Table of Contents:

Introduction

Members

- Atharva Gupta (axg1395)
- Sean Liu (shl91)
- Dylan Zhao (dxz365)
- Project Team 11

Demo Date

April 22, 2025 - Atharva Presented, Sean Display

Application

F1 Database

Abstract

F1 is a dynamic sport with drivers changing every race and an ever growing fanbase, which needs a centralised dataset. We seek to do that by constructing a dataset that contains all relevant data from past seasons that still fall under the current regulations (2022, 2023, 2024 seasons). This F1 Database allows users to access various data regarding drivers, races, results, championship, etc, allowing for a multitude of applications such as data analytics and management for teams as well as to enhance fan experiences through engagement with the data of the sport. This database seeks to provide these users with both historical data as well as live data allowing them to see trends from past seasons as well as the current one.

Data Source: https://openf1.org/ (an API), https://docs.fastf1.dev/ergast.html (has more telemetry data)

Reproduction Instructions

DB Backup Name

F1Database.bak

Zip File Name

F1Database.zip

Demo Schedule Date

April 22, 2025 (3:00 pm - 3:30 pm)
TA: Tyler Powers
Virtual Environment Owner: Sean Liu
Demonstration Leader: Atharva Gupta

How to Access Virtual Environment

I.
II.
III.

<u>Database Design and Business Rules</u>

<u>Relationship Schema and Description</u>

Driver(dID, tID, fname, lname, dob, nationality, carNum)
The ‘Driver’ relation holds information on the driver. The attributes in ‘Driver’ mean the following: ‘dID’ is a unique identifier for a driver, ‘tID’ corresponds to the Team.tID, ‘fname’ is the first name of the diver, ‘lname’ is the last name of the driver, ‘dob’ is the date of birth, ‘nationality’ is the nationality, ‘carNum’ is the number the driver has on the car. None of the values should be null.

Team(tID, name, fullname, country, engine)
The ‘Team’ relation holds information on the team. The attributes in ‘Team’ mean the following: ‘tID’ is a unique identifier for a team, ‘name’ is the short name of a team, ‘fullname’ is the full name of a team, ‘country’ is where a team is based, ‘engine’ is the engine provider of a team. None of the values should be null.

Race(rID, cID, season, round, date)
The ‘Race’ relation holds information on the race. The attributes in ‘Race’ mean the following: ‘rID’ is a unique identifier for a race, ‘cID’ corresponds to the Circuit.cID, ‘season’ is the race season for a race, ‘round’ is which race in a season the race is, ‘date’ is the date of the race. None of the values should be null.

DriverRace(dID, rID)
‘dID’ refers to the primary key for the driver in the Driver table participating in a particular race using ‘rID’, the primary key for a Race from the Race table. This table facilitates driver-race lookup.

Circuit(cID, name, country, city, length)
The ‘Circuit’ relation holds information on the circuit. The attributes in ‘Circuit’ mean the following: ‘cID’ is a unique identifier for a race, ‘name’ is the name of the circuit, ‘country’ is the country where the circuit is located, ‘city’ is the city where the circuit is located, ‘length’ is the total length of one lap around the circuit. None of the values should be null.

Result(resultID, rID, dID, position, points, status, time)
The ‘Result’ relation holds information on the result of a driver at a specific race. The attributes of ‘Result’ mean the following: ‘resultID’ is a unique identifier for a result, ‘rID’ and ‘dID’ reference Race.rID and Driver.dID respectively, ‘position’ is the position of the driver in the race, ‘points’ is the amount of points the driver receives, ‘status’ is how the driver ended the race (‘finished’, ‘DNF’, ‘DNS’, ‘DSQ’), ‘time’ is the total time it took the driver to finish the race. None of the values should be null outside of time.

cChampionship(cChampID, tID, season, points, position)
The ‘cChampionship’ relation holds information on a team in the constructors championship. The attributes of ‘cChampionship’ mean the following: ‘cChampID’ is a unique identifier of the constructors championship, ‘tID’ corresponds to the Team.tID, ‘season’ is the year of the constructors championship, ‘points’ is the number of points scored by the team in the constructors championship, ‘position’ is the position of the team in the constructors championship. None of the values should be null.

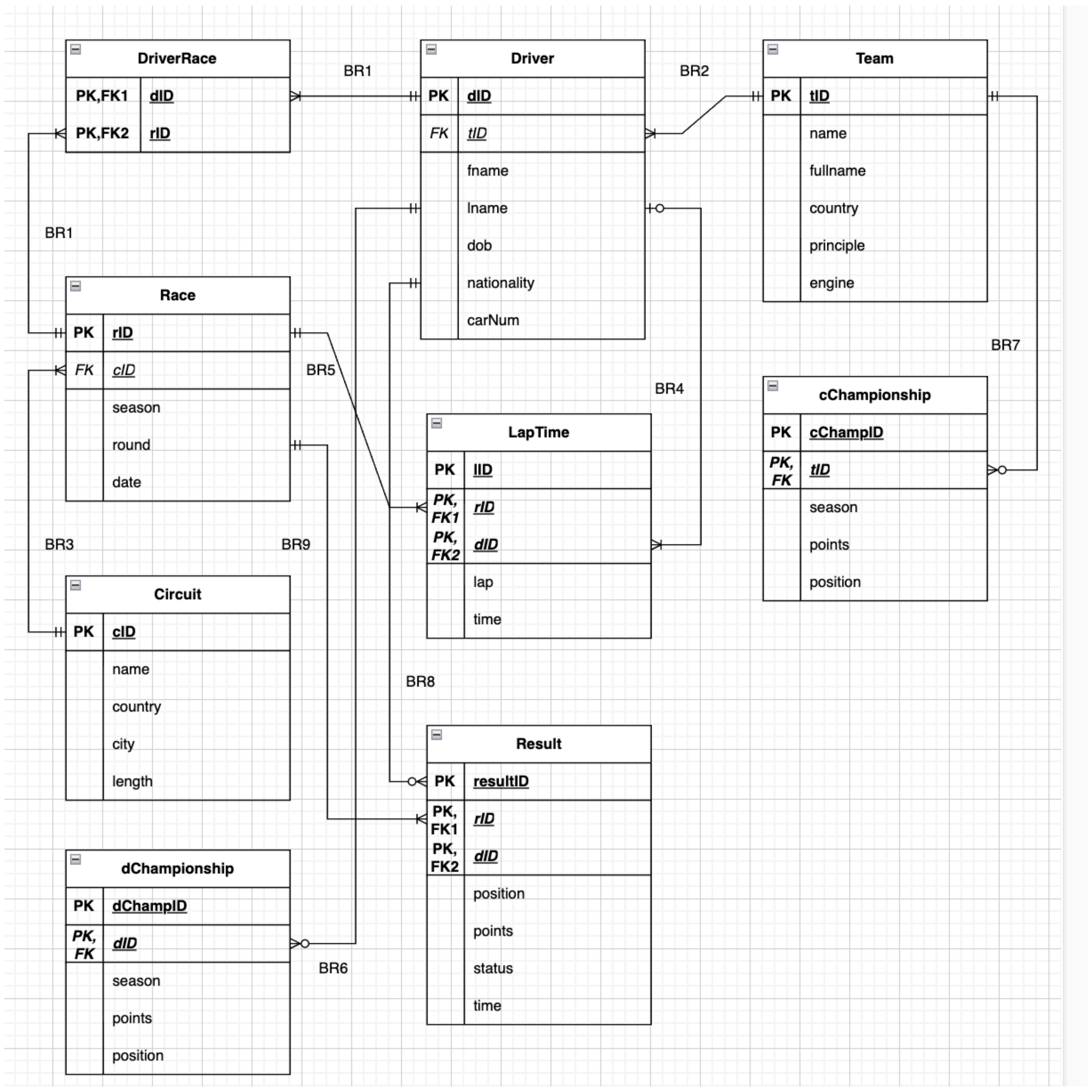
dChampionship(dChampID, dID, season, points, position)
The ‘dChampionship’ relation holds information on a driver in the drivers championship. The attributes of ‘dChampionship’ mean the following: ‘dChampID’ is a unique identifier of the drivers championship, ‘tID’ corresponds to the Team.tID, ‘season’ is the year of the drivers championship, ‘points’ is the number of points scored by the driver in the drivers championship, ‘position’ is the position of the driver in the drivers championship. None of the values should be null.

LapTime(lID, rID, dID, lap, time)
The ‘LapTime’ relation holds information on the lap time of a driver in a specific race. The attributes of ‘LapTime’ mean the following: ‘lID’ is a unique identifier of the lap time, ‘rID’ and ‘dID’ reference Race.rID and Driver.dID respectively, ‘lap’ is the lap number of the lap time, ‘time’ is how long the lap took.

Business Rules

1. A Driver has multiple Races, a Race can have multiple Drivers (M: M)
2. Teams can have multiple drivers over a season (1: M), each driver can only have one team at a time (M: 1). Additionally, a Team cannot have less than two drivers.
3. Each Circuit can have many Races (1: M), A Race can only have one circuit (M: 1)
4. A Driver can have (optional) one or multiple LapTime(s) (1: M), but a LapTime can have exactly one Driver
5. A Race can have multiple LapTimes (1: M), A LapTime has exactly one Race (M: 1)
6. A Driver can have (optional) one or multiple dChampion(s) (1: M) and dChampion can only have one Driver (M: 1) and cannot be null
7. A Team can have one or many cChampion(s) (1: M), a cChampion can exactly have one Team (M: 1)
8. A Result has exactly one Driver, a Driver has (optional) one or multiple Result(s) (1: M)
9. A Result can have exactly one Race , a Race can have multiple Result(s) (1: M)

ERD


DriverRace
PK,FK1 dID
PK,FK2 rID
Driver
PK dID
FK tID
fname
lname
dob
nationality
carNum
Team
PK tID
name
fullname
country
principle
engine
Race
PK rID
FK cID
season
round
date
Circuit
PK cID
name
country
city
length
dChampionship
PK dChampID
PK, FK dID
season
points
position
LapTime
PK lID
PK, FK1 rID
PK, FK2 dID
lap
time
Result
PK resultID
PK, FK1 rID
PK, FK2 dID
position
points
status
time
cChampionship
PK cChampID
PK, FK tID
season
points
position
BR1
BR1
BR2
BR3
BR4
BR5
BR6
BR7
BR8
BR9

Use Cases

Use Case 1: Register New Driver and Register Them To a Race

- **Team Member: Atharva Gupta**
- **Description:**
    - This use case allows the system to add a new driver to the database. All driver details (unique driver ID, team ID, first and last names, date of birth, nationality, car number) are collected. It ensures that none of the values are null and that the driver is associated with an existing team and that the car number is not repeated. Additionally, it adds the driver to a new race.
- **Details:**
    - **Tables Involved:** Driver, Team, DriverRace
    - **Operations:** Insert, Select
    - **Business Rules:**
        - A Driver can only be part of one team per season, but a Team can have multiple drivers.
        - A Driver can race multiple times, and a Race can have multiple drivers.
- The end user must supply all driver information through a form. Validation will ensure all fields are complete. The user inserts the new driver information and selects the correct tID. Additionally, the driver ID is selected to insert them into DriverRace to register them for a race.

Sql code:

```
Unset
CREATE PROCEDURE insertDriver
  @tname        VARCHAR(255),
  @fname        VARCHAR(255),
  @lname        VARCHAR(255),
  @dob          DATE,
  @nationality  VARCHAR(255),
  @carNum       INT,
  @dID          INT OUTPUT
AS
BEGIN
  IF EXISTS (
    SELECT 1
      FROM driver
     WHERE carNum = @carNum
  )
  BEGIN
```

```
    RAISERROR('Cannnot insert: driver number already exists', 16, 1);
    RETURN;
  END

  INSERT INTO dbo.Driver
    (tID, fname, lname, dob, nationality, carNum)
  VALUES
    (
      (SELECT tID
         FROM dbo.Team
        WHERE name = @tname
      ),
      @fname,
      @lname,
      @dob,
      @nationality,
      @carNum
    );

  SELECT @dID = SCOPE_IDENTITY();
END;
```

Java code:

```java
System.out.println("Team short-name       :");
            String tname = sc.nextLine();

            System.out.println("Driver first name     :");
            String fname = sc.nextLine();

            System.out.println("Driver last  name     :");
            String lname = sc.nextLine();

            System.out.println("Date of birth (YYYY-MM-DD):");
            java.sql.Date dob = java.sql.Date.valueOf(sc.nextLine());

            System.out.println("Nationality           :");
            String nationality = sc.nextLine();

            System.out.println("Car number (int)      :");
            int carNum = Integer.parseInt(sc.nextLine());

            String call = "{call dbo.insertDriver(?,?,?,?,?,?,?)}";
            try (Connection cn = DriverManager.getConnection(connectionUrl);
                    CallableStatement cs = cn.prepareCall(call)) {

                cn.setAutoCommit(false);

                cs.setString(1, tname);
                cs.setString(2, fname);
                cs.setString(3, lname);
                cs.setDate(4, dob);
                cs.setString(5, nationality);
                cs.setInt(6, carNum);

                cs.registerOutParameter(7, java.sql.Types.INTEGER); // @dID
                cs.execute();

                System.out.println("→ New Driver ID = " + cs.getInt(7));

                System.out.print("Commit or Rollback? ");
                if (sc.nextLine().equalsIgnoreCase("Commit"))
                    cn.commit();
                else
                    cn.rollback();

            } catch (SQLException ex) {
                // if RAISERROR fired in the proc, the message and error-code
appear here
                ex.printStackTrace();
            }
```

Use Case 2: Update Driver Information

- **Team Member: Not Implemented**
- **Description:**
    - This use case allows updating a driver's information for an existing driver record. This would if any information regarding the driver would need to be changed due to a previous error or if the driver for example changes teams. As a result, this use case allows for this to occur.
- **Details:**
    - This use case utilizes two tables: Driver and Team. Two DML statements SELECT and UPDATE are needed for this use case. The old information

regarding the driver is replaced and updated with the new information. If a team is updated, then the Team ID (tID) will be selected from the Team table based on the name of the team. The entire tuple will then be returned for verification by the user in order to commit or rollback the transaction.
    - **Business Rules:**
        - A driver must belong to one team at a time.
- **User Requirements:** The user provides the Driver's first name, last name, and date of birth (YYYY-MM-DD) as well as information they wish to update. None of this information will be null.

Use Case 3: Register New Team

- **Team Member: Dylan Zhao**
- **Description:**
    - This use case adds a new F1 team to the database. This would be needed in case a new team joins F1, and their information would need to be added to the database. Additionally, since each team is required to have two drivers, this use case also adds in two new drivers as part of the new team. As a result, this use case is specifically for sourcing the new drivers from outside the database, and not from any other teams.
- **Details:**
    - This use case utilizes two tables: Team and Driver. Additionally, two DML statements SELECT and INSERT are used to perform this use case. The new team's information is inserted into the Team table, and two new drivers are also inserted into the Driver table with team ID (tID) being selected based off the new Team. To verify the operation went through, the new tID and the two new drivers' IDs (dID) are returned to the user to which the user can verify the results. If the users approve of the results, they can commit the transaction or choose to rollback if they notice an issue.
    - **Business Rule:**
        - Teams can have multiple drivers over a season but they cannot have less than two drivers
- **User Requirements:**
    - Users will be required to enter the new team's short name, full name, country, and engine as well as each new driver's first name, last name, date of birth (YYYY-MM-DD format), nationality, and car number. None of these values are allowed to be blank.

Use Case 4: Record a New Race Event

- **Team Member: Not Implemented**

- **Description:**
    - This use case is used to add a new race event to the database. This would be used for scheduling a race before the start of the season. Details regarding the race, including circuit ID (cID) will be added into the Race table.
- **Details:**
    - This use case utilizes the Race and Circuit tables using two DML statements: SELECT and INSERT. Data regarding the new race will be inserted into the Race table, and the cID of the circuit the race will be a part of will be selected from the Circuit table and added as well. The number of rows of in the added to the Race table will be returned to the user to verify the insertion and be given a choice to commit or rollback the transaction.
    - **Business Rules:**
        - A race can have only one circuit (M:1).
- The user is required to provide information regarding the season, round, and as well as circuit name. None of these values may be null.

Use Case 5: Enter Race Results

- **Team Member:**
- **Description:**
    - This use case allows the entry of race results. For a given race, it records each driver's performance including their finishing position, points earned, status, and time.
- **Details:**
    - **Tables Involved:** Result, Driver, dChampionship, cChampionship
    - **Operations:** Insert, Select, Update
    - **Business Rules:**
        - Each result must have exactly one driver and one race.
- The user must select for resultID, race ID, driver ID, position, points, status, and time via the interface, and insert the information into the Result table. The points for each corresponding driver and their team is added to the dChampionship and cChampionship respectively.

Sql code:

```
Unset
USE F1Database;
GO

IF OBJECT_ID('dbo.enterRaceResults','P') IS NOT NULL
    DROP PROCEDURE dbo.enterRaceResults;
```

```
GO

CREATE PROCEDURE dbo.enterRaceResults
  @cname     VARCHAR(255),
  @fname     VARCHAR(255),
  @lname     VARCHAR(255),
  @carNum    INT,
  @position  INT,
  @points    INT,
  @status    VARCHAR(8),
  @time      INT,
  @date      DATE,
  @resultID  INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @dID INT, @rID INT;

  -- 1) Find the driver
  SELECT @dID = dID
  FROM dbo.Driver
  WHERE fname  = @fname
    AND lname  = @lname
    AND carNum = @carNum;

  IF @dID IS NULL
    BEGIN
    RAISERROR('Driver not found: %s %s (#%d)', 16, 1,
                  @fname, @lname, @carNum);
    RETURN;
  END

  -- 2) Find the race
  SELECT @rID = rID
  FROM dbo.Race
  WHERE cID   = (
               SELECT cID
    FROM dbo.Circuit
    WHERE name = @cname
             )
    AND [date] = @date;
  -- bracketed because DATE is a keyword
```

```
IF @rID IS NULL
  BEGIN
  DECLARE @errMsg NVARCHAR(255) =
          N'Race on ' + CONVERT(CHAR(10), @date, 23)
        + N' at circuit ' + @cname + N' not found.';
  RAISERROR(@errMsg, 16, 1);
  RETURN;
END

-- 3) Prevent duplicate result entries
IF EXISTS (
      SELECT 1
FROM dbo.Result
WHERE rID = @rID
  AND dID = @dID
  )
  BEGIN
  RAISERROR('Result already exists for this driver/race.', 16, 1);
  RETURN;
END

-- 4) Insert the new result
INSERT INTO dbo.Result
  (rID, dID, position, points, status, [time])
VALUES
  (@rID, @dID, @position, @points, @status, @time);

-- 5) Return the new identity
SET @resultID = SCOPE_IDENTITY();

--6 update the dChampionship table
UPDATE dbo.dChampionship
  SET points = points + @points
WHERE dID = @dID;
RETURN;

--7 update the cChampionship table
UPDATE dbo.cChampionship
  SET points = points + @points
WHERE tID = (
  SELECT tID
FROM dbo.Driver
WHERE dID = @dID
);
```

```
  RETURN;

END;
```

Java code:

```Java
Java

           System.out.println("Circuit name  :");
            String cname = sc.nextLine();

            System.out.println("Driver’s first name :");
            String fname = sc.nextLine();

            System.out.println("Driver’s last  name :");
            String lname = sc.nextLine();

            System.out.println("Car number (int)    :");
            int carNum = Integer.parseInt(sc.nextLine());

            System.out.println("Finishing position  :");
            int position = Integer.parseInt(sc.nextLine());

            System.out.println("Points awarded      :");
            int points = Integer.parseInt(sc.nextLine());

            System.out.println("Status (e.g. \"Finished\", \"DNF\"):");
            String status = sc.nextLine();

            System.out.println("Finish time (hh:mm:ss[.fff]) :");
            java.sql.Time time = java.sql.Time.valueOf(sc.nextLine());

            System.out.println("Race date (YYYY-MM-DD)      :");
            java.sql.Date date = java.sql.Date.valueOf(sc.nextLine());

            System.out.printf(
                    "%nCONFIRM →  circuit=%s, driver=%s %s, car=%d,
position=%d, points=%d, status=%s, time=%s, date=%s%n",
                    cname, fname, lname, carNum, position, points, status,
time, date);

            String call = "{call dbo.enterRaceResults(?,?,?,?,?,?,?,?,?,?)}";

            try (Connection cn = DriverManager.getConnection(connectionUrl);
                    CallableStatement cs = cn.prepareCall(call)) {

                cn.setAutoCommit(false);

                cs.setString(1, cname);
                cs.setString(2, fname);
                cs.setString(3, lname);
                cs.setInt(4, carNum);
                cs.setInt(5, position);
                cs.setInt(6, points);
                cs.setString(7, status);
                cs.setTime(8, time);
                cs.setDate(9, date);

                cs.registerOutParameter(10, java.sql.Types.INTEGER); //
@resultID
                cs.execute();

                int resultID = cs.getInt(10);
                System.out.println("→ New Result ID = " + resultID);

                System.out.print("Commit or Rollback? ");
                if (sc.nextLine().equalsIgnoreCase("Commit"))
                    cn.commit();
                else
                    cn.rollback();

            } catch (SQLException ex) {
                ex.printStackTrace();
            }
```

Use Case 6: Record Lap Times for a Race

- **Team Member: Not Implemented**
- **Description:**
    - This use case allows recording individual lap times for a driver in a race. It supports the collection of multiple lap time records per driver per race. This would be used during a race as each driver completes a lap and information regarding that lap would be recorded.
- **Details:**

- The use case involves the LapTime, Race, and Driver tables and utilizes INSERT and SELECT DML statements. Information regarding the driver will be used to select for the driver ID (dID), and information regarding the race will be used for the race ID (rID). This information in addition to the lap time information will be inserted into the LapTime table. The number of rows that is added will be returned to the user who will verify and determine to commit or rollback the transaction.
    - **Business Rules:**
        - A lap time record must be associated with exactly one race and one driver.
- The user is required to input the information regarding the driver's first name, driver’s last name, driver’s date of birth, season of the race, round of the race, lap number, and time of the lap. None of these values should be null.

Use Case 7: Generate Driver Championship Standings

- **Team Member: Not Implemented**
- **Description:**
    - This use case retrieves and displays the driver championship standings for a specific season by aggregating data from the dChampionship table along with driver details.
- **Details:**
    - **Tables Involved:** dChampionship, Driver
    - **Operations:** Select, Update
    - **Business Rules:**
        - Each dChampionship record must be associated with exactly one driver and cannot be null.
- The user selects a season from the interface, and the system returns the standings sorted by points and position. The user also updates any incorrect information in the dChampionship table.

Use Case 8: Generate Team Championship Standings

- **Team Member: Not Implemented**
- **Description:**
    - This use case displays the team championship standings by retrieving data from the cChampionship table joined with the Team table for a given season.
- **Details:**
    - **Tables Involved:** cChampionship, Team
    - **Operations:** Select, Update
    - **Business Rules:**
        - Each cChampionship record must correspond to exactly one team.

- The user chooses a season, and the system presents the team standings ordered by position and points. The user also updates any incorrect information in the cChampionship table.

Use Case 9: Adjusting Race Results After the Race Ended (Disqualification)

- **Team Member: Sean Liu**
- **Description:**
    - This use case allows users to adjust the race results in case of disqualifications that can affect the results.
- **Details:**
    - **Tables Involved:** Result, Driver, dChampionship, cChampionship
    - **Operations:** Select, Update
    - **Business Rules:**
        - A Driver can have multiple Results, but each Result has exactly one Driver.
        - A Driver can have (optional) one or multiple dChampion(s) (1: M) and dChampion can only have one Driver (M: 1) and cannot be null.
- The user is able to update the race results in cases such as a disqualification where the race results would need to be changed afterwards. This is done by nullifying the time of the Result time for the specific driver. Additionally, this new result must be adjusted within the dChampionship table which requires selecting for the driver ID and within the cChampionship table which requires selecting for the tID.
- SQL Code:

```
create or alter procedure AdjustResultDSQ
@dID int output,
@tID int output,
@rID int,
@season int output,
@OGposition int output
as
begin
select @OGposition = Result.position
from Result
where Result.dID = @dID and Result.rID = @rID

select @season = Race.season
from Race
where Race.rID = @rID

select @tID = Driver.tID
from Driver
where Driver.dID = @dID

update Result
```

```
set Result.status = 'DSQ', Result.position = 21
from Result
where Result.dID = @dID and Result.rID = @rID
end;
```

```
create or alter procedure FixFirstDSQ
@points int,
@season int,
@dID int,
@tID int
as
begin
update dChampionship
set dChampionship.points = dChampionship.points - @points
from dChampionship
where dChampionship.dID = @dID and dChampionship.season = @season

update cChampionship
set cChampionship.points = cChampionship.points - @points
from cChampionship
where cChampionship.tID = @tID and cChampionship.season = @season
end;
```

```
create or alter procedure FixResultDSQ
@rID int,
@season int,
@position int,
@oldPoints int,
@newPoints int,
@tID int,
@dID int
as
begin
select @dID = Result.dID
from Result
where Result.rID = @rID and Result.position = @position

select @tID = Driver.tID
from Driver
where Driver.dID = @dID

update dChampionship
set dChampionship.points = dChampionship.points - @oldPoints + @newPoints
from dChampionship
where dChampionship.dID = @dID and dChampionship.season = @season

update cChampionship
set cChampionship.points = cChampionship.points - @oldPoints + @newPoints
from cChampionship
where cChampionship.tID = @tID and cChampionship.season = @season

update Result
set Result.position = @position - 1, Result.points = @newPoints
from Result
where Result.dID = @dID and Result.rID = @rID
end;
```

- Java UI Code:

```
public static void AdjustResultDSQ(String connectionUrl, Scanner sc) {
    ArrayList<Integer> points = new ArrayList<>();
    points.add(25);
    points.add(18);
    points.add(15);
    points.add(12);
    points.add(10);
    points.add(8);
    points.add(6);
    points.add(4);
    points.add(2);
    points.add(1);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);
    points.add(0);

    System.out.println("Please enter the dID of the driver:");
    int dID = Integer.parseInt(sc.nextLine());

    System.out.println("Please enter the rID of the race:");
    int rID = Integer.parseInt(sc.nextLine());

    String callAdjustResultDSQ = "{call dbo.AdjustResultDSQ(?,?,?,?,?)}";
    String callAdjustPoint = "{call dbo.FixFirstDSQ(?,?,?,?)}";
    String callFixRest = "{call dbo.FixResultDSQ(?,?,?,?,?,?,?)}";

    try (Connection connection = DriverManager.getConnection(connectionUrl);
        CallableStatement AdjustResult =
connection.prepareCall(callAdjustResultDSQ);
        CallableStatement AdjustPoint =
connection.prepareCall(callAdjustPoint);
        CallableStatement FixRest = connection.prepareCall(callFixRest);
        ) {

        connection.setAutoCommit(false);

        AdjustResult.registerOutParameter(1, dID);
        AdjustResult.registerOutParameter(2, java.sql.Types.INTEGER);
        AdjustResult.setInt(3, rID);
        AdjustResult.registerOutParameter(4, java.sql.Types.INTEGER);
        AdjustResult.registerOutParameter(5, java.sql.Types.INTEGER);

        AdjustResult.execute();

        AdjustPoint.setInt(1, points.get(AdjustResult.getInt(5)-1));
        AdjustPoint.setInt(2, AdjustResult.getInt(4));
        AdjustPoint.setInt(3, AdjustResult.getInt(1));
        AdjustPoint.setInt(4, AdjustResult.getInt(2));

        AdjustPoint.execute();

        for(int i = AdjustResult.getInt(5); i<21; i++){
            FixRest.setInt(1, rID);
            FixRest.setInt(2, AdjustResult.getInt(4));
            FixRest.setInt(3, i+1);
            FixRest.setInt(4, points.get(i));
            FixRest.setInt(5, points.get(i-1));
            FixRest.setInt(6, java.sql.Types.INTEGER);
            FixRest.setInt(7, java.sql.Types.INTEGER);

            FixRest.execute();
        }

        System.out.println("Commit or Rollback?");
        String userInput = sc.nextLine();

        if (userInput.equalsIgnoreCase("Commit")){
            connection.commit();
        } else {
            connection.rollback();
        }
        sc.close();
    }

    catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- Snapshot:

Use Case 10: Replacing a Driver Mid-Season

- **Team Member: Not Implemented**
- **Description:**
    - This use case is for replacing the driver of a team with a new driver. This will be used when a team decides to replace one of their drivers due reasons such as underperforming or having an injury. As a result, races for when they are scheduled are changed to the new driver.
- **Details:**
    - The use case involves the Driver, DriverRace, and Team tables. It will utilize INSERT, SELECT, UPDATE DML statements to perform this task. Information regarding the team will be used to select for the team ID (tID) which in addition to the information of the new driver will be inserted into the Driver table. On the

DriverRace table, every occurrence of the former driver’s ID (dID) will be updated to the new driver’s dID for race IDs (rID) of races that occur after the date of the change.
    - **Business Rules:**
        - A Driver belongs to exactly one Team at a time, but a Team can have multiple Drivers
        - A Driver can participate in multiple Races, but each Race belongs to one Driver.
- The user must provide the first name, last name, date of birth (YYYY-MM-DD) of the old driver. They must also provide information of the first name, last name, date of birth, nationality, and car number of the new driver. Additionally, team name and date of the change is also required. None of this information can be null.

Use Case 11: Awarding Fastest Lap Bonus Point

- **Team Member: Sean Liu**
- **Description:**
    - This use case adds an extra bonus point for drivers and teams with the fastest lap. This will be used when a driver sets the fastest lap, they get awarded a bonus point in the championship for themselves and their team.
- **Details:**
    - The tables involved are Race, LapTime, dChampionship, cChampionship, Driver tables, and the DML used are SELECT and UPDATE. Information regarding the race will be used to find the driver ID (dID) with the lowest lap time that race by joining the race and lap time table. The dID will then be selected for and used to update the points attribute in the dChampionship and cChampionship tables. Using the Driver table, the team ID (tID) that matches the dID will be used to add an extra point for the cChampionship table. The updated points for each table will be returned to the user for verification, and they can determine whether to commit or rollback.
    - **Business Rules:**
        - A Driver can have multiple LapTimes, but each LapTime belongs to one Driver.
        - A Race can have multiple LapTimes, but each LapTime belongs to one Race.
- The user must provide information regarding the race season and race round. None of these values are null.

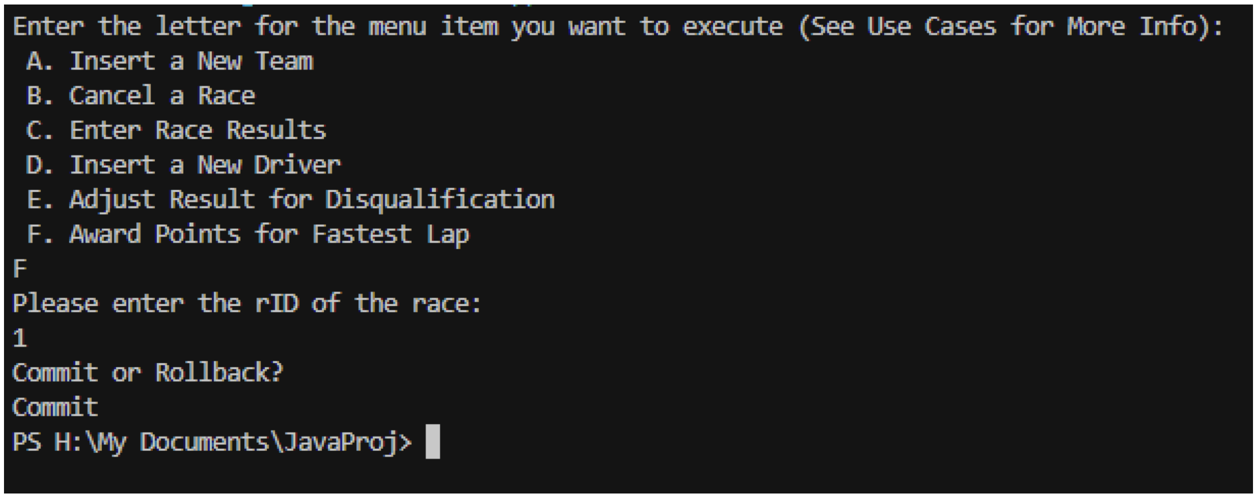
SQL Code:

```
create or alter procedure AwardFastestLap
@rID int,
@tID int output,
@dID int output,
@season int output
as
begin
select top 1 @dID = Laptime.dID
from Laptime
where Laptime.rID = @rID
order by Laptime.time ASC

select @tID = Driver.tID
from Driver
where Driver.dID = @dID

select @season = Race.season
from Race
where Race.rID = @rID

update dChampionship
set dChampionship.points = dChampionship.points+1
from dChampionship
where dChampionship.dID = @dID and dChampionship.season = @season

update cChampionship
set cChampionship.points = cChampionship.points+1
from cChampionship
where cChampionship.tID = @tID and cChampionship.season = @season
end;
```

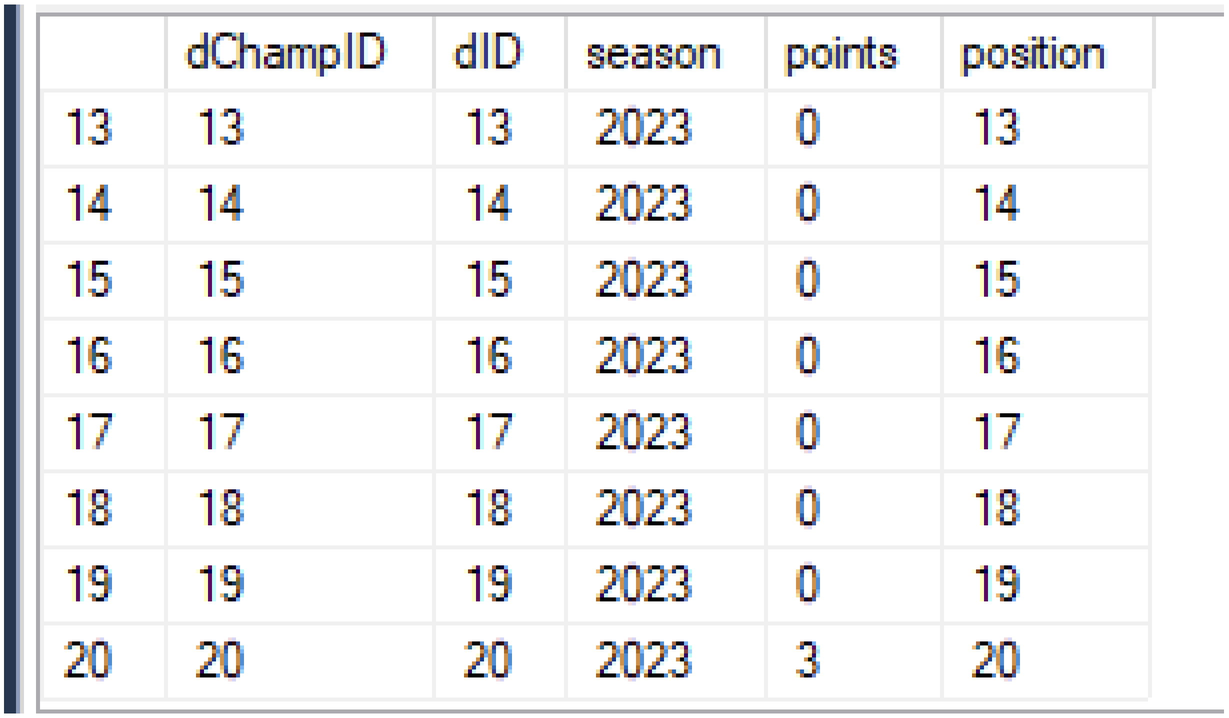
<u>Java UI Code</u>:

```
public static void AwardFastestLap(String connectionUrl, Scanner sc) {
    System.out.println("Please enter the rID of the race:");
    int race = Integer.parseInt(sc.nextLine());

    String callAwardFastestLap = "{call dbo.AwardFastestLap(?,?,?,?)}";

    try (Connection connection = DriverManager.getConnection(connectionUrl);
        CallableStatement AwardPoint =
connection.prepareCall(callAwardFastestLap);
        ) {

        connection.setAutoCommit(false);

        AwardPoint.setInt(1, race);
        AwardPoint.registerOutParameter(2, java.sql.Types.INTEGER);
        AwardPoint.registerOutParameter(3, java.sql.Types.INTEGER);
        AwardPoint.registerOutParameter(4, java.sql.Types.INTEGER);

        AwardPoint.execute();

        System.out.println("Commit or Rollback?");
        String userInput = sc.nextLine();

        if (userInput.equalsIgnoreCase("Commit")){
            connection.commit();
        } else {
            connection.rollback();
        }
        sc.close();
        }

    catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Screenshots:



```
Enter the letter for the menu item you want to execute (See Use Cases for More Info):
 A. Insert a New Team
 B. Cancel a Race
 C. Enter Race Results
 D. Insert a New Driver
 E. Adjust Result for Disqualification
 F. Award Points for Fastest Lap
F
Please enter the rID of the race:
1
Commit or Rollback?
Commit
PS H:\My Documents\JavaProj>
```



	dChampID	dID	season	points	position
13	13	13	2023	0	13
14	14	14	2023	0	14
15	15	15	2023	0	15
16	16	16	2023	0	16
17	17	17	2023	0	17
18	18	18	2023	0	18
19	19	19	2023	0	19
20	20	20	2023	3	20

Points of 3 is impossible without fastest lap.

Use Case 12: Handling a Race Cancellation and Adjusting Championship Calendar

- **Team Member Dylan Zhao**
- **Description:**
    - This use case deletes a race from the database and any results of that race as well. This would be needed if a future race needs to be canceled due to weather for example, or a race can be canceled if a major problem occurs. As a result, the race will be canceled, and any results that have been recorded for the race will also be removed.
- **Details:**
    - This use case utilizes two tables: Race and Result. Additionally, two DML statements SELECT and DELETE are used for this use case. First, the results with the same race ID (rID) as the one chosen by the user will be deleted from the table and the number of results deleted will be stored and returned. Next, the race information is deleted from the Race table and the number of rows deleted from that table is also returned. The user is then able to check the results and given the choice to commit or rollback the transaction.
    - **Business Rule:**
        - A Race has multiple Results, but each Result belongs to one Race
- **User Requirements:**
    - The user will be required to provide information regarding the race's season, round, and date (YYYY-MM-DD format) to be deleted. None of these values can be blank.

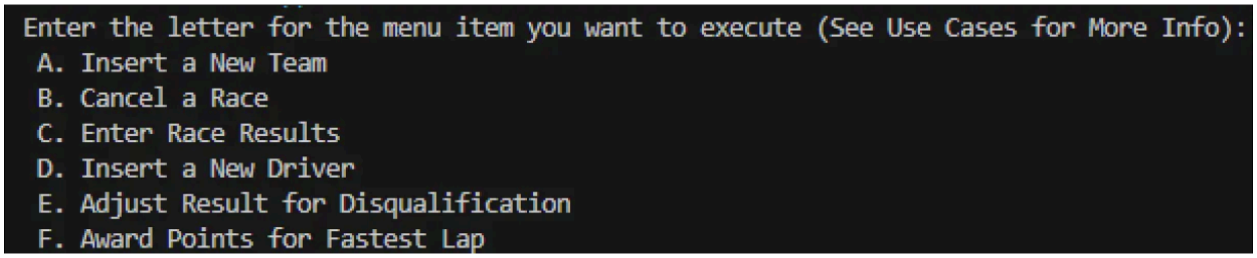
Use Case 13: Adjusting Race Results After the Race Ended (Penalty)

- **Team Member: Not Implemented**
- **Description:**
    - This use case allows users to adjust the race results in case of penalties which can affect the results.
- **Details:**
    - **Tables Involved:** Result, Driver, dChampionship, cChampionship
    - **Operations:** Select, Update
    - **Business Rules:**
        - A Driver can have multiple Results, but each Result has exactly one Driver.
        - A Driver can have (optional) one or multiple dChampion(s) (1: M) and dChampion can only have one Driver (M: 1) and cannot be null.

The user is able to update the race results in cases such as a penalty where the race results would need to be changed afterwards. This is done by adding the penalty time addition to the overall time of the Result time for the specific driver. Additionally, this new result must be adjusted within the dChampionship table which requires selecting for the driver ID and within the cChampionship table which requires selecting for the tID.

User Manual

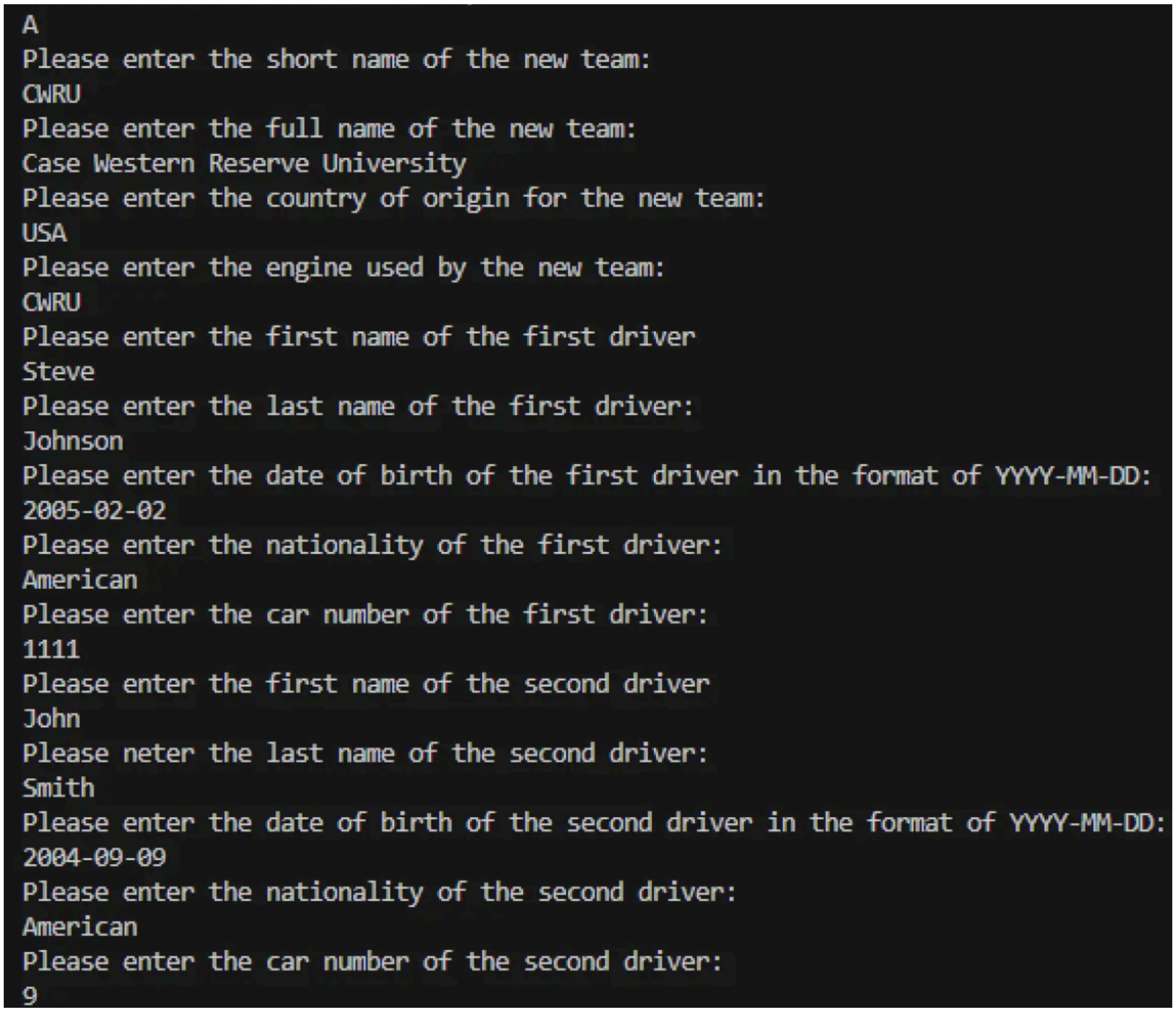
When running the Java UI program, you will be greeted with this main menu:



```
Enter the letter for the menu item you want to execute (See Use Cases for More Info):
 A. Insert a New Team
 B. Cancel a Race
 C. Enter Race Results
 D. Insert a New Driver
 E. Adjust Result for Disqualification
 F. Award Points for Fastest Lap
```

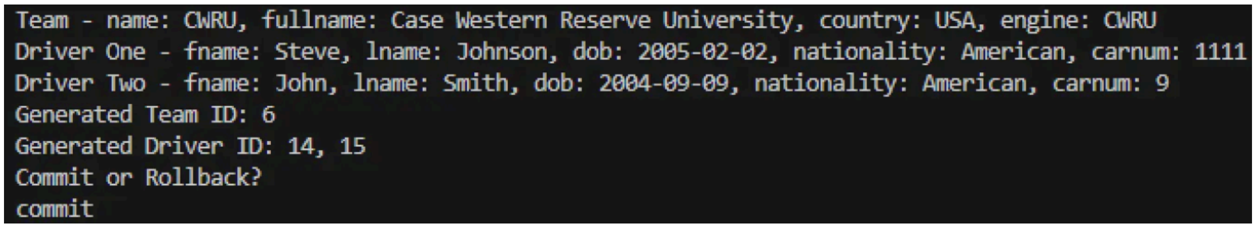
Here the user enters any letter from A-F to execute their desired action.

Pressing ‘A’ (or ‘a’)will lead to a series of prompts asking for the required information from the user to perform the requested action. For example, by inputting ‘A’, the code will display information regarding the new team and the new drivers as shown below:



```
A
Please enter the short name of the new team:
CWRU
Please enter the full name of the new team:
Case Western Reserve University
Please enter the country of origin for the new team:
USA
Please enter the engine used by the new team:
CWRU
Please enter the first name of the first driver
Steve
Please enter the last name of the first driver:
Johnson
Please enter the date of birth of the first driver in the format of YYYY-MM-DD:
2005-02-02
Please enter the nationality of the first driver:
American
Please enter the car number of the first driver:
1111
Please enter the first name of the second driver
John
Please neter the last name of the second driver:
Smith
Please enter the date of birth of the second driver in the format of YYYY-MM-DD:
2004-09-09
Please enter the nationality of the second driver:
American
Please enter the car number of the second driver:
9
```

Upon entering all the data needed, the code will display a summary of the inputs and the results. It will also prompt the user to confirm the results by asking them whether they want to commit or rollback the query as shown below:



```
Team - name: CWRU, fullname: Case Western Reserve University, country: USA, engine: CWRU
Driver One - fname: Steve, lname: Johnson, dob: 2005-02-02, nationality: American, carnum: 1111
Driver Two - fname: John, lname: Smith, dob: 2004-09-09, nationality: American, carnum: 9
Generated Team ID: 6
Generated Driver ID: 14, 15
Commit or Rollback?
commit
```

<u>Reflection</u>

What we would like to do differently next time would be to create more tables that would allow us to store more information regarding different aspects of F1. This would allow the database to be more useful to users and provide users with more interesting and relevant information. Some lessons we learned from this project is the importance of documentation for working with large and complex databases. As inserting data and manipulating can have many constraints, having good documentation allows for each group member to be on the same page as what is implemented for each table. Some hurdles we overcame were the errors we ran into while we merged our Java code together as well as creating insert statements due to the complexity of the database. Finally, a better understanding of the fastF1 API would've helped us speed up our DML process of populating the relevant tables. It was tedious to understand the output format of the API and in hindsight there were simpler APIs that we didn't explore which could've boosted our productivity.

Responsibility:

- Atharva
    - Past Work: Use Cases, Indexes, Business rules
    - Future Work: Use Cases 1, 5; Java UI Code
- Sean
    - Past Work: Relational Schemas, Use Cases, Problem Statement
    - Future Work: Use Cases 9, 11; Java UI code
- Dylan
    - Past Work: ERD, Use Cases, Planned Applications
    - Future: Use Cases 12, 3; Java UI Code

Indexing (Non-clustering index)

Driver
(lname)
Justification: Dense secondary index for fast lookups by last name, a common query filter.
sample use-case: Quickly find details for drivers named “Verstappen.”

Team
(name)
Justification: Dense secondary index to speed up searches by short team name, a popular lookup.
sample use-case: Immediately retrieve team info for “Ferrari.”

Race
(season, round)
Justification: Dense composite index to efficiently locate or order races by season and round.
sample use-case: Get the 3rd race from season 2022 without scanning all races.

DriverRace
(rID)
Justification: Dense secondary index helps listing all drivers in a specific race, minimizing overhead of a dID-based index.
sample use-case: Display drivers participating in race rID=15 quickly.

Circuit
(country)
Justification: Dense secondary index to handle frequent queries filtering by circuit location.
sample use-case: Retrieve circuits based in Italy instantly.

Result
(rID, position)
Justification: Dense composite index supporting popular result listings per race in finishing order.
sample use-case: Quickly view podium finishers for race rID=8.

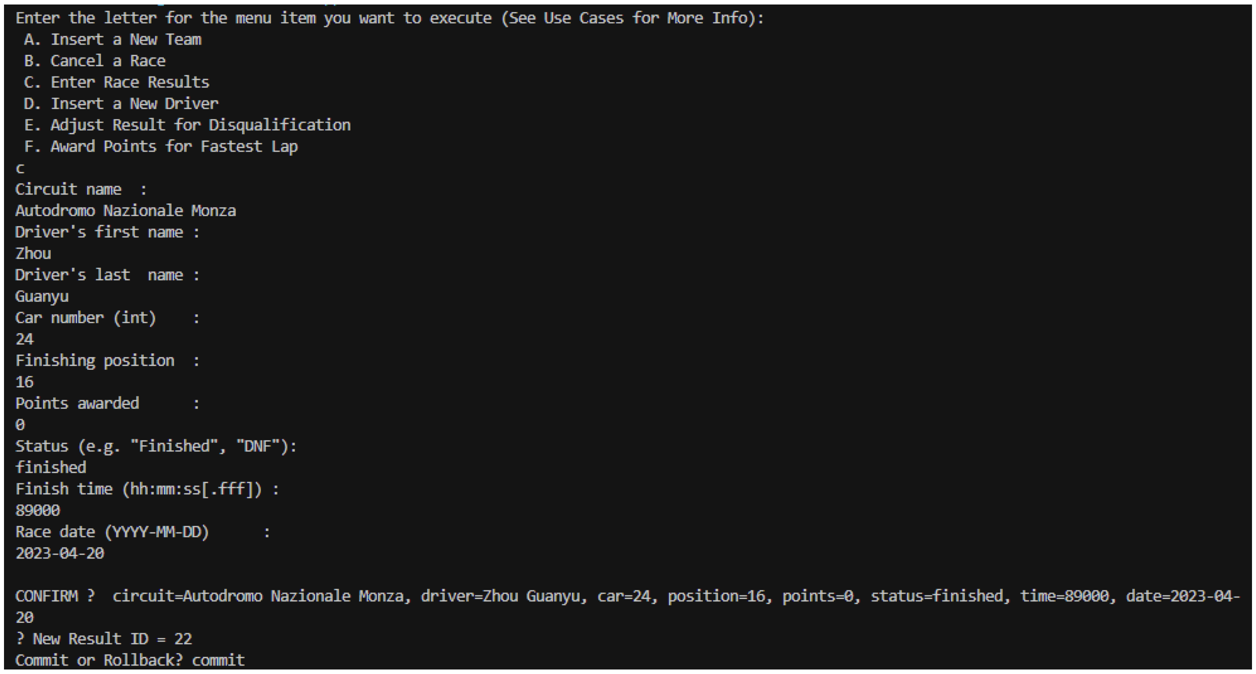
cChampionship
(season, position)
Justification: Dense composite index for common queries on constructor standings in each

season.
sample use-case: Quickly see which team topped the table in 2021.

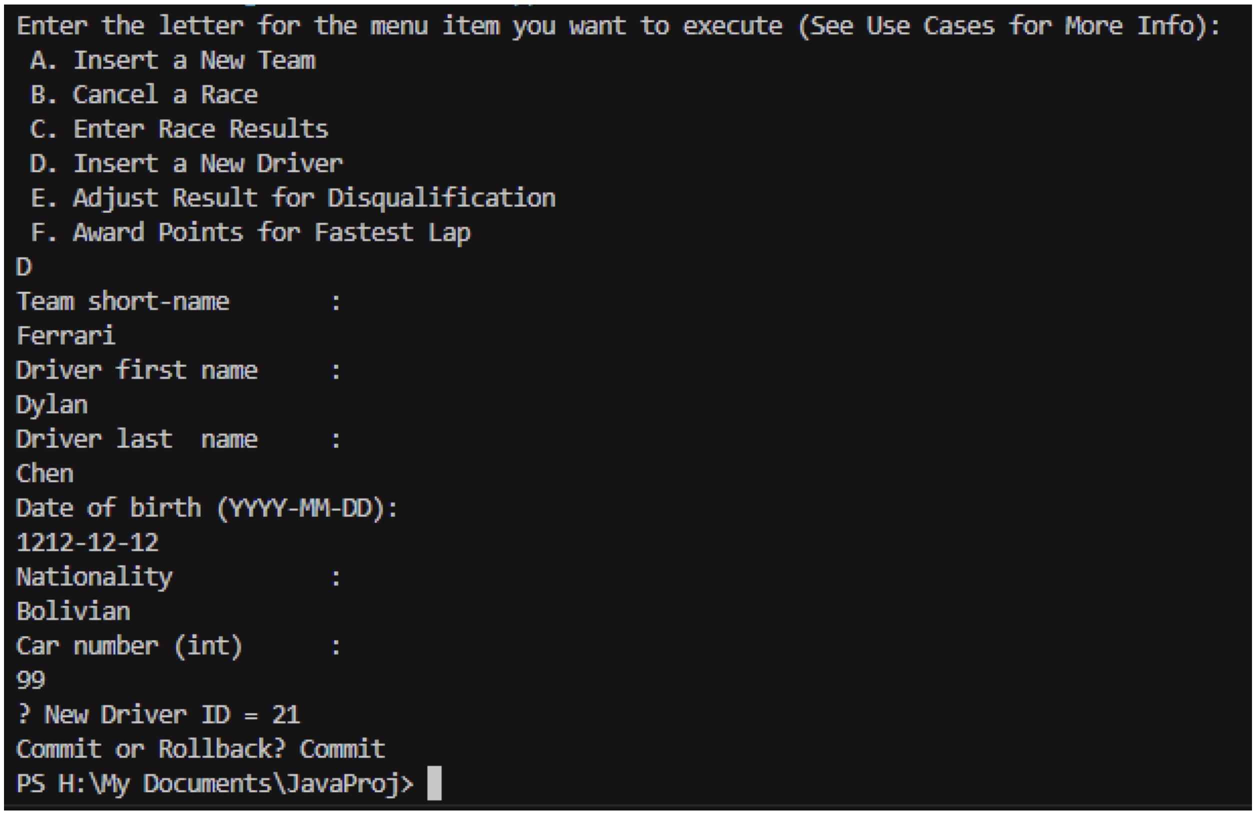
dChampionship
(season, position)
Justification: Dense composite index for frequent lookups of driver standings each season.
sample use-case: Find the top driver of 2022 without scanning the entire table.

LapTime
(rID, lap)
Justification: Dense composite index to optimize retrieval of lap records by race and lap number.
sample use-case: Rapidly list each lap for race rID=3 in ascending order.

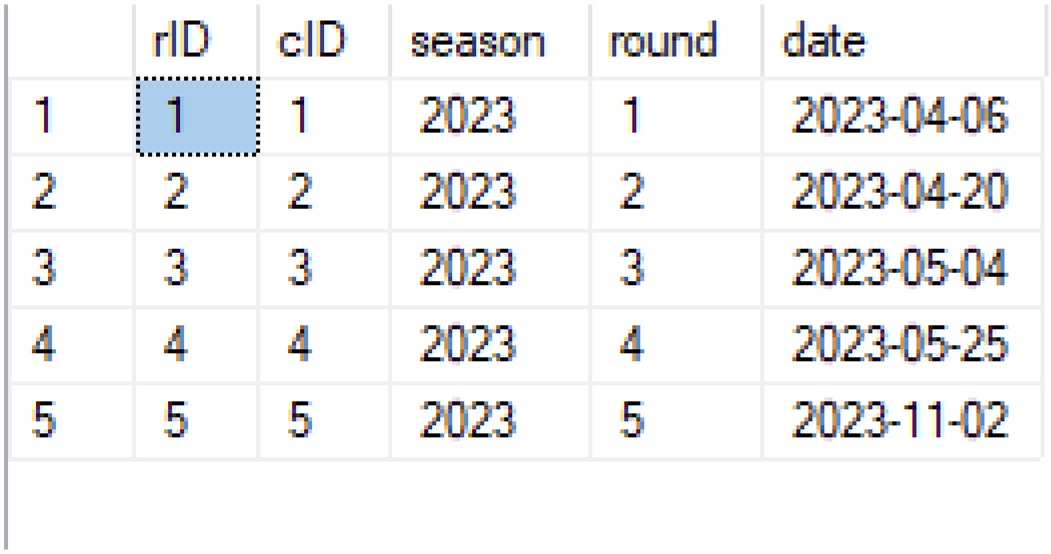
1. Use Case 11 (Complete)



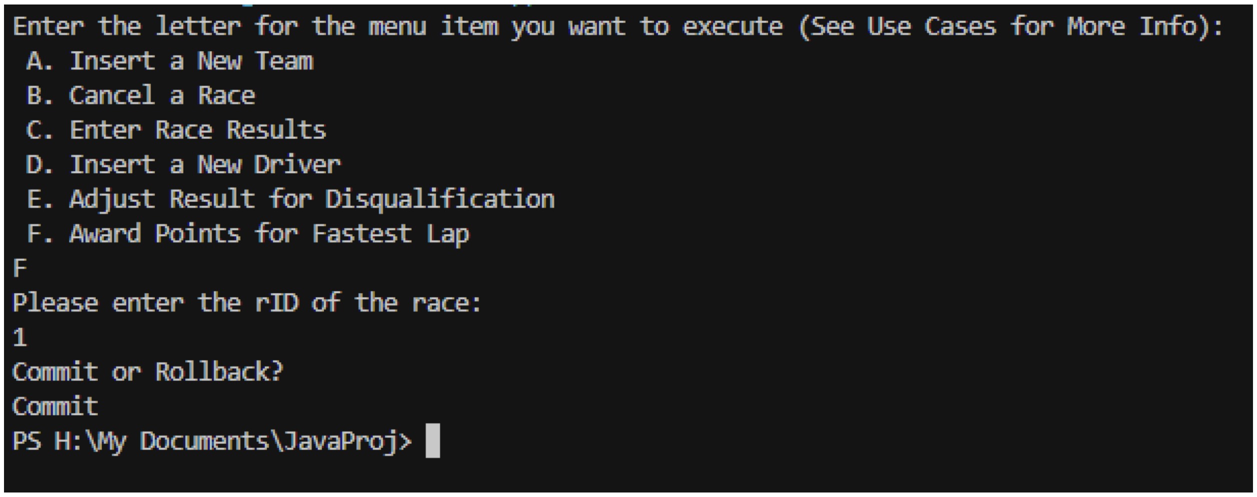
```
Enter the letter for the menu item you want to execute (See Use Cases for More Info):
 A. Insert a New Team
 B. Cancel a Race
 C. Enter Race Results
 D. Insert a New Driver
 E. Adjust Result for Disqualification
 F. Award Points for Fastest Lap
c
Circuit name  :
Autodromo Nazionale Monza
Driver's first name :
Zhou
Driver's last  name :
Guanyu
Car number (int)    :
24
Finishing position  :
16
Points awarded      :
0
Status (e.g. "Finished", "DNF"):
finished
Finish time (hh:mm:ss[.fff]) :
89000
Race date (YYYY-MM-DD)      :
2023-04-20

CONFIRM ?  circuit=Autodromo Nazionale Monza, driver=Zhou Guanyu, car=24, position=16, points=0, status=finished, time=89000, date=2023-04-
20
? New Result ID = 22
Commit or Rollback? commit
```



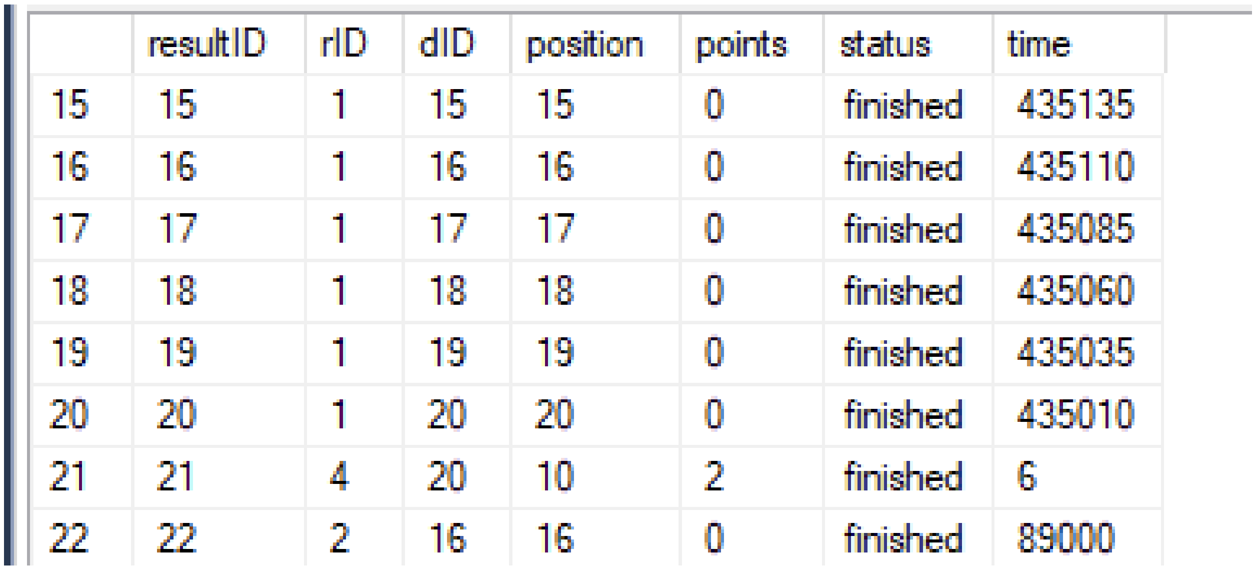
```
Enter the letter for the menu item you want to execute (See Use Cases for More Info):
 A. Insert a New Team
 B. Cancel a Race
 C. Enter Race Results
 D. Insert a New Driver
 E. Adjust Result for Disqualification
 F. Award Points for Fastest Lap
D
Team short-name         :
Ferrari
Driver first name       :
Dylan
Driver last  name       :
Chen
Date of birth (YYYY-MM-DD):
1212-12-12
Nationality             :
Bolivian
Car number (int)        :
99
? New Driver ID = 21
Commit or Rollback? Commit
PS H:\My Documents\JavaProj>
```



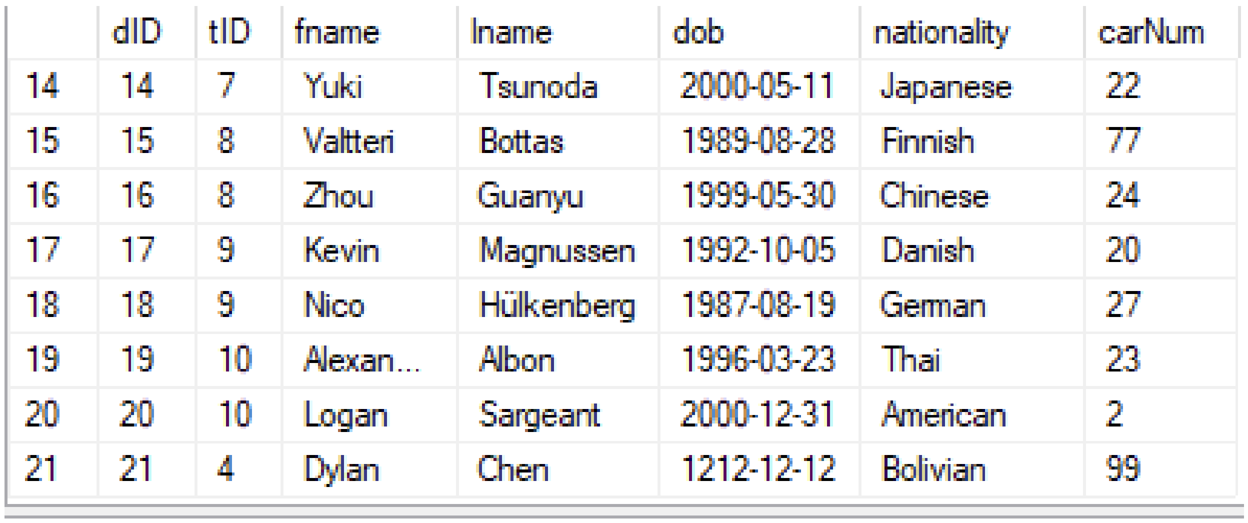
	rID	cID	season	round	date
1	1	1	2023	1	2023-04-06
2	2	2	2023	2	2023-04-20
3	3	3	2023	3	2023-05-04
4	4	4	2023	4	2023-05-25
5	5	5	2023	5	2023-11-02



```
Enter the letter for the menu item you want to execute (See Use Cases for More Info):
 A. Insert a New Team
 B. Cancel a Race
 C. Enter Race Results
 D. Insert a New Driver
 E. Adjust Result for Disqualification
 F. Award Points for Fastest Lap
F
Please enter the rID of the race:
1
Commit or Rollback?
Commit
PS H:\My Documents\JavaProj>
```



	resultID	rID	dID	position	points	status	time
15	15	1	15	15	0	finished	435135
16	16	1	16	16	0	finished	435110
17	17	1	17	17	0	finished	435085
18	18	1	18	18	0	finished	435060
19	19	1	19	19	0	finished	435035
20	20	1	20	20	0	finished	435010
21	21	4	20	10	2	finished	6
22	22	2	16	16	0	finished	89000



	dID	tID	fname	lname	dob	nationality	carNum
14	14	7	Yuki	Tsunoda	2000-05-11	Japanese	22
15	15	8	Valtteri	Bottas	1989-08-28	Finnish	77
16	16	8	Zhou	Guanyu	1999-05-30	Chinese	24
17	17	9	Kevin	Magnussen	1992-10-05	Danish	20
18	18	9	Nico	Hülkenberg	1987-08-19	German	27
19	19	10	Alexan...	Albon	1996-03-23	Thai	23
20	20	10	Logan	Sargeant	2000-12-31	American	2
21	21	4	Dylan	Chen	1212-12-12	Bolivian	99