

ISTANBUL TECHNICAL UNIVERSITY

BLG317E
Database Systems

Formula 1 Project Report

150200062: Yasin İbiş
150200064: Semih Gençten
150200089: Eda Işık

Contents

1. Abstract	4
2. Home Page.....	4
2.1. Top 5 Drivers SQL Query	4
2.2. Top 5 Constructors SQL Query.....	5
2.3. Last 5 Races Query.....	5
3. Drivers Page	6
3.1. Index of Drivers Page	6
3.1.1. Query for Search.....	7
3.1.2. Query for Insert	8
3.1.3. Delete Query.....	8
3.1.4. Update Query	9
3.2. Details Page for Drivers.....	9
3.2.1. Query for First Table	10
3.2.2. Query for Second Table	11
4. Constructors Page.....	12
4.1. Index of Constructors Page.....	12
4.1.1. User Operations.....	12
4.1.1.1. Search and Filtering.....	12
4.1.1.2. Pagination.....	13
4.1.2. Admin Operations.....	14
4.1.2.1. Create Constructor	14
4.1.2.2. Update Constructor	14
4.1.2.3. Delete Constructor	15
4.2. Details Page of Constructors.....	16
4.2.1. Main Table	16
4.2.2. Extra Details.....	17
5. Races Page	18
5.1. User Functions Overview of Races Page.....	18
5.1.1. Search and Filtering	18
5.1.1.1. Basic Search.....	18
5.1.1.2. Advanced Search	18
5.1.2. Race Details	19
5.1.3. Pagination.....	19
5.1.4. Admin Functions.....	19
5.1.4.1. Create	19

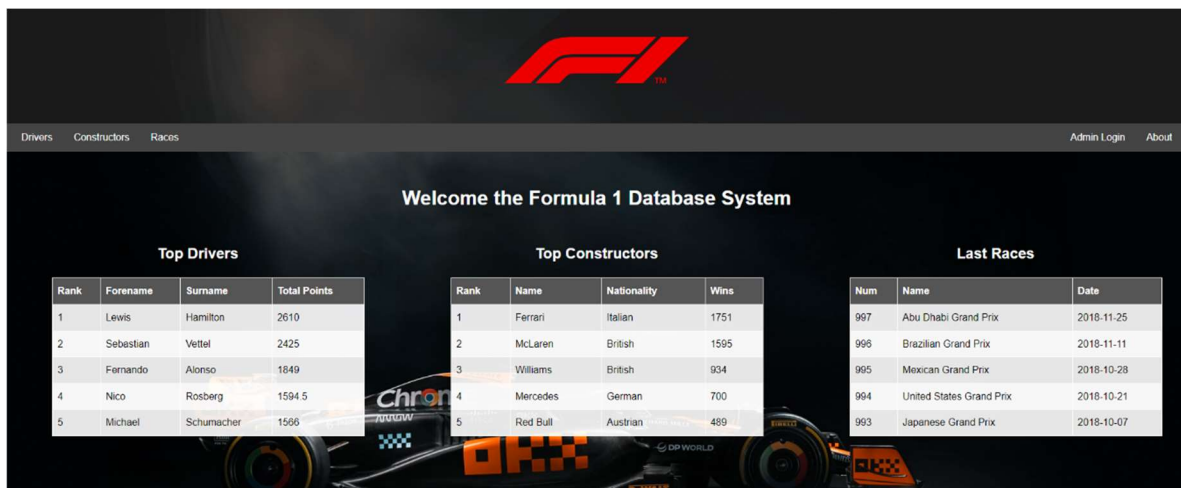
5.1.4.2.	Edit and Delete	20
5.2.	Code Outlines and Queries for the Races Page	20
5.2.1.	Main Search Function	20
5.2.1.1.	get_search_results Function	20
5.3.	Routes Definition	22
5.3.1.	Main Races Page	22
5.3.2.	Race Creation.....	22
5.3.2.1.	Inserting Circuit Information	22
5.3.2.2.	Inserting Race Information	23
5.3.3.	Race Deletion.....	23
5.3.3.1.	Deleting Race Entries.....	23
5.3.4.	Race Update	23
5.3.4.1.	Updating Race Information	24
5.3.4.2.	Updating Circuit Information.....	24
6.	Log In Admin Module	25
7.	About Page	26
8.	ERD Diagram	27
9.	About Members and Responsibilities.....	28

1. Abstract

Our Flask-based web application for the Database class represents a comprehensive system that seamlessly integrates SQL functionality for managing information related to drivers, constructors, and races in the realm of motorsports. The user interface offers dynamic pages for each category, allowing users to perform efficient search operations and access detailed information on individual drivers, constructors, and races. With a user-friendly design, the application empowers users to explore and analyze the data effortlessly. For administrators, the system provides robust functionality, enabling them to execute critical operations such as creating, updating, and deleting records within the respective tables. This project not only serves as an educational tool for learning SQL in a practical setting but also demonstrates the effective implementation of a web-based database management system tailored for motorsports enthusiasts.

2. Home Page

Home page is the entering point of the web application and it greets the guests. We considered that showing the best drivers, races and last races in motorsports is a good idea for home page. You can find the best 5 drivers according to their total points, the best 5 constructors according to their total wins, also the last 5 races.



Top Drivers			
Rank	Forename	Surname	Total Points
1	Lewis	Hamilton	2610
2	Sebastian	Vettel	2425
3	Fernando	Alonso	1849
4	Nico	Rosberg	1594.5
5	Michael	Schumacher	1566

Top Constructors			
Rank	Name	Nationality	Wins
1	Ferrari	Italian	1751
2	McLaren	British	1595
3	Williams	British	934
4	Mercedes	German	700
5	Red Bull	Austrian	489

Last Races		
Num	Name	Date
997	Abu Dhabi Grand Prix	2018-11-25
996	Brazilian Grand Prix	2018-11-11
995	Mexican Grand Prix	2018-10-28
994	United States Grand Prix	2018-10-21
993	Japanese Grand Prix	2018-10-07

2.1. Top 5 Drivers SQL Query

```
'SELECT '
' ROW_NUMBER() OVER (ORDER BY SUM(res.points) DESC) AS row_num, '
' d.forename, '
' d.surname, '
' SUM(res.points) as sum_of_points'
' FROM drivers d'
' JOIN results res ON res.driverId=d.driverId'
' GROUP BY d.driverId'
' ORDER BY sum_of_points desc'
' LIMIT 5'
```

The query selects data from drivers table and joins it with results table to get points of the drivers. Since driverId column is foreign key, it uses that column while making join operation. The result is grouped by the column driverId to sum up points of each driver separately. SUM(res.points)

adds together all points of drivers. The result is first 5 instance which is ordered by sum of points. ROW_NUMBER() command gives number according to given parameter after OVER keyword, in this way we can numerate the result.

2.2. Top 5 Constructors SQL Query

```
' SELECT '  
' ROW_NUMBER() OVER (ORDER BY SUM(cs.wins) DESC) AS row_num, '  
' c.name, '  
' c.nationality, '  
' SUM(cs.wins) AS sum_of_wins '  
' FROM constructor_standings cs '  
' JOIN constructors c ON c.constructorId = cs.constructorId '  
' GROUP BY cs.constructorId '  
' ORDER BY sum_of_wins DESC '  
' LIMIT 5 '
```

The query makes a similar operation with the query for drivers. It selects from constructor_standings table and joins it with constructors table, groups by constructorId to sum up all wins for each constructor separately. Sum(cs.wins) command gives the result of summation of wins for each constructor. Row number is given in the same way.

2.3. Last 5 Races Query

```
' SELECT '  
' ROW_NUMBER() OVER (ORDER BY date) AS row_num, '  
' r.name, '  
' r.date '  
' FROM races r '  
' ORDER BY r.date DESC '  
' LIMIT 5 '
```

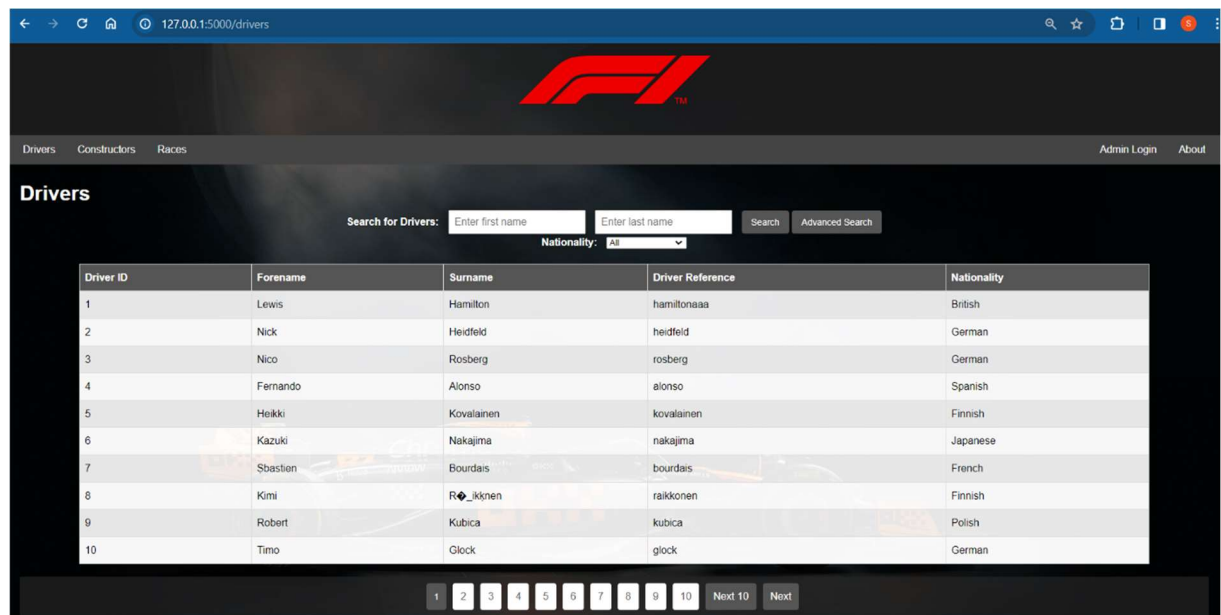
Query selects the last five races according to year of the race by using ORDER BY and LIMIT keywords. Also gives row numbers like the previous queries.

3. Drivers Page

3.1. Index of Drivers Page

Driver page shows general information of drivers by using 'drivers' table, also give the chance to see detailed information if you click the row of that driver. User can see driver Id, forename, surname, driver reference and nationality information of every driver in this page. By using search bar, you can make search operation with forename and surname, you can also filter the results with the nationality of drivers.

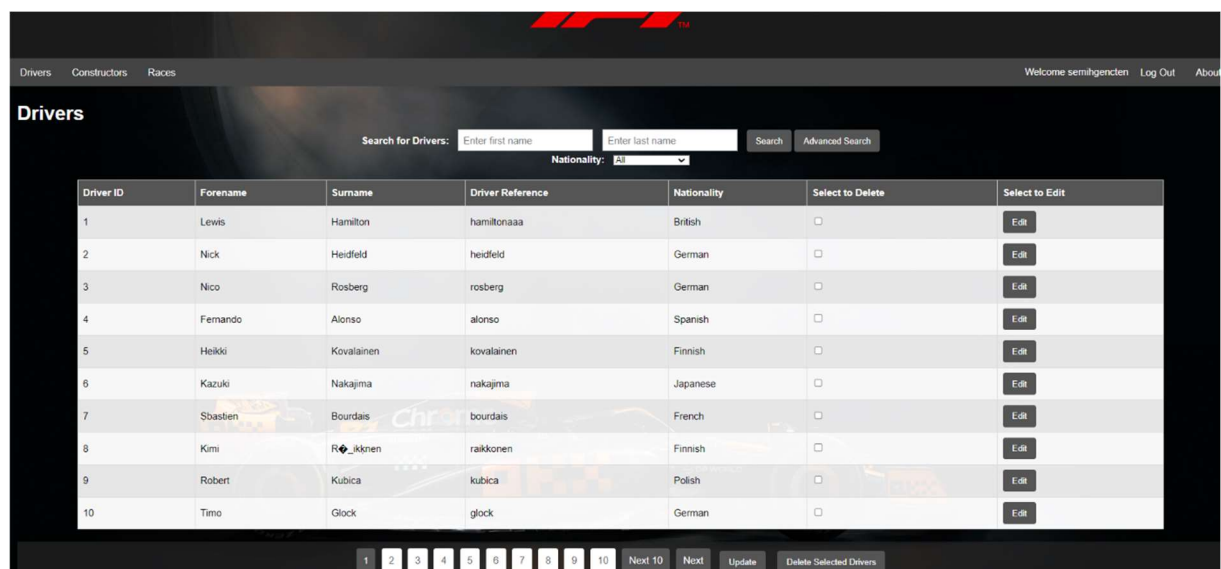
To enhance the both performance and user experience we developed a pagination procedure all of our pages. You can see 10 rows at most on the page, this property makes it looking more beautiful, easier to use and simple, also reduces query time dramatically since we get only 10 rows instead of all of the instances from the table.



The screenshot shows the 'Drivers' page for guests. It features a search bar with fields for 'Enter first name' and 'Enter last name', a 'Search' button, and an 'Advanced Search' button. Below the search bar is a 'Nationality' dropdown menu set to 'All'. The table displays 10 drivers with columns: Driver ID, Forename, Surname, Driver Reference, and Nationality. The pagination bar at the bottom shows 10 pages, with 'Next 10' and 'Next' buttons.

Driver ID	Forename	Surname	Driver Reference	Nationality
1	Lewis	Hamilton	hamiltonaaa	British
2	Nick	Heidfeld	heidfeld	German
3	Nico	Rosberg	rosberg	German
4	Fernando	Alonso	alonso	Spanish
5	Heikki	Kovalainen	kovalainen	Finnish
6	Kazuki	Nakajima	nakajima	Japanese
7	Sbastien	Bourdais	bourdais	French
8	Kimi	Räikkönen	raikkonen	Finnish
9	Robert	Kubica	kubica	Polish
10	Timo	Glock	glock	German

Above looking is for guests of the website, there is also an admin looking which allows the admins make create, update, delete operations as you can see below.

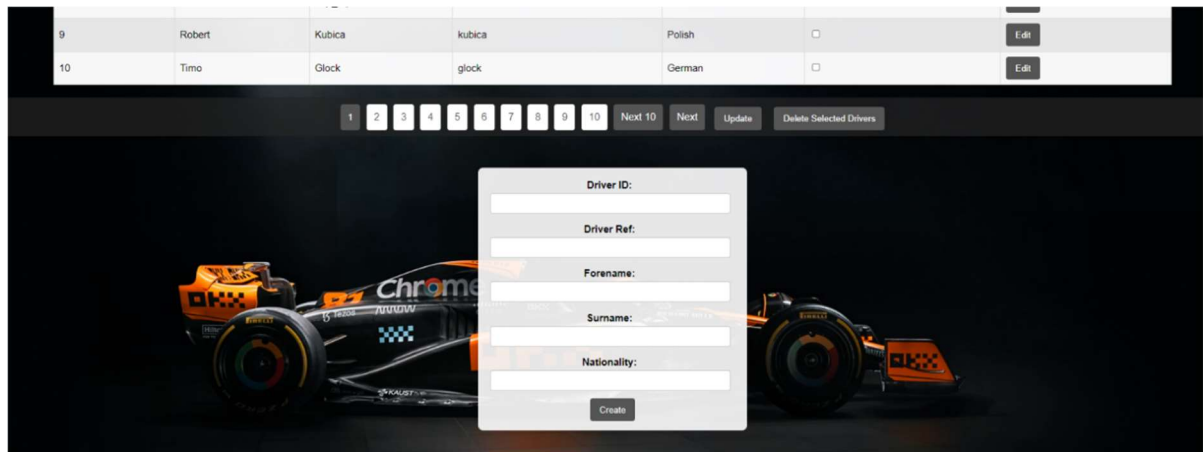


The screenshot shows the 'Drivers' page for admins. It features the same search bar and 'Nationality' dropdown as the guest view. The table displays 10 drivers with columns: Driver ID, Forename, Surname, Driver Reference, Nationality, Select to Delete, and Select to Edit. The pagination bar at the bottom shows 10 pages, with 'Next 10', 'Next', 'Update', and 'Delete Selected Drivers' buttons.

Driver ID	Forename	Surname	Driver Reference	Nationality	Select to Delete	Select to Edit
1	Lewis	Hamilton	hamiltonaaa	British	<input type="checkbox"/>	<button>Edit</button>
2	Nick	Heidfeld	heidfeld	German	<input type="checkbox"/>	<button>Edit</button>
3	Nico	Rosberg	rosberg	German	<input type="checkbox"/>	<button>Edit</button>
4	Fernando	Alonso	alonso	Spanish	<input type="checkbox"/>	<button>Edit</button>
5	Heikki	Kovalainen	kovalainen	Finnish	<input type="checkbox"/>	<button>Edit</button>
6	Kazuki	Nakajima	nakajima	Japanese	<input type="checkbox"/>	<button>Edit</button>
7	Sbastien	Bourdais	bourdais	French	<input type="checkbox"/>	<button>Edit</button>
8	Kimi	Räikkönen	raikkonen	Finnish	<input type="checkbox"/>	<button>Edit</button>
9	Robert	Kubica	kubica	Polish	<input type="checkbox"/>	<button>Edit</button>
10	Timo	Glock	glock	German	<input type="checkbox"/>	<button>Edit</button>

When the admin is logged in to the app, he can edit the columns of the rows by clicking the edit button. When you click the update button at the bottom it calls the related query to update the edited columns. When you select the rows by clicking the checkboxes, you can delete them by clicking 'delete selected drivers' button.

At the bottom of the page, you can make create operation which calls the insert query for the table drivers as you can see below.



3.1.1. Query for Search

```
'SELECT d.driverId, d.driverRef, d.forename, d.surname, d.nationality'
' FROM drivers d '
' WHERE d.forename LIKE ? AND d.surname LIKE ? AND (CASE WHEN ? IS NULL
THEN 1=1 ELSE nationality=? END)'
' ORDER BY d.driverId'
f' LIMIT {RESULTS_PER_PAGE} OFFSET {offset}'
```

The above query runs when the user entered forename and surname , as you can see in WHERE clause, it filter the results according to both forename and surname. It also checks is nationality null, if it is not null then it adds this filter and fetches only the drivers who are in that nationality. The results are ordered by driverId and limited to a specific page size with an offset for pagination.

The question marks are parameters, the related parameters are given in python code. As an example we will give the parameters here:

```
posts = db.execute(query, (search_term_with_percent,
search_term_surname_with_percent,nationality_filter,
nationality_filter)).fetchall()
```

```
'SELECT d.driverId, d.driverRef, d.forename, d.surname, d.nationality'
' FROM drivers d '
' WHERE d.forename LIKE ? AND (CASE WHEN ? IS NULL THEN 1=1
ELSE nationality=? END)'
' ORDER BY d.driverId'
f' LIMIT {RESULTS_PER_PAGE} OFFSET {offset}'
```

This query will be used when the user gives only forename as a search term.

The condition check for the search terms is applied in the python code.

```
'SELECT d.driverId, d.driverRef, d.forename, d.surname, d.nationality'
' FROM drivers d'
' WHERE d.surname LIKE ? AND (? IS NULL OR d.nationality =?)'
' ORDER BY d.driverId'
f' LIMIT {RESULTS_PER_PAGE} OFFSET {offset}'
```

This query will be called when the user gives only the surname.

There is one more query which will be used when the user do not give any of the forename or surname. It is similar to these queries.

There are also total_driver queries which helps us to compute the number of the rows as result of the search query. It is used to make pagination operations.

```
SELECT COUNT(*) FROM drivers WHERE forename LIKE ? AND (CASE WHEN ? IS NULL
THEN 1=1 ELSE nationality=? END)
```

This query gives the count of results by considering the restrictions in the WHERE clause. The case when structure is similar to above queries, it checks whether nationality is null.

3.1.2. Query for Insert

```
INSERT INTO drivers (driverId, driverRef, forename, surname,
nationality)
VALUES (?, ?, ?, ?, ?)
```

This query adds a new row to 'drivers' table with the given values. The values is replaced with question marks because they are given as parameters in the python code. Parameters:

```
(driverId, driverRef, forename, surname, nationality)
```

3.1.3. Delete Query

Delete query is written in a different approach since the number of drivers that will be deleted is uncertain. The first part is written normally but the following part is parametrized with the help

of loops in python. By iterating through driver_id's which will be deleted, we concatenated them to 'delete_query' and a comma is added at the end of the query.

```
delete_query = f"DELETE FROM drivers WHERE driverId in ("
    for driver_id in driver_ids:
        if driver_id != driver_ids[0]:
            delete_query += ", "
        delete_query += f" {driver_id}"
    delete_query += ")"
    db.execute(delete_query, )
    db.commit()
```

In the provided code snippet, 'db.commit()' is a command that finalizes and confirms the changes made to the database after executing the DELETE query. In database operations, changes are often performed within a transaction, and the 'commit' method is responsible for persisting these changes permanently.

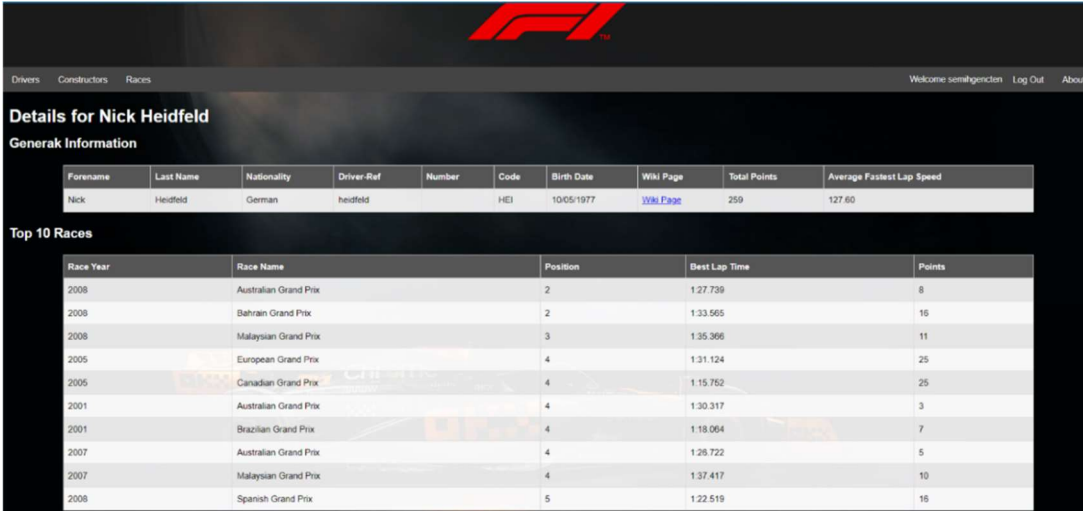
3.1.4. Update Query

```
query = ('UPDATE drivers SET forename = ?, surname = ?, driverRef = ?,
nationality = ? '
' WHERE driverId = ?')
db.execute(query, (forename, surname, driverRef, nationality, driver_id))
db.commit()
```

The update query for drivers table which is setting the given values is given. It is parametrized with question mark and related parameters are shown.

3.2. Details Page for Drivers

When the user clicks anywhere on the row, app goes to details page for that driver. In this page there are two tables, first shows the general information, total points and average fastest lap speed of the driver. The second table shows the top 10 races for the driver according to his position in the race.



Details for Nick Heidfeld

General Information

Forename	Last Name	Nationality	Driver Ref	Number	Code	Birth Date	Wiki Page	Total Points	Average Fastest Lap Speed
Nick	Heidfeld	German	heidfeld		HEI	10/05/1977	Wiki Page	259	127.80

Top 10 Races

Race Year	Race Name	Position	Best Lap Time	Points
2008	Australian Grand Prix	2	1:27.739	8
2008	Bahrain Grand Prix	2	1:33.565	16
2008	Malaysian Grand Prix	3	1:35.366	11
2005	European Grand Prix	4	1:31.124	25
2005	Canadian Grand Prix	4	1:15.752	25
2001	Australian Grand Prix	4	1:30.317	3
2001	Brazilian Grand Prix	4	1:18.064	7
2007	Australian Grand Prix	4	1:28.722	5
2007	Malaysian Grand Prix	4	1:37.417	10
2008	Spanish Grand Prix	5	1:22.519	16

3.2.1. Query for First Table

```
general_info_query = (  
    'SELECT '  
    ' d.forename,'  
    ' d.surname,'  
    ' d.nationality,'  
    ' d.driverRef,'  
    ' d.number,'  
    ' d.code,'  
    ' d.dob,'  
    ' d.url,'  
    ' SUM(res.points) AS sum_of_points'  
    ' , AVG(fastestLapSpeed) AS average_fastest_speed'  
    ' from drivers d'  
    ' JOIN results res on d.driverId=res.driverId '  
    ' JOIN races r ON r.raceId=res.raceId'  
    ' WHERE d.driverId = {driver_id}'  
    ' GROUP BY d.driverId'  
    )
```

The provided SQL query retrieves general information about a specific driver which the user intended to see details for from the "drivers" table. It selects the driver's forename, surname, nationality, driver reference, number, code, date of birth, and URL. Additionally, it calculates the sum of points and the average fastest lap speed for the specified driver by joining the "results" table on driverId and the "races" table on raceId. The query filters results based on the driver's driverId, and the grouping is done on the driverId to aggregate the data. This query is designed to consolidate and present comprehensive details about the driver, including performance metrics such as total points and average fastest lap speed.

3.2.2. Query for Second Table

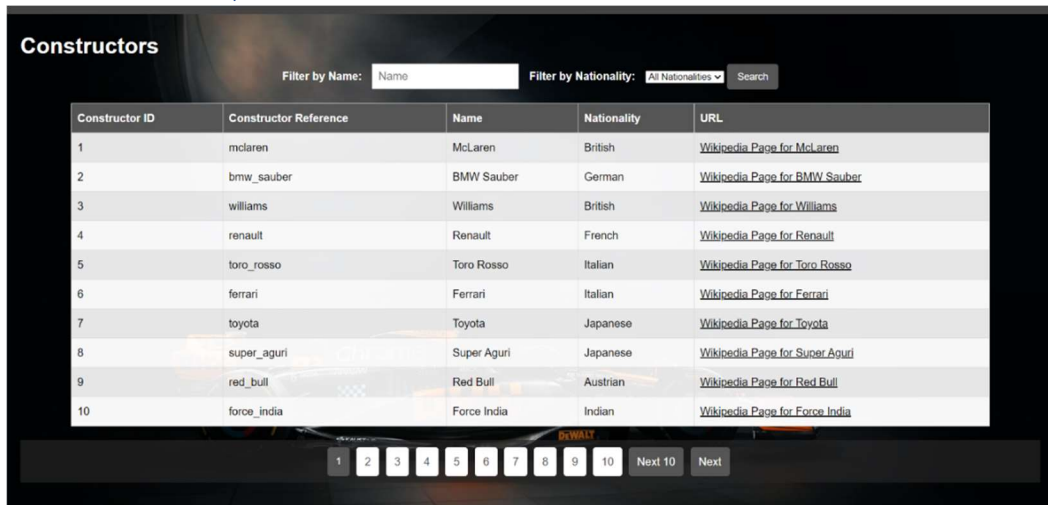
```
details_query = (  
'select d.forename, d.surname, r.year , r.name, ds.position, ds.points,  
bestLap.time as time'  
'from drivers d'  
' join driver_standings ds on d.driverId = ds.driverId'  
' join races r on ds.raceId = r.raceId'  
' join ('  
' SELECT l.raceId, l.driverId, MIN(l.time) AS time          FROM laptimes l  
GROUP BY l.raceId, l.driverId'  
) AS bestLap'  
' ON bestLap.raceId = r.raceId AND bestLap.driverId = d.driverId'  
' where d.driverId = ?'  
' order by position '  
' limit 10'  
)
```

This query is designed to get the best 10 races of the driver according to its lap times. The query retrieves detailed information about drivers, including their forename, surname, the year and name of races, position, points, and the best lap time. It accomplishes this by joining the "drivers" table with the "driver_standings" table on driverId, linking the "driver_standings" with the "races" table based on raceId, and further connecting with a subquery that identifies the best lap time for each driver in each race from the "laptimes" table. The subquery uses the MIN() function to find the minimum lap time. The results are organized to present a comprehensive overview of driver performance, including specific race details and the best lap time achieved.

4. Constructors Page

4.1. Index of Constructors Page

4.1.1. User Operations



The screenshot shows the 'Constructors' page with a table of 10 constructors. The table has columns: Constructor ID, Constructor Reference, Name, Nationality, and URL. The filters are 'Filter by Name: Name' and 'Filter by Nationality: All Nationalities'. The table lists constructors from McLaren to Force India.

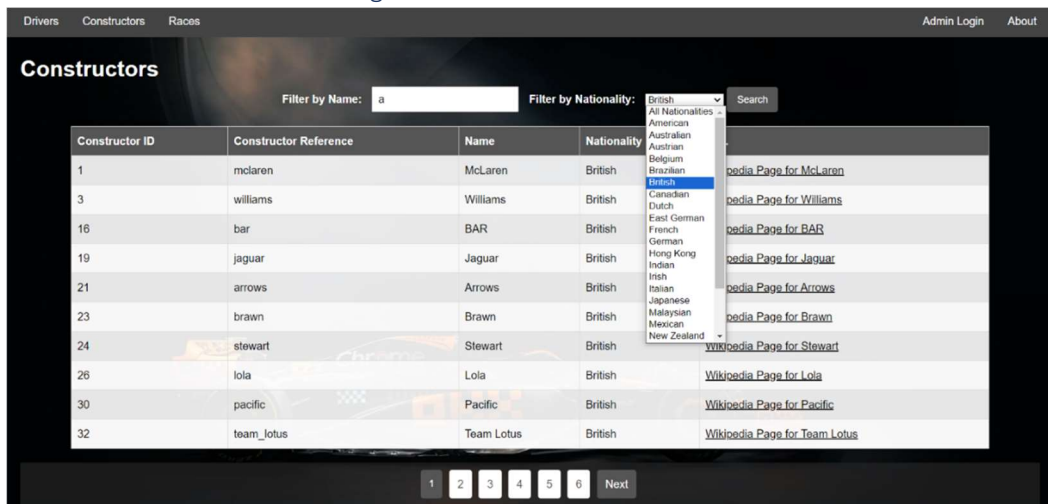
Constructor ID	Constructor Reference	Name	Nationality	URL
1	mclaren	McLaren	British	Wikipedia Page for McLaren
2	bmw_sauber	BMW Sauber	German	Wikipedia Page for BMW Sauber
3	williams	Williams	British	Wikipedia Page for Williams
4	renault	Renault	French	Wikipedia Page for Renault
5	toro_rosso	Toro Rosso	Italian	Wikipedia Page for Toro Rosso
6	ferrari	Ferrari	Italian	Wikipedia Page for Ferrari
7	toyota	Toyota	Japanese	Wikipedia Page for Toyota
8	super_aguri	Super Aguri	Japanese	Wikipedia Page for Super Aguri
9	red_bull	Red Bull	Austrian	Wikipedia Page for Red Bull
10	force_india	Force India	Indian	Wikipedia Page for Force India

When a user enters to the *Constructors* page, a base table shows up. The base table consist of a not filtered Constructors table with columns Constructor Id, Constructor Reference, Name, Nationality and URL which directs users to relevant Wikipedia pages for additional information.

The query for base appearance is below:

```
base_query = "SELECT * FROM constructors WHERE 1 = 1 "
```

4.1.1.1. Search and Filtering



The screenshot shows the 'Constructors' page with a search bar and a dropdown menu for Nationality. The search bar contains 'a' and the Nationality dropdown is open, showing a list of nationalities. The table lists constructors from McLaren to Team Lotus.

Constructor ID	Constructor Reference	Name	Nationality	URL
1	mclaren	McLaren	British	Wikipedia Page for McLaren
3	williams	Williams	British	Wikipedia Page for Williams
16	bar	BAR	British	Wikipedia Page for BAR
19	jaguar	Jaguar	British	Wikipedia Page for Jaguar
21	arrows	Arrows	British	Wikipedia Page for Arrows
23	brawn	Brawn	British	Wikipedia Page for Brawn
24	stewart	Stewart	British	Wikipedia Page for Stewart
26	lola	Lola	British	Wikipedia Page for Lola
30	pacific	Pacific	British	Wikipedia Page for Pacific
32	team_lotus	Team Lotus	British	Wikipedia Page for Team Lotus

Users can search and filter for constructors by name and nationality. Name input should be entered by user and nationality should be selected from the dropdown list. The results can be filtered by only name, only nationality or both of them. The query for these operations are built like below:

```
if filter_name:
    base_query += f"AND lower(name) LIKE '%{filter_name}%'"
if nationality_dropdown:
    base_query += f"AND lower(nationality) LIKE '%{nationality_dropdown}%'"
```

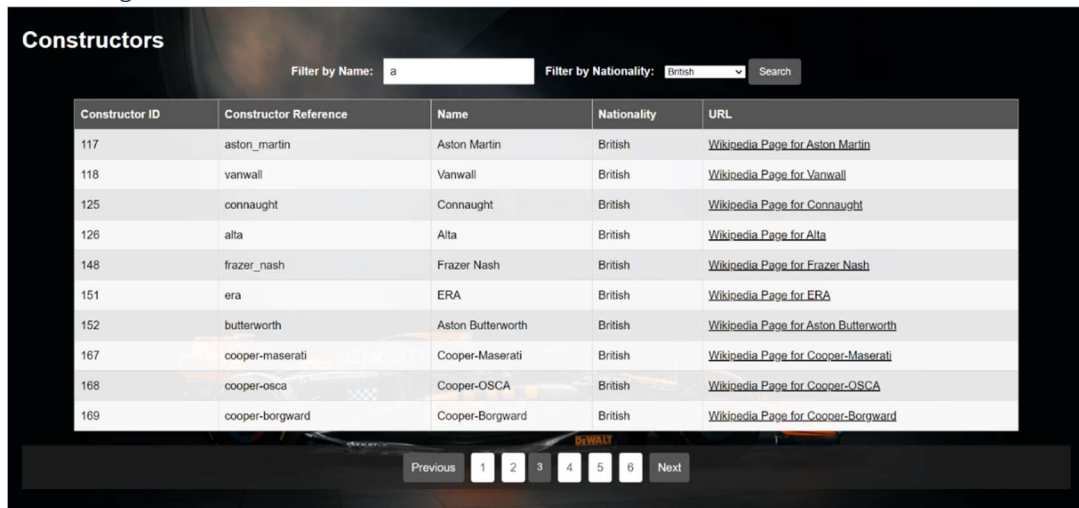
Search inputs for name and nationality are taken from user by filter_name and nationality_dropdown, respectively. lower() function is used for a better search experience. Additionally, % symbol is used in both sides of filter_name because it was desired for every record containing the searched item to be displayed.

Filtering part of queries are added to the base_query with += operation of python.

```
unique_nationalities = con.execute("SELECT DISTINCT nationality FROM
constructors ORDER BY nationality").fetchall()
```

Nationality dropdown list is created by the query above.

4.1.1.2. Pagination



Pagination feature can be seen above screenshot. It makes the webpage more useful. The pagination feature can also works at filtered results.

```
query = base_query + f" LIMIT {RESULTS_PER_PAGE} OFFSET {offset}"
```

With the help of the code line above, pagination is ensured. RESULT_PER_PAGE is 10 defaultly and offset is determined according to the page selected. For example, for page 5, offset will be 40 here.

```
total_count_query = "SELECT COUNT(*) FROM constructors WHERE 1 = 1 "
if filter_name:
    total_count_query += f"AND lower(name) LIKE '%{filter_name}%'"
if nationality_dropdown:
    total_count_query += f"AND lower(nationality) LIKE '%{nationality_dropdown}%'"
```

total_count_query is used for showing how much page can be visited at the filtered results. COUNT() function of sql is used for calculating total count, filtering queries are also added to total_count_query like base_query.

4.1.2. Admin Operations

When a user logs in, the admin operations are enabled, which are Create, Update and Delete.

4.1.2.1. Create Constructor

ID	Constructor Reference	Name	Nationality	URL	Select to Delete	Select to Edit
8	super_aguri	Super Aguri	Japanese	Wikipedia Page for Super Aguri	<input type="checkbox"/>	<button>Edit</button>
9	red_bull	Red Bull	Austrian	Wikipedia Page for Red Bull	<input type="checkbox"/>	<button>Edit</button>
10	force_india	Force India	Indian	Wikipedia Page for Force India	<input type="checkbox"/>	<button>Edit</button>

1 2 3 4 5 6 7 8 9 10 Next 10 Next Delete Selected Constructors Update

Constructor ID:

Constructor Ref:

Name:

Nationality:

Uri:

Create

Create interface is like above. It is located below the results table, which can be seen from the screenshot. All fields are required. When admin fills the input fields and clicks the Create button, the below code works:

```
db.execute(
    'INSERT INTO constructors (constructorId, constructorRef, name,
nationality, url)'
    ' VALUES (?, ?, ?, ?, ?)',
    (constructorId, constructorRef, name, nationality, url)
)
db.commit()
```

'?' 's are parameters for this query. The parameters are taken from admin by using create interface. db.execute() executes the query and then db.commit() saves the changes to the database.

4.1.2.2. Update Constructor

Constructors

Filter by Name: Filter by Nationality: Search

Constructor ID	Constructor Reference	Name	Nationality	URL	Select to Delete	Select to Edit
1	mclaren	McLaren	British	Wikipedia Page for McLaren	<input type="checkbox"/>	<button>Edit</button>
2	bmw_sauber	BMW Sauber	German	http://en.wikipedia.org/wiki/Bi Wikipedia Page for BMW Sauber	<input type="checkbox"/>	<button>Edit</button>
3	williams	Williams	British	Wikipedia Page for Williams	<input type="checkbox"/>	<button>Edit</button>
4	renault	Renault	French	Wikipedia Page for Renault	<input type="checkbox"/>	<button>Edit</button>
5	toro_rosso	Toro Rosso	Italian	http://en.wikipedia.org/wiki/St Wikipedia Page for Toro Rosso	<input type="checkbox"/>	<button>Edit</button>
6	ferrari	Ferrari	Italian	Wikipedia Page for Ferrari	<input type="checkbox"/>	<button>Edit</button>
7	toyota	Toyota	Japanese	Wikipedia Page for Toyota	<input type="checkbox"/>	<button>Edit</button>
8	super_aguri	Super Aguri	Japanese	Wikipedia Page for Super Aguri	<input type="checkbox"/>	<button>Edit</button>
9	red_bull	Red Bull	Austrian	Wikipedia Page for Red Bull	<input type="checkbox"/>	<button>Edit</button>
10	force_india	Force India	Indian	Wikipedia Page for Force India	<input type="checkbox"/>	<button>Edit</button>

1 2 3 4 5 6 7 8 9 10 Next 10 Next Delete Selected Constructors Update

If the admin has logged in, selection and editing columns appear on the right of the results table. A record can be edited through the following steps: First, the update is initiated by clicking on the edit button of the record to be updated. The desired new values are then entered, and the update button at the bottom is pressed to save the changes.

The above screenshot shows an example for editing records. BMW Sauber and Toro Rosso can be edited by using the input fields. Previous versions are also shown for a better user experience.

```
query = ('UPDATE constructors SET constructorRef = ?, name = ?,
nationality = ?, url = ? WHERE constructorId = ? ')
db.execute(query, (constructorRef, name, nationality, url,
constructorId))
db.commit()
```

For updating constructors, above code works. '?' 's are again parameters and taken from admin. ConstructorId is constant here, meaning it can not be changed.

4.1.2.3. Delete Constructor

Constructors

Filter by Name: Name Filter by Nationality: All Nationalities Search

Constructor ID	Constructor Reference	Name	Nationality	URL	Select to Delete	Select to Edit
1	mclaren	McLaren	British	Wikipedia Page for McLaren	<input checked="" type="checkbox"/>	Edit
2	bmw_sauber	BMW Sauber	German	Wikipedia Page for BMW Sauber	<input checked="" type="checkbox"/>	Edit
3	williams	Williams	British	Wikipedia Page for Williams	<input type="checkbox"/>	Edit
4	renault	Renault	French	Wikipedia Page for Renault	<input checked="" type="checkbox"/>	Edit
5	toro_rosso	Toro Rosso	Italian	Wikipedia Page for Toro Rosso	<input type="checkbox"/>	Edit
6	ferrari	Ferrari	Italian	Wikipedia Page for Ferrari	<input type="checkbox"/>	Edit
7	toyota	Toyota	Japanese	Wikipedia Page for Toyota	<input type="checkbox"/>	Edit
8	super_aguri	Super Aguri	Japanese	Wikipedia Page for Super Aguri	<input type="checkbox"/>	Edit
9	red_bull	Red Bull	Austrian	Wikipedia Page for Red Bull	<input type="checkbox"/>	Edit
10	force_india	Force India	Indian	Wikipedia Page for Force India	<input type="checkbox"/>	Edit

1 2 3 4 5 6 7 8 9 10 Next 10 Next Delete Selected Constructors Update

For deleting constructor records, these steps should be followed: First, the records that wanted to be delete is selected, and “Delete Selected Constructors” button should be clicked.

Above screenshot is an example for deleting, if admin click on Delete Selected Constructors button now, McLaren, BMW Sauber and Renault will be deleted. Admin can select more or less constructors for deleting.

```
query = f'DELETE FROM constructors WHERE constructorId in ('
for const_id in constructorIds:
    if const_id != constructorIds[0]:
        query += ', '
        query += f'{const_id}'
query += ')"'
db.execute(query, )
db.commit()
```

Above code snippet is used for delete operation. constructorIds is coming from the constructors selected by the admin. With help of the for loop, the id's of the constructors will be deleted are added to the query. The query is executed and committed.

4.2. Details Page of Constructors

In the details page, there is nothing extra for admin. The features that users can utilize are introduced in the later subsections.

4.2.1. Main Table

Best Performances: Williams - Top 10 Races and Points

Race Year	Race Name	Circuit Name	Race Points
2014	Abu Dhabi Grand Prix	Yas Marina Circuit	66
2014	Austrian Grand Prix	Red Bull Ring	27
2014	Italian Grand Prix	Autodromo Nazionale di Monza	27
2015	Italian Grand Prix	Autodromo Nazionale di Monza	27
2012	Spanish Grand Prix	Circuit de Barcelona-Catalunya	25
2015	Austrian Grand Prix	Red Bull Ring	25
2015	Canadian Grand Prix	Circuit Gilles Villeneuve	23
2015	Mexican Grand Prix	Autódromo Hermanos Rodríguez	23
2014	United States Grand Prix	Circuit of the Americas	22
2015	British Grand Prix	Silverstone Circuit	22

When the user clicks anywhere at one of the records of the index page, the details page for that record opens. Above screenshot shows the details page of constructor Williams. Details page consists top 10 performance of selected constructor. It can be seen from the example, the race that Williams collects most points is Abu Dhabi Grand Prix. Additionally, year and circuit name of races can be seen in the details page.

```
name_query = f"SELECT name FROM constructors WHERE constructorRef = '{constructorRef}'"
const_name = con.execute(name_query).fetchone()[0]
id_query = f"SELECT constructorId FROM constructors WHERE constructorRef = '{constructorRef}'"
const_id = con.execute(id_query).fetchone()[0]
```

The app uses constructor references for url of this page. The name and id of the constructor are obtained by above queries. Constructor Id is used for upcoming queries and constructor name is used for title of the page.

```
query = ('SELECT r.year, cr.raceId, r.name as r_name, c.name as c_name, cr.points '
        'FROM constructor_results cr JOIN races r ON r.raceId = cr.raceId '
        'JOIN circuits c ON r.circuitId = c.circuitId '
        f'WHERE cr.constructorId = {const_id} ORDER BY cr.points DESC LIMIT 10')
```

The query for details page is above. constructor_results, races and circuits table were used. SELECT statement includes 5 columns however details page shows 4 columns. The unused column in the page will be used for further information. ORDER BY statement is used with DESC because descending order is needed here. LIMIT 10 is for showing the top 10.

4.2.2. Extra Details

Best Performances: Williams - Top 10 Races and Points			
Race Year	Race Name	Circuit Name	Race Points
2014	Abu Dhabi Grand Prix	Yas Marina Circuit	66
Points and Positions for Each Driver of Williams			
Driver Name		Position	Points Earned
Felipe Massa		2	36
Valtteri Bottas		3	30
2014	Austrian Grand Prix	Red Bull Ring	27
Points and Positions for Each Driver of Williams			
Driver Name		Position	Points Earned
Valtteri Bottas		3	15
Felipe Massa		4	12
2014	Italian Grand Prix	Autodromo Nazionale di Monza	27

If the user clicks to one of the records in details page, more details can be seen with the pop-up table which shows the drivers of the constructors, the positions and how much point that they gain at the clicked race.

At the screenshot, Felipe Massa and Valtteri Bottas can be seen and at Austrian Grand Prix, they became 4th and 3rd with 12 and 15 points respectively.

If the user clicks again to the record, the pop-up table will be close.

```
detail_query = ('SELECT r.constructorId, r.raceId, d.forename, d.surname,
r.position, r.points FROM results r'
                ' JOIN drivers d ON r.driverId = d.driverId WHERE r.raceId IN'
                ' (SELECT cr.raceId FROM constructor_results cr JOIN races r
ON r.raceId = cr.raceId'
                f' JOIN circuits c ON r.circuitId = c.circuitId WHERE
cr.constructorId = {const_id} ORDER BY cr.points DESC LIMIT 10)'
                f' AND r.constructorId = {const_id} ')
```

For the pop-up tables, the query above is used. The data is coming from results, drivers, constructor_results and races tables of database.

The query includes subquery:

```
(SELECT cr.raceId FROM constructor_results cr JOIN races r ON
r.raceId = cr.raceId'
    f' JOIN circuits c ON r.circuitId = c.circuitId WHERE
cr.constructorId = {const_id} ORDER BY cr.points DESC LIMIT 10)
```

The subquery returns the Race Ids of top 10 races. With the using of subquery, the desired result can be reached.

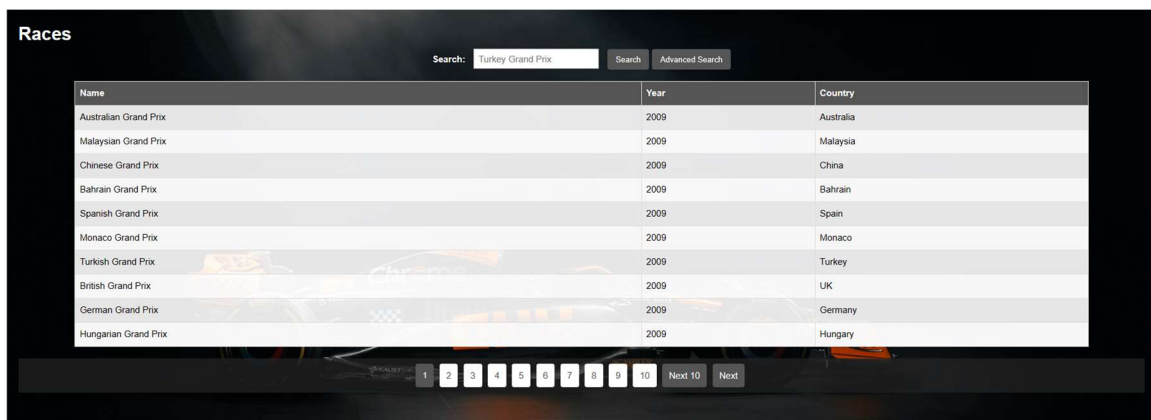
5. Races Page

The Races Page offers interface for exploring information about races, circuits, and race results. Users can initiate a search, either through a basic race name entry or an advanced search with filters for specific years and countries. The page provides detailed race information, including dates, rounds, and circuit details, accessible by clicking on a race name. With pagination for navigation, users can easily explore multiple pages of search results. For administrators, additional functionalities such as creating new races, editing details, and deleting entries are available.

5.1. User Functions Overview of Races Page

5.1.1. Search and Filtering

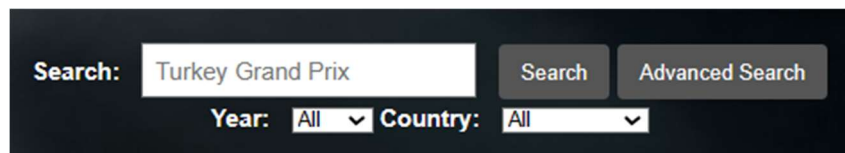
5.1.1.1. Basic Search



Start page of Races Search Page

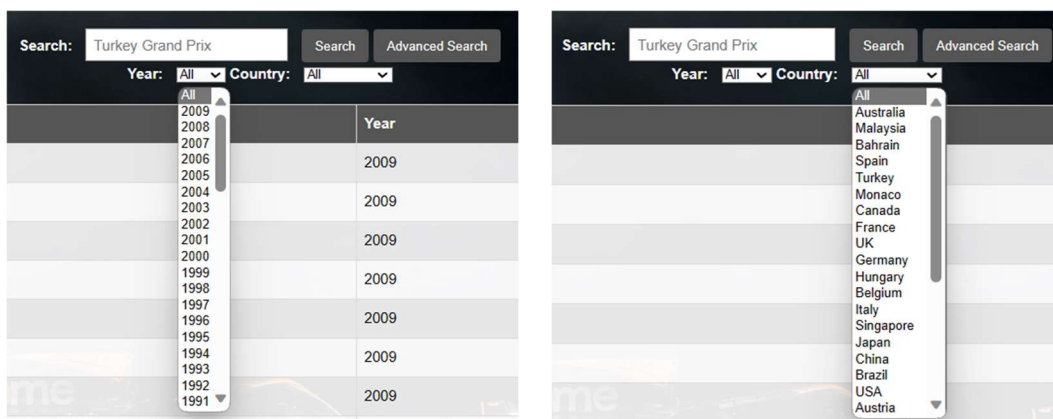
Users can perform a basic search by entering a race name. The system employs the LIKE operator for partial matching, enabling users to find races that contain the provided search term.

5.1.1.2. Advanced Search




Year and Country Filter section is available when advanced search button is clicked

Clicking the advanced search button reveals filters for year and country. Users can narrow down results based on specific years and countries of interest.



Dropdown list of year and country filtering

5.1.2. Race Details

Name	Year	Country				
Australian Grand Prix	2009	Australia				
Race Details		Race Results				
Date	2009-03-29	Position	Name Surname	Code	Fastest Lap	Constructor
Round	1	1	Jenson Button	BUT	34:15.8	Brawn
Place	Melbourne	2	Rubens Barrichello	BAR	0.807	Brawn
		3	-	-	-	-
Circuit Details		Map				
Circuit Name	Albert Park Grand Prix Circuit					
Country	Australia					
Malaysian Grand Prix	2009	Malaysia				
Chinese Grand Prix	2009	China				
Bahrain Grand Prix	2009	Bahrain				
Spanish Grand Prix	2009	Spain				
Monaco Grand Prix	2009	Monaco				
Turkish Grand Prix	2009	Turkey				

Race and Circuit details section of race is available when clicked the name of race

Clicking on the name of a race reveals detailed information about that particular race, including the year, date, round, circuit details (name, country, location), and a link to the circuit's website.

5.1.3. Pagination



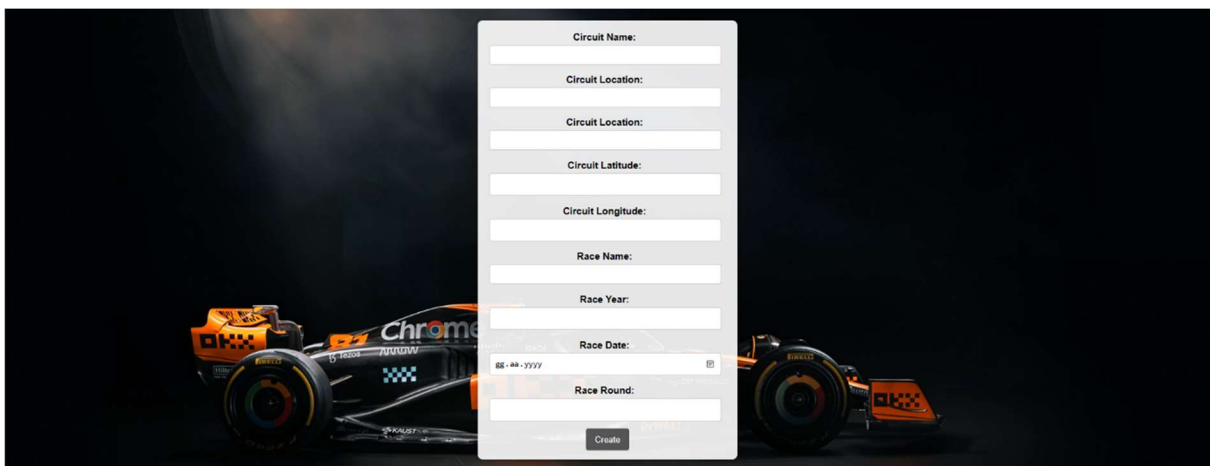
Pagination part for easy access

To enhance user experience, the Races Page incorporates pagination. Users can navigate through multiple pages of search results using intuitive pagination controls.

5.1.4. Admin Functions

When an admin is logged in, additional functionalities become available:

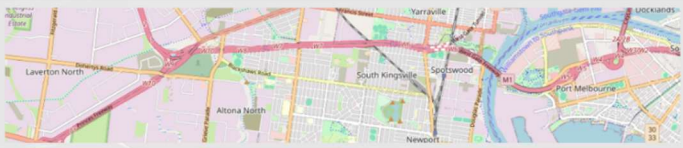
5.1.4.1. Create



Create section is available below the table when admin is logged

Admins can add new races by filling out a form that includes details about the circuit and race. The system dynamically generates new circuit and race IDs and inserts the information into the respective tables.

5.1.4.2. Edit and Delete

Name	Year	Country	Delete	Edit																				
Australian Grand Prix	2009	Australia	<input type="checkbox"/>	Edit																				
Race Details		Race Results																						
Date	2009-03-29	<table border="1"> <thead> <tr> <th>Position</th> <th>Name Surname</th> <th>Code</th> <th>Fastest Lap</th> <th>Constructor</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Jenson Button</td> <td>BUT</td> <td>34:15.8</td> <td>Brawn</td> </tr> <tr> <td>2</td> <td>Rubens Barrichello</td> <td>BAR</td> <td>0.807</td> <td>Brawn</td> </tr> <tr> <td>3</td> <td>-</td> <td>-</td> <td>-</td> <td>-</td> </tr> </tbody> </table>			Position	Name Surname	Code	Fastest Lap	Constructor	1	Jenson Button	BUT	34:15.8	Brawn	2	Rubens Barrichello	BAR	0.807	Brawn	3	-	-	-	-
Position	Name Surname	Code	Fastest Lap	Constructor																				
1	Jenson Button	BUT	34:15.8	Brawn																				
2	Rubens Barrichello	BAR	0.807	Brawn																				
3	-	-	-	-																				
Round	1																							
Place	Melbourne																							
Circuit Details		Map																						
Circuit Name	Albert Park Grand Prix Circuit																							
Country	Australia																							
Malaysian Grand Prix	2009	Malaysia	<input type="checkbox"/>	Edit																				
Chinese Grand Prix	2009	China	<input type="checkbox"/>	Edit																				
Bahrain Grand Prix	2009	Bahrain	<input type="checkbox"/>	Edit																				
Spanish Grand Prix	2009	Spain	<input type="checkbox"/>	Edit																				

Edit and Delete button is available when admin is logged

Admins can modify race details, including the race name, year, date, and round. Additionally, circuit information such as name, location, and country can be updated.

Also, Admins have the ability to delete specific race entries. The system constructs dynamic SQL queries to delete race entries based on the selected race IDs.

5.2. Code Outlines and Queries for the Races Page

5.2.1. Main Search Function

5.2.1.1. get_search_results Function

This function serves as the core mechanism for retrieving race information based on user-provided search terms, year filters, and country filters. It handles pagination and communicates with the database to fetch relevant data.

5.2.1.1.1. Base Queries:

- Base Query with Search Term:

Retrieves information about races and circuits based on a provided search term. Utilizes the LIKE operator for partial matching with a wildcard appended.

- Query: *SELECT ... FROM races r JOIN circuits c ON r.circuitId = c.circuitId WHERE r.name LIKE '{search_term}%'*

- Base Query without Search Term:

Retrieves all races with non-empty names when no search term is provided.

- Query: *SELECT ... FROM races r JOIN circuits c ON r.circuitId = c.circuitId WHERE LENGTH(r.name) > 0*

- Filtering Queries:

Additional queries added to the main query when users want to filter results based on year or country.

- Queries:
 - + "AND r.year = {year_filter}" (when year filter is selected)
 - + "AND c.country = {country_filter}" (when country filter is selected)

- Finalization Queries:

Queries added to the main query to ensure proper ordering, pagination, and presentation of results.

- Queries:
 - + "ORDER BY r.raceId"
 - + "LIMIT {RESULTS_PER_PAGE} OFFSET {offset}"

5.2.1.1.2. Race Results Queries

- Race Results Query with Search Term:

Fetches detailed race results, including driver information, for races matching the provided search term. Filters results to include only positions 1, 2, and 3.

- Query: *SELECT ... FROM races r JOIN circuits c ON r.circuitId = c.circuitId JOIN results rs ON r.raceId = rs.raceId JOIN drivers d ON rs.driverId = d.driverId JOIN constructors cn ON rs.constructorId = cn.constructorId WHERE r.name LIKE '{search_term}%' AND rs.position IN (1, 2, 3)*

- Race Results Query without Search Term:

Similar to the previous race results query but without a specific search term. Fetches race results for all races and filters for positions 1, 2, and 3.

- Query: *SELECT ... FROM races r JOIN circuits c ON r.circuitId = c.circuitId JOIN results rs ON r.raceId = rs.raceId JOIN drivers d ON rs.driverId = d.driverId JOIN constructors cn ON rs.constructorId = cn.constructorId WHERE rs.position IN (1, 2, 3)*

- Filtering Queries:

Additional queries added to the main query when users want to filter results based on year or country.

- Queries:

- + "AND r.year = {year_filter}" (when year filter is selected)
- + "AND c.country = {country_filter}" (when country filter is selected)
- Finalization Queries:

Queries added to the main query to ensure proper ordering, pagination, and presentation of results.

- Queries:
 - + "ORDER BY r.raceId , rs.position"
 - + "LIMIT {RESULTS_PER_PAGE} OFFSET {offset}"

5.2.1.1.3. Total Count Queries

- Total Count Query with Search Term:

Obtains the total count of races matching the provided search term for pagination purposes.

- Query: *SELECT COUNT(*) FROM races r JOIN circuits c ON r.circuitId = c.circuitId WHERE r.name LIKE '{search_term}%'*
- Total Count Query without Search Term:

Obtains the total count of all races for pagination when no search term is provided.

- Query: *SELECT COUNT(*) FROM races r JOIN circuits c ON r.circuitId = c.circuitId WHERE LENGTH(r.name) > 0*

5.2.1.1.4. Distinct Years and Countries Queries

Queries that retrieve distinct years and countries from the 'races' and 'circuits' tables, respectively. Utilized to provide filter options in the user interface.

- Queries:
 - *SELECT DISTINCT year FROM races*
 - *SELECT DISTINCT country FROM circuits*

5.3. Routes Definition

5.3.1. Main Races Page

The /races route is the main entry point for users, displaying a page with search and filter options. It renders the 'races.html' template, presenting the search results and filter options.

5.3.2. Race Creation

The /races/create route handles POST requests for creating new race entries. It incorporates a form for users to input circuit and race details.

5.3.2.1. Inserting Circuit Information

The first part of the code inserts information about a new circuit into the 'circuits' table. It does the following:

- Retrieves the last circuit ID from the 'circuits' table.
- Increments the last circuit ID to create a new circuit ID for the upcoming record.
- Executes an SQL query to insert the new circuit information into the 'circuits' table.

SQL Query for Inserting Circuit:

```
INSERT INTO circuits (circuitId, name, location, lat, lng, alt, url, country, circuitRef) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

- `circuitId`: The newly generated circuit ID.
- `name, location, lat, lng, alt, url, country, circuitRef`: Values for various attributes of the circuit.

5.3.2.2. Inserting Race Information

The second part of the code inserts information about a new race into the 'races' table. It does the following:

- Retrieves the last circuit ID (calculated previously) and uses it for the new race entry.
- Executes an SQL query to insert the new race information into the 'races' table.

SQL Query for Inserting Race:

```
INSERT INTO races (name, year, date, round, time, url, circuitId) VALUES (?, ?, ?, ?, ?, ?, ?)
```

- `name, year, date, round, time, url`: Values for various attributes of the race.
- `circuitId`: The circuit ID obtained from the last circuit entry.

These SQL queries collectively insert a new circuit and race into their respective tables, establishing a relationship between them through the 'circuitId' foreign key in the 'races' table. This code segment is crucial for adding new race and circuit entries to the database.

5.3.3. Race Deletion

The `/races/delete` route handles race deletion, supporting the DELETE method. It requires user authentication and communicates with the database to delete selected race entries.

5.3.3.1. Deleting Race Entries

The code constructs a dynamic SQL query to delete specific race entries from the 'races' table. It does the following:

- Iterates through the list of race IDs to construct the IN clause for the DELETE query.
- Appends each race ID to the IN clause with proper formatting.
- Constructs the final DELETE query to delete race entries with the specified race IDs.

SQL Query for Deleting Race Entries:

```
DELETE FROM races WHERE raceId IN (race_id_1, race_id_2, ..., race_id_n)
```

- `raceId`: The primary key of the 'races' table.
- `race_id_1, race_id_2, ..., race_id_n`: Specific race IDs to be deleted.

This SQL query will effectively delete the race entries in the 'races' table where the 'raceId' matches any of the specified race IDs in the list. The dynamic construction of the query allows for flexibility in handling multiple race deletions in a single operation.

5.3.4. Race Update

The `/races/update` route is designed for updating race information. It requires authentication, supports the POST method, and communicates with the database to update race and circuit details.

5.3.4.1. Updating Race Information

The code executes an UPDATE query to modify information in the 'races' table for a specific race ID (raceId). It does the following:

SQL Query for Updating Race Information:

```
UPDATE races SET name = ?, year = ?, date = ?, round = ? WHERE raceId = ?
```

- name, year, date, round: New values for the attributes of the race.
- raceId: The primary key used to identify the specific race entry to be updated.

5.3.4.2. Updating Circuit Information

Similarly, the code executes an UPDATE query to modify information in the 'circuits' table for a specific circuit ID (circuitId). It does the following:

SQL Query for Updating Circuit Information:

```
UPDATE circuits SET name = ?, location = ?, country = ? WHERE circuitId = ?
```

- name, location, country: New values for the attributes of the circuit.
- circuitId: The primary key used to identify the specific circuit entry to be updated.

These SQL queries allow the application to dynamically update race and circuit information in their respective tables. The use of placeholders (?) in the queries indicates the use of parameterized queries, which helps prevent SQL injection by binding values to the queries in a secure manner.

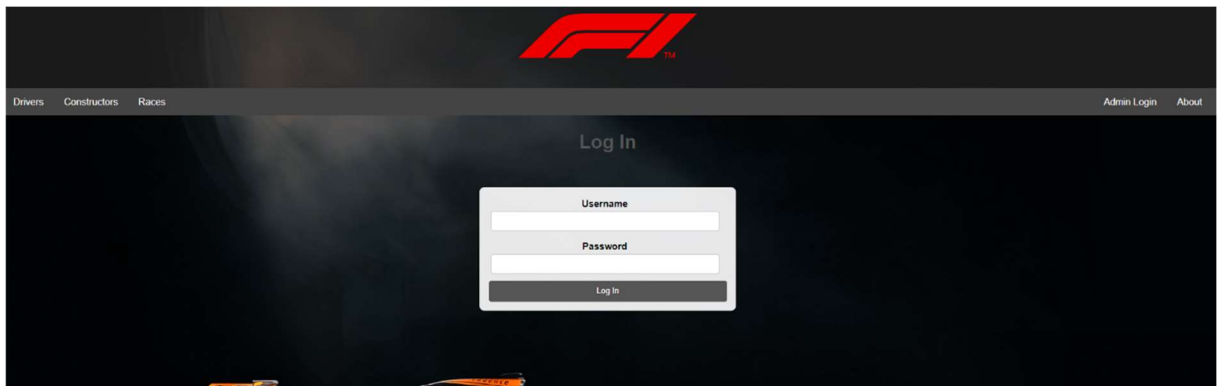
6. Log In Admin Module

To regulate the update, delete, create operations on the database, We developed an authorization module which allows admins to log in and log out to the application. To achieve this, we created a new table named as admin with the following command.

```
CREATE TABLE admin (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    username TEXT UNIQUE NOT NULL,  
    password TEXT NOT NULL  
);
```

We do not implement register page since the app includes only admin as a user and there is no need to add new users on the web application.

As stated before, our segments for create, update, delete will be rendered only if the admin is logged in.

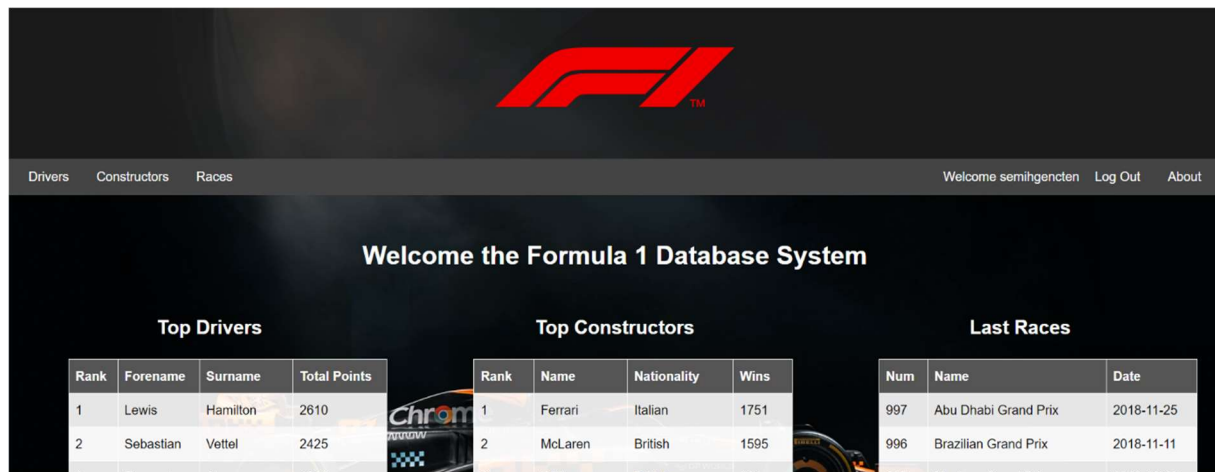


In the log in process, the program uses the following select query to find user and check its password.

```
user = db.execute(  
    'SELECT * FROM admin WHERE username = ?', (username,) )  
).fetchone()
```

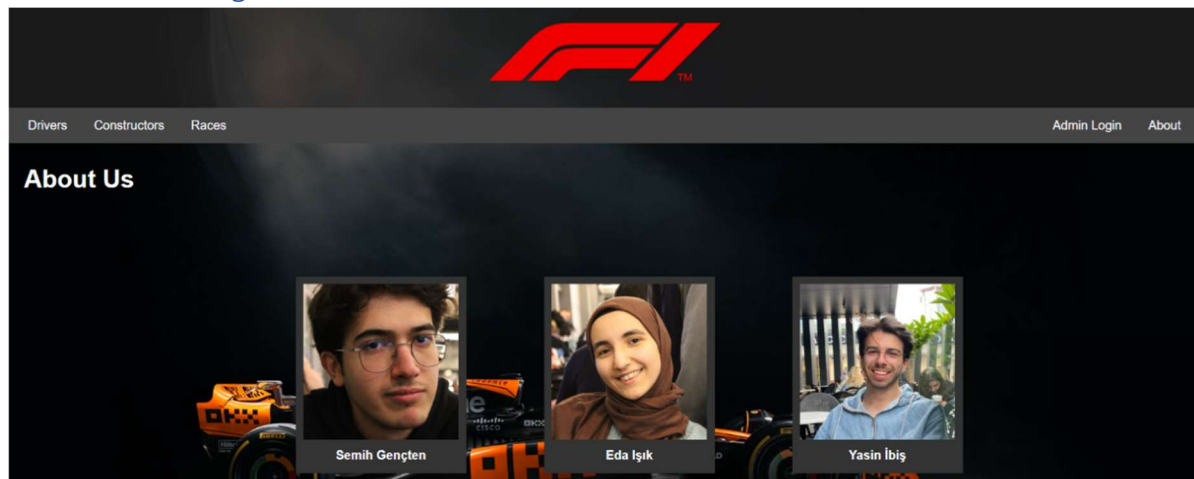
The following query is used to control whether the user is logged in or not.

```
g.user = get_db().execute(  
    'SELECT * FROM admin WHERE id = ?', (user_id,) )  
).fetchone()
```



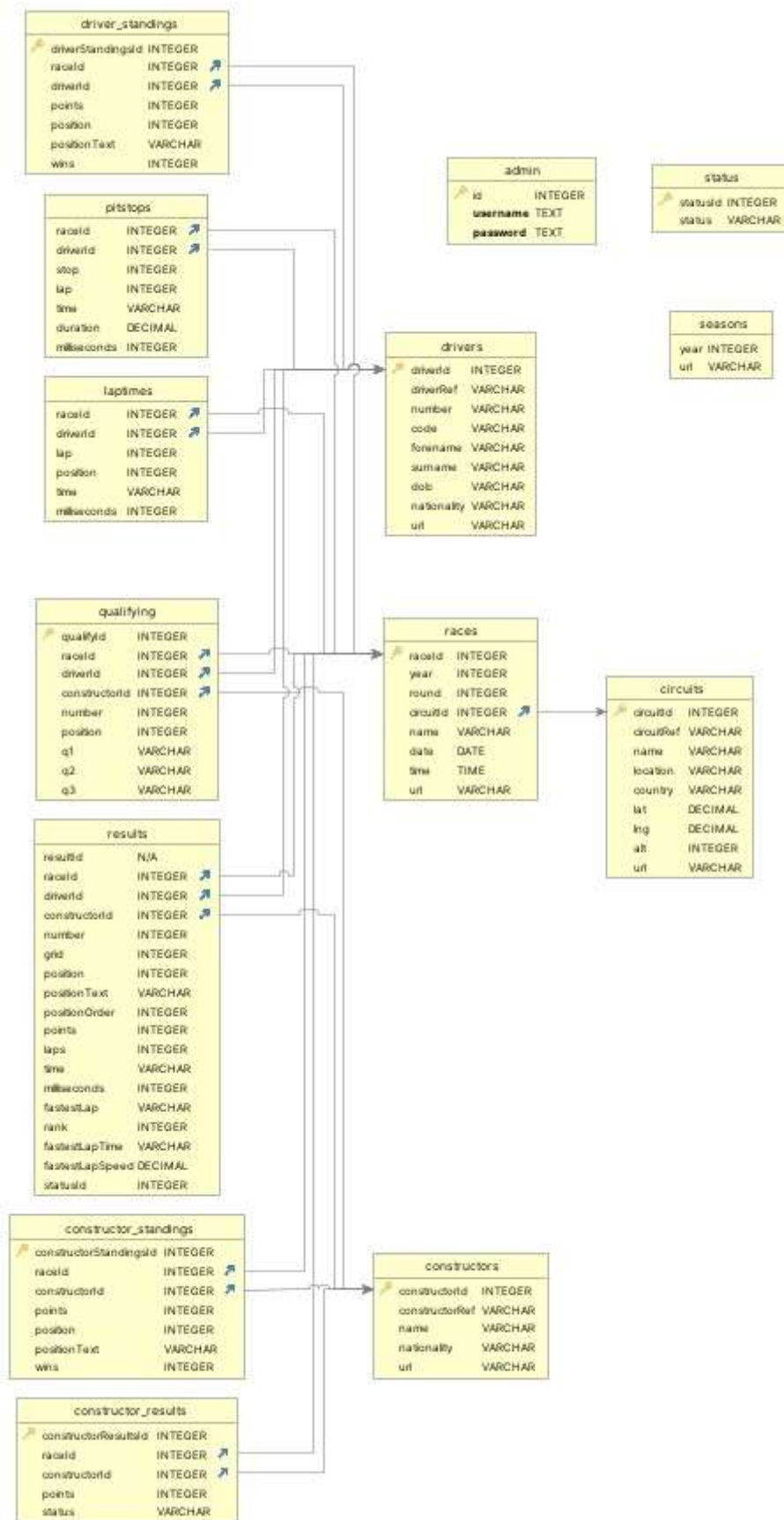
When an admin logged in, Welcome 'username' is visible.

7. About Page



The about page shows project group members.

8. ERD Diagram



ERD Diagram for Formula 1 Database

9. About Members and Responsibilities

The members of the project group are:

- 150200062 Yasin İbiş
 - 150200064 Semih Gençten
 - 150200089 Eda Işık
-
- Yasin was responsible for creating the page related to races and the overall appearance of the application.
 - Semih was in charge of creating the page related to drivers, handling the login functionality, and the project's index page.
 - Eda was responsible for the page related to constructors, as well as the 'About' page, and compiling/editing the report.