****

Lebanese American University

School of Arts and Sciences

Department of Computer Science and Mathematics

Instructor: Khaleel Mershad

**Formula 1 Database Project**

****

Done By:

Mohammad Shouaib

Mohammad Mehdi Soueid

Jihad Tiba

**Introduction:**

In the fast-paced realm of Formula 1, a reliable database is essential for streamlined operations. Serving as a centralized hub, it houses critical data like race history, car metrics, and driver stats. This information fuels real-time decision-making during races, enabling engineers to adjust strategies on the fly based on telemetry data. Beyond the track, teams use the database for long-term planning, analyzing trends and refining strategies over multiple seasons. Moreover, the database plays a pivotal role in technological advancements, facilitating simulations and predictive modeling to fine-tune car setups before races. In essence, the Formula 1 database is the linchpin of the entire system, supporting both immediate race decisions and long-term performance optimization.

**Description:**

In the fast-paced realm of Formula 1, a reliable database is essential for streamlined operations. Serving as a centralized hub, it houses critical data like race history, car metrics, and driver stats. This information fuels real-time decision-making during races, enabling engineers to adjust strategies on the fly based on telemetry data. Beyond the track, teams use the database for long-term planning, analyzing trends and refining strategies over multiple seasons. Moreover, the database plays a pivotal role in technological advancements, facilitating simulations and predictive modeling to fine-tune car setups before races. In essence, the Formula 1 database is the linchpin of the entire system, supporting both immediate race decisions and long-term performance optimization.

The database should include entities for different personnel within the racing team, such as drivers, team principals, mechanics, and engineers. Key attributes to be recorded for all individuals are their respective ID’s(ID), name, address, date of birth (DOB), and nationality. Each specific role also has unique attributes. For example, drivers have a driver ID, contract start and end dates, and their position within the team, where each driver drives a car and has a specific team. Each Race engineer communicates with his driver during the race and has a specific job and ID, while each engineer fabricates car parts for his car and works in a department where each one has their profession and ID. Mechanics are responsible for fixing cars and are possessed with an ID and a specific level of experience. Team principles are responsible for the team as a whole.

Team principals represent the team's leaders and managers, their responsibilities extend beyond just managing the on-track performance and involve a combination of strategic, managerial, and leadership duties with attributes such as ID, contract start and end dates. They manage the team as a whole unit and oversee the team's ID, name, and headquarters. Each team principal is linked to the cars they manage.

Each race has attributes such as race ID, race name, circuit name, location, date, and weather conditions. In formula 1, drivers earn points based on their finishing positions in every race where drivers and teams experience shifts in position due to every result recorded. Thus standing include attributes such as points, position in the table, race results, and previous results, associated with team IDs. Each team has its standings related to the racing teams they are part of.

The database should also include departments representing various functional units within the team, each department has specialized functions and responsibilities, contributing to the overall performance of the team. They contain attributes such as department ID, type, budget, and the number of employees in each department. Engineers analyze and interpret information to identify areas for improvement and understand the car's behavior under different conditions. Their role is to fabricate different car parts specifying their ID, version, type (e.g., engine, chassis, front wing, floor, side pods), and health status. Car parts can get damaged in accidents and are assigned a unique accident ID. Mechanics play a pivotal role in ensuring the race cars are finely tuned, reliable, and ready for the intense demands of racing are mainly responsible for repairing the racing cars, identified by their ID and level of expertise.

During races, the driver is solely responsible for piloting their assigned car. They must navigate the track, make split-second decisions, and compete against other drivers to achieve the best possible result for themselves and the team. Each car includes attributes like car ID, car number, car name, and the year. Sponsors are linked to specific cars and have attributes like sponsor ID, sponsor name, contract start and end dates, and investment amounts. Race engineers serve as the primary point of contact between the driver and the technical team, playing a pivotal role in strategizing, decision-making, and ensuring that the car performs at its best on the track. They are identified by name and ID.

Pit stops are a dynamic and integral aspect of Formula 1 races, serving as a strategic and technical interlude where teams can enhance their competitive position through quick and efficient adjustments to the car, the art of pit stops combines speed, precision, and strategic decision-making to contribute significantly to a team's overall race performance. Pit stops include attributes such as duration, workers, time and tires including new and old tires.

**ER-Model:**

The ER-Model is on a different document because the screenshot was not clear. Thank you.

**Relational Model:**

**Entities:**

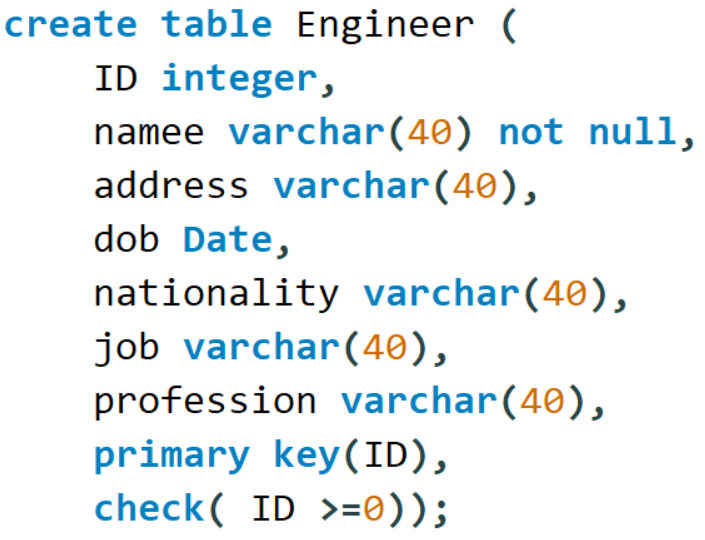
* **Engineer** ( ID , name , address , dob , nationality, job, profession)
* **Mechanic** ( ID , name , address , dob , nationality , level)
* **Car parts** ( parts-ID , version , type , health)
* **Car**( car-ID , CarVersion , year)
* **Standings**(standings-ID, #race\_ID, #driver\_ID,points , raceResult)
* **Race Engineer**( ID ,#Driver-ID(not null) , name , address , dob , nationality, job)
* **Drive**r(ID ,#team-ID(not null) ,# car-ID , name , address , dob , nationality , contract-End , Contract-start , position)
* **Team** ( Team-ID ,Team\_Name,# Team-principal-ID (not null),headquarters , Engine-supplier , main-sponsor, total\_Points)
* **Team Principa**l(Team-Principal-ID , #carID , name , address , dob , nationality , contract-end , contract-start )
* **Sponso**r(Sponsor\_ID, sponsorName , contract\_Start, investmentAmount, Contract\_End)
* **Race**(raceID, date, weather, raceName, circuitName, Region, City, Country)
* **Accident**( #carID, Accident\_ID, date)
* **Department**(building-number , #teamID(not null) , type , budget , nbOfEmployees)
* **PitStops**(#raceID , time, duration, type\_of\_old, type\_of\_new)

**Relations:**

* **Works-IN** ( #Building-number , #Engineer-ID)
* **Participates**(#raceID, #carID)
* **sponsors**(#Sponsor\_ID, #carID)
* **fixes**(#Mechanic-ID, #carID)
* **damages**(#parts-ID, #carID, #Accident\_ID)
* **fabricates(#**parts-ID,#Engineer-ID, #building-number)
* **monitors**(#race-ID , car-ID)

**Data Implementation:**

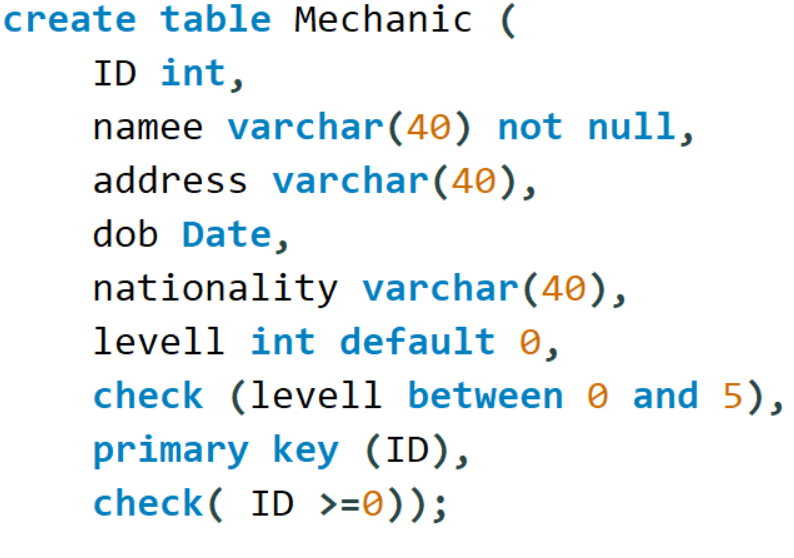
**Table Engineer:**



Check: Every engineers ID should be a nonnegative integer.

Not Null: Every engineer in the table should have a name.

**Table Mechanic:**

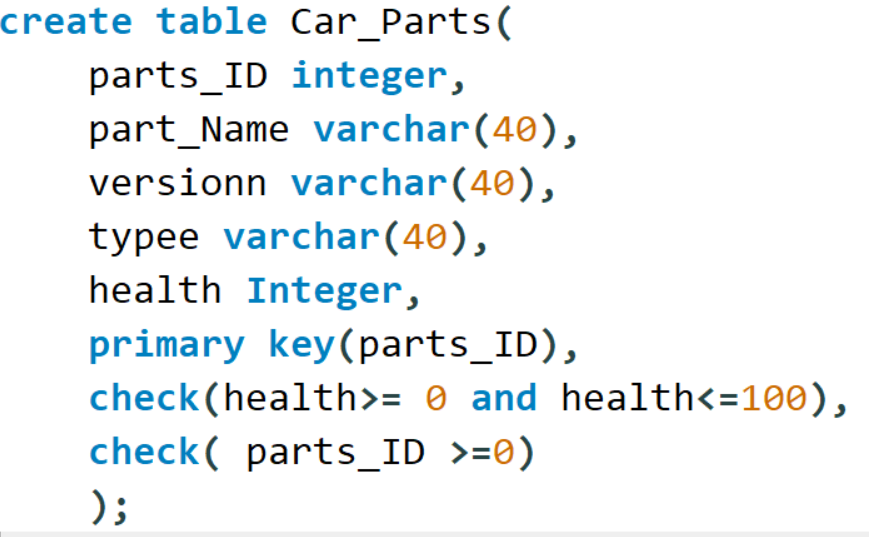


Check and Default: The highest level a mechanic can reach is 5 and the lowest level is 0 which this mechanic has by default if he does not have a professional level before.

Check: Every mechanics ID should be a nonnegative integer.

Not Null: Every Mechanic in the table should have a name.

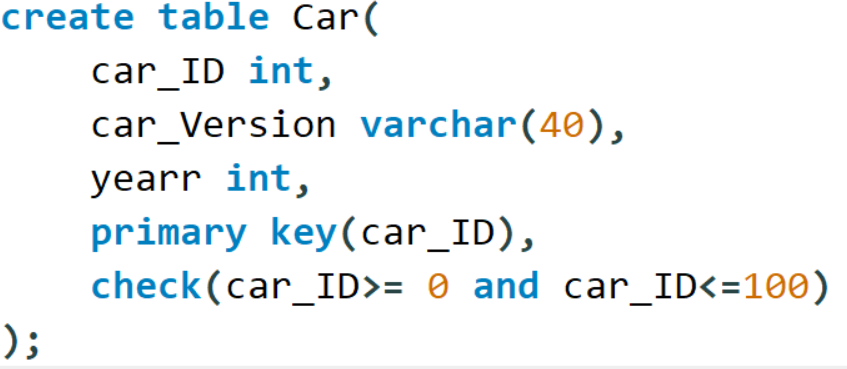
**Table Car\_Parts:**



Check: The health of a car part is a percentage between 0 and 100. When a car is damaged, the health of the car part decreases.

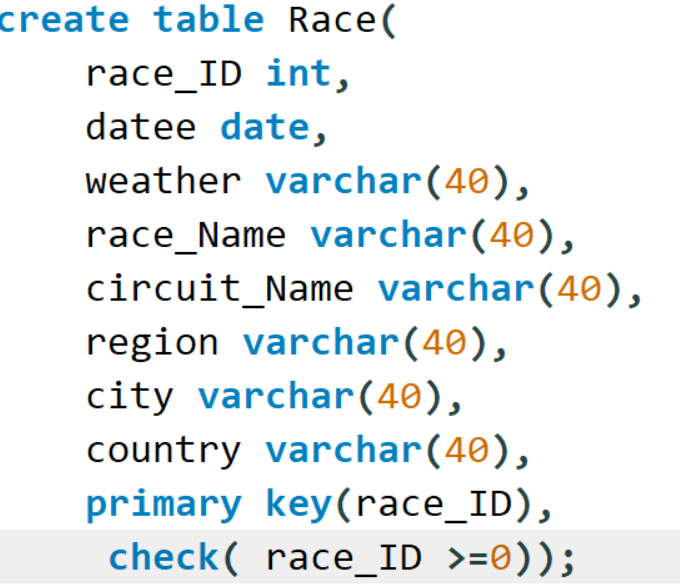
Check: Every car parts ID should be a nonnegative integer.

**Table Car:**



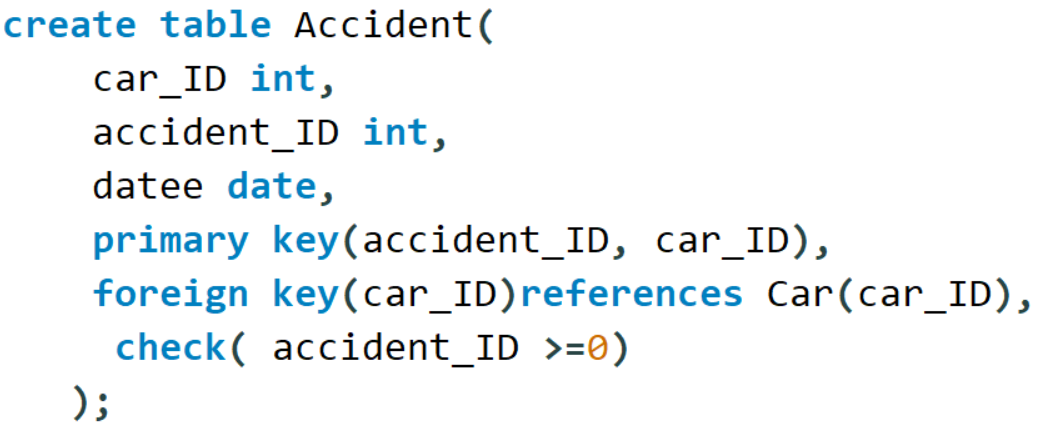
Check: Every car ID should be between 0 and 100 based on the formula1 system rules (This is going to be the car number on the car during races).

**Table Race:**



Check: Every race ID should be a nonnegative integer.

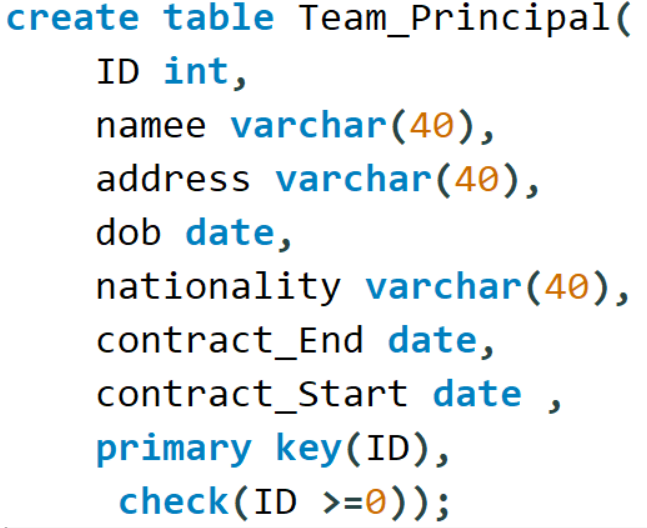
**Table Accident:**

****

Check: Every accident ID should be a nonnegative integer.

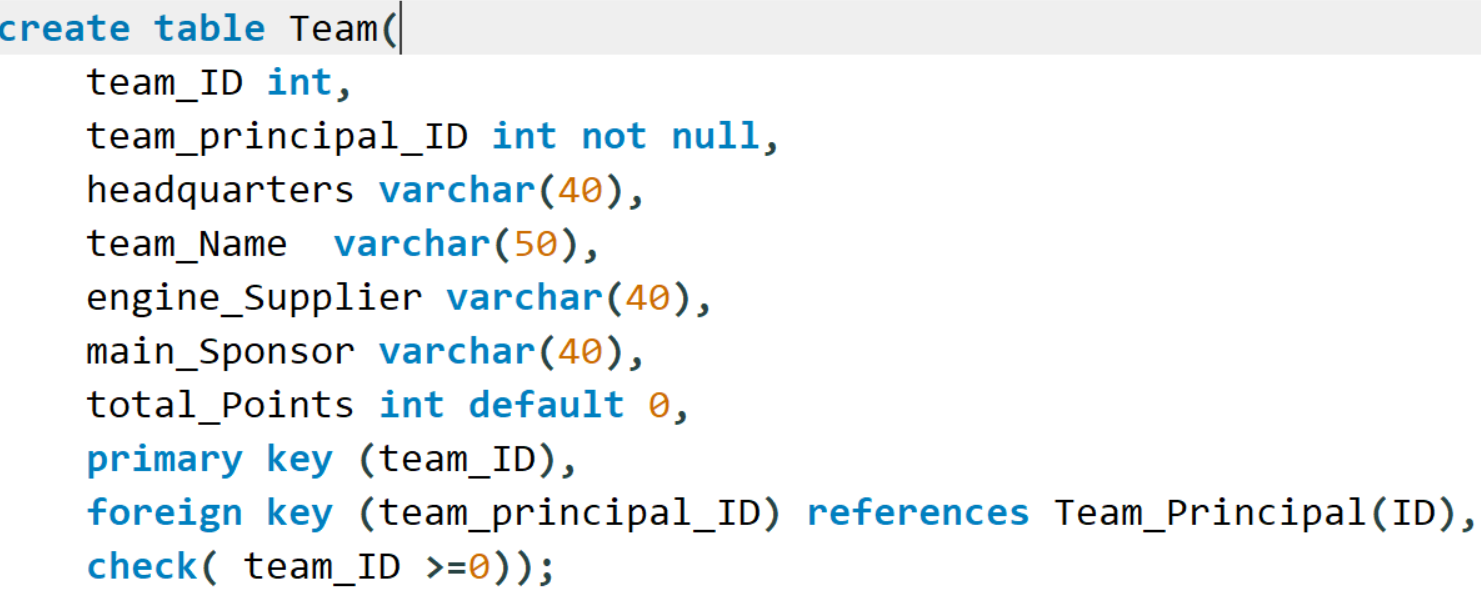
Foreign Key: car\_ID is a foreign key of the primary key in Car. This way we can identify the car that made an accident.

**Table Team Principal:**



Check: Every team principal’s ID should be a nonnegative integer.

**Table Team:**

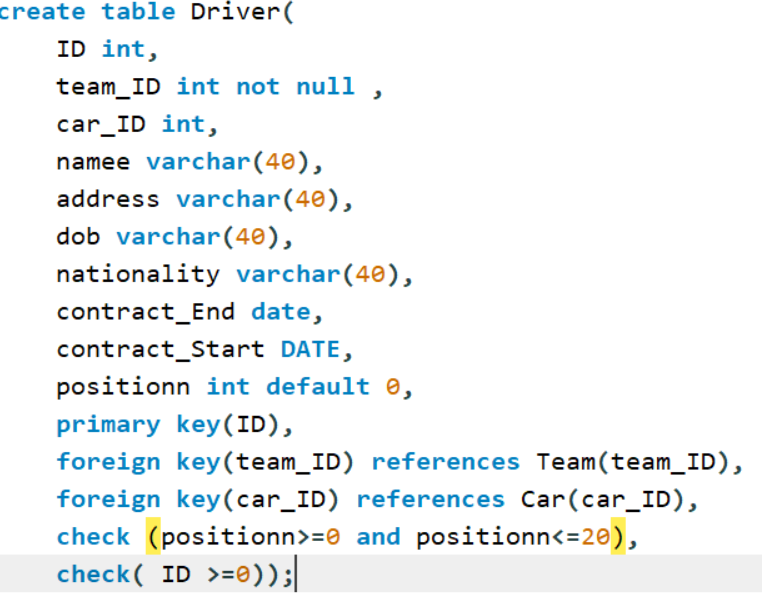


Check: Every team’s ID should be a nonnegative integer.

Not Null: Team principal’s ID should not be null because every team should have a team principal.

Foreign Key: Since every team has a team principal that should be identified by his ID, the team\_principal\_ID is a foreign key of the primary key in Team\_Principal.

**Table Driver:**



Not Null: Team ID should not be null so that we can identify the team and there is no Driver if there is no team.

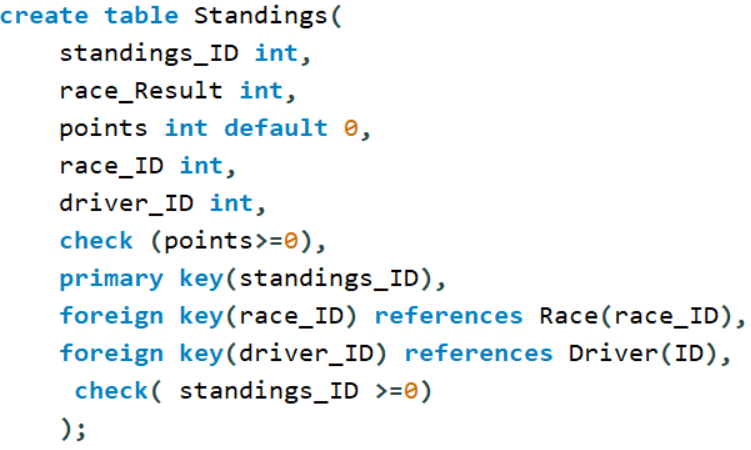
Foreign Key: team\_ID is a foreign key for the primary key in Team, this way we can identify to which team does this driver belong.

Foreign Key: car\_ID is a foreign key to the primary key in Car, this way we can identify which car does this driver drive.

Check: ID of the driver is nonnegative.

Check: Position of the driver is between 0 and 20 where 0 is disqualified, and the other 20 positions are the number of drivers in the race

**Table Standings:**



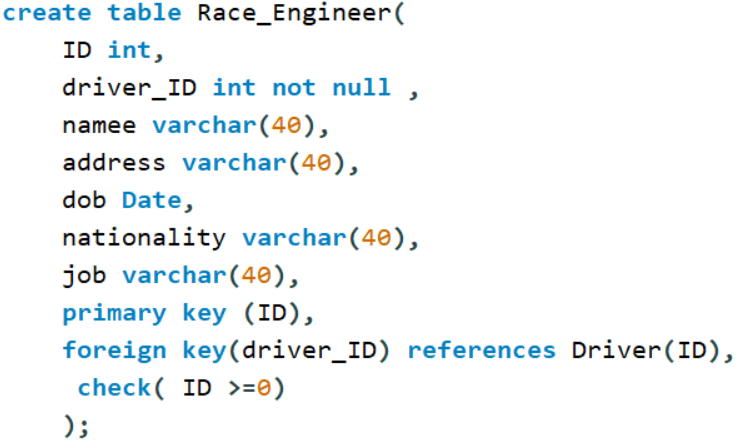
Default: By default, the point in the standings are zero at first if no data was entered.

Check: Points of the standings should be always greater than or equal to zero. Since rhe minimum number of points is zero.

Foreign Key: race\_ID is a foreign key for the primary key in Race. Since every standing is going to be affected by a specific race.

Foreign Key: driver\_ID is a foreign key for primary key in Driver. Every standings has a list of its drivers participated in it.

**Table Race\_Engineer:**

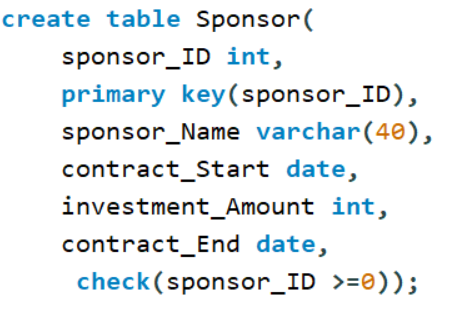


Not Null: Driver’s ID should not be null since every race engineer monitors a certain driver.

Foreign Key: driver\_ID is a foreign key for the primary key in Driver.

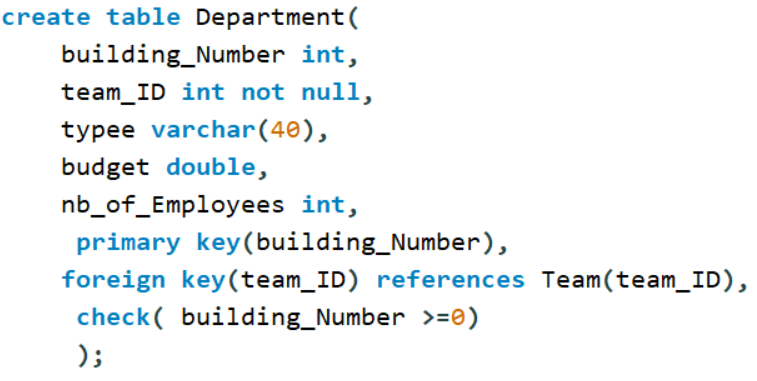
Check: ID is nonnegative.

**Table Sponsor:**



Check: Sponsor ID is nonnegative.

**Table Department:**

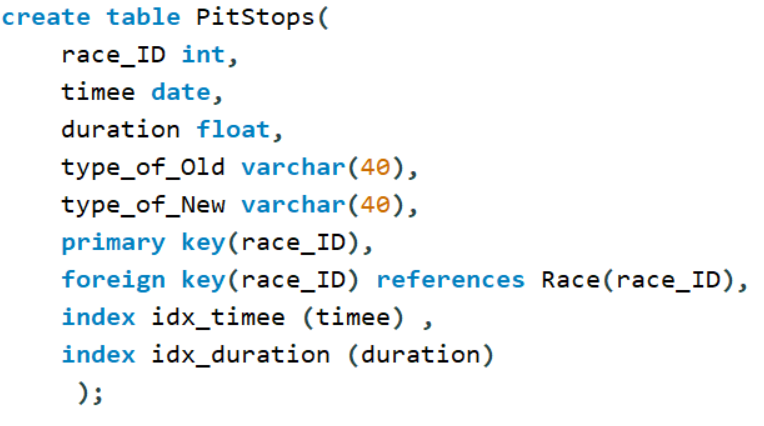


Not Null: Every department is created for a specific team (refers for a specific team) so there is no department without a team. So it should be not null.

Foreign Key: team\_ID is a foreign key for the primary key in Team because Every department is created for a specific team (refers for a specific team) so there is no department without a team.

Check: Building number shouldn’t be negative.

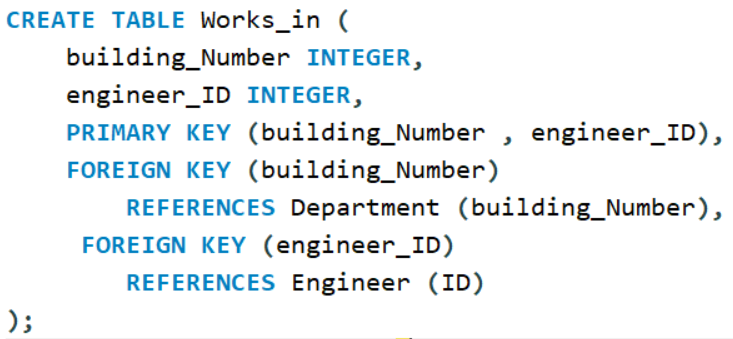
**Table Pitstops:**



Foreign key: race\_ID is a foreign key for the primary key in Race. Since there is no pitstop without a race so a pitstop is identified by the race.

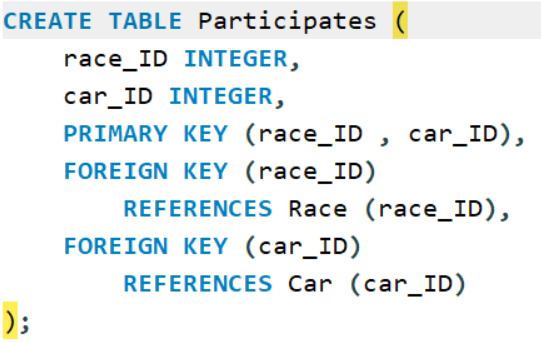
We used the index to remove the errors.

**Table Works\_In:**



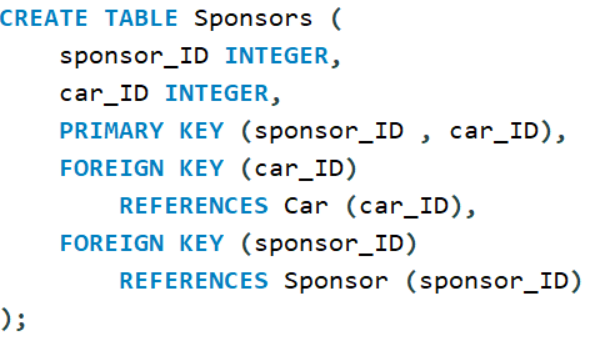
Foreign Key: building\_Number and engineer\_ID are foreign keys for the primary keys in Department and Engineer respectively. Since to identify which engineer works in which department, we need the primary key of both tables.

**Table Participates:**



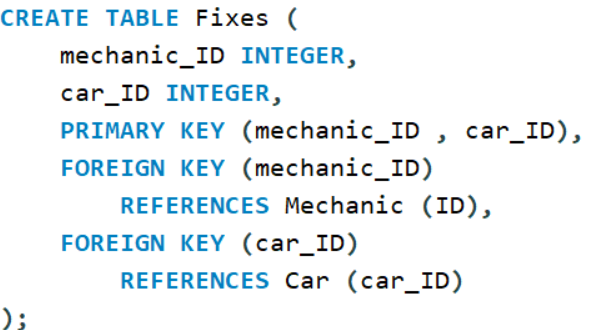
Foreign Key: race\_ID and car\_ID for both primary keys in Race and Car tables respectively. Since each car participates in a race and both attributes should be identified.

**Table Sponsors:**



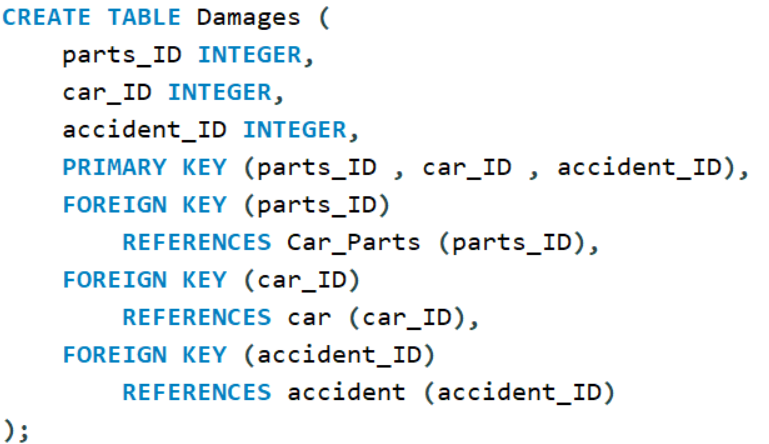
Foreign Key: car\_ID and sponsor\_ID are foreign keys in both tables Car and Sponsor respectively so that we can identify which sponsor sponsors which car.

**Table Fixes:**



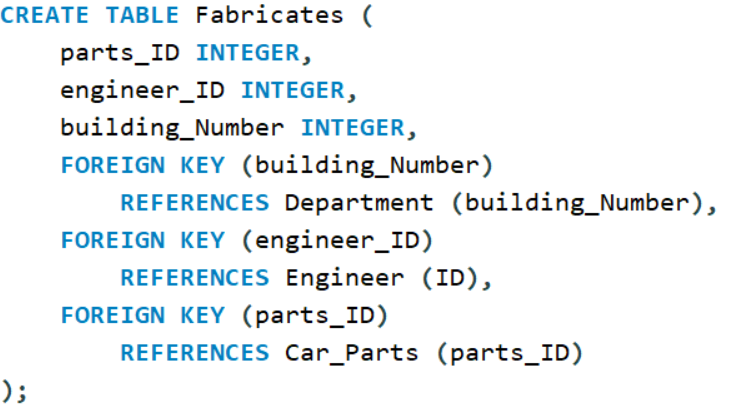
Foreign keys: mechanic\_ID and car\_ID are both foreign keys for the primary keys in Mechanic and Car respectively. This way we can identify which mechanic fixes a specific car

**Table Damages:**



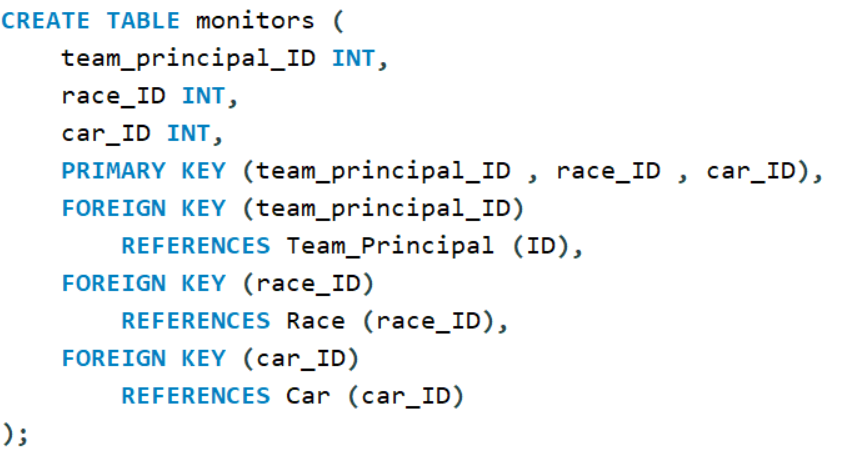
Foreign Keys: parts\_ID, car\_ID, and accident\_ID are foreign keys for the primary keys in Car\_Parts, Car, and Accident respectively. This way we can identify which car part belongs to which car and the accident that damaged this car.

**Table Fabricates:**



Foreign Keys: Building\_Number, engineer\_ID, and parts\_ID are foreign keys for the primary keys in Department, Engineer, and Car\_Parts respectively. This way we can identify which engineer fabricates which part and for what department.

**Table Monitors:**

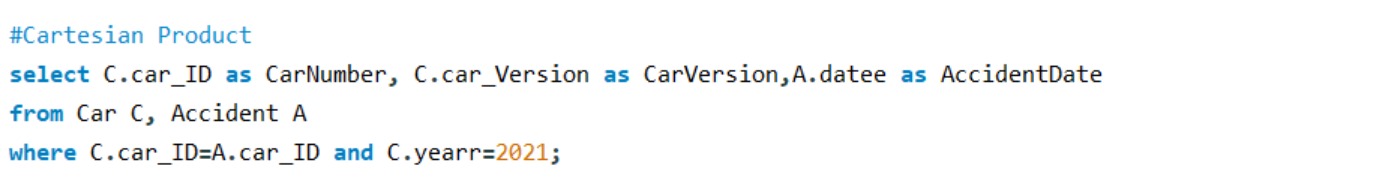


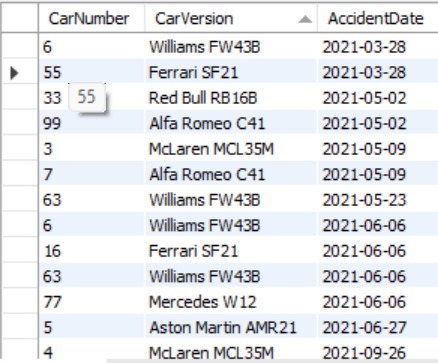
Foreign Keys are team\_Principal, race\_ID and car\_ID for the primary keys in Team\_Principal, Race, and Car respectively. This way we can find which team principal monitors what car and in which race.

**Basic Queries:**

**Cartesian Product:**

Return the car number and version of the cars that have done an accident in year 2021.



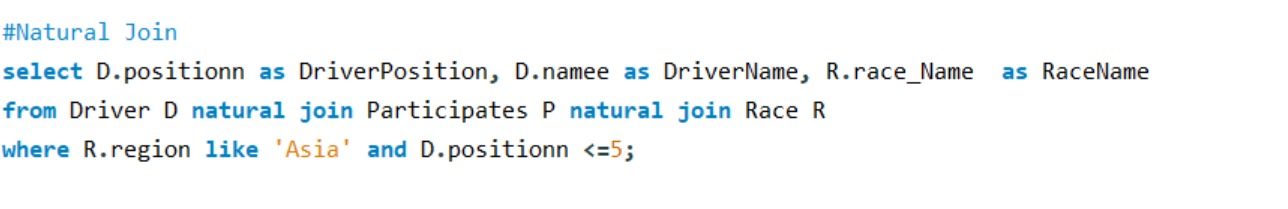


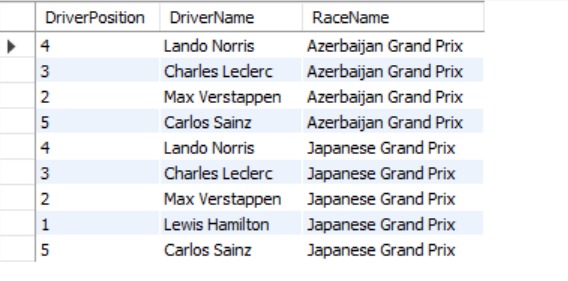
Explanation:

Retrieve the car ID (as CarNumber), car version (as CarVersion), and the date of accidents (as AccidentDate) for cars with a production year of 2021. The data is obtained through the "Car" table (abbreviated as C) and the "Accident" table (abbreviated as A), where the car ID from both tables matches.

**Natural Join:**

Return the Race name, driver’s name and position if those drivers participated in races that happened in Asia where they got top 5.



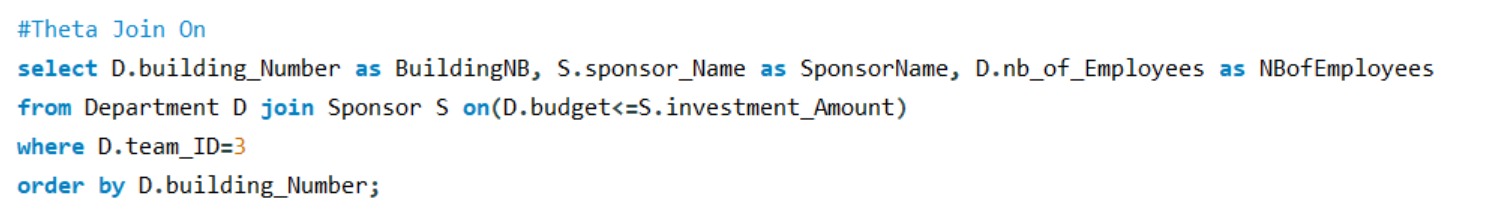


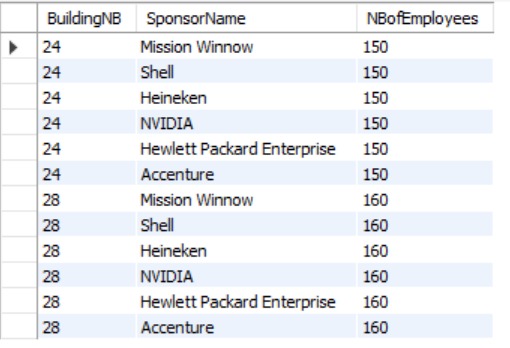
Explanation:

Retrieve the driver's position (as DriverPosition), name (as DriverName), and the race name (as RaceName) for drivers who participated in races held in the 'Asia' region and achieved a position of 5 or lower. The data is obtained through the "Driver" table (abbreviated as D), the "Participates" table (abbreviated as P), and the "Race" table (abbreviated as R), with natural joins between them.

**Theta Join On:**

Retrieve the building numbers, number of employees and sponsor names for departments where the team ID is 3 (Ferrari) and the budget of the department is less than the investment amount of the associated sponsor.



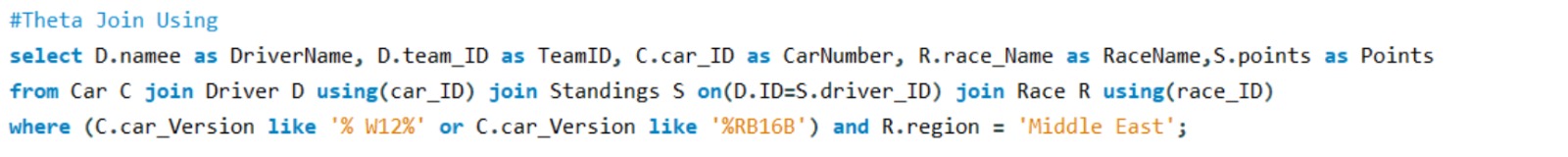


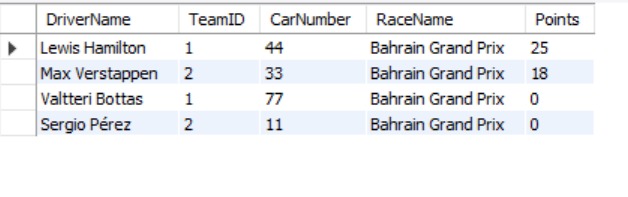
Explanation:

Retrieve the building number (as BuildingNB), sponsor name (as SponsorName), and the number of employees (as NBofEmployees) for departments with a team ID of 3. The data is obtained through a theta join between the "Department" table (abbreviated as D) and the "Sponsor" table (abbreviated as S), where the budget of the department is less than or equal to the investment amount of the associated sponsor. The results are ordered by building number in ascending order.

**Theta Join Using**

Return the driver name, car number, team ID, points of the racer that has a car version W12 or RB16B and has raced in the middle east.





Explanation:

Retrieve the driver name (as DriverName), team ID (as TeamID), car number (as CarNumber), race name (as RaceName), and points (as Points) for drivers whose cars have versions containing 'W12' or 'RB16B' and who participated in races held in the 'Middle East' region. The data is obtained through theta joins using the "Car," "Driver," "Standings," and "Race" tables based on matching car IDs, driver IDs, and race IDs.

**Self-Join:**

Return the ID of the Engine V6 with the max health



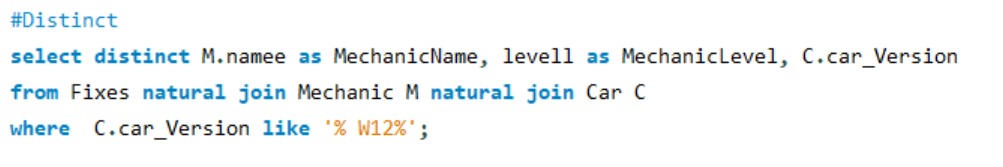
Error: MySQL doesn’t support except

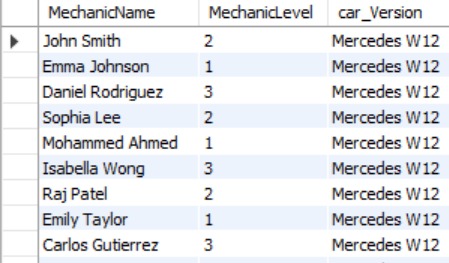
Explanation:

Retrieve the parts IDs from the "Car\_Parts" table for parts with the name 'engine' and version 'V6,' excluding those parts with lower health than any other part with the same version. The data is obtained through a self-join on the "Car\_Parts" table, comparing health values, and ensuring that the part names are 'engine' and the versions are 'V6.' The EXCEPT keyword is used to exclude parts that meet the specified conditions in the second part of the query.

**Distinct**

Retrieve the name and level of the mechanics who have worked on fixing cars with version 'W12'.



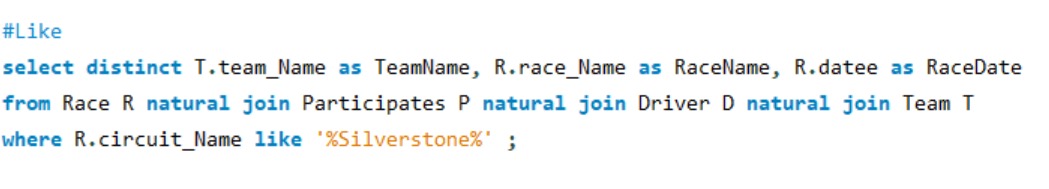


Explanation:

Retrieve distinct mechanic names (as MechanicName), mechanic levels (as MechanicLevel), and car versions for mechanics who have worked on fixing cars with a version containing 'W12.' The data is obtained through natural joins between the "Fixes," "Mechanic" (abbreviated as M), and "Car" (abbreviated as C) tables, where the car version is 'W12.' The DISTINCT keyword is used to ensure unique combinations of mechanic name, mechanic level, and car version in the results.

**Like**

Retrieve the race name, team ID, and date for the races that where held at the Silverstone circuit.



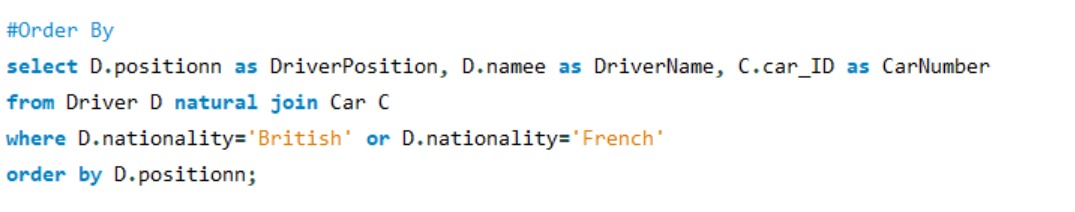


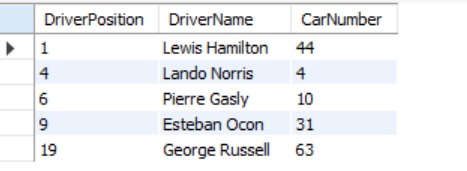
Explanation:

Retrieve distinct team names (as TeamName), race names (as RaceName), and race dates (as RaceDate) for races held at a circuit with a name containing 'Silverstone.' The data is obtained through natural joins between the "Race" (abbreviated as R), "Participates" (abbreviated as P), "Driver" (abbreviated as D), and "Team" (abbreviated as T) tables. The LIKE keyword is used to filter races based on the circuit name containing 'Silverstone.' The DISTINCT keyword ensures unique combinations of team name, race name, and race date in the results.

**Order By**

Retrieve information -in increasing order based on position- about British and French drivers who participated in the 2021 season, including their position, name, car number, and team ID.





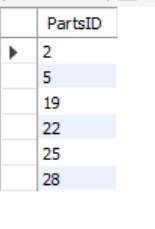
Explanation:

Retrieve the driver's position (as DriverPosition), name (as DriverName), and car ID (as CarNumber) for British or French drivers. The data is obtained through a natural join between the "Driver" (abbreviated as D) and "Car" (abbreviated as C) tables, filtered based on the nationality criteria. The results are then ordered by the driver's position in ascending order.

**Union**

Retrieve all the parts IDs that are RB-Turbo or ES-Superlight





Explanation:

Combine the parts IDs (as PartsID) from the "Car\_Parts" table for parts with the name 'Chassis' and version 'V5,' and parts with the name 'Engine' and version 'V5' using the UNION operator.

**Except**

Return the sponsor ID with maximum contract length.



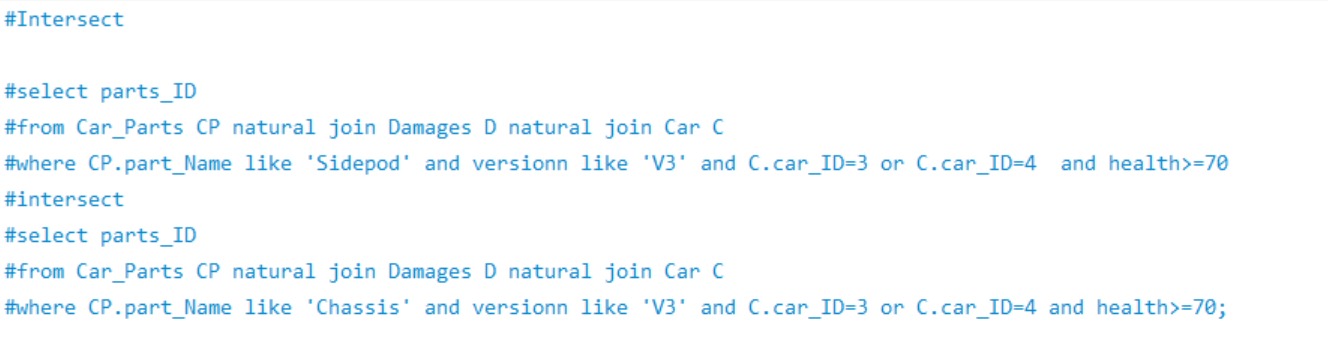
Error: MySQL doesn’t support except

Explanation:

Retrieve the distinct Sponsor IDs from the "Sponsor" table where the contract duration is greater than or equal to the contract duration of any other sponsor. The EXCEPT keyword is used to exclude Sponsor IDs that meet the condition specified in the second part of the query, where sponsors are self-joined based on contract durations. The WHERE clause in the first part ensures that all rows from the "Sponsor" table are considered.

**Intersect**

Retrieve the car parts IDs for the McLaren car of number 3, 4 that got damaged by an accident and are of a health>=70 where the part is a V3 chassis or a V3 sidepod.



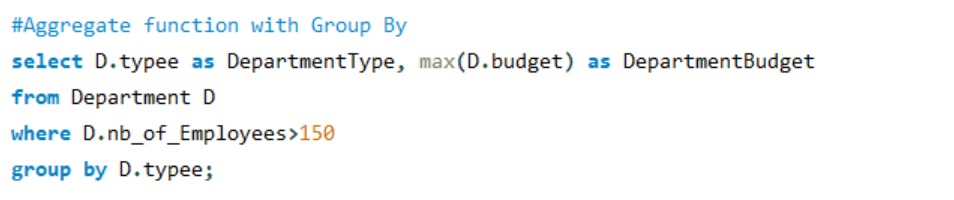
Error: MySQL doesn’t support Intersect

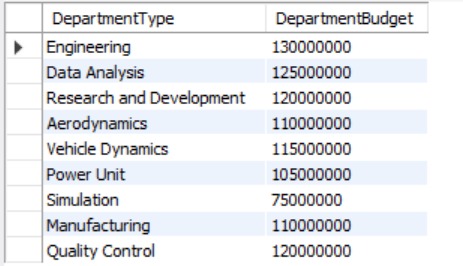
Explanation:

Retrieve the common parts IDs from the "Car\_Parts," "Damages," and "Car" tables where the part name is 'Sidepod,' version is 'V3,' the car ID is either 3 or 4, and the health is greater than or equal to 70. The INTERSECT operator is used to find the intersection of two sets of parts IDs, where the second set is based on similar criteria but with the part name 'Chassis.'

**Aggregate function without Group By**

Select the name and dob of the youngest and oldest driver that got a podium in the 2021 season.

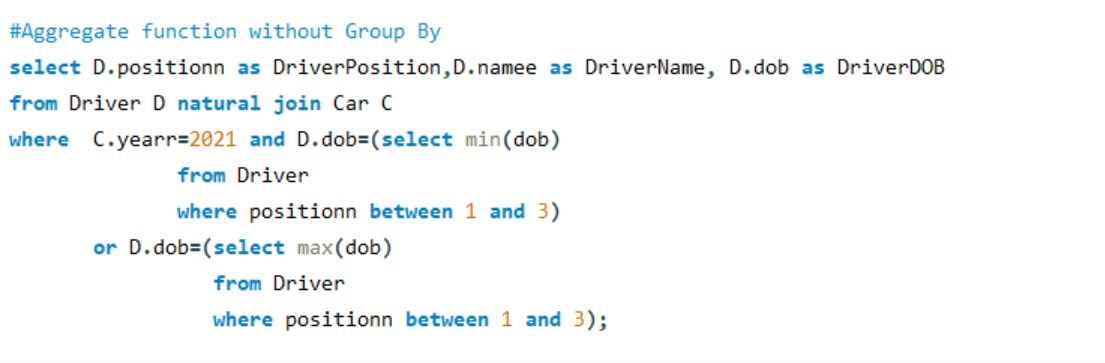


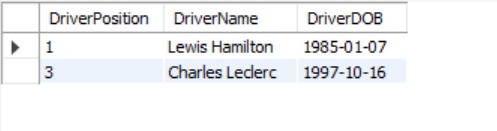


Explanation: Retrieve the driver's position (as DriverPosition), name (as DriverName), and date of birth (as DriverDOB) for drivers whose cars were active in the year 2021. The data is obtained through a natural join between the "Driver" (abbreviated as D) and "Car" (abbreviated as C) tables. The results are filtered based on the condition that the driver's date of birth is either the minimum or maximum date of birth among drivers with positions between 1 and 3. Note that this query uses aggregate functions without a GROUP BY clause.

**Aggregate function with Group By**

For each type of department with number of employees greater than 150, return the maximum budget for this department



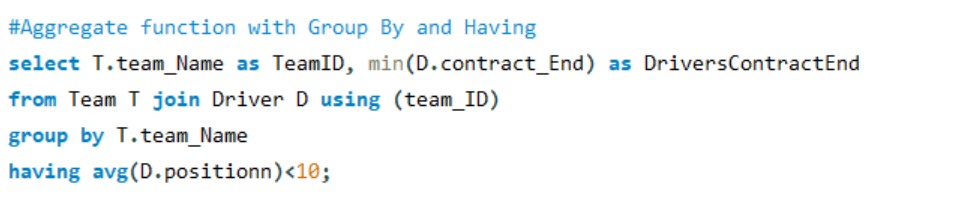


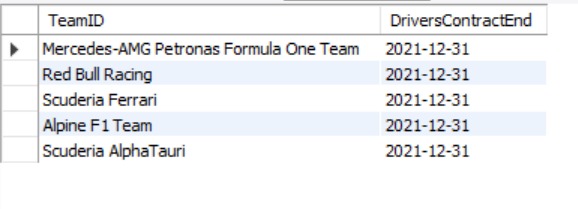
Explanation:

Retrieve the department type (as DepartmentType) and the maximum budget (as DepartmentBudget) for departments with more than 150 employees. The data is obtained from the "Department" table (abbreviated as D). The results are grouped by department type, and the maximum budget for each department type is calculated using the MAX aggregate function.

**Aggregate function with Group By and Having**

Retrieve the minimum contract end date in each team where the team’s drivers achieved an average position of top 10





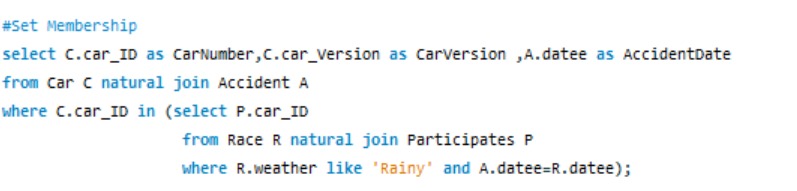
Explanation:

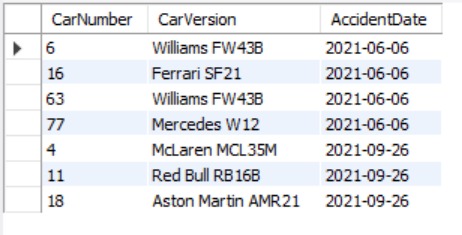
Retrieve the team name (as TeamID) and the minimum contract end date (as DriversContractEnd) for teams that have an average driver position of less than 10. The data is obtained through a join between the "Team" table (abbreviated as T) and the "Driver" table (abbreviated as D) using the team ID. The results are grouped by team name, and the HAVING clause is used to filter teams based on the condition that the average driver position is less than 10. The MIN aggregate function is applied to find the minimum contract end date for each team.

**Advanced Queries**:

**Set Membership:**

Select the car number of the car that has done an accident in a race with rainy weather.





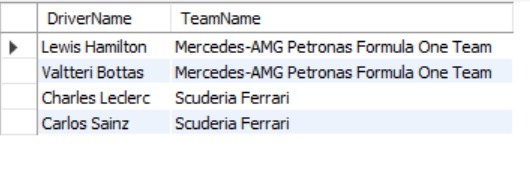
Explanation:

Retrieve the car ID (as CarNumber), car version (as CarVersion), and accident date (as AccidentDate) for cars involved in accidents that occurred on the same date as races with rainy weather conditions. The data is obtained through a natural join between the "Car" (abbreviated as C) and "Accident" (abbreviated as A) tables. The results are filtered based on the condition that the car ID is present in the set of car IDs associated with races on rainy days. The set of car IDs is obtained through a subquery that involves a natural join between the "Race" (abbreviated as R) and "Participates" (abbreviated as P) tables.

**Set Comparison and 2 or more nesting levels:**

Select the name of the team that has a driver with the highest sponsor investment amount.



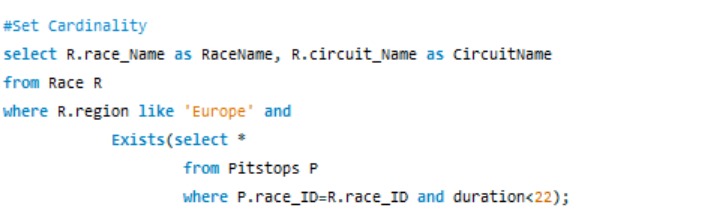


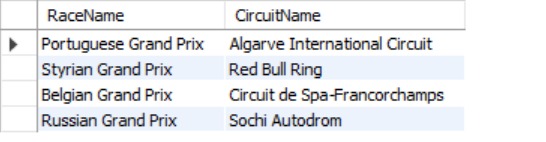
Explanation:

Retrieve the driver name (as DriverName) and team name (as TeamName) for drivers belonging to teams that have sponsors with the highest investment amount among all sponsors. The data is obtained through a natural join between the "Team" (abbreviated as T) and "Driver" (abbreviated as D) tables. The results are filtered based on the condition that the driver's ID is in the set of driver IDs associated with sponsors with the highest investment amount. The set of sponsor IDs with the highest investment amount is obtained through a subquery with a set comparison using the ALL keyword. The nested subquery involves a natural join between the "Driver" and "Sponsors" tables.

**Set Cardinality:**

Select the name and circuit of the races that happened in Europe and had a fastest pitstop duration less than 22 seconds



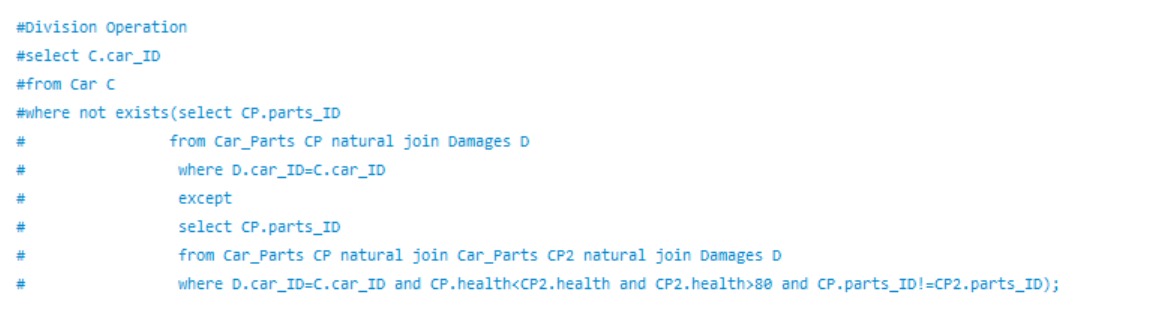


Explanation:

Retrieve the race name (as RaceName) and circuit name (as CircuitName) for races in the 'Europe' region where there exists at least one pitstop with a duration less than 22 seconds. The data is obtained from the "Race" table (abbreviated as R), and the results are filtered based on the conditions specified for the region and the existence of pitstops with the specified duration. The EXISTS keyword is used to check the set cardinality, ensuring that at least one pitstop meets the criteria for each race.

**Division Operation:**

Return the car number of the car that has done an accident and its max car part health is greater than 80.



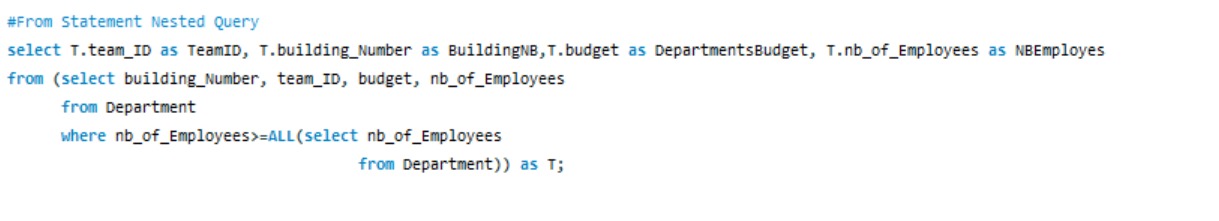
Error: MySQL doesn’t support except

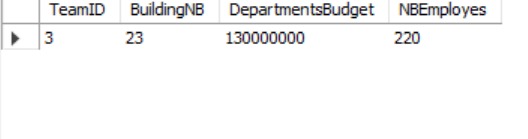
Explanation:

Retrieve the car IDs from the "Car" table for cars where all parts have suffered damages but no part has a higher health than any other part with a health greater than 80. The data is obtained through a division operation using the EXCEPT keyword within a NOT EXISTS subquery. The subquery compares the health of parts for each car, ensuring that no part has higher health than any other part with health greater than 80.

**From Statement Nested Query:**

Return some information about the department with maximum number of employees. Where the information includes team ID, building number, number of employees, and departments budget.



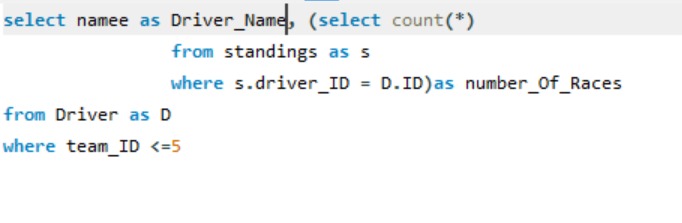


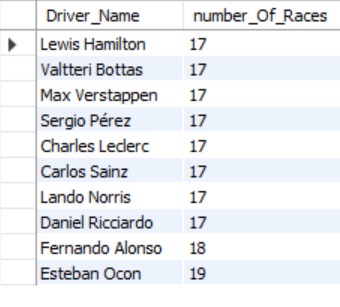
Explanation:

Retrieve the team ID (as TeamID), building number (as BuildingNB), budget (as DepartmentsBudget), and number of employees (as NBEmployes) for teams associated with the department that has the highest number of employees. The data is obtained using a nested query in the FROM statement, where the inner query selects the building number, team ID, budget, and number of employees from the "Department" table, filtering based on the condition that the number of employees is greater than or equal to all numbers of employees in the same table. The outer query then selects the desired columns from the result of the nested query.

**Select Statement Nested Query:**

Number of races each driver has participated in.



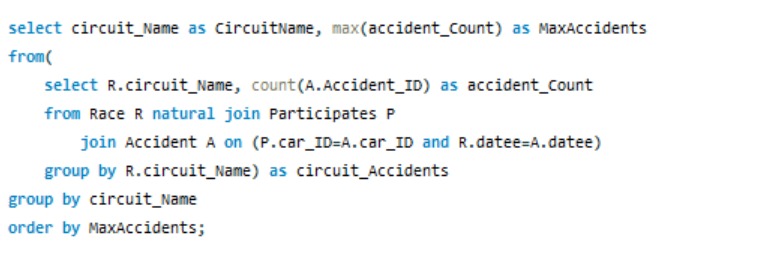


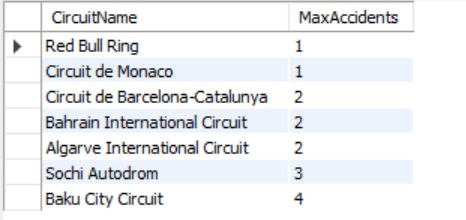
Explanation:

Retrieve the driver name (as Driver\_Name) and the count of races in which each driver participated (as number\_Of\_Races), considering only drivers associated with teams where the team ID is less than or equal to 5. The data is obtained from the "Driver" table (abbreviated as D) and the subquery in the SELECT statement, which counts the occurrences in the "Standings" table (abbreviated as S) where the driver ID matches the driver's ID from the outer query.

Extra Nice Query:

Select the maximum number of accidents that happened per circuit.



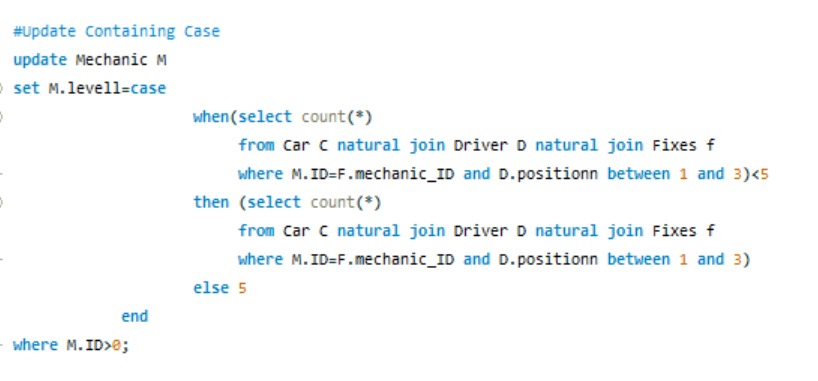


Explanation:

Retrieve the circuit name (as CircuitName) and the maximum count of accidents (as MaxAccidents) that occurred in races. The data is obtained through a nested subquery, where the inner query joins the "Race" (abbreviated as R), "Participates" (abbreviated as P), and "Accident" (abbreviated as A) tables to count the number of accidents for each circuit. The results are then grouped by circuit name in the outer query, and the maximum accident count for each circuit is determined. Finally, the results are ordered by the maximum accidents in ascending order.

**Update Containing Case:**

Update the level of the mechanic according to the number of podiums the car he fixes got. His level should be 5 if the car he fixes got 5 or more podiums. And if its less than 5, update the level to the number of podiums his car got

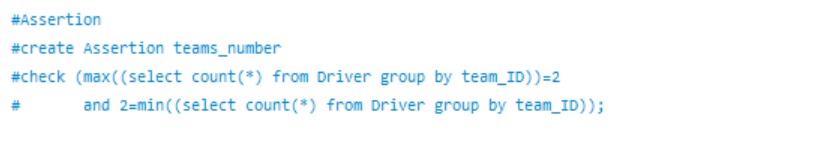


Explanation:

Update the "Mechanic" table by setting the level (levell) for mechanics based on a conditional case. If the count of fixes performed by a mechanic on cars driven by drivers with positions between 1 and 3 is less than 5, then set the level to that count; otherwise, set the level to 5. This update is applied to mechanics with an ID greater than 0.

**Assertion:**

Make sure that each team has only 2 drivers.



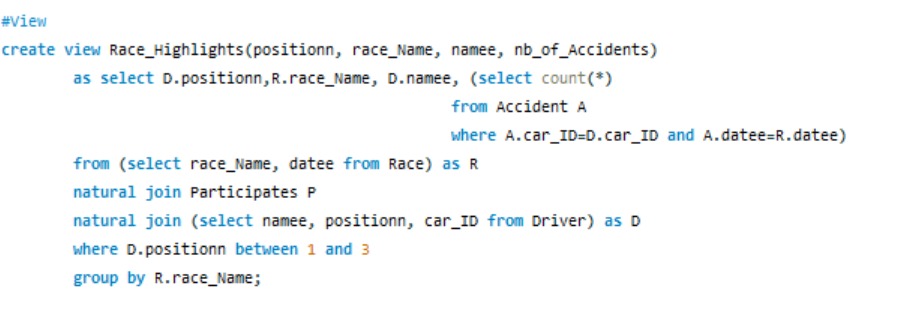
Error: MySQL doesn’t support Insertion

Explanation:

Create an assertion named "teams\_number" that checks whether the maximum count of drivers per team is 2, and the minimum count of drivers per team is also 2. This assertion ensures that every team has exactly 2 drivers.

**View:**

Return a view of the race highlights for each race such as (position, race ID, drivers name, and number of accidents) where the driver in this view should have gotten a podium.

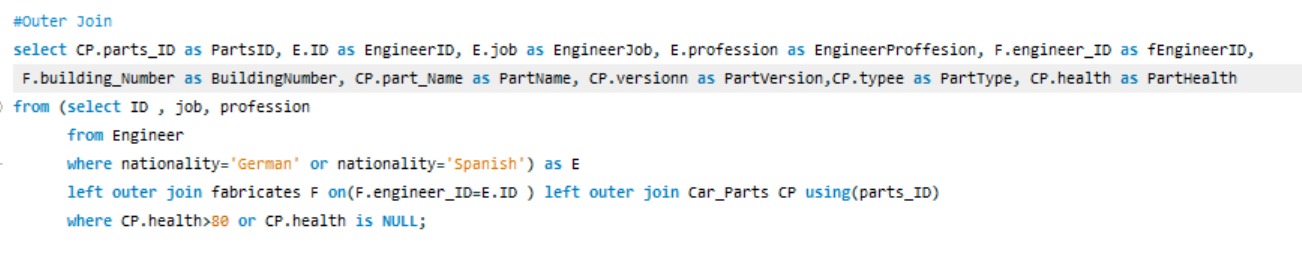


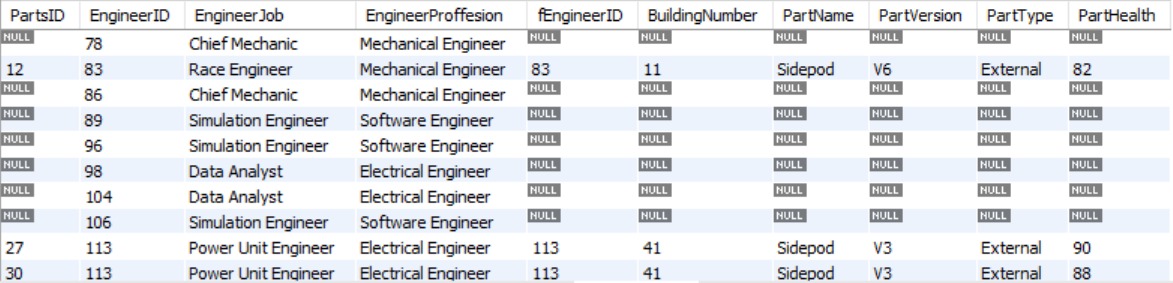
Explanation:

Create a view named "Race\_Highlights" that provides information about driver positions, race names, driver names, and the number of accidents for races where the driver position is between 1 and 3. The view is created by joining the "Race," "Participates," and "Driver" tables and includes columns such as positionn, race\_Name, namee, and nb\_of\_Accidents. The nb\_of\_Accidents column is calculated using a subquery that counts the number of accidents for each driver in a specific race. The results are grouped by race name.

**Outer Join:**

Return all the information about the German and Spanish Engineer that fabricated car parts with health greater than 80, and those engineers who did not fabricate car parts yet.



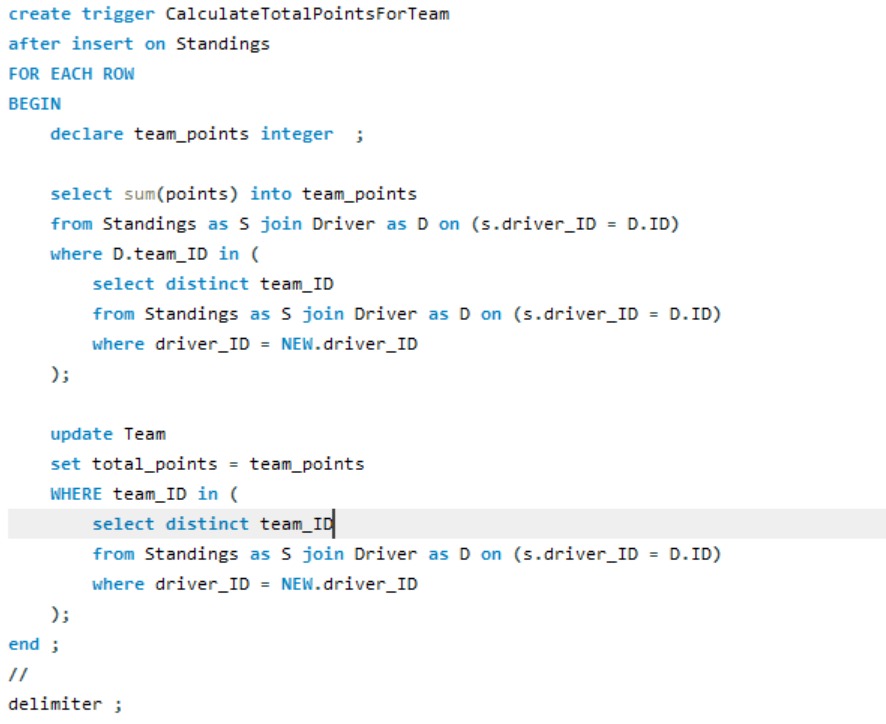


Explanation:

Retrieve information about parts, engineers, and the fabrication details for parts with a health greater than 80 or parts with no health information. The data is obtained through a LEFT OUTER JOIN between a subquery (aliased as E) containing engineers with German or Spanish nationality, the "fabricates" table (aliased as F) joined on engineer ID, and the "Car\_Parts" table (aliased as CP) using the parts ID. The results include columns such as PartsID, EngineerID, EngineerJob, EngineerProffesion, fEngineerID, BuildingNumber, PartName, PartVersion, PartType, and PartHealth. The WHERE clause filters the results based on part health conditions. The LEFT OUTER JOIN ensures that all engineers are included, even if they have no corresponding fabrication records or parts information.

**Trigger:**

This trigger is designed to automatically calculate and update the total\_points column in the Team table after each new entry in the Standings table, considering the total points of each driver in the team.



Explanation:

Create a trigger named "CalculateTotalPointsForTeam" that automatically calculates and updates the total points for a team after a new record is inserted into the "Standings" table. The trigger is set to execute for each new row (AFTER INSERT ON Standings). The trigger involves the use of variables, a SELECT statement to calculate the sum of points for a team, and an UPDATE statement to set the total points for the corresponding team in the "Team" table.

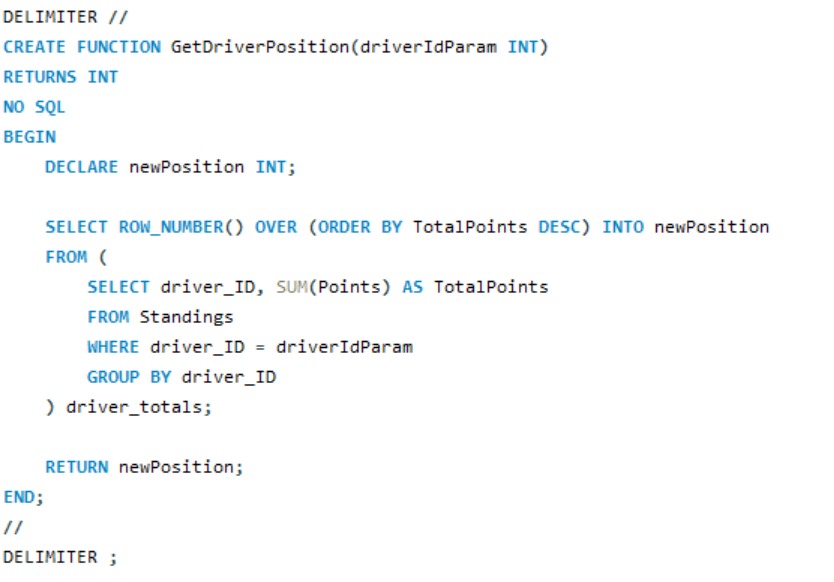
The trigger logic works by selecting the team IDs associated with the driver whose standings are being inserted (NEW.driver\_ID) and calculating the sum of points for that team. Subsequently, it updates the "Team" table to set the total points for the identified team.

\

Bonus:

**Function:**

The GetDriverPosition function is designed to calculate and return the position of a specific driver based on their total points in the Standings table

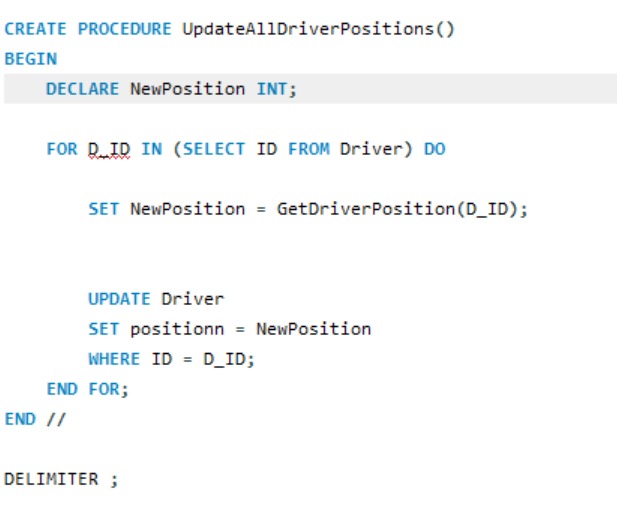


Explanation:

Create a MySQL function named "GetDriverPosition" that takes a driver ID as a parameter and returns the position of the driver based on their total points in the "Standings" table. The function uses the ROW\_NUMBER() window function to assign a position to each driver based on the total points in descending order. The function then returns the calculated position.

**Procedure:**

This stored procedure updates the position column for each driver in the Driver table based on their total points calculated by the GetDriverPosition function. The procedure iterates over all drivers, calculates the new position, and updates the corresponding record in the table.



Explanation:

Create a MySQL stored procedure named "UpdateAllDriverPositions" that updates the positions of all drivers in the "Driver" table based on their total points in the "Standings" table. The procedure utilizes a FOR loop to iterate through each driver ID, calls the "GetDriverPosition" function to calculate the new position, and then updates the "Driver" table with the calculated position.

**Conclusion:**

The Formula 1 database presented here captures the intricate relationships and entities within the world of Formula 1 racing. The design reflects the diverse roles and responsibilities of individuals, teams, and components involved in the sport. Key entities such as drivers, teams, cars, races, sponsors, and personnel have been modeled to

Despite the comprehensive nature of the database, continuous improvement is essential to keep pace with evolving requirements and technological advancements. General methods to enhance the database include:

Improving the datasets to ensure the avoidance of redundancy across the data which improves runtime of queries and saves storage. In addition to reconsidering the relationships between certain entities which could be improved to increase the efficiency by implementing other approaches depending on the formula 1 team biases. Furthermore , adding more triggers , functions and procedures to provide real-time response and manipulation saved data and ensure no errors or misconceptions are found in the database because the security of the data for the clients is of top priority. To finalize , adding extra entities and relationships to cover the whole services needed for a formula one team in order to help it advance, facilitate its research and development projects and improve it’s future strategies.