itucsdb Documentation

Release 1.0

Team Name

December 25, 2015

Team itucsdb1525

Members

- Member 1
- Cemal Türkoğlu
- Member 3
- Ercan Alp Serteli
- Member 5

project description goes here (a few paragraphs)

Contents:

CHAPTER

ONE

USER GUIDE

explain how your application works from the user perspective, use screenshots whereever appropriate

to add a picture, use the following example:

```
.. figure:: picture.png
   :scale: 50 %
   :alt: map to buried treasure

   This is the caption of the figure (a simple paragraph).
```

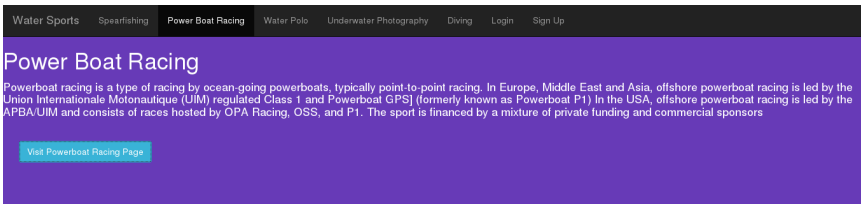
1.1 Parts Implemented by Member Name

1.2 Parts Implemented by Cemal Türkoğlu

The parts that I have implemented are the *Power Boat Racing* pages and functionalities related to users.

1.2.1 Power Boat Racing

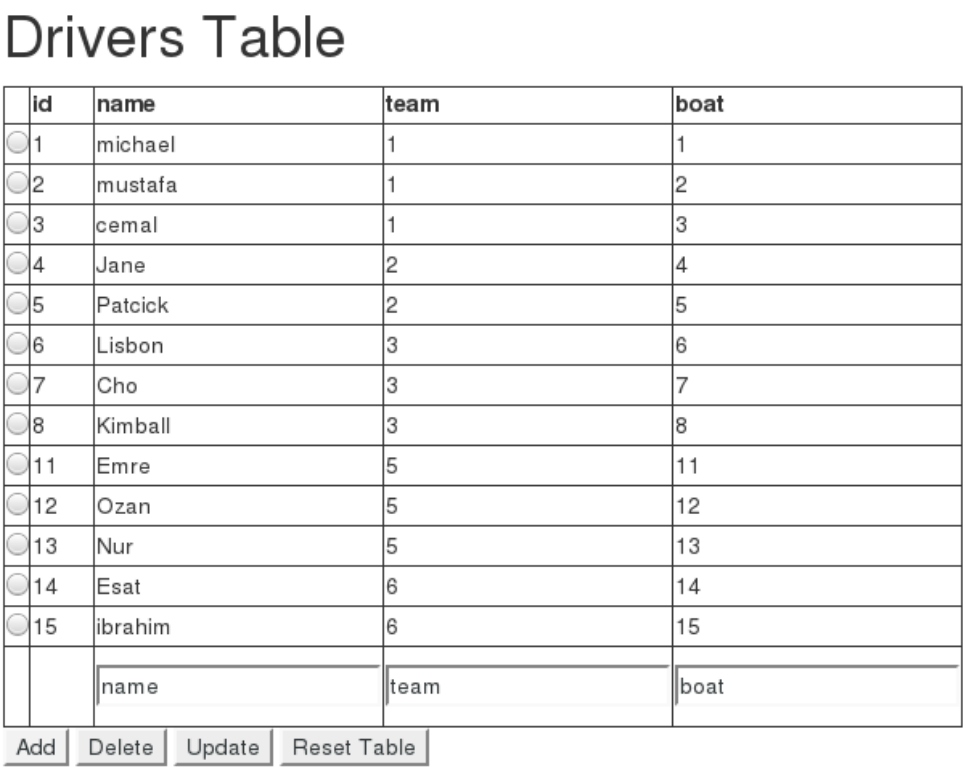
The power boat racing pages can be reached by clicking this button in the home page.



Power boat racing pages locates under /drivers route. In the main page you can see some complex statistics and all tables. There are 5 table: Drivers,Teams,Boats,League15,OldRaces. In table operation you can make add,delete,update and resetting table

Adding

To add a new driver, you need to enter a name,team and boat using the textboxes.Also click the *Add* button.



As a result, the new driver you entered will be added to the database and the new list will be shown to you.

Deleting

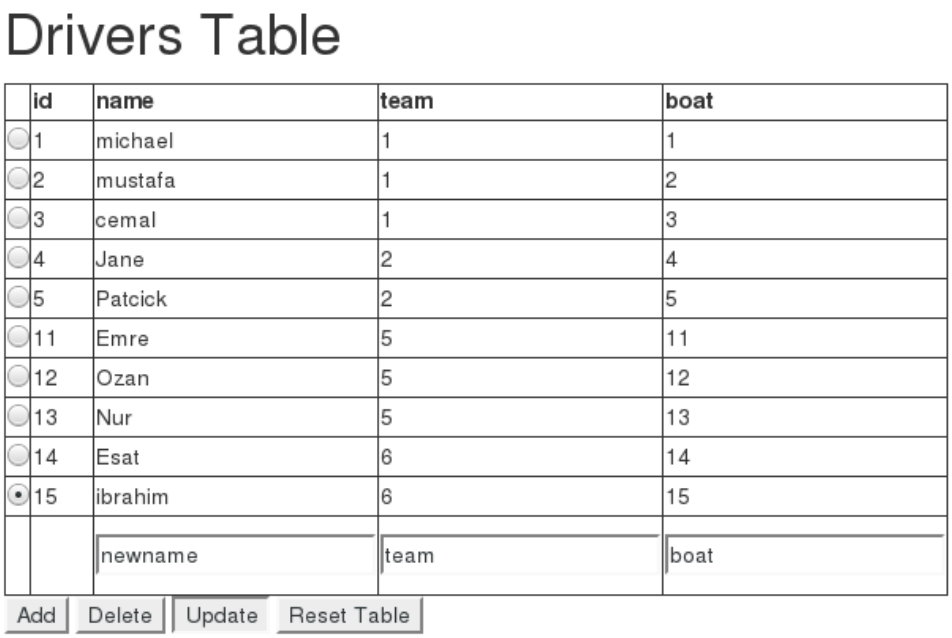
To delete a driver, click the circle to its left in the list. Then press the *Delete* button.

Note: If you try to break referencial integrity , it will not be allowed.

Then, the entry will be removed from the database and the resulting list will be displayed.

Updating

To update the information of a competitor, select the comptitor in the same manner as deleting (by clicking the circle to its left) and then enter the information as you would in adding. After that, click the *Update* button and watch it happen.



Information in the entry will be updated and shown back.

> **Note:** Again breaking referencial integrity is not allowed

Resetting the Table

Clicking the *Reset Table* button reverts any changes done to both the competitors and the teams table and fills them with default values. Not much has to be said about this function.

Tables

All tables are listed in main page.Also from the left menu one of the table can be selected to display.
Drivers

Drivers Table

	id	name	team	boat
○	1	michael	1	1
○	2	mustafa	1	2
○	3	cemal	1	3
○	4	Jane	2	4
○	5	Patcick	2	5
○	6	Lisbon	3	6
○	7	Cho	3	7
○	8	Kimball	3	8
○	11	Emre	5	11
○	12	Ozan	5	12
○	13	Nur	5	13
○	14	Esat	6	14
○	15	ibrahim	6	15
		name	team	boat

Add Delete Update Reset Table

Teams

Teams Table

id	name	captain	championship
1	Lions	1	20
2	Rose	4	18
5	BigBoys	13	12
6	Sacremento	15	10

Boats

Boats Table

id	driver	speed
1	1	165
2	2	158
3	3	140
4	4	150
5	5	140
11	11	140
12	12	122
13	13	143
14	14	144
15	15	115

League

League 2015 Table

pos	boat	points
1	1	1580
3	2	1400
5	3	1300
6	5	1275
7	4	1250
9	11	1233
10	12	1230
11	13	1220
12	14	1210

OldRaces

Old Races Table

race id	class	year	location	winner
1001	1	1999	Istanbul	1
1002	1	2003	Paris	4
1004	3	2004	Berlin	1
1005	1	2005	Barcelona	1
1008	3	2009	Bursa	5
1009	2	2010	Madrid	4
10010	1	2010	Manchester	3

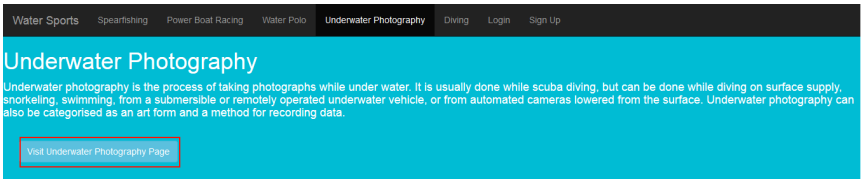
1.3 Parts Implemented by Member Name

1.4 Parts Implemented by Ercan Alp Serteli

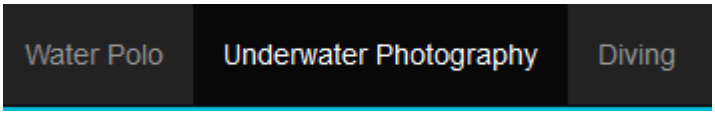
The parts that I have implemented are the *Underwater Photography* pages and functionalities related to users.

1.4.1 Underwater Photography

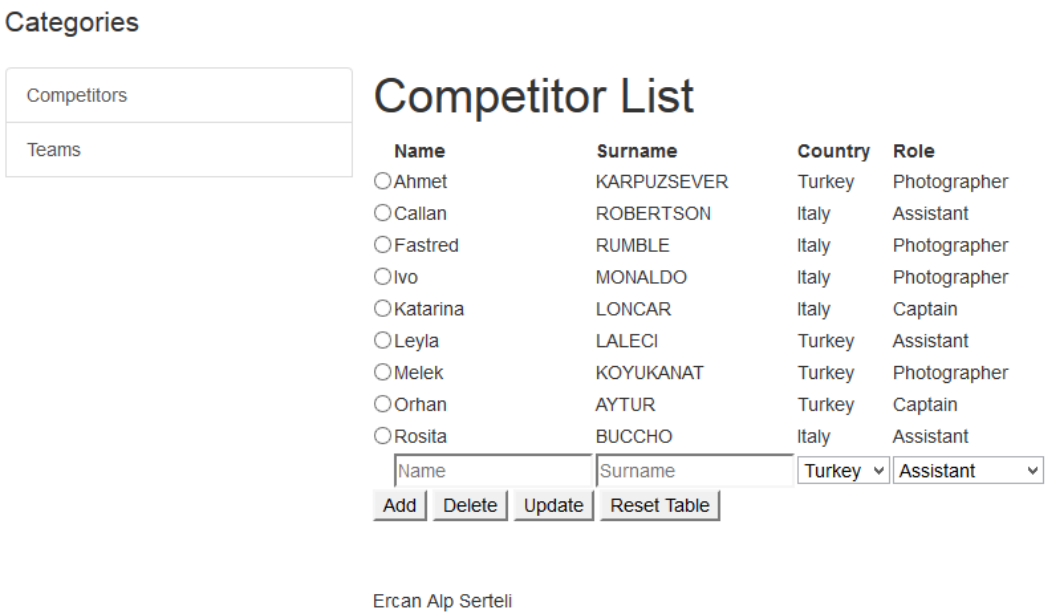
The underwater photography pages can be reached by clicking this button in the home page.



Or from this menu item in the navigation bar from any other page.



Now that you clicked one of them, you are in the *Competitors* page of Underwater Photography.



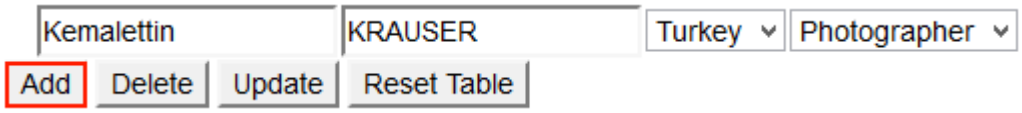
From the *Categories* menu you see on the left, it is possible to navigate between the two pages of Underwater Photography: *Competitors* and *Teams*

What you see in the middle is a representation of the underlying Competitors database table. It lists the information of the existing competitors in the table and allows anyone to modify the table via adding, updating or deleting entries or via resetting the table to its default state.

The entries are sorted according to the alphabetical order of their first names.

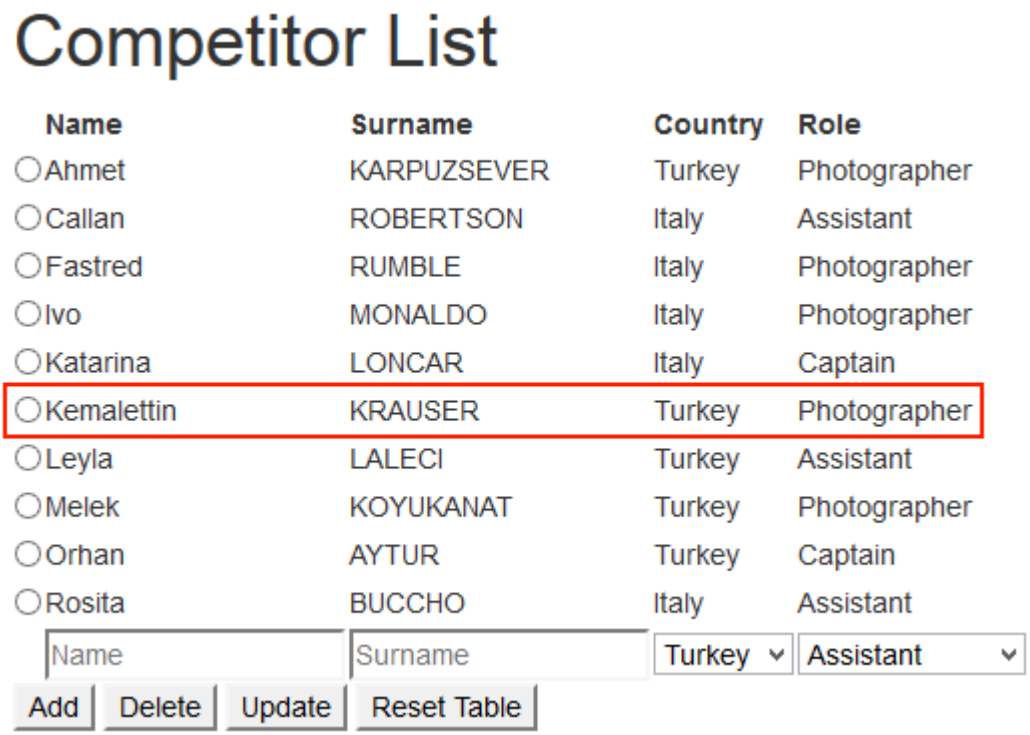
Adding

To add a new competitor, you need to enter a name and a surname using the textboxes. Then you need to select a country and a role using the dropdown menus and click the *Add* button.



Note: To add a competitor from a team of some non-existing country, you first need to add the team from the *Teams* page. On the other hand, the team roles are fixed. They cannot be changed and a new role cannot be added.

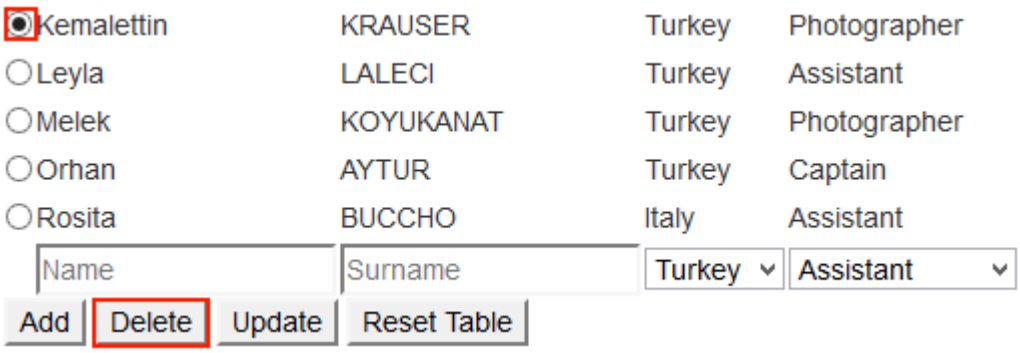
As a result, the new competitor you entered will be added to the database and the new list will be shown to you.



Note: If you leave the name or the surname boxes empty, it will not be added and the page will show you a warning message

Deleting

To delete a competitor, click the circle to its left in the list. Then press the *Delete* button.



Then, the entry will be removed from the database and the resulting list will be displayed.

Updating

To update the information of a competitor, select the comptitor in the same manner as deleting (by clicking the circle to its left) and then enter the information as you would in adding. After that, click the *Update* button and watch it happen.

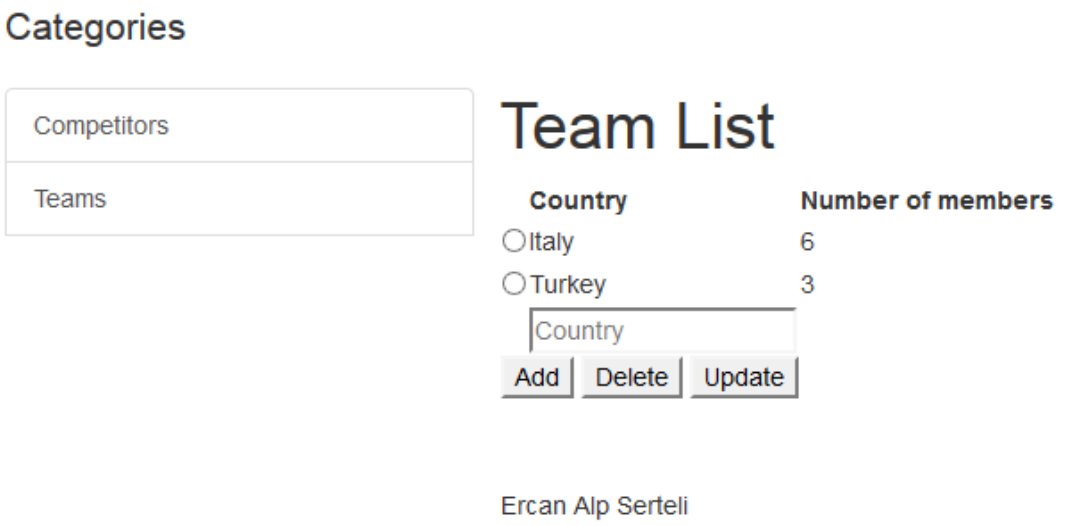
Information in the entry will be updated and shown back.

Resetting the Table

Clicking the *Reset Table* button reverts any changes done to both the competitors and the teams table and fills them with default values. Not much has to be said about this function.

Teams Page

When you click the *Teams* button in the left menu, you will end up in this page.



What you see is the same structure as the Competitors page, but with a smaller table. That is the teams list. Teams only have a *Country* field and an ID, since competitions are held with national teams. In fact, the competitors list you have seen before is taking its country information from this table.

All the operations as in the Competitors page are possible here as well, and they all work the same.

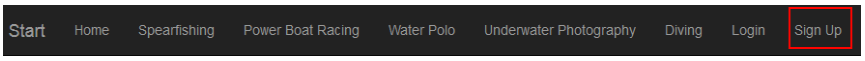
Note: If you try to delete a team that has members in the competitors list, you cannot do it. To do this, delete the members from the Competitors page first.

1.4.2 User Operations

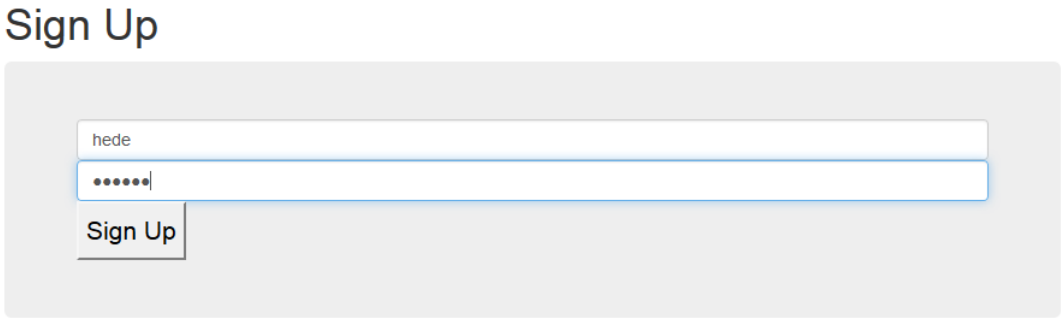
The website supports basic user functionalities such as signing up, logging in, logging out. Modifying the information of users is also possible for admin type accounts.

Signing up

You can move to the sign up page by clicking the *Sign Up* button from the navigation bar.



In the sign up page, enter a username and a password and press the *Sign Up* button. This will enable you to login using those credentials. You will be redirected to the Login page, since you will probably want to login after signing up.

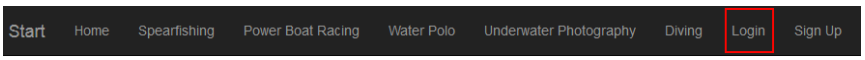


Note: If you try to sign up with a blank username or password, the page will warn you. Also a different warning message will be shown if you enter an username that already exists in the system.

> **Warning:** Do not use a real password to sign up because it is not secure, at all. Literally anyone can see your information if they want to.

Logging in

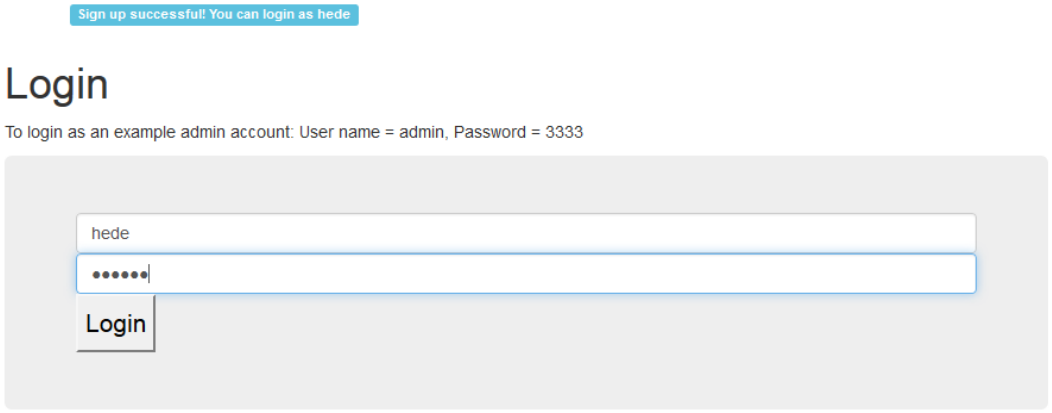
You can get to the login page by clicking the *Login* button from the navigation bar.



You can also get redirected here by signing up.

You will be given a default admin account's credentials in case you want to try out being an admin. You can login using that or your own credentials.

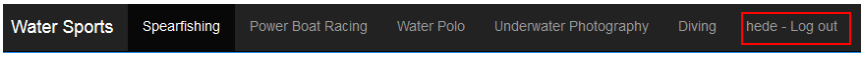
Note: Any user who signs up is a *User* type user. To change a user's type, you have to be logged in as an *Admin* type user.



Note: If you try to login using wrong information, the page will show you a message and let you try again.

Logging out

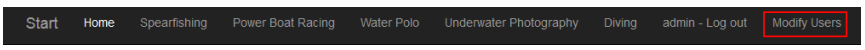
Once you are logged in, a new item appears in the navigation bar while the sign up and login items disappear. This new button lets you log out from the system.



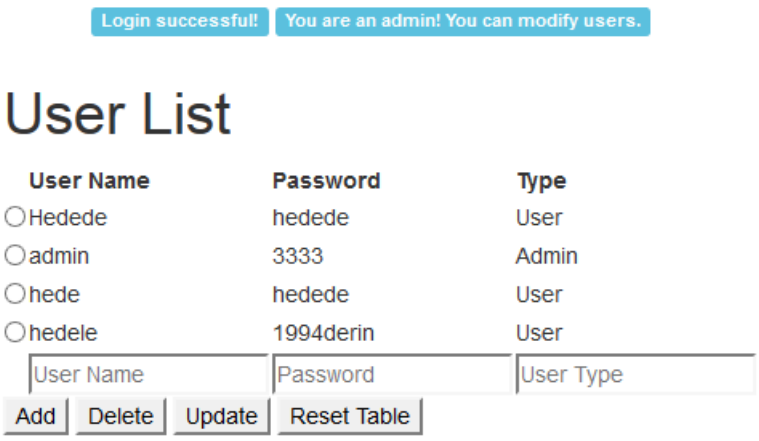
Once you click it, you will no longer be logged in.

Users Page

If you login as an admin, you will be directed to the *Users* page. You can also go there using the navigation bar item that only shows up if you are an admin.



In the users page, you will see the list of users. You can add, delete or update users or you can reset the users table. The operations are identical to the ones in the Underwater Photography pages.



1.5 Parts Implemented by Turker Unlu

1.6 Divers

This section of the page is used for managing information about dive sporters.

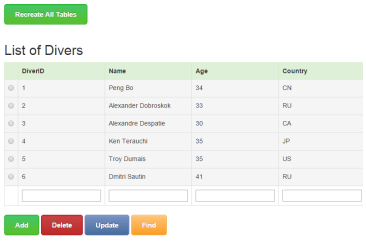


Fig. 1.1: Fig. 1.1: Screenshot of Divers Table

Information about divers is listed on the screen. Data table consist of ID, name, age and country information of the sporter. ID is unique sporter ID of a sporter. Name and age are sporter's personal information. Country is the country sporter represents.

Operations add, delete, update and find can be done from this section alone.

1.7 Competitions

This section of the page is used for managing information about diving competitions.

Fig. 1.2: Fig. 1.2: Screenshot of Competition Table

Information about diving competitions is listed on the screen. Data table consist of competition ID, winner sporter ID and the year which competition is done. CompetitionID is the unique ID of a diving competition. WinnerID is the ID of the sporter who won the competition. Year is the date which the competition held.

Operations add, delete, update and find can be done from this section alone. The sporter with the ID number winnerID must exist in order to add a new competition.

1.8 Records

This section of the page is used for managing information about a sporter record at a specific competition. For example sporter "DiverID" jumped from 5 meter at competition "CompetitionID"

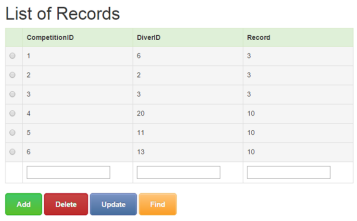


Fig. 1.3: Fig. 1.3: Screenshot of Record Table

Information about record is listed on the screen. Data table consist of competitionID, DiverID and record. CompetitionID is the unique ID of the competition. DiverID is the unique ID of the sporter. Record is the amount of height sporter dived from.

Operations add, delete, update and find can be done from this section alone. The sporter with the ID number DiverID and competition with the CompetitionID must exist in order to add a new record.

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Database Design

explain the database design of your project

include the E/R diagram(s)

2.2 Code

explain the technical structure of your code

to include a code listing, use the following example:

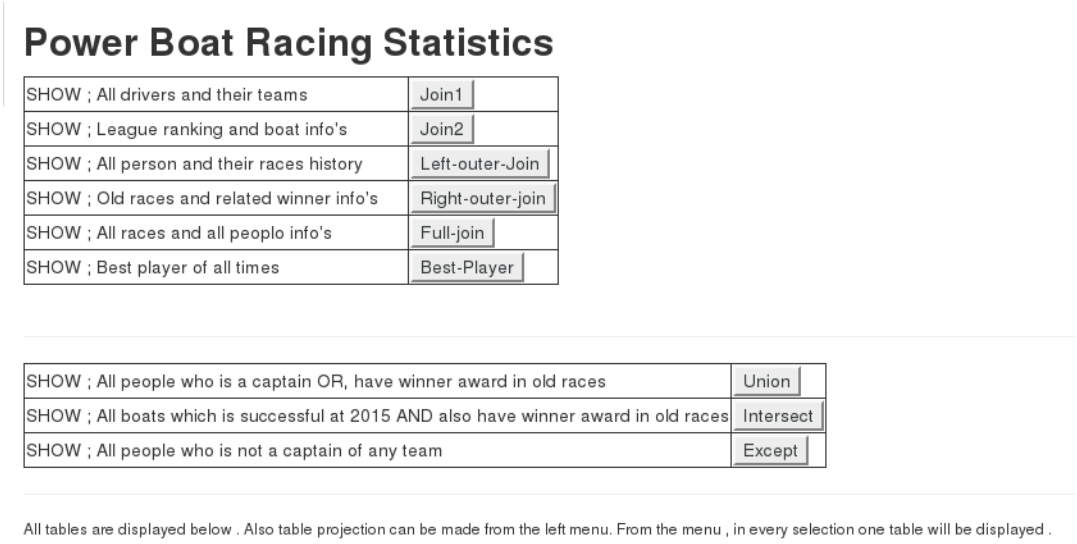
```
.. code-block:: python

   class Foo:

      def __init__(self, x):
         self.x = x
```

2.2.1 Parts Implemented by Member Name

2.2.2 Parts Implemented by Cemal Türkoğlu

Statistics Operations

The power boat racing page supply some statistics for user .



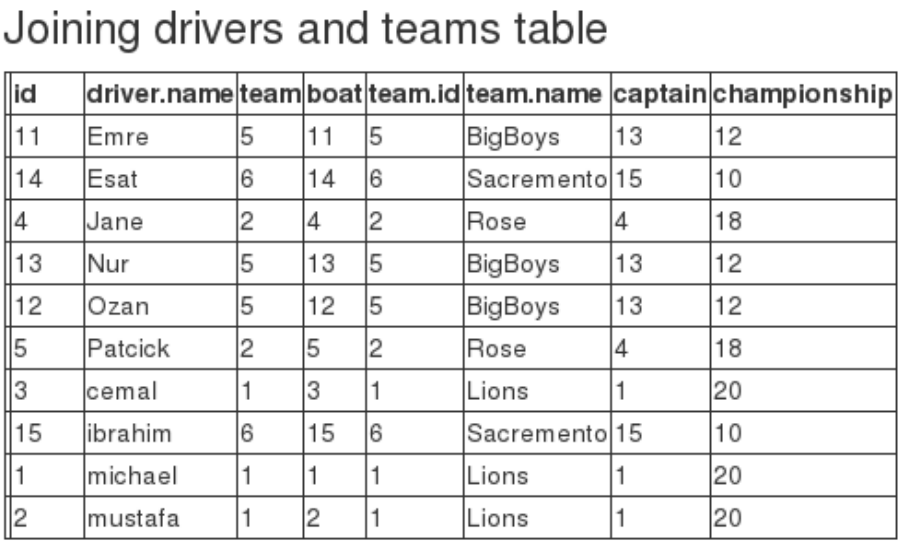
Power Boat Racing Statistics

SHOW ; All drivers and their teams	Join1
SHOW ; League ranking and boat info's	Join2
SHOW ; All person and their races history	Left-outer-Join
SHOW ; Old races and related winner info's	Right-outer-join
SHOW ; All races and all peoplo info's	Full-join
SHOW ; Best player of all times	Best-Player

SHOW ; All people who is a captain OR, have winner award in old races	Union
SHOW ; All boats which is successful at 2015 AND also have winner award in old races	Intersect
SHOW ; All people who is not a captain of any team	Except

All tables are displayed below . Also table projection can be made from the left menu. From the menu , in every selection one table will be displayed .

All Drivers And Their Teams



Joining drivers and teams table

id	driver.name	team	boat	team.id	team.name	captain	championship
11	Emre	5	11	5	BigBoys	13	12
14	Esat	6	14	6	Sacremento	15	10
4	Jane	2	4	2	Rose	4	18
13	Nur	5	13	5	BigBoys	13	12
12	Ozan	5	12	5	BigBoys	13	12
5	Patcick	2	5	2	Rose	4	18
3	cemal	1	3	1	Lions	1	20
15	ibrahim	6	15	6	Sacremento	15	10
1	michael	1	1	1	Lions	1	20
2	mustafa	1	2	1	Lions	1	20

The code:

```
def join1(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT * FROM drivers JOIN teams ON (drivers.team=teams.id)
                    """
        cursor.execute(query)
        joinfirst = cursor.fetchall()
        cursor.close()
    return render_template('PowerBoat/join1.html',joinfirst = sorted(joinfirst , key=lambda p: p[1]))
```

League ranking and boat info's

Joining league 2015 and boats tables.

pos	league15.boat	point	boats.id	driver	speed
1	1	1580	1	1	165
3	2	1400	2	2	158
5	3	1300	3	3	140
6	5	1275	5	5	140
7	4	1250	4	4	150
9	11	1233	11	11	140
10	12	1230	12	12	122
11	13	1220	13	13	143
12	14	1210	14	14	144

The code:

```
def join2(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT * FROM league15 JOIN boats ON (league15.boat=boats.id)
                    """
        cursor.execute(query)
        joinsec = cursor.fetchall()
        cursor.close()
    return render_template('PowerBoat/join2.html',joinsec=joinsec)
```

All person and their races history

Outer LEFT Joining the drivers and oldraces .

id	name	team	driver.boat	race_id	class	year	location	winner
1	michael	1	1	1001	1	1999	Istanbul	1
4	Jane	2	4	1002	1	2003	Paris	4
1	michael	1	1	1004	3	2004	Berlin	1
1	michael	1	1	1005	1	2005	Barcelona	1
5	Patcick	2	5	1008	3	2009	Bursa	5
4	Jane	2	4	1009	2	2010	Madrid	4
3	cemal	1	3	10010	1	2010	Manchester	3
11	Emre	5	11	None	None	None	None	None
12	Ozan	5	12	None	None	None	None	None
2	mustafa	1	2	None	None	None	None	None
15	ibrahim	6	15	None	None	None	None	None
13	Nur	5	13	None	None	None	None	None
14	Esat	6	14	None	None	None	None	None

The code:

```
def Ljoin(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT * FROM drivers LEFT OUTER JOIN oldraces ON (drivers.id=oldraces.winner)
                    """
        cursor.execute(query)
        join = cursor.fetchall()
        cursor.close()
        direction = "LEFT"
    return render_template('PowerBoat/join3.html',join=join,direction=direction)
```

Old races and related winner info's

developer/images/cemal/rigthj.png

The code:

```
def Rjoin(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT * FROM drivers RIGHT OUTER JOIN oldraces ON (drivers.id=oldraces.winner)
                    """
        cursor.execute(query)
        join = cursor.fetchall()
        cursor.close()
        direction = "RIGHT"
    return render_template('PowerBoat/join3.html',join=join,direction=direction)
```

All races and all peoplo info's

Outer FULL Joining the drivers and oldraces .

id	name	team	driver.boat	race_id	class	year	location	winner
1	michael	1	1	1001	1	1999	Istanbul	1
4	Jane	2	4	1002	1	2003	Paris	4
1	michael	1	1	1004	3	2004	Berlin	1
1	michael	1	1	1005	1	2005	Barcelona	1
5	Patcick	2	5	1008	3	2009	Bursa	5
4	Jane	2	4	1009	2	2010	Madrid	4
3	cemal	1	3	10010	1	2010	Manchester	3
11	Emre	5	11	None	None	None	None	None
12	Ozan	5	12	None	None	None	None	None
2	mustafa	1	2	None	None	None	None	None
15	ibrahim	6	15	None	None	None	None	None
13	Nur	5	13	None	None	None	None	None
14	Esat	6	14	None	None	None	None	None

The code:

```
def Fjoin(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT * FROM drivers FULL JOIN oldraces ON (drivers.id=oldraces.winner)
                    """
        cursor.execute(query)
        join = cursor.fetchall()
        cursor.close()
        direction = "FULL"
    return render_template('PowerBoat/join3.html',join=join,direction=direction)
```

Best player of all times

Best player ever

* best player

Id	Name	Total championship
1	michael	3

The code:

```
def best(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """ SELECT winner,name,count(*) from oldraces join drivers on (winner=id)
                    group by winner,name
                    having count(*) = (SELECT count(*) from oldraces
                                       GROUP BY winner
                                       order by count(*) DESC limit 1 offset 0
                                       )
                """
        cursor.execute(query)
        best = cursor.fetchall()
        cursor.close()
    return render_template('PowerBoat/best.html',best=best)
```

All people who is a captain OR, have winner award in old races

Union

* hem takım kaptanı olan hemde eski yarışlarda derecesi olan kişiler

id	Person
15	ibrahim
5	Patcick
1	michael
13	Nur
4	Jane
3	cemal

The code:

```
def union(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """SELECT id,name from drivers where id in (SELECT winner FROM oldraces
                                                            UNION
                                                            SELECT captain FROM teams)
                """
        cursor.execute(query)

        union = cursor.fetchall()

        cursor.close()
    return render_template('PowerBoat/union.html',union=union)
```

All boats which is successful at 2015 AND also have winner award in old races

Intersect

(first ten boat of league 15 list) INTERSECT (oldraces join drivers {boat})
* yapılan işlemde hem bu sene ligde başarılı hemde eski yarışlarda derecesi olan * kişilerin bulunmasıdır.

boat
4
1
5
3

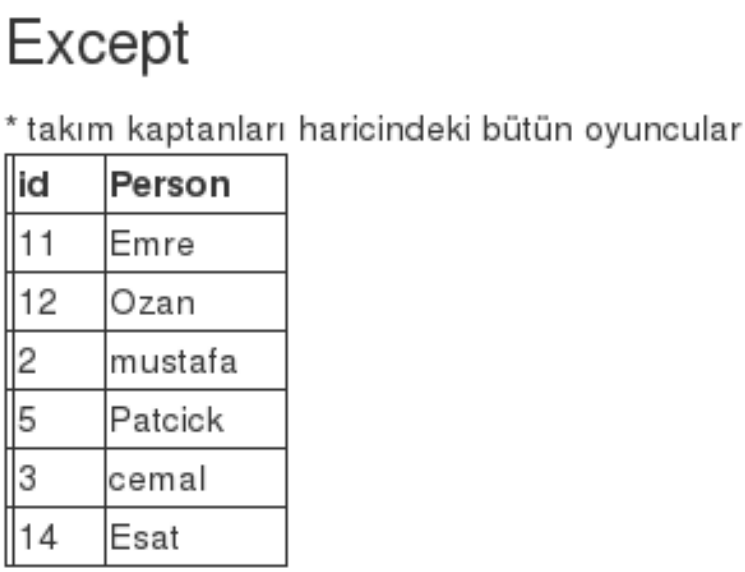
(first 3 boat of league15 list) INTERSECT (oldraces join drivers {boat})
* bu işlemde ise bu sene ligde ilk 3'te olup eski yarışlarda da * dereceye girmiş boat'lar listelendi.

boat
1

The code:

```
def intersect(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """SELECT drivers.boat FROM oldraces JOIN drivers ON (oldraces.winner=drivers.id)
                INTERSECT
                SELECT boat FROM league15
                """
        cursor.execute(query)

        intersect = cursor.fetchall()

        query = """SELECT drivers.boat FROM oldraces JOIN drivers ON (oldraces.winner=drivers.id)
                INTERSECT
                SELECT boat FROM league15 where pos in (1,2,3)
                """
        cursor.execute(query)
        intersect2 = cursor.fetchall()
        cursor.close()
    return render_template('PowerBoat/intersect.html',intersect=intersect,intersect2=intersect2)
```

All people who is not a captain of any team



id	Person
11	Emre
12	Ozan
2	mustafa
5	Patcick
3	cemal
14	Esat

The code:

```
def minus(self):
    with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = """SELECT id,name FROM drivers where id in (SELECT id from drivers
                                                            EXCEPT
                                                            SELECT captain FROM teams)
                """
        cursor.execute(query)

        minus = cursor.fetchall()

        cursor.close()
    return render_template('PowerBoat/minus.html',minus=minus)
```

2.2.3 Parts Implemented by Member Name

2.2.4 Parts Implemented by Ercan Alp Serteli

I have implemented three database tables: competitors, teams and users. The first two tables are regarding the Underwater Photography and they are managed by the *UWP* class. The users table is managed by the *Users* class.

Here, I will explain everything technical that is related to these entities including SQL code, Python code and HTML code.

UWP Class

The *UWP* class includes all the methods that are needed to manage the *competitor* and *team* tables

init Method

```
    def __init__(self, dsn):
      self.dsn = dsn
      with dbapi2.connect(self.dsn) as connection:
```

```
            cursor = connection.cursor()
            query =  """DO $$
                        BEGIN
                            IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'role') THEN
                                CREATE TYPE role AS ENUM ('Photographer', 'Assistant', 'Captain');
                            END IF;
                        END$$"""
            cursor.execute(query)

            query = """CREATE TABLE IF NOT EXISTS team (
                        id serial PRIMARY KEY,
                        country text NOT NULL)"""
            cursor.execute(query)

            query = """CREATE TABLE IF NOT EXISTS competitor (
                        id serial PRIMARY KEY,
                        name text NOT NULL,
                        surname text NOT NULL,
                        role role NOT NULL,
                        team_id integer REFERENCES team (id))"""
            cursor.execute(query)
        return
```

This is the method that gets called when the UWP object is created. It gets the dsn by a parameter and saves it in its field for later use. Then it attempts to create the role type and the two tables if they do not exist. These SQL statements do nothing if the database has been initialized properly. They are there in case the application might be needed to run in a new database or some of the tables are dropped for some reason.

show_page_comp Method

```
    def show_page_comp(self):
      with dbapi2.connect(self.dsn) as connection:
            cursor = connection.cursor()

            query = """SELECT competitor.id, name, surname, country, role FROM competitor, team
                        WHERE (team_id = team.id)"""
            cursor.execute(query)
            competitors = cursor.fetchall()

            query = "SELECT country FROM team"
            cursor.execute(query)
            countries = []
            for row in cursor:
                countries.append(row[0])
            query = "SELECT DISTINCT role FROM competitor"
            cursor.execute(query)
            roles = []
            for row in cursor:
                roles.append(row[0])

        return render_template('uwp/uwp_comp.html', competitors = sorted(competitors, key=lambda p:
                               countries = countries, roles = roles)
```

This method is called when a HTTP GET is requested for the competitors page. It gets the data that is needed to construct the competitor list from the database and renders the page with that data as the parameter. It sorts the list by the first names before using it. In the first select statement, we get the data of all competitors by joining the competitor and team tables over team_id. This way, we can display the countries that each competitor belongs to as well as their other attributes. The second select statement is used to get all the countries from the team table so that they can be shown in a dropdown menu when adding or updating rows. The last select statement gets the roles that are used in the competitor table for a dropdown menu again. I just noticed that this is a very bad way of doing this. It should get them from the type instead of the table. However it is too late to fix this now.

show_page_team Method

```
    def show_page_team(self):
      with dbapi2.connect(self.dsn) as connection:
          cursor = connection.cursor()

          query = """SELECT team.id, country, COUNT(competitor.id)
                      FROM team LEFT JOIN competitor  ON team_id = team.id
                      GROUP BY team.id"""

          cursor.execute(query)
          teams = cursor.fetchall()

      return render_template('uwp/uwp_teams.html', teams = teams)
```

This method is for showing the teams page. It gets the team data and the number of competitors in each team by joining the team and competitor tables. Left join is used because we want to get all the teams in the list whether they have any members or not.

add_competitor Method

```
    query = """INSERT INTO competitor (name, surname, role, team_id)
                    SELECT %s, %s, %s, team.id
                    FROM team WHERE team.country = %s
                    """
    cursor.execute(query, (name, surname, role, country))
```

Note: From now on, in most methods only a relevant part of the method is pasted

This method is used to insert a row into the competitor table. The team_id needed is selected from team using the country.

delete_competitor Method

```
    query = "DELETE FROM competitor WHERE id = %s"
    cursor.execute(query, (id,))
```

This is a very simple method as it only gets an id and deletes the row with that id from the table.

update_competitor Method

```
    query = """UPDATE competitor
              SET name=%s,surname=%s,role=%s,team_id=subquery.id
              FROM (SELECT team.id FROM team WHERE team.country = %s)
              AS subquery
              WHERE competitor.id = %s"""
    cursor.execute(query, (name, surname, role, country, id))
```

This method updates a row in the competitor table with the new data. It uses the name, surname and role inputs directly while it gets the team_id using a subquery that uses the country input.

add_team Method

```
    query = """INSERT INTO team (country)
              VALUES (%s)"""
    cursor.execute(query, (country,))
```

Self explanatory method that adds a new row to the team table.

update_team Method

```
    query = """UPDATE team
                SET country = %s
                WHERE id = %s"""
    cursor.execute(query, (country, id))
```

Self explanatory method that updates new row in the team table.

delete_team Method

```
    query = """SELECT team.id, country, COUNT(competitor.id)
                FROM team LEFT JOIN competitor  ON team_id = team.id
                WHERE team.id = %s GROUP BY team.id
                """
    cursor.execute(query, (id,))
    id, country, members = cursor.fetchone()

    if members != 0:
        flash("You cannot delete a team that has members in the competitor list!")
    else:
        query = "DELETE FROM team WHERE id = %s"
        cursor.execute(query, (id,))
```

This method is used for deleting a row from the team table. However, it has to check if there are any members of the team before attempting to delete it. So, it gets the number of members in a team in the same way show_page_team does. Then checks if the number is 0. If it is, then it is deleted. Otherwise, it pops a flash message and reloads the page.

reset_table Method

```
    query = """DROP TABLE IF EXISTS competitor;
                DROP TABLE IF EXISTS team;
                DROP TYPE IF EXISTS role;"""
    cursor.execute(query)

    query = "CREATE TYPE role AS ENUM ('Photographer', 'Assistant', 'Captain')"
    cursor.execute(query)

    query = """CREATE TABLE team (
                id serial PRIMARY KEY,
                country text NOT NULL)"""
    cursor.execute(query)

    query = """CREATE TABLE competitor (
                id serial PRIMARY KEY,
                name text NOT NULL,
                surname text NOT NULL,
                role role NOT NULL,
                team_id integer REFERENCES team (id))"""
    cursor.execute(query)

    query = """INSERT INTO team (country)
                VALUES
                ('Turkey'),
                ('Italy')
                """
```

```
    cursor.execute(query)

    query = """INSERT INTO competitor (name, surname, role, team_id)
                VALUES
                ('Ahmet', 'KARPUZSEVER', 'Photographer', 1),
                ('Orhan', 'AYTUR', 'Captain', 1),
                ('Melek', 'KOYUKANAT', 'Photographer', 1),
                ('Leyla', 'LALECI', 'Assistant', 1),
                ('Ivo', 'MONALDO', 'Photographer', 2),
                ('Rosita', 'BUCCHO', 'Assistant', 2),
                ('Callan', 'ROBERTSON', 'Assistant', 2),
                ('Katarina', 'LONCAR', 'Captain', 2),
                ('Fastred', 'RUMBLE', 'Photographer', 2)
                """
    cursor.execute(query)
```

This method drops the team and competitor tables, creates them again and fills them with default data.

Users Class

This class has all the methods that are needed for managing the users table.

init Method

```
    def __init__(self, dsn):
      self.dsn = dsn
      with dbapi2.connect(self.dsn) as connection:
          cursor = connection.cursor()

          query = """CREATE TABLE IF NOT EXISTS users(
                      id serial PRIMARY KEY,
                      username text UNIQUE NOT NULL,
                      password text NOT NULL,
                      type text NOT NULL)"""
          cursor.execute(query)

          query = """INSERT INTO users (username, password, type)
                      SELECT 'admin', '3333', 'Admin'
                      WHERE
                      NOT EXISTS (
                          SELECT username FROM users WHERE username = 'admin'
                      )"""
          cursor.execute(query)

          connection.commit()
      return
```

The init method saves the dsn in a field and attempts to create the users table if it does not exists. Then it tries to insert an admin account into it if one does not exist.

delete_user Method

```
    query = "DELETE FROM users WHERE id = %s"
    cursor.execute(query, (id,))
```

This is a self explanatory method that deletes a row from the users table by an id.

update_user Method

```
    query = """UPDATE users
                SET username=%s,password=%s,type=%s
                WHERE id = %s"""
    cursor.execute(query, (username, password, type, id))
```

This is a self explanatory method that updates a row in the users table with the given data.

add_user Method

```
    query = """SELECT username FROM users
                WHERE username = %s"""
    cursor.execute(query, (username,))
    if cursor.fetchall():
        return False
    else:
        query = """INSERT INTO users (username, password, type)
                VALUES (%s, %s, %s)"""
        cursor.execute(query, (username, password, type))
```

This is the method that adds rows into the users table. However, first it checks if there is a row with that username and if there is, it returns false because the username must be unique. If there is no match, the user is added. This is a lower level method that is called through higher level methods.

add_user_from_table Method

```
    def add_user_from_table(self, username, password, type):
      if self.add_user(username, password, type):
          flash("User added!")
          return redirect(url_for('users'))
      else:
          flash("There is already a user with that name! Use a different name")
          return redirect(url_for('users'))
```

This method is used to try and add a user from the users page by an admin. It calls the add_user method explained above.

sign_up Method

```
    if self.add_user(username, password, 'User'):
        flash("Sign up successful! You can login as "+ username)
        return redirect(url_for('login'))
    else:
        flash("There is already a user with that name! Use a different name")
        return redirect(url_for('sign_up'))
```

This method is used to try and add a user from the sign up page. It calls the add_user method.

login Method

```
    def login(self, username, password):
      with dbapi2.connect(self.dsn) as connection:
          cursor = connection.cursor()

          query = """SELECT id,type FROM users
                      WHERE username = %s and password = %s
```

```
                    """
        cursor.execute(query, (username, password))

        connection.commit()
        user = cursor.fetchone()
    if user:
        session['username'] = username
        flash("Login successful!")
        if(user[1] == 'Admin'):
            session['admin'] = True
            flash("You are an admin! You can modify users.")
            return redirect(url_for('users'))
        else:
            session['admin'] = False
            return redirect(url_for('home'))
    else:
        flash("User name or password is wrong!")
        return redirect(url_for('login'))
```

This method handles the logging in functionality. It executes a select statement and checks if anything returned. If so, it assigns the username to the 'username' key of the session and checks if the user type is admin. If it is, then the 'admin' key of the session is assigned true. These are the values that are used to determine which user can do/see what.

If nothing is returned from the select, a flash message is shown to the user.

show_users Method

```
    def show_users(self):
      with dbapi2.connect(self.dsn) as connection:
        cursor = connection.cursor()

        query = "SELECT id, username, password, type FROM users"
        cursor.execute(query)
        users = cursor.fetchall()

      return render_template('user/users.html', users = sorted(users, key=lambda p: p[1]))
```

This is simply the method that is called when an admin wants to open the users page. All users are selected and sorted according to their usernames.

reset_table Method

```
    query = "DROP TABLE IF EXISTS users"
    cursor.execute(query)

    query = """CREATE TABLE users(
                id serial PRIMARY KEY,
                username text UNIQUE NOT NULL,
                password text NOT NULL,
                type text NOT NULL)"""
    cursor.execute(query)

    query = """INSERT INTO users (username, password, type)
                VALUES ('admin', '3333', 'Admin')"""
    cursor.execute(query)
```

This method drops the users table and creates it again. Then it adds the default admin user to the table.

Methods in server.py

server.py is the main part of our application. It is the file where the methods that flask calls when it gets a request, are in. These methods create instances of a class and call its methods according to the type of the request.

uwp_comp Method

```
@app.route('/underwater_photography/competitors', methods=['GET', 'POST'])
def uwp_comp():
    dsn = app.config['dsn']
    page = UWP(dsn)
    if request.method == 'GET':
        return page.show_page_comp()
    elif 'Add' in request.form:
        name = request.form['Name']
        surname = request.form['Surname']
        country = request.form['Country']
        role = request.form['Role']
        if name and surname:
            return page.add_competitor(name, surname, country, role)
        else:
            flash("You need to enter a name and a surname!")
            return page.show_page_comp()
    elif 'Delete' in request.form:
        id = request.form.get('select', '')
        if id:
            return page.delete_competitor(id)
        else:
            flash("You need to select the competitor you want to delete using the radio butto
            return page.show_page_comp()

    elif 'Update' in request.form:
        id = request.form.get('select', '')
        name = request.form['Name']
        surname = request.form['Surname']
        country = request.form['Country']
        role = request.form['Role']
        if id and name and surname:
            return page.update_competitor(name, surname, country, role, id)
        else:
            if not id:
                flash("You need to select the competitor you want to update using the radio b
            if not name or not surname:
                flash("You need to enter a name and a surname!")
            return page.show_page_comp()
    elif 'Reset' in request.form:
        return page.reset_table()
```

This method handles the competitor page requests. If the HTTP method is GET, it calls show_page_comp(). If it is POST, then it determines the kind (add/update/delete/reset), gets the required inputs from the form fields and does some input checking before calling the appropriate function. In add, it checks if the name and surname boxes are not empty, in delete it checks if any one of the rows is selected and in update it checks both of those conditions.

uwp_team Method

```
@app.route('/underwater_photography/teams', methods=['GET', 'POST'])
def uwp_team():
    dsn = app.config['dsn']
    page = UWP(dsn)
    if request.method == 'GET':
```

```
            return page.show_page_team()
        elif 'Add' in request.form:
            country = request.form['Country']
            if country:
                return page.add_team(country)
            else:
                flash("You need to enter a country name!")
                return page.show_page_team()
        elif 'Delete' in request.form:
            id = request.form.get('select', '')
            if id:
                return page.delete_team(id)
            else:
                flash("You need to select the team you want to delete using the radio buttons!")
                return page.show_page_team()
        elif 'Update' in request.form:
            id = request.form.get('select', '')
            country = request.form['Country']
            if country and id:
                return page.update_team(country, id)
            else:
                if not id:
                    flash("You need to select the team you want to update using the radio buttons
                if not country:
                    flash("You need to enter a country name!")
                return page.show_page_team()
```

This method handles the team page requests. It works almost identically to the uwp_comp method, so check it out for any explanations.

users method

```
    @app.route('/users', methods=['GET', 'POST'])
    def users():
        dsn = app.config['dsn']
        page = Users(dsn)
        if session['admin']:
            if request.method == 'GET':
                return page.show_users()
            elif 'Add' in request.form:
                username = request.form['UserName']
                password = request.form['Password']
                type = request.form['UserType']
                if username and password and type:
                    return page.add_user_from_table(username, password, type)
                else:
                    flash("You need to enter all the fields")
                    return page.show_users()
            elif 'Delete' in request.form:
                id = request.form.get('select', '')
                if id:
                    return page.delete_user(id)
                else:
                    flash("You need to select the user you want to delete using the radio buttons
                    return page.show_users()
            elif 'Update' in request.form:
                id = request.form.get('select', '')
                username = request.form['UserName']
                password = request.form['Password']
                type = request.form['UserType']
                if username and password and type and id:
```

```
                    return page.update_user(username, password, type, id)
                else:
                    if not id:
                        flash("You need to select the user you want to update using the radio but
                    if not username or not password or not type:
                        flash("You need to enter all the fields")
                    return page.show_users()
            elif 'Reset' in request.form:
                return page.reset_table()

        else:
            flash("Only admins can access that page!")
            return redirect(url_for('home'))
```

This method handles the users page requests. First thing it does is check if the user is an admin. If it is, then the method works similarly to the uwp_comp method. Otherwise, the user is redirected to the home_page.

logout Method

```
    @app.route('/logout', methods=['GET'])
    def logout():
        flash("Logged out!")
        session.pop('username', None)
        session.pop('admin', None)
        return redirect(url_for('home'))
```

This method simply logs the user out.

login Method

```
    @app.route('/login', methods=['GET', 'POST'])
    def login():
        dsn = app.config['dsn']
        page = Users(dsn)
        if request.method == 'GET':
            return render_template('user/login.html')
        else:
            username = request.form['username']
            password = request.form['password']
            return page.login(username, password)
```

This is the method that handles the login page requests. If the request is GET, it justs renders the html template. Otherwise it is a POST, so it gets the username and password and calls the login method of Users class.

sign_up Method

```
    @app.route('/signup', methods=['GET', 'POST'])
    def sign_up():
        dsn = app.config['dsn']
        page = Users(dsn)
        if request.method == 'GET':
            return render_template('user/signup.html')
        else:
            username = request.form['username']
            password = request.form['password']
            if username and password:
                return page.sign_up(username, password)
            else:
```

```
                flash("You must enter a username and a password!")
                return render_template('user/signup.html')
```

This is the method that handles the sign up page requests. It is similar to the login method but it checks if the username or password boxes are empty before calling the sign_up method of Users.

HTML Templates

These are the files that are rendered to create what the user sees in their browser. They are templates because they have parameters which we can fill up in run time.

uwp_comp.html

```
    {% block body %}
    <h1> Competitor List </h1>
    <form action="{{ url_for('uwp_comp') }}" method="post">
       <table>
          <tr>
             <th> </th>
             <th>Name</th>
             <th>Surname</th>
             <th>Country</th>
             <th>Role</th>
          </tr>
          {% for id, name, surname, country, role in competitors %}
             <tr>
                <td> <input type="radio" name="select"value="{{id}}" /> </td>
                <td> {{name}} </td>
                <td> {{surname}} </td>
                <td> {{country}} </td>
                <td> {{role}} </td>
             </tr>
          {% endfor %}
          <tr>
             <td> </td>
             <td> <input type="text" name="Name" value="" placeholder="Name" /> </td>
             <td> <input type="text" name="Surname" value="" placeholder="Surname"/> </td>
             <td> <select name="Country">
                {% for country in countries %}
                  <option value="{{country}}">{{country}}</option>
                {% endfor %}
                </select>
             </td>
             <td> <select name="Role">
                 {% for role in roles %}
                  <option value="{{role}}">{{role}}</option>
                {% endfor %}
                </select>
             </td>
          </tr>
       </table>
       <input type="submit" name="Add"value="Add" />
       <input type="submit" name="Delete"value="Delete" />
       <input type="submit" name="Update"value="Update" />
       <input type="submit" name="Reset"value="Reset Table" />
    </form>

    <footer>Ercan Alp Serteli</footer>
    {% endblock %}
```

This is the main body of the template for the underwater photography competitors page. There is a table that is filled with the values from *competitors*. Also the values of country and role selections are taken from parameters. Name and surname data is taken using text boxes and add, update, delete, reset table operations are determined by checking which submit has been pressed.

uwp_teams.html

```
{% block body %}
<h1> Team List </h1>
<form action="{{ url_for('uwp_team') }}" method="post">
   <table>
      <tr>
         <th> </th>
         <th>Country</th>
         <th>Number of members</th>
      </tr>
      {% for id, country, num_mem in teams %}
         <tr>
            <td> <input type="radio" name="select"value="{{id}}" /> </td>
            <td> {{country}} </td>
            <td> {{num_mem}} </td>
          </tr>
      {% endfor %}
      <tr>
         <td> </td>
         <td> <input type="text" name="Country" value="" placeholder="Country"/>    </td>
       </tr>
   </table>
   <input type="submit" name="Add"value="Add" />
   <input type="submit" name="Delete"value="Delete" />
   <input type="submit" name="Update"value="Update" />
</form>

<footer>Ercan Alp Serteli</footer>
{% endblock %}
```

This is the template for the underwater photography teams page. It works very similarly to the uwp_comp page

login.html

```
<h1>Login</h1>
<p> To login as an example admin account: User name = admin, Password = 3333 </p>
<div class="jumbotron">
<form class="form-login" action="{{ url_for('login') }}" method="post">
    <label for="username" class="sr-only">Username</label>
    <input type="username" name="username" id="username" class="form-control" placeholder="Us
    <label for="password" class="sr-only">Password</label>
    <input type="password" name="password" id="password" class="form-control" placeholder="Pa
    <input type="submit" name="Login" value="Login" />
</form>
</div>
```

This is the template for the login page. Bootstrap classes are used for styling. A simple form with two text boxes and a submit button.

signup.html

```
<h1>Sign Up</h1>
<div class="jumbotron">
<form class="form-signup" action="{{ url_for('sign_up') }}" method="post">
    <label for="username" class="sr-only">Username</label>
    <input type="username" name="username" id="username" class="form-control" placeholder="Us
    <label for="password" class="sr-only">Password</label>
    <input type="password" name="password" id="password" class="form-control" placeholder="Pa
    <input type="submit" name="SignUp" value="Sign Up" />
</form>
</div>
```

This is the template for the signup page. Almost identical to the login page.

users.html

```
<h1> User List </h1>
<form action="{{ url_for('users') }}" method="post">
   <table>
      <tr>
         <th> </th>
         <th>User Name</th>
         <th>Password</th>
         <th>Type</th>
      </tr>
      {% for id, username, password, type in users %}
         <tr>
            <td> <input type="radio" name="select"value="{{id}}" /> </td>
            <td> {{username}} </td>
            <td> {{password}} </td>
            <td> {{type}} </td>
          </tr>
      {% endfor %}
      <tr>
         <td> </td>

         <td><input type="text" name="UserName" value="" placeholder="User Name" /></td>
         <td> <input type="text" name="Password" value="" placeholder="Password"/> </td>
         <td> <input type="text" name="UserType" value="" placeholder="User Type"/> </td>
       </tr>
   </table>
   <input type="submit" name="Add"value="Add" />
   <input type="submit" name="Delete"value="Delete" />
   <input type="submit" name="Update"value="Update" />
   <input type="submit" name="Reset"value="Reset Table" />
</form>
```

This is the template for the users page. This is the template for the underwater photography teams page. It works very similarly to the uwp_comp and uwp_team pages

2.2.5 Parts Implemented by ANIL YILDIRIM

Database Tables

DIVER table is the main table for diving section, contains sporters. COMPETITION table refers to DIVER table, contains diving competitions. RECORD table refers to both DIVER and COMPETITION tables, contains sporter records.

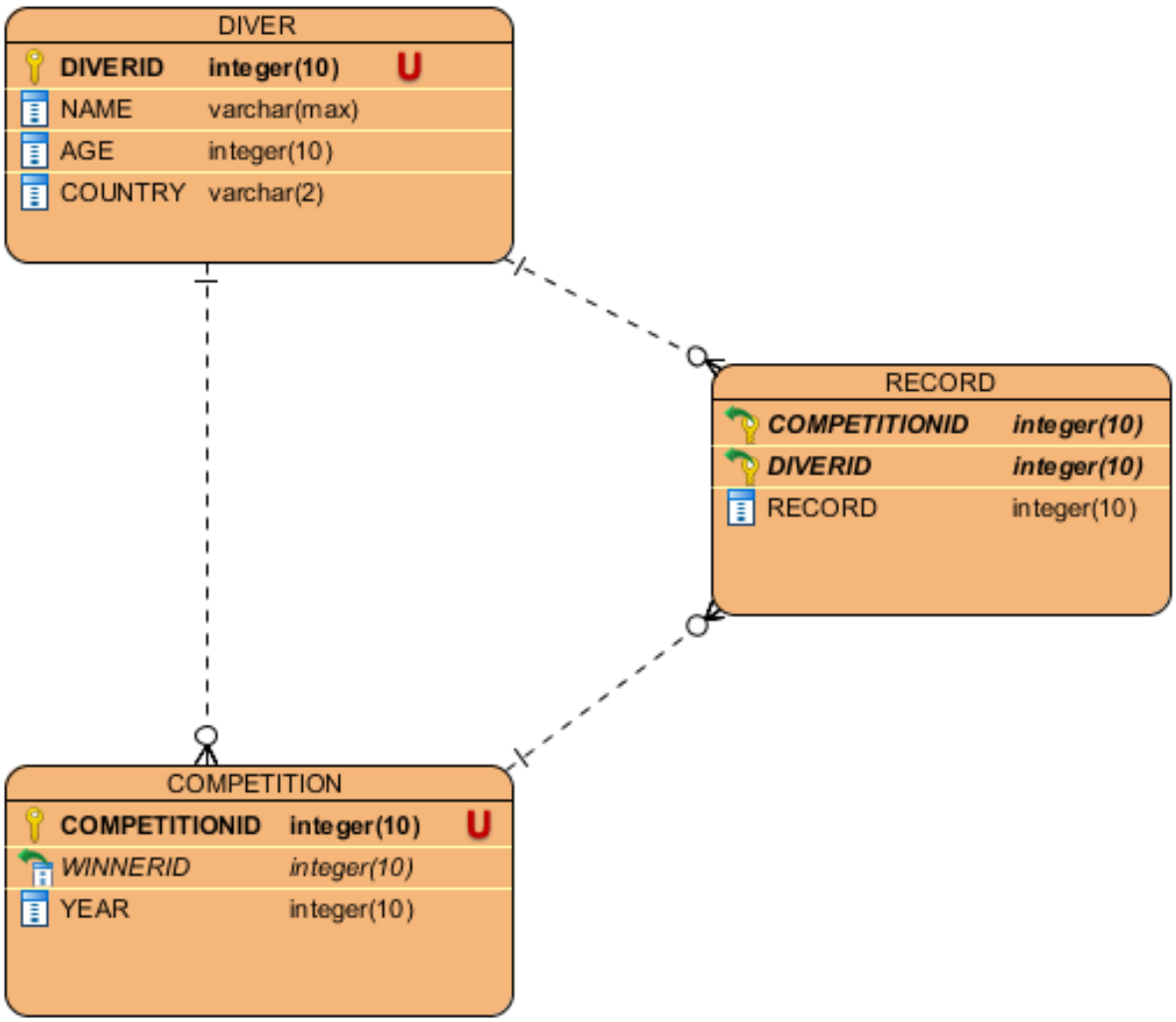


Fig. 2.1: Fig. 1: Entity Relationship Diagram for the DIVER, COMPETITION and RECORD Tables

DIVER:

```
CREATE TABLE IF NOT EXISTS DIVER (
     DIVERID INTEGER PRIMARY KEY NOT NULL,
     NAME text,
     AGE INTEGER,
     COUNTRY text)
```

DIVER table contains information about diving sporters.It has 1 major column which is primary key and 3 other information columns. DIVER table is independent from other tables, it references no other table.

DIVERID is the unique ID of the sporter. NAME is the name of the sporter. AGE is the age of the sporter. COUNTRY is the country which sporter represents.

COMPETITION:

```
CREATE TABLE IF NOT EXISTS COMPETITION (
     COMPETITIONID INTEGER PRIMARY KEY NOT NULL,
     WinnerID INTEGER REFERENCES DIVER(DIVERID) ON DELETE RESTRICT ON UPDATE CASCADE,
     YEAR INTEGER)
```

COMPETITION table contains information about diving competitions. References to DIVER table due to its foreign key WinnerID.

COMPETITIONID is the unique ID of a diving competition. WINNERID is the ID of the sporter who won the competition. YEAR is the date which the competition held.

RECORD:

```
CREATE TABLE IF NOT EXISTS RECORD (
      COMPETITIONID INTEGER REFERENCES COMPETITION(COMPETITIONID) ON DELETE RESTRICT ON UPDATE CA
      DIVERID INTEGER REFERENCES DIVER(DIVERID) ON DELETE RESTRICT ON UPDATE CASCADE,
      RECORD INTEGER DEFAULT NULL,
      PRIMARY KEY(COMPETITIONID, DIVERID))
```

RECORD table contains information about sporters record at a specific competition. It refers to both DIVER table and COMPETITION table via its foreign keys. Primary key is the combination of COMPETITIONID and DIVERID.

CompetitionID is the unique ID of the competition. DiverID is the unique ID of the sporter. Record is the amount of height sporter dived from.

Class codes of Tables from diving.py:

```
class Diver:
 def __init__(self, diverID, name, age, country):
     self.diverID = diverID
     self.name = name
     self.age = age
     self.country = country


class Competition:
 def __init__(self, competitionID, winnerID, year):
     self.competitionID = competitionID
     self.winnerID = winnerID
     self.year = year

class Record:
```

```
 def __init__(self, competitionID, diverID, record):
     self.competitionID = competitionID
     self.diverID = diverID
     self.record = record
```

Diving category codes from server.py file:

add1, update1, delete1 and find1 refers to DIVER table. add2, update2, delete2 and find2 refers to COMPETITION table. add3, update3, delete3 and find3 refers to RECORD table.

```
@app.route('/diving', methods=['GET', 'POST'])
def diving():
 ds = DiverStore(app.config['dsn'])

 if request.method == 'GET':
     return ds.firstrun()
 else:
     if 'add1' in request.form:
         diverID = request.form['id']
         name = request.form['name']
         age = request.form['age']
         country = request.form['country']
         return ds.add_diver(Diver(diverID, name, age, country), 1)
     elif 'add2' in request.form:
         competitionID = request.form['competitionID']
         winnerID = request.form['winnerID']
         year = request.form['year']
         return ds.add_diver(Competition(competitionID, winnerID, year), 2)
     elif 'add3' in request.form:
         competitionID = request.form['competitionID']
         diverID = request.form['diverID']
         record = request.form['record']
         return ds.add_diver(Record(competitionID, diverID, record), 3)


     elif 'update1' in request.form:
         diverID = request.form['id']
         name = request.form['name']
         age = request.form['age']
         country = request.form['country']
         searchID = request.form['select']
         return ds.update_diver(Diver(diverID, name, age, country), searchID, 1)
     elif 'update2' in request.form:
         competitionID = request.form['competitionID']
         winnerID = request.form['winnerID']
         year = request.form['year']
         searchID = request.form['select']
         return ds.update_diver(Competition(competitionID, winnerID, year), searchID, 2)
     elif 'update3' in request.form:
         competitionID = request.form['competitionID']
         diverID = request.form['diverID']
         record = request.form['record']
         combinedSearchID = request.form['select']
         return ds.update_diver(Record(competitionID, diverID, record), combinedSearchID, 3)


     elif 'find1' in request.form:
         diverID = request.form['id']
         name = request.form['name']
         age = request.form['age']
         country = request.form['country']
```

```
        return ds.find_diver(Diver(diverID, name, age, country), 1)
    elif 'find2' in request.form:
        competitionID = request.form['competitionID']
        winnerID = request.form['winnerID']
        year = request.form['year']
        return ds.find_diver(Competition(competitionID, winnerID, year), 2)
    elif 'find3' in request.form:
        competitionID = request.form['competitionID']
        diverID = request.form['diverID']
        record = request.form['record']
        return ds.find_diver(Record(competitionID, diverID, record), 3)


    elif 'recreate' in request.form:
        #  create new tables and add some rows
        ds.recreate()
        return redirect(url_for('diving'))
    elif 'return' in request.form:
        #  return to main diving page
         return redirect(url_for('diving'))
```

Add operation from diving.py file:

```
def add_diver(self, data, table):
    try:
        ValidateInput(data, table)
    except ValueError:
        return render_template('Diving/InvalidValue.html', divers=divers)

    with dbapi2.connect(self.dbf) as connection:
        cursor = connection.cursor()
        query = ""

        if table == 1:
            query = """INSERT INTO DIVER (DIVERID, NAME, AGE, COUNTRY)
                       VALUES ('%s', '%s', '%s', '%s')""" % (data.diverID, data.name, data.age,
        elif table == 2:
            query = """INSERT INTO COMPETITION (COMPETITIONID, WinnerID, YEAR)
                       VALUES ('%s', '%s', '%s')""" % (data.competitionID, data.winnerID, data.
        elif table == 3:
            query = """INSERT INTO RECORD (COMPETITIONID, DIVERID, Record)
                       VALUES ('%s', '%s', '%s')""" % (data.competitionID, data.diverID, data.re

        cursor.execute(query)
        connection.commit()
        return redirect(url_for('diving'))
```

Update operation from diving.py file:

```
def update_diver(self, data, id, table):
    try:
        ValidateInput(data, table)
    except ValueError:
        return render_template('Diving/InvalidValue.html', divers=divers)

    with dbapi2.connect(self.dbf) as connection:
        cursor = connection.cursor()
        query = ""
```

```
        if table == 1:
            query = """UPDATE DIVER
                       SET DIVERID='%s', NAME='%s', AGE='%s', COUNTRY='%s'
                       WHERE (DIVERID = '%s')""" % (data.diverID, data.name, data.age, data.cou
        elif table == 2:
            query = """UPDATE COMPETITION
                       SET COMPETITIONID='%s', WinnerID='%s', YEAR='%s'
                       WHERE (COMPETITIONID = '%s')""" % (data.competitionID, data.winnerID, dat
        elif table == 3:
            ids = id.split('-')
            query = """UPDATE DIVER
                       SET COMPETITIONID='%s', DIVERID='%s', RECORD='%s'
                       WHERE ((COMPETITIONID = '%s') AND (DIVERID = '%s'))""" % (data.competitio

        cursor.execute(query)
        connection.commit()
        return redirect(url_for('diving'))
```

Delete operation from diving.py file:

Deletes a record from database by ID

```
def delete_diver(self, id, table):
    with dbapi2.connect(self.dbf) as connection:
        cursor = connection.cursor()
        query = ""

        if table == 1:
            query = "DELETE FROM DIVER WHERE (DIVERID = '%s')" %(id)
        elif table == 2:
            query = "DELETE FROM COMPETITION WHERE (COMPETITIONID = '%s')" %(id)
        elif table == 3:
            ids = id.split('-')
            query = "DELETE FROM RECORD WHERE ((COMPETITIONID = '%s') AND (DIVERID = '%s'))" %(i

        cursor.execute(query)
        connection.commit()
        return redirect(url_for('diving'))
```

Find operation from diving.py file:

Find operation lets you find a record by using only a part of information. By searching for name “J” you can find names with John and James. For searching operation it is not neccesary to use data for all columns.

```
def find_diver(self, data, table):
    with dbapi2.connect(self.dbf) as connection:
        cursor = connection.cursor()
        query = ""

        if table == 1:
            if data.diverID == '':
                data.diverID = '-1'
            if data.age == '':
                data.age = '-1'

            data.name = data.name.upper()
            data.country = data.country.upper()
            query = """SELECT DIVERID, NAME, AGE, COUNTRY FROM DIVER
                       WHERE (
                       (('%s' = '-1') OR (DIVERID = '%s'))
```

```
                        AND( ('%s' = '' ) OR (UPPER(NAME) LIKE '%s') )
                        AND( ('%s' = '-1') OR (AGE = '%s') )
                        AND( ('%s' = '' ) OR (UPPER(COUNTRY) = '%s') )
                        )"""% (data.diverID, data.diverID, data.name, data.name+'%', data.age, da
            cursor.execute(query)
            divers = cursor.fetchall()
            return render_template('Diving/searchdiver.html', divers=divers)

        elif table == 2:
            if data.competitionID == '':
                data.competitionID = '-1'
            if data.winnerID == '':
                data.winnerID == '-1'
            if data.year == '':
                data.year == '-1'
            query = """SELECT * FROM COMPETITION
                        WHERE (
                        (('%s' = '-1') OR (COMPETITIONID = '%s'))
                        AND( ('%s' = '-1') OR (WinnerID = '%s') )
                        AND( ('%s' = '-1') OR (YEAR = '%s') )
                        )"""% (data.competitionID, data.competitionID, data.winnerID, data.winne
            cursor.execute(query)
            competitions = cursor.fetchall()
            return render_template('Diving/searchcompetition.html', competitions=competitions)

        elif table == 3:
            if data.competitionID == '':
                data.competitionID = '-1'
            if data.diverID == '':
                data.diverID = '-1'
            if data.record == '':
                data.record = '-1'
            query = """SELECT * FROM RECORD
                        WHERE (
                        (('%s' = '-1') OR (COMPETITIONID = '%s'))
                        AND( ('%s' = '-1') OR (DIVERID = '%s') )
                        AND( ('%s' = '-1') OR (RECORD = '%s') )
                        )"""% (data.competitionID, data.competitionID, data.diverID, data.diverI
            cursor.execute(query)
            records = cursor.fetchall()
            return render_template('Diving/searchrecord.html', records=records)
```

Table creations from diving.py file:

```
def create_tables(self):
    with dbapi2.connect(self.dbf) as connection:
        cursor = connection.cursor()
        query = """CREATE TABLE IF NOT EXISTS DIVER (
                    DIVERID INTEGER PRIMARY KEY NOT NULL,
                    NAME text,
                    AGE INTEGER,
                    COUNTRY text)"""
        cursor.execute(query)

        query = """CREATE TABLE IF NOT EXISTS COMPETITION (
                    COMPETITIONID INTEGER PRIMARY KEY NOT NULL,
                    WinnerID INTEGER REFERENCES DIVER(DIVERID) ON DELETE RESTRICT ON UPDATE CASC
                    YEAR INTEGER)"""
        cursor.execute(query)

        query = """CREATE TABLE IF NOT EXISTS RECORD (
```

```
                    COMPETITIONID INTEGER REFERENCES COMPETITION(COMPETITIONID) ON DELETE RESTRIC
                    DIVERID INTEGER REFERENCES DIVER(DIVERID) ON DELETE RESTRICT ON UPDATE CASCA
                    RECORD INTEGER DEFAULT NULL,
                    PRIMARY KEY(COMPETITIONID, DIVERID))"""
        cursor.execute(query)

        connection.commit()
```