make such a system.

d) Observations:

System: Pro Cycling

Observations by:

1) Varshil Nayak **Designation:** Business Development Executive

2) Hiten Mistry **Designation:** Developer at Pro Cycling

Date: 30/9/2002 **Time:** 18:00

Duration: 30 minutes **Place:** Google Meet

Observations:

- Lack of security for user data as data is accessible to all.
- Real-time update is needed for data as per results of a particular event which was recently conducted.
- There is proper storing of cyclists data as well as event information data.
- No proper central medium which manages the whole system.
- No classification of users based on category.
- Privileges to data should be given based on user classification.

- **Combined Requirements gathered from Questionnaires:**

➢ System should be capable of updating data in real-time.
➢ Users should be classified and based on classification they should be given privileges.
➢ System should be capable of providing security to user data by denying access to unauthorized users.

3. Fact finding Chart

Objective	Technique	Subjects	Time commitment
To get background knowledge of different types of cyclists stats and about organizing cycling events.	Background reading	Online website/apps /articles	0.5 Day
To find the problems faced by cyclists regarding the cyclist statistics system.	Interview	Professional Cyclist	45 min
To find the problems faced by event organizers in organizing a cycling event.	Interview	Event Organizer	45 min
To understand the user's perspective	Questionnaire	General Users	1 Day
To observe the system's working	Observation	2 Admin Staff	2 x 1 Hour each

4. List of Requirements

- A proper database management system is needed for maintaining all the information about cyclists, their stats and events, which should be capable of maintaining all the basic functionality that is related to a cycling event.

- System should have a user friendly interface so that the users can easily and fastly access the data required along with avoiding redundancy.

- Systems should have restrictions based on the user classification so that security of data is maintained because of user restriction.

- System should be capable of updating the database in real-time based on the outcomes of events.

- System should have a proper listing of events along with regular updates about the event. Also, the events should be shown properly which attracts sponsors.

- System should store cyclists information uniquely which enables them to participate in upcoming events directly using stored information.

- The system environment should be designed such that it suits people from the age group which is most active in cycling. Moreover, organizing events based on gender should also be considered.

- Event location should be decided based on the preference of cyclists to increase participation and audience.

5. Users Categories and Privileges

1. **Cyclists:**

➢ They are the ones who can participate in cycling events. They can be distinguished using cyclist_id.

- **Privileges:**
    - ➢ They can only view the information related to them such as their name, id, events participated, etc. and related to event information.
    - ➢ They have access to participate in upcoming cycling events.

2. **Event Organizers:**

➢ They are the ones who organize a particular cycling event. They can be identified by organizer_id.

- **Privileges:**
    - ➢ They can view information related to events such as timings,location,etc.
    - ➢ They have access to organize a new cycling event.

3. **Sponsors:**

➢ These are the companies which sponsor a particular cycling event.

- **Privileges:**
    - ➢ They can view basic information about cyclists and cycling events.
    - ➢ They have the access to sponsor a particular event.

4. **Developers(Admins):**

➢ They are responsible for the development and working of the software. In case of technical issues, they will be responsible for solving them. They are the admin of the database.

- **Privileges:**
    - ➢ They are the admin of the system so they have all the privileges of the system.

5. **Audience:**

➢ They are general users who are interested in cycling events.

- **Privileges:**
  - ➢ They have read-only privilege for the system.

6. Assumptions

➢ It is assumed that users of this system have a basic understanding of how the system works.

➢ It is also assumed that information in the website from which we got the idea is correct.

➢ It is assumed that people that we surveyed, provided correct information without any influence.

➢ It is assumed that the sponsor companies have an idea about this system so that they can sponsor events.

7. Business Constraints

➢ The user participation is limited as the event may be scheduled at a location very far from where they reside.

➢ The organizing of an event is dependent on the sponsors which may lead to cancellation of an event if there are no sponsors.

➢ The database storage is limited whereas its usage is more so there may arise a need to increase storage along with maximizing profits.

➢ As there is no automation, the changes in data have to be done manually.

➢ Data needs to be consistently monitored in order to provide accurate details as there is no verification of its correctness.

Section 2: Noun Analysis

1. Noun & Verb Analysis
2. Entity-Attribute
3. Rejected Nouns
4. Rejected Verbs

1. Noun & Verb Analysis

Noun	verb
Our	
system	is
Cyclists	get
analysts	get
globe	
data	cycling
events	
system	be
redundancy	avoid
inconsistency	avoid
data	
Database	affects
system	ensure
data	
accessibility	is
people	
privileges	
relations	are
we	planning,implement

cyclists	consists
cyclist	registers
information	includes
Cyclist_name	
DOB	
Age	
Gender	
Country	
Cyclist_type	
Email_id	
Cycle_type	
identifier	
Cyclist_id	identifies
cyclist	participates
event	
relation	consists
event	
information	includes
Event_name	
Location	
Date	
Time	
Event_type	

Organizer_id	
organizer	
Sponsor_name	
identifier	
Event_id	identifies
relation	keep
events	
organizers	organized
relation	consists
organizer	
information	includes
Organizer_name	
Contact	
Address	
identifier	
organizer_id	identifies
organizer	organizes
relation	consists
sponsor	
information	includes
Sponsor_name	
Sponsor_type	
Location	

Contact	
event	
sponsor	sponsors
relation	keep
cyclist	
performance	
information	includes
Event_id	
Event_name	
Ranker_id	
Ranker_country	
rank	
runnerup	
2ndrunnerup	
country	
we	want
way	design
analysts	check
history	
performances	
cyclists	plan
future	
events	based

performance	Be done
relation	stores
events	conducted
relation	consists
event	
outcome	
information	includes
Status	
date	
admin	
relation	consists
information	includes
admin_id	
admin_name	
system	has
operations	related
data	Inserting, deleting, updating
cyclist	Existing, login, check
events	
he/she	participate
event	
they	wish
cyclist	register

system	Be assigned
access	has
organizer	Existing, login, update, insert
event	organized
it	Is going, organize
organizer	Register, have
access	organize
sponsor	Check, sponsor
events	listed
completion	
event	
organizer	update
outcome	
mark	

2. Entity-Attribute

Candidate entity set	**Candidate attribute set**	**Candidate relationship set**
Cyclist	**Cyclist_id** Cyclist_name DOB Age Gender Country Email_id	Registers,History
Event	**Event_id** Event_name Location Date Time Event_type	Registers,Organizes, Sponsors,Conclusion, Results,Participates
Participants	**Event_id** Event_name **Cyclist_id** Cyclist_type Cycle_type	Participates
Organizer	**Organizer_id** Organizer_name Contact Address	Organizes
Sponsor	**Sponsor_name** Sponsor_type Location Contact	Sponsors
Performance	**Event_id** Event_name **Rank**	Results,History

	Cyclist_id (Winner, Runnerup, Second_runnerup) Cyclist_country (Winner, Runnerup, Second_runnerup)	
Event Status	**Status** Date	Conclusion

3. Rejected Nouns

Noun	Reject Reason
Our	General
system	Duplicate
Cyclists	General
analysts	Irrelevant
globe	Irrelevant
data	Vague
events	General
redundancy	Irrelevant
inconsistency	Irrelevant
Database	Irrelevant
accessibility	Irrelevant
people	General

privileges	Irrelevant
relation	General,Duplicate
we	General
information	Vague,Duplicate
identifier	Irrelevant,Duplicate
organizers	General
we	General
way	Irrelevant
analysts	Irrelevant
performances	General
future	Irrelevant
outcome	Irrelevant
operations	Irrelevant
he/she	General
they	General
access	Vague
it	General
completion	Irrelevant
mark	Irrelevant

4. Rejected Verbs

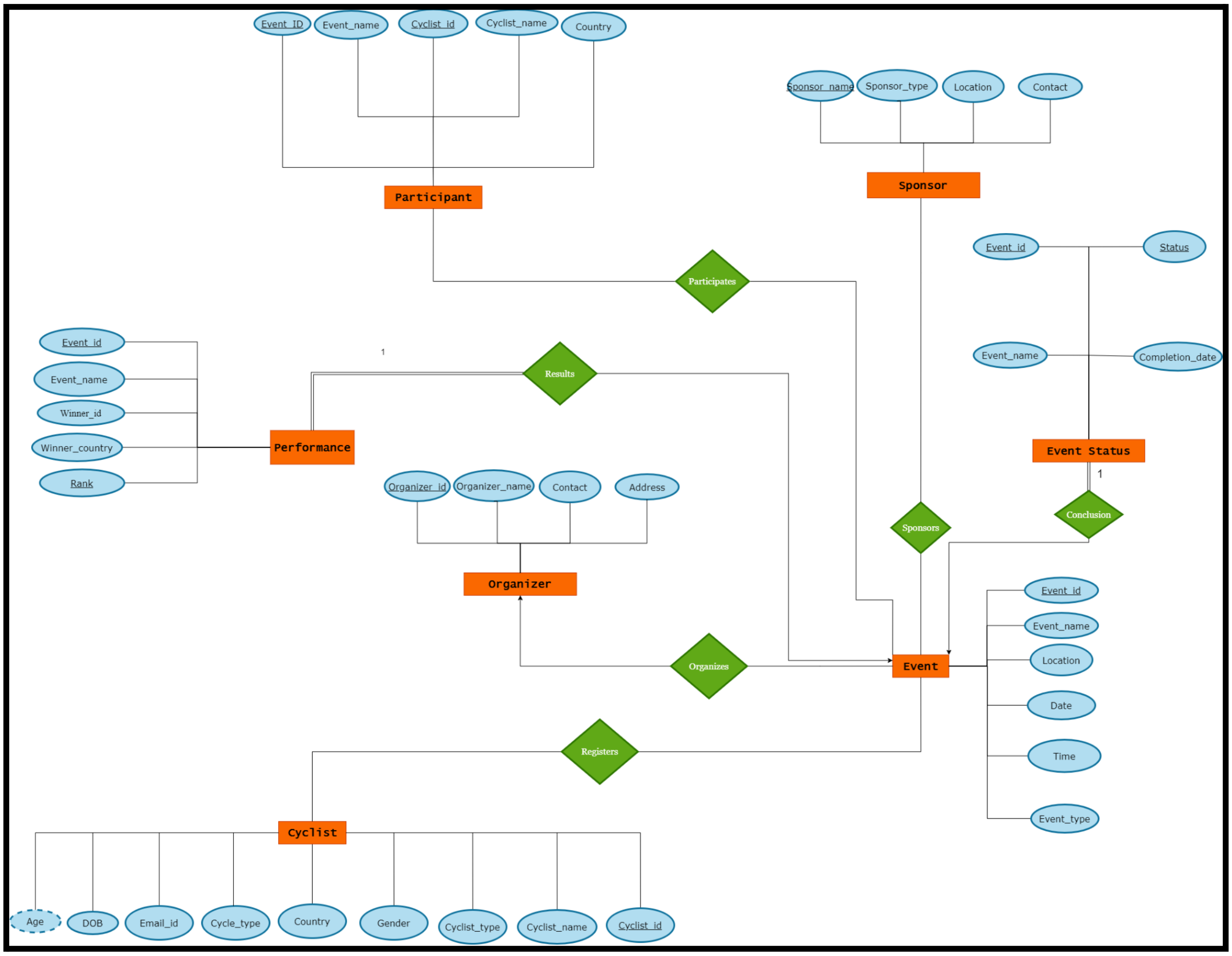
verb	Reject Reason
is	Duplicate,General
get	Duplicate
cycling	Irrelevant
be	General
avoid	Duplicate,Irrelevant
affects	Irrelevant
ensure	Irrelevant
are	General,Irrelevant
planning,implement	Irrelevant
consists	Duplicate,Irrelevant
includes	Duplicate,Irrelevant
identifies	Duplicate,Irrelevant
keep	Duplicate,General
organized	Duplicate
want	General,
design	Irrelevant
check	Irrelevant,General
plan	Irrelevant
based	Irrelevant

Be done	Irrelevant
stores	General
conducted	Irrelevant
has	General
related	Irrelevant
Inserting, deleting, updating	Irrelevant
Existing, login, check	Irrelevant
wish	General
Be assigned	Irrelevant
has	General
listed	Irrelevant

Section 3: Final E-R Diagram

1. ER Diagram Version-1
2. ER Diagram Version-2
3. ER Diagram Final Version
4. 8

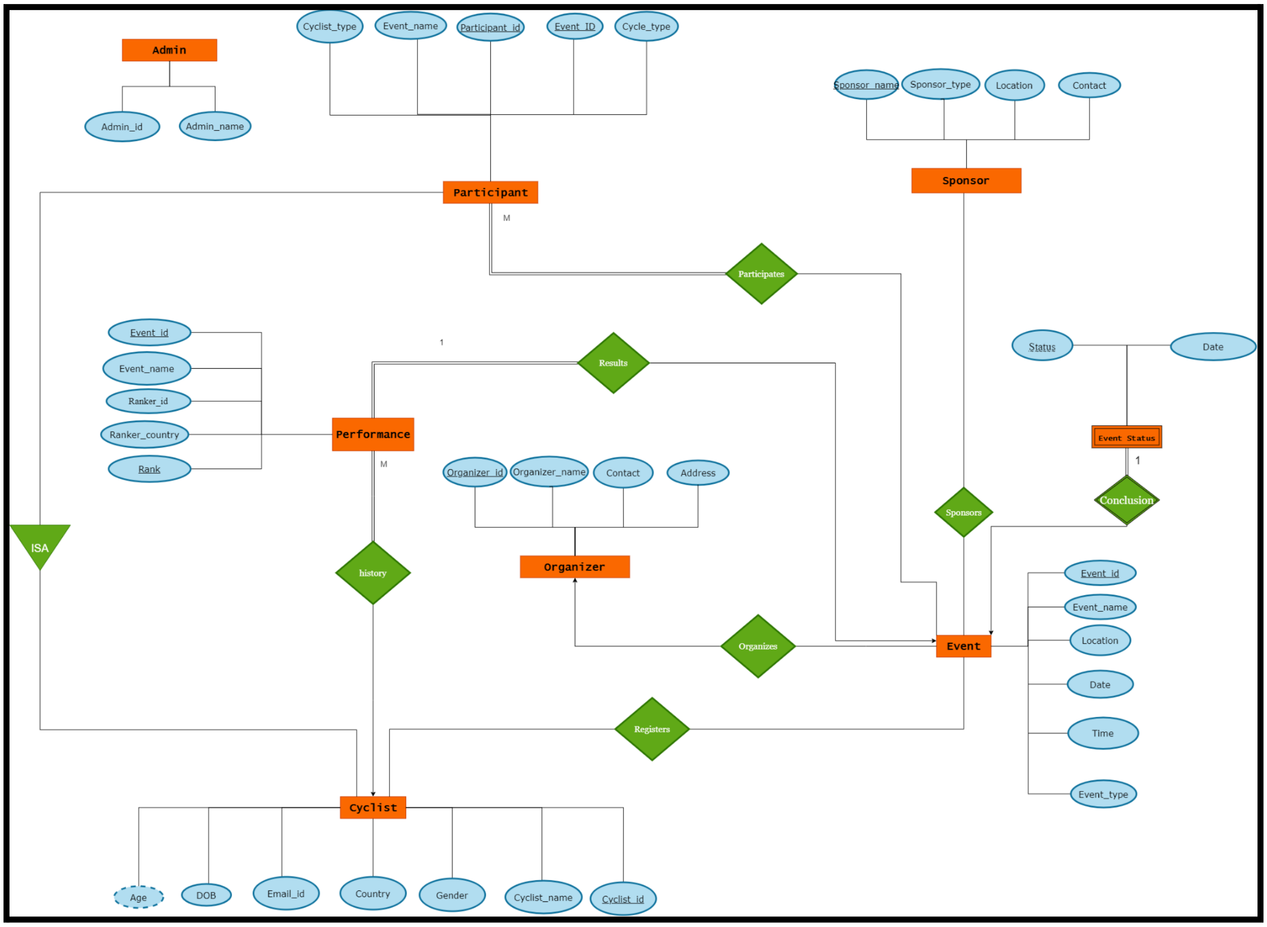
1. ER Diagram Version-1



➢For cardinality, we have indicated partial participation by (—) line for many (M) cardinality and by (→) for one (1) cardinality.

➢And for total participation, we have indicated 1 or M near the relationship links.

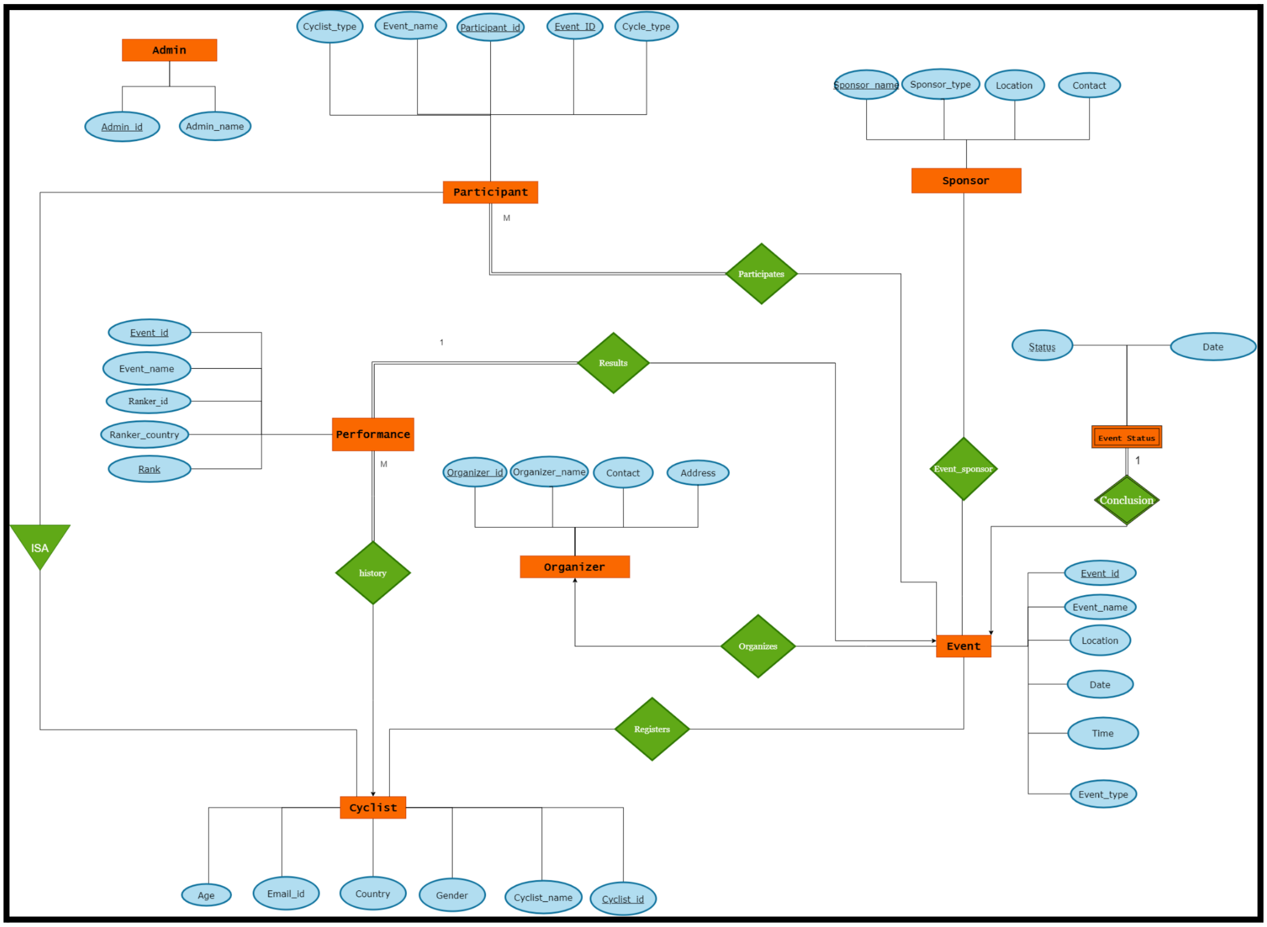
2. ER Diagram Version-2



➢For cardinality, we have indicated partial participation by (—) line for many (M) cardinality and by (→) for one (1) cardinality.

➢And for total participation, we have indicated 1 or M near the relationship links.

3. ER Diagram Final Version



- ➢For cardinality, we have indicated partial participation by (—) line for many (M) cardinality and by (→) for one (1) cardinality.
- ➢And for total participation, we have indicated 1 or M near the relationship links.

Section 4: E-R to Relational Mapping

1. Relational Schemas

I. Cyclist

(**Cyclist_id,** Cyclist_name, Age, Country, Gender, Email_id)

- **Explanation:**

➢ Here, the cyclist is a strong entity so we have to create a table for it.
It has attributes Cyclist_id**,** Cyclist_name, Age, Country, Gender, Email_id
Primary Key: Cyclist_id
Constraints: PK cannot be NULL
Check if Age > 0
Check if Gender is M or F and not any other.

II. Event

(**Event_id,** Event_name, Location, Date, Time, Event_type, Organizer_id)

- **Explanation:**

➢ Here, the event is a strong entity so we have to create a table for it.
As we have the One-to-many relation with the Organizer relation we need to add Foreign Key for reference.
It has attributes Event_id**,** Event_name, Location, Date, Time, Event_type, Organizer_id
Primary Key: Event_id
Foreign Key: Organizer_id references to Organizer
Constraints: PK cannot be NULL
Date cannot be NULL
Time cannot be NULL
Referential integrity constraint:

➢ Here FK is Organizer_id so on update to any tuple in Organizer relation,we have to update tuples corresponding to that updated tuple in referencing relation. So we need to use CASCADE on update

- ➢ Similarly, for delete operation we need not to delete the tuple in referencing relation so we need to use SET NULL on delete.

III. Organizer

(**Organizer_id**, Organizer_name, Contact, Address)

- **Explanation:**
- ➢ Here, the Organizer is a strong entity so we have to create a table for it.
  As we have the One-to-many relation with the Event relation we need to use Primary Key of this relation as Foreign Key for reference.
  It has attributes Organizer_name,Organizer_id, Contact, Address
  Primary Key: Organizer_id
  Constraints: PK cannot be NULL
  Check if length(contact) is 10

IV. Sponsor

(**Sponsor_name,** sponsor_type, Location,Contact)

- **Explanation:**
- ➢ Here, the Sponsor is a strong entity so we have to create a table for it.
  It has attributes Sponsor_name, sponsor_type, Location,Contact
  Primary Key : Sponosor_name
  Constraint: PK cannot be NULL

V. Performance

(**Event_id,Rank** **,**Event_name, Cyclist_id,Cyclist_country)

- **Explanation:**
  Here, the Performance is a strong entity so we have to create a table for it.
  It has attributes Event_id,Rank ,Event_name, Ranker_id,Ranker_conutry

  Primary Key : Event_id , Rank
  Here Event_id comes multiple times in the table so we have to add a rank attribute to make Primary Key.

  Foreign Key: Event_id,Cyclist_id

Domain Constraints: PK cannot be NULL
Cyclist_id cannot be NULL
Check if Rank ranges from 1 to 3.

Referential Constraints:

- ➢ Here FK is Event_id so on update to any tuple in Event relation,we have to update tuples corresponding to that updated tuple in referencing relation. So we need to use CASCADE on update
- ➢ Similarly, for delete operation we need to delete the tuple in referencing relation so we need to use CASCADE on delete

VI. Participant

(**<u>Event_id,Cyclist_id</u>,** Event_name, Cyclist_type,Cycle_type)

- **Explanation:**

Primary Key : Event_id , Cyclist_id
Here Event_id comes multiple times in the table so we have to add a participant_id attribute to make Primary Key.

Foreign Key:
FK Event_id referencing to Event table

- ➢ Here FK is Event_id so on update to any tuple in Event relation,we have to update tuples corresponding to that updated tuple in referencing relation. So we need to use CASCADE on update
- ➢ Similarly, for delete operation we need to delete the tuple in referencing relation so we need to use CASCADE on delete

VII. Event_status

(**<u>Event_id ,Status</u>**, Date)

- **Explanation:**

  Primary Key : Event_id , Status
  Here Event_id comes multiple times in the table so we have to add a participant_id attribute to make Primary Key.

  Foreign Key:
  FK Event_id referencing to Event table

  Domain Constraints:
  We have used here check constraints for **Status** and values always come from canceled , rescheduled and completed.
  referential constraint:

➢ Here FK is Event_id so on update to any tuple in Event relation,we have to update tuples corresponding to that updated tuple in referencing relation. So we need to use CASCADE on update

➢ Similarly, for delete operation we need to delete the tuple in referencing relation so we need to use CASCADE on delete

VIII. Event_sponsor

(<u>Event_id, Sponsor_name</u>, Sponsorship_amount)

- FK Event_id references to **Event**
- FK Sponsor_name references to **Sponsor**

- **Explanation:**
  We have a Many-to-many relation between Event and Sponsor so we need to make another relation named Event_sponsor.
  Primary Key : Event_id , Sponsor_name
  Foreign Key: Event_id , Sponsor_name
  FK Event_id referencing to Event table

- ➢ Here FK is Event_id so on update to any tuple in Event relation,we have to update tuples corresponding to that updated tuple in referencing relation. So we need to use CASCADE on update
- ➢ Similarly, for delete operation we need to delete the tuple in referencing relation so we need to use CASCADE on delete
- ➢ Similar for FK Sponsor_name

IX. Admin

(<u>Admin_id</u> , Admin_name)

- **Explanation:**

Primary Key : Admin_id

Domain Constraints:

Admin_id constraints **NOT NULL**

Section 5: Normalization

1. Normalization & Schema Refinement

1) **Cyclist**(**<u>Cyclist_id</u>,** Cyclist_name, Age, Country, Gender, Email_id)

   Primary key: Cyclist_id
   Foreign Key: None
   Candidate Key: Email_id
   Prime attributes: Cyclist_id, Email_id
   Non-prime attributes: Cyclist_name, Age, Country, Gender

   Functional dependency:
   - Cyclist_id → (Cyclist_name, Age, Country, Gender, Email_id)
   - Email_id → (Cyclist_id, Cyclist_name, Age, Country, Gender)

   Redundancies:
   - None

   Anomalies:
   - Insertion: None
   - Deletion/Updation: We cannot delete or update a tuple in which Cyclist_id is used as a foreign_key in Participant (referencing)relation.

   Normalization:
   - **1NF:** Here, all attributes are atomic so it is in 1NF.
     (We considered a cyclist to have only one email_id.)
   - **2NF:** Here, there is no partial dependency so it is in 2NF.
   - **3NF:** Here, there is no transitive dependency so it is in 3NF.
   - **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

2) **Event(<u>Event_id</u>,** Event_name, Location, Date, Time, Event_type, Organizer_id)

Primary_key: Event_id
Foreign key: Organizer_id
Candidate_key: None
Prime attributes: Event_id
Non-prime attributes: Event_name, Location, Date, Time, Event_type, Organizer_id

Functional dependency:

- Event_id → (Event_name, Location, Date, Time, Event_type, Organizer_id)

Redundancies:

- None

Anomalies:

- **Insertion:** We cannot insert a tuple in which Organizer_id is such that it is not present in Organizer (referenced)relation.

- **Deletion/Updation:** We cannot delete or update a tuple in which Event_id is used as a foreign_key in referencing relation.

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

3) **Organizer**(**Organizer_id**, Organizer_name, Contact, Address)

Primary_key: Organizer_id
Foreign key: None
Candidate_key: Contact
Prime attributes: Organizer_id,Contact
Non-prime attributes: Organizer_name, Address

Functional dependency:

- Organizer_id → (Organizer_name, Contact, Address)
- Contact → (Organizer_id, Organizer_name, Address)

Redundancies:

- None

Anomalies:

- **Insertion:** None

- **Deletion/Updation:** We cannot delete or update a tuple in which Organizer_id is used as a foreign_key in Event (referencing)relation.

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.
  (We considered an organizer to have only one Contact and here we considered Address as single string)

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

4) **Sponsor**(**<u>Sponsor_name</u>,** Sponsor_type, Location,Contact)

Primary key: Sponsor_name
Foreign Key: None
Candidate Key: Contact
Prime attributes: Sponsor_name
Non-prime attributes: Sponsor_type, Location,Contact

Functional dependency:

- Sponsor_name → (Sponsor_type, Location,Contact)
- Contact → (Sponsor_name**,** Sponsor_type, Location)

Redundancies:

- None

Anomalies:

- **Insertion:** None

- **Deletion/Updation:** We cannot delete or update a tuple in which Sponsor_name is used as a foreign_key in Event_sponsor (referencing)relation.

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.
  (We considered a sponsor to have only one Contact.)

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

5) **Performance(<u>Event_id,Rank</u> ,**Event_name, Cyclist_id,Cyclist_country)

Primary key: Event_id, Rank
Foreign Key: Event_id
Candidate Key: Event_id,Cyclist_id
Prime attributes: Event_id, Rank, Cyclist_id
Non-Prime attributes: Event_name, Cyclist_country

Functional Dependencies:

- Event_id,Rank → **(**Event_name, Cyclist_id,Cyclist_country)
- Event_id,Cyclist_id → **(**Event_name,Rank,Cyclist_country)
- Cyclist_id → Cyclist_country
- Event_id → Event_name

Anomalies:

- **Insertion:** We cannot insert a tuple in which Event_id is such that it is not present in Event (referenced)relation.

- **Deletion/Updation:** None

Redundancies:

- **None**

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is partial dependency so it is not in 2NF.

➢ So, we decompose the performance table as :

- Cyclist_performance(**<u>Event_id,Rank,Event_name</u>**,Cyclist_id)
  - ➢ It is in 3NF.
- Country_performance(**<u>Event_id,Rank</u>**,Cyclist_country)
  - ➢ It is in BCNF.

6) **Participant(<u>Event_id,Cyclist_id,Event_name</u>**, Cyclist_type,Cycle_type)

Primary key: Event_id, Cyclist_id, Event_name
Foreign Key: Event_id,Cyclist_id
Candidate Key: None
Prime attributes: Event_id, Cyclist_id, Event_name
Non-prime attributes: Cyclist_type,Cycle_type

Functional dependency:

- Event_id,Cyclist_id → (Event_name, Cyclist_type,Cycle_type)
- Event_id → Event_name

Redundancies:

- None

Anomalies:

- **Insertion:** We cannot insert a tuple in which Event_id or Cyclist_id is such that it is not present in Event or Cyclist (referenced)relation.

- **Deletion/Updation:** None

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here LHS of a FD is not a superkey so it is in 3NF.

➢ **So, this relation is in 3NF.**

7) **Event_status(<u>Event_id ,Status</u>**, Date)

Primary key: Event_id, Status
Foreign Key: Event_id
Candidate Key: None
Prime attributes: Event_id, Status
Non-Prime attributes: Date

Functional dependency:

- Event_id,Status → Date

Redundancies:

Anomalies:

- **Insertion:** None

- **Deletion/Updation:** We cannot delete or update a tuple in which Organizer_id is used as a foreign_key in Event (referencing)relation.

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

8) **Event_sponsor(<u>Event_id, Sponsor_name</u>,** Sponsorship_amount)

Primary key: Event_id, Sponsor_name
Foreign Key: Event_id, Sponsor_name
Candidate Key: None
Prime attributes: Event_id, Sponsor_name
Non-Prime attributes: Sponsorship_amount

Functional Dependencies:

- Event_id, Sponsor_name → Sponsorship_amount

Anomalies:

- **Insertion:** We cannot insert a tuple in which Event_id or Sponsor_name is such that it is not present in Event or Sponsor (referenced)relation.

- **Deletion/Updation:** None

Redundancies:

- None

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

9) **Admin(<u>Admin_id</u>** , Admin_name)

Primary key: Admin_id
Foreign Key: None
Candidate Key: None
Prime attributes: Admin_id
Non-Prime attributes: Admin_name

Functional Dependencies:

- Admin_id → Admin_name

Anomalies:

- **Insertion:** None

- **Deletion/Updation:** None

Redundancies:

- None

Normalization:

- **1NF:** Here, all attributes are atomic so it is in 1NF.

- **2NF:** Here, there is no partial dependency so it is in 2NF.

- **3NF:** Here, there is no transitive dependency so it is in 3NF.

- **BCNF:** Here, every FD has a superkey in LHS so it is in BCNF.

➢ **So, this relation is in BCNF.**

Normal forms before schema refinement based ER diagram:

Table	Normal form (before refinement)
Cyclist	BCNF
Event	BCNF
Organizer	BCNF
Sponsor	BCNF
Performance	1NF
Participant	3NF
Event_status	BCNF
Event_sponsor	BCNF
Admin	BCNF

Normal forms after schema refinement based ER diagram:

Table	Normal form (after refinement)
Cyclist	BCNF
Event	BCNF
Organizer	BCNF
Sponsor	BCNF
Cyclist_performance	3NF
Country_performance	BCNF
Participant	3NF
Event_status	BCNF
Event_sponsor	BCNF
Admin	BCNF

Section 6: DDL Scripts & SQL Queries with Snapshots

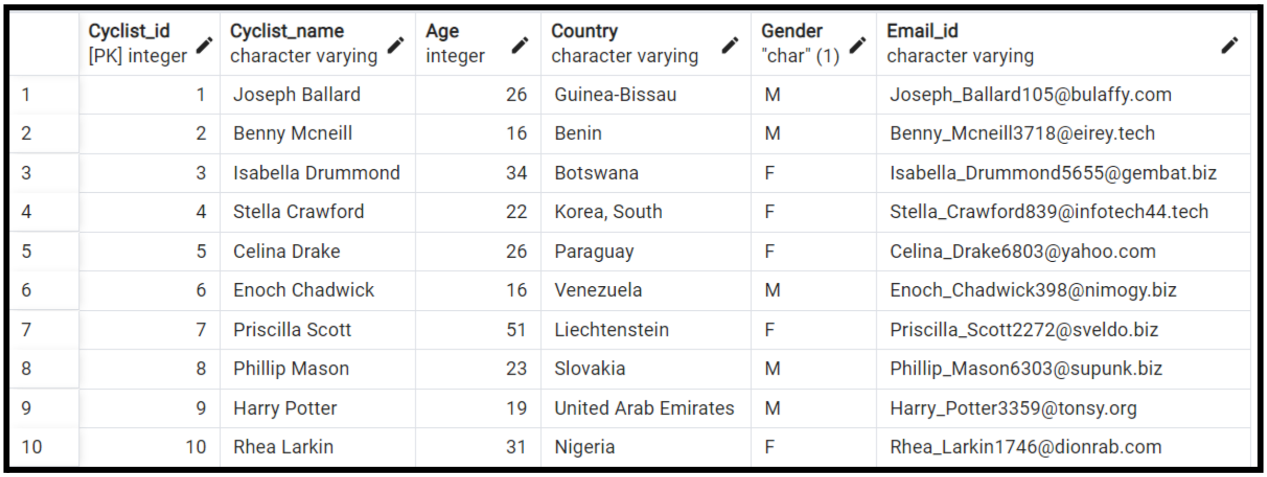
1. DDL Scripts with Snapshots of tables
2. SQL Queries with Snapshots

1. DDL Scripts with Snapshots of tables

1) **Cyclist** (**Cyclist_id**, Cyclist_name, Age, Country, Gender, Email_id)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Cyclist"
(
    "Cyclist_id" integer NOT NULL,
    "Cyclist_name" character varying COLLATE pg_catalog."default",
    "Age" integer NOT NULL,
    "Country" character varying COLLATE pg_catalog."default",
    "Gender" "char",
    "Email_id" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Cyclist_pkey" PRIMARY KEY ("Cyclist_id"),
    CONSTRAINT "Cyclist_Age_check" CHECK ("Age" > 0),
    CONSTRAINT "Cyclist_Gender_check" CHECK ("Gender" = ANY (ARRAY['M'::"char",
'F'::"char"]))
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Cyclist"
    OWNER to postgres;
```

- **After inserting data:(Total tuple=50)**

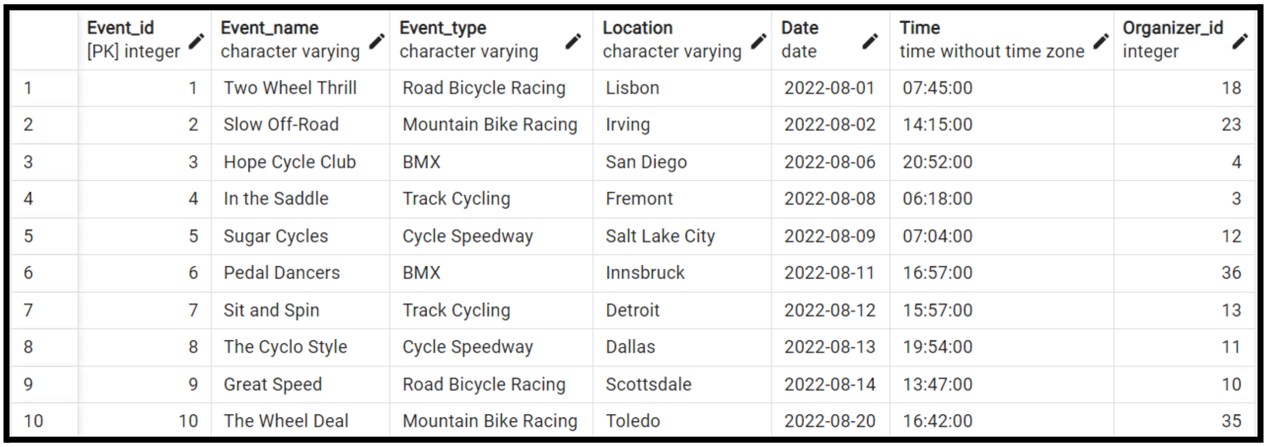


	Cyclist_id [PK] integer	Cyclist_name character varying	Age integer	Country character varying	Gender "char" (1)	Email_id character varying
1	1	Joseph Ballard	26	Guinea-Bissau	M	Joseph_Ballard105@bulaffy.com
2	2	Benny Mcneill	16	Benin	M	Benny_Mcneill3718@eirey.tech
3	3	Isabella Drummond	34	Botswana	F	Isabella_Drummond5655@gembat.biz
4	4	Stella Crawford	22	Korea, South	F	Stella_Crawford839@infotech44.tech
5	5	Celina Drake	26	Paraguay	F	Celina_Drake6803@yahoo.com
6	6	Enoch Chadwick	16	Venezuela	M	Enoch_Chadwick398@nimogy.biz
7	7	Priscilla Scott	51	Liechtenstein	F	Priscilla_Scott2272@sveldo.biz
8	8	Phillip Mason	23	Slovakia	M	Phillip_Mason6303@supunk.biz
9	9	Harry Potter	19	United Arab Emirates	M	Harry_Potter3359@tonsy.org
10	10	Rhea Larkin	31	Nigeria	F	Rhea_Larkin1746@dionrab.com

2) **Event(Event_id,** Event_name, Event_type, Location, Date, Time, Organizer_id)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Event"
(
   "Event_id" integer NOT NULL,
   "Event_name" character varying COLLATE pg_catalog."default",
   "Event_type" character varying COLLATE pg_catalog."default",
   "Location" character varying COLLATE pg_catalog."default" NOT NULL,
   "Date" date NOT NULL,
   "Time" time without time zone NOT NULL,
   "Organizer_id" integer,
   CONSTRAINT "Event_pkey" PRIMARY KEY ("Event_id"),
   CONSTRAINT "Event_Organizer_id_fkey" FOREIGN KEY ("Organizer_id")
      REFERENCES cyclist_db."Organizer" ("Organizer_id") MATCH SIMPLE
      ON UPDATE CASCADE
      ON DELETE SET NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Event"
   OWNER to postgres;
```

- **After inserting data:(Total tuple=50)**

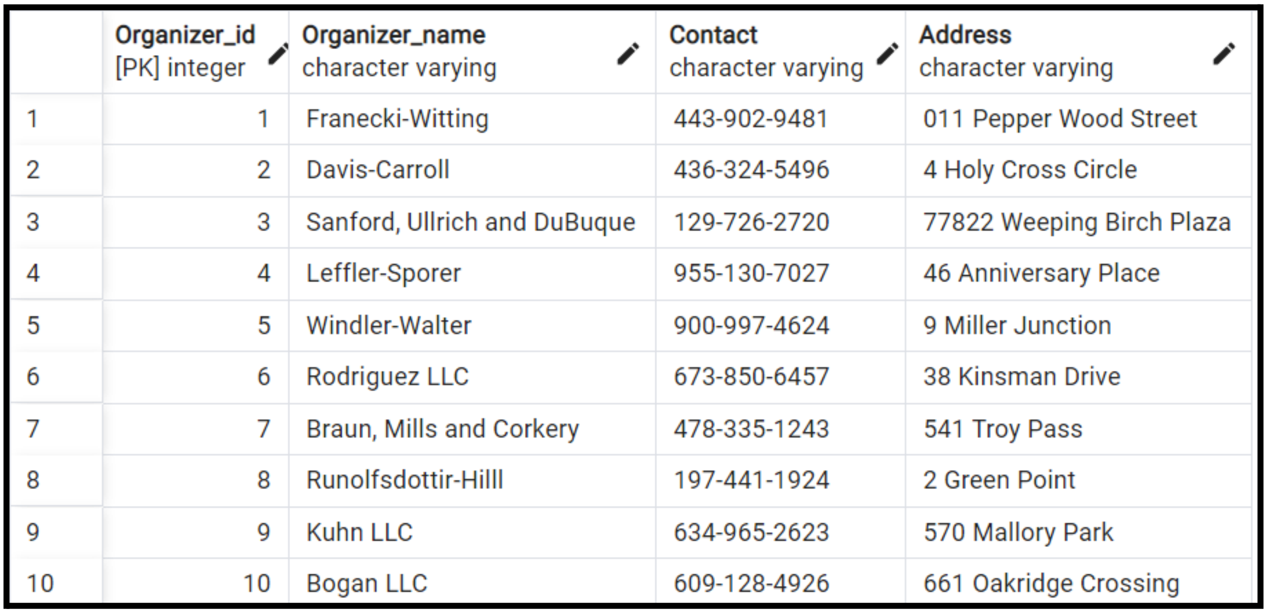


	Event_id [PK] integer	Event_name character varying	Event_type character varying	Location character varying	Date date	Time time without time zone	Organizer_id integer
1	1	Two Wheel Thrill	Road Bicycle Racing	Lisbon	2022-08-01	07:45:00	18
2	2	Slow Off-Road	Mountain Bike Racing	Irving	2022-08-02	14:15:00	23
3	3	Hope Cycle Club	BMX	San Diego	2022-08-06	20:52:00	4
4	4	In the Saddle	Track Cycling	Fremont	2022-08-08	06:18:00	3
5	5	Sugar Cycles	Cycle Speedway	Salt Lake City	2022-08-09	07:04:00	12
6	6	Pedal Dancers	BMX	Innsbruck	2022-08-11	16:57:00	36
7	7	Sit and Spin	Track Cycling	Detroit	2022-08-12	15:57:00	13
8	8	The Cyclo Style	Cycle Speedway	Dallas	2022-08-13	19:54:00	11
9	9	Great Speed	Road Bicycle Racing	Scottsdale	2022-08-14	13:47:00	10
10	10	The Wheel Deal	Mountain Bike Racing	Toledo	2022-08-20	16:42:00	35

3) **Organizer**(**Organizer_id**, Organizer_name, Contact, Address)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Organizer"
(
    "Organizer_id" integer NOT NULL,
    "Organizer_name" character varying COLLATE pg_catalog."default",
    "Contact" character varying COLLATE pg_catalog."default",
    "Address" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Organizer_pkey" PRIMARY KEY ("Organizer_id")
)

TABLESPACE pg_default;

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

- **After inserting data:(Total tuple=50)**

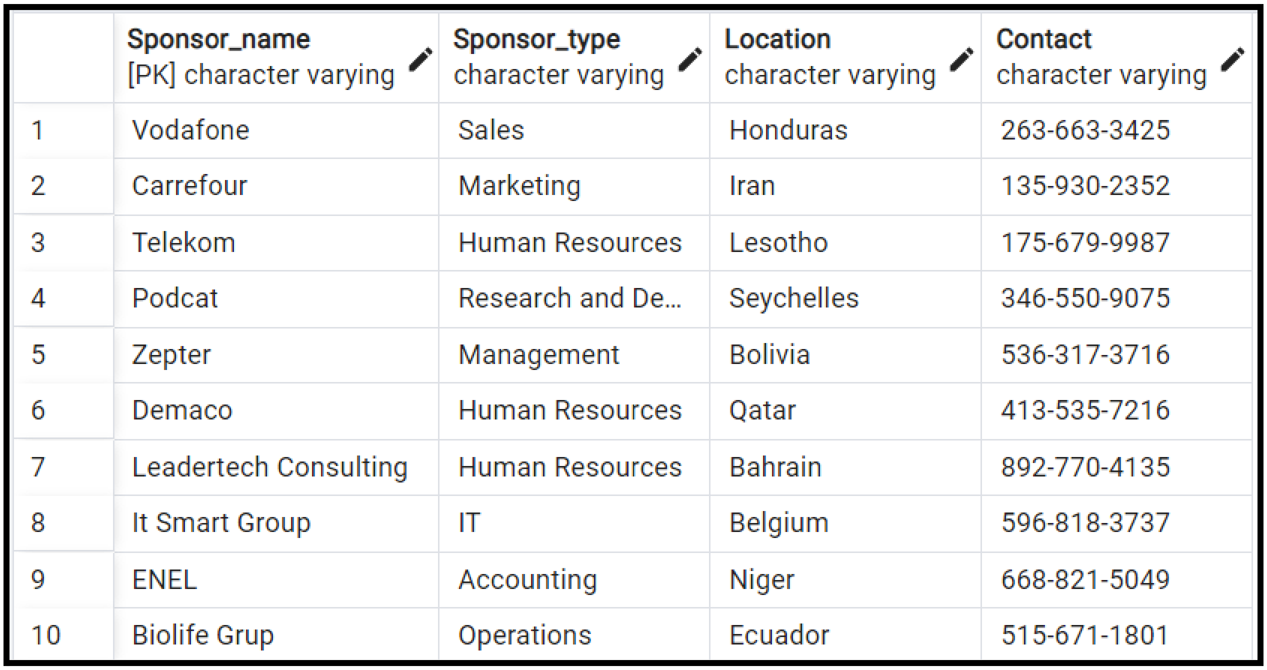


	Organizer_id [PK] integer	Organizer_name character varying	Contact character varying	Address character varying
1	1	Franecki-Witting	443-902-9481	011 Pepper Wood Street
2	2	Davis-Carroll	436-324-5496	4 Holy Cross Circle
3	3	Sanford, Ullrich and DuBuque	129-726-2720	77822 Weeping Birch Plaza
4	4	Leffler-Sporer	955-130-7027	46 Anniversary Place
5	5	Windler-Walter	900-997-4624	9 Miller Junction
6	6	Rodriguez LLC	673-850-6457	38 Kinsman Drive
7	7	Braun, Mills and Corkery	478-335-1243	541 Troy Pass
8	8	Runolfsdottir-Hilll	197-441-1924	2 Green Point
9	9	Kuhn LLC	634-965-2623	570 Mallory Park
10	10	Bogan LLC	609-128-4926	661 Oakridge Crossing

4) **Sponsor(Sponsor_name,** Sponsor_type, Location,Contact)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Sponsor"
(
    "Sponsor_name" character varying COLLATE pg_catalog."default" NOT NULL,
    "Sponsor_type" character varying COLLATE pg_catalog."default",
    "Location" character varying COLLATE pg_catalog."default",
    "Contact" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Sponsor_pkey" PRIMARY KEY ("Sponsor_name")
)

TABLESPACE pg_default;

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

- **After inserting data:(Total tuple=50)**

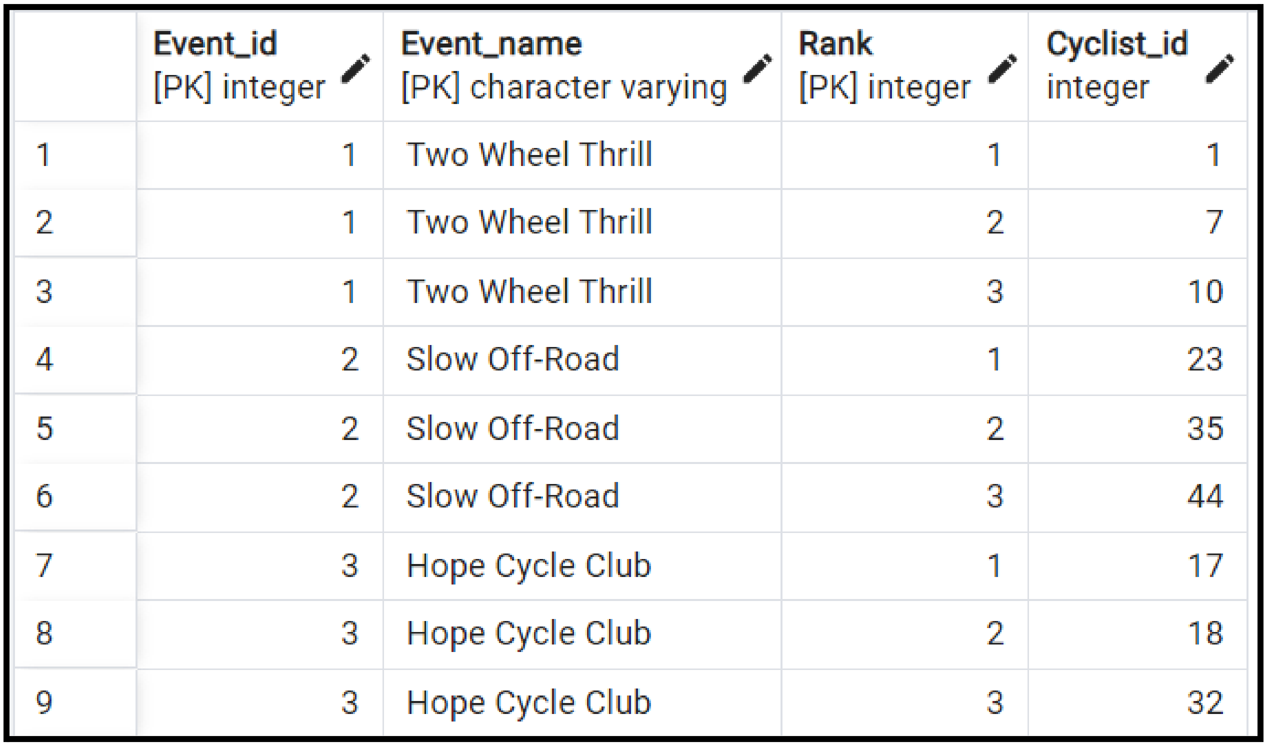


	Sponsor_name [PK] character varying	**Sponsor_type** character varying	**Location** character varying	**Contact** character varying
1	Vodafone	Sales	Honduras	263-663-3425
2	Carrefour	Marketing	Iran	135-930-2352
3	Telekom	Human Resources	Lesotho	175-679-9987
4	Podcat	Research and De...	Seychelles	346-550-9075
5	Zepter	Management	Bolivia	536-317-3716
6	Demaco	Human Resources	Qatar	413-535-7216
7	Leadertech Consulting	Human Resources	Bahrain	892-770-4135
8	It Smart Group	IT	Belgium	596-818-3737
9	ENEL	Accounting	Niger	668-821-5049
10	Biolife Grup	Operations	Ecuador	515-671-1801

5) Cyclist_performance(**Event_id, Event_name, Rank**,Cyclist_id)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Cyclist_performance"
(
    "Event_id" integer NOT NULL,
    "Event_name" character varying COLLATE pg_catalog."default" NOT NULL,
    "Rank" integer NOT NULL,
    "Cyclist_id" integer NOT NULL,
    CONSTRAINT "Cyclist_performance_pkey" PRIMARY KEY ("Event_id",
"Event_name", "Rank"),
    CONSTRAINT "Cyclist_performance_Cyclist_id_fkey" FOREIGN KEY ("Cyclist_id")
        REFERENCES cyclist_db."Cyclist" ("Cyclist_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Cyclist_performance_Event_id_fkey" FOREIGN KEY ("Event_id")
        REFERENCES cyclist_db."Event" ("Event_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Cyclist_performance_Rank_check" CHECK ("Rank" > 0 AND "Rank"
< 4)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Cyclist_performance"
    OWNER to postgres;
```

- **After inserting data:(Total tuple=30)**

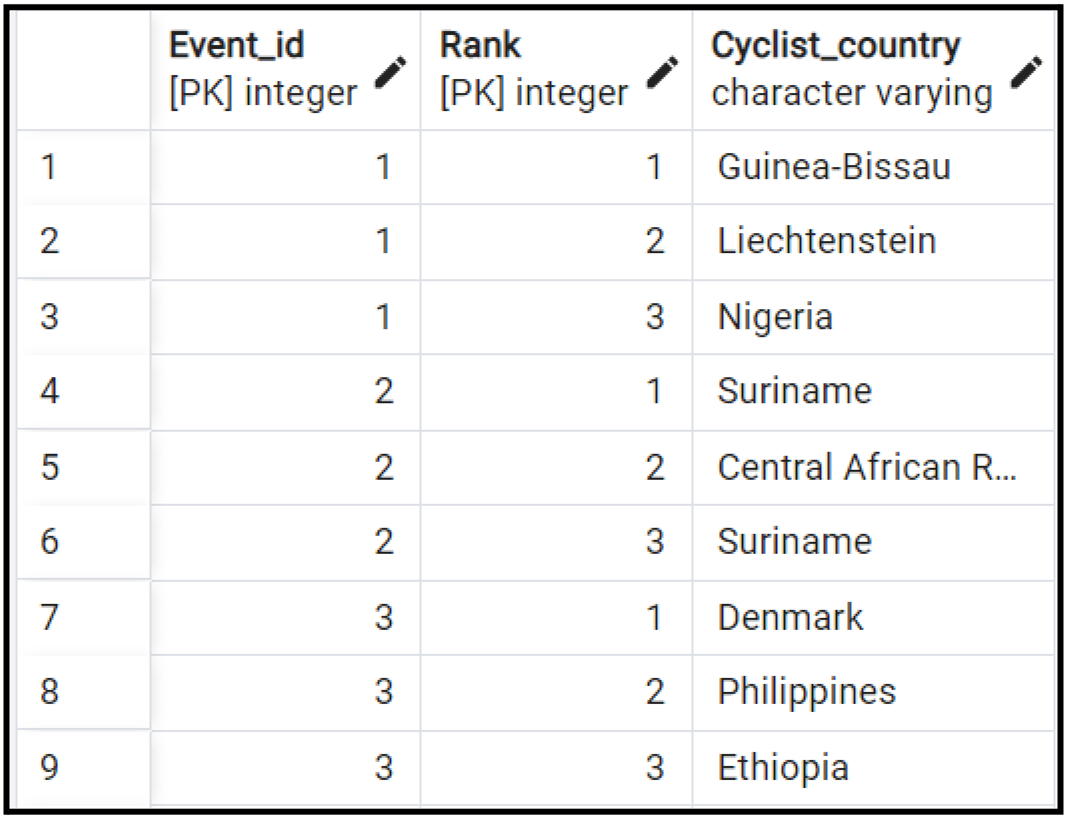


	Event_id [PK] integer	Event_name [PK] character varying	Rank [PK] integer	Cyclist_id integer
1	1	Two Wheel Thrill	1	1
2	1	Two Wheel Thrill	2	7
3	1	Two Wheel Thrill	3	10
4	2	Slow Off-Road	1	23
5	2	Slow Off-Road	2	35
6	2	Slow Off-Road	3	44
7	3	Hope Cycle Club	1	17
8	3	Hope Cycle Club	2	18
9	3	Hope Cycle Club	3	32

6) Country_performance(**Event_id,Rank**,Cyclist_country)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Country_performance"
(
    "Event_id" integer NOT NULL,
    "Rank" integer NOT NULL,
    "Cyclist_country" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Country_performance_pkey" PRIMARY KEY ("Event_id", "Rank"),
    CONSTRAINT "Country_performance_Event_id_fkey" FOREIGN KEY ("Event_id")
        REFERENCES cyclist_db."Event" ("Event_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Country_performance_Rank_check" CHECK ("Rank" > 0 AND "Rank"
< 4)
)

TABLESPACE pg_default;
ALTER TABLE IF EXISTS cyclist_db."Country_performance"
    OWNER to postgres;
```

- **After inserting data:(Total tuple=30)**

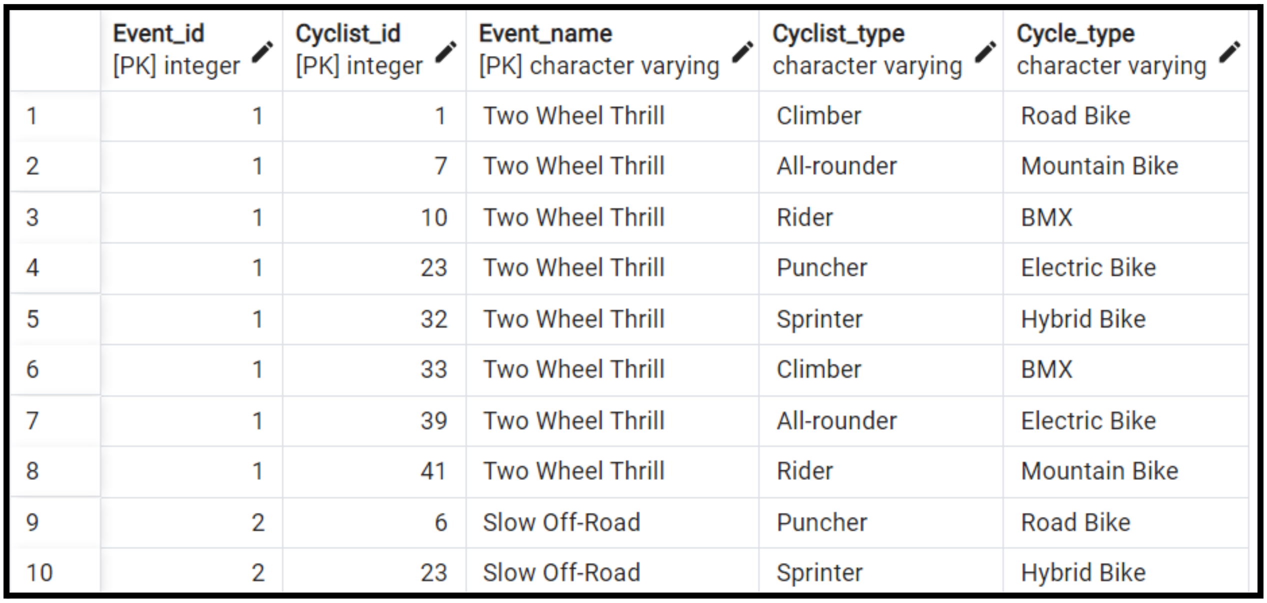


	Event_id [PK] integer	Rank [PK] integer	Cyclist_country character varying
1	1	1	Guinea-Bissau
2	1	2	Liechtenstein
3	1	3	Nigeria
4	2	1	Suriname
5	2	2	Central African R...
6	2	3	Suriname
7	3	1	Denmark
8	3	2	Philippines
9	3	3	Ethiopia

7) **Participant(Event_id,Cyclist_id, Event_name**, Cyclist_type,Cycle_type)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Participant"
(
   "Event_id" integer NOT NULL,
   "Cyclist_id" integer NOT NULL,
   "Event_name" character varying COLLATE pg_catalog."default" NOT NULL,
   "Cyclist_type" character varying COLLATE pg_catalog."default",
   "Cycle_type" character varying COLLATE pg_catalog."default",
   CONSTRAINT "Participant_pkey" PRIMARY KEY ("Event_id", "Cyclist_id",
"Event_name"),
   CONSTRAINT "Participant_Cyclist_id_fkey" FOREIGN KEY ("Cyclist_id")
      REFERENCES cyclist_db."Cyclist" ("Cyclist_id") MATCH SIMPLE
      ON UPDATE CASCADE
      ON DELETE CASCADE,
   CONSTRAINT "Participant_Event_id_fkey" FOREIGN KEY ("Event_id")
      REFERENCES cyclist_db."Event" ("Event_id") MATCH SIMPLE
      ON UPDATE CASCADE
      ON DELETE CASCADE
)

TABLESPACE pg_default;

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

- **After inserting data:(Total tuple=40)**

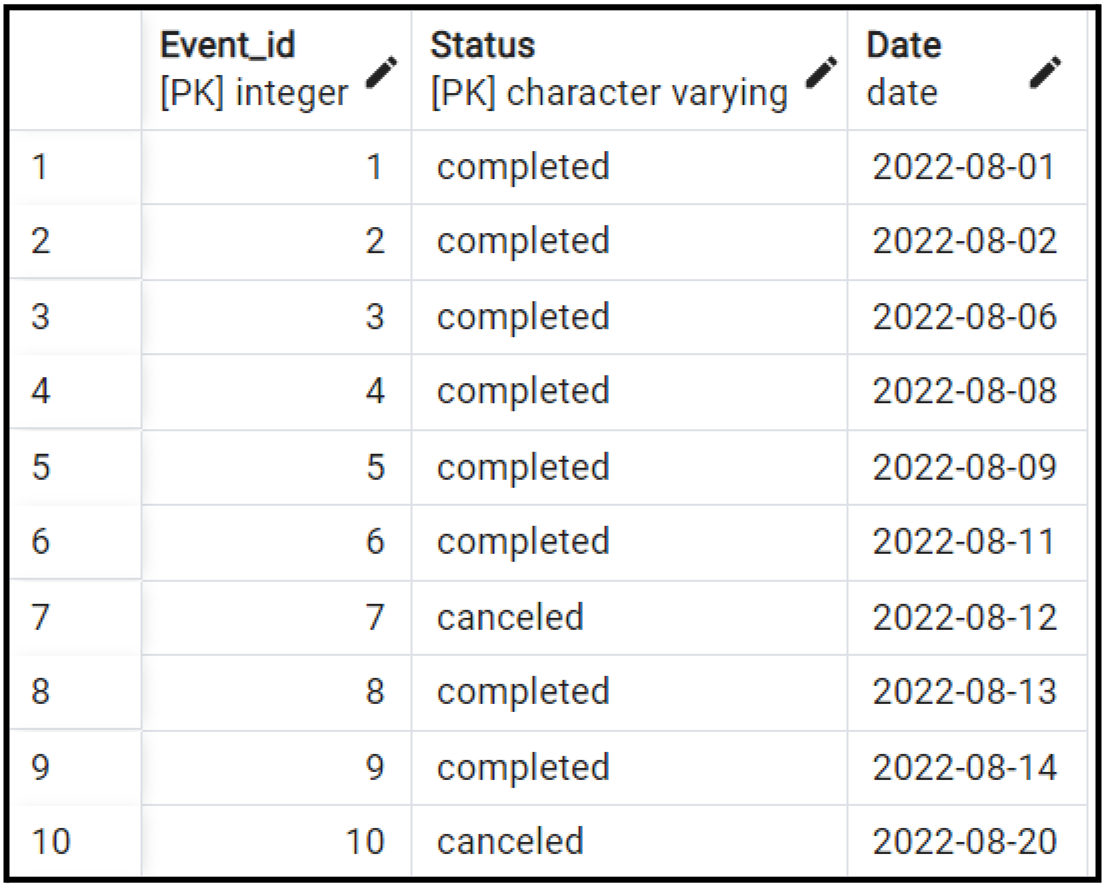


	Event_id [PK] integer	Cyclist_id [PK] integer	Event_name [PK] character varying	Cyclist_type character varying	Cycle_type character varying
1	1	1	Two Wheel Thrill	Climber	Road Bike
2	1	7	Two Wheel Thrill	All-rounder	Mountain Bike
3	1	10	Two Wheel Thrill	Rider	BMX
4	1	23	Two Wheel Thrill	Puncher	Electric Bike
5	1	32	Two Wheel Thrill	Sprinter	Hybrid Bike
6	1	33	Two Wheel Thrill	Climber	BMX
7	1	39	Two Wheel Thrill	All-rounder	Electric Bike
8	1	41	Two Wheel Thrill	Rider	Mountain Bike
9	2	6	Slow Off-Road	Puncher	Road Bike
10	2	23	Slow Off-Road	Sprinter	Hybrid Bike

8) **Event_status(Event_id ,Status**, Date)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Event_status"
(
    "Event_id" integer NOT NULL,
    "Status" character varying COLLATE pg_catalog."default" NOT NULL,
    "Date" date NOT NULL,
    CONSTRAINT "Event_status_pkey" PRIMARY KEY ("Event_id", "Status"),
    CONSTRAINT "Event_status_Event_id_fkey" FOREIGN KEY ("Event_id")
        REFERENCES cyclist_db."Event" ("Event_id") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Event_status_Status_check"
            CHECK ("Status"::text = ANY (ARRAY['canceled'::character varying::text,
                                                 'rescheduled'::character varying::text,
                                                 'completed'::character varying::text]))
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Event_status"
    OWNER to postgres;
```

- **After inserting data:(Total tuple=40)**

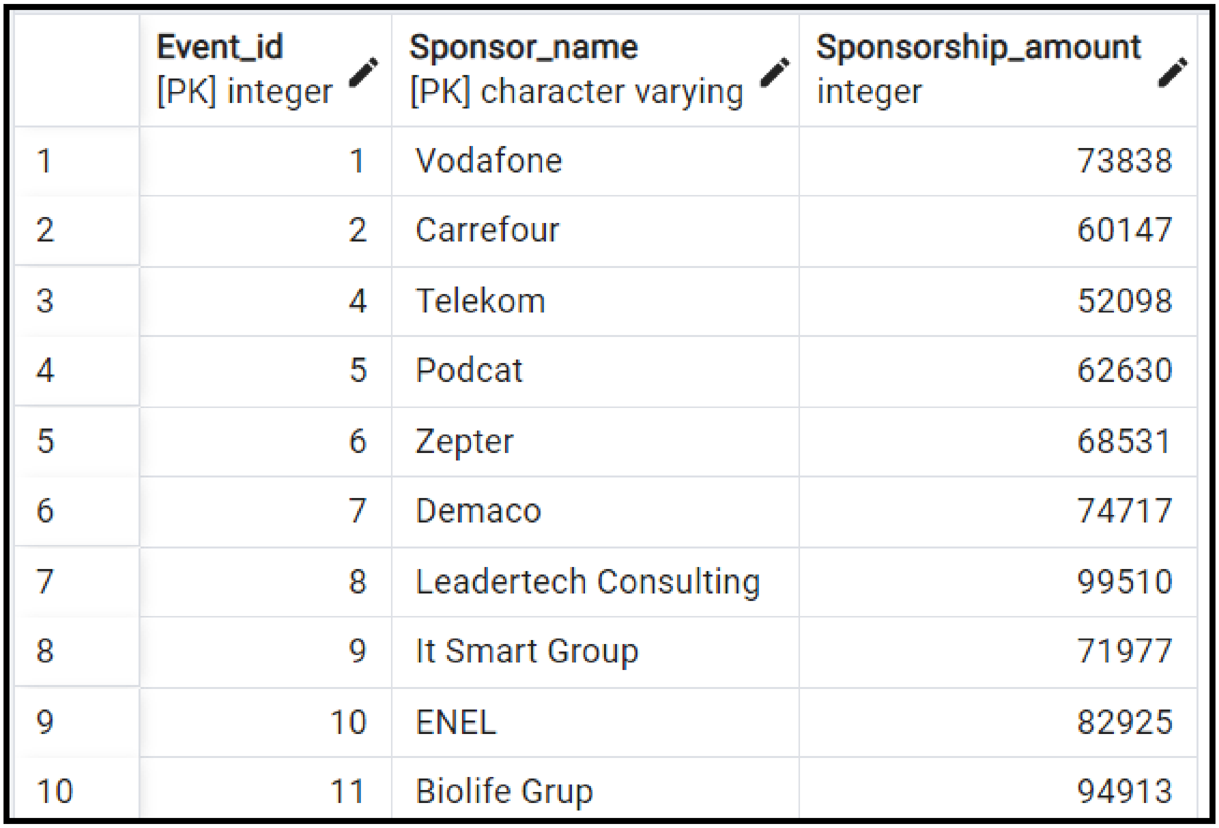


	Event_id [PK] integer	Status [PK] character varying	Date date
1	1	completed	2022-08-01
2	2	completed	2022-08-02
3	3	completed	2022-08-06
4	4	completed	2022-08-08
5	5	completed	2022-08-09
6	6	completed	2022-08-11
7	7	canceled	2022-08-12
8	8	completed	2022-08-13
9	9	completed	2022-08-14
10	10	canceled	2022-08-20

9) **Event_sponsor(Event_id, Sponsor_name,** Sponsorship_amount)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Event_sponsor"
(
   "Event_id" integer NOT NULL,
   "Sponsor_name" character varying COLLATE pg_catalog."default" NOT NULL,
   "Sponsorship_amount" integer,
   CONSTRAINT "Event_sponsor_pkey" PRIMARY KEY ("Event_id", "Sponsor_name"),
   CONSTRAINT "Event_sponsor_Event_id_fkey" FOREIGN KEY ("Event_id")
      REFERENCES cyclist_db."Event" ("Event_id") MATCH SIMPLE
      ON UPDATE CASCADE
      ON DELETE CASCADE,
   CONSTRAINT "Event_sponsor_Sponsor_name_fkey" FOREIGN KEY
("Sponsor_name")
      REFERENCES cyclist_db."Sponsor" ("Sponsor_name") MATCH SIMPLE
      ON UPDATE CASCADE
      ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Event_sponsor"
   OWNER to postgres;
```

- **After inserting data:(Total tuple=40)**

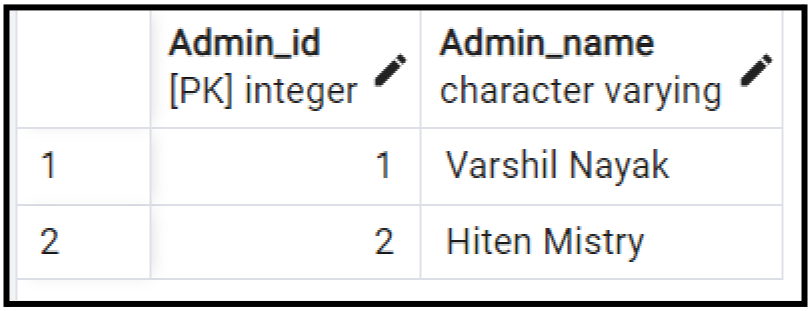


	Event_id [PK] integer	Sponsor_name [PK] character varying	Sponsorship_amount integer
1	1	Vodafone	73838
2	2	Carrefour	60147
3	4	Telekom	52098
4	5	Podcat	62630
5	6	Zepter	68531
6	7	Demaco	74717
7	8	Leadertech Consulting	99510
8	9	It Smart Group	71977
9	10	ENEL	82925
10	11	Biolife Grup	94913

10) Admin(Admin_id , Admin_name)

```
CREATE TABLE IF NOT EXISTS cyclist_db."Admin"
(
    "Admin_id" integer NOT NULL,
    "Admin_name" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Admin_pkey" PRIMARY KEY ("Admin_id")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS cyclist_db."Admin"
    OWNER to postgres;
```

- **After inserting data:(Total tuple=2)**

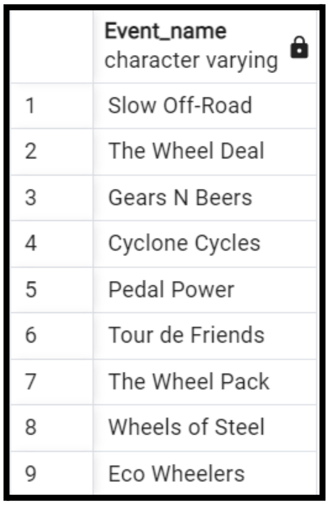


	Admin_id [PK] integer	**Admin_name** character varying
1	1	Varshil Nayak
2	2	Hiten Mistry

2. SQL Queries with Snapshots

1) **Display Event_name of type ‘Mountain Bike Racing’.**

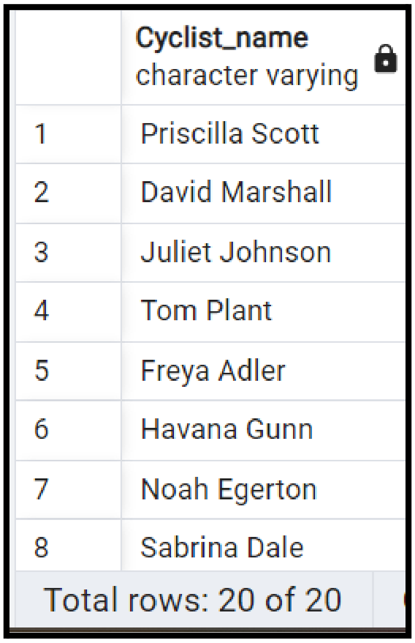
```
SELECT "Event_name"
FROM cyclist_db."Event"
WHERE "Event_type"='Mountain Bike Racing';
```



	Event_name character varying
1	Slow Off-Road
2	The Wheel Deal
3	Gears N Beers
4	Cyclone Cycles
5	Pedal Power
6	Tour de Friends
7	The Wheel Pack
8	Wheels of Steel
9	Eco Wheelers

2) **Display Cyclist_name of cyclists having ‘Age>40’.**

```
SELECT "Cyclist_name"
FROM cyclist_db."Cyclist"
WHERE "Age">40;
```

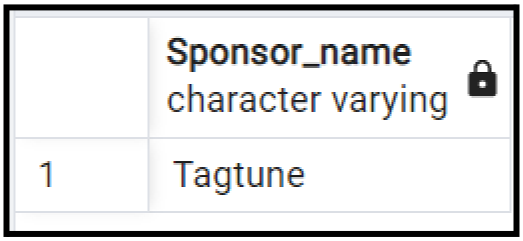


	Cyclist_name character varying
1	Priscilla Scott
2	David Marshall
3	Juliet Johnson
4	Tom Plant
5	Freya Adler
6	Havana Gunn
7	Noah Egerton
8	Sabrina Dale

Total rows: 20 of 20

3) **Display Sponsor_name of Sponsor with Maximum ‘Sponsorship_amount’.**

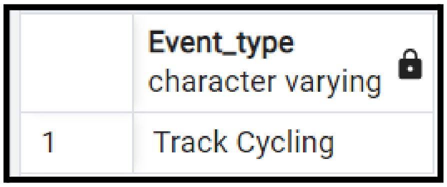
```
SELECT "Sponsor_name"
FROM cyclist_db."Event_sponsor"
ORDER BY "Sponsorship_amount" DESC LIMIT 1;
```



	Sponsor_name character varying
1	Tagtune

4) **Display ‘Event_type’ of Event that is organized mostly.**

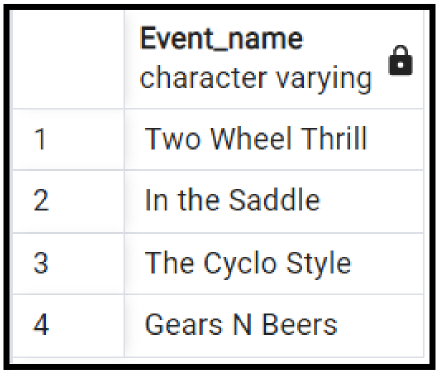
```
SELECT "Event_type"
FROM  cyclist_db."Event"
GROUP BY "Event_type"
ORDER BY COUNT("Event_type") DESC LIMIT 1;
```



	Event_type character varying
1	Track Cycling

5) **Display list of Events in which Cyclist_id=10 took part.**

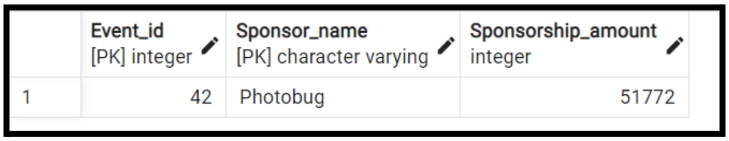
```
SELECT "Event_name"
FROM cyclist_db."Participant"
WHERE "Cyclist_id" = 10;
```



	Event_name character varying
1	Two Wheel Thrill
2	In the Saddle
3	The Cyclo Style
4	Gears N Beers

6) **Print the details of event and sponsor_name having lowest Sponsorship_amount**

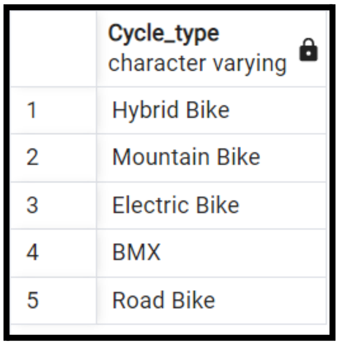
```
SELECT *
FROM cyclist_db."Event_sponsor"
ORDER BY "Sponsorship_amount" ASC LIMIT 1;
```



	Event_id [PK] integer	Sponsor_name [PK] character varying	Sponsorship_amount integer
1	42	Photobug	51772

7) **Display types of Cycles used by Cyclists.**

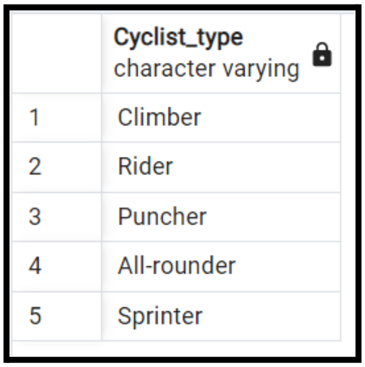
```
SELECT DISTINCT("Cycle_type")
FROM cyclist_db."Participant";
```



	Cycle_type character varying
1	Hybrid Bike
2	Mountain Bike
3	Electric Bike
4	BMX
5	Road Bike

8) **Display types of Cyclists who participated in any Event.**

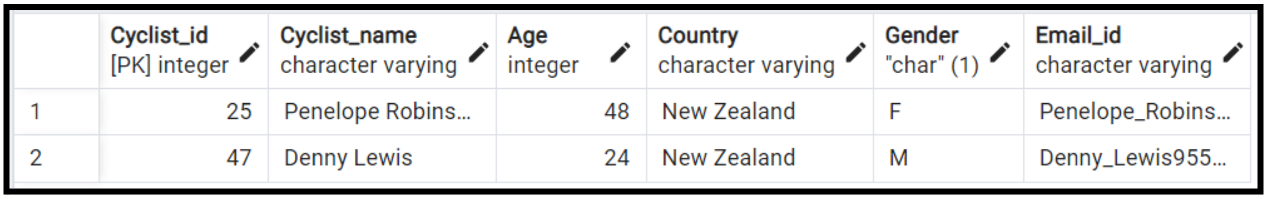
```
SELECT DISTINCT("Cyclist_type")
FROM cyclist_db."Participant";
```



	Cyclist_type character varying
1	Climber
2	Rider
3	Puncher
4	All-rounder
5	Sprinter

9) **Print details of cyclist_name who comes from New Zealand**

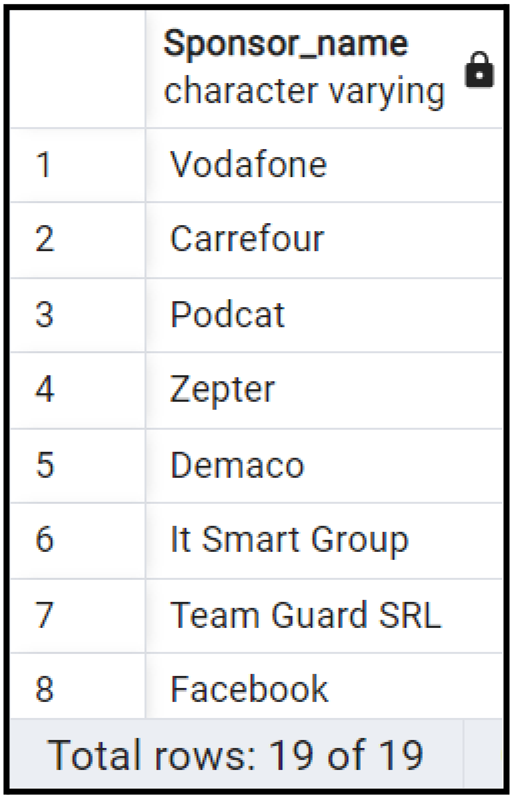
```
SELECT *
FROM cyclist_db."Cyclist"
WHERE "Country" = 'New Zealand';
```



	Cyclist_id [PK] integer	Cyclist_name character varying	Age integer	Country character varying	Gender "char" (1)	Email_id character varying
1	25	Penelope Robins...	48	New Zealand	F	Penelope_Robins...
2	47	Denny Lewis	24	New Zealand	M	Denny_Lewis955...

10) **Print the Sponsor_name who sponsored between 60000 and 80000.**

```
SELECT "Sponsor_name"
FROM cyclist_db."Event_sponsor"
WHERE "Sponsorship_amount" between 60000 and 80000;
```

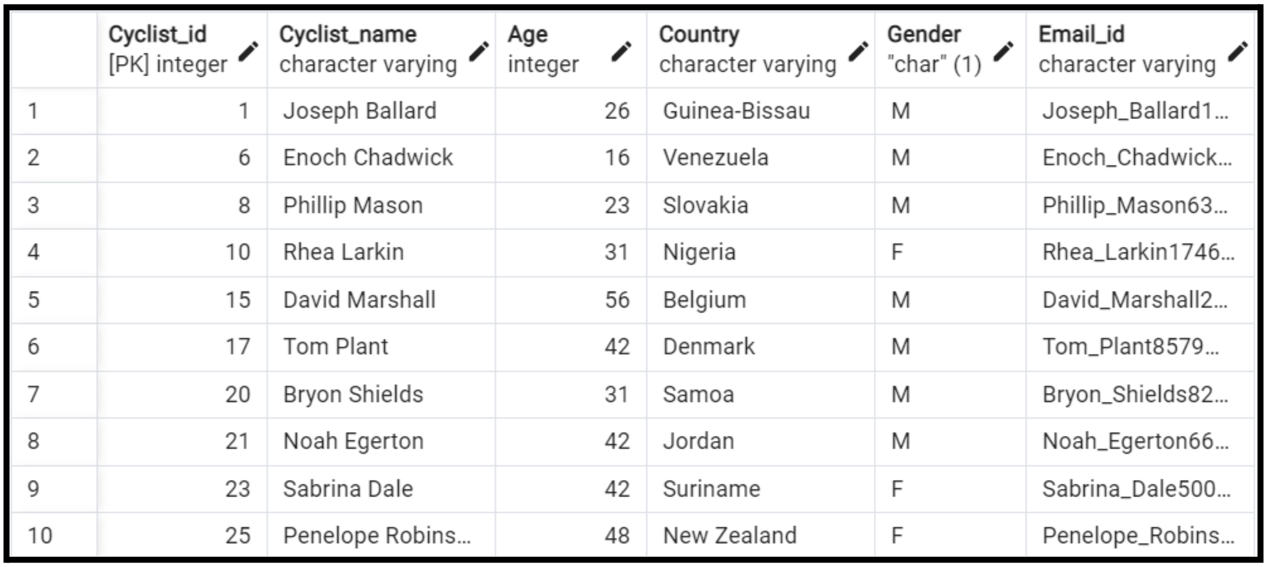


	Sponsor_name character varying
1	Vodafone
2	Carrefour
3	Podcat
4	Zepter
5	Demaco
6	It Smart Group
7	Team Guard SRL
8	Facebook

Total rows: 19 of 19

11) Display Cyclist information who got ‘1st Rank’ in any event.

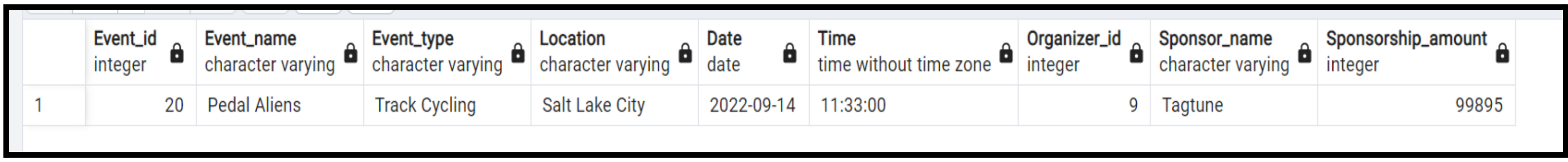
```
SELECT DISTINCT(P1.*)
FROM cyclist_db."Cyclist" P1 NATURAL JOIN
cyclist_db."Cyclist_performance" P2
WHERE P2."Rank"=1
ORDER BY P1."Cyclist_id";
```



	Cyclist_id [PK] integer	Cyclist_name character varying	Age integer	Country character varying	Gender "char" (1)	Email_id character varying
1	1	Joseph Ballard	26	Guinea-Bissau	M	Joseph_Ballard1...
2	6	Enoch Chadwick	16	Venezuela	M	Enoch_Chadwick...
3	8	Phillip Mason	23	Slovakia	M	Phillip_Mason63...
4	10	Rhea Larkin	31	Nigeria	F	Rhea_Larkin1746...
5	15	David Marshall	56	Belgium	M	David_Marshall2...
6	17	Tom Plant	42	Denmark	M	Tom_Plant8579...
7	20	Bryon Shields	31	Samoa	M	Bryon_Shields82...
8	21	Noah Egerton	42	Jordan	M	Noah_Egerton66...
9	23	Sabrina Dale	42	Suriname	F	Sabrina_Dale500...
10	25	Penelope Robins...	48	New Zealand	F	Penelope_Robins...

12) Print details of the Event having the highest ‘Sponsorship_amount’.

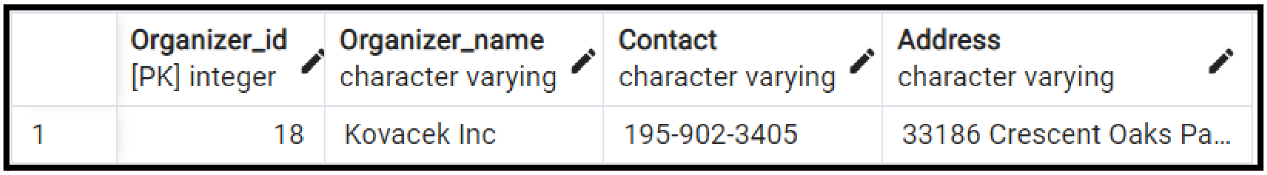
```
SELECT  *
FROM cyclist_db."Event" P1 NATURAL JOIN
cyclist_db."Event_sponsor" P2
ORDER BY P2."Sponsorship_amount" DESC LIMIT 1
```



	Event_id integer	Event_name character varying	Event_type character varying	Location character varying	Date date	Time time without time zone	Organizer_id integer	Sponsor_name character varying	Sponsorship_amount integer
1	20	Pedal Aliens	Track Cycling	Salt Lake City	2022-09-14	11:33:00	9	Tagtune	99895

13) Display the details of ‘Organizer’ that organized most Events.

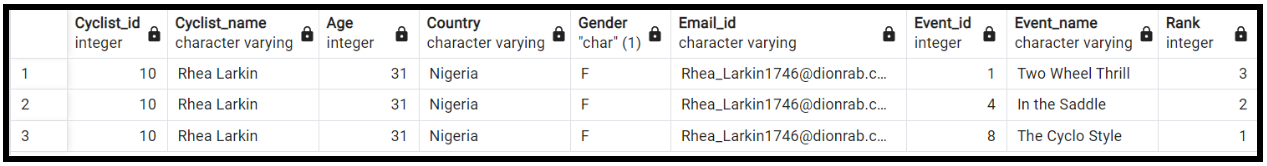
```
SELECT *
FROM cyclist_db."Organizer"
WHERE "Organizer_id"=(SELECT "Organizer_id"
                                        FROM cyclist_db."Event"
                                        GROUP BY "Organizer_id"
                                        ORDER BY COUNT("Organizer_id")
                                        DESC LIMIT 1
                                        );
```



	Organizer_id [PK] integer	Organizer_name character varying	Contact character varying	Address character varying
1	18	Kovacek Inc	195-902-3405	33186 Crescent Oaks Pa...

14) Display details of Cyclists who won in most events.

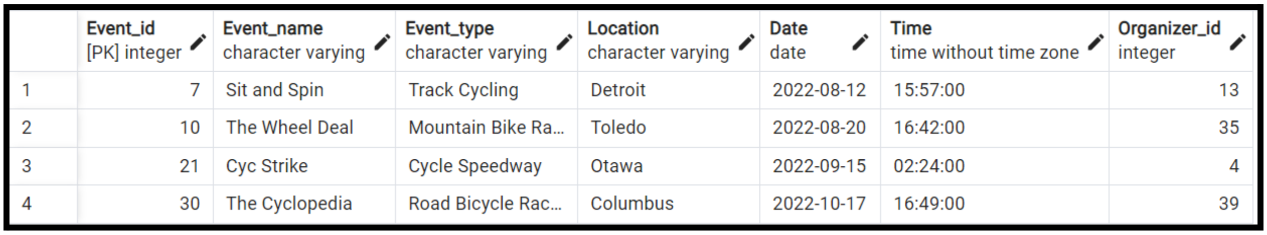
```
SELECT *
FROM cyclist_db."Cyclist" P1 NATURAL JOIN
cyclist_db."Cyclist_performance"
WHERE "Cyclist_id"=(SELECT "Cyclist_id"
                                     FROM cyclist_db."Cyclist_performance"
                                     GROUP BY "Cyclist_id"
                                     ORDER BY COUNT("Cyclist_id") DESC LIMIT 1
                                    );
```



	Cyclist_id integer	Cyclist_name character varying	Age integer	Country character varying	Gender "char" (1)	Email_id character varying	Event_id integer	Event_name character varying	Rank integer
1	10	Rhea Larkin	31	Nigeria	F	Rhea_Larkin1746@dionrab.c...	1	Two Wheel Thrill	3
2	10	Rhea Larkin	31	Nigeria	F	Rhea_Larkin1746@dionrab.c...	4	In the Saddle	2
3	10	Rhea Larkin	31	Nigeria	F	Rhea_Larkin1746@dionrab.c...	8	The Cyclo Style	1

15) Display details of Events which got ‘canceled’.

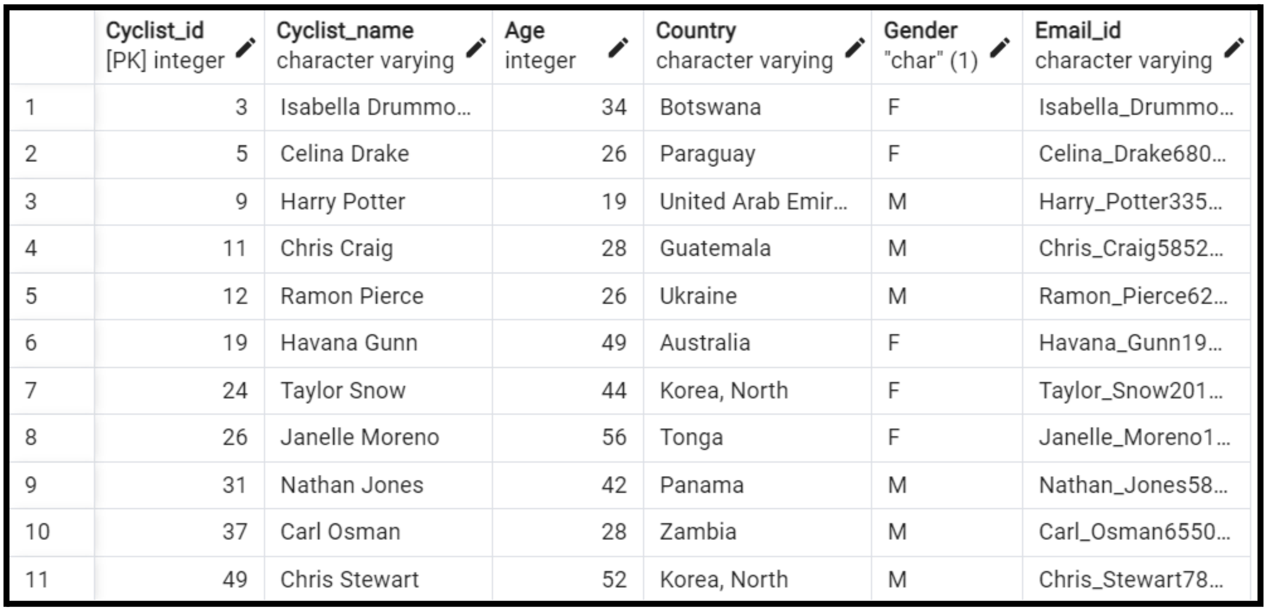
```
SELECT P1.*
FROM cyclist_db."Event" P1 NATURAL JOIN cyclist_db."Event_status" P2
WHERE P2."Status"='canceled';
```



	Event_id [PK] integer	Event_name character varying	Event_type character varying	Location character varying	Date date	Time time without time zone	Organizer_id integer
1	7	Sit and Spin	Track Cycling	Detroit	2022-08-12	15:57:00	13
2	10	The Wheel Deal	Mountain Bike Ra...	Toledo	2022-08-20	16:42:00	35
3	21	Cyc Strike	Cycle Speedway	Otawa	2022-09-15	02:24:00	4
4	30	The Cyclopedia	Road Bicycle Rac...	Columbus	2022-10-17	16:49:00	39

16) Display details of Cyclists who have not participated in any Event.

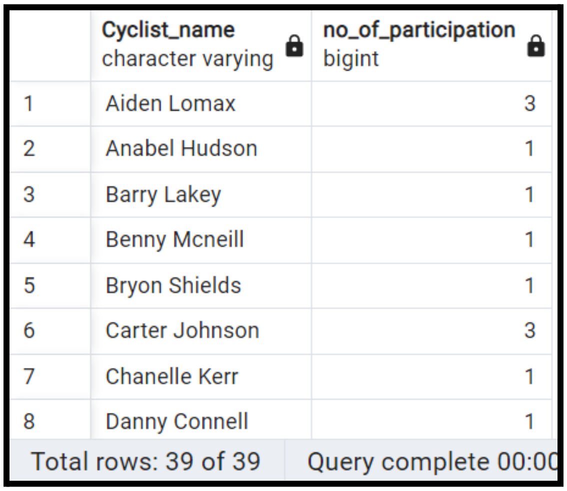
```
SELECT *
FROM cyclist_db."Cyclist"
WHERE "Cyclist_id" NOT IN (SELECT DISTINCT("Cyclist_id")
                           FROM cyclist_db."Participant"
                           );
```



	Cyclist_id [PK] integer	Cyclist_name character varying	Age integer	Country character varying	Gender "char" (1)	Email_id character varying
1	3	Isabella Drummo...	34	Botswana	F	Isabella_Drummo...
2	5	Celina Drake	26	Paraguay	F	Celina_Drake680...
3	9	Harry Potter	19	United Arab Emir...	M	Harry_Potter335...
4	11	Chris Craig	28	Guatemala	M	Chris_Craig5852...
5	12	Ramon Pierce	26	Ukraine	M	Ramon_Pierce62...
6	19	Havana Gunn	49	Australia	F	Havana_Gunn19...
7	24	Taylor Snow	44	Korea, North	F	Taylor_Snow201...
8	26	Janelle Moreno	56	Tonga	F	Janelle_Moreno1...
9	31	Nathan Jones	42	Panama	M	Nathan_Jones58...
10	37	Carl Osman	28	Zambia	M	Carl_Osman6550...
11	49	Chris Stewart	52	Korea, North	M	Chris_Stewart78...

17) Display ‘Cyclist_name’ with their no. of participation in different Events.

```
SELECT P1."Cyclist_name",COUNT(P2."Cyclist_id") AS
No_of_participation
FROM cyclist_db."Cyclist" P1 NATURAL JOIN cyclist_db."Participant" P2
GROUP BY P1."Cyclist_name"
ORDER BY P1."Cyclist_name";
```

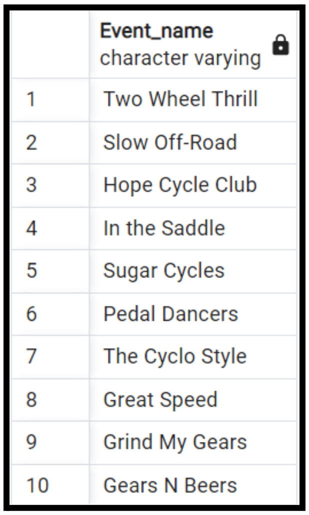


	Cyclist_name character varying	no_of_participation bigint
1	Aiden Lomax	3
2	Anabel Hudson	1
3	Barry Lakey	1
4	Benny Mcneill	1
5	Bryon Shields	1
6	Carter Johnson	3
7	Chanelle Kerr	1
8	Danny Connell	1

Total rows: 39 of 39 Query complete 00:00

18) Display name of Events held in the month of August 2022 and were completed.

```
SELECT P1."Event_name"
FROM cyclist_db."Event" P1 NATURAL JOIN cyclist_db."Event_status" P2
WHERE (P1."Date" BETWEEN '2022-08-01' AND '2022-08-31') AND
(P2."Status"='completed');
```



	Event_name character varying
1	Two Wheel Thrill
2	Slow Off-Road
3	Hope Cycle Club
4	In the Saddle
5	Sugar Cycles
6	Pedal Dancers
7	The Cyclo Style
8	Great Speed
9	Grind My Gears
10	Gears N Beers

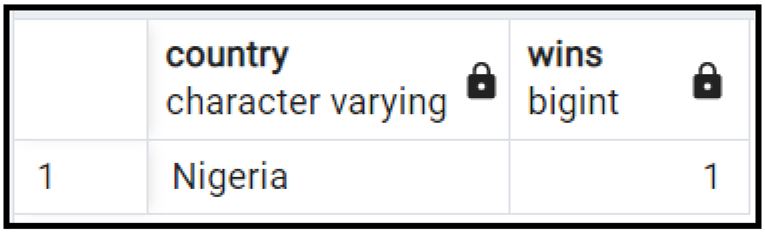
19) Create a function to check performance of a country by giving inputs as country_name and rank, the function returns no of times the country has got the rank.

- **Function:**

```
CREATE OR REPLACE function cyclist_db.winCheck(a character varying,
                                                b int)
RETURNS TABLE (Country character varying,
               Wins bigint)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    RETURN QUERY
    SELECT "Cyclist_country",COUNT("Rank")
    FROM cyclist_db."Country_performance"
    WHERE "Cyclist_country"=a AND "Rank"=b
    GROUP BY "Cyclist_country";
END;
$BODY$;
```

- **Query:**

```
SELECT *
FROM cyclist_db.winCheck('Nigeria',2);
```



	country character varying 🔒	wins bigint 🔒
1	Nigeria	1

20) Create a trigger on the relation Cyclist_performance to check before insert, if the status of the event is completed, then add that tuple to the relation.

- **Trigger function:**

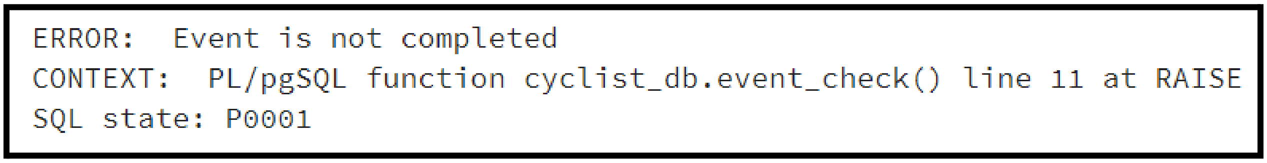
```
CREATE OR REPLACE FUNCTION cyclist_db.event_check()
RETURNS TRIGGER
LANGUAGE 'plpgsql'
AS
$BODY$
DECLARE
        current_status varchar;
BEGIN
        SELECT "Status" INTO current_status
        FROM cyclist_db."Event_status"
        WHERE "Event_id" = NEW."Event_id";
        IF (current_status = 'canceled')
        THEN
        RAISE EXCEPTION 'Event is not completed';
        END IF;
        RETURN NEW;
END;
$BODY$
```

- **Trigger:**

```
CREATE OR REPLACE TRIGGER eventChecker
BEFORE INSERT
ON cyclist_db."Cyclist_performance"
FOR EACH ROW
EXECUTE FUNCTION cyclist_db.event_check();
```

- **Query:**

```
INSERT INTO cyclist_db."Cyclist_performance"
VALUES (7,'test',1,1);
```



```
ERROR:  Event is not completed
CONTEXT:  PL/pgSQL function cyclist_db.event_check() line 11 at RAISE
SQL state: P0001
```

Section 7: Project Interface

1. Back-End Code
2. Interface Screenshots

1. Back-End Code

- **Connects frontend with backend (using Django):**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': '202001182_new',
        'USER' : 'postgres',
        'PASSWORD' : 'admin',
        'HOST' : 'localhost',
        'PORT' : '5433',
    }
}
```

- **Code for setting up the URL:**

```
urlpatterns = [
   path('admin/', admin.site.urls),
   path(", views.homepage, name="homepage"),
   path(", views.showemp, name="showemp"),
   path('insert_form', views.insert_form, name="insert_form"),
   path('insertemp', views.insertemp, name="insertemp"),
   path('editemp/<int:cyclist_id>', views.editemp, name="editemp"),
   path('update/<int:cyclist_id>', views.updateemp, name="updateemp"),
   path('delemp/<int:cyclist_id>', views.delemp, name="delemp"),
   path('showcyclist', views.showcyclist, name="showcyclist"),
   path('showorganizer', views.showorganizer, name="showorganizer"),
   path('insert_organizer', views.insert_organizer, name="insert_organizer"),
   path('organizer_temp', views.organizer_temp, name="organizer_temp"),
   path('organizer_editemp/<int:organizer_id>',
        views.organizer_editemp, name="organizer_editemp"),
   path('organizer_updateemp/<int:organizer_id>',
        views.organizer_updateemp, name="organizer_updateemp"),
   path('organizer_delemp/<int:organizer_id>',
        views.organizer_delemp, name="organizer_delemp"),
   path('showevent', views.showevent, name="showevent"),
   path('event_temp', views.event_temp, name="event_temp"),
   path('insert_event', views.insert_event, name="insert_event"),
   path('showsponsor', views.showsponsor, name="showsponsor"),
   path('event_editemp/<int:event_id>',
        views.event_editemp, name="event_editemp"),
   path('event_updateemp/<int:event_id>',
        views.event_updateemp, name="event_updateemp"),
   path('event_delemp/<int:event_id>', views.event_delemp, name="event_delemp"),
   path('sortCyclist', views.sortCyclist, name="sortCyclist"),
   path('sortOrganizer', views.sortOrganizer, name="sortOgranizer"),
   path('sortevent', views.sortevent, name="sortevent"),
   path('custom_query', views.custom_query, name="custom_query"),
   path('run_query', views.run_query, name="run_query"),
]
```

- **Code for setting up the module used for frontend:**

```
class Cyclist(models.Model):
  # Field name made lowercase.
  cyclist_id = models.IntegerField(db_column='Cyclist_id', primary_key=True)
  # Field name made lowercase.
  cyclist_name = models.CharField(
    db_column='Cyclist_name', max_length=50, blank=True, null=True)
  age = models.IntegerField(db_column='Age')  # Field name made lowercase.
  # Field name made lowercase.
  country = models.CharField(
    db_column='Country', max_length=50, blank=True, null=True)
  # Field name made lowercase. This field type is a guess.
  gender = models.TextField(db_column='Gender', blank=True, null=True)
  # Field name made lowercase.
  email_id = models.CharField(
    db_column='Email_id', max_length=50, blank=True, null=True)

  class Meta:
    managed = False
    db_table = 'Cyclist'


class Organizer(models.Model):
  # Field name made lowercase.
  organizer_id = models.IntegerField(
    db_column='Organizer_id', primary_key=True)
  # Field name made lowercase.
  organizer_name = models.CharField(
    db_column='Organizer_name', max_length=50, blank=True, null=True)
  # Field name made lowercase.
  contact = models.CharField(
    db_column='Contact', max_length=50, blank=True, null=True)
  # Field name made lowercase.
  address = models.CharField(
    db_column='Address', max_length=50, blank=True, null=True)
  class Meta:
    managed = False
    db_table = 'Organizer'

  def __str__(self):
    return str(self.organizer_id)


class Event(models.Model):
  # Field name made lowercase.
  event_id = models.IntegerField(db_column='Event_id', primary_key=True)
  # Field name made lowercase.
  event_name = models.CharField(
    db_column='Event_name', max_length=50, blank=True, null=True)
  # Field name made lowercase.
  event_type = models.CharField(
    db_column='Event_type', max_length=50, blank=True, null=True)
  # Field name made lowercase.
  location = models.CharField(db_column='Location', max_length=50)
  date = models.DateField(db_column='Date')  # Field name made lowercase.
  time = models.TimeField(db_column='Time')  # Field name made lowercase.
  # Field name made lowercase.
  organizer = models.ForeignKey(
    'Organizer', models.DO_NOTHING, db_column='Organizer_id', blank=True, null=True)

  class Meta:
    managed = False
    db_table = 'Event'


class Participant(models.Model):
  # Field name made lowercase.
  event = models.OneToOneField(
    Event, models.DO_NOTHING, db_column='Event_id', primary_key=True)
  # Field name made lowercase.
  cyclist = models.ForeignKey(
    Cyclist, models.DO_NOTHING, db_column='Cyclist_id')
  # Field name made lowercase.
  event_name = models.CharField(db_column='Event_name', max_length=50)
  # Field name made lowercase.
  cyclist_type = models.CharField(
    db_column='Cyclist_type', max_length=50, blank=True, null=True)
  # Field name made lowercase.
  cycle_type = models.CharField(
    db_column='Cycle_type', max_length=50, blank=True, null=True)
  class Meta:
    managed = False
    db_table = 'Participant'
    unique_together = (('event', 'cyclist', 'event_name'),)


class CountryPerformance(models.Model):
  # Field name made lowercase.
  event = models.OneToOneField(
    'Event', models.DO_NOTHING, db_column='Event_id', primary_key=True)
  rank = models.IntegerField(db_column='Rank')  # Field name made lowercase.
  # Field name made lowercase.
  cyclist_country = models.CharField(
    db_column='Cyclist_country', max_length=50, blank=True, null=True)

  class Meta:
    managed = False
    db_table = 'Country_performance'
    unique_together = (('event', 'rank'),)


class CyclistPerformance(models.Model):
  # Field name made lowercase.
  event = models.OneToOneField(
    'Event', models.DO_NOTHING, db_column='Event_id', primary_key=True)
  # Field name made lowercase.
  event_name = models.CharField(db_column='Event_name', max_length=50)
  rank = models.IntegerField(db_column='Rank')  # Field name made lowercase.
  # Field name made lowercase.
  cyclist = models.ForeignKey(
    Cyclist, models.DO_NOTHING, db_column='Cyclist_id')

  class Meta:
    managed = False
    db_table = 'Cyclist_performance'
    unique_together = (('event', 'event_name', 'rank'),)


class EventSponsor(models.Model):
  # Field name made lowercase.
  event = models.OneToOneField(
    Event, models.DO_NOTHING, db_column='Event_id', primary_key=True)
  # Field name made lowercase.
  sponsor_name = models.ForeignKey(
    'Sponsor', models.DO_NOTHING, db_column='Sponsor_name')
  # Field name made lowercase.
  sponsorship_amount = models.IntegerField(
        db_column='Sponsorship_amount', blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'Event_sponsor'
        unique_together = (('event', 'sponsor_name'),)


class EventStatus(models.Model):
    # Field name made lowercase.
    event = models.OneToOneField(
        Event, models.DO_NOTHING, db_column='Event_id', primary_key=True)
    # Field name made lowercase.
    status = models.CharField(db_column='Status', max_length=50)
    date = models.DateField(db_column='Date')  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Event_status'
        unique_together = (('event', 'status'),)


class Sponsor(models.Model):
    # Field name made lowercase.
    sponsor_name = models.CharField(
        db_column='Sponsor_name', primary_key=True, max_length=50)
    # Field name made lowercase.
    sponsor_type = models.CharField(
        db_column='Sponsor_type', max_length=50, blank=True, null=True)
    # Field name made lowercase.
    location = models.CharField(
        db_column='Location', max_length=50, blank=True, null=True)
    # Field name made lowercase.
    contact = models.CharField(
        db_column='Contact', max_length=50, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'Sponsor'
```

- **Code for fetching, editing, deleting and sorting in the database:**

```
def homepage(request):
  return render(request, 'homepage.html')


def showemp(request):
  showall = Cyclist.objects.all()
  print(showall)
  return render(request, 'show_cyclist.html', {"data": showall})


def insert_form(request):

  return render(request, 'insert.html')


def insert_organizer(request):
  return render(request, 'insert_organizer.html')


def insertemp(request):
  print("Inserting")
  saverecord = Cyclist()
  saverecord.cyclist_id = request.POST.get('cyclist_id')
  saverecord.cyclist_name = request.POST.get('cyclist_name')
  saverecord.age = request.POST.get('age')
  saverecord.country = request.POST.get('country')
  saverecord.gender = request.POST.get('gender')
  saverecord.email_id = request.POST.get('email_id')
  allval = Cyclist.objects.all()
  for i in allval:
    if int(i.cyclist_id) == int(request.POST.get('cyclist_id')):
      messages.warning(request, 'cycslist_id already exists....!')
      return render(request, 'insert.html')
  saverecord.save()
  messages.success(request, 'Cyclist ' +
          saverecord.cyclist_name + 'is saved successfully..! ')
  return render(request, 'insert.html')


def editemp(request, cyclist_id):
  editempobj = Cyclist.objects.get(cyclist_id=cyclist_id)
  return render(request, 'edit.html', {"Cyclist": editempobj})
```

```
def updateemp(request, cyclist_id):
  updateemp = Cyclist.objects.get(cyclist_id=cyclist_id)
  cid = request.POST.get("cyclist_id")
  cname = request.POST.get("cyclist_name")
  age = request.POST.get("age")
  country = request.POST.get("country")
  gend = request.POST.get("gender")
  email = request.POST.get("email_id")

  original = Cyclist.objects.get(cyclist_id=cid)

  original.cyclist_name = cname
  original.age = age
  original.country = country
  original.gender = gend
  original.email_id = email
  original.save()
  # form=cyclistform(request.POST,instance=updateemp)
  # if form.is_valid():
  #     form.save()
  messages.success(request, 'record updated succesfully..!')
  return render(request, 'edit.html', {"Cyclist": original})


def delemp(request, cyclist_id):
  delecyclist = Cyclist.objects.get(cyclist_id=cyclist_id)
  delecyclist.delete()
  showall = Cyclist.objects.all()

  return render(request, 'show_cyclist.html', {"data": showall})


def showcyclist(request):
  showall = Cyclist.objects.all()
  print(showall)
  return render(request, 'show_cyclist.html', {"data": showall})


def showorganizer(request):
  showall = Organizer.objects.all()
  print(showall)
  return render(request, 'showorganizer.html', {"data": showall})


def organizer_temp(request):

  return render(request, 'insert_organizer.html')
```

```
def insert_organizer(request):
 # if request.method=="POST":
  # if request.POST.get('organizer_id') and request.POST.get('organizer_name') and
request.POST.get('contact') and request.POST.get('address'):

  saverecord = Organizer()
  saverecord.organizer_id = request.POST.get('organizer_id')
  saverecord.organizer_name = request.POST.get('organizer_name')
  saverecord.contact = request.POST.get('contact')
  saverecord.address = request.POST.get('address')

  allval = Organizer.objects.all()
  for i in allval:
    if int(i.organizer_id) == int(request.POST.get('organizer_id')):
      messages.warning(request, 'organizer_id already exists....!')
      return render(request, 'insert_organizer.html')
  saverecord.save()
  messages.success(request, 'organizer ' +
           saverecord.organizer_name + ' is saved successfully..! ')
  return render(request, 'insert_organizer.html')


def organizer_editemp(request, organizer_id):
  editempobj = Organizer.objects.get(organizer_id=organizer_id)
  return render(request, 'organizer_edit.html', {"Organizer": editempobj})


def organizer_updateemp(request, organizer_id):
  updateemp = Organizer.objects.get(organizer_id=organizer_id)
  cid = request.POST.get("organizer_id")
  cname = request.POST.get("organizer_name")
  age = request.POST.get("contact")
  country = request.POST.get("address")

  original = Organizer.objects.get(organizer_id=cid)

  original.organizer_name = cname
  original.contact = age
  original.address = country
  original.save()
  # form=cyclistform(request.POST,instance=updateemp)
  # if form.is_valid():
  #    form.save()
  messages.success(request, 'record updated succesfully..!')
  return render(request, 'organizer_edit.html', {"Organizer": original})
```

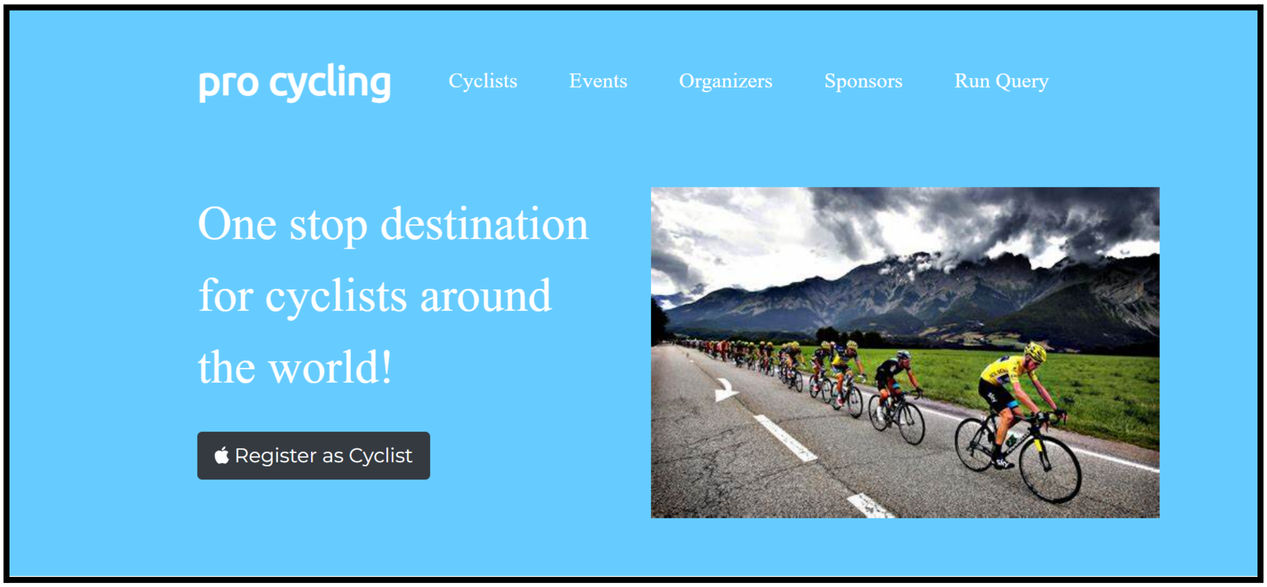
```
def organizer_delemp(request, organizer_id):
  delecyclist = Organizer.objects.get(organizer_id=organizer_id)
  delecyclist.delete()
  showall = Organizer.objects.all()

  return render(request, 'showorganizer.html', {"data": showall})


def showevent(request):
  showall = Event.objects.all()
  print(showall)
  return render(request, 'showevent.html', {"data": showall})


def event_temp(request):

  return render(request, 'insert_event.html')


def insert_event(request):
  print("Inserting")
  saverecord = Event()
  saverecord.event_id = request.POST.get('event_id')
  saverecord.event_name = request.POST.get('event_name')
  saverecord.event_type = request.POST.get('event_type')
  saverecord.location = request.POST.get('location')
  saverecord.date = request.POST.get('date')
  saverecord.time = request.POST.get('time')
  saverecord.organizer = request.POST.get('organizer_id')

  saverecord.save()
  messages.success(request, 'event ' +
                   saverecord.event_name + ' is saved successfully..! ')
  return render(request, 'insert_event.html')


def showsponsor(request):
  showall = Sponsor.objects.all()
  print(showall)
  return render(request, 'showsponsor.html', {"data": showall})


def event_editemp(request, event_id):
  editempobj = Event.objects.get(event_id=event_id)
  return render(request, 'event_edit.html', {"Event": editempobj})
```

```
def event_updateemp(request, event_id):
    updateemp = Event.objects.get(event_id=event_id)
    cid = request.POST.get("event_id")
    cname = request.POST.get("event_name")
    age = request.POST.get("event_type")
    country = request.POST.get("location")
    dat = request.POST.get("date")
    tim = request.POST.get("time")
    org = request.POST.get("organizer_id")

    original = Event.objects.get(event_id=cid)

    original.event_name = cname
    original.event_type = age
    original.location = country
    original.date = dat
    original.time = tim
    original.organizer_id = org

    original.save()
    # form=cyclistform(request.POST,instance=updateemp)
    # if form.is_valid():
    #     form.save()
    messages.success(request, 'record updated succesfully..!')
    return render(request, 'event_edit.html', {"Event": original})


def event_delemp(request, event_id):
    delecyclist = Event.objects.get(event_id=event_id)
    delecyclist.delete()
    showall = Event.objects.all()

    return render(request, 'showevent.html', {"data": showall})


def sortCyclist(request):
    if request.method == "POST":
        if request.POST.get('Sort'):
            type = request.POST.get('Sort')
            sorted = Cyclist.objects.all().order_by(type)
            context = {
                'data': sorted
            }
            return render(request, 'sortcyclist.html', context)
        else:
            return render(request, 'sortcyclist.html')


def sortOrganizer(request):
    if request.method == "POST":
        if request.POST.get('Sort'):
            type = request.POST.get('Sort')
            sorted = Organizer.objects.all().order_by(type)
            context = {
                'data': sorted
            }
            return render(request, 'sortorganizer.html', context)
        else:
            return render(request, 'sortorganizer.html')


def sortevent(request):
    if request.method == "POST":
        if request.POST.get('Sort'):
            type = request.POST.get('Sort')
            sorted = Event.objects.all().order_by(type)
            context = {
                'data': sorted
            }
            return render(request, 'sortevent.html', context)
        else:
            return render(request, 'sortevent.html')


def runQuerycyclist(request):
    raw_query = 'select "Cyclist_name" from public."Cyclist" where "Age" >= 40; '

    cursor = connection.cursor()
    cursor.execute(raw_query)
    alldata = cursor.fetchall()

    return render(request, 'runQuerycyclist.html', {'data': alldata})


def runQuery2(request):
    raw_query = 'SELECT * FROM "Organizer" WHERE "Organizer_id"=(SELECT "Organizer_id" FROM "Event" GROUP BY "Organizer_id" ORDER BY COUNT("Organizer_id")  DESC LIMIT 1);'

    cursor = connection.cursor()
    cursor.execute(raw_query)
    alldata = cursor.fetchall()
```

```
  return render(request, 'runquery2.html', {'data': alldata})


def runQuery3(request):
  raw_query = 'SELECT * FROM "Cyclist" P1 NATURAL JOIN "Cyclist_performance" WHERE
"Cyclist_id"=(SELECT "Cyclist_id" FROM "Cyclist_performance" GROUP BY "Cyclist_id"  ORDER BY
COUNT("Cyclist_id") DESC LIMIT 1);'

  cursor = connection.cursor()
  cursor.execute(raw_query)
  alldata = cursor.fetchall()

  return render(request, 'runquery3.html', {'data': alldata})


def custom_query(request):
  custom_query = request.POST.get("custom_query")
  print(custom_query)
  #raw_query = "select * from \"Organization\" where org_rating=1;"

  cursor = connection.cursor()
  cursor.execute(custom_query)
  alldata = cursor.fetchall()

  return render(request, 'runquery2.html', {'data': alldata})


def run_query(request):

  return render(request, 'search.html', {})
```

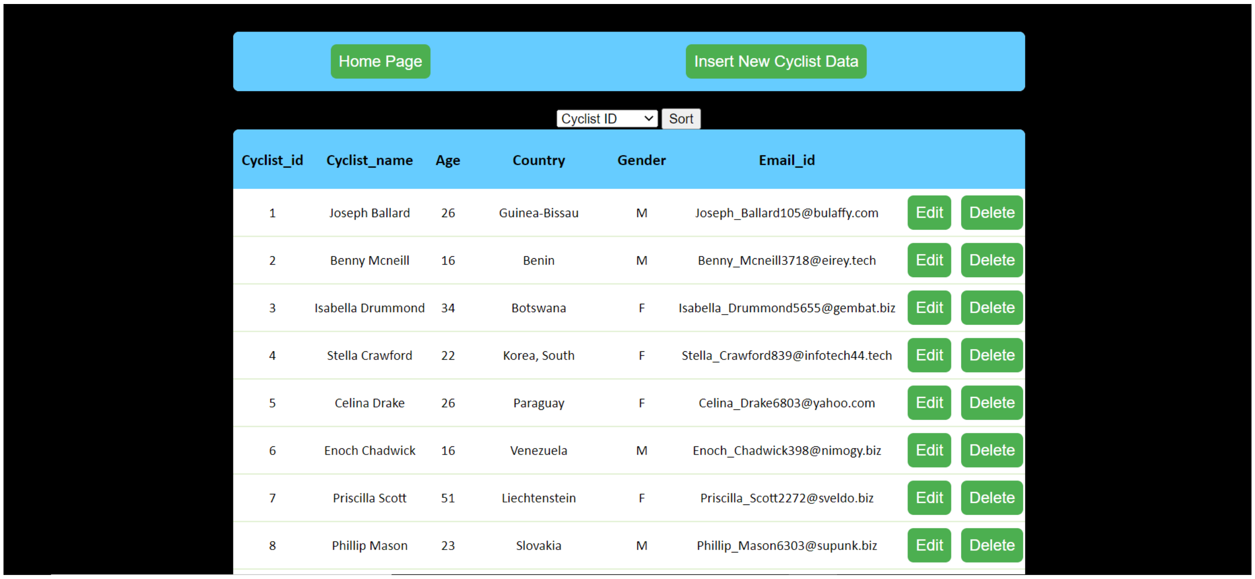
2. Interface Screenshots

1. Home Page:



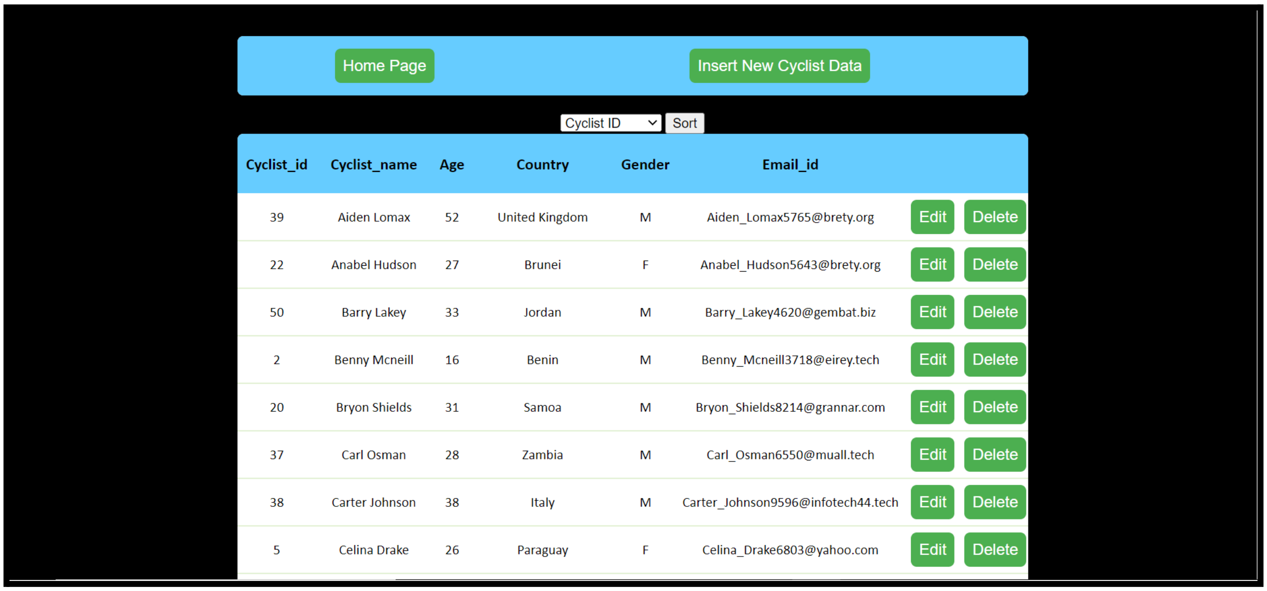
One stop destination for cyclists around the world!

2. Show Cyclist Data:



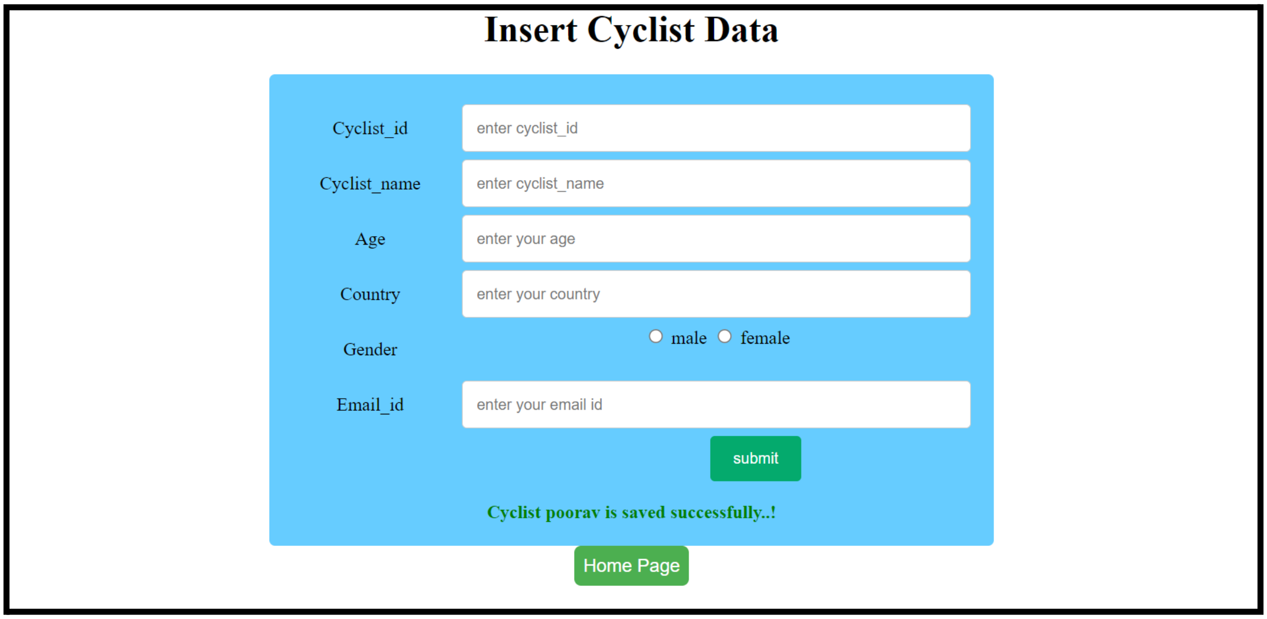
Cyclist_id	Cyclist_name	Age	Country	Gender	Email_id		
1	Joseph Ballard	26	Guinea-Bissau	M	Joseph_Ballard105@bulaffy.com	Edit	Delete
2	Benny Mcneill	16	Benin	M	Benny_Mcneill3718@eirey.tech	Edit	Delete
3	Isabella Drummond	34	Botswana	F	Isabella_Drummond5655@gembat.biz	Edit	Delete
4	Stella Crawford	22	Korea, South	F	Stella_Crawford839@infotech44.tech	Edit	Delete
5	Celina Drake	26	Paraguay	F	Celina_Drake6803@yahoo.com	Edit	Delete
6	Enoch Chadwick	16	Venezuela	M	Enoch_Chadwick398@nimogy.biz	Edit	Delete
7	Priscilla Scott	51	Liechtenstein	F	Priscilla_Scott2272@sveldo.biz	Edit	Delete
8	Phillip Mason	23	Slovakia	M	Phillip_Mason6303@supunk.biz	Edit	Delete

3. Sort Cyclist Database Using Cyclist_name:



Cyclist_id	Cyclist_name	Age	Country	Gender	Email_id		
39	Aiden Lomax	52	United Kingdom	M	Aiden_Lomax5765@brety.org	Edit	Delete
22	Anabel Hudson	27	Brunei	F	Anabel_Hudson5643@brety.org	Edit	Delete
50	Barry Lakey	33	Jordan	M	Barry_Lakey4620@gembat.biz	Edit	Delete
2	Benny Mcneill	16	Benin	M	Benny_Mcneill3718@eirey.tech	Edit	Delete
20	Bryon Shields	31	Samoa	M	Bryon_Shields8214@grannar.com	Edit	Delete
37	Carl Osman	28	Zambia	M	Carl_Osman6550@muall.tech	Edit	Delete
38	Carter Johnson	38	Italy	M	Carter_Johnson9596@infotech44.tech	Edit	Delete
5	Celina Drake	26	Paraguay	F	Celina_Drake6803@yahoo.com	Edit	Delete

4. Insert a Cyclist Data:

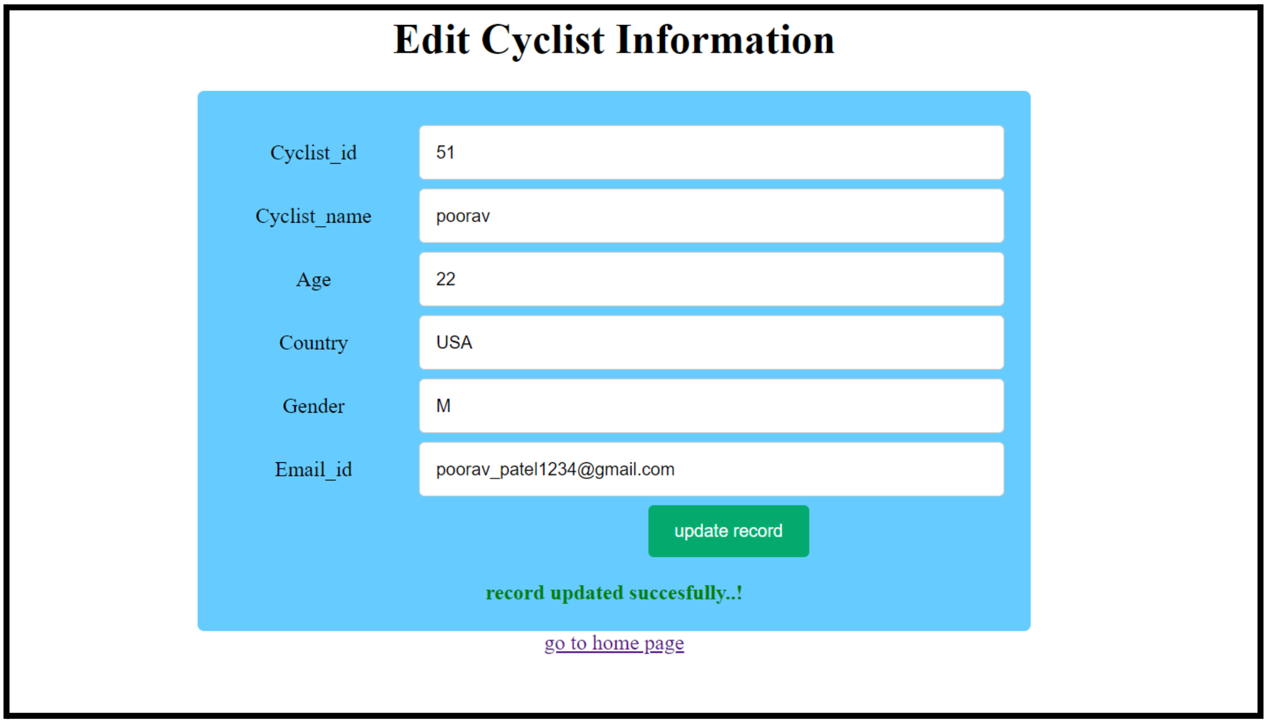


Insert Cyclist Data

Cyclist_id	enter cyclist_id
Cyclist_name	enter cyclist_name
Age	enter your age
Country	enter your country
Gender	○ male ○ female
Email_id	enter your email id

Cyclist poorav is saved successfully..!

5. Update a Cyclist Data:



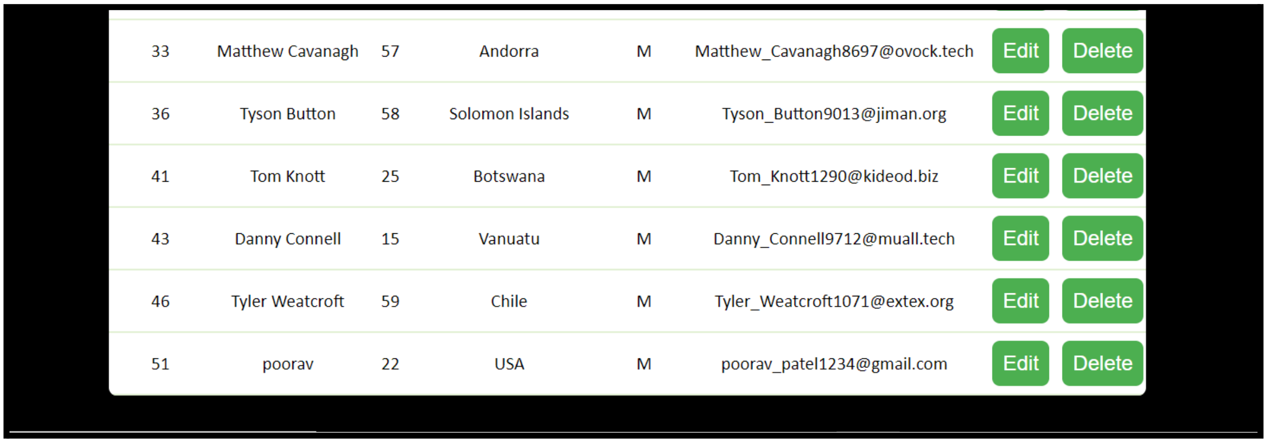
Edit Cyclist Information

Cyclist_id	51
Cyclist_name	poorav
Age	22
Country	USA
Gender	M
Email_id	poorav_patel1234@gmail.com

record updated succesfully..!

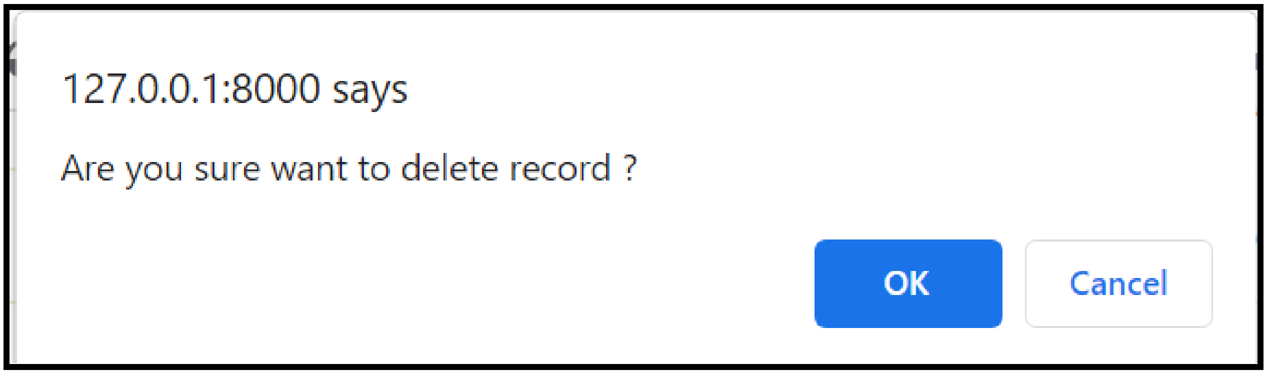
go to home page

6. Delete a Cyclist Data:



33	Matthew Cavanagh	57	Andorra	M	Matthew_Cavanagh8697@ovock.tech	Edit	Delete
36	Tyson Button	58	Solomon Islands	M	Tyson_Button9013@jiman.org	Edit	Delete
41	Tom Knott	25	Botswana	M	Tom_Knott1290@kideod.biz	Edit	Delete
43	Danny Connell	15	Vanuatu	M	Danny_Connell9712@muall.tech	Edit	Delete
46	Tyler Weatcroft	59	Chile	M	Tyler_Weatcroft1071@extex.org	Edit	Delete
51	poorav	22	USA	M	poorav_patel1234@gmail.com	Edit	Delete

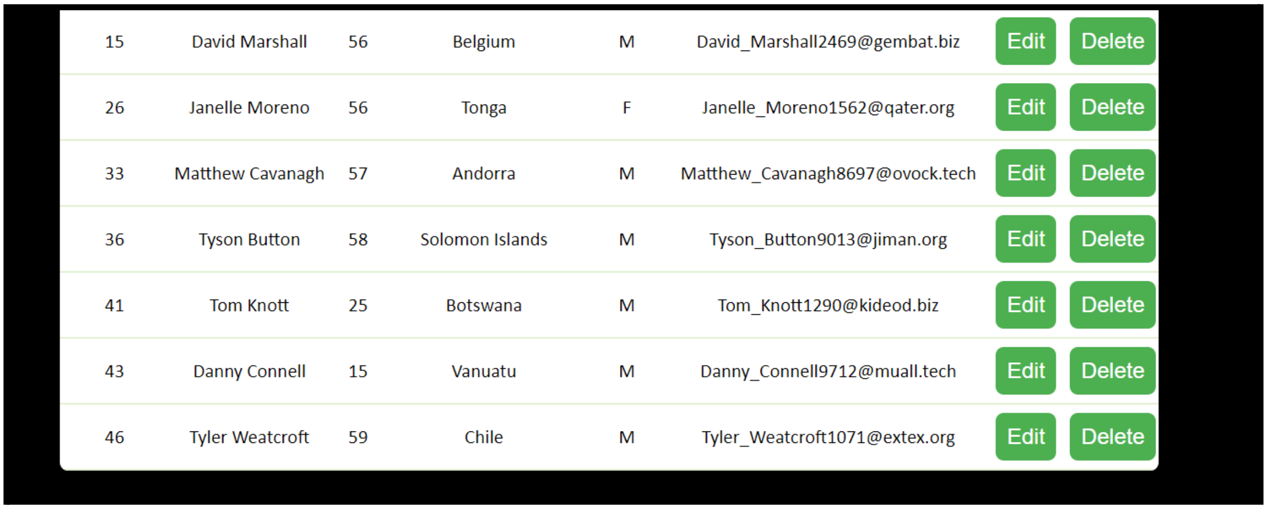
- ➢ On clicking delete button we have a pop-up as shown below:



127.0.0.1:8000 says

Are you sure want to delete record ?

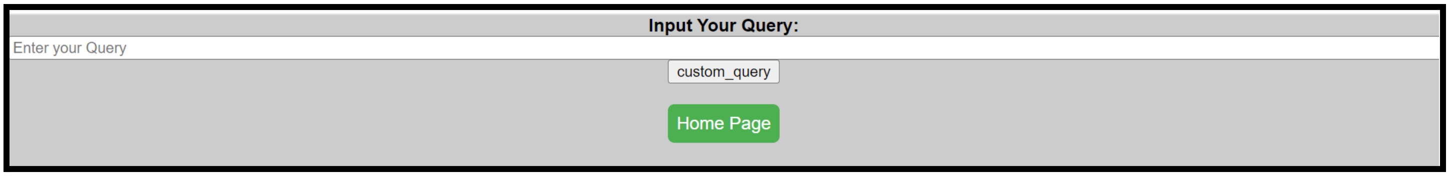
➢ After clicking on ‘OK’, the selected tuple gets deleted from the table as we can see in updated table below:



15	David Marshall	56	Belgium	M	David_Marshall2469@gembat.biz	Edit	Delete
26	Janelle Moreno	56	Tonga	F	Janelle_Moreno1562@qater.org	Edit	Delete
33	Matthew Cavanagh	57	Andorra	M	Matthew_Cavanagh8697@ovock.tech	Edit	Delete
36	Tyson Button	58	Solomon Islands	M	Tyson_Button9013@jiman.org	Edit	Delete
41	Tom Knott	25	Botswana	M	Tom_Knott1290@kideod.biz	Edit	Delete
43	Danny Connell	15	Vanuatu	M	Danny_Connell9712@muall.tech	Edit	Delete
46	Tyler Weatcroft	59	Chile	M	Tyler_Weatcroft1071@extex.org	Edit	Delete

7. Query 1: Display details of Events which got ‘canceled’.

➢ First click on ‘Run Query’ on Home Page so below page loads:

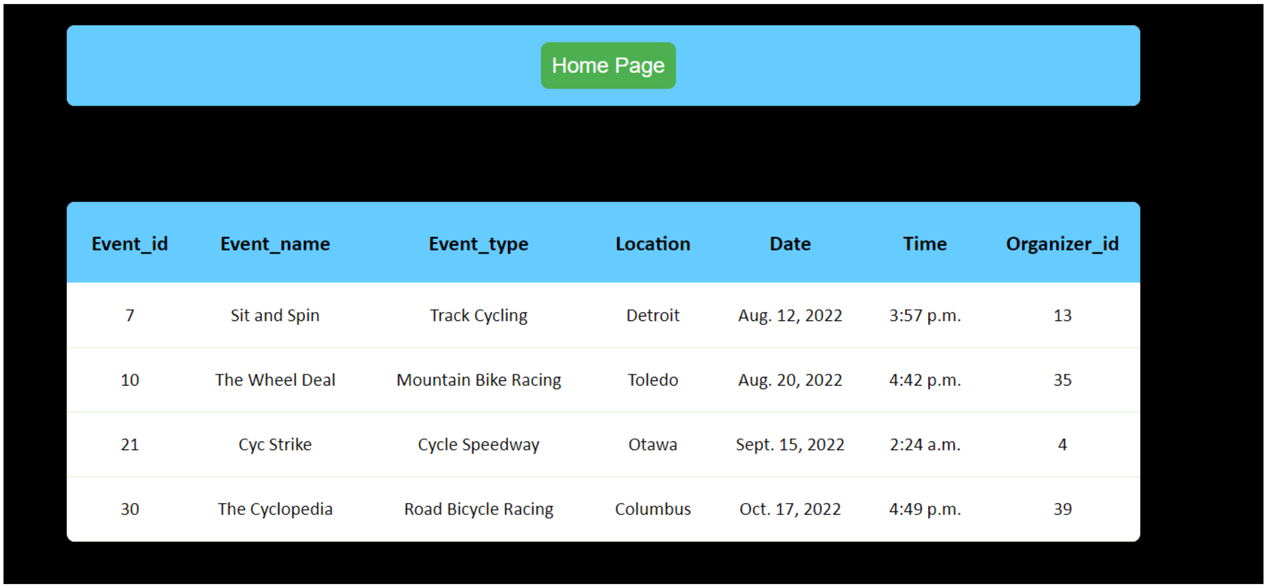


Input Your Query:

Enter your Query

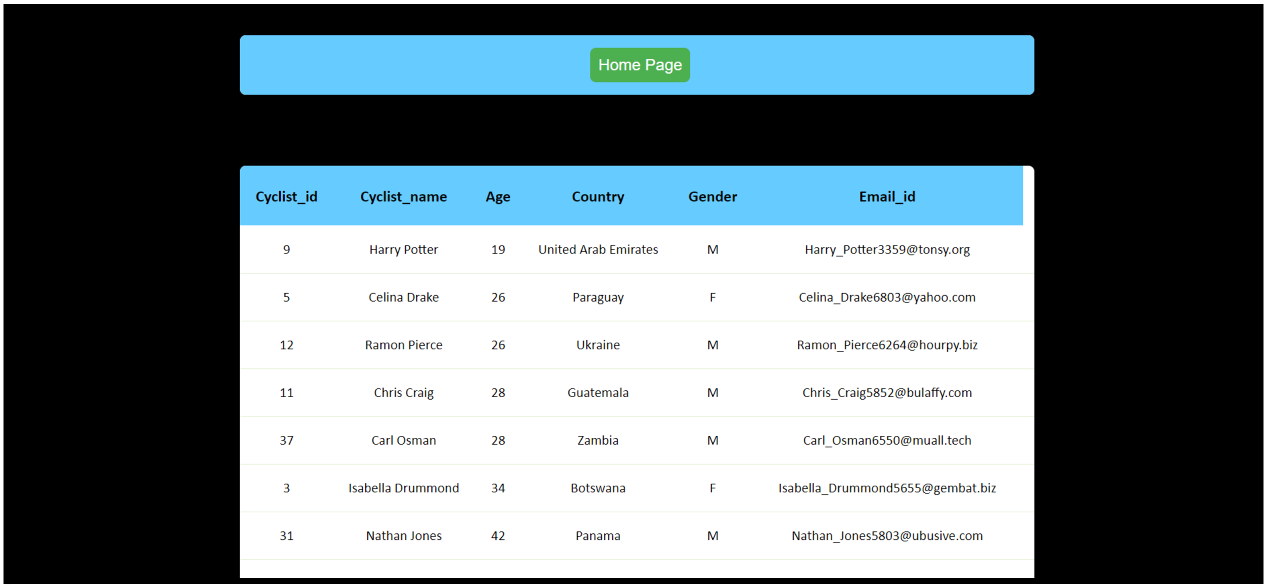
➢ Enter SQL query here and click on ‘custom_query’:

➢ Output is shown below:



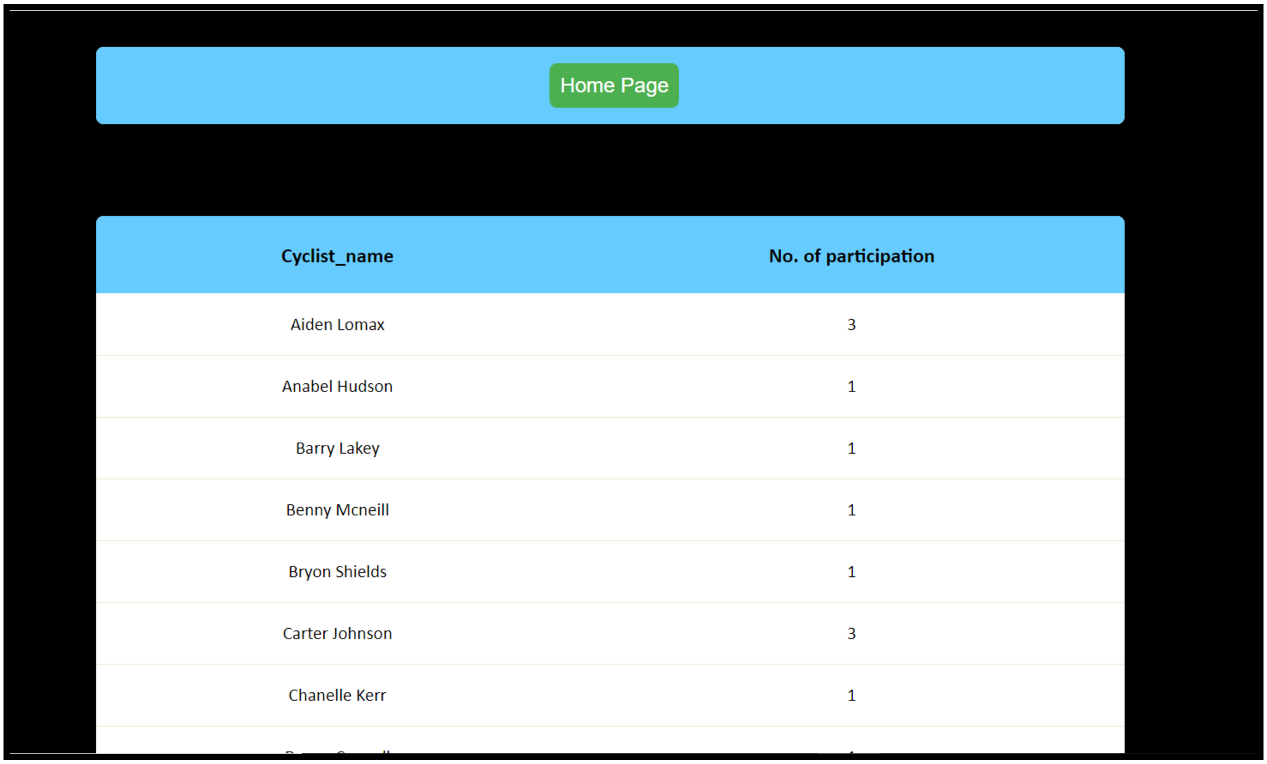
Event_id	Event_name	Event_type	Location	Date	Time	Organizer_id
7	Sit and Spin	Track Cycling	Detroit	Aug. 12, 2022	3:57 p.m.	13
10	The Wheel Deal	Mountain Bike Racing	Toledo	Aug. 20, 2022	4:42 p.m.	35
21	Cyc Strike	Cycle Speedway	Otawa	Sept. 15, 2022	2:24 a.m.	4
30	The Cyclopedia	Road Bicycle Racing	Columbus	Oct. 17, 2022	4:49 p.m.	39

8. Query 2: Display details of Cyclists who have not participated in any Event.



Cyclist_id	Cyclist_name	Age	Country	Gender	Email_id
9	Harry Potter	19	United Arab Emirates	M	Harry_Potter3359@tonsy.org
5	Celina Drake	26	Paraguay	F	Celina_Drake6803@yahoo.com
12	Ramon Pierce	26	Ukraine	M	Ramon_Pierce6264@hourpy.biz
11	Chris Craig	28	Guatemala	M	Chris_Craig5852@bulaffy.com
37	Carl Osman	28	Zambia	M	Carl_Osman6550@muall.tech
3	Isabella Drummond	34	Botswana	F	Isabella_Drummond5655@gembat.biz
31	Nathan Jones	42	Panama	M	Nathan_Jones5803@ubusive.com

9. Query 3: Display ‘Cyclist_name’ with their no. of participation in different Events.



Cyclist_name	No. of participation
Aiden Lomax	3
Anabel Hudson	1
Barry Lakey	1
Benny Mcneill	1
Bryon Shields	1
Carter Johnson	3
Chanelle Kerr	1

GitHub Repository Link:

https://github.com/Hiten324812/DBMS-PROJECT