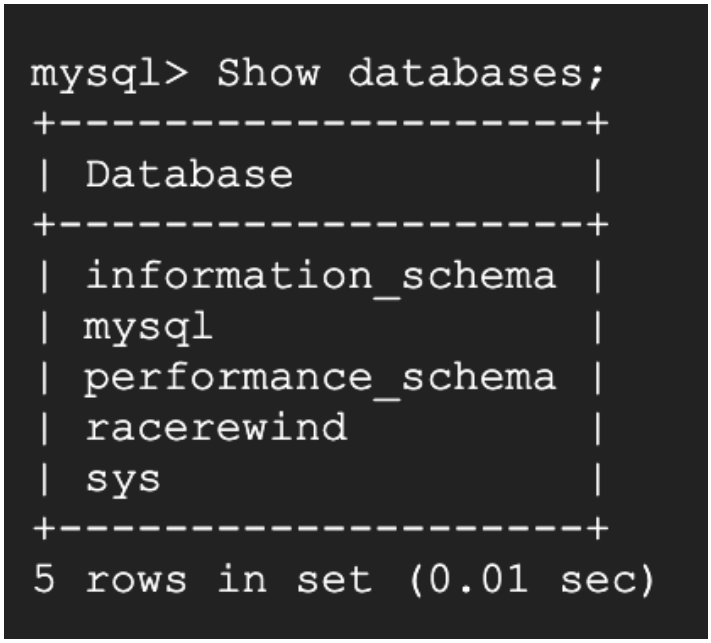
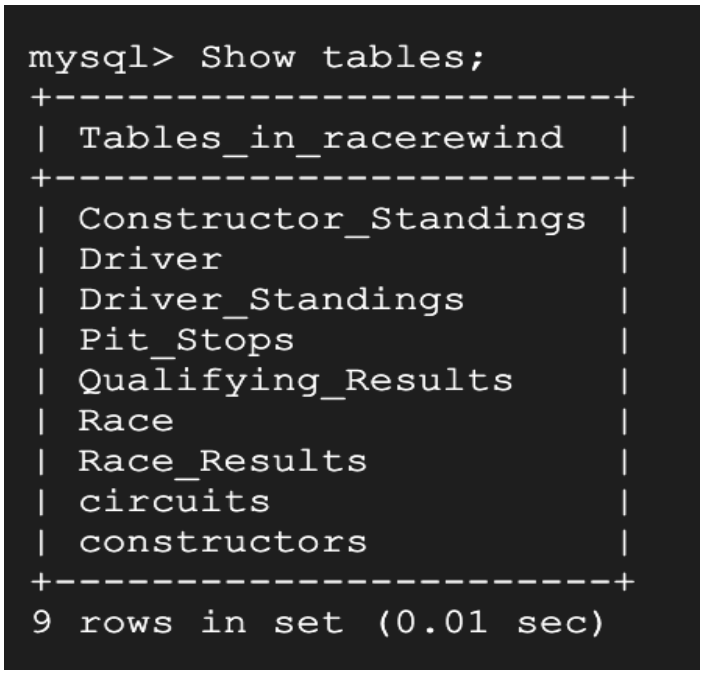
CS 411 - Database Systems
Project Track 1 - Stage 3
Database Design
Team - 055 - desi - racerewind

Database Connection Screenshots:



```
mysql> Show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| racerewind         |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

USE racerewind -> show tables;



```
mysql> Show tables;
+----------------------+
| Tables_in_racerewind |
+----------------------+
| Constructor_Standings |
| Driver               |
| Driver_Standings     |
| Pit_Stops            |
| Qualifying_Results   |
| Race                 |
| Race_Results         |
| circuits             |
| constructors         |
+----------------------+
9 rows in set (0.01 sec)
```

Data Definition Language (DDL) Commands

```
CREATE TABLE circuits (
    circuitID INT PRIMARY KEY,
    Name VARCHAR(100),
    Location VARCHAR(100),
    Country VARCHAR(50)
);

CREATE TABLE Race (
    raceID INT PRIMARY KEY,
    Year INT,
    Name VARCHAR(100),
    circuitID INT,
    TotalLaps INT,
    FOREIGN KEY (circuitID) REFERENCES circuits(circuitID)
);

CREATE TABLE Qualifying_Results (
    qualifyID INT PRIMARY KEY,
    raceID INT,
    driverID INT,
    constructorID INT,
    Q1Time TIME,
    Q2Time TIME,
    Q3Time TIME,
    GridPosition INT,
    FOREIGN KEY (raceID) REFERENCES Race(raceID),
    FOREIGN KEY (driverID) REFERENCES Driver(driverID),
    FOREIGN KEY (constructorID) REFERENCES constructors(constructorID)
);

CREATE TABLE Race_Results (
    resultID INT PRIMARY KEY,
    raceID INT,
    driverID INT,
    constructorID INT,
    qualifyID INT,
    FinalPosition INT,
    RacePoints INT,
```

```
    FastestLap TIME,
    FOREIGN KEY (raceID) REFERENCES Race(raceID),
    FOREIGN KEY (driverID) REFERENCES Driver(driverID),
    FOREIGN KEY (constructorID) REFERENCES Constructor(constructorID),
    FOREIGN KEY (qualifyID) REFERENCES Qualifying_Results(qualifyID)
);

CREATE TABLE Pit_Stops (
    pitStopID INT PRIMARY KEY,
    raceID INT,
    driverID INT,
    StopNumber INT,
    StopDuration TIME,
    FOREIGN KEY (raceID) REFERENCES Race(raceID),
    FOREIGN KEY (driverID) REFERENCES Driver(driverID)
);

CREATE TABLE Constructor_Standings (
    constructorStandingID INT PRIMARY KEY,
    raceID INT,
    constructorID INT,
    ConstructorPoints INT,
    ConstructorPosition INT,
    FOREIGN KEY (raceID) REFERENCES Race(raceID),
    FOREIGN KEY (constructorID) REFERENCES constructors(constructorID)
);

CREATE TABLE Driver_Standings (
    driverStandingID INT PRIMARY KEY,
    raceID INT,
    driverID INT,
    constructorID INT,
    DriverPoints INT,
    DriverPosition INT,
    FOREIGN KEY (raceID) REFERENCES Race(raceID),
    FOREIGN KEY (driverID) REFERENCES Driver(driverID),
    FOREIGN KEY (constructorID) REFERENCES constructors(constructorID)
);
```

```
CREATE TABLE constructors (
   constructorID INT PRIMARY KEY,
   Name VARCHAR(100),
   ConstructorNationality VARCHAR(50)
);
```

```
CREATE TABLE Driver (
   driverID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   DriverNationality VARCHAR(50)
);
```

Table Row Counts:

```
mysql> Select Count(*) From Constructor_Standings;
+----------+
| Count(*) |
+----------+
|    13272 |
+----------+
1 row in set (0.04 sec)
```

```
mysql> Select Count(*) From Driver;
+----------+
| Count(*) |
+----------+
|      860 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> Select Count(*) From Driver_Standings;
+----------+
| Count(*) |
+----------+
|    34209 |
+----------+
1 row in set (0.42 sec)
```

```
mysql> Select Count(*) From Pit_Stops;
+----------+
| Count(*) |
+----------+
|    10491 |
+----------+
1 row in set (0.11 sec)
```

```
mysql> Select Count(*) From Qualifying_Results;
+----------+
| Count(*) |
+----------+
|    10255 |
+----------+
1 row in set (0.27 sec)
```

```
mysql> Select Count(*) From Race;
+----------+
| Count(*) |
+----------+
|     1126 |
+----------+
1 row in set (0.04 sec)
```

```
mysql> Select Count(*) From Race_Results;
+----------+
| Count(*) |
+----------+
|    26520 |
+----------+
1 row in set (0.25 sec)
```

```
mysql> Select Count(*) From circuits;
+----------+
| Count(*) |
+----------+
|       78 |
+----------+
1 row in set (0.06 sec)
```

```
mysql> Select Count(*) From constructors;
+----------+
| Count(*) |
+----------+
|      213 |
+----------+
1 row in set (0.06 sec)
```

Advance Query:

Query 1:

This SQL query calculates the average pit stop duration per year for a specified Grand Prix, allowing users to analyze trends over time. It uses a JOIN to combine the Race and Pit_Stops tables based on raceID, enabling data from both tables to be included in the calculation. The GROUP BY clause aggregates pit stop durations by year, producing an annual average that shows whether pit stop times have increased, decreased, or stabilized. Additionally, the CASTfunction converts StopDuration to a floating-point value for precise calculations, while the ORDER BY clause sorts the results chronologically by year. This combination of SQL features helps users gain insights into pit stop efficiency trends for specific races.

```
mysql> SELECT
    ->     r.Year,
    ->     r.Name,
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    -> FROM
    ->     Race r
    -> JOIN
    ->     Pit_Stops ps ON r.raceID = ps.raceID
    -> WHERE
    ->     r.Name = 'Australian Grand Prix'
    -> GROUP BY
    ->     r.Year, r.Name
    -> ORDER BY
    ->     r.Year
    -> LIMIT 15;
+------+-----------------------+--------------------+
| Year | Name                  | avg_pitstop_time   |
+------+-----------------------+--------------------+
| 2011 | Australian Grand Prix | 24.333333333333332 |
| 2012 | Australian Grand Prix | 24.658536585365855 |
| 2013 | Australian Grand Prix |  23.20754716981132 |
| 2014 | Australian Grand Prix | 23.852941176470587 |
| 2015 | Australian Grand Prix |                 25 |
| 2016 | Australian Grand Prix |  22.85185185185185 |
| 2017 | Australian Grand Prix |              24.05 |
| 2018 | Australian Grand Prix |               22.3 |
| 2019 | Australian Grand Prix |  23.59090909090909 |
| 2022 | Australian Grand Prix | 18.454545454545453 |
| 2023 | Australian Grand Prix |             19.125 |
| 2024 | Australian Grand Prix |  19.77777777777778 |
+------+-----------------------+--------------------+
12 rows in set (0.11 sec)
```

Query 2:

This SQL query identifies the year with the minimum average pit stop duration for each Grand Prix, enabling users to see the fastest pit stop year for each race. It begins by joining the Race and Pit_Stops tables on the raceID column to integrate pit stop data with each race record. To calculate yearly averages, it uses the GROUP BY clause on both Name and Year, allowing the aggregation of pit stop durations for each race by year. Within the HAVING clause, a subquery is used to determine the minimum average pit stop time for each Grand Prix. This subquery groups records by Name and Year, calculates average pit stop times, and then filters to find the minimum average for the specified race name. By including a subquery that isn't easily replaceable by a simple join, this query ensures that only the year with the minimum average duration for each race is displayed. Additionally, the CAST function converts StopDuration into a floating-point number to improve the accuracy of the average calculations. Finally, the ORDER BY clause organizes the results by race name and year, making it easy for users to review and compare the fastest pit stop years across different races.

```
mysql> SELECT
    ->     r.Name,
    ->     r.Year,
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    -> FROM
    ->     Race r
    -> JOIN
    ->     Pit_Stops ps ON r.raceID = ps.raceID
    -> GROUP BY
    ->     r.Name, r.Year
    -> HAVING
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) = (
    ->         SELECT
    ->             MIN(avg_pitstop_time)
    ->         FROM (
    ->             SELECT
    ->                 r.Name,
    ->                 AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    ->             FROM
    ->                 Race r
    ->             JOIN
    ->                 Pit_Stops ps ON r.raceID = ps.raceID
    ->             GROUP BY
    ->                 r.Name, r.Year
    ->         ) AS subquery
    ->         WHERE subquery.Name = r.Name
    ->     )
    -> ORDER BY
    ->     r.Name, r.Year
    -> LIMIT 15;
```

```
+----------------------------+------+--------------------+
| Name                       | Year | avg_pitstop_time   |
+----------------------------+------+--------------------+
| 70th Anniversary Grand Prix | 2020 | 28.634146341463413 |
| Abu Dhabi Grand Prix       | 2011 | 21.095238095238095 |
| Australian Grand Prix      | 2022 | 18.454545454545453 |
| Austrian Grand Prix        | 2016 | 20.486842105263158 |
| Azerbaijan Grand Prix      | 2021 |             18.825 |
| Bahrain Grand Prix         | 2013 |  22.91549295774648 |
| Belgian Grand Prix         | 2018 | 22.323529411764707 |
| Brazilian Grand Prix       | 2017 |  20.26086956521739 |
| British Grand Prix         | 2011 | 25.944444444444443 |
| Canadian Grand Prix        | 2013 | 21.454545454545453 |
| Chinese Grand Prix         | 2017 | 20.365853658536587 |
| Dutch Grand Prix           | 2022 |  19.61111111111111 |
| Eifel Grand Prix           | 2020 | 23.606060606060606 |
| Emilia Romagna Grand Prix  | 2024 | 30.666666666666668 |
| European Grand Prix        | 2016 | 21.666666666666668 |
+----------------------------+------+--------------------+
15 rows in set (0.02 sec)
```

Query 3:

This SQL query recalculates driver standings by applying the modern F1 points system to any season before 2010, offering fans a view of how the rankings might have differed under today's scoring method. The query begins by creating a temporary table (Race_Results_2004_Points) that assigns modern points to drivers based on their final race positions. It uses a CASE statement to assign points (e.g., 25 for 1st, 18 for 2nd) for each finishing position down to 10th place, with any position beyond that receiving zero points.
Next, it aggregates the points for each driver in 2004 within the Driver_Total_Points_2004 temporary table. Here, it uses the SUM function to calculate the total points for each driver, grouping by driverID to ensure each driver's season total is computed accurately. Finally, in the main query, it retrieves each driver's name and total points from 2004, joining the Driver table to include the driver's full name. The results are ordered by total_points in descending order to display the drivers ranked by their recalculated points for that season. This allows users to explore potential shifts in driver rankings for 2004, as if the modern points system were in place.

```
mysql> WITH Race_Results_2004_Points AS (
    ->     SELECT
    ->         rr.driverID,
    ->         r.raceID,
    ->         r.Year,
    ->         rr.FinalPosition,
    ->         CASE
    ->             WHEN rr.FinalPosition = 1 THEN 25
    ->             WHEN rr.FinalPosition = 2 THEN 18
    ->             WHEN rr.FinalPosition = 3 THEN 15
    ->             WHEN rr.FinalPosition = 4 THEN 12
    ->             WHEN rr.FinalPosition = 5 THEN 10
    ->             WHEN rr.FinalPosition = 6 THEN 8
    ->             WHEN rr.FinalPosition = 7 THEN 6
    ->             WHEN rr.FinalPosition = 8 THEN 4
    ->             WHEN rr.FinalPosition = 9 THEN 2
    ->             WHEN rr.FinalPosition = 10 THEN 1
    ->             ELSE 0
    ->         END AS points
    ->     FROM
    ->         Race_Results rr
    ->     JOIN
    ->         Race r ON rr.raceID = r.raceID
    ->     WHERE
    ->         r.Year = 2004
    -> ),
    -> Driver_Total_Points_2004 AS (
    ->     SELECT
    ->         rr.driverID,
    ->         SUM(rr.points) AS total_points
    ->     FROM
    ->         Race_Results_2004_Points rr
    ->     GROUP BY
    ->         rr.driverID
    -> )
    -> SELECT
    ->     d.driverID,
    ->     CONCAT(d.FirstName, ' ', d.LastName) AS DriverName,
    ->     dtp.total_points AS points_2004_system
    -> FROM
    ->     Driver_Total_Points_2004 dtp
    -> JOIN
    ->     Driver d ON dtp.driverID = d.driverID
    -> ORDER BY
    ->     dtp.total_points DESC
    -> LIMIT 15;
```

driverID	DriverName	points_2004_system
30	Michael Schumacher	367
22	Rubens Barrichello	271
18	Jenson Button	206
4	Fernando Alonso	146
31	Juan Pablo Montoya	146
15	Jarno Trulli	117
8	Kimi Räikkönen	112
11	Takuma Sato	89
14	David Coulthard	71
21	Giancarlo Fisichella	71
23	Ralf Schumacher	61
13	Felipe Massa	43
17	Mark Webber	28
44	Olivier Panis	20
42	Antônio Pizzonia	18

15 rows in set (0.01 sec)

Query 4:

This SQL query recalculates constructor standings for any season before 2010. using the modern F1 points system to show how team rankings might differ under today's scoring. It starts by creating a temporary table (Race_Results_2004_Points) that assigns points based on current F1 rules (e.g., 25 for 1st, 18 for 2nd) to drivers' positions. Then, the Constructor_Race_Points_New CTE aggregates these driver points by constructor for each race, while Constructor_Total_Points_New further sums these race-level points to calculate each constructor's season total.

Finally, the query joins the constructors table to retrieve full constructor names, ordering results by total points in descending order. This allows fans to see how constructor rankings might shift if modern points were applied to past seasons.

```
mysql> WITH Race_Results_2019_Points AS (
    ->     SELECT
    ->         rr.driverID,
    ->         rr.constructorID,
    ->         r.raceID,
    ->         r.Year,
    ->         rr.FinalPosition,
    ->         CASE
    ->             WHEN rr.FinalPosition = 1 THEN 25
    ->             WHEN rr.FinalPosition = 2 THEN 18
    ->             WHEN rr.FinalPosition = 3 THEN 15
    ->             WHEN rr.FinalPosition = 4 THEN 12
    ->             WHEN rr.FinalPosition = 5 THEN 10
    ->             WHEN rr.FinalPosition = 6 THEN 8
    ->             WHEN rr.FinalPosition = 7 THEN 6
    ->             WHEN rr.FinalPosition = 8 THEN 4
    ->             WHEN rr.FinalPosition = 9 THEN 2
    ->             WHEN rr.FinalPosition = 10 THEN 1
    ->             ELSE 0
    ->         END AS driver_points
    ->     FROM
    ->         Race_Results rr
    ->     JOIN
    ->         Race r ON rr.raceID = r.raceID
    ->     WHERE
    ->         r.Year = 2004
    -> ),
    -> Constructor_Race_Points_New AS (
    ->     SELECT
    ->         rp.constructorID,
    ->         rp.raceID,
    ->         SUM(rp.driver_points) AS race_points_new_system
    ->     FROM
    ->         Race_Results_2019_Points rp
    ->     GROUP BY
    ->         rp.constructorID, rp.raceID
    -> ),
    -> Constructor_Total_Points_New AS (
    ->     SELECT
    ->         crp.constructorID,
    ->         SUM(crp.race_points_new_system) AS total_points_new_system
    ->     FROM
    ->         Constructor_Race_Points_New crp
    ->     GROUP BY
    ->         crp.constructorID
    -> )
    -> SELECT
    ->     c.constructorID,
    ->     c.name AS ConstructorName,
    ->     ctp.total_points_new_system AS Updated_Points
    -> FROM
    ->     Constructor_Total_Points_New ctp
    -> JOIN
    ->     constructors c ON ctp.constructorID = c.constructorID
    -> ORDER BY
    ->     ctp.total_points_new_system DESC;
+---------------+-----------------+----------------+
| constructorID | ConstructorName | Updated_Points |
+---------------+-----------------+----------------+
|             6 | Ferrari         |            638 |
|            16 | BAR             |            295 |
|             4 | Renault         |            265 |
|             3 | Williams        |            226 |
|             1 | McLaren         |            183 |
|            15 | Sauber          |            114 |
|            19 | Jaguar          |             40 |
|             7 | Toyota          |             32 |
|            17 | Jordan          |             17 |
|            18 | Minardi         |              7 |
+---------------+-----------------+----------------+
10 rows in set (0.00 sec)
```

Indexing Analysis:

Query 1

Original Cost:

```
mysql> explain analyze SELECT
    ->     r.Year,
    ->     r.Name,
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    -> FROM
    ->     Race r
    -> JOIN
    ->     Pit_Stops ps ON r.raceID = ps.raceID
    -> WHERE
    ->     r.Name = 'Australian Grand Prix'
    -> GROUP BY
    ->     r.Year, r.Name
    -> ORDER BY
    ->     r.Year;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
| EXPLAIN



              |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
| -> Sort: r.`Year`  (actual time=56.933..56.934 rows=12 loops=1)
    -> Table scan on <temporary>  (actual time=56.132..56.137 rows=12 loops=1)
        -> Aggregate using temporary table  (actual time=56.128..56.128 rows=12 loops=1)
            -> Nested loop inner join  (cost=4857.43 rows=1075) (actual time=0.129..55.657 rows=353 loops=1)
                -> Filter: (ps.raceID is not null)  (cost=1093.53 rows=10754) (actual time=0.068..47.157 rows=10491 loops=1)
                    -> Table scan on ps  (cost=1093.53 rows=10754) (actual time=0.066..46.033 rows=10491 loops=1)
                -> Filter: (r.`Name` = 'Australian Grand Prix')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=10491)
                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
1 row in set (0.06 sec)
```

First Index Attempt and Cost

```
mysql> CREATE INDEX idx_pit_stops_raceid ON Pit_Stops(raceID);
ERROR 1061 (42000): Duplicate key name 'idx_pit_stops_raceid'
mysql> explain analyze SELECT
    ->     r.Year,
    ->     r.Name,
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    -> FROM
    ->     Race r
    -> JOIN
    ->     Pit_Stops ps ON r.raceID = ps.raceID
    -> WHERE
    ->     r.Name = 'Australian Grand Prix'
    -> GROUP BY
    ->     r.Year, r.Name
    -> ORDER BY
    ->     r.Year;
+-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| EXPLAIN



     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| -> Sort: r.`Year`  (actual time=9.326..9.327 rows=12 loops=1)
    -> Table scan on <temporary>  (actual time=9.296..9.299 rows=12 loops=1)
        -> Aggregate using temporary table  (actual time=9.293..9.293 rows=12 loops=1)
            -> Nested loop inner join  (cost=4846.55 rows=1075) (actual time=0.072..9.003 rows=353 loops=1)
                -> Filter: (ps.raceID is not null)  (cost=1082.65 rows=10754) (actual time=0.044..3.638 rows=10491 loops=1)
                    -> Table scan on ps  (cost=1082.65 rows=10754) (actual time=0.043..2.924 rows=10491 loops=1)
                -> Filter: (r.`Name` = 'Australian Grand Prix')  (cost=0.25 rows=0.1) (actual time=0.000..0.000 rows=0 loops=10491)
                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

Second Index Attempt and Cost:

```
mysql> CREATE INDEX idx_race_name ON Race(Name);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT
    ->     r.Year,
    ->     r.Name,
    ->     AVG(CAST(ps.StopDuration AS FLOAT)) AS avg_pitstop_time
    -> FROM
    ->     Race r
    -> JOIN
    ->     Pit_Stops ps ON r.raceID = ps.raceID
    -> WHERE
    ->     r.Name = 'Australian Grand Prix'
    -> GROUP BY
    ->     r.Year, r.Name
    -> ORDER BY
    ->     r.Year;
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: r.`Year`  (actual time=1.440..1.441 rows=12 loops=1)
    -> Table scan on <temporary>  (actual time=1.388..1.418 rows=12 loops=1)
        -> Aggregate using temporary table  (actual time=1.386..1.386 rows=12 loops=1)
            -> Nested loop inner join  (cost=530.30 rows=1491) (actual time=0.650..1.193 rows=353 loops=1)
                -> Index lookup on r using idx_race_name (Name='Australian Grand Prix')  (cost=8.30 rows=38) (actual time=0.515..0.524 rows=38 loops=1)
                -> Index lookup on ps using idx_pit_stops_raceid (raceID=r.raceID)  (cost=9.92 rows=39) (actual time=0.015..0.017 rows=9 loops=38)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

The final index design, CREATE INDEX idx_race_name ON Race(Name);, was chosen to optimize filtering by Name = 'Australian Grand Prix'. This index significantly reduced query cost by improving filtering efficiency in the WHERE clause and enhancing the join operation with Pit_Stops. Analysis showed this index effectively minimized the rows scanned, resulting in better performance with minimal storage impact.

Query 2:

Original Cost:

```
-> Sort: r.`Name`, r.`Year`  (actual time=33.776..33.780 rows=41 loops=1)
    -> Filter: (avg(cast(ps.StopDuration as float)) = (select #2))  (actual time=33.545..33.732 rows=41 loops=1)
        -> Table scan on <temporary>  (actual time=33.541..33.709 rows=274 loops=1)
            -> Aggregate using temporary table  (actual time=33.539..33.539 rows=274 loops=1)
                -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.072..7.375 rows=10491 loops=1)
                    -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.055..3.677 rows=10491 loops=1)
                        -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.052..2.892 rows=10491 loops=1)
                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
        -> Select #2 (subquery in condition; dependent)
            -> Aggregate: min(subquery.avg_pitstop_time)  (cost=0.00..0.00 rows=1) (actual time=0.065..0.065 rows=1 loops=274)
                -> Index lookup on subquery using <auto_key0> (Name=r.`Name`)  (actual time=0.061..0.063 rows=10 loops=274)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=16.436..16.436 rows=274 loops=1)
                        -> Table scan on <temporary>  (actual time=16.123..16.213 rows=274 loops=1)
                            -> Aggregate using temporary table  (actual time=16.119..16.119 rows=274 loops=1)
                                -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.026..7.930 rows=10491 loops=1)
                                    -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.021..3.915 rows=10491 loops=1)
                                        -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.021..3.111 rows=10491 loops=1)
                                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: min(subquery.avg_pitstop_time)  (cost=0.00..0.00 rows=1) (actual time=0.065..0.065 rows=1 loops=274)
        -> Index lookup on subquery using <auto_key0> (Name=r.`Name`)  (actual time=0.061..0.063 rows=10 loops=274)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=16.436..16.436 rows=274 loops=1)
                -> Table scan on <temporary>  (actual time=16.123..16.213 rows=274 loops=1)
                    -> Aggregate using temporary table  (actual time=16.119..16.119 rows=274 loops=1)
                        -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.026..7.930 rows=10491 loops=1)
                            -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.021..3.915 rows=10491 loops=1)
                                -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.021..3.111 rows=10491 loops=1)
                            -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
```

First Index Attempt:

```
mysql>
mysql> CREATE INDEX idx_race_name_year_raceid ON Race (Name, Year, raceID);
```

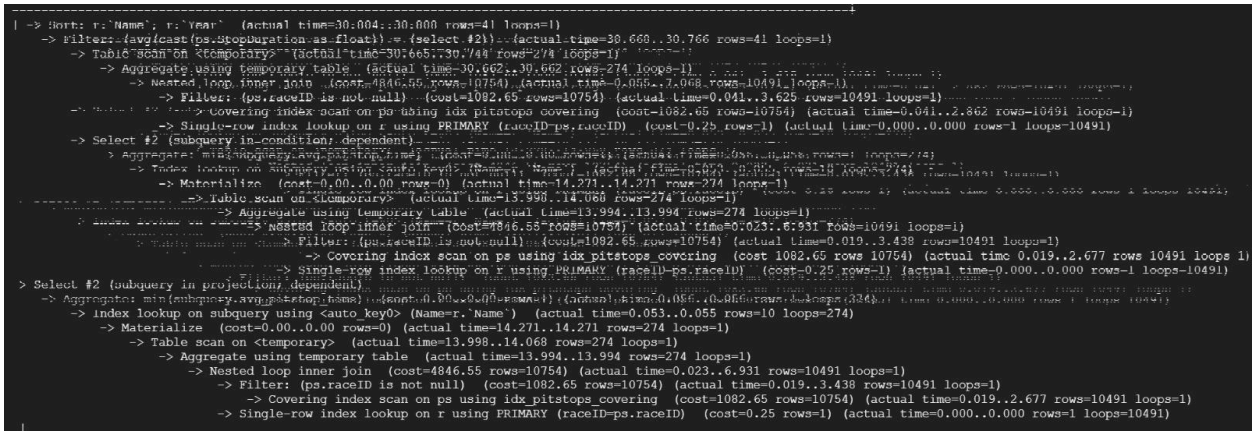
Cost:

```
-> Sort: r.`Name`, r.`Year`  (actual time=31.215..31.219 rows=41 loops=1)
    -> Filter: (avg(cast(ps.StopDuration as float)) = (select #2))  (actual time=31.065..31.176 rows=41 loops=1)
        -> Table scan on <temporary>  (actual time=31.062..31.155 rows=274 loops=1)
            -> Aggregate using temporary table  (actual time=31.059..31.059 rows=274 loops=1)
                -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.042..7.440 rows=10491 loops=1)
                    -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.032..3.622 rows=10491 loops=1)
                        -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.031..2.805 rows=10491 loops=1)
                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
        -> Select #2 (subquery in condition; dependent)
            -> Aggregate: min(subquery.avg_pitstop_time)  (cost=0.00..0.00 rows=1) (actual time=0.056..0.056 rows=1 loops=274)
                -> Index lookup on subquery using <auto_key0> (Name=r.`Name`)  (actual time=0.053..0.055 rows=10 loops=274)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=14.086..14.086 rows=274 loops=1)
                        -> Table scan on <temporary>  (actual time=13.809..13.883 rows=274 loops=1)
                            -> Aggregate using temporary table  (actual time=13.805..13.805 rows=274 loops=1)
                                -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.023..6.878 rows=10491 loops=1)
                                    -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.019..3.404 rows=10491 loops=1)
                                        -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.019..2.640 rows=10491 loops=1)
                                    -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: min(subquery.avg_pitstop_time)  (cost=0.00..0.00 rows=1) (actual time=0.056..0.056 rows=1 loops=274)
        -> Index lookup on subquery using <auto_key0> (Name=r.`Name`)  (actual time=0.053..0.055 rows=10 loops=274)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=14.086..14.086 rows=274 loops=1)
                -> Table scan on <temporary>  (actual time=13.809..13.883 rows=274 loops=1)
                    -> Aggregate using temporary table  (actual time=13.805..13.805 rows=274 loops=1)
                        -> Nested loop inner join  (cost=4867.52 rows=10754) (actual time=0.023..6.878 rows=10491 loops=1)
                            -> Filter: (ps.raceID is not null)  (cost=1103.62 rows=10754) (actual time=0.019..3.404 rows=10491 loops=1)
                                -> Covering index scan on ps using idx_pitstops_raceid_stopduration  (cost=1103.62 rows=10754) (actual time=0.019..2.640 rows=10491 loops=1)
                            -> Single-row index lookup on r using PRIMARY (raceID=ps.raceID)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=10491)
```

Second Index Attempt:

```
mysql> CREATE INDEX idx_pitstops_stopduration ON Pit_Stops (StopDuration);
```

Cost:



Adding the three indexes did not reduce the query cost because they don't tackle the main performance issue, which is the heavy aggregations and the correlated subquery in the HAVING clause. Indexes are helpful for speeding up data retrieval when filtering specific rows, but they don't significantly improve performance for aggregate functions like AVG that process many rows. The indexes on StopDuration and on Name and Year didn't help because the query still needs to process all relevant data to compute averages and evaluate the subquery. As a result, the indexes couldn't reduce the amount of data scanned or the computations required, so there was no reduction in cost.

Query 3

Original Cost:

```
| -> Nested loop inner join  (cost=676.60 rows=0) (actual time=1.323..1.369 rows=25 loops=1)
    -> Sort: dtp.total_points DESC  (cost=2.60..2.60 rows=0) (actual time=1.309..1.310 rows=25 loops=1)
        -> Filter: (dtp.driverID is not null)  (cost=2.50..2.50 rows=0) (actual time=1.279..1.284 rows=25 loops=1)
            -> Table scan on dtp  (cost=2.50..2.50 rows=0) (actual time=1.278..1.280 rows=25 loops=1)
                -> Materialize CTE Driver_Total_Points_2004  (cost=0.00..0.00 rows=0) (actual time=1.277..1.277 rows=25 loops=1)
                    -> Table scan on <temporary>  (actual time=1.248..1.251 rows=25 loops=1)
                        -> Aggregate using temporary table  (actual time=1.246..1.246 rows=25 loops=1)
                            -> Nested loop inner join  (cost=1078.95 rows=2697) (actual time=0.176..0.962 rows=360 loops=1)
                                -> Filter: (r.`Year` = 2004)  (cost=114.10 rows=113) (actual time=0.114..0.471 rows=18 loops=1)
                                    -> Table scan on r  (cost=114.10 rows=1126) (actual time=0.068..0.377 rows=1126 loops=1)
                                -> Index lookup on rr using raceID (raceID=r.raceID)  (cost=6.20 rows=24) (actual time=0.023..0.026 rows=20 loops=18)
    -> Single-row index lookup on d using PRIMARY (driverID=dtp.driverID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=25)
 |
1 row in set (0.03 sec)
```

First Index Attempt

```
mysql> CREATE INDEX idx_race_year ON Race(Year);
Query OK, 0 rows affected (0.52 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Cost After First Index Attempt

```
| -> Nested loop inner join  (cost=110.35 rows=0) (actual time=0.922..0.968 rows=25 loops=1)
    -> Sort: dtp.total_points DESC  (cost=2.60..2.60 rows=0) (actual time=0.908..0.910 rows=25 loops=1)
        -> Filter: (dtp.driverID is not null)  (cost=2.50..2.50 rows=0) (actual time=0.875..0.879 rows=25 loops=1)
            -> Table scan on dtp  (cost=2.50..2.50 rows=0) (actual time=0.873..0.876 rows=25 loops=1)
                -> Materialize CTE Driver_Total_Points_2004  (cost=0.00..0.00 rows=0) (actual time=0.873..0.873 rows=25 loops=1)
                    -> Table scan on <temporary>  (actual time=0.855..0.858 rows=25 loops=1)
                        -> Aggregate using temporary table  (actual time=0.853..0.853 rows=25 loops=1)
                            -> Nested loop inner join  (cost=156.68 rows=431) (actual time=0.067..0.554 rows=360 loops=1)
                                -> Covering index lookup on r using idx_race_year (Year=2004)  (cost=2.44 rows=18) (actual time=0.016..0.021 rows=18 loops=1)
                                -> Index lookup on rr using raceID (raceID=r.raceID)  (cost=6.31 rows=24) (actual time=0.026..0.028 rows=20 loops=18)
    -> Single-row index lookup on d using PRIMARY (driverID=dtp.driverID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=25)
 |
```

Second Index Attempt

```
mysql> CREATE INDEX idx_race_results_raceID ON Race_Results(raceID);
Query OK, 0 rows affected (0.32 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Cost After Second Index Attempt

```
| -> Nested loop inner join  (cost=676.60 rows=0) (actual time=1.444..1.695 rows=25 loops=1)
    -> Sort: dtp.total_points DESC  (cost=2.60..2.60 rows=0) (actual time=1.424..1.426 rows=25 loops=1)
        -> Filter: (dtp.driverID is not null)  (cost=2.50..2.50 rows=0) (actual time=1.386..1.391 rows=25 loops=1)
            -> Table scan on dtp  (cost=2.50..2.50 rows=0) (actual time=1.385..1.387 rows=25 loops=1)
                -> Materialize CTE Driver_Total_Points_2004  (cost=0.00..0.00 rows=0) (actual time=1.384..1.384 rows=25 loops=1)
                    -> Table scan on <temporary>  (actual time=1.268..1.271 rows=25 loops=1)
                        -> Aggregate using temporary table  (actual time=1.267..1.267 rows=25 loops=1)
                            -> Nested loop inner join  (cost=1057.89 rows=2697) (actual time=0.188..1.003 rows=360 loops=1)
                                -> Filter: (r.`Year` = 2004)  (cost=114.10 rows=113) (actual time=0.117..0.468 rows=18 loops=1)
                                    -> Table scan on r  (cost=114.10 rows=1126) (actual time=0.097..0.404 rows=1126 loops=1)
                                -> Index lookup on rr using idx_race_results_raceID (raceID=r.raceID)  (cost=6.01 rows=24) (actual time=0.026..0.028 rows=20 loops=18)
    -> Single-row index lookup on d using PRIMARY (driverID=dtp.driverID)  (cost=0.25 rows=1) (actual time=0.011..0.011 rows=1 loops=25)
 |
```

The final index design, CREATE INDEX idx_race_year ON Race(Year);, was chosen to optimize filtering by Year = 2004, reducing the query cost by minimizing the rows scanned in the Race table. Analysis showed this index directly benefits the WHERE clause, significantly improving performance with minimal additional storage overhead.

Query 4

Original Cost

```
| -> Sort: ctp.total_points_new_system DESC  (actual time=1.552..1.553 rows=10 loops=1)
    -> Stream results  (cost=695.45 rows=0) (actual time=1.388..1.538 rows=10 loops=1)
        -> Nested loop inner join  (cost=695.45 rows=0) (actual time=1.384..1.530 rows=10 loops=1)
            -> Table scan on c  (cost=21.45 rows=212) (actual time=0.049..0.092 rows=212 loops=1)
            -> Index lookup on ctp using <auto_key0> (constructorID=c.constructorID)  (actual time=0.007..0.007 rows=0 loops=212)
                -> Materialize CTE Constructor_Total_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.331..1.331 rows=10 loops=1)
                    -> Table scan on <temporary>  (actual time=1.315..1.316 rows=10 loops=1)
                        -> Aggregate using temporary table  (actual time=1.315..1.315 rows=10 loops=1)
                            -> Table scan on crp  (cost=2.50..2.50 rows=0) (actual time=1.221..1.239 rows=180 loops=1)
                                -> Materialize CTE Constructor_Race_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.221..1.221 rows=180 loops=1)
                                    -> Table scan on <temporary>  (actual time=1.183..1.201 rows=180 loops=1)
                                        -> Aggregate using temporary table  (actual time=1.182..1.182 rows=180 loops=1)
                                            -> Nested loop inner join  (cost=1057.89 rows=2697) (actual time=0.092..0.834 rows=360 loops=1)
                                                -> Filter: (r.`Year` = 2004)  (cost=114.10 rows=113) (actual time=0.038..0.339 rows=18 loops=1)
                                                    -> Table scan on r  (cost=114.10 rows=1126) (actual time=0.024..0.278 rows=1126 loops=1)
                                                -> Index lookup on rr using idx_race_results_raceID (raceID=r.raceID)  (cost=6.01 rows=24) (actual time=0.023..0.026 rows=20 loops=
18)
 |
```

First Index Attempt

```
mysql> CREATE INDEX idx_race_year ON Race(Year);
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Cost After First Index Attempt

```
| -> Nested loop inner join  (cost=110.35 rows=0) (actual time=1.277..1.370 rows=10 loops=1)
    -> Sort: ctp.total_points_new_system DESC  (cost=2.60..2.60 rows=0) (actual time=1.242..1.243 rows=10 loops=1)
        -> Filter: (ctp.constructorID is not null)  (cost=2.50..2.50 rows=0) (actual time=1.221..1.223 rows=10 loops=1)
            -> Table scan on ctp  (cost=2.50..2.50 rows=0) (actual time=1.220..1.221 rows=10 loops=1)
                -> Materialize CTE Constructor_Total_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.219..1.219 rows=10 loops=1)
                    -> Table scan on <temporary>  (actual time=1.198..1.200 rows=10 loops=1)
                        -> Aggregate using temporary table  (actual time=1.198..1.198 rows=10 loops=1)
                            -> Table scan on crp  (cost=2.50..2.50 rows=0) (actual time=1.104..1.122 rows=180 loops=1)
                                -> Materialize CTE Constructor_Race_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.104..1.104 rows=180 loops=1)
                                    -> Table scan on <temporary>  (actual time=1.008..1.027 rows=180 loops=1)
                                        -> Aggregate using temporary table  (actual time=1.006..1.006 rows=180 loops=1)
                                            -> Nested loop inner join  (cost=153.31 rows=431) (actual time=0.074..0.602 rows=360 loops=1)
                                                -> Covering index lookup on r using idx_race_year (Year=2004)  (cost=2.44 rows=18) (actual time=0.023..0.027 rows=18 loops=1)
                                                -> Index lookup on rr using idx_race_results_raceID (raceID=r.raceID)  (cost=6.12 rows=24) (actual time=0.028..0.031 rows=20 loops=
18)
    -> Single-row index lookup on c using PRIMARY (constructorID=ctp.constructorID)  (cost=0.25 rows=1) (actual time=0.012..0.012 rows=1 loops=10)
 |
```

Second Index Attempt

```
mysql> CREATE INDEX idx_race_results_raceID_2  ON Race_Results(raceID);
Query OK, 0 rows affected, 1 warning (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Cost After Second Index Attempt

```
| -> Sort: ctp.total_points_new_system DESC  (actual time=1.772..1.773 rows=10 loops=1)
    -> Stream results  (cost=695.45 rows=0) (actual time=1.602..1.755 rows=10 loops=1)
        -> Nested loop inner join  (cost=695.45 rows=0) (actual time=1.597..1.747 rows=10 loops=1)
            -> Table scan on c  (cost=21.45 rows=212) (actual time=0.173..0.219 rows=212 loops=1)
            -> Index lookup on ctp using <auto_key0> (constructorID=c.constructorID)  (actual time=0.007..0.007 rows=0 loops=212)
                -> Materialize CTE Constructor_Total_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.420..1.420 rows=10 loops=1)
                    -> Table scan on <temporary>  (actual time=1.402..1.403 rows=10 loops=1)
                        -> Aggregate using temporary table  (actual time=1.402..1.402 rows=10 loops=1)
                            -> Table scan on crp  (cost=2.50..2.50 rows=0) (actual time=1.310..1.327 rows=180 loops=1)
                                -> Materialize CTE Constructor_Race_Points_New  (cost=0.00..0.00 rows=0) (actual time=1.309..1.309 rows=180 loops=1)
                                    -> Table scan on <temporary>  (actual time=1.270..1.289 rows=180 loops=1)
                                        -> Aggregate using temporary table  (actual time=1.269..1.269 rows=180 loops=1)
                                            -> Nested loop inner join  (cost=1184.29 rows=2697) (actual time=0.131..0.892 rows=360 loops=1)
                                                -> Filter: (r.`Year` = 2004)  (cost=114.10 rows=113) (actual time=0.039..0.357 rows=18 loops=1)
                                                    -> Table scan on r  (cost=114.10 rows=1126) (actual time=0.026..0.297 rows=1126 loops=1)
                                                -> Index lookup on rr using idx_race_results_raceID (raceID=r.raceID)  (cost=7.13 rows=24) (actual time=0.026..0.028 rows=20 loops=
18)
 |
```

The selected index, CREATE INDEX idx_race_year ON Race(Year);, targets the filtering condition Year = 2004 in the query's WHERE clause. Analysis demonstrated that this index minimizes the row scans required from Race, effectively lowering query cost. This design provides efficient filtering with low storage impact, improving performance significantly.