UE22CS351A - Database Management System

Project
GrandPrix Hub

Team Members
Vishal Srinivasa - PES1UG22CS701
Vijval M - PES1UG22CS694

Github repo link - https://github.com/Vishal-Srinivasa/grandprix_hub

User Requirement Specification (URS)

Purpose of the Project:
The purpose of this project is to develop a comprehensive web-based application for managing and interacting with a Formula 1 Grand Prix database system, named "Grandprix Hub." This system enables users, maintainers, and administrators to perform operations such as viewing data, modifying records, and fetching championship standings and driver statistics efficiently. The project aims to provide a secure and user-friendly interface leveraging MySQL, Flask for web functionalities, and bcrypt for authentication.

Scope of the Project:
The Grandprix Hub project covers data management and query capabilities for Formula 1 race data. It is designed for different user roles, including maintainers (who perform updates and insertions) and end-users (who access and retrieve statistics). Key functionalities include managing season data, constructors, driver records, race rounds, championship points, and user authentication. This project integrates secure data management protocols and aims to offer comprehensive, real-time data insights for Formula 1 enthusiasts and data maintainers.

Detailed Description:
The project is built with Flask for web interfacing and MySQL for database management. User authentication is handled through bcrypt for secure password management. The application provides various routes for managing user sign-in/up, updating driver statistics, inserting and deleting race data, and viewing records from a central database. The system allows maintainers to update specific tables, while users can access race and championship data based on selected seasons and rounds.

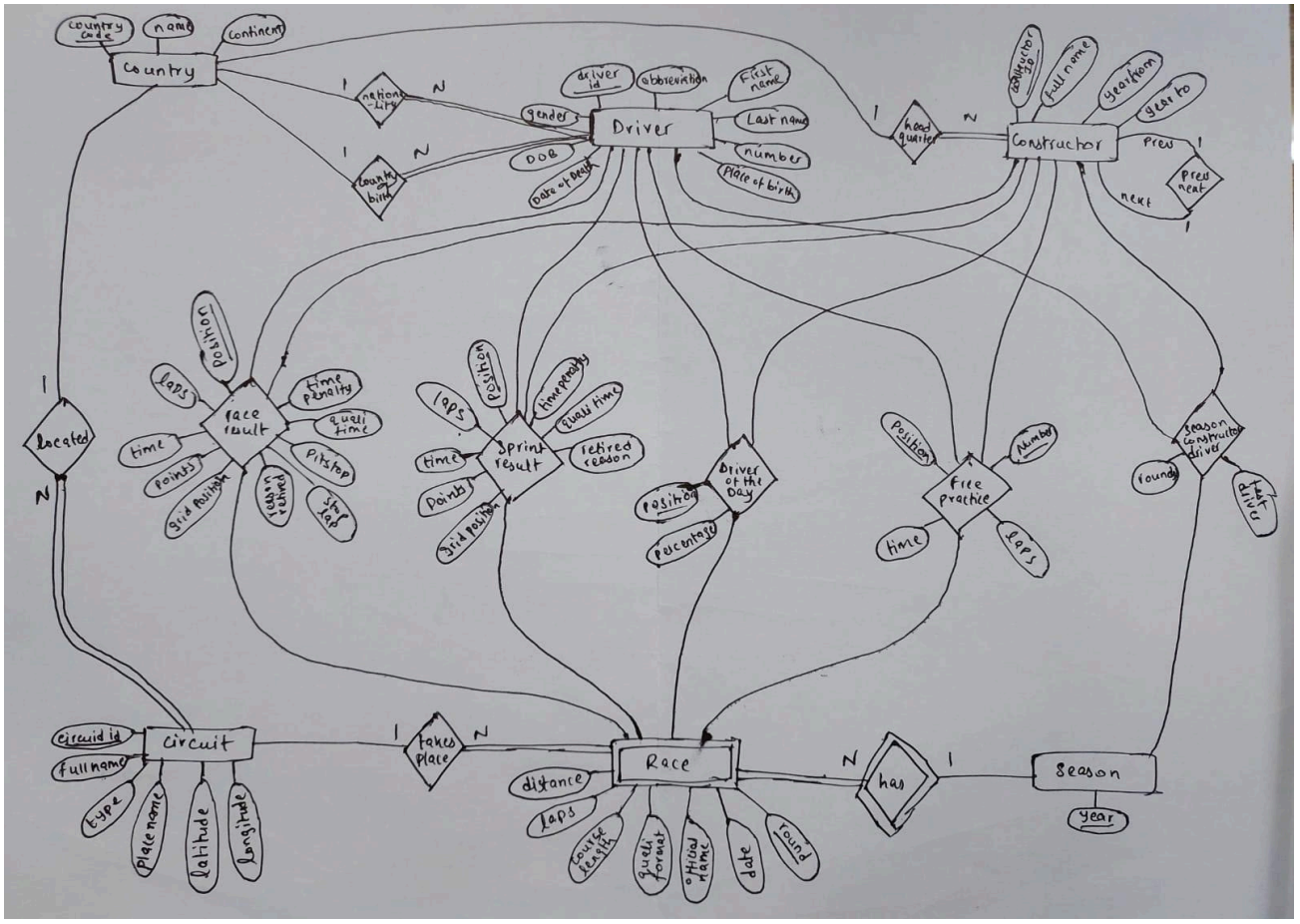
Functional Requirements:

1. **User Authentication and Registration**
    - Description: Provides routes and mechanisms for user and maintainer sign-in/sign-up with secure bcrypt password handling.
2. **Data Viewing**
    - Description: Allows users to view tables such as circuits, constructors, drivers, and season-specific data.
3. **Maintainer Data Insertion**
    - Description: Enables maintainers to select tables and insert new data records through a form-driven interface.
4. **Maintainer Data Update**
    - Description: Maintainers can select a table and update existing data records based on conditions specified in forms.
5. **Data Deletion**
    - Description: Maintainers can delete data entries from tables by specifying conditions via a form interface.
6. **Race Statistics Viewing**
    - Description: Users can view driver standings, constructor standings, race wins, pole positions, podiums, and sprint wins.
7. **Data Filtering by Year and Round**
    - Description: Users can select data for specific years and rounds, viewing results accordingly.
8. **Driver Statistics Retrieval**
    - Description: Users can fetch detailed race statistics and achievements for specific drivers.
9. **Stored Procedure Integration**
    - Description: The application leverages MySQL stored procedures for calculating standings and statistics efficiently.
10. **User Interface Rendering**
    - Description: Provides HTML templates for displaying data views, inserting forms, updating records, and more.
11. **Secure Environment Variable Management**
    - Description: Sensitive keys and passwords are stored in environment variables to ensure security.

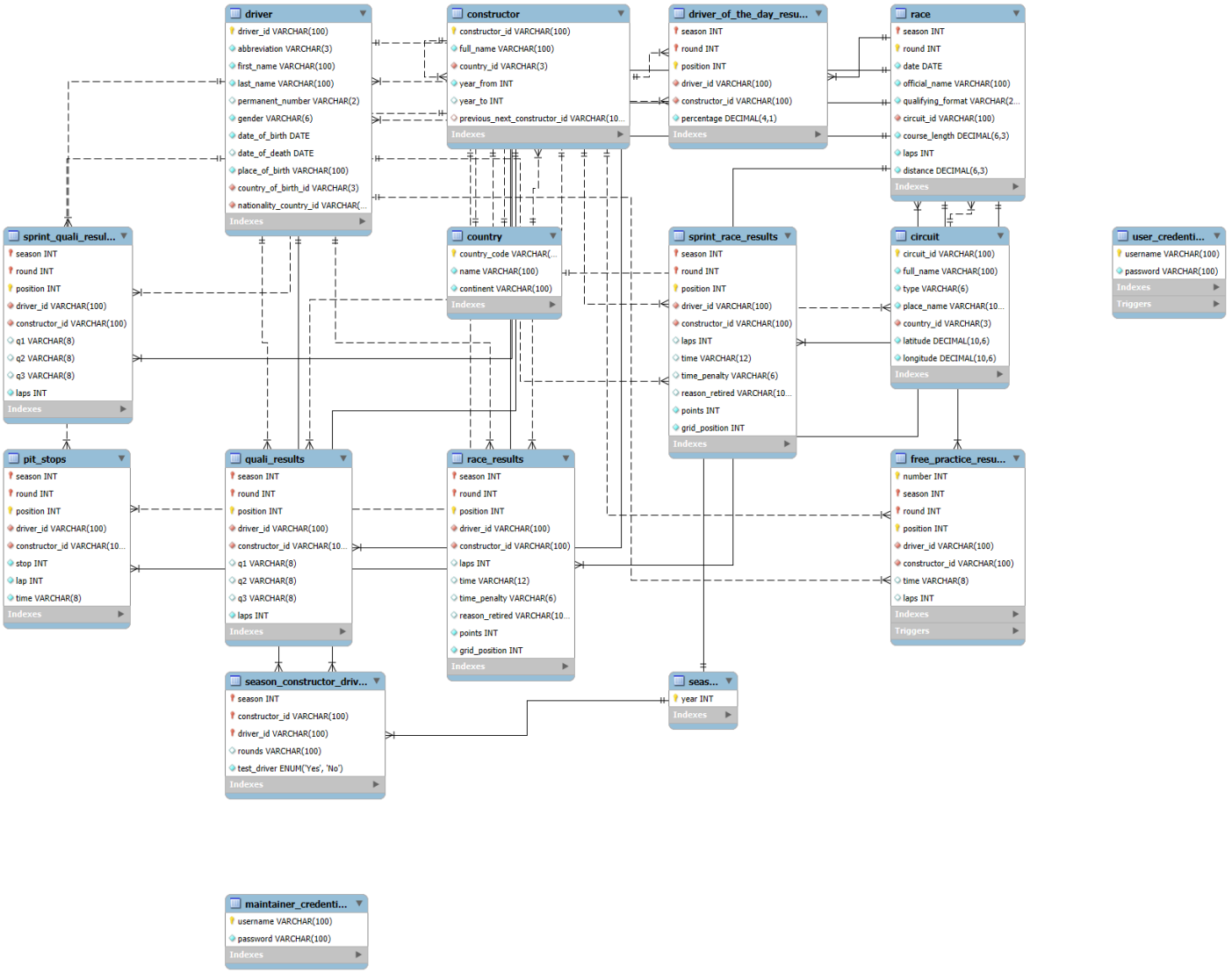
Softwares/Tools/Programming Languages Used:

1. **Database Interfaces**
    - MySQL - Workbench and Client
2. **Application Frameworks and Libraries**
    - **Backend Framework**: Flask
    - **Frontend Framework**: HTML, CSS
3. **Programming Languages**
    - **Backend**: Python
    - **Frontend**: JavaScript

ER Diagram:


country code
name
continent
Country
nationality
driver id
abbreviation
First name
gender
Driver
Last name
DOB
number
Date of Death
Place of birth
country of birth
head quarter
Constructor
constructor
full name
year from
year to
Prev
Prev next
next
Located
race result
position
laps
time penalty
time
quali time
points
Pitstop
grid position
reason retired
fast lap
Sprint result
position
laps
time penalty
time
quali time
Points
retired reason
Grid position
Driver of the Day
position
percentage
Free Practice
Position
Number
time
laps
Season Constructor driver
rounds
fast driver
circuit id
full name
type
Place name
latitude
longitude
circuit
takes place
Race
distance
laps
course length
quali format
official name
date
round
has
Season
year

Relational Schema:



DDL Commands:

Link to commands -
https://github.com/Vishal-Srinivasa/grandprix_hub/blob/master/f1_db_create_table.sql
(Includes ***Create*** operations)

#	Time	Action	Message	Duration / Fetch
1	11:40:39	CREATE DATABASE gp_hub	1 row(s) affected	0.000 sec
2	11:40:39	USE gp_hub	0 row(s) affected	0.000 sec
3	11:40:39	CREATE TABLE country (country_code VARCHAR(3) PRIMARY KEY, name VARCHAR(100) ...	0 row(s) affected	0.016 sec
4	11:40:39	CREATE TABLE constructor (constructor_id VARCHAR(100) PRIMARY KEY, full_name VARC...	0 row(s) affected	0.031 sec
5	11:40:39	CREATE TABLE circuit (circuit_id VARCHAR(100) PRIMARY KEY, full_name VARCHAR(100) ...	0 row(s) affected	0.015 sec
6	11:40:39	CREATE TABLE driver (driver_id VARCHAR(100) PRIMARY KEY, abbreviation VARCHAR(3) N...	0 row(s) affected	0.032 sec
7	11:40:39	CREATE TABLE season (year INTEGER PRIMARY KEY)	0 row(s) affected	0.015 sec
8	11:40:39	CREATE TABLE race (season INTEGER NOT NULL, round INTEGER NOT NULL, date DA...	0 row(s) affected	0.032 sec
9	11:40:39	CREATE TABLE season_constructor_driver (season INTEGER NOT NULL, constructor_id VA...	0 row(s) affected	0.031 sec
10	11:40:39	CREATE TABLE driver_of_the_day_results (season INTEGER NOT NULL, round INTEGER N...	0 row(s) affected	0.031 sec
11	11:40:39	CREATE TABLE free_practice_results (number INTEGER NOT NULL, season INTEGER NOT ...	0 row(s) affected	0.031 sec
12	11:40:39	CREATE TABLE race_results (season INTEGER NOT NULL, round INTEGER NOT NULL, p...	0 row(s) affected	0.016 sec
13	11:40:39	CREATE TABLE quali_results (season INTEGER NOT NULL, round INTEGER NOT NULL, ...	0 row(s) affected	0.031 sec
14	11:40:39	CREATE TABLE pit_stops (season INTEGER NOT NULL, round INTEGER NOT NULL, posi...	0 row(s) affected	0.031 sec
15	11:40:39	CREATE TABLE sprint_quali_results (season INTEGER NOT NULL, round INTEGER NOT NU...	0 row(s) affected	0.032 sec
16	11:40:39	CREATE TABLE sprint_race_results (season INTEGER NOT NULL, round INTEGER NOT NU...	0 row(s) affected	0.031 sec
17	11:40:39	CREATE TABLE user_credentials (username VARCHAR(100) PRIMARY KEY, password VARC...	0 row(s) affected	0.016 sec
18	11:40:39	CREATE TABLE maintainer_credentials (username VARCHAR(100) PRIMARY KEY, password ...	0 row(s) affected	0.015 sec
19	11:40:39	CREATE USER 'gp_hub_admin'@'localhost' IDENTIFIED BY 'gp_hub_admin_password'	0 row(s) affected	0.000 sec
20	11:40:39	GRANT ALL PRIVILEGES ON gp_hub.* TO 'gp_hub_admin'@'localhost'	0 row(s) affected	0.000 sec
21	11:40:39	CREATE USER 'gp_hub_maintainer'@'localhost' IDENTIFIED BY 'gp_hub_maintainer_password'	0 row(s) affected	0.000 sec
22	11:40:39	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.country TO 'gp_hub_maintainer'@'localh...	0 row(s) affected	0.000 sec

#	Time	Action	Message	Duration / Fetch
23	11:40:39	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.constructor TO 'gp_hub_maintainer'@'lo...	0 row(s) affected	0.015 sec
24	11:40:39	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.circuit TO 'gp_hub_maintainer'@'localhost'	0 row(s) affected	0.000 sec
25	11:40:39	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.driver TO 'gp_hub_maintainer'@'localhost'	0 row(s) affected	0.000 sec
26	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.season TO 'gp_hub_maintainer'@'localh...	0 row(s) affected	0.016 sec
27	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.race TO 'gp_hub_maintainer'@'localhost'	0 row(s) affected	0.000 sec
28	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.season_constructor_driver TO 'gp_hub_...	0 row(s) affected	0.000 sec
29	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.driver_of_the_day_results TO 'gp_hub_...	0 row(s) affected	0.000 sec
30	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.free_practice_results TO 'gp_hub_maint...	0 row(s) affected	0.016 sec
31	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.race_results TO 'gp_hub_maintainer'@'lo...	0 row(s) affected	0.000 sec
32	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.quali_results TO 'gp_hub_maintainer'@'l...	0 row(s) affected	0.000 sec
33	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.sprint_quali_results TO 'gp_hub_maintain...	0 row(s) affected	0.000 sec
34	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.sprint_race_results TO 'gp_hub_maintain...	0 row(s) affected	0.015 sec
35	11:40:40	GRANT SELECT, INSERT, UPDATE, DELETE ON gp_hub.pit_stops TO 'gp_hub_maintainer'@'local...	0 row(s) affected	0.000 sec
36	11:40:40	CREATE USER 'gp_hub_user'@'localhost' IDENTIFIED BY 'gp_hub_user_password'	0 row(s) affected	0.000 sec
37	11:40:40	GRANT SELECT ON gp_hub.country TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.016 sec
38	11:40:40	GRANT SELECT ON gp_hub.constructor TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
39	11:40:40	GRANT SELECT ON gp_hub.circuit TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
40	11:40:40	GRANT SELECT ON gp_hub.driver TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
41	11:40:40	GRANT SELECT ON gp_hub.season TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.016 sec
42	11:40:40	GRANT SELECT ON gp_hub.race TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
43	11:40:40	GRANT SELECT ON gp_hub.season_constructor_driver TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
44	11:40:40	GRANT SELECT ON gp_hub.driver_of_the_day_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec

45	11:40:40	GRANT SELECT ON gp_hub.free_practice_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.015 sec
46	11:40:40	GRANT SELECT ON gp_hub.race_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
47	11:40:40	GRANT SELECT ON gp_hub.quali_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
48	11:40:40	GRANT SELECT ON gp_hub.sprint_quali_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.016 sec
49	11:40:40	GRANT SELECT ON gp_hub.sprint_race_results TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
50	11:40:40	GRANT SELECT ON gp_hub.pit_stops TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
51	11:40:40	CREATE TRIGGER check_new_user BEFORE INSERT ON user_credentials FOR EACH ROW BEG...	0 row(s) affected	0.000 sec
52	11:40:40	CREATE PROCEDURE get_constructor_standings(IN season_year INT) BEGIN SELECT ROW_...	0 row(s) affected	0.000 sec
53	11:40:40	CREATE PROCEDURE get_driver_standings(IN season_year INT) BEGIN SELECT ROW_NUM...	0 row(s) affected	0.000 sec
54	11:40:40	GRANT EXECUTE ON PROCEDURE gp_hub.get_driver_standings TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
55	11:40:40	GRANT EXECUTE ON PROCEDURE gp_hub.get_constructor_standings TO 'gp_hub_user'@'localh...	0 row(s) affected	0.016 sec
56	11:40:40	CREATE FUNCTION get_race_wins(id VARCHAR(100)) RETURNS INTEGER DETERMINISTIC BE...	0 row(s) affected	0.000 sec
57	11:40:40	CREATE FUNCTION get_pole_positions(id VARCHAR(100)) RETURNS INTEGER DETERMINISTIC...	0 row(s) affected	0.000 sec
58	11:40:40	CREATE FUNCTION get_podiums(id VARCHAR(100)) RETURNS INTEGER DETERMINISTIC BEGI...	0 row(s) affected	0.016 sec
59	11:40:40	CREATE FUNCTION get_sprint_wins(id VARCHAR(100)) RETURNS INTEGER DETERMINISTIC B...	0 row(s) affected	0.000 sec
60	11:40:40	GRANT EXECUTE ON FUNCTION gp_hub.get_race_wins TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
61	11:40:40	GRANT EXECUTE ON FUNCTION gp_hub.get_pole_positions TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
62	11:40:40	GRANT EXECUTE ON FUNCTION gp_hub.get_podiums TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.015 sec
63	11:40:40	GRANT EXECUTE ON FUNCTION gp_hub.get_sprint_wins TO 'gp_hub_user'@'localhost'	0 row(s) affected	0.000 sec
64	11:40:40	CREATE PROCEDURE get_driver_and_race_stats(IN p_driver_id VARCHAR(100), IN p_season INT...	0 row(s) affected	0.000 sec
65	11:40:40	GRANT EXECUTE ON PROCEDURE gp_hub.get_driver_and_race_stats TO 'gp_hub_user'@'localh...	0 row(s) affected	0.000 sec
66	11:40:40	CREATE TRIGGER check_sprint_fp_sessions AFTER INSERT ON free_practice_results FOR EACH...	0 row(s) affected	0.016 sec

Functionalities and Features of the Application

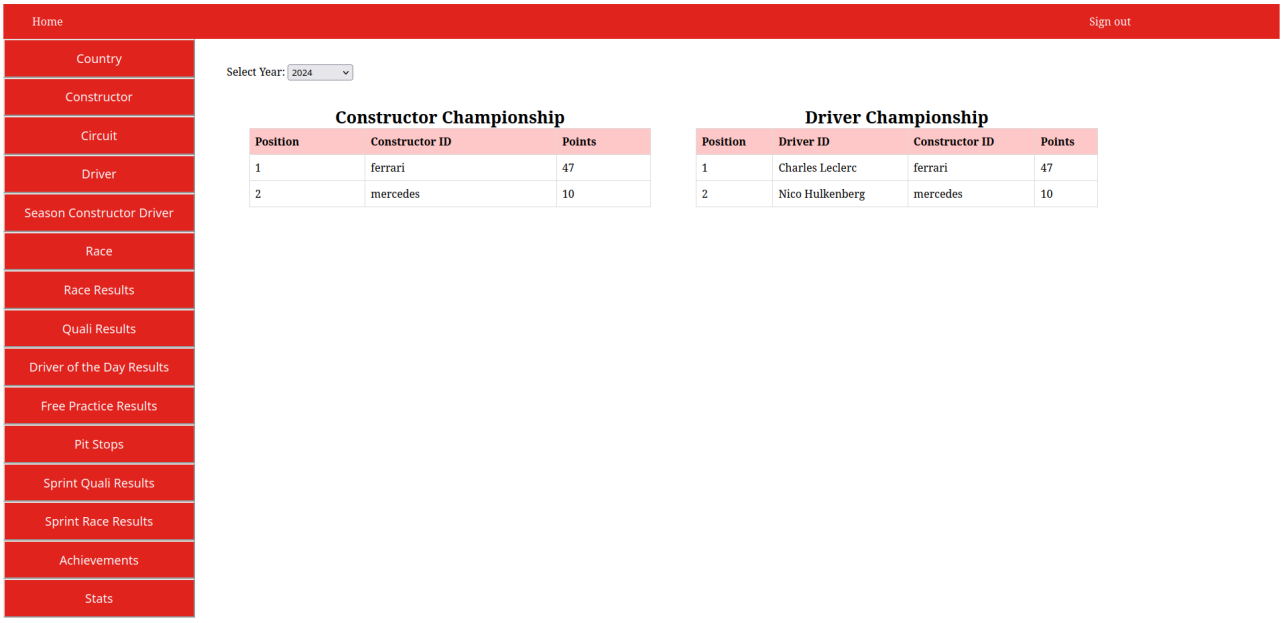
The first page that appears when you visit the website. Here you can select the type of account you want to login as.

If user is selected, you can create an account if you're new, or login with your registered credential.

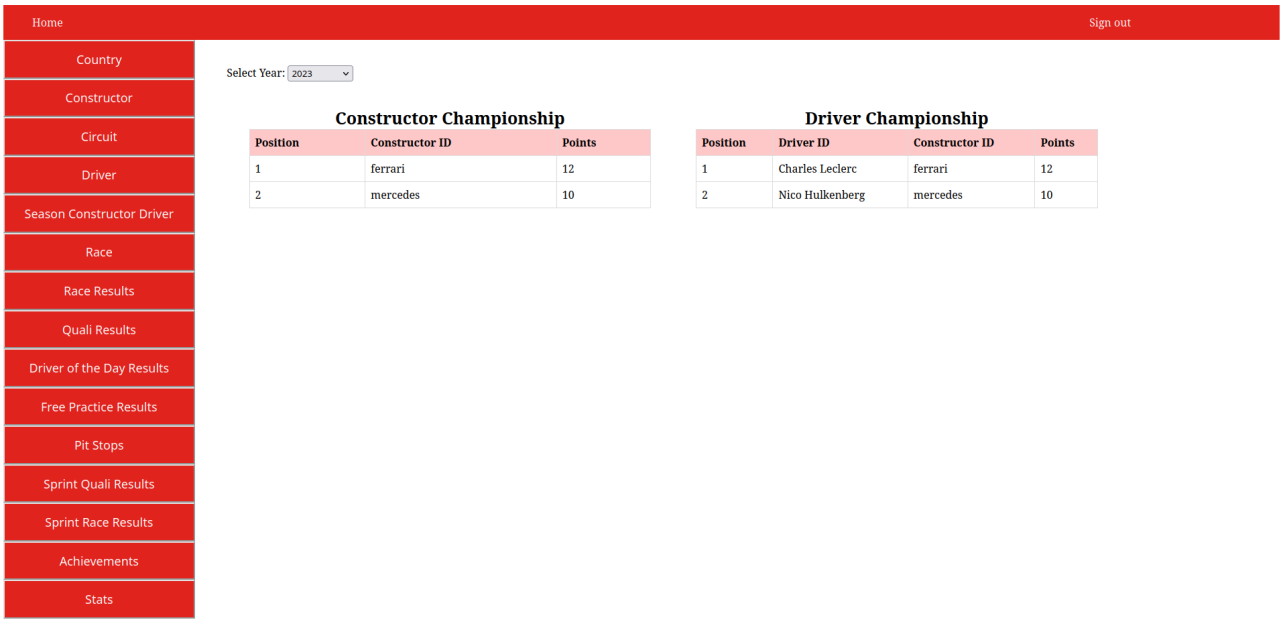
To sign in

To sign up

When you enter as a user, the home page displays the driver and constructor standings for the latest year. There are 2 ***procedures*** written which will fetch the results for the specified year. These procedures use ***aggregate*** and ***nested*** queries to fetch the results.



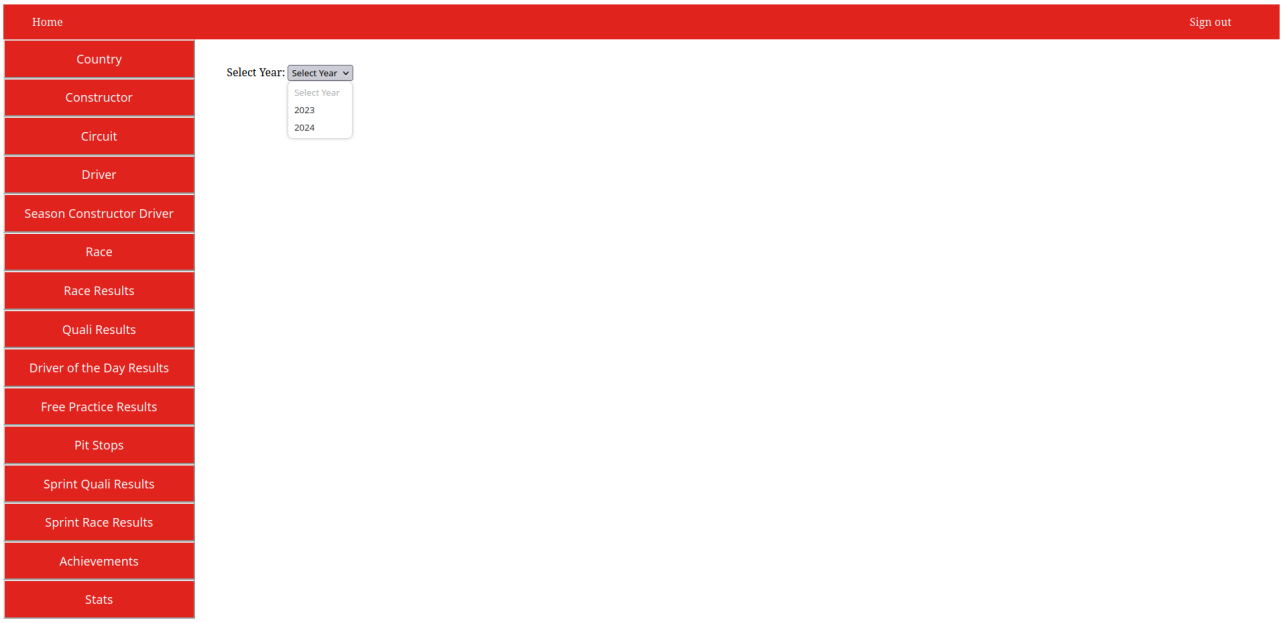
You can change the year using the drop down menu.



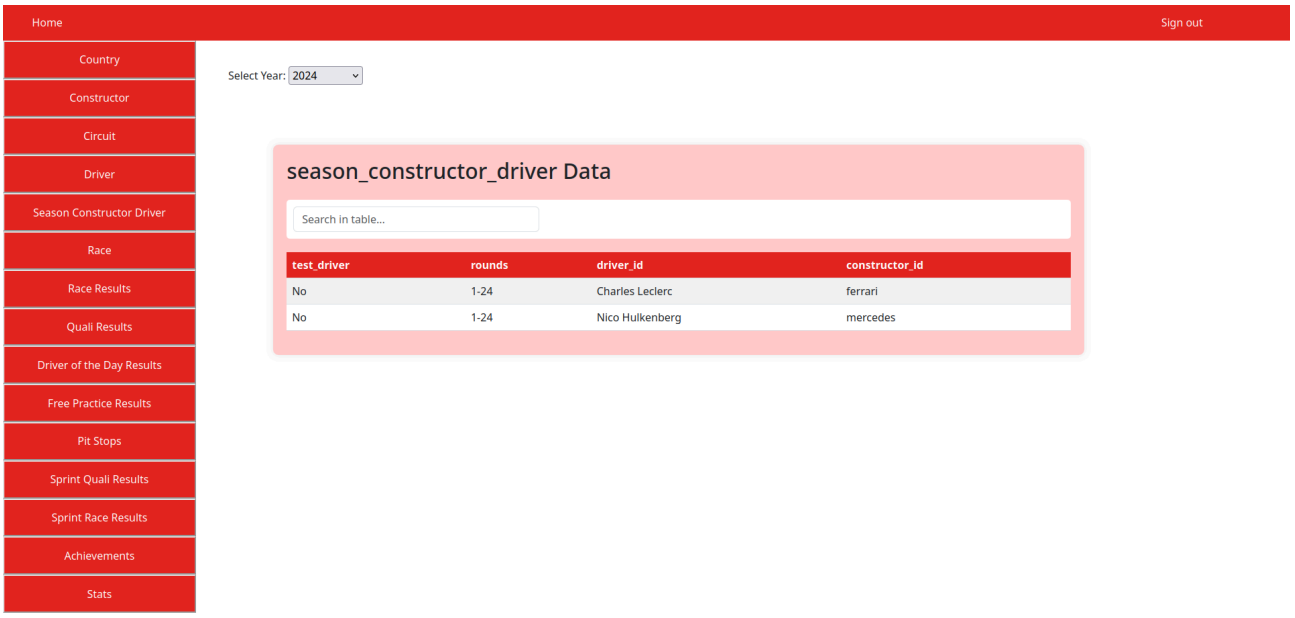
When clicked on constructor in the left side menu, the user can see the contents in the constructor table. Similarly he can see the contents of country, circuit and the driver tables. This is a ***read*** operation.

Using the home button he can go back to the home page, or he can even choose to sign out and go back to the first login page.

When clicked on season constructor driver in the left side menu. This is also read but with a condition on the year.

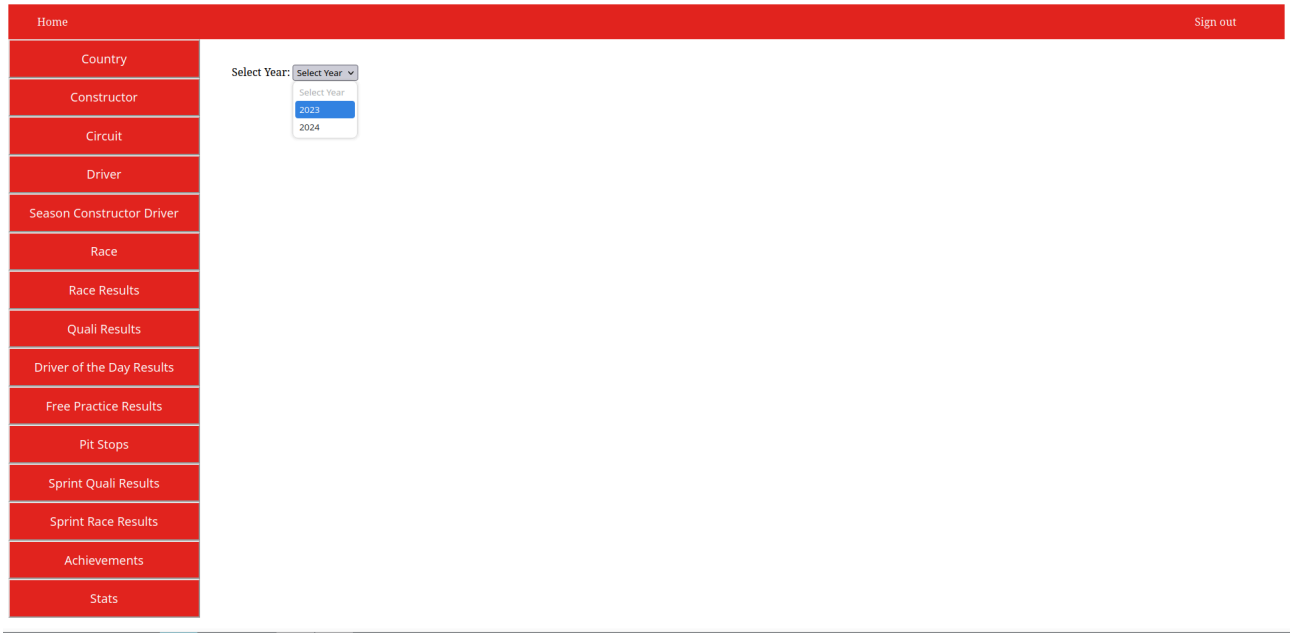


You get a drop down menu to select a year, when selected it gives all the contents of the table for that year.

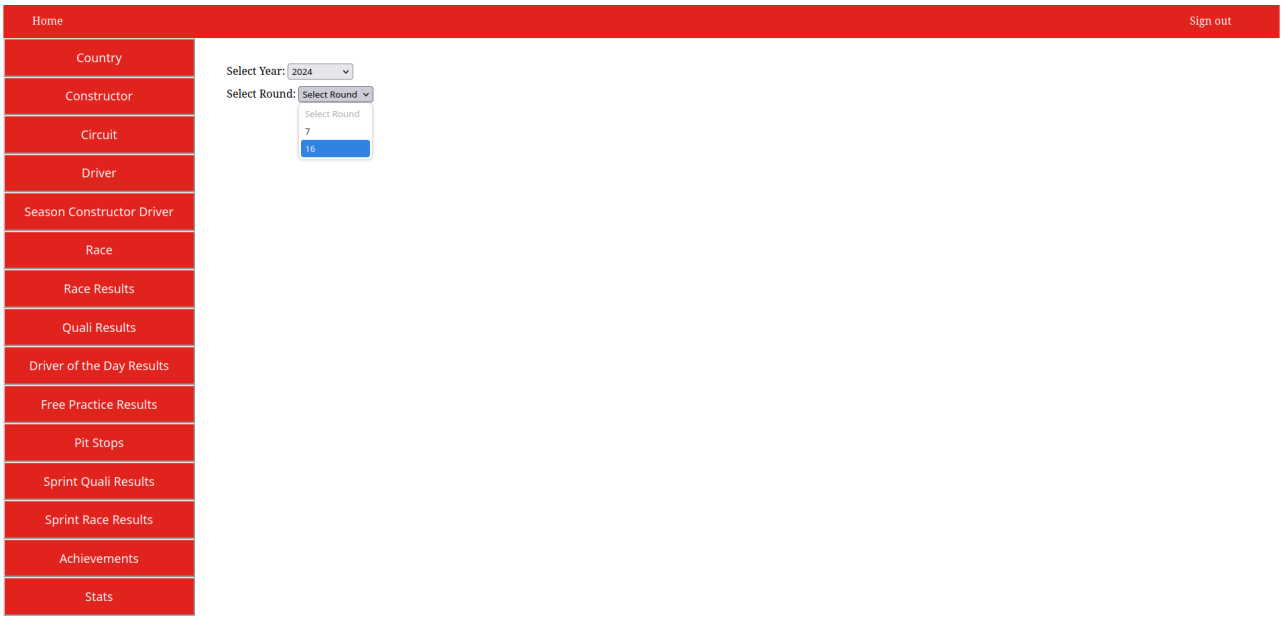


To view for another year the user and simply change in the drop down menu and the details for that year will be fetched.

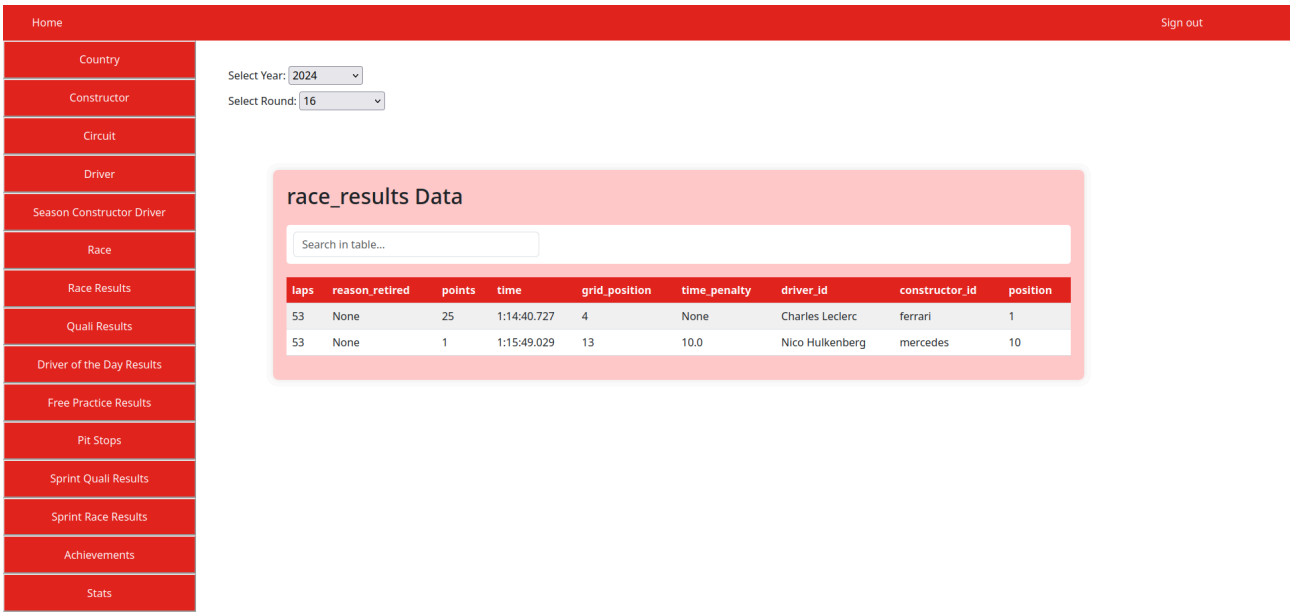
For the remaining tables (except achievements and stats), when clicked first we get a drop down to select a year.



When the year is selected we then have to select the round for which you want see the contents for



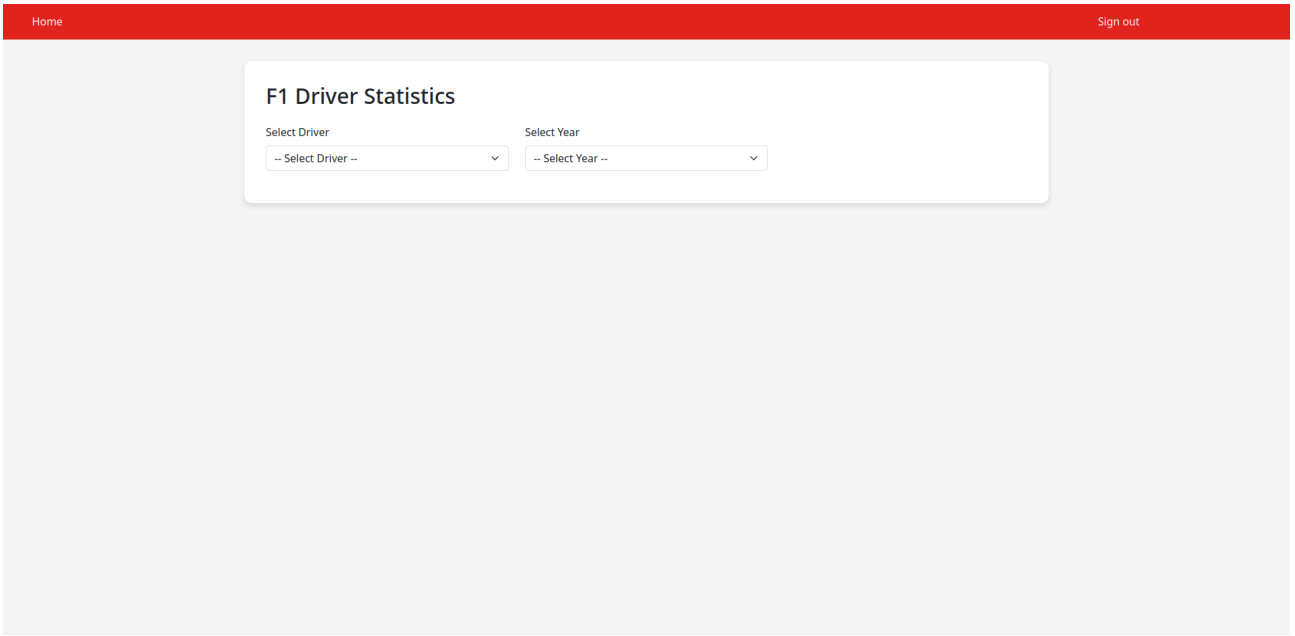
When the round is also selected we get the contents of the table for that year and round.



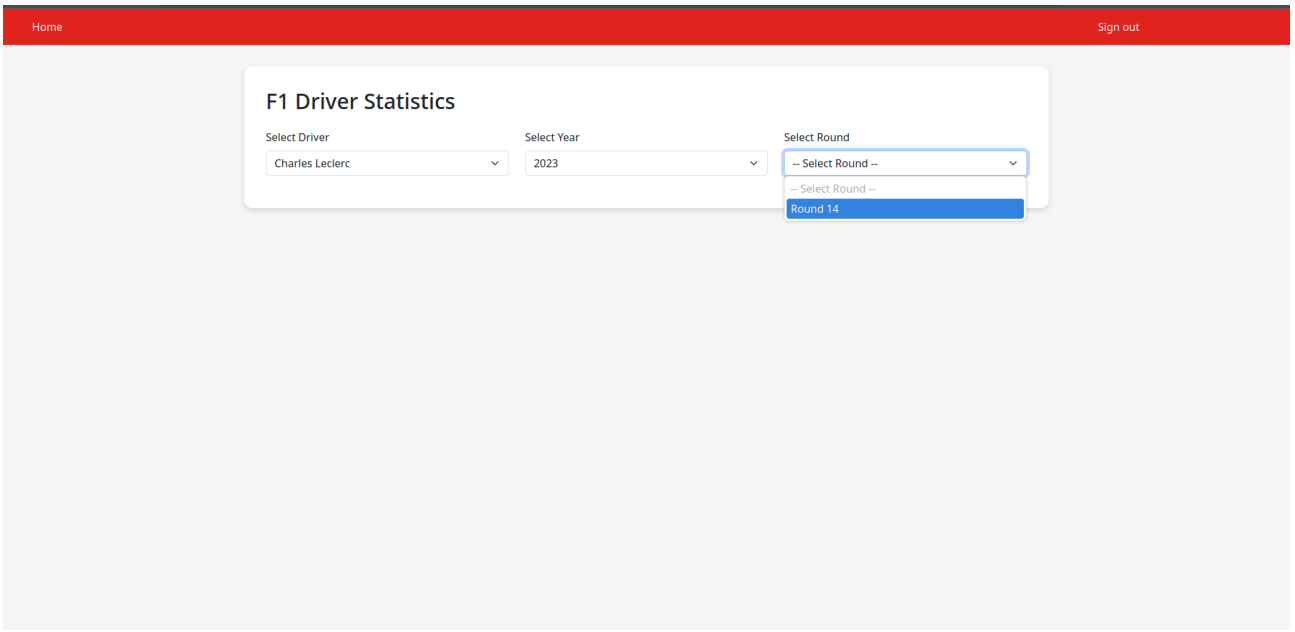
When achievements is clicked, we get a drop down to select the driver whose achievements we want to see.

When selected, there are 4 ***functions*** which fetch the value for the given driver.

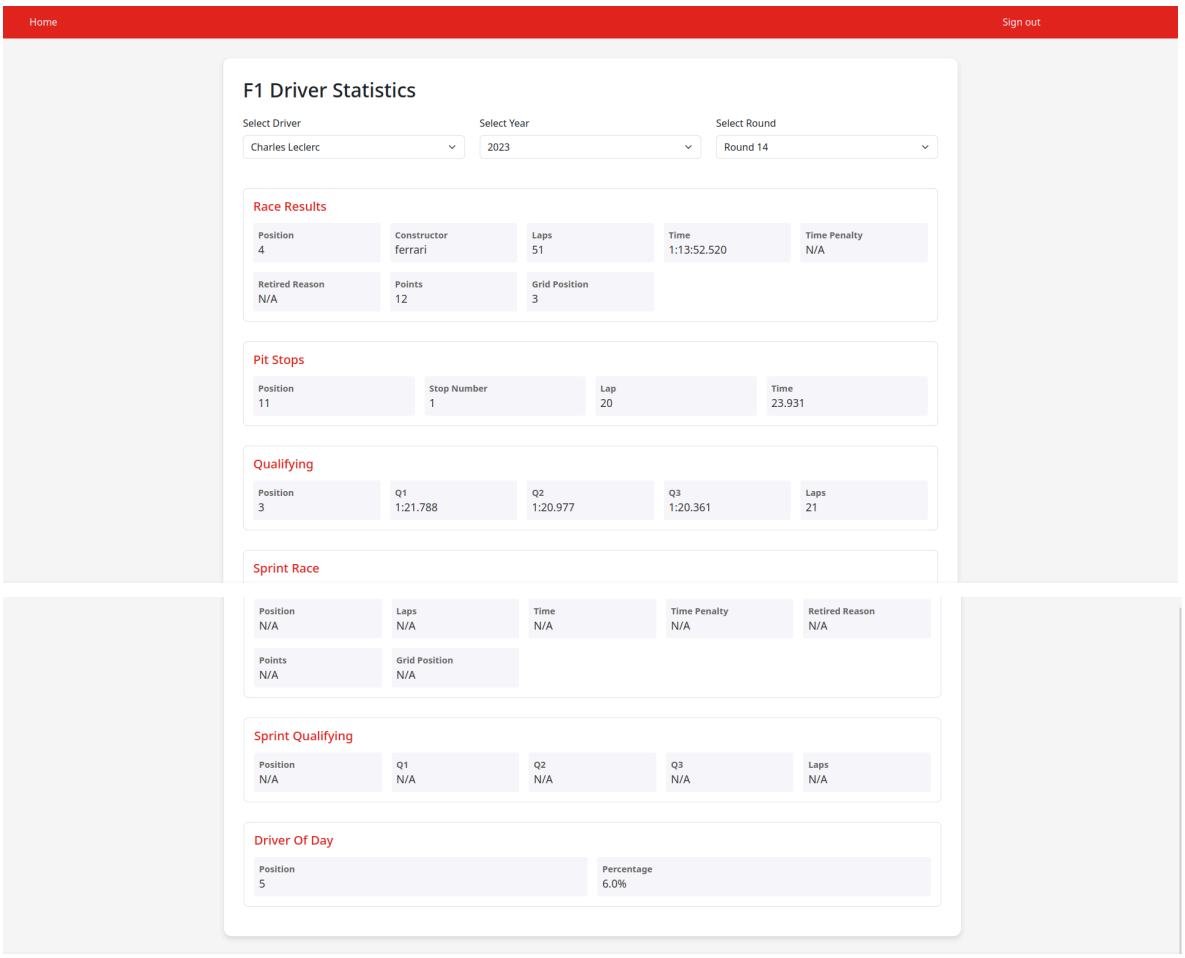
When stats is clicked, we get 2 drop downs, 1 for selecting driver and the other for selecting year.



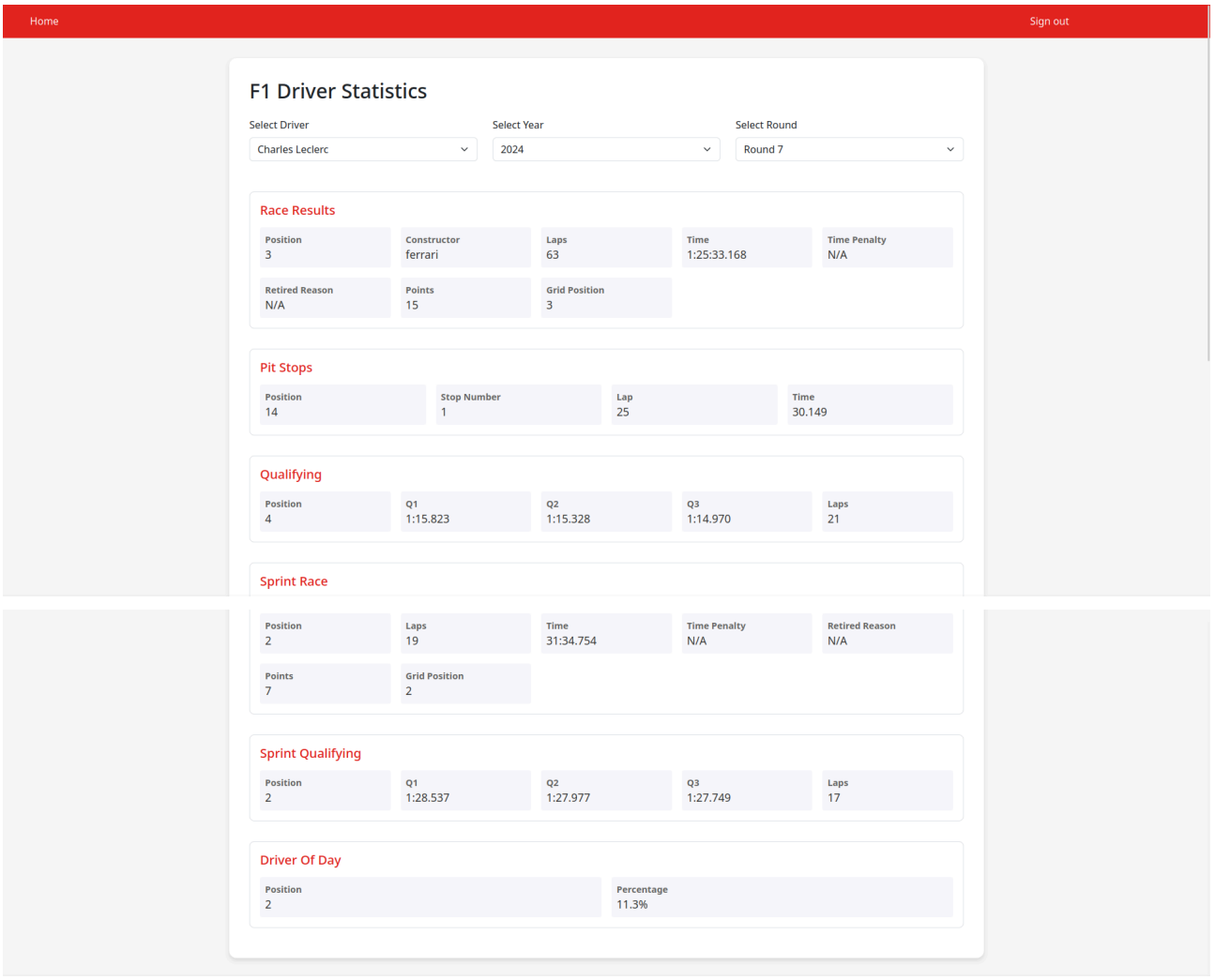
When selected we get another dropdown for selecting the round.



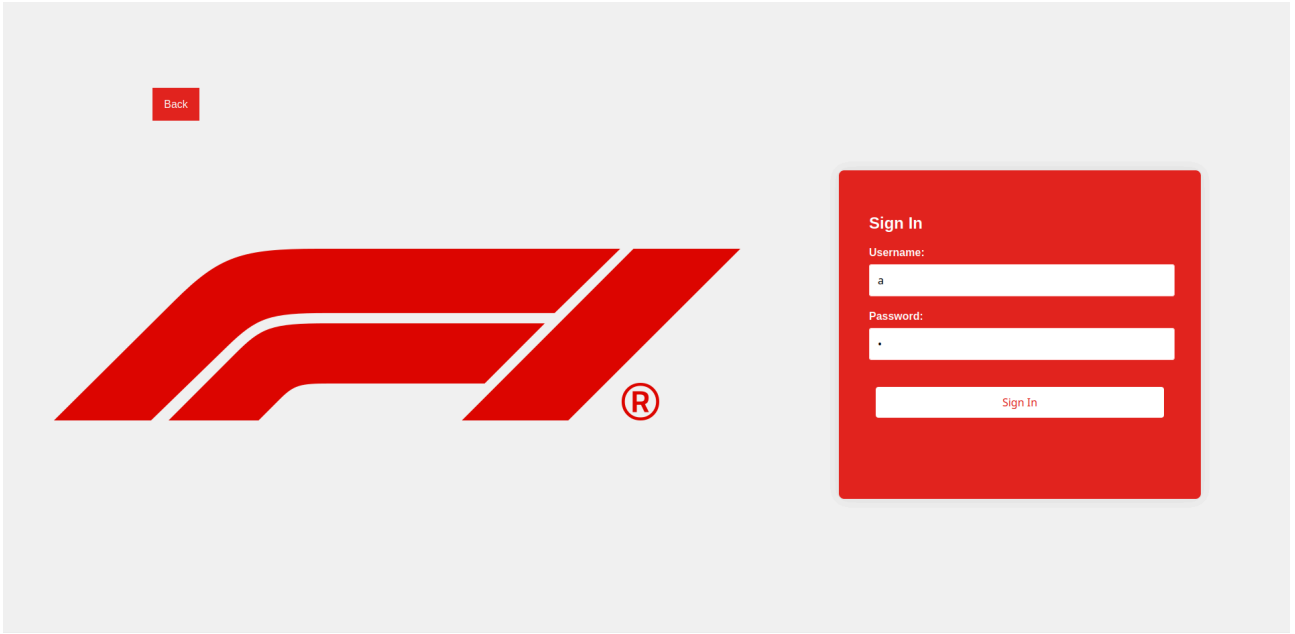
When the round is selected a ***procedure*** which ***joins*** the content of multiple tables and gives us the result for that particular driver, year, and round values.



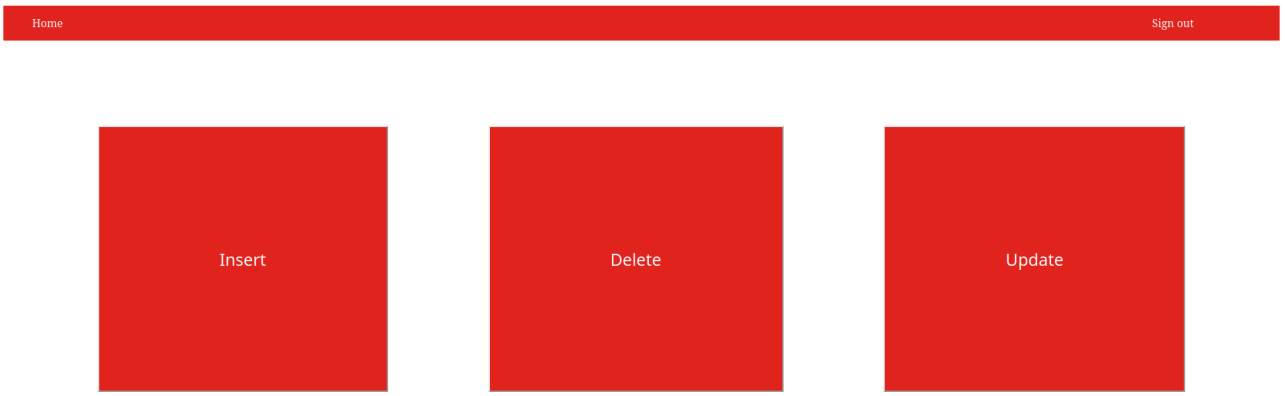
This round does not have a sprint race, so we will change to a round where there is a sprint.



Now we move on to the maintainer. The maintainer has only the sign in option, no one can create an account as a maintainer by themselves.

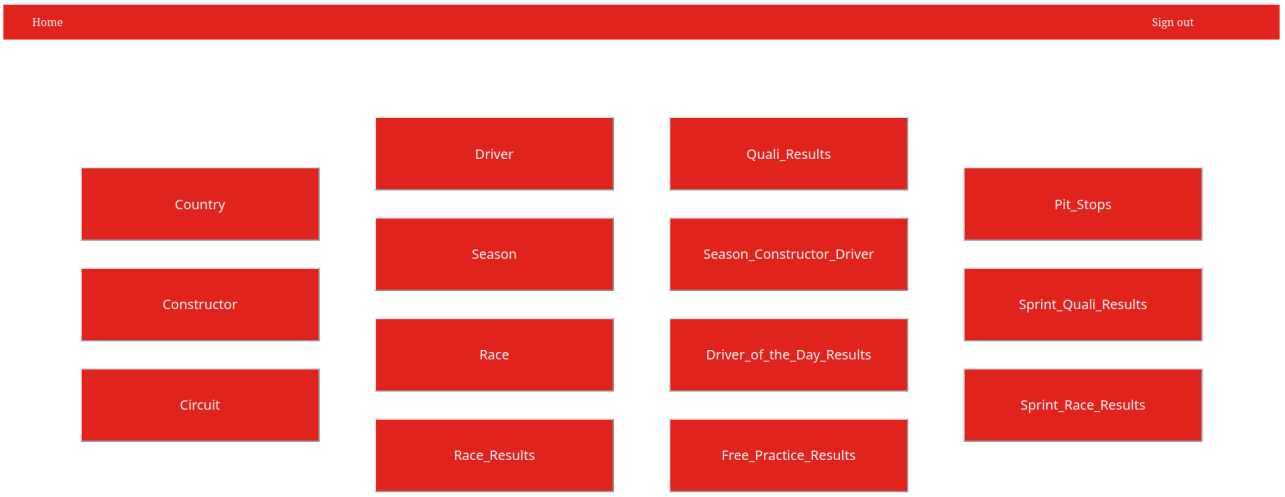


When we enter the correct credentials, we are taken to the homepage of the maintainer.

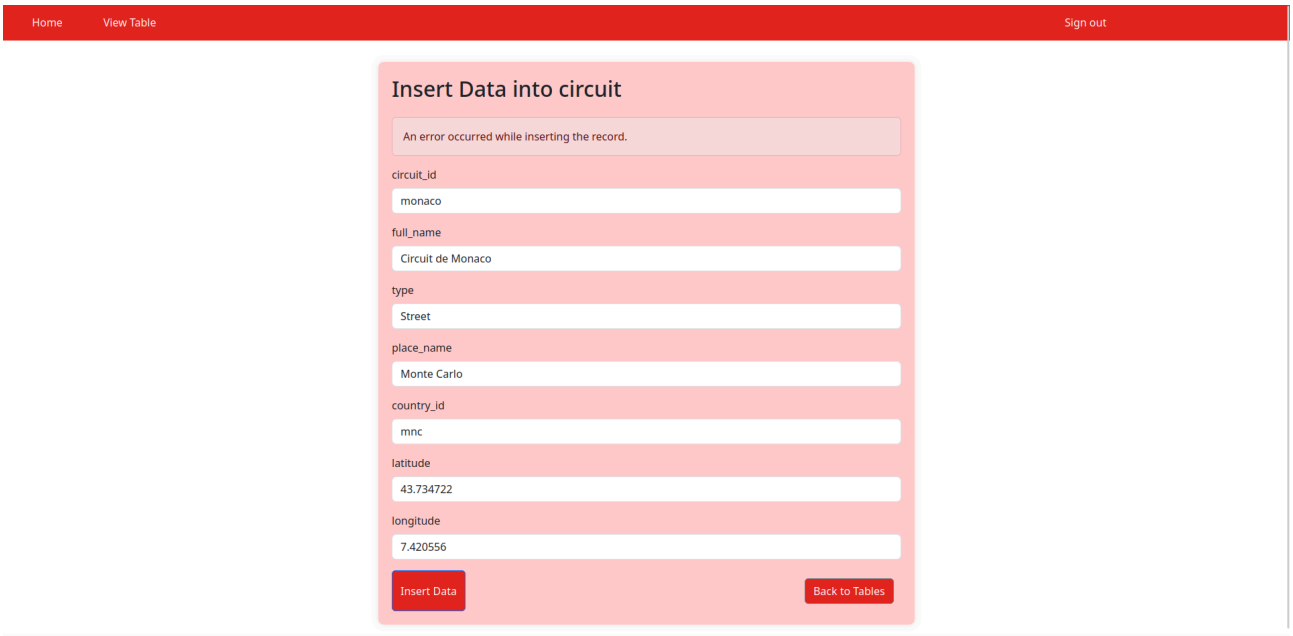


The maintainer can ***insert*** values into the tables, ***delete*** from the tables and also ***update*** the values in the tables.

When insert is selected we get a page to select the table to which you want to the enter the values into.

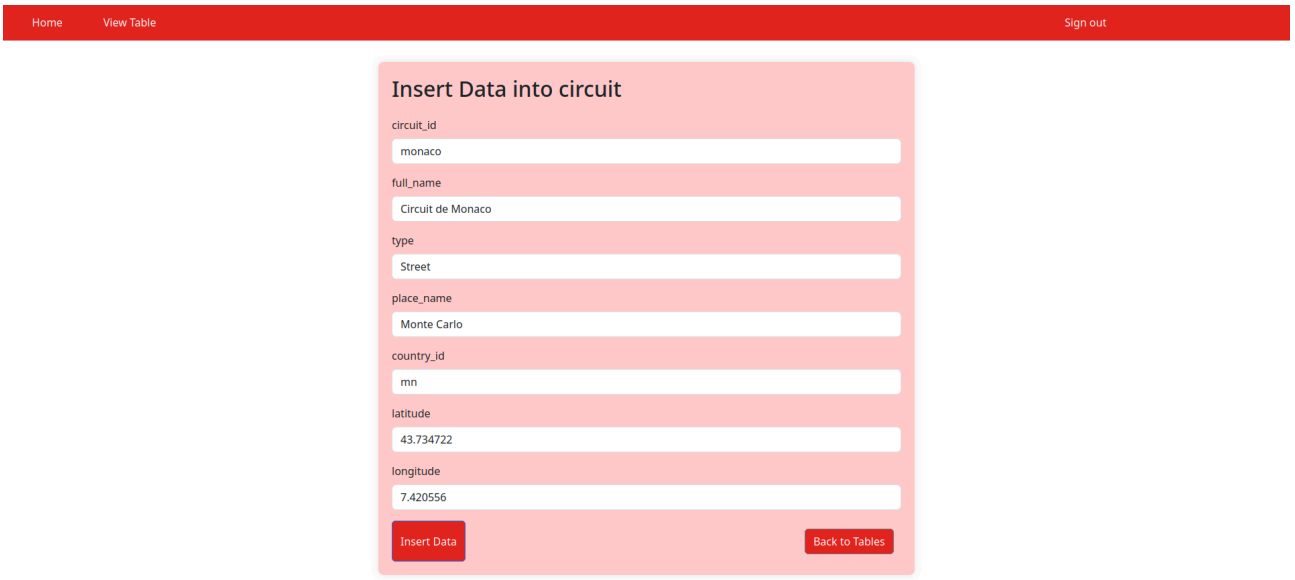


When a table is selected we get all the column names of the table and an input box next to it.

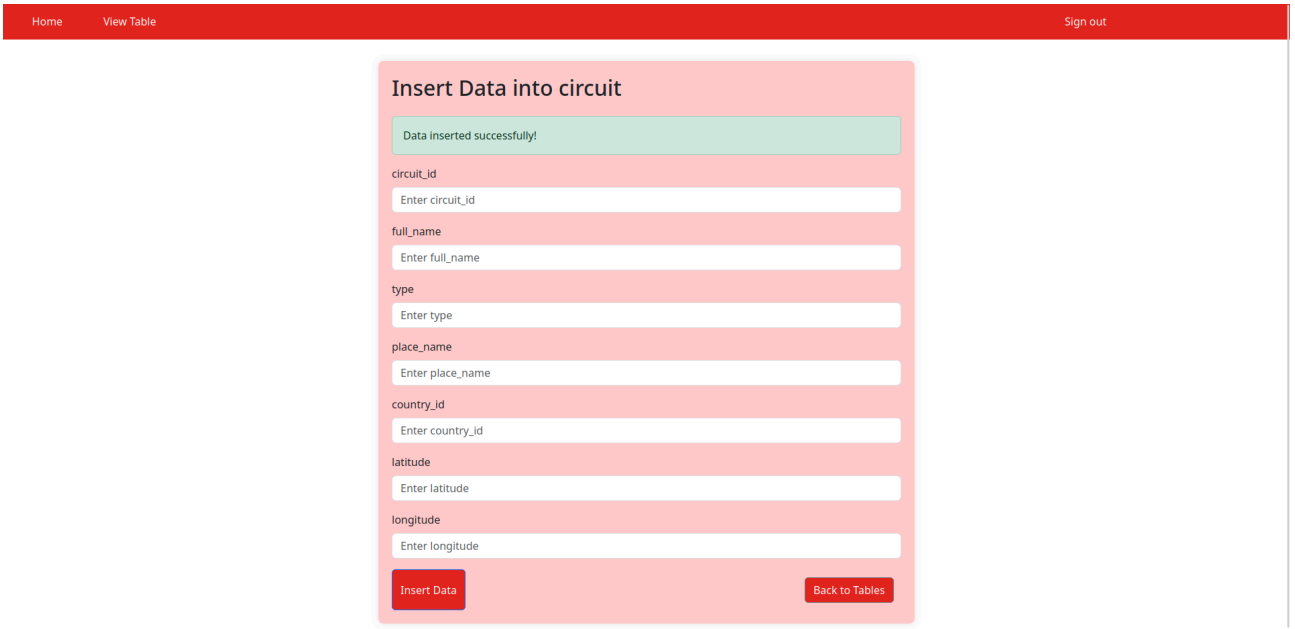


Back to tables button will take us back to the previous page where we can select the tables. We can click on the view tables to see the content present in that table. When we enter a wrong value, (here there is a foreign key constraint in country id where I have used mnc instead of mn), it is not inserted and we get an error message.

Then table before inserting the values.



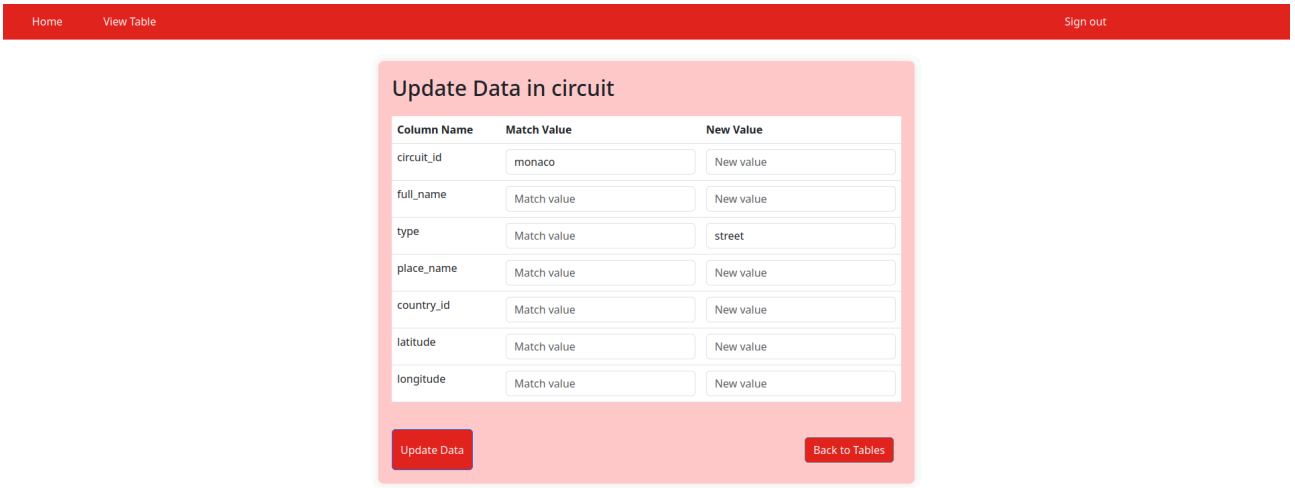
We have now filled in the correct values and when we click on insert.



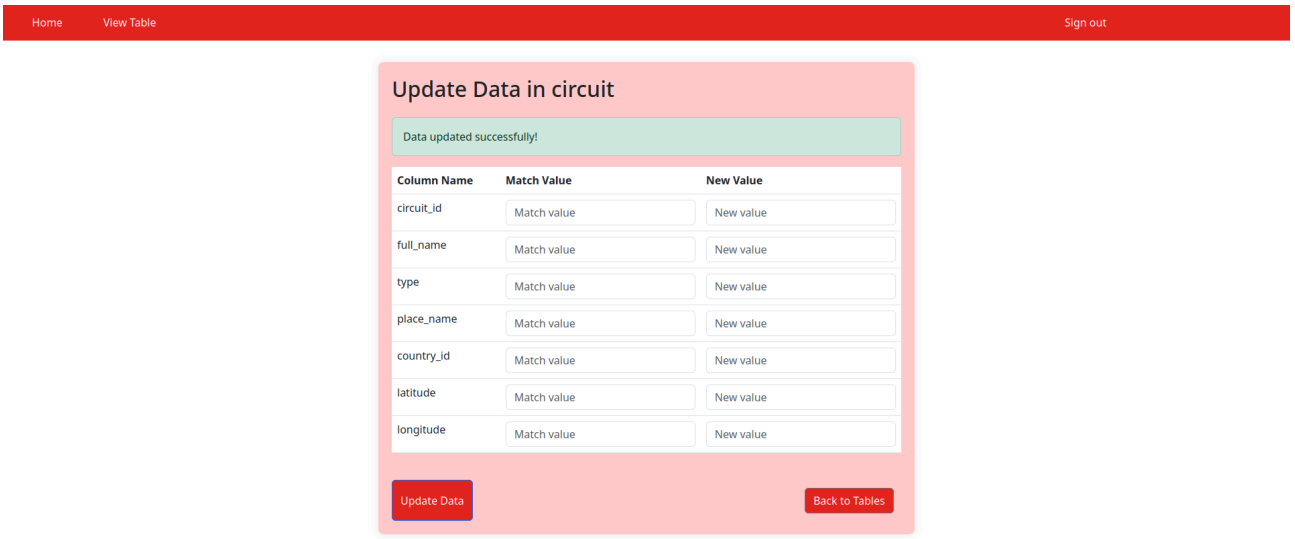
The values are inserted, we get a success message and the form is cleared for us to enter the next values.

We see that the new value is inserted.

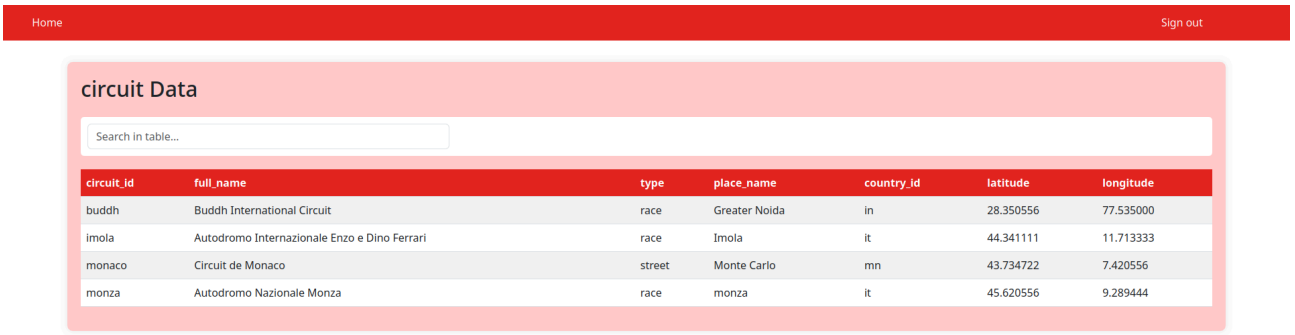
Now we move onto the ***update*** part. When update button is clicked in the homepage we get the select tables page and when the table is selected we get this page.



Now we enter values, the first input column is for matching the values and the next column is for the values we want to change. Here we are changing the value of type from ‘Street’ to ‘street’.

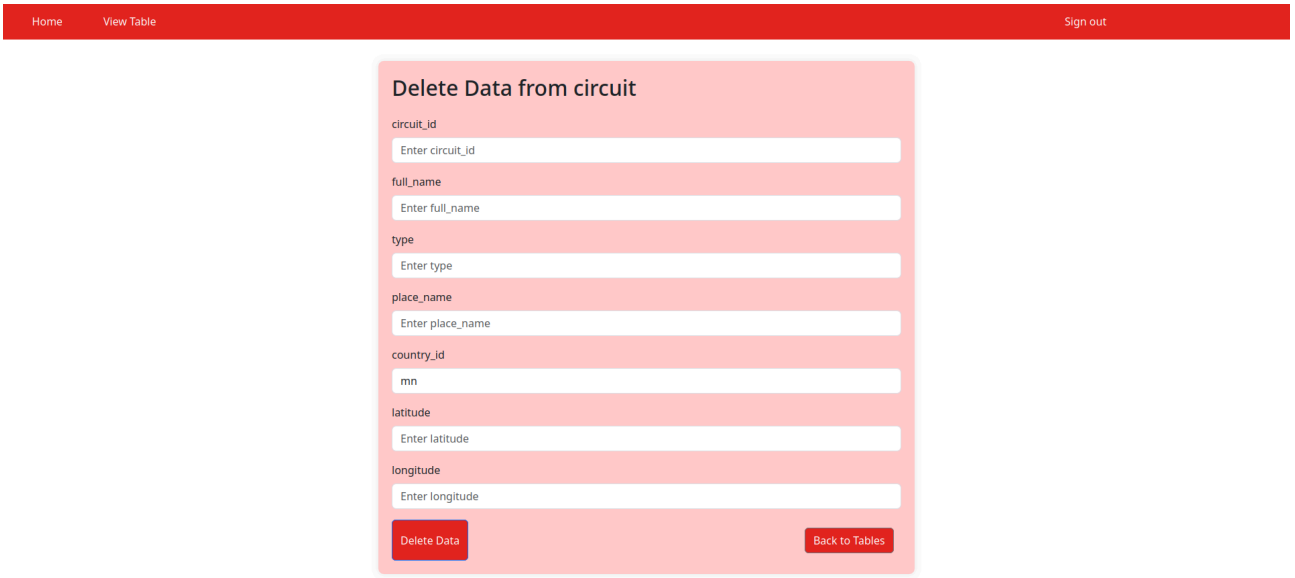


We have successfully updated the value.

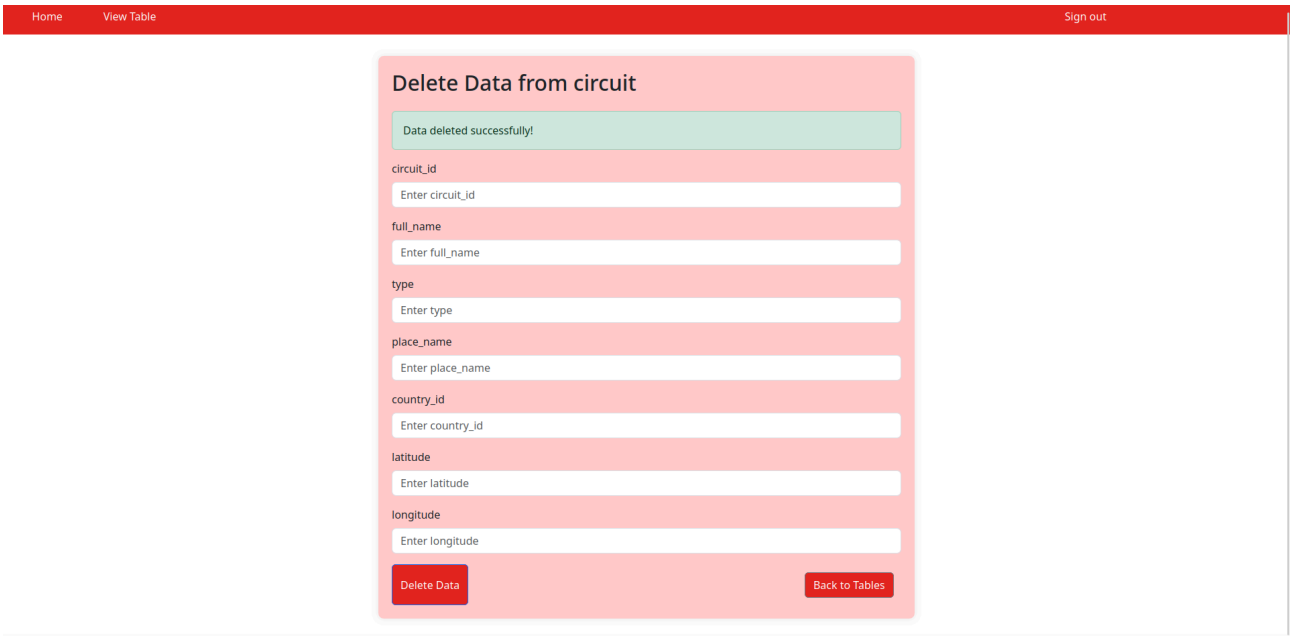


We see that the value has changed.

Now we go to ***delete.***



When the value is filled and the delete button is clicked, it checks for that value and if present it gets deleted.

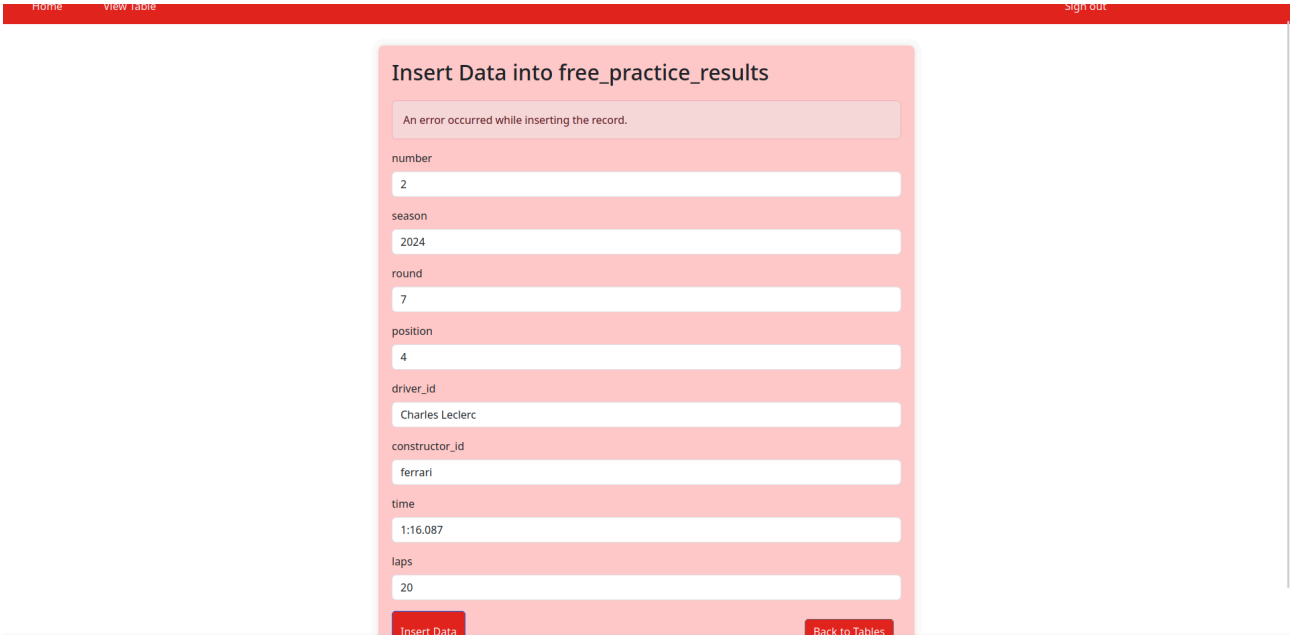


Successfully deleted.

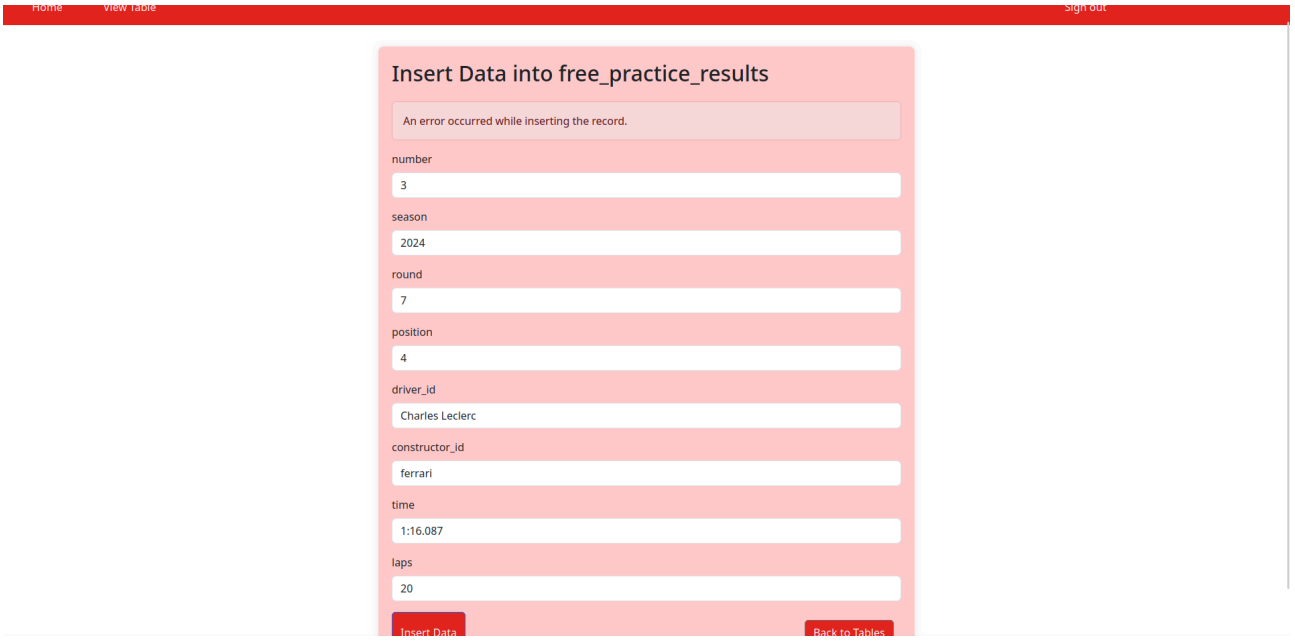
We see that it is deleted.

There is a trigger for checking insertion into the free practice results table. It checks if there is sprint race in that round and if there is a sprint it will not allow us to enter the free practice number as 2 or 3 and only 1 is allowed, if there is no sprint we can enter 1, 2 and 3.

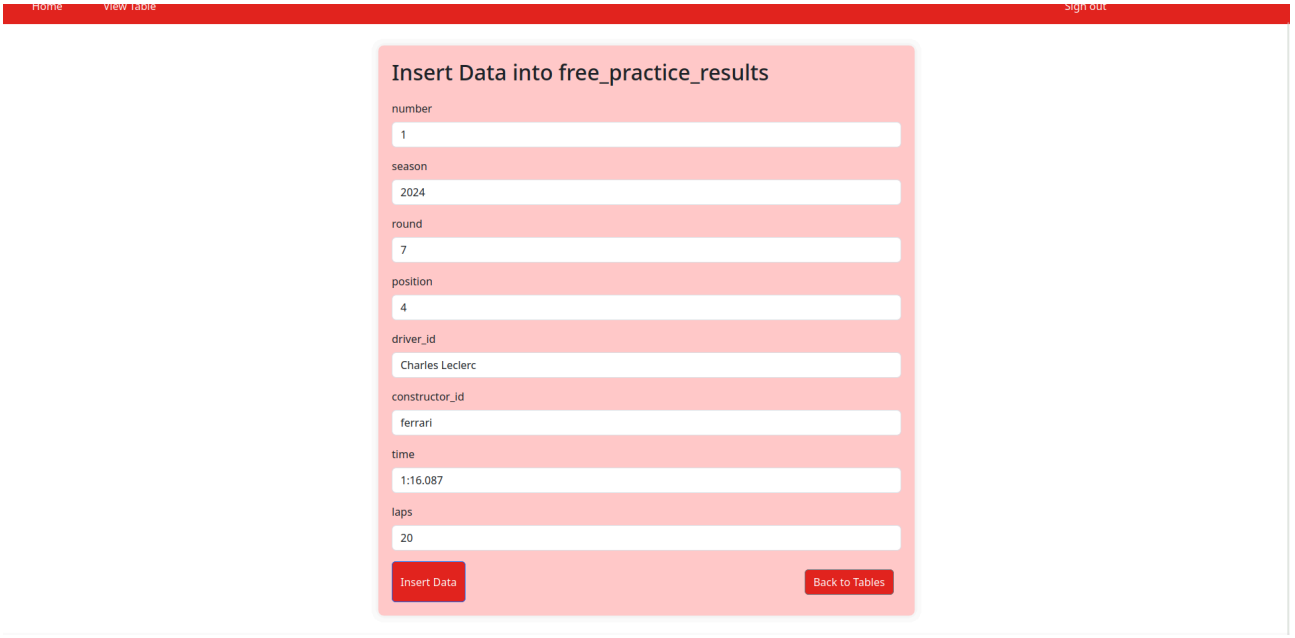
The table before inserting.



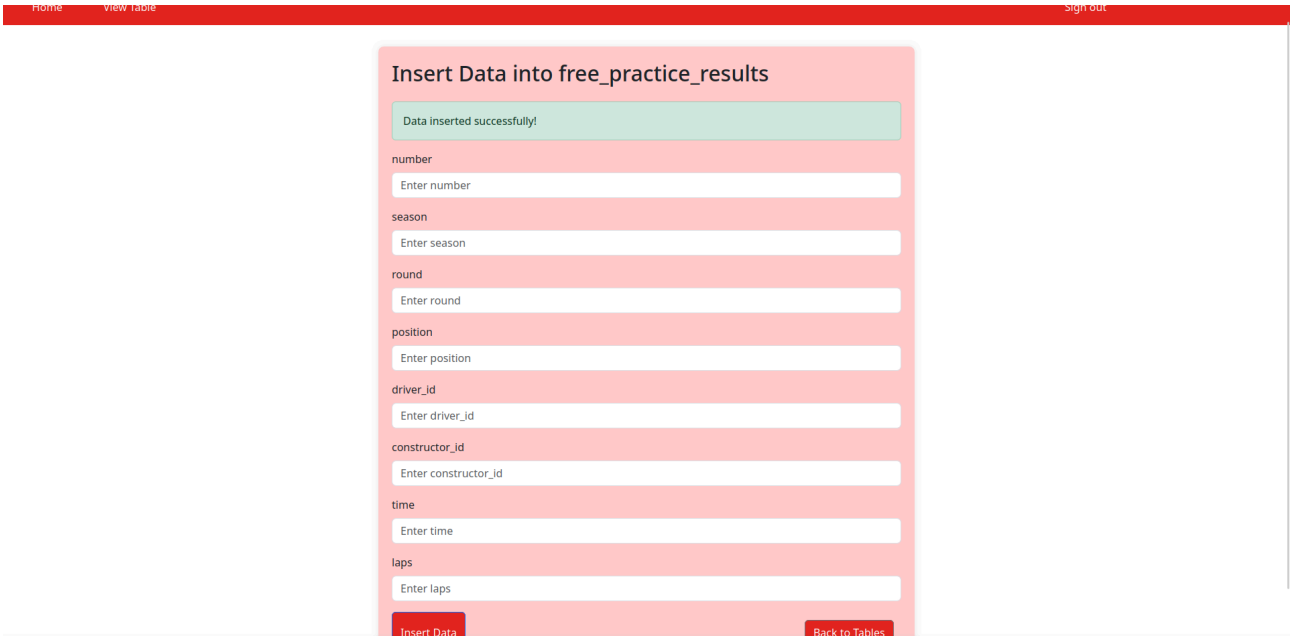
We try inserting 2 in a round with a sprint, but it fails.



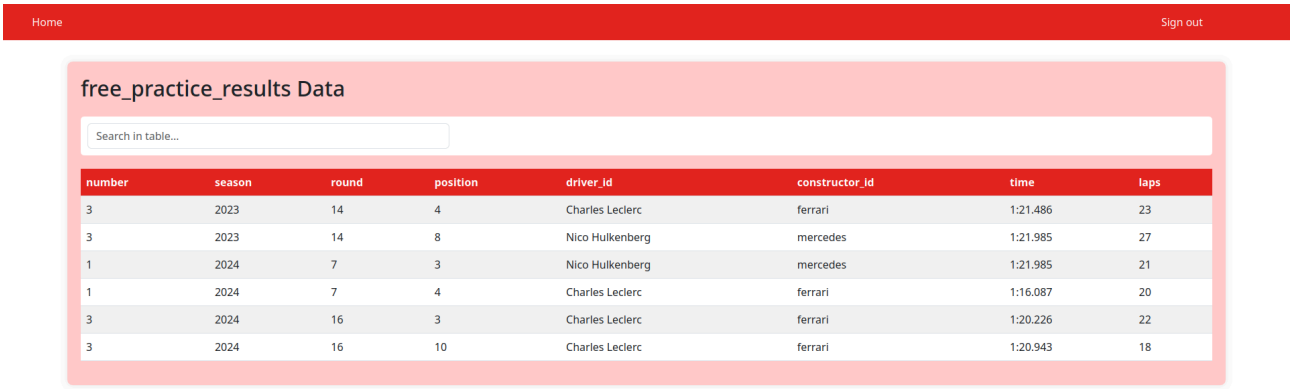
We try inserting 3 in a round with a sprint, but it fails.



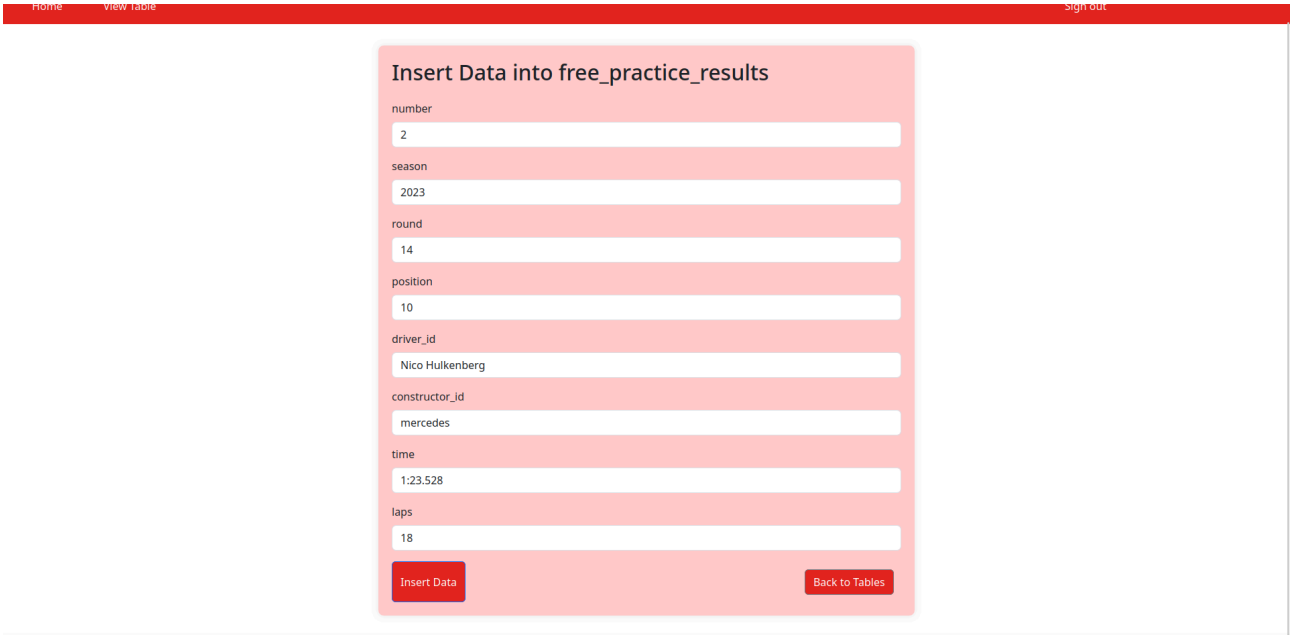
Now we try inserting 1.


Home
View Table
Sign out
Insert Data into free_practice_results
Data inserted successfully!
number
Enter number
season
Enter season
round
Enter round
position
Enter position
driver_id
Enter driver_id
constructor_id
Enter constructor_id
time
Enter time
laps
Enter laps
Insert Data
Back to Tables

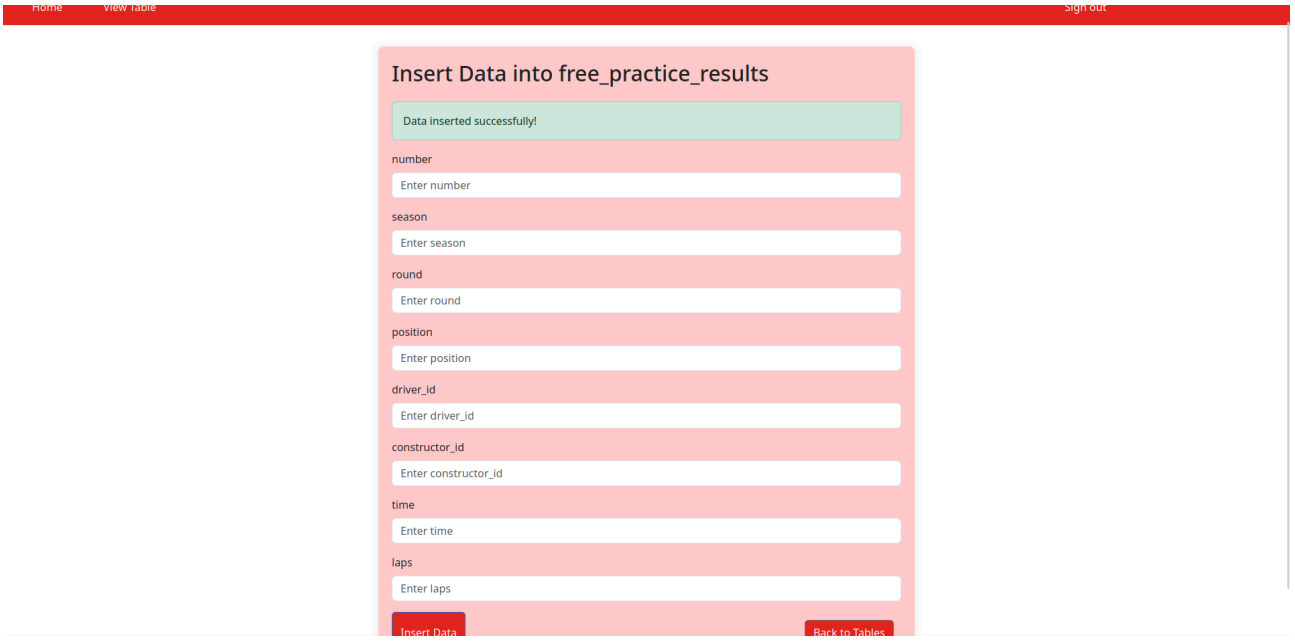
We succeed.

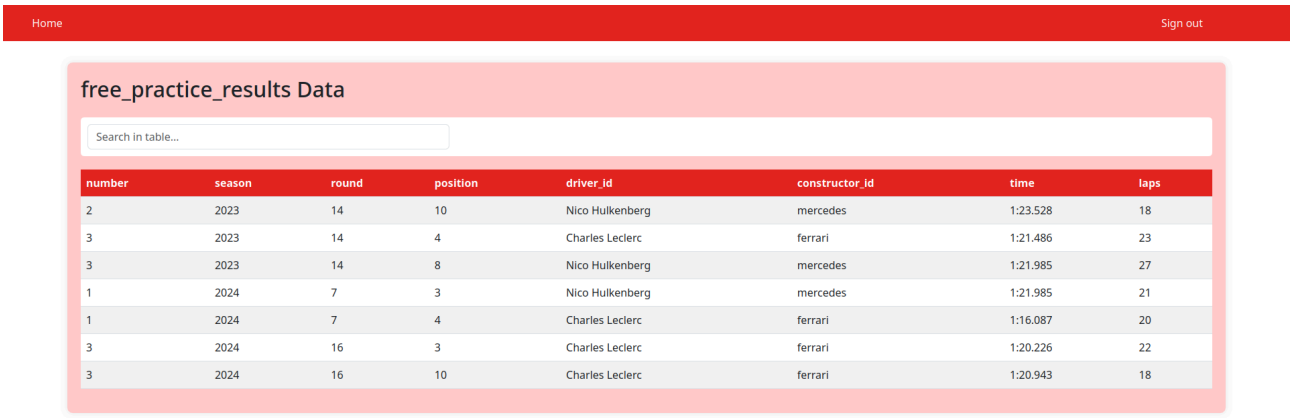

Home
Sign out
free_practice_results Data
Search in table...
number season round position driver_id constructor_id time laps
3 2023 14 4 Charles Leclerc ferrari 1:21.486 23
3 2023 14 8 Nico Hulkenberg mercedes 1:21.985 27
1 2024 7 3 Nico Hulkenberg mercedes 1:21.985 21
1 2024 7 4 Charles Leclerc ferrari 1:16.087 20
3 2024 16 3 Charles Leclerc ferrari 1:20.226 22
3 2024 16 10 Charles Leclerc ferrari 1:20.943 18

We try for a non sprint round.



We are inserting free practice 2 results.





Success

Create Operation:

```
CREATE TABLE race_results (
    season INTEGER NOT NULL,
    round INTEGER NOT NULL,
    position INTEGER NOT NULL,
    driver_id VARCHAR(100) NOT NULL,
    constructor_id VARCHAR(100) NOT NULL,
    laps INTEGER,
    time VARCHAR(12),
    time_penalty VARCHAR(6),
    reason_retired VARCHAR(100),
    points INTEGER NOT NULL,
    grid_position INTEGER NOT NULL,
    PRIMARY KEY (season, round, position),
    FOREIGN KEY (season, round) REFERENCES race(season, round),
    FOREIGN KEY (driver_id) REFERENCES driver(driver_id),
    FOREIGN KEY (constructor_id) REFERENCES constructor(constructor_id)
);
```

(more create operations available in link provided under DDL commands)

Insert Operation:

```
@app.route('/insert', methods = ['POST'])
def insert():
    table_name = request.form['table_name']
    column_names_list = request.form['column_names'].split(',')
    values_list = request.form['values'].split(',')
    print(table_name, column_names_list, values_list)
    for i in range(len(column_names_list)):
        values_list[i] = change_type(values_list[i])
        if values_list[i] == '':
            values_list.pop(i)
            column_names_list.pop(i)
        else:
            if type(values_list[i]) == str:
                values_list[i] = "'{}'".format(values_list[i])
    column_names = ",".join(column_names_list)
    values = ",".join([str(value) for value in values_list])
    try:
        print("insert into {} ({}) values ({})".format(table_name, column_names, values))
        maintainer_cursor.execute("insert into {} ({}) values ({})".format(table_name, column_names, values))
        maintainer_db.commit()
        return {'success': True, 'message': 'Data inserted successfully!'}
    except:
        return {'success': False, 'message': 'An error occurred while inserting the record.'}
```

Triggers:

```
DELIMITER //

CREATE TRIGGER check_sprint_fp_sessions
AFTER INSERT ON free_practice_results
FOR EACH ROW
BEGIN
  DECLARE is_sprint_weekend BOOLEAN;

  SELECT EXISTS (
    SELECT 1 FROM sprint_race_results
    WHERE season = NEW.season AND round = NEW.round
  ) INTO is_sprint_weekend;

  IF is_sprint_weekend = TRUE AND NEW.number IN (2, 3) THEN
    DELETE FROM free_practice_results
    WHERE season = NEW.season
    AND round = NEW.round
    AND number = NEW.number;

    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Record deleted: Sprint weekends cannot have FP2 or FP3 sessions';
  END IF;
END //

DELIMITER ;
```

The `check_sprint_fp_sessions` trigger enforces that during a Sprint weekend in Formula 1, only Free Practice 1 (FP1) sessions are valid. It deletes any attempted insertion of FP2 or FP3 results during such weekends and raises an error to maintain data integrity.

(Invoking of triggers specified in Functionalities and features)

Procedures:

```
DELIMITER //
CREATE PROCEDURE get_constructor_standings(IN season_year INT)
BEGIN
  SELECT
    ROW_NUMBER() OVER (ORDER BY SUM(points) DESC) AS position,
    constructor_id,
    SUM(points) AS points
  FROM (
    SELECT season, constructor_id, points FROM sprint_race_results
    UNION ALL
    SELECT season, constructor_id, points FROM race_results
  ) AS all_results
  WHERE season = season_year
  GROUP BY constructor_id
  ORDER BY points DESC;
END //
DELIMITER ;
```

This procedure retrieves the constructor standings for a specified season by aggregating points from both sprint race results and regular race results. It ranks constructors based on their total points.

```
DELIMITER //
CREATE PROCEDURE get_driver_standings(IN season_year INT)
BEGIN
  SELECT
    ROW_NUMBER() OVER (ORDER BY SUM(points) DESC) AS position,
    driver_id,
    constructor_id,
    SUM(points) AS points
  FROM (
    SELECT season, driver_id, constructor_id, points FROM sprint_race_results
    UNION ALL
    SELECT season, driver_id, constructor_id, points FROM race_results
  ) AS all_results
  WHERE season = season_year
  GROUP BY driver_id, constructor_id
  ORDER BY points DESC;
END //
DELIMITER ;
```

This procedure generates driver standings for a given season by summing points from sprint and regular race results. It ranks drivers based on their total points while also displaying their associated constructor.

Invoking the procedure:

```python
@app.route('/championship_points/<year>')
def championship_points(year):
    year = int(year)
    user_cursor.callproc("get_driver_standings", [year])
    driver_standings = []
    for result in user_cursor.stored_results():
        driver_standings.extend(result.fetchall())
    user_cursor.callproc("get_constructor_standings", [year])
    constructor_standings = []
    for result in user_cursor.stored_results():
        constructor_standings.extend(result.fetchall())
    return {'driver_standings': driver_standings, 'constructor_standings': constructor_standings}
```

Functions:

```sql
CREATE FUNCTION get_race_wins(id VARCHAR(100))
RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE race_wins INTEGER;
    SELECT COUNT(*) INTO race_wins FROM race_results WHERE position = 1 AND driver_id = id;
    RETURN race_wins;
END //

CREATE FUNCTION get_pole_positions(id VARCHAR(100))
RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE pole_positions INTEGER;
    SELECT COUNT(*) INTO pole_positions FROM quali_results WHERE position = 1 AND driver_id = id;
    RETURN pole_positions;
END //

CREATE FUNCTION get_podiums(id VARCHAR(100))
RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE podiums INTEGER;
    SELECT COUNT(*) INTO podiums FROM race_results WHERE position <= 3 AND driver_id = id;
    RETURN podiums;
END //

CREATE FUNCTION get_sprint_wins(id VARCHAR(100))
RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE sprint_wins INTEGER;
    SELECT COUNT(*) INTO sprint_wins FROM sprint_race_results WHERE position = 1 AND driver_id = id;
    RETURN sprint_wins;
END //
```

- **`get_race_wins` returns the total number of race wins for a given driver based on their ID by counting first-place finishes in `race_results`.**
- **`get_pole_positions` retrieves the total number of pole positions for a given driver by counting first-place qualifying results in `quali_results`.**
- **`get_podiums` returns the number of podium finishes (top 3 positions) for a specified driver based on their results in `race_results`.**
- **`get_sprint_wins` calculates the total number of sprint race wins for a driver by counting first-place finishes in `sprint_race_results`.**

Invoking the function:

```python
@app.route('/career_achievements/<driver_id>')
def get_wins(driver_id):
    user_cursor.execute(f"select get_race_wins('{driver_id}') from driver")
    race_wins = user_cursor.fetchall()[0][0]
    user_cursor.execute(f"select get_pole_positions('{driver_id}') from driver")
    pole_positions = user_cursor.fetchall()[0][0]
    user_cursor.execute(f"select get_podiums('{driver_id}') from driver")
    podiums = user_cursor.fetchall()[0][0]
    user_cursor.execute(f"select get_sprint_wins('{driver_id}') from driver")
    sprint_wins = user_cursor.fetchall()[0][0]
    return {'race_wins': race_wins, 'pole_positions': pole_positions, 'podiums': podiums, 'sprint_wins': sprint_wins}
```

Nested Query:

```sql
SELECT
  ROW_NUMBER() OVER (ORDER BY SUM(points) DESC) AS position,
  constructor_id,
  SUM(points) AS points
FROM (
  SELECT season, constructor_id, points FROM sprint_race_results
  UNION ALL
  SELECT season, constructor_id, points FROM race_results
) AS all_results
WHERE season = season_year
GROUP BY constructor_id
ORDER BY points DESC;
```

The nested query merges `season`, `constructor_id`, and `points` data from `sprint_race_results` and `race_results` tables to aggregate total points for constructors across all races.

```sql
SELECT
  ROW_NUMBER() OVER (ORDER BY SUM(points) DESC) AS position,
  driver_id,
  constructor_id,
  SUM(points) AS points
FROM (
  SELECT season, driver_id, constructor_id, points FROM sprint_race_results
  UNION ALL
  SELECT season, driver_id, constructor_id, points FROM race_results
) AS all_results
WHERE season = season_year
GROUP BY driver_id, constructor_id
ORDER BY points DESC;
```

The nested query combines `season`, `driver_id`, `constructor_id`, and `points` data from `sprint_race_results` and `race_results` tables to calculate total points for drivers across both types of races.

Join Query:

```
CREATE PROCEDURE get_driver_and_race_stats(IN p_driver_id VARCHAR(100), IN p_season INT, IN p_round INT)
BEGIN
  SELECT
    rr.position AS race_position,
    rr.constructor_id AS race_constructor_id,
    rr.laps AS race_laps,
    rr.time AS race_time,
    rr.time_penalty AS race_time_penalty,
    rr.reason_retired AS race_reason_retired,
    rr.points AS race_points,
    rr.grid_position AS race_grid_position,
    ps.position AS pit_stop_position,
    ps.stop AS pit_stop_number,
    ps.lap AS pit_stop_lap,
    ps.time AS pit_stop_time,
    qr.position AS quali_position,
    qr.q1 AS quali_q1,
    qr.q2 AS quali_q2,
    qr.q3 AS quali_q3,
    qr.laps AS quali_laps,
    srr.position AS sprint_race_position,
    srr.laps AS sprint_race_laps,
    srr.time AS sprint_race_time,
    srr.time_penalty AS sprint_race_time_penalty,
    srr.reason_retired AS sprint_race_reason_retired,
    srr.points AS sprint_race_points,
    srr.grid_position AS sprint_race_grid_position,
    sqr.position AS sprint_quali_position,
    sqr.q1 AS sprint_quali_q1,
    sqr.q2 AS sprint_quali_q2,
    sqr.q3 AS sprint_quali_q3,
    sqr.laps AS sprint_quali_laps,
    ddr.position AS driver_of_day_position,
    ddr.percentage AS driver_of_day_percentage
  FROM race_results rr
  LEFT JOIN driver_of_the_day_results ddr
    ON ddr.season = rr.season AND ddr.round = rr.round AND ddr.driver_id = rr.driver_id
  LEFT JOIN quali_results qr
    ON rr.season = qr.season AND rr.round = qr.round AND rr.driver_id = qr.driver_id
  LEFT JOIN pit_stops ps
    ON rr.season = ps.season AND rr.round = ps.round AND rr.driver_id = ps.driver_id
  LEFT JOIN sprint_quali_results sqr
    ON rr.season = sqr.season AND rr.round = sqr.round AND rr.driver_id = sqr.driver_id
  LEFT JOIN sprint_race_results srr
    ON rr.season = srr.season AND rr.round = srr.round AND rr.driver_id = srr.driver_id
  WHERE rr.driver_id = p_driver_id
    AND rr.round = p_round AND rr.season = p_season;
END //

DELIMITER ;
```

This query retrieves comprehensive performance data for a specific driver in a given round and season by joining `race_results` with related tables such as `driver_of_the_day_results`, `quali_results`, `pit_stops`, `sprint_quali_results`, and `sprint_race_results`. It combines data on race, qualifying, pit stops, sprint events, and driver performance awards to provide a detailed view of the driver's performance metrics and standings.

Aggregate Query:

```
SELECT COUNT(*) INTO race_wins FROM race_results WHERE position = 1 AND driver_id = id;
```

The aggregate query counts the total number of race wins (first-place finishes) for a specified driver in the `race_results` table based on their `driver_id`.

```
SELECT
  ROW_NUMBER() OVER (ORDER BY SUM(points) DESC) AS position,
  constructor_id,
  SUM(points) AS points
FROM (
  SELECT season, constructor_id, points FROM sprint_race_results
  UNION ALL
  SELECT season, constructor_id, points FROM race_results
) AS all_results
WHERE season = season_year
GROUP BY constructor_id
ORDER BY points DESC;
```

This aggregate query calculates and ranks constructors based on their total points in a specified season by summing points from both `sprint_race_results` and `race_results` tables.