FORMULA 1 RACING

PHASE 2

Mansi Goel, Urvi Rawat
615, 615
mgoel2@jh.edu, urawat1@jh.edu

APPLICATION DOMAIN: Our focus area centres around the world of sports, with a particular emphasis on Formula 1 (F1) racing. In this project, we will dive into the analysis of various aspects of F1 racing, including driver performance, race events, and circuits. Our inspiration for selecting this topic comes from our enthusiasm for F1 racing.

PROJECT WEBSITE: https://www.ugrad.cs.jhu.edu/~urawat1/project_main.html

PROJECT CHANGES SINCE PHASE I: We have included several more complex queries to the questions we would like to answer from the database. These queries are mentioned below in this document as part of the Phase I section.

DATABASE SOURCE & LOADING TECHNIQUE: The F1 racing dataset is obtained from Kaggle. The dataset given here is modified according to the needs of this project. This involves selecting certain csv files and selecting relevant columns from them. These csv files are then converted into SQL insert queries with the help of Python code to populate our database tables. This was done so that further queries can be run smoothly. The attributes, including primary key and foreign key constraints are specified based on our relation model. A sample create table statement is shown below.

```
create table Qualifying (
    raceID          INTEGER NOT NULL, -- 1
    driverID        INTEGER NOT NULL, -- 1
    constructorID   INTEGER NOT NULL, -- 1
    finalPosition   INTEGER, -- 1
    FOREIGN KEY (raceID) REFERENCES Races(raceID),
    FOREIGN KEY (driverID) REFERENCES Drivers(driverID),
    FOREIGN KEY (constructorID) REFERENCES Constructors(constructorID)
);
```

USER GUIDE: Our project is implemented on the standard platform using dbase.cs.jhu.edu. We first run the SQL files which consist our data and create table statements. We then run the SQL file which contains the views and stored procedures for the queries in our project. After this, we have created multiple PHP and HTML files which help us take user input and display the query results on a public website. These files are also run on the standard platform. The final step is to open the website link in the browser and view results of your choice.

AREAS OF SPECIALIZATION:

- Advanced Graphic User Interface: The project incorporates advanced GUI concepts to create a **visually appealing** and **user-friendly** website dedicated to Formula 1 racing using **HTML and CSS**. The design employs a clean and responsive layout, ensuring an optimal viewing experience across various devices. Multiple pages are organized in a **card-based layout**, each representing different facets of Formula 1, such as drivers, circuits, races, and more. The aesthetic appeal is enhanced by **background images**. The project also incorporates complex methods of taking user input, such as drop downs. The website **integrates PHP** to connect to a MySQL database and execute **stored procedures** for dynamic data retrieval. Notably, **Chart.js** is utilized to generate an **interactive bar chart**s for certain queries. The inclusion of **Google Fonts** and **error-handling mechanisms** further contributes to the advanced GUI implementation.

- Data Extraction and Preprocessing: This project implements a **streamlined data preprocessing** strategy using **Python** and **Pandas** to transform raw data from multiple **CSV files** into a format suitable for **relational databases**. The process involves reading files such as "circuits.csv," "constructors.csv," etc. and selecting relevant columns. A key function, `**df_to_sql_insert**`, generates SQL INSERT statements for the selected data. The function iterates through rows of a dataframe, formatting values and creating SQL statements. It ensures proper handling of strings by enclosing them in single quotes and escaping existing single quotes within the strings. This approach makes it easier to get the data ready for insertion into a relational database.

PROJECT STRENGHTS:

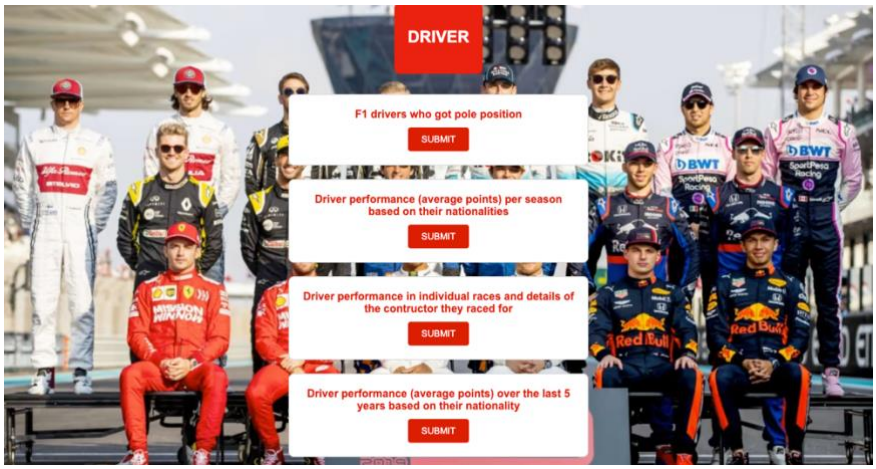
- The project website has an aesthetic appeal to it. All queries are neatly organized into sub sections making it easy for the user to retrieve results of their choice.
- The implementation of user-driven queries allows for dynamic exploration of the dataset, enhancing user engagement and interactivity.
- The queries also provide an easy way for users to input by creating drop-down menus. This allows users to easily navigate through the options and minimize the chances of typing errors.
- The results are displayed in organized tables which makes it easier for the user to understand and interpret results.
- The project's emphasis on data visualization and statistical analysis empowers users to make informed decisions and draw meaningful conclusions from the Formula 1 dataset.
- The diverse range of English queries demonstrates the project's versatility, covering topics from driver achievements to circuit statistics, offering a holistic exploration of Formula 1.

PROJECT LIMITATIONS:

- Some tables in the database contain NULL values which might affect certain results. This could be improved by implementing better data collection practices.
- The project connects to the database using a single set of credentials, potentially posing security risks. Implementing secure credential management and could enhance overall database security.
- The queries currently retrieve results based on existing data in the database. We could explore possibilities for real-time data integration or periodic updates to keep the dataset reflective of the latest Formula 1 events.

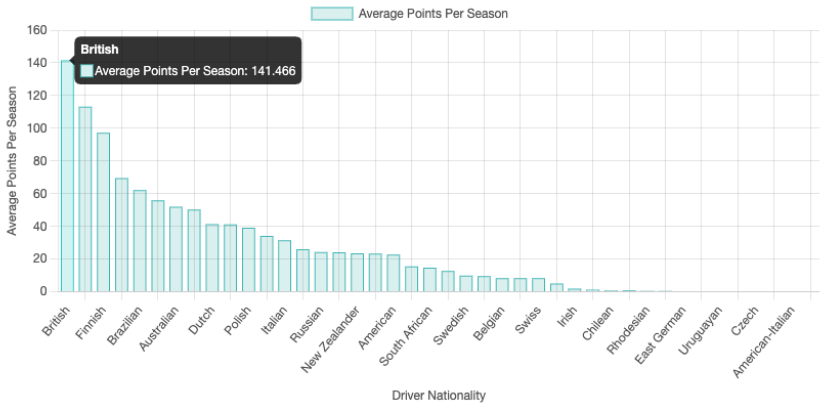
SAMPLE OUTPUT: The screenshots below only depict a subset of our results. The entire query results can be viewed on the website.

Sample Cover Page (for a category)



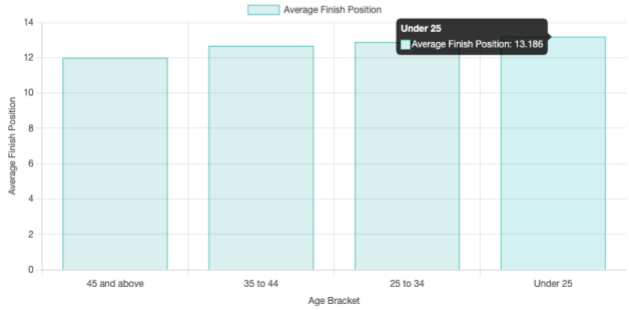
Query 1: Driver Performance (average points) per season based on their nationality & Visualization. Clicking on the bars in the plot displays the average points.

Nationality	Number of Drivers	Average Points Per Season
British	165	141.4658
German	50	113.2429
Finnish	9	97.2444
Spanish	15	69.6000
Brazilian	32	62.2545
Monegasque	4	56.0000
Australian	17	52.0556
French	73	50.3382

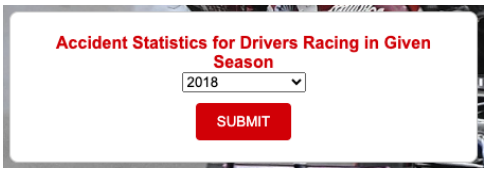


Query 2: Driver Performance Based on Age Bracket

Age Bracket	Number of Drivers	Average Finish Position
45 and above	50	11.9885
35 to 44	289	12.6747
25 to 34	602	12.8847
Under 25	209	13.1862

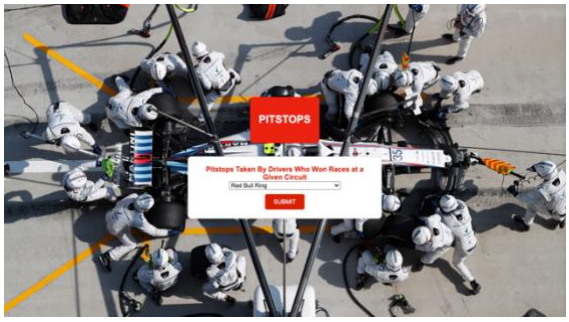


Query 3: Accident Statistics in Season Input by User



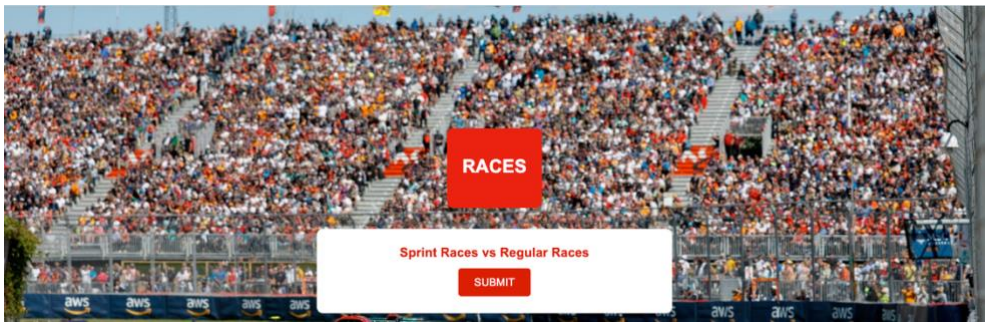
Driver Name	Accident Count	Total Races	Accident Percentage
Nico Hulkenberg	2	21	9.52
Fernando Alonso	1	21	4.76
Charles Leclerc	1	21	4.76
Marcus Ericsson	1	21	4.76
Romain Grosjean	1	21	4.76
Sebastian Vettel	1	21	4.76

Query 4: Pitstops Taken by Driver who Won Races at Given Circuit



First Name	Last Name	Average Pitstops
Max	Verstappen	1.2500
Nico	Rosberg	1.5000
Valtteri	Bottas	1.5000
Lewis	Hamilton	2.5000
Charles	Leclerc	3.0000

Query 5: Sprint Race vs Regular Race Performance



Driver Name	Average Sprint Race Position	Average Regular Race Position
Lewis Hamilton	6.8000	4.7076
Fernando Alonso	11.8000	8.4097
Kimi Raikkonen	15.5000	8.4915
Robert Kubica	18.0000	10.6465
Sebastian Vettel	12.4000	6.9725
Sergio Perez	8.2000	9.8596
Daniel Ricciardo	7.6000	9.7578
Valtteri Bottas	4.4000	7.2813
Kevin Magnussen	7.5000	13.1729

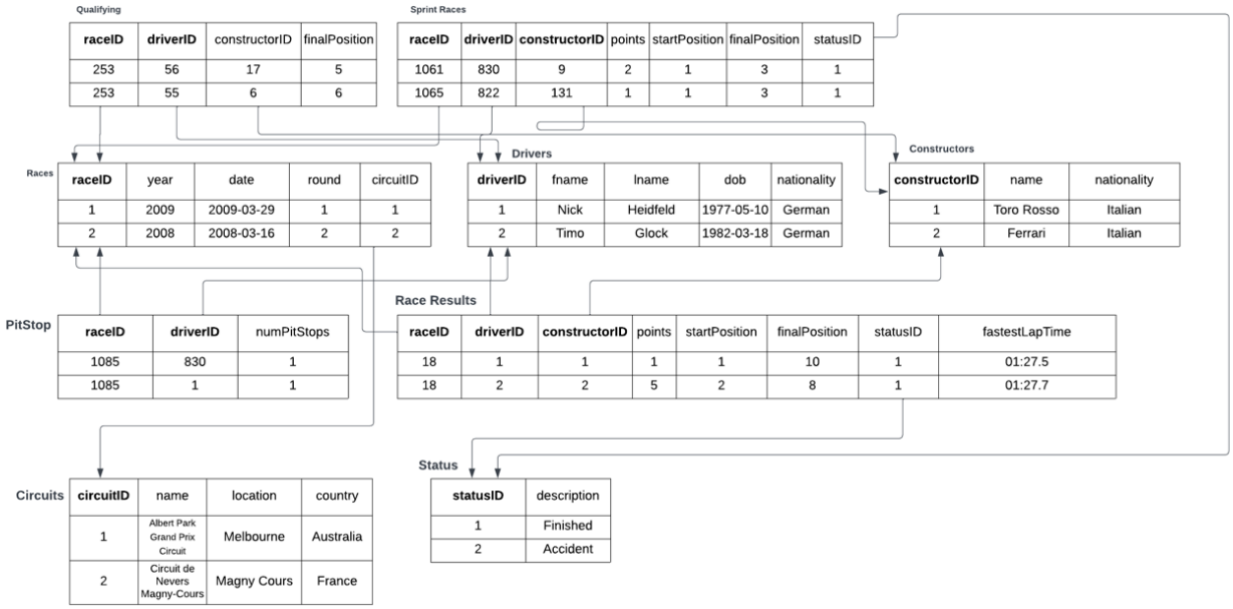
PHASE 1

TARGET DOMAIN: Our focus area centres around the world of sports, with a particular emphasis on Formula 1 (F1) racing. In this project, we will dive into the analysis of various aspects of F1 racing, including driver performance, race events, and circuits. Our inspiration for selecting this topic comes from our enthusiasm for F1 racing.

ENGLISH QUERIES:

1. Driver names who have gotten pole (fastest lap time in qualifying race).
2. List the nationality, number of drivers of each nationality and average points per season.
3. Calculate the average points earned per race by drivers of each nationality over the last 5 years.
4. List the driver names, their nationality, constructor they have raced for, in which year the points they earned in that race.
5. Display the number of drivers and their average finishing position in the age brackets 0-25, 26-34, 35-44, and 45+.
6. List the count of circuits in each country.
7. List the name of constructors at a circuit (user input) and they average points.
8. List the driver names, number of wins and fastest lap time for drivers who have raced at a circuit.
9. List the driver name and their average race position in sprint and regular races.
10. For a year (user input), list the driver name, total races they participated in, total points and their average finishing positions.
11. For a race (user input), list the raceID, race year, driver in that race, their final position and whether they had an accident or not.
12. For a year (user input), list the driver name, the total races they raced in, out of which the number of races that lead to an accident. Also list the accident occurrence percentage.
13. List the most common reason for drivers not finishing a race.
14. List the driver name and the type of race-ending incidents they are more prone to such as mechanical failures or accidents.
15. List the circuit name. location, total races at that circuit and the accident percentage at that circuit.
16. List the average number of pitstops of all drivers that have won a race at a circuit (user input).
17. For a raceID (user input), list the driver in that race, their start and finish position, pitstops taken and the change in their position.
18. For any driver (user input), list the circuit location where they have raced, total races in that country and their average finish position, total points.
19. List the constructor nationalities, the races they have participated in, average finish position and total points obtained.
20. List the constructor name, the total races they have participated in, out of which how many ended in first place and the corresponding percentage.

RELATIONAL MODEL:



SAMPLE SQL QUERIES:

```
1.  SELECT
      C.name AS CircuitName,
      C.location AS Location,
      C.country AS Country,
      COUNT(DISTINCT R.raceID) AS TotalRaces,
      COUNT(DISTINCT CASE WHEN S.description LIKE '%accident%' THEN R.raceID ELSE NULL END) AS AccidentRaces,
      ROUND((COUNT(DISTINCT CASE WHEN S.description LIKE '%accident%' THEN R.raceID ELSE NULL END) * 100.0 / COUNT(DISTINCT
R.raceID)), 2) AS AccidentPercentage
    FROM Circuits C
    JOIN Races R ON C.circuitID = R.circuitID
    JOIN RaceResults RR ON R.raceID = RR.raceID
    JOIN Status S ON RR.statusID = S.statusID
    GROUP BY C.circuitID
    HAVING COUNT(DISTINCT R.raceID) > 0
    ORDER BY AccidentPercentage DESC, Country DESC;
```

```
2.  SELECT
      CASE
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) < 25 THEN 'Under 25'
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 25 AND 34 THEN '25 to 34'
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 35 AND 44 THEN '35 to 44'
        ELSE '45 and above'
      END AS AgeBracket,
      COUNT(DISTINCT D.driverID) AS NumberOfDrivers,
      AVG(RR.finalPosition) AS AverageFinishPosition
    FROM Drivers D
    JOIN RaceResults RR ON D.driverID = RR.driverID
    JOIN Races R ON RR.raceID = R.raceID
    GROUP BY
      CASE
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) < 25 THEN 'Under 25'
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 25 AND 34 THEN '25 to 34'
        WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 35 AND 44 THEN '35 to 44'
```

```
        ELSE '45 and above'
    END
ORDER BY AVG(TIMESTAMPDIFF(YEAR, D.dob, R.date));
```

```
3.  SELECT
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
    SUM(CASE WHEN S.description LIKE '%accident%' THEN 1 ELSE 0 END) AS AccidentCount,
    COUNT(R.raceID) AS TotalRaces,
    ROUND((SUM(CASE WHEN S.description LIKE '%accident%' THEN 1 ELSE 0 END) * 100.0 /  COUNT(R.raceID)), 2) AS AccidentPercentage
FROM Races R
JOIN RaceResults RR ON R.raceID = RR.raceID
JOIN Drivers D ON RR.driverID = D.driverID
JOIN Status S ON RR.statusID = S.statusID
WHERE R.year = myyear
GROUP BY DriverName
HAVING AccidentCount > 0
ORDER BY AccidentPercentage DESC;
```

```
4.  SELECT D.fname, D.lname, AVG(P.numPitStops) AS avgPits
    FROM (
        SELECT W.raceID, W.driverID
        FROM RaceWinnersView AS W
        WHERE W.circuitID = mycircuitID
    ) AS R, Drivers AS D, PitStops AS P
    WHERE D.driverID = R.driverID AND
        P.driverID = R.driverID AND
        P.raceID = R.raceID
    GROUP BY R.driverID
    ORDER BY AVG(P.numPitStops) ASC;
```

```
5.  SELECT
    CONCAT (D.fName, ' ', D.lName) AS DriverName,
    R.raceID,
    RR.startPosition,
    RR.finalPosition,
    PS.numPitStops,
    (RR.startPosition - RR.finalPosition) AS PositionChange
```

DATASET: The F1 racing dataset is obtained from Kaggle. The dataset given here is modified according to the needs of this project. This involves selecting certain csv files and selecting relevant columns from them. These csv files are then converted into SQL tables so that further queries can be run smoothly. For our questions/queries, we will take inputs from the user through a web interface built in HTML, PHP and CSS.

VIEWS: As part of our project, we will implement the following views.

1. A view with a row consisting of driver, constructor, circuit, year and points for each pair of driver and race. These come from the following tables: Races, RaceResults
2. A view with total number of races and the accidents using tables Races, RaceResults and Circuits.
3. A view with all races at circuits a driver has won utilizing Races and RaceResults.

ADVANCED TOPIC IMPLEMENTATION: We plan to implement triggers (minor) and an advanced GUI (major) in this project.

APPENDIX (Create Table Statements and SQL Code for Procedures)

SQL DATABASE DEFINITION LANGUAGE: The tables are created after careful consideration of primary and foreign key constraints as well as NOT NULL constraints. The create table statements below give the table specification.

```
create table Circuits (
    circuitID          INTEGER NOT NULL, -- 1
    name               VARCHAR(100), -- Yarowsky Circuit
    location           VARCHAR(100), -- Maryland
    country            VARCHAR(100), -- USA
    PRIMARY KEY (circuitID)
);
create table Constructors (
    constructorID      INTEGER NOT NULL, -- 1
    name               VARCHAR(100), -- Yarowsky Team
    nationality        VARCHAR(100), -- American
    PRIMARY KEY (constructorID)
);

create table Drivers (
    driverID           INTEGER NOT NULL, -- 1
    fName              VARCHAR(100), -- David
    lName              VARCHAR(100), -- Yarowsky
    dob                DATE, -- 1982-10-01
    nationality        VARCHAR(100), -- American
    PRIMARY KEY (driverID)
);

create table Status (
    statusID           INTEGER NOT NULL, -- 1
    description        VARCHAR(100), -- Finished
    PRIMARY KEY (statusID)
);

create table Races (
    raceID             INTEGER NOT NULL, -- 1
    year               INTEGER, -- 2022
    round              INTEGER, -- 1
    circuitID          INTEGER, -- 1
    date               DATE, -- 2022-12-17
    PRIMARY KEY (raceID)
);

create table RaceResults (
    raceID             INTEGER NOT NULL, -- 1
    driverID           INTEGER NOT NULL, -- 1
    constructorID      INTEGER NOT NULL, -- 1
    startPosition      INTEGER, -- 1
    finalPosition      INTEGER, -- 1
    points             INTEGER, -- 22
    fastestLapTime     TIME, -- 01:34.2
    statusID           INTEGER NOT NULL, -- 1
    FOREIGN KEY (raceID) REFERENCES Races(raceID),
    FOREIGN KEY (driverID) REFERENCES Drivers(driverID),
    FOREIGN KEY (constructorID) REFERENCES Constructors(constructorID),
```

```
    FOREIGN KEY (statusID) REFERENCES Status(statusID)
);

create table Qualifying (
    raceID          INTEGER NOT NULL, -- 1
    driverID        INTEGER NOT NULL, -- 1
    constructorID   INTEGER NOT NULL, -- 1
    finalPosition   INTEGER, -- 1
    FOREIGN KEY (raceID) REFERENCES Races(raceID),
    FOREIGN KEY (driverID) REFERENCES Drivers(driverID),
    FOREIGN KEY (constructorID) REFERENCES Constructors(constructorID)
);

create table SprintRaces (
    raceID          INTEGER NOT NULL, -- 1
    driverID        INTEGER NOT NULL, -- 1
    constructorID   INTEGER NOT NULL, -- 1
    startPosition   INTEGER, -- 1
    finalPosition   INTEGER, -- 1
    points          INTEGER, -- 3
    statusID        INTEGER, -- 1
    FOREIGN KEY (raceID) REFERENCES Races(raceID),
    FOREIGN KEY (driverID) REFERENCES Drivers(driverID),
    FOREIGN KEY (constructorID) REFERENCES Constructors(constructorID)

);

create table PitStops (
    raceID          INTEGER NOT NULL, -- 1
    driverID        INTEGER NOT NULL, -- 1
    numPitStops  INTEGER, -- 0
    FOREIGN KEY (raceID) REFERENCES Races(raceID),
    FOREIGN KEY (driverID) REFERENCES Drivers(driverID)
);
```

SQL CODE FOR VIEWS:

```
DELIMITER //

-- Views
CREATE OR REPLACE VIEW RacePointsView AS
SELECT RR.driverID, RR.constructorID, R.circuitID, year, RR.points
FROM Races AS R, RaceResults AS RR
WHERE R.raceID = RR.raceID; //

CREATE OR REPLACE VIEW AccidentsView AS
SELECT A.name, A.location, A.accidents, COUNT(R.circuitID) AS numRaces
FROM (
   SELECT C.circuitID, C.name, C.location, COUNT(C.circuitID) AS accidents
   FROM Circuits AS C, Races AS R, RaceResults AS RR
   WHERE C.circuitID = R.circuitID AND
```

```
      R.raceID = RR.raceID AND
      (RR.statusID = 3 OR RR.statusID = 4)
   GROUP BY C.circuitID
) AS A, Races AS R
WHERE A.circuitID = R.circuitID
GROUP BY R.circuitID; //

CREATE OR REPLACE VIEW RaceWinnersView AS
SELECT RR.raceID, RR.driverID, R.circuitID
FROM Races AS R, RaceResults AS RR
WHERE R.raceID = RR.raceID AND
   RR.finalPosition = 1; //
```

SQL CODE FOR PROCEDURES:

```
--DRIVERS

CREATE OR REPLACE PROCEDURE D1()
BEGIN
   SELECT DISTINCT D.fname AS firstName, D.lname AS lastName
   FROM Drivers AS D, Qualifying AS Q
   WHERE D.driverID = Q.driverID AND Q.finalPosition = 1
   ORDER BY D.lname ASC;
END; //

CREATE OR REPLACE PROCEDURE D2()
BEGIN
   SELECT
      D.nationality,
      COUNT(DISTINCT D.driverID) AS numDrivers,
      SUM(RR.points) / COUNT(DISTINCT R.year) AS AveragePointsPerSeason
   FROM Drivers D
   JOIN RaceResults RR ON D.driverID = RR.driverID
   JOIN Races R ON RR.raceID = R.raceID
   GROUP BY D.nationality
   ORDER BY
      COUNT(DISTINCT D.driverID) DESC;
END; //


CREATE OR REPLACE PROCEDURE D3()
BEGIN
   SELECT
```

```
    D.nationality AS DriverNationality,
    COUNT(DISTINCT R.raceID) AS TotalRaces,
    SUM(RR.points) AS TotalPoints,
    AVG(RR.points) AS AveragePointsPerRace
  FROM Drivers D
  JOIN RaceResults RR ON D.driverID = RR.driverID
  JOIN Races R ON RR.raceID = R.raceID
  WHERE R.year >= (SELECT MAX(year) - 5 FROM Races)
  GROUP BY D.nationality
  ORDER BY AveragePointsPerRace DESC;
END; //

CREATE OR REPLACE PROCEDURE D4()
BEGIN
  SELECT
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
    D.nationality AS DriverNationality,
    C.name AS ConstructorName,
    R.year AS RaceYear,
    -- R.round AS RaceRound,
    R.raceID AS Race,
    RR.FinalPosition AS RacePosition,
    RR.points AS PointsEarned
  FROM RaceResults RR
  JOIN Drivers D ON RR.driverID = D.driverID
  JOIN Constructors C ON RR.constructorID = C.constructorID
  JOIN Races R ON RR.raceID = R.raceID
  ORDER BY R.year DESC, R.round DESC, RR.FinalPosition ASC;
END; //

CREATE OR REPLACE PROCEDURE D5()
BEGIN
  SELECT
    CASE
      WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) < 25 THEN 'Under 25'
      WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 25 AND 34 THEN '25 to 34'
      WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 35 AND 44 THEN '35 to 44'
      ELSE '45 and above'
    END AS AgeBracket,
    COUNT(DISTINCT D.driverID) AS NumberOfDrivers,
    AVG(RR.finalPosition) AS AverageFinishPosition
  FROM Drivers D
  JOIN RaceResults RR ON D.driverID = RR.driverID
  JOIN Races R ON RR.raceID = R.raceID
```

```
    GROUP BY
        CASE
            WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) < 25 THEN 'Under 25'
            WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 25 AND 34 THEN '25 to 34'
            WHEN TIMESTAMPDIFF(YEAR, D.dob, R.date) BETWEEN 35 AND 44 THEN '35 to 44'
            ELSE '45 and above'
        END
    ORDER BY AVG(TIMESTAMPDIFF(YEAR, D.dob, R.date));
END; //

CREATE OR REPLACE PROCEDURE C1()
BEGIN
    SELECT country, COUNT(circuitID) as numCircuits
    FROM Circuits
    GROUP BY country
    ORDER BY COUNT(country) DESC;
END; //

CREATE OR REPLACE PROCEDURE C2(IN mycircuitID VARCHAR(100))
BEGIN
    SELECT CS.name, AVG(R.points) As avgPts
    FROM (
        SELECT P.constructorID, P.points
        FROM RacePointsView AS P
        WHERE P.circuitID = mycircuitID
    ) AS R, Constructors AS CS
    WHERE CS.constructorID = R.constructorID
    GROUP BY R.constructorID
    ORDER BY AVG(R.points) DESC;
END; //

CREATE OR REPLACE PROCEDURE C3(IN mycircuitID VARCHAR(100))
BEGIN
    SELECT
        D.fname,
        D.lname,
        R.numWins,
        MIN(RR.fastestLapTime) AS FastestLapTime
    FROM (
        SELECT
            W.driverID,
            COUNT(W.raceID) AS numWins
        FROM RaceWinnersView AS W
        WHERE W.circuitID = mycircuitID
```

```
    GROUP BY W.driverID
  ) AS R
  JOIN Drivers AS D ON D.driverID = R.driverID
  JOIN RaceResults RR ON D.driverID = RR.driverID
  JOIN Races Ra ON RR.raceID = Ra.raceID AND Ra.circuitID = mycircuitID
  WHERE RR.FastestLapTime IS NOT NULL
  GROUP BY D.fname, D.lname, R.numWins
  ORDER BY R.numWins DESC, FastestLapTime;
END; //

CREATE OR REPLACE PROCEDURE R1()
BEGIN
  SELECT
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
    AVG(SR.finalPosition) AS AverageSprintRacePosition,
    AVG(RR.finalPosition) AS AverageRegularRacePosition
  FROM Drivers D
  JOIN SprintRaces SR ON D.driverID = SR.driverID
  JOIN RaceResults RR ON D.driverID = RR.driverID
  GROUP BY D.driverID;
END; //

CREATE OR REPLACE PROCEDURE R2(IN myyear VARCHAR(100))
BEGIN
  SELECT
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
    COUNT(RR.raceID) AS TotalRaces,
    COUNT(CASE WHEN RR.FinalPosition = 1 THEN 1 ELSE NULL END) AS TotalWins,
    AVG(RR.FinalPosition) AS AverageFinishPosition,
    SUM(RR.points) AS TotalPoints
  FROM Drivers D
  JOIN RaceResults RR ON D.driverID = RR.driverID
  JOIN Races R ON RR.raceID = R.raceID
  WHERE R.year = myyear
  GROUP BY D.driverID
  ORDER BY TotalPoints DESC, AverageFinishPosition DESC, TotalWins DESC;
END; //

CREATE OR REPLACE PROCEDURE A1(IN myraceID VARCHAR(100))
BEGIN
  SELECT
    R.raceID AS Race,
    R.year AS RaceYear,
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
```

```
    RR.finalPosition,
    S.description AS IncidentDescription,
    RR.fastestLapTime
  FROM Races R
  JOIN RaceResults RR ON R.raceID = RR.raceID
  JOIN Drivers D ON RR.driverID = D.driverID
  JOIN Status S ON RR.statusID = S.statusID
  WHERE R.raceID = myraceID AND S.description LIKE '%accident%'
  ORDER BY RR.finalPosition;
END; //

CREATE OR REPLACE PROCEDURE A2(IN myyear VARCHAR(100))
BEGIN
  SELECT
    CONCAT(D.fName, ' ', D.lName) AS DriverName,
    SUM(CASE WHEN S.description LIKE '%accident%' THEN 1 ELSE 0 END) AS AccidentCount,
    COUNT(R.raceID) AS TotalRaces,
    ROUND((SUM(CASE WHEN S.description LIKE '%accident%' THEN 1 ELSE 0 END) * 100.0 / COUNT(R.raceID)), 2) AS
AccidentPercentage
  FROM Races R
  JOIN RaceResults RR ON R.raceID = RR.raceID
  JOIN Drivers D ON RR.driverID = D.driverID
  JOIN Status S ON RR.statusID = S.statusID
  WHERE R.year = myyear
  GROUP BY DriverName
  HAVING AccidentCount > 0
  ORDER BY AccidentPercentage DESC;
END; //

CREATE OR REPLACE PROCEDURE A3()
BEGIN
  SELECT
    S.description AS Status,
    COUNT(RR.statusID) AS Frequency
  FROM RaceResults RR
  JOIN Status S ON RR.statusID = S.statusID
  WHERE S.description NOT LIKE 'Finished'
  GROUP BY S.description
  ORDER BY Frequency DESC;
END; //

CREATE OR REPLACE PROCEDURE A4()
BEGIN
  SELECT
```

```
    CONCAT (D.fName, ' ', D.lName) AS DriverName,
    S.description AS IncidentType,
    COUNT(RR.statusID) AS IncidentCount
  FROM RaceResults RR
  JOIN Drivers D ON RR.driverID = D.driverID
  JOIN Status S ON RR.statusID = S.statusID
  WHERE S.description IN ('Accident', 'Engine', 'Gearbox', 'Mechanical', 'Electrical')
  GROUP BY DriverName, IncidentType
  HAVING IncidentCount > 15
  ORDER BY IncidentCount DESC;
END; //

CREATE OR REPLACE PROCEDURE A5()
BEGIN
  SELECT
    C.name AS CircuitName,
    C.location AS Location,
    C.country AS Country,
    COUNT(DISTINCT R.raceID) AS TotalRaces,
    COUNT(DISTINCT CASE WHEN S.description LIKE '%accident%' THEN R.raceID ELSE NULL END) AS AccidentRaces,
    ROUND((COUNT(DISTINCT CASE WHEN S.description LIKE '%accident%' THEN R.raceID ELSE NULL END) * 100.0 /
COUNT(DISTINCT R.raceID)), 2) AS AccidentPercentage
  FROM Circuits C
  JOIN Races R ON C.circuitID = R.circuitID
  JOIN RaceResults RR ON R.raceID = RR.raceID
  JOIN Status S ON RR.statusID = S.statusID
  GROUP BY C.circuitID
  HAVING COUNT(DISTINCT R.raceID) > 0
  ORDER BY AccidentPercentage DESC, Country DESC;
END; //

CREATE OR REPLACE PROCEDURE P1(IN mycircuitID VARCHAR(100))
BEGIN
  SELECT D.fname, D.lname, AVG(P.numPitStops) AS avgPits
    FROM (
      SELECT W.raceID, W.driverID
      FROM RaceWinnersView AS W
      WHERE W.circuitID = mycircuitID
    ) AS R, Drivers AS D, PitStops AS P
    WHERE D.driverID = R.driverID AND
      P.driverID = R.driverID AND
      P.raceID = R.raceID
    GROUP BY R.driverID
    ORDER BY AVG(P.numPitStops) ASC;
```

```
END; //

CREATE OR REPLACE PROCEDURE P2(IN myraceID VARCHAR(100))
BEGIN
   SELECT
      CONCAT (D.fName, ' ', D.lName) AS DriverName,
      R.raceID,
      RR.startPosition,
      RR.finalPosition,
      PS.numPitStops,
      (RR.startPosition - RR.finalPosition) AS PositionChange
   FROM
      PitStops PS
   JOIN
      Drivers D ON PS.driverID = D.driverID
   JOIN
      RaceResults RR ON PS.raceID = RR.raceID AND PS.driverID = RR.driverID
   JOIN
      Races R ON PS.raceID = R.raceID
   WHERE R.raceID = myraceID
   ORDER BY
      R.raceID, PositionChange DESC;
END; //

CREATE OR REPLACE PROCEDURE G1(IN myDriverName VARCHAR(100))
BEGIN
   SELECT
      C.country AS CircuitCountry,
      CONCAT(D.fName, ' ', D.lName) AS DriverName,
      COUNT(RR.raceID) AS TotalRacesInCountry,
      AVG(RR.finalPosition) AS AverageFinishPosition,
      SUM(RR.points) AS TotalPoints
   FROM
      Circuits C
   JOIN
      Races R ON C.circuitID = R.circuitID
   JOIN
      RaceResults RR ON R.raceID = RR.raceID
   JOIN
      Drivers D ON RR.driverID = D.driverID
   WHERE
      CONCAT(D.fName, ' ', D.lName) = myDriverName
   GROUP BY
      C.country, D.driverID
```

```
    ORDER BY
        C.country, AverageFinishPosition;
END; //

CREATE OR REPLACE PROCEDURE CO1()
BEGIN
    SELECT
        Co.nationality AS ConstructorNationality,
        COUNT(RR.raceID) AS TotalRaces,
        AVG(RR.finalPosition) AS AverageFinishPosition,
        SUM(RR.points) AS TotalPoints
    FROM Constructors Co
    JOIN RaceResults RR ON Co.constructorID = RR.constructorID
    JOIN Races R ON RR.raceID = R.raceID
    WHERE R.year >= YEAR(CURDATE()) - 5
    GROUP BY Co.nationality
    ORDER BY AverageFinishPosition, TotalPoints DESC;
END; //

CREATE OR REPLACE PROCEDURE CO2()
BEGIN
    SELECT
        Co.name AS ConstructorName,
        COUNT(DISTINCT RR.raceID) AS TotalRaces,
        COUNT(DISTINCT CASE WHEN RR.finalPosition = 1 THEN RR.raceID ELSE NULL END) AS FirstPlaceFinishes,
        (COUNT(DISTINCT CASE WHEN RR.finalPosition = 1 THEN RR.raceID ELSE NULL END) * 100.0 / COUNT(DISTINCT
RR.raceID)) AS FirstPlacePercentage
    FROM Constructors Co
    JOIN RaceResults RR ON Co.constructorID = RR.constructorID
    JOIN Races R ON RR.raceID = R.raceID
    WHERE R.year >= YEAR(CURDATE()) - 5
    GROUP BY Co.name
    ORDER BY FirstPlaceFinishes DESC;
END; //
```