CSC3170 Database System

Management Database of Sport Meeting

Group Members:

119010321	Wang Yuhao
119010322	Wang Yupeng
119010054	Dai Yulong
119010249	Qin Peiran
119010344	Xiao Nan

The Chinese University of Hong Kong, Shenzhen

1. Introduction and Motivation

For many contemporary large-scale sports events, data management is a tough job. For example, 204 countries and regions participate in the 2008 Beijing Olympic Games. And there are 11438 athletes, including 28 major events and 302 minor events. How to store these data in an organized and efficient way is worth discussing. Thus, our group would like to build a management database for monitoring a particular sports event.

Recently, the playground of CUHKSZ is finally coming into use, which means students can have the first track-and-field sports meeting. This project aims to design a particular database for CUHKSZ's track-and-field sport event. Generally, the database stores information of all players, events, volunteers, organization committee, referees, and so on. Mainly two types of users are considered in this database, player or committee. For players, they can register in the system and choose the events, which update the personal information in the database. For the organizing committee, by accessing the database, they may initialize and track the progress of the event and update the corresponding event results.

2. Design and implementation

2.1 ER Diagram, relational schemas, and normalization

In order to design a concise database including all necessary information meanwhile with as little redundancy as possible, the process of design is divided into four parts. In the first part, entities, attributes, and relationships are decided. In the second part, an Entities-Relationship diagram is drawn according to the first part. In the third part, the entity-relationship diagram is reduced to relational schemas and all schemas are normalized into 3NF. In the last part, the database is built by MySQL and the indexes are created.

2.1.1 Design of entities, attributes, and relationships

The entities in this sport event database are Volunteer, Player, Referee, College, Field, Event, Org Comm (organizing committee), Event result.

Attributes in each entity are identified to be:

Entity	Attributes
player	Player id, Player name, Gender, College name
college	College name, College score, College ranking, Manager
event	Event name, Category, Round, Starting time, Ending time
Event result	Event name, Category, Round, ranking, id, performance
volunteer	Volunteer id, Volunteer name, available starting time, available ending time
Org comm	Org comm id, Org comm name
field	Field name, Available starting time, Available ending time, Org comm
referee	Referee id, referee name, salary, Org comm id

In addition, the relationships between each pair of two entities together with constraints and properties are identified as below:

a) The relationship between player and college is a many-to-one relationship. Each player must belong to a specific college and each college can have many players. Meanwhile, the College name, an attribute in the player entity, is a foreign key of the college entity. It plays the role of foreign key constraint in the database.

b) The relationship between player and event is a many-to-many relationship. It is assumed that one player can sign up for many sports events, and each sport event involves many players.

c) The relationship between event and event result is a one-to-many relationship. Moreover, event result is a weak entity because it is uniquely identified by event name, category, and round. In other words, its existence is dependent on the event entity.

d) The relationship between event and referee is a many-to-many relationship. It is assumed that one referee can judge many events, and each event needs at least one referee.

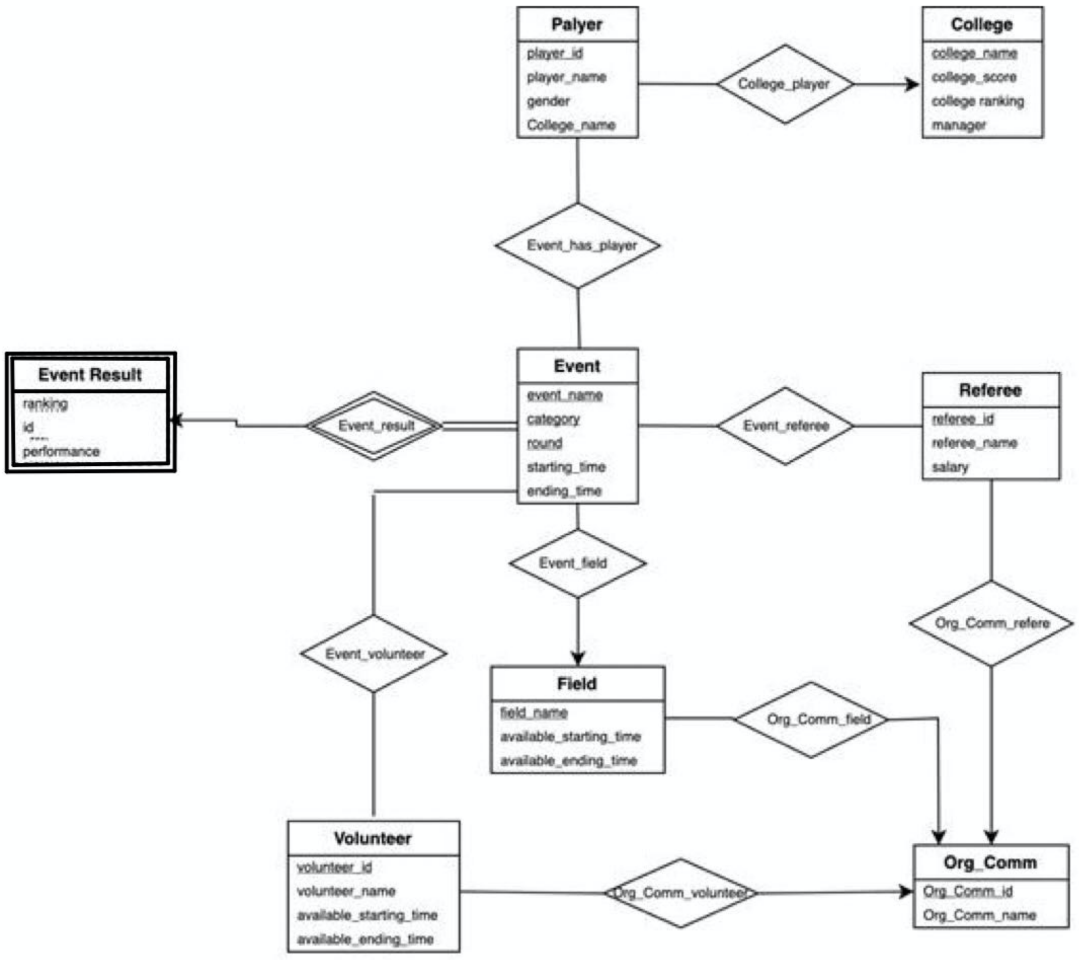
e) The relationship between event and volunteer is a many-to-many relationship. It is assumed that one volunteer can participate in many events, and each event needs at least one volunteer.

f) The relationship between field and event is a one-to-many relationship. Since the design assumes that one field can hold many events, and one event should be held on just one field. For example, a track can be used to hold a 100-meter run and 200-meter run, but both 100-meter run and 200-run cannot be held on other fields.

g) Organizing committee has a one-to-many relationship with field, volunteer, and referee. One organizing committee oversees at least one of the volunteers, fields, or referees.

2.1.2 Entities-Relationships Diagram

According to the entities, attributes, and relationships identified in the first part, an Entities-relationships diagram can be produced for the database:



Graph 2.2: E-R diagram for the sports events database

2.1.3 Relational schemas and normalization

In this part, the E-R diagram is converted to relational schemas. In addition, for each schema, primary keys, foreign keys, and functional dependencies are indicated as follows.

schema	Primary keys	Foreign keys	Function dependencies
player	Player_id	College_name	Player_id -> Player_name, Gender, College_name
college	College_name	College_score	College_name -> College_score, College_ranking, Manager College_score -> College_ranking
event	Event_name, Category, Round	Field_name	Event_name, Category, Round -> Starting_time, Ending_time
event_ha s_player	Event_name, Category, Round, Player_id	Event_name, Category, Round, Player_id	
event_ result	Event_name, Event_category, Event_round, Id	Event_name, Event_category, Event_round	Event_name, Event_category, Event_round-> ranking, id, performance
volunteer	Volunteer_id	Org_comm_id	Volunteer_id -> Volunteer_name, available_staring_time, available_ending_time
volunteer_ has_event	Event_name, Category, Round Volunteer_id	Event_name, Category, Round Volunteer_id	
org_comm	Org_comm_id		Org_comm_id -> Org_comm_name
field	Field_name	Org_comm_id	Field_name -> Available_starting_time, Available_ending_time, Org_comm
referee	Referee_id	Org_comm_id	Referee_id -> referee_name, salary, Org_comm_id
Event_has _referee	Event_name, Category, Round, Referee_id	Event_name, Category, Round, Referee_id	

Table 2.3: primary keys, foreign keys, and functional dependencies

After identifying the functional dependencies, each schema is checked to justify whether it is in good normalization. Some schemas violate the requirement of 3NF:

For schema field as well as volunteer, available_starting_time and available_ending_time are multi-value, which violates 1NF. Therefore, each of them is decomposed into 2 schemas:

field_time(field_name, available_staring_time, available_ending_time)

field_org (Field_name, Org_comm)

volunteer_name(Volunteer_id, Volunteer_name)

volunteer_time(volunteer_id,available_starting_time,available_ending_time)

After these decompositions, all schemas are in 1NF. Then, further checks are performed to justify whether all the schemas are in 2NF and 3NF.

For schema college, there is a function dependency, because college_ranking depends on either college_name or college_score. Therefore, schema college is decomposed into two schemas:

college (<u>College_name</u>, College_score, College_ranking, Manager)

college_rank (College_score, College_ranking)

After these decompositions, all the schemas in this database are in a third normal form.

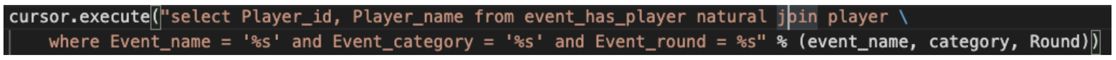
2.1.4 Database construction and index creation

We constructed our database according to the schema specified above using MySQL server. An SQL script was written to create tables and indexes, which is shown in the appendix. In particular, college_score is selected as an index in table College since we can make use of the order given by the index to search for the score, which speeds up searching a lot. In addition, in table Event_has_Player, we selected an index containing Event_name, Event_category, and Event_round which also serves as the foreign key referencing event table. It would speed up the process of searching the participants (players) in a particular event since we can search from the joint table of event and event_has_player and make use of the indexes inside event_has_player to search through the event table instead, which makes the joining process faster.

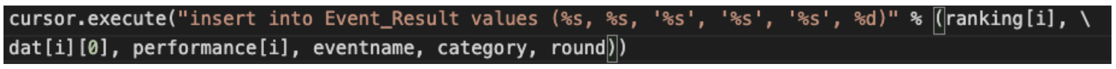
2.2 Sample SQL queries

After getting a well-formed database, some sample SQL queries are produced to equip the database with some daily functions.

Firstly, the database can update the result of a certain sports event. In the beginning, a select clause is used to get the participants of the event.

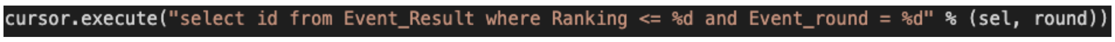

```
cursor.execute("select Player_id, Player_name from event_has_player natural join player \
    where Event_name = '%s' and Event_category = '%s' and Event_round = %s" % (event_name, category, Round))
```

After the operation, the performance of each participant will be inserted into the Event_result entity.


```
cursor.execute("insert into Event_Result values (%s, %s, '%s', '%s', '%s', %d)" % (ranking[i], \
dat[i][0], performance[i], eventname, category, round))
```

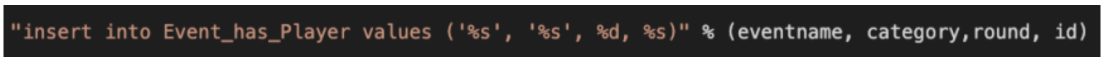
Now the result of the event can be searched in the database.

Besides, if the players have not entered the last round, the database is supposed to qualify the top players of the event to the next round. To achieve this goal, the top players are selected from the Event_Result entity.



```
cursor.execute("select id from Event_Result where Ranking <= %d and Event_round = %d" % (sel, round))
```

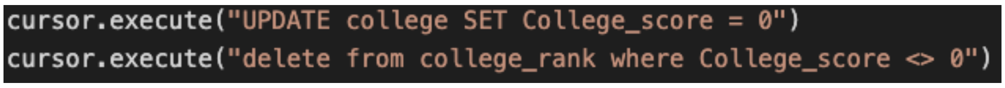
Afterward, these selected players will be inserted into the relationship named Event_has_Player.



```
"insert into Event_has_Player values ('%s', '%s', %d, %s)" % (eventname, category,round, id)
```

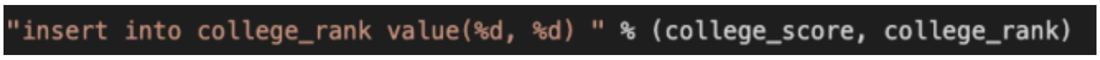
Finally, the participants of the next round will be updated in the database.

Moreover, if the players have entered the last round, the database is supposed to calculate the score of each college and then rank the colleges automatically. The idea of implementing the function is simple. The score and rank of each college will be reset to be 0 firstly. Then the score of each college will be recalculated according to the performances of players and then the ranking and score of each college will be updated. However, the information in the college_rank entity cannot be updated directly because of the foreign key constraint. The information of the college_ranking entity will not be deleted until the data in the college entity is deleted. In addition, when the users want to assign each college to a new score, the score should be inserted into the college_ranking entity first. Therefore, in the queries, the score of each college is updated to 0 firstly and then the rank of each college is deleted except 0.



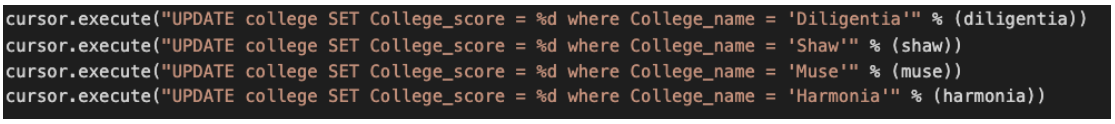
```
cursor.execute("UPDATE college SET College_score = 0")
cursor.execute("delete from college_rank where College_score <> 0")
```

After calculating the score of each college, the score will be ranked first and then insert the rank of each score into the college_ranking entity.



```
"insert into college_rank value(%d, %d) " % (college_score, college_rank)
```

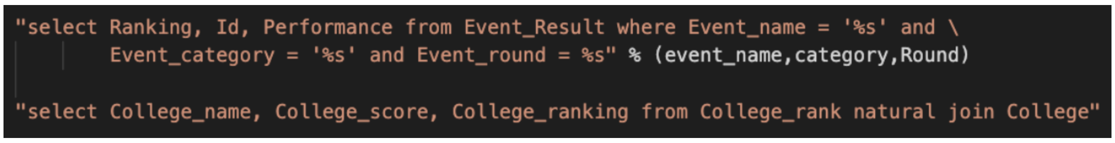
Afterward, we can update the score of each college in the database.



```
cursor.execute("UPDATE college SET College_score = %d where College_name = 'Diligentia'" % (diligentia))
cursor.execute("UPDATE college SET College_score = %d where College_name = 'Shaw'" % (shaw))
cursor.execute("UPDATE college SET College_score = %d where College_name = 'Muse'" % (muse))
cursor.execute("UPDATE college SET College_score = %d where College_name = 'Harmonia'" % (harmonia))
```

Then, the new score and rank of each college are updated in the database.

Finally, some select clauses can be used to get the results of a certain sports event and the real-time rank of the colleges.



```
"select Ranking, Id, Performance from Event_Result where Event_name = '%s' and \
        Event_category = '%s' and Event_round = %s" % (event_name,category,Round)

"select College_name, College_score, College_ranking from College_rank natural join College"
```

Above all, these SQL queries are used to implement many functions of the database. After using these queries, the database can qualify the top players into the next round and update the college ranking automatically. Besides, many contents of the database can be gotten through simple select clauses.

2.3 Data analytics

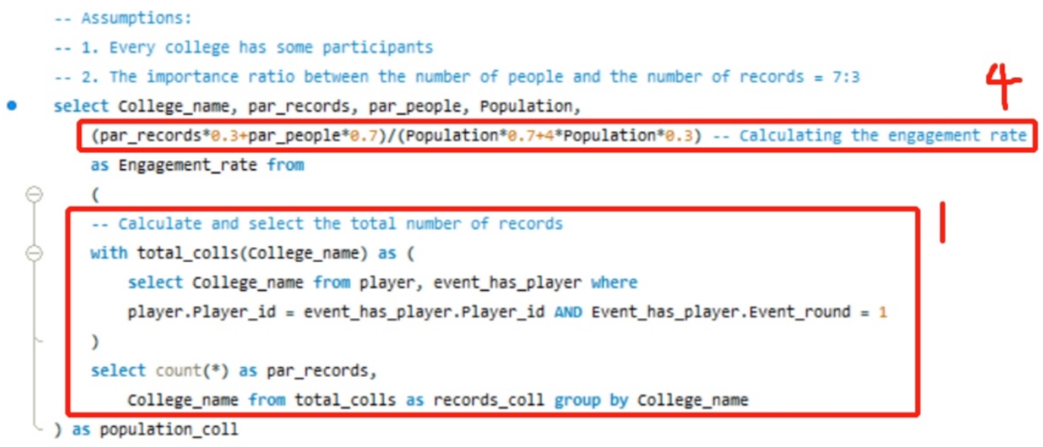
Our group performed the analysis of the engagement of each college in the sport's festival with our database using MySQL queries. The engagement rate (briefly known as "engagement" in the following) indicates the vitality of a particular college in the university. Colleges take it as an important reference to decide whether or not and how to encourage more students to participate in the sport's events so as to get a better reputation for the college. Typically for a college manager, the number of participants would be a more important factor in determining the engagement of a college than the number of participations, since a college manager would like more students in the college to participate in multiple events (meaning popularity) instead of only a few students participating in all of the events (meaning outstanding individuals but silent majority).

As a result, the engagement of a college is determined by multiple factors like the number of participants and the number of participation records. We used the formula shown in figure 2.1 to calculate the engagement, using the weighted sum of the number of participants and the number of records normalized by the maximum engagement index, which is exactly the maximum weighted sum. We assume that the ratio of importance between the registration records and the number of participants is 3/7. That is, we assign 0.3 to W_1 and 0.7 to W_2, respectively. If the engagement is large, we conclude that students in that college are actively involved in the sport's festival. If the engagement is low, we conclude that the students are passive on the sport's festival and the college needs to encourage more of its students to participate in the events.

$$\text{Engagement (of a college)} = \frac{W_1 \cdot \#\,\text{registration records} + W_2 \cdot \#\,\text{students}}{\text{Maximum engagement index}}$$

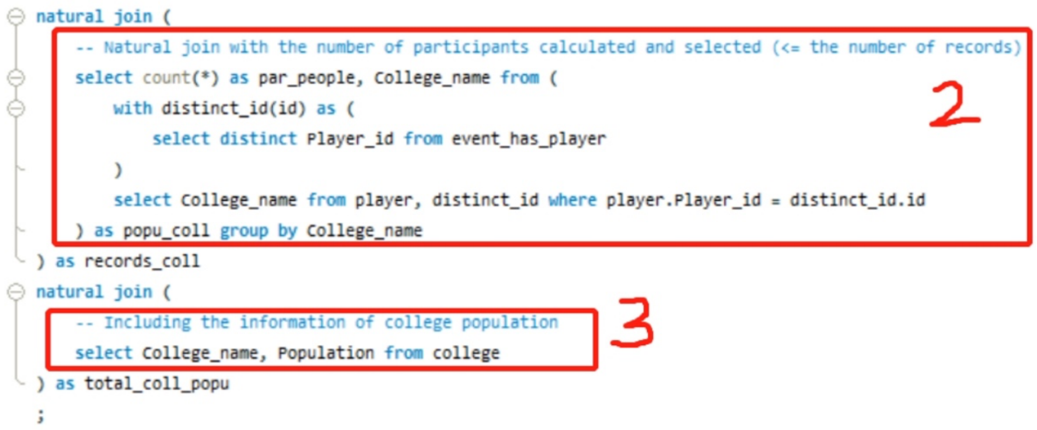
Figure 2.1: Formula for college engagement

Our group implemented the measurement of engagement using MySQL queries show in figure 2.2 and figure 2.3. In the first part, our group calculated and selected the total number of registration records and grouped them by the College_name attribute. In the second part, our group natural joined the previous result with the number of participants calculated and selected and also grouped by the College_name attribute. In the third part, our group included the college population in the table. In the fourth part, our group calculate the engagement and store it under an attribute named “Engagement_rate”.



```
-- Assumptions:
-- 1. Every college has some participants
-- 2. The importance ratio between the number of people and the number of records = 7:3
select College_name, par_records, par_people, Population,
    (par_records*0.3+par_people*0.7)/(Population*0.7+4*Population*0.3) -- Calculating the engagement rate
    as Engagement_rate from
    (
    -- Calculate and select the total number of records
    with total_colls(College_name) as (
        select College_name from player, event_has_player where
        player.Player_id = event_has_player.Player_id AND Event_has_player.Event_round = 1
    )
    select count(*) as par_records,
        College_name from total_colls as records_coll group by College_name
) as population_coll
```

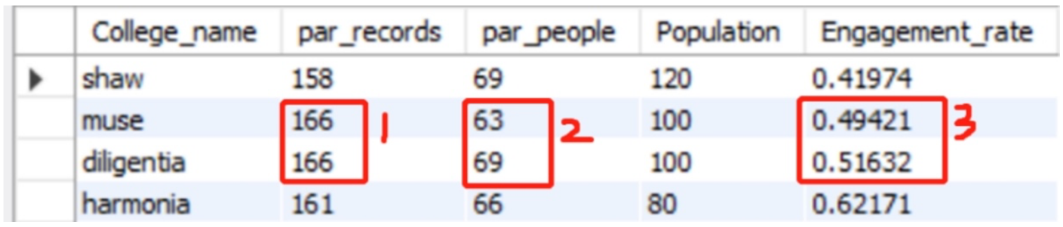
Figure 2.2: Calculation of engagement using MySQL queries (section 1)



```
natural join (
    -- Natural join with the number of participants calculated and selected (<= the number of records)
    select count(*) as par_people, College_name from (
        with distinct_id(id) as (
            select distinct Player_id from event_has_player
        )
        select College_name from player, distinct_id where player.Player_id = distinct_id.id
    ) as popu_coll group by College_name
) as records_coll
natural join (
    -- Including the information of college population
    select College_name, Population from college
) as total_coll_popu
;
```

Figure 2.3: Calculation of engagement using MySQL queries (section 2)

The sample analytic results are shown in figure 2.4. our group can compare the results from Muse and Diligentia to see the reasonability of giving the criteria of engagement. Muse and Diligentia share the same number of participating records (shown in block 1), however, the number of participants in Diligentia is higher than that of Muse (shown in block 2), indicating that more students in Diligentia are involved in the sport's events. As a result, we treat Diligentia to have a higher engagement rate (shown in block 3).



College_name	par_records	par_people	Population	Engagement_rate
shaw	158	69	120	0.41974
muse	166	63	100	0.49421
diligentia	166	69	100	0.51632
harmonia	161	66	80	0.62171

Figure 2.4: Sample analytic results

3. Conclusion

In this project, our group designs a management database for CUHK(SZ) sports event. This database can efficiently manage information of all players, events, volunteers, organization committee, referees, and so on. Finally, the schema is reduced to 3NF, which greatly reduces data redundancy and improves data integrity. Some sample SQL queries are also created for practical use, such as selecting top players into the next round and updating college ranking automatically. These functions are implemented on the website demo and verified correctly. In the data analytics part, our group analyzes the engagement of each college in the sports event. In conclusion, the designed database is comprehensive and efficient. It has the potential to monitor the data of large-scale sports events.

4. Self-evaluation

The highlights of the database are:

- The database is reduced to 3NF, which greatly reduces data redundancy and improves data integrity.
- The front-end website contacts with the database closely. The website is not just data visualization. But users can use it to operate the database. For example, players

can register and select the events. The organization committee can update the event results. The total college score can update automatically. These operations are all implemented by interacting with the database

- Data analytics is rooted in real situations and proposes a new model to deal with this problem. Our group performs the analysis of the engagement of each college in the sports meeting using MySQL queries.

The basic functions of this database have been developed. But there is still room for improvement:

- A match function needs to be implemented to match the available time of volunteers with the event time. The function is to automatically assign volunteers for each event.
- More operations of organization committee can be implemented. Currently, the organization committee only supports updating the event results. Other operations, like initializing the events, can also be developed.

5. Member Contribution

Database design: all group members

Sample SQL queries (back-end): Wang Yupeng, Wang Yuhao, Qin Peiran, Xian Nan

Front-end website: Dai Yulong, Qin Peiran

Data analytics, Xiao Nan

All members actively participate in this project and have the same contribution (20%).

Appendix

The attached files are as follow:

Presentation PowerPoint: 3170_project_pre.pptx

Data Insertion File: insert_data.sql

(insert data into database)

Create Database File: create_database.sql

Django Implementation File (both back-end and front-end): sport folder

Data Analytics SQL File: Engagement_analytics.sql

Website Demo Video: web_demo