**OBJECTIVE**

1. List the different types of columns in table “ball\_by\_ball” (using information schema).

Ans-

'Match\_Id', 'Over\_Id', 'Ball\_Id', 'Innings\_No', 'Team\_Batting', 'Team\_Bowling', 'Striker\_Batting\_Position', 'Striker', 'Non\_Striker', 'Bowler'

|  |
| --- |
| SELECT  COLUMN\_NAME,  DATA\_TYPE  FROM  INFORMATION\_SCHEMA.COLUMNS  WHERE  TABLE\_NAME = 'ball\_by\_ball'; |

Explanation-

This SQL query retrieves the column names and their corresponding data types from the table named ball\_by\_ball.

* **INFORMATION\_SCHEMA.COLUMNS**: A system view that contains metadata about all the columns in the database.
* **COLUMN\_NAME**: Represents the name of each column in the specified table.
* **DATA\_TYPE**: Specifies the data type of each column (e.g., INT, VARCHAR, DATE).
* **TABLE\_NAME = 'ball\_by\_ball'**: Filters the results to only include metadata for the ball\_by\_ball table.

1. What is the total number of runs scored in 1st season by RCB (bonus: also include the extra runs using the extra runs table)?

ANS-

|  |
| --- |
| SELECT  SUM (COALESCE (bs.Runs\_Scored, 0)) + SUM(COALESCE(er.Extra\_Runs, 0)) AS Total\_Runs  FROM matches m  JOIN ball\_by\_ball bb ON m.Match\_Id = bb.Match\_Id  LEFT JOIN batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id  AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id  AND bb.Innings\_No = bs.Innings\_No  LEFT JOIN extra\_runs er ON bb.Match\_Id = er.Match\_Id  AND bb.Over\_Id = er.Over\_Id  AND bb.Ball\_Id = er.Ball\_Id  AND bb.Innings\_No = er.Innings\_No  JOIN team t ON bb.Team\_Batting = t.Team\_Id  WHERE t.Team\_Name = 'RCB'  AND m.Season\_Id = 1; |

Explanation-

This SQL query calculates the **total runs scored by RCB (Royal Challengers Bangalore) in Season 1** of the IPL by summing up:

1. **Runs\_Scored**: Runs scored by batsmen, using the batsman\_scored table.
2. **Extra\_Runs**: Additional runs (e.g., wides, no-balls) from the extra\_runs table.

**Key Points:**

* **SUM(COALESCE(..., 0))**: Ensures null values are treated as 0 during summation.
* **Joins**:
  + matches and ball\_by\_ball: Links match details with ball-by-ball data.
  + batsman\_scored and extra\_runs: Links scoring details using match, over, ball, and innings numbers.
  + team: Identifies the batting team by its ID.
* **Filters**:
  + t.Team\_Name = 'RCB': Restricts data to RCB.
  + m.Season\_Id = 1: Limits the analysis to Season 1.
* The query sums up all the runs scored by the Royal Challengers Bangalore (RCB) in the first season, including both runs scored by the batsmen and any extra runs (such as wides, no-balls, leg-byes, etc.
* The **extra runs** are included in the calculation via the table, which is joined to the Ball\_by\_ball table. This accounts for runs not directly credited to the batsman but added to the team's score, like no-balls, wides, and byes.
* Result Output:



* The total of **1983 runs scored by RCB in the IPL season 1** including extra runs they got.

1. How many players were over 25 during season 2?

ANS-

|  |
| --- |
| SELECT COUNT(DISTINCT pm.Player\_Id) AS Players\_Over\_25  FROM player\_match pm  JOIN matches m ON pm.Match\_Id = m.Match\_Id  JOIN player p ON pm.Player\_Id = p.Player\_Id  WHERE m.Season\_Id = 2  AND TIMESTAMPDIFF(YEAR, p.DOB, m.Match\_Date) > 25; |

**Explanation-**

1. **COUNT(DISTINCT pm.Player\_Id)**: Counts unique players from the player\_match table.
2. **JOINs**:
   * matches link players to matches.
   * player retrieves player details, including their date of birth (DOB).
3. **Age Calculation**:
   * TIMESTAMPDIFF(YEAR, p.DOB, m.Match\_Date) > 25: Calculates the player's age on the match date and filters those older than 25.
4. **Filter**:
   * m.Season\_Id = 2: Limits the data to matches from Season 2.

**Result:-**

|  |
| --- |
| Player\_Over\_25 |
| 82 |

1. How many matches did RCB win in season 1?

ANS-

|  |
| --- |
| SELECT COUNT(\*) AS RCB\_Wins  FROM season s join matches m  on s.season\_id = m.season\_id  JOIN team t on m.match\_winner= t.team\_id  where s.season\_id=1  and t.team\_name ='Royal Challengers Bangalore'; |

* **Explanation:**

This SQL query counts the number of matches won by the Royal Challengers Bangalore (RCB) during Season 1.

1. Selecting Win Count: The outer query selects a count of all rows that meet the specified criteria, labelling the result as **RCB\_Wins**.
2. Joining Tables:
   * season (s): This table contains information about the different seasons. It is joined to the matches table using the season\_id to link the season with its respective matches.
   * matches (m): This table records the match details, including the winner of each match.
   * team (t): This table contains team information, which is joined to the matches table to identify the winning team.
3. Filtering Conditions:
   * The where clause specifies that the query should only consider matches from Season 1 (**s.season\_id=1**).
   * It further filters the results to include only those matches where the winning team is the Royal Challengers Bangalore (**t.team\_name='Royal Challengers Bangalore'**).

The final output provides the total count of matches won by RCB in Season 1, helping analyze their performance during that season.

* **Summary:**
* The query looks for matches in season 1 where the team\_id of the match\_winner corresponds to RCB and then counts how many such matches exist.
* **Result**

|  |
| --- |
| **RCB\_Wins** |
| 4 |

* RCB have won total 4 matches in the IPL Season 1

1. List the top 10 players according to their strike rate in the last 4 seasons.

ANS-

|  |
| --- |
| select P.Player\_Id,P.Player\_Name,Total\_Runs,Balls\_Faced,  round(Total\_Runs / Balls\_Faced \* 100, 2) AS Strike\_Rate  from(select PM.Player\_Id,  sum(BS.Runs\_Scored) as Total\_Runs,  count(BB.Ball\_Id) as Balls\_Faced  from Player\_Match PM join Matches M on PM.Match\_Id = M.Match\_Id  join Batsman\_Scored BS on M.Match\_Id = BS.Match\_Id  join Ball\_by\_Ball BB on M.Match\_Id = BB.Match\_Id  and BS.Over\_Id = BB.Over\_Id  and BS.Ball\_Id = BB.Ball\_Id  where M.Season\_Id >= (select max(Season\_Id) from Season)-4  group by PM.Player\_Id) as PlayerStats  join Player P on PlayerStats.Player\_Id = P.Player\_Id  order by Strike\_Rate desc  limit 10; |

**Explanation:**

1. **Main Tables Involved**:
   * Player: Contains player details (e.g., Player\_Id, Player\_Name).
   * Player\_Match: Maps players to specific matches (Player\_Id, Match\_Id).
   * Matches: Provides match-specific information, including Season\_Id.
   * Batsman\_Scored: Tracks runs scored by a batsman in each ball (Runs\_Scored, Match\_Id, Over\_Id, Ball\_Id).
   * Ball\_by\_Ball: Tracks ball-by-ball events (Ball\_Id, Over\_Id, etc.).
2. **Subquery (PlayerStats)**:
   * **Joins**:
     + Player\_Match is joined with Matches to link players to match details.
     + Matches are further joined with Batsman\_Scored and Ball\_by\_Ball to count balls faced and calculate runs scored.
     + The Over\_Id and Ball\_Id ensure runs scored and ball details align correctly.
   * **Filters**:
     + Only considers matches from the last 4 seasons (M.Season\_Id >= (SELECT MAX(Season\_Id) - 4 FROM Season)).
   * **Aggregations**:
     + SUM(BS.Runs\_Scored) calculates the total runs scored (Total\_Runs).
     + COUNT(BB.Ball\_Id) counts balls faced by each player (Balls\_Faced).
   * **Grouping**:
     + The results are grouped by PM.Player\_Id to calculate the statistics for each player.
3. **Main Query**:
   * The subquery (PlayerStats) is joined with the Player table to fetch player names.
   * **Strike Rate Calculation**:
     + ROUND(Total\_Runs / Balls\_Faced \* 100, 2) computes the strike rate.
   * **Ordering**:
     + Players are ordered by Strike\_Rate in descending order.
   * **Limiting**:
     + Only the top 10 players are selected (LIMIT 10).

**Final Summary:**

The query identifies the **top 10 players** with the **highest strike rates** in the last 4 IPL seasons. It computes key metrics like:

* Total runs scored (Total\_Runs),
* Balls faced (Balls\_Faced), and
* Strike rate (Strike\_Rate).

The results are presented in descending order of strike rate, including the player name for better readability.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player\_Id** | **Player\_Name** | **Total\_Runs** | **Ball\_Faced** | **Strike\_Rate** |
| 469 | T Mishra | 734 | 485 | 151.34 |
| 463 | T Shamsi | 2713 | 1875 | 144.69 |
| 461 | AF Milne | 770 | 472 | 163.14 |
| 454 | MG Neser | 681 | 442 | 154.07 |
| 428 | F Behardien | 1913 | 1285 | 148.87 |
| 423 | PSP Handscomb | 1322 | 918 | 144.01 |
| 417 | TM Head | 2005 | 1384 | 144.87 |
| 337 | Sunny Gupta | 633 | 431 | 146.87 |
| 317 | RR Bhatkal | 771 | 500 | 154.20 |
| 289 | AC Blizzard | 708 | 489 | 144.79 |

1. What average runs are scored by each batsman considering all the seasons?

ANS-

|  |
| --- |
| SELECT  p.Player\_Name,  SUM(bs.Runs\_Scored) AS Total\_Runs,  COUNT(DISTINCT CONCAT(bs.Match\_Id, bs.Innings\_No)) AS Innings\_Played,  ROUND(SUM(bs.Runs\_Scored) / COUNT(DISTINCT CONCAT(bs.Match\_Id, bs.Innings\_No)), 2) AS Average\_Runs  FROM batsman\_scored bs  JOIN ball\_by\_ball bb ON bs.Match\_Id = bb.Match\_Id  AND bs.Over\_Id = bb.Over\_Id  AND bs.Ball\_Id = bb.Ball\_Id  AND bs.Innings\_No = bb.Innings\_No  JOIN player p ON bb.Striker = p.Player\_Id  GROUP BY p.Player\_Name  ORDER BY Average\_Runs DESC; |

**Explanation**

1. **Main Tables Involved**:
   * batsman\_scored (bs): Contains data on runs scored by batsmen (Runs\_Scored, Match\_Id, Over\_Id, Ball\_Id, Innings\_No).
   * ball\_by\_ball (bb): Tracks events for each ball, linking it to players (Striker, Match\_Id, etc.).
   * player (p): Contains player details (Player\_Name, Player\_Id).
2. **Joins**:
   * batsman\_scored (bs) is joined with ball\_by\_ball (bb):
     + Ensures that runs scored (Runs\_Scored) align with the correct ball events (Match\_Id, Over\_Id, Ball\_Id, Innings\_No).
   * ball\_by\_ball (bb) is joined with the player (p):
     + Fetches the name of the player (Player\_Name) associated with the Striker (batsman on strike).
3. **Columns Selected**:
   * p.Player\_Name: The name of the player.
   * SUM(bs.Runs\_Scored) AS Total\_Runs: Total runs scored by each player.
   * COUNT(DISTINCT CONCAT(bs.Match\_Id, bs.Innings\_No)) AS Innings\_Played:
     + Calculates the number of distinct innings played by combining Match\_Id and Innings\_No.
   * ROUND(SUM(bs.Runs\_Scored) / COUNT(DISTINCT CONCAT(bs.Match\_Id, bs.Innings\_No)), 2) AS Average\_Runs:
     + Batting average = Total runs / Number of innings played.
     + The result is rounded to 2 decimal places.
4. **Grouping**:
   * The data is grouped by p.Player\_Name, ensuring calculations are done for each player.
5. **Ordering**:
   * Results are sorted in descending order of Average\_Runs to highlight players with the best batting averages.

**Final Summary**

This query calculates and ranks players based on their batting averages across all matches. It provides the following details for each player:

* Player Name,
* Total Runs Scored,
* Innings Played,
* Batting Average (Average Runs per Innings).

**Result output:**

* These are the top 10 players from the above query who have the highest batting average considering all the seasons.

|  |  |  |  |
| --- | --- | --- | --- |
| **Player\_Name** | **Total\_Runs** | **Innings\_Played** | **Average\_Runs** |
| LMP Simmons | 942 | 22 | 42.82 |
| CH Gayle | 3447 | 94 | 36.67 |
| SE Marsh | 2225 | 62 | 35.89 |
| N Rana | 104 | 3 | 34.67 |
| MEK Hussey | 1977 | 58 | 34.09 |
| ML Hayden | 1107 | 33 | 33.55 |
| DA Warner | 3373 | 101 | 33.40 |
| MN van Wyk | 167 | 5 | 33.40 |
| V Kohli | 4105 | 132 | 31.10 |
| AM Rahane | 2675 | 89 | 30.06 |

1. What are the average wickets each bowler takes considering all the seasons?

ANS-

|  |
| --- |
| SELECT  p.Player\_Name,  COUNT(bb.Bowler) AS Wickets,  COUNT(bb.Bowler) / NULLIF(COUNT(DISTINCT bb.Match\_Id), 0) AS Average\_Wickets  FROM  ball\_by\_ball bb  JOIN  player p ON bb.Bowler = p.Player\_Id  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id  AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id  WHERE  bs.Runs\_Scored = 0  GROUP BY  p.Player\_Name  ORDER BY  Average\_Wickets DESC; |

**Explanation:**

1. **Tables Involved**:
   * **ball\_by\_ball (bb)**: Contains ball-by-ball data, including Bowler and Match\_Id.
   * **player (p)**: Stores player details like Player\_Name and Player\_Id.
   * **batsman\_scored (bs)**: Tracks runs scored on each ball, including outs.
2. **Joins**:
   * **bb joined with p**:
     + Links the Bowler in bb to their name in the player table.
   * **LEFT JOIN with bs**:
     + Matches each ball's details (Match\_Id, Over\_Id, Ball\_Id) to determine if a wicket was taken (Runs\_Scored = 0).
3. **Filters**:
   * bs.Runs\_Scored = 0:
     + Assumes that when Runs\_Scored is 0, it indicates the batsman was out, i.e., a wicket.
4. **Aggregations**:
   * **COUNT(bb.Bowler)**:
     + Counts the total number of wickets taken by each bowler.
   * **COUNT(DISTINCT bb.Match\_Id)**:
     + Counts the number of unique matches the bowler participated in.
   * **COUNT(bb.Bowler) / NULLIF(COUNT(DISTINCT bb.Match\_Id), 0)**:
     + Calculates the **average wickets per match**.
     + NULLIF prevents division by zero by returning NULL if the denominator is 0.
5. **Grouping**:
   * Results are grouped by p.Player\_Name to calculate stats for each bowler.
6. **Ordering**:
   * Bowlers are ordered by Average\_Wickets in descending order to highlight the most effective bowlers.

**Final Summary:**

This query retrieves bowlers’ performance metrics:

* **Player\_Name**: Name of the bowler.
* **Wickets**: Total wickets taken.
* **Average\_Wickets**: Average number of wickets per match.

It filters out instances where no wickets were taken (bs.Runs\_Scored = 0) and ranks bowlers by their **average wickets per match**, showing the most impactful bowlers at the top.

**Result Output:**

These are the top 10 players who have the highest wicket average per inning.

|  |  |  |
| --- | --- | --- |
| **Player\_Name** | **Wickets** | **Average\_Wickets** |
| Sohail Tanvir | 227 | 20.6364 |
| WPUJC Vaas | 267 | 20.5385 |
| MG Johnson | 881 | 20.4884 |
| NM Coulter-Nile | 347 | 20.4118 |
| JE Taylor | 102 | 20.4000 |
| RJ Harris | 753 | 20.3514 |
| M Morkel | 1411 | 20.1571 |
| SM Pollock | 262 | 20.1538 |
| SL Malinga | 1969 | 20.0918 |
| GR Napier | 20 | 20.0000 |

1. List all the players who have average runs scored greater than the overall average and who have taken wickets greater than the overall average.

ANS-

|  |
| --- |
| WITH PlayerAverageRuns AS (  SELECT  p.Player\_Id,  p.Player\_Name,  AVG(bs.Runs\_Scored) AS Avg\_Runs\_Scored  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id -- Linking players to matches  LEFT JOIN  batsman\_scored bs ON pm.Match\_Id = bs.Match\_Id  AND pm.Role\_Id = 1 -- Assuming Role\_Id 1 represents batsmen  GROUP BY  p.Player\_Id, p.Player\_Name  ),  PlayerWickets AS (  SELECT  p.Player\_Id,  COUNT(bb.Bowler) AS Wickets  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  JOIN  ball\_by\_ball bb ON bb.Bowler = p.Player\_Id  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id  AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id  WHERE  bs.Runs\_Scored = 0 -- Assuming a runs scored of 0 indicates a wicket  GROUP BY  p.Player\_Id  ),  OverallAverages AS (  SELECT  AVG(Avg\_Runs\_Scored) AS Overall\_Avg\_Runs  FROM  PlayerAverageRuns  ),  OverallWickets AS (  SELECT  AVG(Wickets) AS Overall\_Avg\_Wickets  FROM  PlayerWickets  )  SELECT  ar.Player\_Name,  ar.Avg\_Runs\_Scored,  pw.Wickets  FROM  PlayerAverageRuns ar  JOIN  PlayerWickets pw ON ar.Player\_Id = pw.Player\_Id  CROSS JOIN  OverallAverages oa  CROSS JOIN  OverallWickets ow  WHERE  ar.Avg\_Runs\_Scored > oa.Overall\_Avg\_Runs  AND pw.Wickets > ow.Overall\_Avg\_Wickets  ORDER BY  ar.Player\_Name; |

**Explanation:**

1. **CTE 1: PlayerAverageRuns**:
   * **Purpose**: Calculates the average runs scored by each player who is a batsman.
   * **Joins**:
     + player is joined with player\_match to link players to matches.
     + batsman\_scored is joined to fetch the runs scored in each match.
   * **Condition**:
     + pm.Role\_Id = 1 ensures only batsmen are considered.
   * **Aggregation**:
     + AVG(bs.Runs\_Scored) calculates the average runs scored for each player.
   * **Grouping**:
     + Grouped by Player\_Id and Player\_Name.
2. **CTE 2: PlayerWickets**:
   * **Purpose**: Counts the total wickets taken by each player who is a bowler.
   * **Joins**:
     + player is joined with player\_match and ball\_by\_ball to track bowlers and their bowling data.
     + batsman\_scored is joined to ensure accurate mapping of runs scored on specific balls.
   * **Condition**:
     + bs.Runs\_Scored = 0 assumes a score of 0 indicates a wicket.
   * **Aggregation**:
     + COUNT(bb.Bowler) calculates the total wickets for each player.
   * **Grouping**:
     + Grouped by Player\_Id.
3. **CTE 3: OverallAverages**:
   * **Purpose**: Calculates the overall average runs scored across all players.
   * Uses AVG(Avg\_Runs\_Scored) from PlayerAverageRuns.
4. **CTE 4: OverallWickets**:
   * **Purpose**: Calculates the overall average number of wickets across all players.
   * Uses AVG(Wickets) from PlayerWickets.
5. **Final Query**:
   * **Joins**:
     + Combines batting and bowling stats by joining PlayerAverageRuns and PlayerWickets on Player\_Id.
   * **CROSS JOIN**:
     + Joins with OverallAverages and OverallWickets to access global performance benchmarks.
   * **Filters**:
     + ar.Avg\_Runs\_Scored > oa.Overall\_Avg\_Runs ensures players' average runs exceed the overall average.
     + pw.Wickets > ow.Overall\_Avg\_Wickets ensures players' wickets exceed the overall average.
   * **Ordering**:
     + Results are sorted alphabetically by Player\_Name.

**Final Summary:**

This query identifies **all-rounder players** who perform better than average in:

* **Batting**: Their average runs exceed the overall average runs scored by all batsmen.
* **Bowling**: Their wickets exceed the overall average wickets taken by all bowlers.

**Output**

|  |  |  |
| --- | --- | --- |
| **Player\_Name** | **Avg\_Runs\_Scored** | **Wickets** |
| JP Duminy | 1.2749 | 36498 |
| RG Sharma | 1.2923 | 29110 |
| SK Raina | 1.2933 | 80446 |
| SR Watson | 1.3322 | 126712 |
| Z Khan | 1.2606 | 147576 |

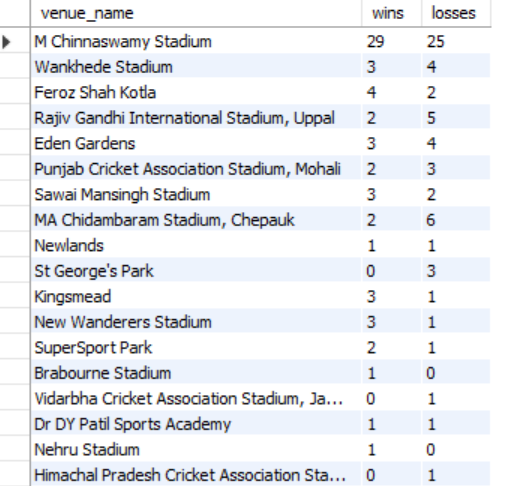
1. Create a table rcb\_record table that shows the wins and losses of RCB in an individual venue.

ANS-

|  |
| --- |
| create table rcb\_record  (venue\_name varchar(100),  wins int,  losses int);  insert into rcb\_record (venue\_name,wins,losses)  (select v.venue\_name,  count(case when m.match\_winner=t.team\_id then 1 end) as wins,  count(case when m.match\_winner<>t.team\_id then 1 end) as losses  from venue v join matches m on v.venue\_id=m.venue\_id  join team t on m.team\_1=t.team\_id or m.team\_2=t.team\_id  where t.team\_name='Royal Challengers Bangalore'  group by v.venue\_name);  select \* from rcb\_record; |

* Explanation:
* **Create the Table**:
* A table named rcb\_record is created with three columns: venue\_name, wins, and losses.
* **Insert Data**:
* The table is filled with data showing how many matches the Royal Challengers Bangalore (RCB) has won and lost at each venue.
* It counts wins and losses by checking the match results where RCB is involved.
* Summary:

1. The SQL script creates a table called rcb\_record to show the Royal Challengers Bangalore's (RCB) wins and losses at different venues. It counts how many matches RCB won and lost at each venue and stores this information in the table. Finally, it retrieves and displays all the records from the rcb\_record table.

* Output:

1. What is the impact of bowling style on wickets taken?

ANS-

|  |
| --- |
| SELECT  bs.Bowling\_skill AS Bowling\_Style,  COUNT(bb.Bowler) AS Total\_Wickets,  AVG(wickets\_per\_bowler) AS Average\_Wickets  FROM  ball\_by\_ball bb  JOIN  player p ON bb.Bowler = p.Player\_Id  JOIN  bowling\_style bs ON p.Bowling\_skill = bs.Bowling\_Id  LEFT JOIN (  SELECT  bb.Bowler,  COUNT(\*) AS wickets\_per\_bowler  FROM  ball\_by\_ball bb  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id  AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id  WHERE  bs.Runs\_Scored = 0 -- Assuming runs scored of 0 indicates a wicket  GROUP BY  bb.Bowler  ) AS Wickets ON bb.Bowler = Wickets.Bowler  GROUP BY  bs.Bowling\_skill  ORDER BY  Total\_Wickets DESC; |

**Explanation:**

1. **Tables Involved**:
   * ball\_by\_ball: Tracks details of each ball bowled in matches, including the bowler (Bowler).
   * player: Contains player details, including the player ID (Player\_Id) and bowling skill (Bowling\_skill).
   * bowling\_style: Maps Bowling\_Id to specific bowling styles (e.g., spin, fast).
2. **Wickets Subquery**:

* Calculates the total wickets taken by each bowler (wickets\_per\_bowler).
* **Join and Condition**:
  + Joins ball\_by\_ball with batsman\_scored to match each ball with its corresponding runs scored.
  + Filters only balls where Runs\_Scored = 0, assumed to indicate a wicket.
* **Grouping**:
  + Groups by Bowler to compute the number of wickets per bowler.

1. **Main Query**:

* **Joins**:
  + ball\_by\_ball is joined with player to fetch player details like bowling style.
  + bowling\_style is joined to map the Bowling\_skill from player to its description.
  + The Wickets subquery is joined on Bowler to get the number of wickets for each bowler.
* **Aggregations**:
  + COUNT(bb.Bowler) AS Total\_Wickets: Calculates the total number of wickets taken by bowlers of each style.
  + AVG(wickets\_per\_bowler) AS Average\_Wickets: Computes the average number of wickets per bowler for each style.
* **Grouping**:
  + Groups the results by Bowling\_skill to summarize the performance by bowling style.
* **Sorting**:
  + Results are ordered by Total\_Wickets in descending order to prioritize bowling styles with the highest wicket counts.

**Final Summary:**

This query provides insights into bowler performance based on their bowling styles, including:

* **Total Wickets**: The cumulative wickets taken by all bowlers of a particular style.
* **Average Wickets per Bowler**: The mean number of wickets taken by individual bowlers in each style.

The results are sorted by total wickets, helping identify the most effective bowling styles in terms of wicket-taking capabilities.

**Result**

|  |  |  |
| --- | --- | --- |
| **Bowling\_Style** | **Total\_Wickets** | **Average\_Wickets** |
| Right-arm medium | 23397 | 867.0723 |
| Right-arm offbreak | 18791 | 934.9229 |
| Right-arm fast-medium | 17540 | 737.0214 |
| Right-arm medium-fast | 17442 | 853.7432 |
| Slow left-arm orthodox | 16127 | 689.0697 |
| Right-arm fast | 12313 | 1165.8753 |
| Left-arm fast-medium | 8033 | 1018.6853 |
| Legbreak | 6314 | 1412.5036 |
| Left-arm medium-fast | 5830 | 1221.7955 |
| Legbreak googly | 5377 | 567.1351 |
| Left-arm fast | 2344 | 689.5201 |
| Left-arm medium | 2297 | 374.1402 |
| Slow left-arm chinaman | 778 | 236.8072 |

1. Write the SQL query to provide a status of whether the performance of the team is better than the previous year's performance based on the number of runs scored by the team in the season and the number of wickets taken

ANS-

|  |
| --- |
| WITH TeamPerformance AS (  SELECT  s.Season\_Year,  SUM(bs.Runs\_Scored) AS Total\_Runs,  COUNT(bb.Bowler) AS Total\_Wickets  FROM  season s  JOIN  matches m ON s.Season\_Id = m.Season\_Id  LEFT JOIN  batsman\_scored bs ON m.Match\_Id = bs.Match\_Id  LEFT JOIN  ball\_by\_ball bb ON m.Match\_Id = bb.Match\_Id  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Assuming Team\_Id 1 represents the team in question  GROUP BY  s.Season\_Year  ),  PerformanceComparison AS (  SELECT  curr.Season\_Year AS Current\_Year,  curr.Total\_Runs AS Current\_Runs,  curr.Total\_Wickets AS Current\_Wickets,  prev.Total\_Runs AS Previous\_Runs,  prev.Total\_Wickets AS Previous\_Wickets  FROM  TeamPerformance curr  LEFT JOIN  TeamPerformance prev ON curr.Season\_Year = prev.Season\_Year + 1  )  SELECT  Current\_Year,  Current\_Runs,  Current\_Wickets,  Previous\_Runs,  Previous\_Wickets,  CASE  WHEN Current\_Runs > Previous\_Runs AND Current\_Wickets > Previous\_Wickets THEN 'Better'  WHEN Current\_Runs < Previous\_Runs AND Current\_Wickets < Previous\_Wickets THEN 'Worse'  ELSE 'Same or Mixed'  END AS Performance\_Status  FROM  PerformanceComparison  ORDER BY  Current\_Year DESC; |

**Explanation of the Query**

1. **Purpose**:  
   The query aims to analyze the year-over-year performance of a specific cricket team (Team\_Id = 1) based on its total runs scored and total wickets taken during each season. It compares the current season's performance with the previous season to determine if the performance was "Better," "Worse," or "Same or Mixed."
2. **Breaking Down the Query**:
   * **TeamPerformance CTE**:
     + Groups data by season (s.Season\_Year) and calculates:
       - Total\_Runs: The total runs scored by the team.
       - Total\_Wickets: The total number of wickets taken by the team.
     + Uses the batsman\_scored and ball\_by\_ball tables to compute runs and wickets.
     + Filters matches where the team in question is either Team\_1 or Team\_2.
   * **PerformanceComparison CTE**:
     + Joins the TeamPerformance CTE with itself to align the current season (curr) with the previous season (prev) based on the year.
     + Retrieves:
       - Current\_Year, Current\_Runs, and Current\_Wickets for the current season.
       - Previous\_Runs and Previous\_Wickets for the previous season.
   * **Final Query**:
     + Compares Current\_Runs and Current\_Wickets with Previous\_Runs and Previous\_Wickets to categorize performance as:
       - **Better**: Both runs and wickets improved.
       - **Worse**: Both runs and wickets declined.
       - **Same or Mixed**: Any other scenario.
     + Orders the results by Current\_Year in descending order for easy review.

**Output:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Current\_Year** | **Current\_Runs** | **Current\_Wickets** | **Previous\_Runs** | **Previous\_Wickets** | **Performance\_Status** |
| 2016 | 1015809 | 777469 | 917853 | 691390 | Better |
| 2015 | 917853 | 691390 | 1130746 | 897290 | Worse |
| 2014 | 1130746 | 897290 | 1047476 | 911457 | Same or Mixed |
| 2013 | 1047476 | 911457 | 1108035 | 978904 | Worse |
| 2012 | 1108035 | 978904 | 906424 | 778285 | Better |
| 2011 | 906424 | 778285 | 982577 | 783925 | Worse |
| 2010 | 982577 | 783925 | 844836 | 712731 | Better |
| 2009 | 844836 | 712731 | 773812 | 639333 | Better |
| 2008 | 773812 | 639333 | Null | Null | Same or Mixed |

1. Can you derive more KPIs for the team strategy if possible?

**ANS-**

Deriving more Key Performance Indicators (KPIs) can offer deeper insights into a team's strategy and performance. Here are several additional KPIs that could be useful for evaluating a cricket team's overall performance, strengths, and areas for improvement:

**1. Batting KPIs:**

* **Average Runs per Match**: Measures the average runs scored by the team in each match over a season.

|  |
| --- |
| SELECT  Season\_Year,  AVG(Total\_Runs) AS Average\_Runs\_Per\_Match  FROM (  SELECT  s.Season\_Year,  m.Match\_Id,  SUM(bs.Runs\_Scored) AS Total\_Runs  FROM  season s  JOIN  matches m ON s.Season\_Id = m.Season\_Id  LEFT JOIN  batsman\_scored bs ON m.Match\_Id = bs.Match\_Id  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Replace with actual Team\_Id  GROUP BY  s.Season\_Year, m.Match\_Id  ) AS SeasonPerformance  GROUP BY Season\_Year; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **average runs scored per match** for a specific team (Team\_Id = 1) across different seasons. This serves as a Key Performance Indicator (KPI) for batting performance.
2. **Breaking Down the Query**:
   * **Inner Query (SeasonPerformance)**:
     + Calculates the total runs scored by the team in each match:
       - Filters matches where the team in question is either Team\_1 or Team\_2.
       - Joins the season, matches, and batsman\_scored tables.
       - Groups by Season\_Year and Match\_Id to compute the total runs scored (Total\_Runs) for each match.
     + Outputs Season\_Year, Match\_Id, and Total\_Runs for every match in each season.
   * **Outer Query**:
     + Aggregates the data by season to compute the **average runs per match**:
       - Uses the AVG function on Total\_Runs for each season.
       - Groups by Season\_Year to provide season-level metrics.

**Final Summary**

The query produces a table with:

* Season\_Year: The year of the season.
* Average\_Runs\_Per\_Match: The average number of runs scored by the team in each match during the season.

This metric is essential for assessing the team's batting consistency and overall performance across seasons. It helps identify trends in scoring efficiency and evaluate how the team's batting has evolved.

**Output**

|  |  |
| --- | --- |
| **Season\_Year** | **Average\_Runs\_Per\_Match** |
| 2008 | 258.5385 |
| 2009 | 271.2308 |
| 2010 | 290.8571 |
| 2011 | 257.4000 |
| 2012 | 270.6471 |
| 2013 | 272.0625 |
| 2014 | 294.6250 |
| 2015 | 300.2308 |
| 2016 | 288.8000 |

* **Strike Rate**: Measures the rate at which runs are scored (per 100 balls faced).

|  |
| --- |
| SELECT  p.Player\_Name,  SUM(bs.Runs\_Scored) AS Total\_Runs,  COUNT(bb.Ball\_Id) AS Balls\_Faced,  (SUM(bs.Runs\_Scored) / COUNT(bb.Ball\_Id)) \* 100 AS Strike\_Rate  FROM  player p  JOIN  ball\_by\_ball bb ON p.Player\_Id = bb.Striker -- Assuming the Striker column links to the player  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id AND bb.Over\_Id = bs.Over\_Id AND bb.Ball\_Id = bs.Ball\_Id  LEFT JOIN  extra\_runs er ON bb.Match\_Id = er.Match\_Id AND bb.Over\_Id = er.Over\_Id AND bb.Ball\_Id = er.Ball\_Id  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  WHERE  pm.Team\_Id = 1 -- Replace with actual Team\_Id  AND er.Extra\_Type\_Id IS NULL -- Exclude extra deliveries like wides and no-balls  GROUP BY  p.Player\_Name; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **strike rate** of each player for a specific team (***Team\_Id = 1***) by considering the runs scored and the legitimate balls faced.
2. **Breaking Down the Query**:
   * **Tables and Joins**:
     + ***player***: Contains player details.
     + ***ball\_by\_ball***: Tracks details for every ball, including the *Striker* (batsman facing the ball).
     + ***batsman\_scored***: Provides runs scored on each delivery.
     + ***extra\_runs****:* Identifies extras (like wides or no-balls), which are excluded from legitimate balls faced.
     + ***player\_match***: Links players to their respective teams for specific matches.
   * **Filters**:
     + ***pm.Team\_Id* = 1**: Includes only players from the specified team.
     + ***er.Extra\_Type\_Id IS NULL***: Excludes extra deliveries from the calculation of balls faced.
   * **Metrics Calculation**:
     + ***Total\_Runs***: The sum of runs scored by the batsman from batsman\_scored.
     + ***Balls\_Faced***: The count of legitimate balls faced (total deliveries minus extras).
     + ***Strike\_Rate****:* Calculated as:

Strike Rate=(Total Runs/Balls Faced)×100

* + **Grouping**:
    - Groups the data by ***p.Player\_Name*** to calculate individual performance metrics.

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Player\_Name** | **Total\_Runs** | **Balls\_Faced** | **Strike\_Rate** |
| A Chopra | 693 | 819 | 84.6154 |
| AB Agarkar | 7695 | 6021 | 127.8027 |
| AB Dinda | 1827 | 1624 | 112.5000 |
| AD Mathews | 18960 | 14760 | 128.4553 |
| AD Russell | 27027 | 16011 | 168.8027 |
| AN Ghosh | 38 | 32 | 118.7500 |
| Anureet Singh | 141 | 135 | 104.4444 |
| AS Rajpoot | 64 | 44 | 145.4545 |
| Azhar Mahmood | 734 | 559 | 131.3059 |
| B Lee | 5825 | 3975 | 146.5409 |
| BAW Mendis | 84 | 77 | 109.0909 |
| BB McCullum | 148190 | 117950 | 125.6380 |
| BJ Haddin | 28 | 20 | 140.0000 |
| BJ Hodge | 47063 | 38152 | 123.3566 |
| C Munro | 256 | 208 | 123.0769 |
| CA Lynn | 668 | 548 | 121.8978 |
| CA Pujara | 8010 | 7080 | 113.1356 |
| CH Gayle | 91296 | 64800 | 140.8889 |
| CK Langeveldt | 136 | 72 | 188.8889 |
| DB Das | 18290 | 14012 | 130.5310 |
| DJ Hussey | 57891 | 44666 | 129.6087 |
| EJG Morgan | 37752 | 30316 | 124.5283 |

* **Boundary Percentage**: Calculates the percentage of runs scored through boundaries (4s and 6s). High values indicate aggressive play.

|  |
| --- |
| SELECT  Season\_Year,  SUM(CASE WHEN bs.Runs\_Scored = 4 OR bs.Runs\_Scored = 6 THEN bs.Runs\_Scored ELSE 0 END) / SUM(bs.Runs\_Scored) \* 100 AS Boundary\_Percentage  FROM  season s  JOIN  matches m ON s.Season\_Id = m.Season\_Id  LEFT JOIN  batsman\_scored bs ON m.Match\_Id = bs.Match\_Id  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Replace with actual Team\_Id  GROUP BY Season\_Year; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **Boundary Percentage** for a specific team (Team\_Id = 1) in each season. The boundary percentage indicates the proportion of runs scored from boundaries (fours and sixes) compared to the total runs scored by the team.
2. **Breaking Down the Query**:
   * **Joins**:
     + Joins the season, matches, and batsman\_scored tables to link season and match information with batting performance.
     + Filters matches where the team in question is either Team\_1 or Team\_2.
   * **Boundary Calculation**:
     + **CASE Statement**: Identifies runs scored from boundaries:
       - *bs.Runs\_Scored* = 4 OR bs.Runs\_Scored = 6: Includes only fours and sixes.
       - ELSE 0: Ignores runs scored from other deliveries.
     + **Numerator**: Sums the runs scored from boundaries.
     + **Denominator**: Sums all runs scored (bs.Runs\_Scored).
     + **Percentage Calculation**:

Boundary Percentage=(Boundary Runs/Total Runs)×100

* + **Grouping**:
    - Groups data by *Season\_Year* to compute the boundary percentage for each season.

**Output**

|  |  |
| --- | --- |
| **Season\_year** | **Boundary\_Percentage** |
| 2008 | 62.1839 |
| 2009 | 55.0766 |
| 2010 | 58.0550 |
| 2011 | 53.8203 |
| 2012 | 52.0322 |
| 2013 | 56.0533 |
| 2014 | 56.2580 |
| 2015 | 62.7210 |
| 2016 | 57.1099 |

**2. Bowling KPIs:**

* **Bowling Economy**: Measures the average runs conceded per over bowled. Lower values indicate better performance.

|  |
| --- |
| SELECT  p.Player\_Name,  SUM(bs.Runs\_Scored + COALESCE(er.Extra\_Runs, 0)) / (COUNT(DISTINCT bb.Over\_Id) / 6) AS Economy\_Rate  FROM  player p  LEFT JOIN  ball\_by\_ball bb ON p.Player\_Id = bb.Bowler  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id AND bb.Over\_Id = bs.Over\_Id AND bb.Ball\_Id = bs.Ball\_Id  LEFT JOIN  extra\_runs er ON bb.Match\_Id = er.Match\_Id AND bb.Over\_Id = er.Over\_Id AND bb.Ball\_Id = er.Ball\_Id  WHERE  bb.Bowler IS NOT NULL  AND p.Player\_Id IN (  SELECT pm.Player\_Id  FROM player\_match pm  WHERE pm.Team\_Id = 1 -- Replace with the actual Team\_Id  )  GROUP BY  p.Player\_Name; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **economy rate** for each bowler from a specific team (Team\_Id = 1). The economy rate measures the average number of runs conceded per over bowled.
2. **Breaking Down the Query**:
   * **Tables and Joins**:
     + **player**: Contains player details (e.g., names and roles).
     + **ball\_by\_ball**: Tracks details for every delivery bowled.
     + **batsman\_scored**: Provides the runs scored off each delivery.
     + **extra\_runs**: Captures extra runs conceded (like wides or no-balls).
     + **player\_match**: Links players to their respective teams and matches.
   * **Filters**:
     + bb.Bowler IS NOT NULL: Includes only deliveries bowled by a player who is a bowler.
     + p.Player\_Id IN (SELECT pm.Player\_Id FROM player\_match pm WHERE pm.Team\_Id = 1): Filters for bowlers belonging to the specified team.
   * **Runs Conceded**:
     + bs.Runs\_Scored + COALESCE(er.Extra\_Runs, 0): Adds the runs scored by batsmen (bs.Runs\_Scored) and extra runs (er.Extra\_Runs). COALESCE ensures null values in er.Extra\_Runs are treated as 0.
   * **Overs Bowled**:
     + (COUNT(DISTINCT bb.Over\_Id) / 6): Calculates the total overs bowled by counting distinct over IDs and dividing by 6 (assuming 6-ball overs).
   * **Grouping**:
     + Groups by p.Player\_Name to calculate individual economy rates for each bowler.
3. **Output:**

|  |  |  |
| --- | --- | --- |
| Player\_Name | | Economy\_Rate |
| AB Agarkar | 666.9000 | |
| AB Dinda | 1113.3000 | |
| AD Mathews | 685.3333 | |
| AD Russell | 533.4000 | |
| Anureet Singh | 329.6842 | |
| AS Rajpoot | 122.0000 | |
| Azhar Mahmood | 437.7000 | |
| B Lee | 748.6667 | |
| BAW Mendis | 224.8235 | |
| BJ Hodge | 324.0000 | |
| C Munro | 168.0000 | |
| CH Gayle | 465.0000 | |
| CK Langeveldt | 179.5385 | |
| DJ Hussey | 314.8235 | |
| GB Hogg | 450.0000 | |
| I Sharma | 1220.7000 | |
| Iqbal Abdulla | 649.8000 | |
| J Botha | 515.1000 | |
| JD Unadkat | 600.6000 | |
| JH Kallis | 1385.4000 | |
| JO Holder | 350.7273 | |
| JW Hastings | 119.1429 | |
| KC Cariappa | 175.5000 | |

* **Average Wickets per Match**: Measures the average number of wickets taken per match.

|  |
| --- |
| SELECT  Season\_Year,  AVG(Total\_Wickets) AS Average\_Wickets\_Per\_Match  FROM (  SELECT  s.Season\_Year,  m.Match\_Id,  COUNT(bb.Bowler) AS Total\_Wickets  FROM  season s  JOIN  matches m ON s.Season\_Id = m.Season\_Id  LEFT JOIN  ball\_by\_ball bb ON m.Match\_Id = bb.Match\_Id  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Replace with actual Team\_Id  GROUP BY  s.Season\_Year, m.Match\_Id  ) AS SeasonPerformance  GROUP BY Season\_Year; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **average number of wickets taken per match** by a specific team (Team\_Id = 1) for each season. This helps assess the team's bowling performance across seasons.
2. **Breaking Down the Query**:
   * **Subquery**:
     + Retrieves the total wickets taken by the team in each match for every season.
     + **Tables**:
       - **season**: Provides the year (Season\_Year).
       - **matches**: Links matches to their corresponding seasons.
       - **ball\_by\_ball**: Tracks each ball's bowler, enabling the calculation of wickets.
     + **Filters**:
       - m.Team\_1 = 1 OR m.Team\_2 = 1: Ensures only matches involving the specified team are included.
     + **Calculation**:
       - COUNT(bb.Bowler): Counts instances of the Bowler column in the ball\_by\_ball table, representing the total wickets taken in a match.
     + **Grouping**:
       - Groups by s.Season\_Year (season) and m.Match\_Id (individual match) to compute match-level performance.
   * **Outer Query**:
     + Aggregates the match-level performance to calculate season-level performance.
     + **Metric**:
       - AVG(Total\_Wickets): Computes the average wickets per match for each season.
     + **Grouping**:
       - Groups by Season\_Year to report the average for every season.
3. **Output**:

The query generates a table with:

* Season\_Year: The year of the season.
* Average\_Wickets\_Per\_Match: The average number of wickets taken per match by the team in that season.

|  |  |
| --- | --- |
| **Season\_Year** | **Average\_Wicket\_Per\_Match** |
| 2008 | 223.6154 |
| 2009 | 237.2308 |
| 2010 | 240.6429 |
| 2011 | 229.6667 |
| 2012 | 238.7059 |
| 2013 | 240.3125 |
| 2014 | 239.3125 |
| 2015 | 232.0769 |
| 2016 | 229.8000 |

* **Dot Ball Percentage**: Represents the percentage of dot balls bowled. Higher percentages indicate more effective bowling.

|  |
| --- |
| SELECT  s.Season\_Year,  (SUM(CASE WHEN bs.Runs\_Scored = 0 AND COALESCE(er.Extra\_Runs, 0) = 0 THEN 1 ELSE 0 END) / COUNT(\*)) \* 100 AS Dot\_Ball\_Percentage  FROM  season s  JOIN  matches m ON s.Season\_Id = m.Season\_Id  LEFT JOIN  ball\_by\_ball bb ON m.Match\_Id = bb.Match\_Id  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id AND bb.Over\_Id = bs.Over\_Id AND bb.Ball\_Id = bs.Ball\_Id  LEFT JOIN  extra\_runs er ON bb.Match\_Id = er.Match\_Id AND bb.Over\_Id = er.Over\_Id AND bb.Ball\_Id = er.Ball\_Id  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Replace with the actual Team\_Id  GROUP BY  s.Season\_Year; |

**Explanation of the Query**

1. **Purpose**:  
   The query calculates the **average number of wickets taken per match** by a specific team (Team\_Id = 1) for each season. This helps assess the team's bowling performance across seasons.
2. **Breaking Down the Query**:
   * **Subquery**:
     + Retrieves the total wickets taken by the team in each match for every season.
     + **Tables**:
       - **season**: Provides the year (Season\_Year).
       - **matches**: Links matches to their corresponding seasons.
       - **ball\_by\_ball**: Tracks each ball's bowler, enabling the calculation of wickets.
     + **Filters**:
       - m.Team\_1 = 1 OR m.Team\_2 = 1: Ensures only matches involving the specified team are included.
     + **Calculation**:
       - COUNT(bb.Bowler): Counts instances of the Bowler column in the ball\_by\_ball table, representing the total wickets taken in a match.
     + **Grouping**:
       - Groups by s.Season\_Year (season) and m.Match\_Id (individual match) to compute match-level performance.
   * **Outer Query**:
     + Aggregates the match-level performance to calculate season-level performance.
     + **Metric**:
       - AVG(Total\_Wickets): Computes the average wickets per match for each season.
     + **Grouping**:
       - Groups by Season\_Year to report the average for every season.
3. **Output**:

The query generates a table with:

* Season\_Year: The year of the season.
* Average\_Wickets\_Per\_Match: The average number of wickets taken per match by the team in that season.

|  |  |
| --- | --- |
| **Season\_Year** | **Dot\_Ball\_Percentage** |
| 2008 | 37.9694 |
| 2009 | 36.9927 |
| 2010 | 32.9533 |
| 2011 | 35.6359 |
| 2012 | 33.0068 |
| 2013 | 36.5880 |
| 2014 | 33.7274 |
| 2015 | 33.9150 |
| 2016 | 32.1367 |

**3. Fielding KPIs:**

* **Catch Success Rate**: Measures the effectiveness of the fielding unit by evaluating how many catches were successfully taken.

|  |
| --- |
| SELECT  p.Player\_Name,  (SUM(CASE WHEN ot.Out\_Name = 'Caught' THEN 1 ELSE 0 END) / COUNT(CASE WHEN bb.Ball\_Id IS NOT NULL THEN 1 ELSE NULL END)) \* 100 AS Catch\_Success\_Rate  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id -- Links players to their team and matches  LEFT JOIN  ball\_by\_ball bb ON pm.Match\_Id = bb.Match\_Id  LEFT JOIN  out\_type ot ON bb.Match\_Id = pm.Match\_Id AND ot.Out\_Name = 'Caught'  WHERE  pm.Team\_Id = 1 -- Replace with the actual team ID  GROUP BY  p.Player\_Name; |

**Explanation of the Query**

1. **Purpose**:  
   This query calculates the **Catch Success Rate** for players of a specific team (Team\_Id = 1). The metric measures the percentage of successful catches (dismissals categorized as 'Caught') relative to the total number of deliveries they were involved in as fielders.
2. **Breaking Down the Query**:
   * **Tables Used**:
     + **player**: Contains information about players (Player\_Name).
     + **player\_match**: Links players to the matches they played and their team affiliations.
     + **ball\_by\_ball**: Tracks every ball bowled in a match.
     + **out\_type**: Specifies the type of dismissal for each wicket, e.g., Caught.
   * **Joins**:
     + **player → player\_match**: Connects players to their match participation.
     + **player\_match → ball\_by\_ball**: Links players to specific deliveries in matches.
     + **out\_type**: Identifies dismissals classified as Caught.
   * **Filters**:
     + pm.Team\_Id = 1: Limits data to players from the specified team.
     + ot.Out\_Name = 'Caught': Filters dismissal data to catches.
   * **Calculations**:
     + **Numerator**:
       - SUM(CASE WHEN ot.Out\_Name = 'Caught' THEN 1 ELSE 0 END):
         * Counts the number of dismissals classified as 'Caught' attributed to the player.
     + **Denominator**:
       - COUNT(CASE WHEN bb.Ball\_Id IS NOT NULL THEN 1 ELSE NULL END):
         * Counts the total number of deliveries the player was involved in. Ensures null-safe counting.
     + **Catch Success Rate**: Catch Success Rate=(Total Catches/Total Deliveries Involved)×100
   * **Grouping**:
     + Groups data by p.Player\_Name to compute the catch success rate for each player.
3. **Output**:

The query produces a table with:

* Player\_Name: The name of the player.
* *Catch\_Success\_Rate:* The percentage of deliveries resulting in successful catches attributed to the player.

|  |  |
| --- | --- |
| **Player\_Name** | **Catch\_Success\_Rate** |
| A Chopra | 100.0000 |
| AB Agarkar | 100.0000 |
| AB Dinda | 100.0000 |
| AD Mathews | 100.0000 |
| AD Russell | 100.0000 |
| AN Ghosh | 100.0000 |
| Anureet Singh | 100.0000 |
| AS Rajpoot | 100.0000 |
| Azhar Mahmood | 100.0000 |
| B Lee | 100.0000 |

* **Run-Out Contribution**: Evaluates how many run-outs a fielder was directly involved in.

|  |
| --- |
| SELECT  p.Player\_Name,  COUNT(\*) AS Run\_Outs  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id -- Link players to their teams via matches  JOIN  ball\_by\_ball bb ON bb.Match\_Id = pm.Match\_Id -- Ensure the player participated in the match  JOIN  out\_type ot ON bb.Ball\_Id = ot.Out\_Id -- Assuming this connects run-outs to a type of out  WHERE  pm.Team\_Id = 1 -- Replace with the actual Team\_Id  AND ot.Out\_Name = 'Run Out'  GROUP BY  p.Player\_Name; |

**Explanation of the Query**

**Purpose**

The query calculates the total number of **run-out dismissals** associated with each player on a specified team (Team\_Id = 1). It provides a breakdown of run-outs executed by each player.

**Detailed Breakdown**

1. **Tables Used**:
   * **player**: Contains player details such as Player\_Name and Player\_Id.
   * **player\_match**: Links players to their respective matches and teams.
   * **ball\_by\_ball**: Records data for every ball in a match.
   * **out\_type**: Specifies the type of dismissals, such as Run Out, Caught, etc.
2. **Joins**:
   * **player → player\_match**:
     + Links players to matches they played in (Player\_Id).
   * **player\_match → ball\_by\_ball**:
     + Ensures the analysis is limited to deliveries during the matches the player participated in.
   * **ball\_by\_ball → out\_type**:
     + Connects deliveries to the type of dismissal that occurred.
3. **Filters**:
   * pm.Team\_Id = 1:
     + Limits the query to players of the specified team (Team\_Id = 1).
   * ot.Out\_Name = 'Run Out':
     + Ensures only dismissals categorized as Run Out are included.
4. **Aggregation**:
   * COUNT(\*) AS Run\_Outs:
     + Counts the total number of run-outs linked to each player, grouped by their name.
5. **Grouping**:
   * GROUP BY p.Player\_Name:
     + Groups the results by player name to compute the number of run-outs executed by each player.

**Output**

The query generates a table with:

* **Player\_Name**: Name of the player.
* **Run\_Outs**: Total number of run-outs attributed to the player.

|  |  |
| --- | --- |
| **Player\_Name** | **Run\_Outs** |
| SC Ganguly | 1498 |
| BB McCullum | 1325 |
| RT Ponting | 149 |
| DJ Hussey | 853 |
| Mohammad Hafeez | 275 |
| WP Saha | 1244 |
| LR Shukla | 1571 |
| AB Agarkar | 1013 |
| M Kartik | 987 |
| I Sharma | 1152 |
| AB Dinda | 1065 |
| DB Das | 1189 |
| Salman Butt | 243 |
| BJ Hodge | 724 |
| Umar Gul | 228 |
| A Chopra | 233 |
| T Taibu | 110 |
| Iqbal Abdulla | 1243 |
| Shoaib Akhtar | 87 |
| BAW Mendis | 273 |
| CH Gayle | 603 |

**4. Team Strategy KPIs:**

* **Win Percentage by Batting First or Chasing**: Shows how the team performs when batting first compared to chasing a target. Can reveal if the team is better suited for one strategy over the other.

|  |
| --- |
| SELECT  CASE  WHEN m.Toss\_Decide = 1 AND m.Toss\_Winner = m.Match\_Winner THEN 'Batting First'  ELSE 'Chasing'  END AS Strategy,  COUNT(\*) AS Total\_Matches,  SUM(CASE  WHEN (m.Toss\_Decide = 1 AND m.Toss\_Winner = m.Match\_Winner AND m.Match\_Winner = 1) OR  (m.Toss\_Decide = 2 AND m.Toss\_Winner != m.Match\_Winner AND m.Match\_Winner = 1)  THEN 1 ELSE 0  END) AS Matches\_Won,  (SUM(CASE  WHEN (m.Toss\_Decide = 1 AND m.Toss\_Winner = m.Match\_Winner AND m.Match\_Winner = 1) OR  (m.Toss\_Decide = 2 AND m.Toss\_Winner != m.Match\_Winner AND m.Match\_Winner = 1)  THEN 1 ELSE 0  END) / COUNT(\*)) \* 100 AS Win\_Percentage  FROM  matches m  WHERE  m.Team\_1 = 1 OR m.Team\_2 = 1 -- Replace with actual Team\_Id  GROUP BY Strategy; |

**Explanation of the Query**

**Purpose**

The query analyzes a team's performance based on the strategy (either "Batting First" or "Chasing") they adopted after winning the toss. The analysis includes:

1. Total number of matches played by the team.
2. Number of matches won when using a specific strategy.
3. Win percentage for each strategy.

**Detailed Breakdown**

1. **Tables Used:**
   * *matches:* Contains details about each match, including toss decisions, winners, and teams involved.
2. **Fields Used:**
   * *Toss\_Decide:* Indicates the decision made by the team that won the toss (1 for "Bat" and 2 for "Bowl").
   * *Toss\_Winner:* The team that won the toss.
   * *Match\_Winner*: The team that won the match.
   * *Team\_1, Team\_2*: Represents the two teams involved in the match.
3. **Logic for Strategy Classification:**
   * The query classifies the strategy as either "Batting First" or "Chasing":
     + "Batting First": This is the strategy when:
       - The team wins the toss and chooses to bat (i.e., *Toss\_Decide = 1* and *Toss\_Winner = Match\_Winner*).
     + "Chasing": This is the strategy when:
       - The team wins the toss and chooses to bowl (i.e., *Toss\_Decide = 2*), or the team wins the toss and loses the match, meaning they are chasing.
4. **Calculation of Matches Won:**
   * **Matches\_Won:**
     + The query counts matches won by the team when they followed either of the strategies.
     + A match is counted as won if:
       - The team chose to bat first (*Toss\_Decide = 1* and *Toss\_Winner = Match\_Winner*) and won.
       - The team chose to bowl first (*Toss\_Decide = 2* and *Toss\_Winner != Match\_Winner*) and still won the match.
5. **Win Percentage Calculation:**
   * **Win\_Percentage:**
     + The win percentage is calculated for each strategy by dividing the total number of matches won by the total number of matches played for each strategy, then multiplying by 100:
   * This calculation is done for each group (Strategy), either "Batting First" or "Chasing".
6. **Grouping:**
   * **GROUP BY Strategy:**
     + The results are grouped by the *Strategy* (either "Batting First" or "Chasing"). This ensures that the performance (number of wins and win percentage) is calculated separately for each strategy.

**Filters:**

* *WHERE m.Team\_1 = 1 OR m.Team\_2 = 1:*
  + This filter ensures that only matches involving Team 1 (replace 1 with the actual team ID) are considered in the analysis.

**Final Output**

The output will include the following columns:

* *Strategy*: Whether the team chose to bat first or bowl first.
* *Total\_Matches*: The total number of matches played by the team under that strategy.
* *Matches\_Won*: The total number of matches won by the team when using that strategy.
* *Win\_Percentage*: The percentage of matches won by the team when following that strategy.

|  |  |  |  |
| --- | --- | --- | --- |
| **Strategy** | **Total\_Matches** | **Matches\_Won** | **Win\_Percentage** |
| Chasing | 89 | 18 | 20.2247 |
| Batting First | 43 | 23 | 53.4884 |

1. Using SQL, write a query to find out the average wickets taken by each bowler in each venue. Also, rank the gender according to the average value.

ANS-

|  |
| --- |
| WITH each\_venue\_BowlerWickets AS (  SELECT  bb.Bowler,  p.Player\_Name,  v.venue\_name,  COUNT(DISTINCT bb.Match\_Id, bb.Over\_Id, bb.Ball\_Id, bb.Innings\_No) AS Total\_Wickets,  COUNT(DISTINCT bb.Match\_Id, bb.Innings\_No) AS Matches\_Played  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  JOIN  matches m ON pm.Match\_Id = m.Match\_Id  JOIN  ball\_by\_ball bb ON m.Match\_Id = bb.Match\_Id AND pm.Player\_Id = bb.Bowler  JOIN  wicket\_taken wt ON bb.Match\_Id = wt.Match\_Id AND bb.Over\_Id = wt.Over\_Id  AND bb.Ball\_Id = wt.Ball\_Id AND bb.Innings\_No = wt.Innings\_No  JOIN  venue v ON m.Venue\_Id = v.Venue\_Id  GROUP BY  bb.Bowler, p.Player\_Name, v.Venue\_Name  )  SELECT  Bowler AS Player\_Id,  Player\_Name,  Venue\_Name,  ROUND(Total\_Wickets / Matches\_Played, 1) AS Average\_Wickets,  DENSE\_RANK() OVER (ORDER BY ROUND(Total\_Wickets / Matches\_Played, 1) DESC) AS 'Rank'  FROM  each\_venue\_BowlerWickets  ORDER BY  Average\_Wickets DESC, 'Rank' DESC; |

**Explanation of the Query**

**1. Common Table Expression (CTE): each\_venue\_BowlerWickets**

* **Purpose**:
  + The CTE is used to gather statistics on each bowler's performance at different venues. It calculates the total wickets taken and the number of matches played by each bowler at each venue.
* **Tables Involved**:
  + **player (p)**: Contains player details (such as Player\_Name).
  + **player\_match (pm)**: Links players to the matches they have played.
  + **matches (m)**: Contains details about the matches, including the venue.
  + **ball\_by\_ball (bb)**: Contains the ball-by-ball data for each match.
  + **wicket\_taken (wt)**: Contains information on the wickets taken, linking it to the ball-by-ball data.
  + **venue (v)**: Contains details about the venues where matches were held.
* **Logic**:
  + **COUNT(DISTINCT bb.Match\_Id, bb.Over\_Id, bb.Ball\_Id, bb.Innings\_No)**: Counts the number of wickets taken (distinct combinations of match, over, ball, and innings).
  + **COUNT(DISTINCT bb.Match\_Id, bb.Innings\_No)**: Counts the number of matches played by the bowler at each venue.

The data is grouped by the bowler, player name, and venue.

**2. Main Query**

* **SELECT**:
  + **Bowler AS Player\_Id**: The bowler's ID (represented by the Bowler column in the ball-by-ball data).
  + **Player\_Name**: The name of the player.
  + **Venue\_Name**: The name of the venue where the bowler played.
  + **ROUND(Total\_Wickets / Matches\_Played, 1) AS Average\_Wickets**: The average number of wickets taken per match by the bowler at each venue, rounded to one decimal place.
  + **DENSE\_RANK() OVER (ORDER BY ROUND(Total\_Wickets / Matches\_Played, 1) DESC)**: This assigns a rank to each bowler based on their average wickets per match, ordered in descending order of their performance. DENSE\_RANK() ensures that no ranks are skipped (i.e., if two bowlers have the same average, they will have the same rank, and the next rank will be the next number).
* **ORDER BY**:
  + The results are sorted by Average\_Wickets in descending order, and in case of ties, the Rank column is used for secondary sorting in descending order.

**Output**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player\_Id** | **Player\_Name** | **Venue\_Name** | **Average\_Wickets** | **RANK** |
| 362 | DJG Sammy | Punjab Cricket Association Stadium, Mohali | 6.0 | 1 |
| 35 | RA Jadeja | Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium | 5.0 | 2 |
| 84 | I Sharma | Nehru Stadium | 5.0 | 2 |
| 104 | BJ Hodge | Holkar Cricket Stadium | 5.0 | 2 |
| 124 | A Kumble | Newlands | 5.0 | 2 |
| 137 | AD Mascarenhas | Punjab Cricket Association Stadium, Mohali | 5.0 | 2 |
| 175 | DL Vettori | Newlands | 5.0 | 2 |
| 310 | JP Faulkner | Rajiv Gandhi International Stadium, Uppal | 5.0 | 2 |
| 35 | RA Jadeja | Dubai International Cricket Stadium | 4.0 | 3 |
| 38 | SK Warne | Vidarbha Cricket Association Stadium, Jamtha | 4.0 | 3 |
| 39 | SK Trivedi | Vidarbha Cricket Association Stadium, Jamtha | 4.0 | 3 |
| 44 | ST Jayasuriya | Eden Gardens | 4.0 | 3 |
| 57 | RG Sharma | SuperSport Park | 4.0 | 3 |
| 71 | DJ Bravo | Punjab Cricket Association IS Bindra Stadium, Mohali | 4.0 | 3 |

1. Which players have consistently performed well in past seasons? (will you use any visualization to solve the problem)

ANS-

|  |
| --- |
| SELECT  p.Player\_Id,  p.Player\_Name,  s.Season\_Year,  SUM(bs.Runs\_Scored) AS Total\_Runs  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  JOIN  matches m ON pm.Match\_Id = m.Match\_Id  JOIN  batsman\_scored bs ON m.Match\_Id = bs.Match\_Id  JOIN  season s ON m.Season\_Id = s.Season\_Id -- Join with season table  WHERE  pm.Role\_Id = 1 -- Assuming Role\_Id for batsmen is 1; adjust if necessary  GROUP BY  p.Player\_Id, p.Player\_Name, s.Season\_Year  ORDER BY  p.Player\_Name, s.Season\_Year; |

Players like **V. Kohli** and **G. Gambhir** consistently improved their performance over the past seasons. In contrast, players such as **DPMD Jayawardene**, **V. Sehwag**, and **SR Tendulkar** had fluctuating performances with periods of high runs and declines.

**Analysis of Each Player's Performance**

1. **V. Kohli**
   * The total runs show a generally increasing trend from 2011 to 2016, with significant peaks in 2013 and 2016.
   * Kohli's performance consistently improved, especially with a notable peak in 2016.
2. **G. Gambhir**
   * Gambhir’s performance shows fluctuations but mostly an upward trend from 2009 to 2016. He had a peak in 2014, followed by a slight dip and then an increase in 2016.
   * Overall, Gambhir had consistent performances, with some variations but a positive trend across seasons.
3. **DPMD Jayawardene**
   * Jayawardene had a strong peak in 2011, but performances were not consistently increasing. There were significant drops in 2010 and 2012.
   * Not a consistent upward trend; his performance varied across seasons.
4. **V. Sehwag**
   * Sehwag's performance fluctuated significantly, with a peak in 2012. However, there was a steep decline by 2015.
   * Sehwag's performance was not consistently increasing; it had high peaks and notable declines.
5. **SR Tendulkar**
   * Tendulkar showed a consistent increase from 2008 to 2010, peaking in 2010. There was a slight drop in 2011.
   * Tendulkar's performance improved initially but did not show consistent growth beyond 2010.

**Visualization in Excel**

To better understand and present these trends, you can use the following chart types:

1. **Column Chart**:
   * We can use column charts to compare the total runs for each player by season.
   * This can highlight the peaks and drops for different players in the same visual.

These visualizations can effectively communicate which players have had consistent performance improvements and who had fluctuating results over the seasons.

1. Are there players whose performance is more suited to specific venues or conditions? (how would you present this using charts?)

ANS-

* Based on the below 2 points analysed this question.

1. The top 3 players in each venue scored the highest runs
2. The top 3 players in each venue took the highest wicket

* **Query**: The top 3 players in each venue scored the highest runs.

|  |
| --- |
| WITH result AS (  SELECT  p.Player\_Id,  p.Player\_Name,  v.Venue\_Name,  SUM(bs.Runs\_Scored) AS Total\_Runs  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  JOIN  ball\_by\_ball bb ON pm.Match\_Id = bb.Match\_Id AND pm.Player\_Id = bb.Striker  JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id AND bb.Innings\_No = bs.Innings\_No  JOIN  matches m ON bs.Match\_Id = m.Match\_Id  JOIN  venue v ON m.Venue\_Id = v.Venue\_Id  JOIN  season s ON m.Season\_Id = s.Season\_Id  JOIN  city c ON v.City\_Id = c.City\_Id  JOIN  country cc ON c.Country\_Id = cc.Country\_Id  WHERE  cc.Country\_Name = 'India'  GROUP BY  p.Player\_Id, p.Player\_Name, v.Venue\_Name  ORDER BY  Total\_Runs DESC  ),  rank\_player AS (  SELECT  \*,  DENSE\_RANK() OVER (PARTITION BY Venue\_Name ORDER BY Total\_Runs DESC) AS Top\_Rank  FROM  result  )  SELECT \*  FROM  rank\_player  WHERE  Top\_Rank BETWEEN 1 AND 3  ORDER BY  Top\_Rank, Total\_Runs DESC; |

* **Explanation:**
* This SQL query identifies the top three players who scored the highest runs at each venue in India. Here's a step-by-step breakdown of the process:

1. **Calculate Total Runs by Player at Each Venue:**
   * The result CTE aggregates the total runs scored by each player at each venue. It joins multiple tables, including player, player\_match, ball\_by\_ball, and batsman\_scored, to link players with their performances in matches played at specific venues. The query filters for matches held in India and groups the results by player and venue.
2. **Rank Players by Runs Scored:**
   * The rank\_player CTE ranks the players within each venue based on their total runs using the DENSE\_RANK() function. This creates a ranking for the players at each venue, allowing for ties to be handled appropriately.
3. **Select Top 3 Players:**
   * The final query retrieves players who rank in the top three for runs scored at each venue by filtering the results where top\_rank is between 1 and 3.
4. **Order the Results:**
   * The output is ordered by venue name and the ranking of the players, ensuring a clear presentation of the top scorers at each venue.

* **Output:**
* Original output is too big to paste here.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player\_Id** | **Player\_Name** | **Venue\_Name** | **totals\_Runs** | **Top\_Rank** |
| 8 | V Kohli | M Chinnaswamy Stadium | 1759 | 1 |
| 21 | SK Raina | MA Chidambaram Stadium, Chepauk | 1302 | 1 |
| 57 | RG Sharma | Wankhede Stadium | 1235 | 1 |
| 40 | G Gambhir | Eden Gardens | 1149 | 1 |
| 41 | V Sehwag | Feroz Shah Kotla | 933 | 1 |
| 42 | S Dhawan | Rajiv Gandhi International Stadium, Uppal | 897 | 1 |
| 32 | SR Watson | Sawai Mansingh Stadium | 836 | 1 |
| 100 | SE Marsh | Punjab Cricket Association Stadium, Mohali | 806 | 1 |
| 46 | RV Uthappa | Subrata Roy Sahara Stadium | 440 | 1 |
| 100 | SE Marsh | Himachal Pradesh Cricket Association Stadium | 334 | 1 |
| 85 | AM Rahane | Sardar Patel Stadium, Motera | 308 | 1 |
| 185 | M Vijay | Punjab Cricket Association IS Bindra Stadium, Mohali | 260 | 1 |
| 133 | SR Tendulkar | Brabourne Stadium | 260 | 1 |
| 42 | S Dhawan | Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium | 259 | 1 |
| 306 | SPD Smith | Maharashtra Cricket Association Stadium | 226 | 1 |

* **Query**: top 3 players in each venue taken highest wicket.

|  |
| --- |
| WITH result AS (  SELECT  p.Player\_Id,  p.Player\_Name,  v.Venue\_Name,  COUNT(DISTINCT wt.Match\_Id, wt.Innings\_No, wt.Over\_Id, wt.Ball\_Id) AS Total\_Wickets  FROM  player p  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  JOIN  ball\_by\_ball bb ON pm.Match\_Id = bb.Match\_Id AND pm.Player\_Id = bb.Bowler  JOIN  wicket\_taken wt ON bb.Match\_Id = wt.Match\_Id AND bb.Over\_Id = wt.Over\_Id  AND bb.Ball\_Id = wt.Ball\_Id AND bb.Innings\_No = wt.Innings\_No  JOIN  matches m ON wt.Match\_Id = m.Match\_Id  JOIN  venue v ON m.Venue\_Id = v.Venue\_Id  JOIN  season s ON m.Season\_Id = s.Season\_Id  JOIN  city c ON v.City\_Id = c.City\_Id  JOIN  country cc ON c.Country\_Id = cc.Country\_Id  WHERE  cc.Country\_Name = 'India'  GROUP BY  p.Player\_Id, p.Player\_Name, v.Venue\_Name  ORDER BY  Total\_Wickets DESC  ),  rank\_player AS (  SELECT  \*,  DENSE\_RANK() OVER (PARTITION BY Venue\_Name ORDER BY Total\_Wickets DESC) AS Top\_Rank  FROM  Result  )  SELECT \*  FROM  rank\_player  WHERE  Top\_Rank BETWEEN 1 AND 3  ORDER BY  Top\_Rank, Total\_Wickets DESC; |

* **Explanation:**

This SQL query retrieves the top three wicket-takers at each cricket venue in India. Here’s a breakdown of how it works:

1. Count Wickets Taken:
   * The result Common Table Expression (CTE) counts the distinct wickets taken by each player at various venues. It joins several tables, including player, player\_match, ball\_by\_ball, wicket\_taken, matches, venue, season, city, and country.
   * The query filters for matches played in India and groups the results by player and venue, counting the distinct combinations of match ID, innings number, over ID, and ball ID to accurately reflect the number of wickets taken.
2. Ranking Players:
   * The rank\_player CTE assigns a rank to each player based on their total wickets at each venue using the DENSE\_RANK() function. This function creates a ranking within each venue based on the total wickets, allowing for ties.
3. Selecting Top 3 Wicket-Takers:

* The final selection retrieves players ranked in the top three for each venue, ordering the results first by rank and then by total wickets taken in descending order
* Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player\_ID** | **Player\_Name** | **Venue\_Name** | **Total\_Wickets** | **top\_rank** |
| 194 | SL Malinga | Wankhede Stadium | 57 | 1 |
| 201 | R Ashwin | MA Chidambaram Stadium, Chepauk | 47 | 1 |
| 39 | SK Trivedi | Sawai Mansingh Stadium | 41 | 1 |
| 315 | SP Narine | Eden Gardens | 37 | 1 |
| 136 | A Mishra | Feroz Shah Kotla | 37 | 1 |
| 232 | UT Yadav | Feroz Shah Kotla | 37 | 1 |
| 81 | R Vinay Kumar | M Chinnaswamy Stadium | 32 | 1 |
| 136 | A Mishra | Rajiv Gandhi International Stadium, Uppal | 31 | 1 |
| 67 | PP Chawla | Punjab Cricket Association Stadium, Mohali | 24 | 1 |
| 67 | PP Chawla | Himachal Pradesh Cricket Association Stadium | 13 | 1 |
| 235 | R Sharma | Subrata Roy Sahara Stadium | 13 | 1 |
| 299 | B Kumar | Subrata Roy Sahara Stadium | 13 | 1 |

**SUBJECTIVE**

1. How does the toss decision affect the result of the match? (which visualisations could be used to present your answer better) And is the impact limited to only specific venues?

ANS-

* Key points
  + 1. Toss win match win and toss win match loss of all teams.
    2. Toss win and match win percentage of each team.
* **Query:** Toss win match win and toss win match loss of all teams.

|  |
| --- |
| -- toss win match\_win vs toss win match loss of all team  WITH union\_table\_1 AS (  SELECT 1 AS number,  COUNT(CASE WHEN team\_1 = toss\_winner AND toss\_winner = match\_winner THEN 1 END) AS toss\_win\_match\_win  FROM matches  UNION ALL  SELECT 1 AS number,  COUNT(CASE WHEN team\_2 = toss\_winner AND toss\_winner = match\_winner THEN 1 END) AS toss\_win\_match\_win  FROM matches  ),  rs1 AS (  SELECT number,  SUM(toss\_win\_match\_win) AS toss\_win\_match\_win  FROM union\_table\_1  GROUP BY number  ),  union\_table\_2 AS (  SELECT 1 AS number,  COUNT(CASE WHEN team\_1 = toss\_winner AND toss\_winner <> match\_winner THEN 1 END) AS toss\_win\_match\_loss  FROM matches  UNION ALL  SELECT 1 AS number,  COUNT(CASE WHEN team\_2 = toss\_winner AND toss\_winner <> match\_winner THEN 1 END) AS toss\_win\_match\_loss  FROM matches  ),  rs2 AS (  SELECT number,  SUM(toss\_win\_match\_loss) AS toss\_win\_match\_loss  FROM union\_table\_2  GROUP BY number  )  SELECT rs1.\*,  rs2.toss\_win\_match\_loss  FROM rs1  JOIN rs2 ON rs1.number = rs2.number; |

The query is broken down into multiple steps using **Common Table Expressions (CTEs)**. Each CTE handles a different part of the process, making the query easier to read and understand.

* **Counting Toss Winner Wins**:
  + The first part of the query focuses on determining how many times the team that won the toss also went on to win the match.
  + This is done separately for two teams (team 1 and team 2) for each match.
  + The results are combined and summed up to get the total number of toss winners who also won the match.
* **Counting Toss Winner Losses**:
  + The second part calculates how many times the team that won the toss ended up losing the match.
  + Similarly, this is done separately for both teams (team 1 and team 2) for each match.
  + The results are combined and summed up to get the total number of toss winners who lost the match.
* **Combining the Results**:
  + After calculating the wins and losses, the query combines both results using a join.
  + It links the counts of toss winners winning and losing using a common identifier (number), which is always set to 1.

**Final Output**

The final output of the query will be a single row with two key pieces of information:

* The **total count of toss winners who won the match**.
* The **total count of toss winners who lost the match**.

|  |  |  |
| --- | --- | --- |
| **number** | **toss\_win\_match\_win** | **toss\_win\_match\_loss** |
| 1 | 291 | 283 |

* **Query: Toss wins and match win percentage of each team.**

|  |
| --- |
| WITH match\_winner\_count AS (  SELECT  toss\_winner,  COUNT(\*) AS match\_win\_count  FROM  matches  WHERE  toss\_winner = match\_winner  GROUP BY  toss\_winner  ORDER BY  match\_win\_count DESC  ),  toss\_winner\_count AS (  SELECT  toss\_winner,  COUNT(\*) AS total\_toss\_win\_count  FROM  matches  GROUP BY  toss\_winner  )  SELECT  t.team\_name,  mwc.match\_win\_count,  twc.total\_toss\_win\_count,  ROUND((mwc.match\_win\_count / twc.total\_toss\_win\_count) \* 100, 1) AS toss\_win\_match\_win\_percentage  FROM  match\_winner\_count mwc  JOIN  toss\_winner\_count twc  ON mwc.toss\_winner = twc.toss\_winner  JOIN  team t  ON mwc.toss\_winner = t.team\_id  ORDER BY  toss\_win\_match\_win\_percentage DESC; |

**Explanation**

1. **match\_winner\_count CTE**:
   * This Common Table Expression (CTE) calculates how many matches a team won when they also won the toss.
   * It groups data by toss\_winner and counts matches where toss\_winner = match\_winner.
   * The result includes toss\_winner and match\_win\_count.
2. **toss\_winner\_count CTE**:
   * This CTE calculates the total number of tosses won by each team.
   * It groups data by toss\_winner and counts all matches won during the toss.
3. **Main Query**:
   * Combines data from the two CTEs and the team table:
     + match\_winner\_count (mwc) is joined with toss\_winner\_count (twc) on the toss\_winner column.
     + This ensures each team’s toss win count and toss win + match win count are available for analysis.
     + Another join is made with the team table to fetch the team name (t.team\_name).
4. **Columns Selected**:
   * t.team\_name: Name of the team.
   * mwc.match\_win\_count: Number of matches a team won after winning the toss.
   * twc.total\_toss\_win\_count: Total tosses won by the team.
   * toss\_win\_match\_win\_percentage: The percentage of matches won after winning the toss, calculated as: toss\_win\_match\_win\_percentage=(match\_win\_count/total\_toss\_win\_count)×100
5. **Order**:
   * Results are ordered by toss\_win\_match\_win\_percentage in descending order to highlight teams with the highest success rate.
6. **Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **team\_name** | **match\_win\_out** | **Total\_toss\_win\_count** | **Toss\_win\_match\_win\_percentage** |
| Gujarat Lions | 6 | 8 | 75.0 |
| Chennai Super Kings | 42 | 66 | 63.6 |
| Mumbai Indians | 41 | 74 | 55.4 |
| Kolkata Knight Riders | 38 | 69 | 55.1 |
| Royal Challengers Bangalore | 33 | 61 | 54.1 |
| Rajasthan Royals | 34 | 63 | 54.0 |
| Kochi Tuskers Kerala | 4 | 8 | 50.0 |
| Sunrisers Hyderabad | 14 | 30 | 46.7 |
| Deccan Chargers | 19 | 43 | 44.2 |
| Delhi Daredevils | 28 | 64 | 43.8 |
| Rising Pune Supergiants | 3 | 7 | 42.9 |
| Kings XI Punjab | 26 | 64 | 40.6 |
| Pune Warriors | 3 | 20 | 15.0 |

This query helps to identify how effectively teams capitalize on winning the toss, giving insights into whether winning the toss significantly influences their chances of winning matches.

**Query: match win percentage after won toss in each venue**

|  |
| --- |
| WITH Toss\_Win\_Stats AS (  SELECT  v.Venue\_Name,  td.Toss\_Name AS Toss\_Decision,  COUNT(\*) AS Total\_Matches,  SUM(CASE WHEN m.Match\_Winner = m.Toss\_Winner THEN 1 ELSE 0 END) AS Matches\_Won\_After\_Toss,  (SUM(CASE WHEN m.Match\_Winner = m.Toss\_Winner THEN 1 ELSE 0 END) / COUNT(\*)) \* 100 AS Win\_Percentage  FROM  matches m  INNER JOIN  toss\_decision td  ON m.Toss\_Decide = td.Toss\_Id  INNER JOIN  venue v  ON m.Venue\_Id = v.Venue\_Id  GROUP BY  v.Venue\_Name, td.Toss\_Name  )  SELECT  Venue\_Name,  Toss\_Decision,  Total\_Matches,  Matches\_Won\_After\_Toss,  Win\_Percentage  FROM  Toss\_Win\_Stats  WHERE  Total\_Matches >= 10; |

**Explanation**

* **CTE: Toss\_Win\_Stats:**
  + Calculates toss-related statistics for each venue and toss decision (batting or fielding first).
  + Joins the matches, toss\_decision, and venue tables to gather relevant data.
  + Columns Calculated:
    1. Venue\_Name: Name of the venue from the venue table.
    2. Toss\_Decision: Toss decision (e.g., bat or field) from the toss\_decision table.
    3. Total\_Matches: Total matches played at the venue with a specific toss decision, counted using COUNT(\*).
    4. Matches\_Won\_After\_Toss: Matches where the toss winner also won the match.
    5. Adds 1 for every match where Match\_Winner = Toss\_Winner, else adds 0.
    6. Win\_Percentage: The percentage of matches won after winning the toss, calculated as: Win\_Percentage=(Matches\_Won\_After\_TossTotal\_Matches)×100\text{Win\\_Percentage} = \left(\frac{\text{Matches\\_Won\\_After\\_Toss}}{\text{Total\\_Matches}}\right) \times 100Win\_Percentage=(Total\_MatchesMatches\_Won\_After\_Toss​)×100
    7. Data is grouped by Venue\_Name and Toss\_Name using GROUP BY.
* **Main Query:**
  + 1. Selects the columns from the Toss\_Win\_Stats CTE.
    2. Filters the results to include only venues with at least 10 matches (WHERE Total\_Matches >= 10), ensuring sufficient data for meaningful analysis.
* **Final Output:**
  + 1. Venue\_Name: Venue name.
    2. Toss\_Decision: Toss decision (e.g., bat or field first).
    3. Total\_Matches: Total matches played with the toss decision at the venue.
    4. Matches\_Won\_After\_Toss: Matches won by the toss-winning team.
    5. Win\_Percentage: Percentage of matches won after winning the toss, rounded to the nearest decimal place**.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Venue\_name** | **toss\_dicission** | | **Total\_matches** | **Match\_won\_After\_loss** | **win\_percentage** |
| M Chinnaswamy Stadium | field | 50 | | 27 | 54 |
| Punjab Cricket Association Stadium, Mohali | bat | 14 | | 5 | 35.7143 |
| Feroz Shah Kotla | bat | 22 | | 9 | 40.9091 |
| Wankhede Stadium | bat | 20 | | 10 | 50 |
| Eden Gardens | bat | 28 | | 12 | 42.8571 |
| Sawai Mansingh Stadium | bat | 14 | | 3 | 21.4286 |
| Rajiv Gandhi International Stadium, Uppal | bat | 20 | | 3 | 15 |
| MA Chidambaram Stadium, Chepauk | field | 14 | | 4 | 28.5714 |
| Rajiv Gandhi International Stadium, Uppal | field | 21 | | 11 | 52.381 |
| Punjab Cricket Association Stadium, Mohali | field | 21 | | 11 | 52.381 |
| MA Chidambaram Stadium, Chepauk | bat | 34 | | 21 | 61.7647 |
| Dr DY Patil Sports Academy | field | 10 | | 6 | 60 |
| Feroz Shah Kotla | field | 31 | | 17 | 54.8387 |
| Sawai Mansingh Stadium | field | 19 | | 12 | 63.1579 |
| Wankhede Stadium | field | 29 | | 14 | 48.2759 |
| Eden Gardens | field | 26 | | 18 | 69.2308 |
| Kingsmead | bat | 10 | | 6 | 60 |
| Subrata Roy Sahara Stadium | bat | 15 | | 9 | 60 |

2. Suggest some of the players who would be best fit for the team.

**ANS-**

* Considering the below points suggest the best players for the team

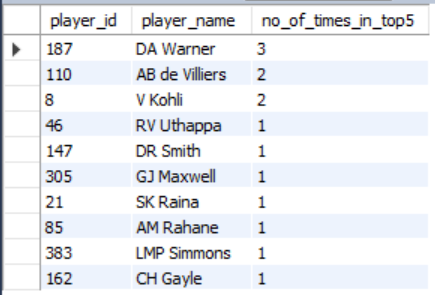
1. Top 5 score rank in each season--players and number of times they are in the top 5 in the last 3 seasons.
2. Top 10 players with the highest strike rate and minimum 1500 runs scored.
3. Top 10 bowlers taken highest wicket.
4. Top 10 bowlers with best economy, minimum 100 overs bowled.

* **Query: Top 5 score rank in each season--players and number of times they are in the top 5 in the last 3 seasons.**

|  |
| --- |
| WITH player\_match\_runs AS (  SELECT  bb.match\_id,  bb. striker,  SUM(runs\_scored) AS total\_runs  FROM  ball\_by\_ball bb  JOIN  batsman\_scored bs  ON bb.Match\_Id = bs.Match\_Id  AND bb.Innings\_No = bs.Innings\_No  AND bb.Over\_Id = bs.Over\_Id  AND bb.Ball\_Id = bs.Ball\_Id  GROUP BY  bb.match\_id, bb.striker  ),  result AS (  SELECT  pmr. striker AS player\_id,  s.season\_year,  SUM(pmr.total\_runs) AS runs\_scored,  DENSE\_RANK() OVER (PARTITION BY season\_year ORDER BY SUM(pmr.total\_runs) DESC) AS season\_runs\_rank  FROM  player\_match\_runs pmr  JOIN  matches m  ON pmr.match\_id = m.match\_id  JOIN  season s  ON m.season\_id = s.season\_id  GROUP BY  pmr. striker, s.season\_year  ),  top\_run\_rank\_player AS (  SELECT  \*  FROM  result  WHERE  season\_runs\_rank BETWEEN 1 AND 5  AND season\_year BETWEEN 2014 AND 2016  ORDER BY  season\_year, season\_runs\_rank  )  SELECT  trp.player\_id,  p.player\_name,  COUNT(trp.player\_id) AS no\_of\_times\_in\_top5  FROM  top\_run\_rank\_player trp  JOIN  player p  ON trp.player\_id = p.player\_id  GROUP BY  trp.player\_id, p.player\_name  ORDER BY  COUNT(trp.player\_id) DESC  LIMIT 10; |

**Explanation**

1. **player\_match\_runs CTE:**
   * Calculates the total runs scored by each batsman in each match.
   * Joins ball\_by\_ball and batsman\_scored tables on match, innings, over, and ball IDs to get runs\_scored data.
   * Groups by match\_id and striker (batsman) to calculate total runs per player per match.
2. **result CTE:**
   * Calculates total runs scored by each player in each IPL season.
   * Joins player\_match\_runs with the matches table to get season details.
   * Uses DENSE\_RANK to rank players in each season based on their total runs (season\_runs\_rank).
   * Groups by player\_id and season\_year for aggregation.
3. **top\_run\_rank\_player CTE:**
   * Filters players ranked in the top 5 run-scorers (season\_runs\_rank BETWEEN 1 AND 5) for seasons between 2014 and 2016.
   * Orders the result by season\_year and season\_runs\_rank.
4. **Main Query:**
   * Joins top\_run\_rank\_player with the player table to get player names.
   * Counts how many times each player appeared in the top 5 run-scorers during the specified seasons (no\_of\_times\_in\_top5).
   * Groups by player\_id and player\_name.
   * Orders the players by their appearances in descending order, returning the top 10 players (LIMIT 10).
5. **Output:**



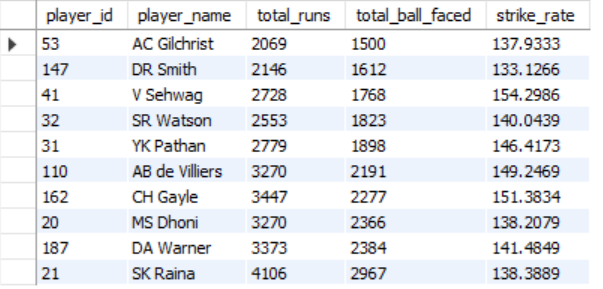
* **Query: Top 10 players with the highest strike rate and minimum 1500 runs scored.**

|  |
| --- |
| with runs\_scored as (select bb.striker,p.player\_name,sum(bs.runs\_scored) as total\_runs  from player p join player\_match pm on p.player\_id=pm.player\_id  join ball\_by\_ball bb on pm.player\_id=bb.striker and pm.match\_id=bb.match\_id  join batsman\_scored bs on bb.match\_id=bs.match\_id and bb.over\_id=bs.over\_id  and bb.ball\_id=bs.ball\_id and bb.innings\_no=bs.innings\_no  group by bb. striker,p.player\_name having(sum(bs.runs\_scored))>2000  order by total\_runs desc),  balls\_faced as (select bb.striker,p.player\_name,count(bs.runs\_scored)total\_ball\_faced  from player p join player\_match pm on p.player\_id=pm.player\_id  join ball\_by\_ball bb on pm.player\_id=bb.striker and pm.match\_id=bb.match\_id  join batsman\_scored bs on bb.match\_id=bs.match\_id and bb.over\_id=bs.over\_id  and bb.ball\_id=bs.ball\_id and bb.innings\_no=bs.innings\_no  group by bb. striker,p.player\_name order by total\_ball\_faced desc)  select rs.striker as player\_id,rs.player\_name,rs.total\_runs,  bf.total\_ball\_faced,(rs.total\_runs/bf.total\_ball\_faced)\*100 as strike\_rate  from runs\_scored rs join balls\_faced bf on rs.striker=bf.striker  order by strike\_rate desc  limit 10; |

**Explanation:**

* This query identifies the top 10 players with the highest strike rate who have scored at least 2000 runs. Here’s a brief explanation of each step:
* **runs\_scored CTE:**
  + Calculates the total runs scored by each player.
  + Only includes players with more than 2000 total runs (HAVING (SUM(bs.runs\_scored)) > 2000).
  + Orders players by total\_runs in descending order.
* **balls\_faced CTE:**
  + Counts the total balls faced by each player (each runs\_scored entry).
  + Groups by player and orders by total\_ball\_faced in descending order.
* **Final Query:**
  + Joins runs\_scored and balls\_faced on player ID to obtain both total runs and balls faced.
  + Calculates each player’s strike rate as (total\_runs / total\_ball\_faced) \* 100.
  + Orders players by strike rate in descending order and limits the result to the top 10.

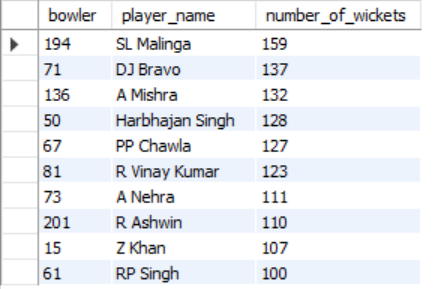
This final list shows the top 10 players with the highest strike rates among those who have scored over 2000 runs.



* **Query: Top 10 bowlers taken highest wicket.**

|  |
| --- |
| select bb.bowler,p.player\_name,count(wt.player\_out) as number\_of\_wickets  from player p join player\_match pm on p.player\_id=pm.player\_id  join ball\_by\_ball bb on pm.player\_id=bb.bowler and pm.match\_id=bb.match\_id  join wicket\_taken wt on bb.match\_id=wt.match\_id and bb.over\_id=wt.over\_id  and bb.ball\_id=wt.ball\_id and bb.innings\_no=wt.innings\_no  group by bb. bowler,p.player\_name order by number\_of\_wickets desc  limit 10; |

* **Explanation:**
* This query retrieves the top 10 bowlers who have taken the highest number of wickets. Here's a brief breakdown:
* **Join Tables**:
  + Joins the player, player\_match, ball\_by\_ball, and wicket\_taken tables to gather data on bowlers and their wickets in each match.
* **Calculate Wickets**:
  + Counts the occurrences of player\_out in the wicket\_taken table for each bowler, representing the number of wickets taken by that bowler.
* **Group and Order**:
  + Group results by bowler and player\_name to get the total wickets for each bowler.
  + Orders by number\_of\_wickets in descending order and limits the result to the top 10 bowlers.
* This final output shows the 10 bowlers with the most wickets, ranked by the highest wicket count
* **Output:**



* **Query: Top 10 bowlers with best economy, minimum 100 overs bowled.**

|  |
| --- |
| with total\_runs\_conceeded as (select bb.bowler as player\_id,p.player\_name,sum(bs.runs\_scored) as runs\_conceeded  from player p join player\_match pm on p.player\_id=pm.player\_id  join ball\_by\_ball bb on pm.player\_id=bb.bowler and pm.match\_id=bb.match\_id  join batsman\_scored bs on bb.match\_id=bs.match\_id and bb.over\_id=bs.over\_id  and bb.ball\_id=bs.ball\_id and bb.innings\_no=bs.innings\_no  group by bb.bowler,p.player\_name order by runs\_conceeded desc),  total\_overs\_bowled as (select bb.bowler as player\_id,count(distinct bs.match\_id,bs.innings\_no,bs.over\_id) as total\_overs  from player p join player\_match pm on p.player\_id=pm.player\_id  join ball\_by\_ball bb on pm.player\_id=bb.bowler and pm.match\_id=bb.match\_id  join batsman\_scored bs on bb.match\_id=bs.match\_id and bb.over\_id=bs.over\_id  and bb.ball\_id=bs.ball\_id and bb.innings\_no=bs.innings\_no  group by bb. bowler order by total\_overs desc)  select trc.\*,tob.total\_overs,(trc.runs\_conceeded/tob.total\_overs) as economy  from total\_runs\_conceeded trc join total\_overs\_bowled tob on trc.player\_id=tob.player\_id  where total\_overs>100  order by economy limit 10; |

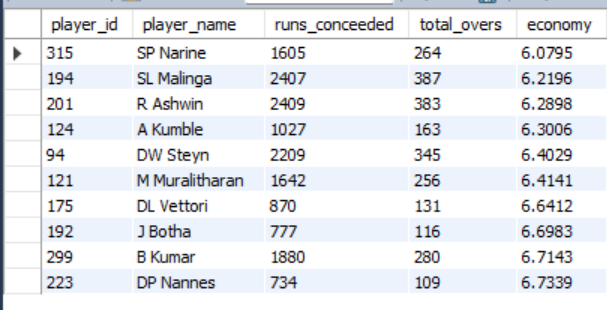
**Explanation:**

This query identifies the top 10 bowlers with the best economy rates (lowest runs conceded per over) who have bowled at least 100 overs.

**Steps:**

* Calculate Runs Conceded: Sum up total runs conceded by each bowler.
* Calculate Overs Bowled: Count unique overs bowled by each bowler.
* Compute Economy Rate: Divide runs conceded by overs bowled for each bowler with more than 100 overs.
* Filter and Sort: Select the top 10 bowlers with the lowest economy rates.

This provides the 10 most economical bowlers with a minimum threshold of overs bowled.



* + 1. What are some of the parameters that should be focused on while selecting the players?

ANS-

When selecting players for an IPL team, focusing on the right parameters ensures that the team is well-balanced and competitive. Here are key parameters to consider:

**1. Player Performance Metrics:**

* ***Batting Average****:* Indicates a batsman's consistency and reliability. A higher average suggests that a batsman score runs consistently.
* ***Strike Rate****:* Shows how quickly a batsman scores runs, which is essential for the T20 format where quick scoring is crucial.
* ***Total Runs/Wickets****:* Helps identify top performers with a proven track record in scoring runs or taking wickets.
* ***Bowling Economy Rate****:* Measures a bowler’s effectiveness in restricting runs, particularly important in T20 matches.
* ***Wicket-taking Ability****:* Indicates a bowler’s impact in breaking partnerships and taking key wickets.

**2. Consistency:**

* ***Recent Form****:* Current season or recent performances should be evaluated to understand the player's recent form.
* ***Consistency Across Matches****:* Use averages and median scores/wickets to identify players who perform consistently over time.

**3. Player Role and Team Balance:**

* ***Role Classification****:* Ensure a balanced mix of batsmen, bowlers, all-rounders, and a wicketkeeper.
* ***Specialists vs. All-rounders****:* Decide the proportion of specialist batsmen and bowlers versus all-rounders based on the team’s strategy.
* ***Finisher Capability****:* Identify batsmen who can score quickly and effectively in the final overs.

**4. Adaptability to Conditions:**

* ***Performance at Specific Venues****:* Review a player's record at various venues to see if they adapt well to different pitches (e.g., spin-friendly or pace-friendly).
* ***Performance Against Top Teams****:* Analyze how players perform under pressure against strong opposition.

**5. Skill Specialization:**

* ***Batting Style****:* Choose players with varying styles (e.g., left-handers, aggressive hitters, technical players) to handle different bowling attacks.
* ***Bowling Type****:* Have a mix of seamers and spinners, and consider variations like wrist spin or left-arm pace for diversity in attack.
* ***Powerplay and Death Over Specialists****:* Pick bowlers who excel at specific phases of the game.

**6. Fielding Ability:**

* ***Fielding Skills****:* Good fielders can save runs and create match-winning moments with catches and run-outs.
* ***Wicketkeeping Performance****:* If selecting a wicketkeeper, evaluate their dismissals, stumping speed, and catching reliability.

**7. Injury and Fitness Levels:**

* ***Recent Injuries****:* Review any recent injuries that could affect the player's availability and performance.
* ***General Fitness****:* Ensure that players can endure the physical demands of the tournament.

**8. Experience and Leadership:**

* ***Seasoned Players****:* Include experienced players who bring calm and strategic thinking during crucial moments.
* ***Young Talent****:* Balance the squad with emerging players who bring energy and are eager to prove themselves.
* ***Captaincy Potential****:* Consider players with leadership skills to assist or take over as captain if needed.

**9. Player Adaptability:**

* ***Versatility****:* Players who can adapt to different batting positions or bowling situations provide more strategic options.
* ***All-Weather Performance***: Players who perform well in both day and night matches or in different climate conditions.

**10. Team Chemistry and Work Ethic:**

* ***Compatibility****:* Assess how well a player integrates with the team’s culture and dynamics.
* ***Work Ethic and Attitude****:* Choose players known for their strong work ethic and positive impact on team morale.

Focusing on these parameters ensures that the team is skilled, adaptable, cohesive, and strategically equipped for the challenges of the tournament.

* + 1. Which players offer versatility in their skills and can contribute effectively with both bat and ball? (can you visualise the data for the same).

ANS-

|  |
| --- |
| -- Create a Common Table Expression (CTE) for top bowlers  WITH top\_bowler AS (  SELECT  bb. bowler,  p.player\_name,  COUNT(wt.player\_out) AS number\_of\_wickets  FROM  player p  JOIN  player\_match pm  ON p.player\_id = pm.player\_id  JOIN  ball\_by\_ball bb  ON pm.player\_id = bb. bowler  AND pm.match\_id = bb.match\_id  JOIN  wicket\_taken wt  ON bb.match\_id = wt.match\_id  AND bb.over\_id = wt.over\_id  AND bb.ball\_id = wt.ball\_id  AND bb.innings\_no = wt.innings\_no  GROUP BY  bb. bowler, p.player\_name  ORDER BY  number\_of\_wickets DESC  ),  -- Create another CTE for top batsmen  top\_batsman AS (  SELECT  bb.striker,  p.player\_name,  SUM(bs.runs\_scored) AS total\_runs  FROM  player p  JOIN  player\_match pm  ON p.player\_id = pm.player\_id  JOIN  ball\_by\_ball bb  ON pm.player\_id = bb. striker  AND pm.match\_id = bb.match\_id  JOIN  batsman\_scored bs  ON bb.match\_id = bs.match\_id  AND bb.over\_id = bs.over\_id  AND bb.ball\_id = bs.ball\_id  AND bb.innings\_no = bs.innings\_no  GROUP BY  bb.striker, p.player\_name  ORDER BY  total\_runs DESC  )  -- Combine data from top\_bowler and top\_batsman  SELECT  tb. bowler AS player\_id,  tb.player\_name,  tb.number\_of\_wickets,  tbs.total\_runs  FROM  top\_bowler tb  JOIN  top\_batsman tbs  ON tb.bowler = tbs.striker  WHERE  tb.number\_of\_wickets > 50  AND tbs.total\_runs > 1000; |

**Explanation:**

**Step 1: Identify Top Bowlers (CTE top\_bowler)**

* **Goal:** Find players who have taken the most wickets.
* **Key Steps:**
  + Join the player, player\_match, ball\_by\_ball, and wicket\_taken tables to track bowlers and their associated wickets.
  + Use COUNT(wt.player\_out) to count the number of wickets taken by each bowler.
  + Group by bb. bowler and p.player\_name to aggregate data per player.
  + Sort the results by number\_of\_wickets in descending order.

**Step 2: Identify Top Batsmen (CTE top\_batsman)**

* **Goal:** Find players who have scored the most runs.
* **Key Steps:**
  + Join the player, player\_match, ball\_by\_ball, and batsman\_scored tables to track batsmen and their associated runs.
  + Use SUM(bs.runs\_scored) to sum the runs scored by each batsman.
  + Group by bb. striker and p.player\_name to aggregate data per player.
  + Sort the results by total\_runs in descending order.

**Step 3: Combine Bowlers and Batsmen (Main Query)**

* **Goal:** Find players who excel as both bowlers and batsmen.
* **Key Steps:**
  + Join top\_bowler and top\_batsman on the player's ID (tb.bowler = tbs.striker).
  + Filter the results to include only players with more than 50 wickets and more than 1000 runs (WHERE tb.number\_of\_wickets > 50 AND tbs.total\_runs > 1000).
  + Select relevant fields like player\_id, player\_name, number\_of\_wickets, and total\_runs.

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **player\_id** | **player\_name** | **number\_of\_wickets** | **total\_runs** |
| 71 | DJ Bravo | 137 | 1262 |
| 29 | IK Pathan | 97 | 1148 |
| 32 | SR Watson | 95 | 2553 |
| 35 | RA Jadeja | 84 | 1574 |
| 9 | JH Kallis | 74 | 2427 |
| 221 | KA Pollard | 67 | 1959 |

**Visualization**

**Insights:**

Players with batting performance and bowling including are well-required in critical situations. They can adapt to both situations, and most of the time match winning situations these players contribute a lot to the team.

* + 1. Are there players whose presence positively influences the morale and performance of the team? (justify your answer using visualisation).

ANS-

* To determine if there are players whose presence positively influences the morale and performance of the team, we can analyse various metrics related to the players, their roles, and their contributions to the team's success. Here are some approaches.

**Win Rates with Player Participation**:

* Analyse the win rates of teams when certain players are part of the lineup compared to when they are not. Higher win rates with their participation might indicate a positive influence.

**Man of the Match Awards**:

* Evaluate how often players receive "Man of the Match" awards, as these reflect significant individual contributions to match outcomes.

|  |
| --- |
| -- win rate with player presence  SELECT pm.Player\_Id, p.player\_name,COUNT(DISTINCT m.Match\_Id) AS Matches\_Played,  SUM(CASE WHEN m.Match\_Winner = pm.Team\_Id THEN 1 ELSE 0 END) AS Wins,  (SUM(CASE WHEN m.Match\_Winner = pm.Team\_Id THEN 1 ELSE 0 END) / COUNT(DISTINCT m.Match\_Id)) \* 100 AS Win\_Rate  FROM player\_match pm  JOIN matches m ON pm.Match\_Id = m.Match\_Id  join player p on p.player\_id=pm.player\_id  GROUP BY pm.Player\_Id,p.player\_name  HAVING Matches\_Played > 80  order by win\_rate desc; |

* **Explanation:**

This SQL query calculates the win rate of players based on their presence in matches. Here’s a short breakdown of its components and functionality:

Explanation of the Query

FROM and JOIN Clauses:

* + The query starts from the player\_match table (pm), which connects players to specific matches.
  + It joins the matches table (m) to access match-related information, such as the match winner.
  + It also joins the player table (p) to get the player's name.

Selecting Fields:

* + pm.Player\_Id and p.player\_name: These columns identify each player.
  + COUNT(DISTINCT m.Match\_Id) AS Matches\_Played: Counts the total number of unique matches each player has participated in.
  + SUM(CASE WHEN m.Match\_Winner = pm.Team\_Id THEN 1 ELSE 0 END) AS Wins: Sums up the number of matches won by the player's team. If the player's team won, it counts as a win; otherwise, it counts as zero.

Calculating Win Rate:

* + The win rate is calculated as the total wins divided by the total matches played, multiplied by 100 to convert it to a percentage.

GROUP BY Clause:

* + Groups the results by player ID and name, allowing aggregation functions (like COUNT and SUM) to operate within each group.

HAVING Clause:

* + Filters the results to only include players who have participated in more than 80 matches.

ORDER BY Clause:

* + Orders the final results by win rate in descending order, so the players with the highest win rates appear first.
* **Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **player\_id** | **player\_name** | **Matches\_played** | **Wins** | **Win\_rate** |
| 208 | AT Rayudu | 109 | 66 | 60.6 |
| 21 | SK Raina | 146 | 88 | 60.3 |
| 71 | DJ Bravo | 105 | 63 | 60.0 |
| 23 | S Badrinath | 94 | 56 | 59.6 |
| 221 | KA Pollard | 106 | 63 | 59.4 |
| 35 | RA Jadeja | 125 | 74 | 59.2 |
| 201 | R Ashwin | 110 | 65 | 59.1 |
| 20 | MS Dhoni | 142 | 83 | 58.5 |
| 194 | SL Malinga | 98 | 57 | 58.2 |
| 31 | YK Pathan | 134 | 77 | 57.5 |
| 40 | G Gambhir | 132 | 75 | 56.8 |
| 50 | Harbhajan Singh | 125 | 71 | 56.8 |
| 57 | RG Sharma | 142 | 76 | 53.5 |
| 89 | R Bhatia | 92 | 49 | 53.3 |
| 110 | AB de Villiers | 119 | 63 | 52.9 |
| 88 | KD Karthik | 138 | 73 | 52.9 |
| 109 | JA Morkel | 90 | 47 | 52.2 |
| 32 | SR Watson | 94 | 49 | 52.1 |
| 67 | PP Chawla | 123 | 64 | 52.0 |

* **Visualization:**

**Query:**

|  |
| --- |
| **-**- most man of the match award  SELECT pm.Player\_Id,p.player\_name,COUNT(m.Man\_of\_the\_Match) AS Man\_of\_the\_Match\_Awards  FROM player\_match pm  JOIN matches m ON pm.Match\_Id = m.Match\_Id  join player p on p.player\_id=pm.player\_id  WHERE pm.Player\_Id = m.Man\_of\_the\_Match  GROUP BY pm.Player\_Id,p.player\_name order by Man\_of\_the\_Match\_Awards desc; |

* **Explanation**

This query identifies players who have won the most "Man of the Match" awards in IPL. It works as follows:

1. **Joins Tables:** Combines player\_match, matches, and player tables to link matches, players, and their names.
2. **Filter:** Includes only the players marked as Man\_of\_the\_Match in each game.
3. **Count Awards:** Uses COUNT() to tally the number of awards for each player.
4. **Group & Sort:** Group results by player and sort them in descending order by the number of awards.

* **Output:**

|  |  |  |
| --- | --- | --- |
| **Player\_id** | **Player\_name** | **Man\_of\_the\_match\_awards** |
| 162 | CH Gayle | 17 |
| 31 | YK Pathan | 16 |
| 110 | AB de Villiers | 15 |
| 187 | DA Warner | 14 |
| 21 | SK Raina | 13 |
| 57 | RG Sharma | 13 |
| 19 | MEK Hussey | 12 |
| 20 | MS Dhoni | 12 |
| 40 | G Gambhir | 12 |
| 85 | AM Rahane | 12 |

* + 1. What would you suggest to RCB before going to a mega auction?

Ans-

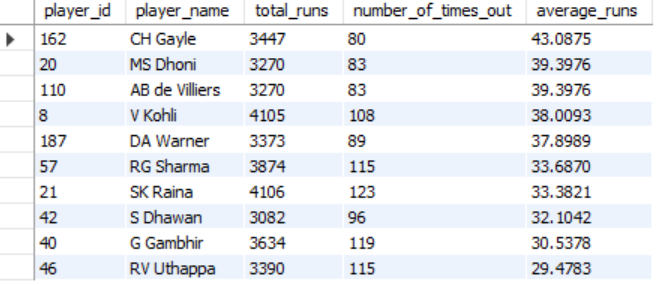
* Before the mega auction, Ill suggest Royal Challengers Bangalore (RCB) to consider the following strategies and recommendations.
* Players with the highest runs.
* Players with the best average.
* Players with the highest strike rate.
* Players with the most wickets taken.
* Players with the best economy in bowling.
* Players with the most success rate in the team winning including the highest man of the match award.etc

By considering the above factors the below players will suggest to RCB before going to the mega auction.

* **Query- Players with the best average.**

|  |
| --- |
| WITH player\_runs AS (  SELECT  p.player\_id,  p.player\_name,  SUM(bs.runs\_scored) AS total\_runs  FROM  player p  JOIN player\_match pm  ON p.player\_id = pm.player\_id  JOIN ball\_by\_ball bb  ON pm.match\_id = bb.match\_id  AND pm.player\_id = bb.striker  JOIN batsman\_scored bs  ON bb.match\_id = bs.match\_id  AND bb.over\_id = bs.over\_id  AND bb.ball\_id = bs.ball\_id  AND bb.innings\_no = bs.innings\_no  GROUP BY  p.player\_id, p.player\_name  ORDER BY  total\_runs DESC  ),  player\_out AS (  SELECT  p.player\_id,  p.player\_name,  COUNT(wt.player\_out) AS number\_of\_times\_out  FROM  player p  JOIN wicket\_taken wt  ON p.player\_id = wt.player\_out  GROUP BY  p.player\_id, p.player\_name  )  SELECT  pr.\*,  po.number\_of\_times\_out,  ROUND(pr.total\_runs / COALESCE(po.number\_of\_times\_out, 1), 1) AS average\_runs  FROM  player\_runs pr  LEFT JOIN player\_out po  ON pr.player\_id = po.player\_id  WHERE  pr.total\_runs > 3000  ORDER BY  average\_runs DESC  LIMIT 10; |

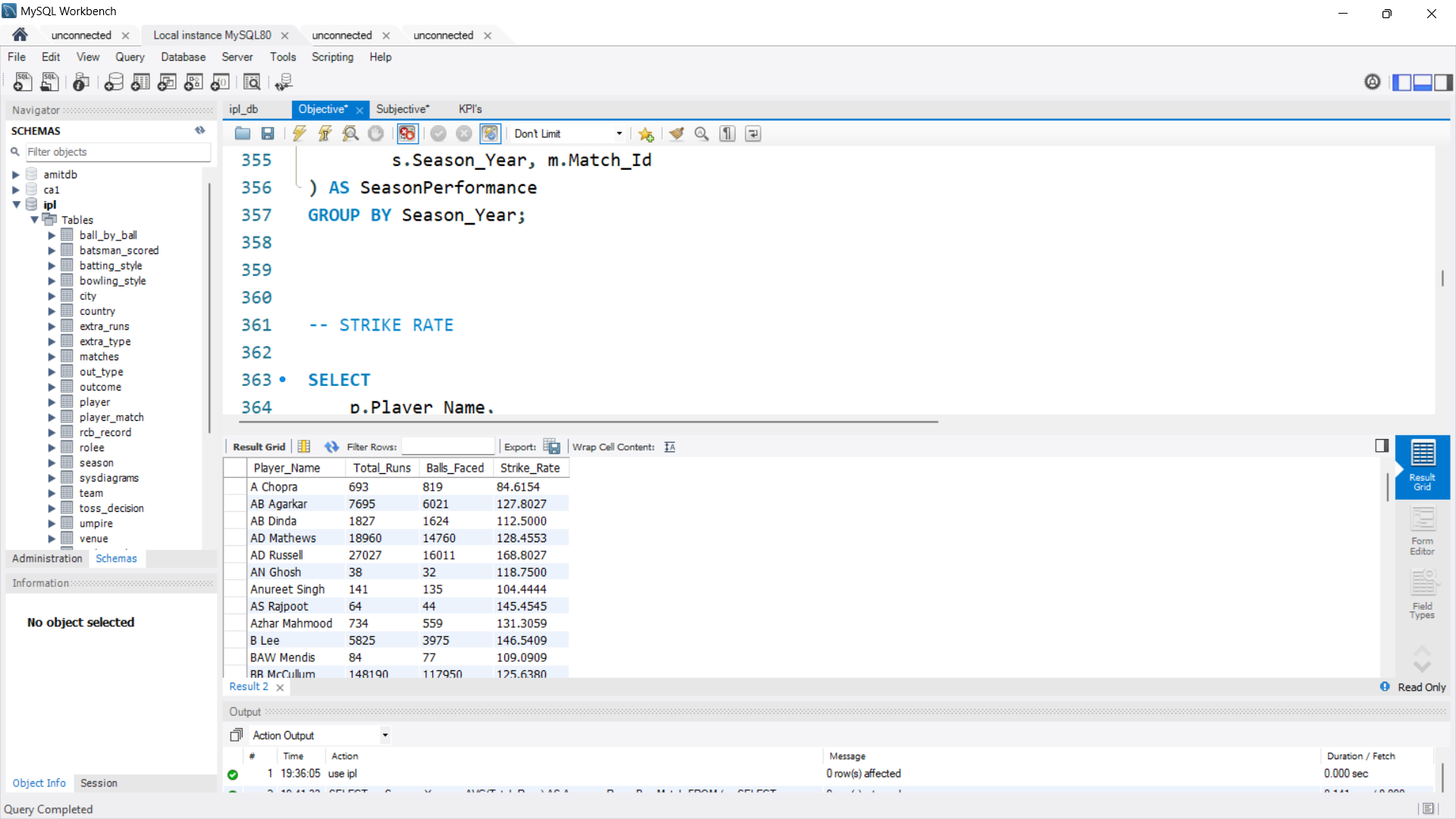
**Result:**



* **Players with the highest strike rate.**

|  |
| --- |
| SELECT  p.Player\_Name,  SUM(bs.Runs\_Scored) AS Total\_Runs,  COUNT(bb.Ball\_Id) AS Balls\_Faced,  (SUM(bs.Runs\_Scored) / COUNT(bb.Ball\_Id)) \* 100 AS Strike\_Rate  FROM  player p  JOIN  ball\_by\_ball bb ON p.Player\_Id = bb.Striker -- Assuming Striker column links to player  LEFT JOIN  batsman\_scored bs ON bb.Match\_Id = bs.Match\_Id AND bb.Over\_Id = bs.Over\_Id AND bb.Ball\_Id = bs.Ball\_Id  LEFT JOIN  extra\_runs er ON bb.Match\_Id = er.Match\_Id AND bb.Over\_Id = er.Over\_Id AND bb.Ball\_Id = er.Ball\_Id  JOIN  player\_match pm ON p.Player\_Id = pm.Player\_Id  WHERE  pm.Team\_Id = 1 -- Replace with actual Team\_Id  AND er.Extra\_Type\_Id IS NULL -- Exclude extra deliveries like wides and no-balls  GROUP BY  p.Player\_Name; |

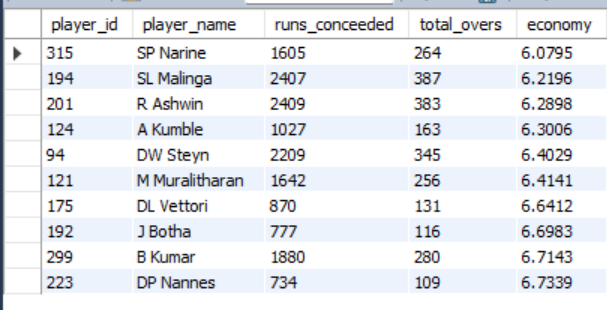
**Result:**



* **Query: Players with the best economy in bowling.**

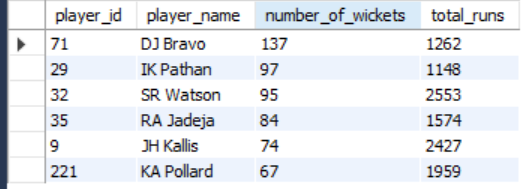
|  |
| --- |
| WITH rs AS (  SELECT  bb.bowler,  p.player\_name,  SUM(bs.runs\_scored) AS total\_runs\_conceded,  COUNT(DISTINCT bb.match\_id, bb.over\_id) AS total\_overs\_bowled  FROM  ball\_by\_ball bb  JOIN batsman\_scored bs  ON bb.match\_id = bs.match\_id  AND bb.over\_id = bs.over\_id  AND bb.ball\_id = bs.ball\_id  AND bb.innings\_no = bs.innings\_no  JOIN matches m  ON bs.match\_id = m.match\_id  JOIN player p  ON bb.bowler = p.player\_id  GROUP BY  bb.bowler, p.player\_name  )  SELECT DISTINCT  rs.\*,  total\_runs\_conceded / total\_overs\_bowled AS economy  FROM  rs  JOIN player\_match pm  ON rs.bowler = pm.player\_id  WHERE  total\_overs\_bowled > 50  ORDER BY  economy DESC; |

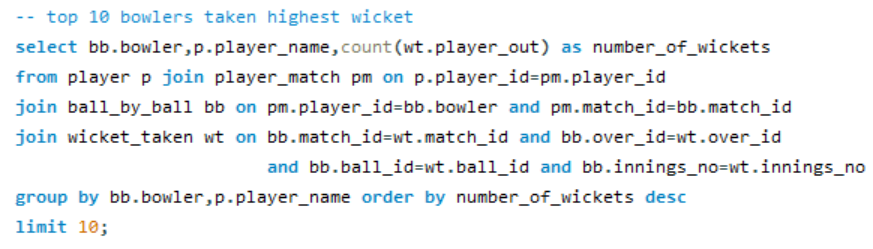
Result:

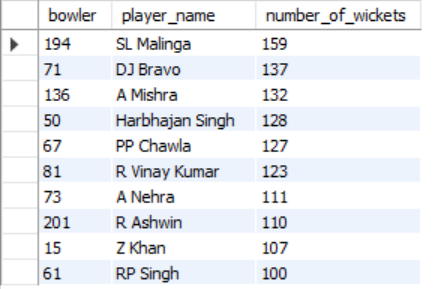


* **All-rounders:**

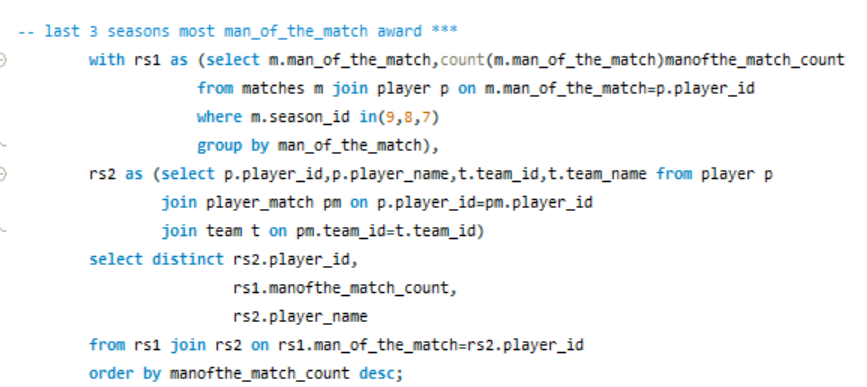
Refer to the query for question number 4

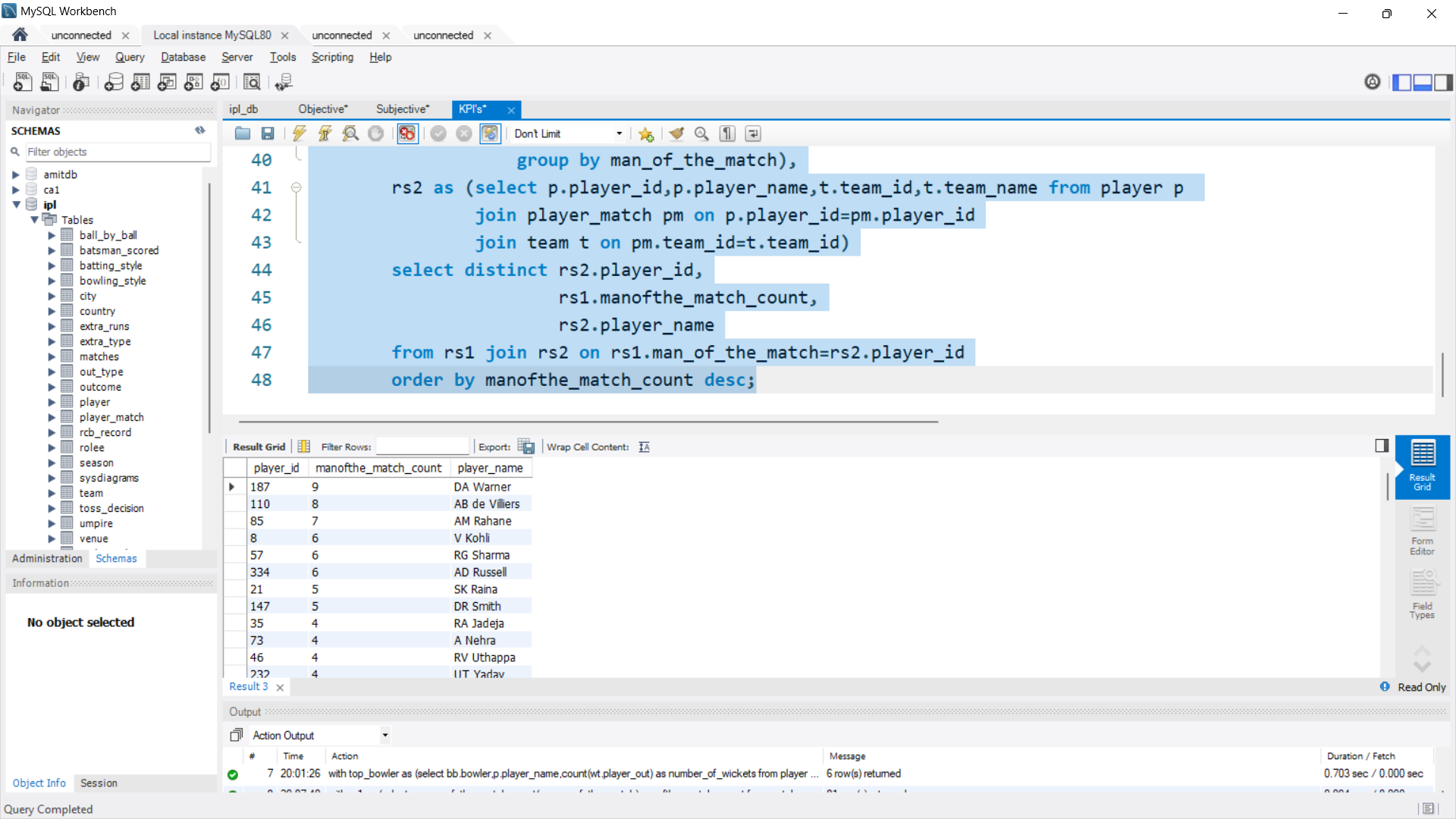
* 
* **Players with the most wickets taken.**





* Players with the most success rate in the team winning including the highest man of the match award.etc





* + 1. What do you think could be the factors contributing to the high-scoring matches and the impact on viewership and team strategies?

Ans-

**Factors Contributing to High-Scoring Matches:**

1. **Pitch Conditions:**
   * Flat pitches with minimal assistance for bowlers encourage high-scoring games.
   * Venues like Mumbai's Wankhede Stadium are known for being batting-friendly.
2. **Ground Dimensions:**
   * Smaller boundaries make it easier for batsmen to score runs quickly.
   * Stadiums with smaller dimensions lead to higher six and boundary counts.
3. **Advancements in Batting Techniques:**
   * Players are increasingly adept at playing innovative shots (e.g., reverse sweeps, scoops).
   * The T20 format encourages aggressive batting right from the start.
4. **Powerplay Rules:**
   * Field restrictions in the first six overs allow batsmen to score freely.
5. **Quality of Batsmen:**
   * Presence of world-class players and power hitters like AB de Villiers, Chris Gayle, and Virat Kohli.
6. **Bowling Challenges:**
   * Pressure on bowlers to contain runs often leads to mistakes, such as poor line and length.
   * Bowling under the dew factor makes gripping the ball difficult.
7. **Weather Conditions:**
   * Humid conditions, especially under lights, can affect bowlers' performance.
8. **Equipment Evolution:**
   * High-quality bats provide more power to players, making it easier to clear boundaries.

**Impact on Viewership:**

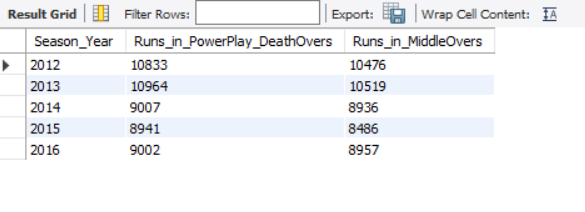
1. **Increased Engagement:**
   * High-scoring matches are thrilling, keeping viewers glued till the last ball.
   * Big scores attract casual fans, expanding the audience base.
2. **Brand Sponsorships:**
   * More viewership translates to better sponsorship deals and higher advertisement revenue.
3. **Fan Expectations:**
   * Fans prefer T20 matches to be action-packed with sixes and boundaries.

**Impact on Team Strategies:**

1. **Batting First Approach:**
   * Teams batting first often aim for 200+ scores to pressure the chasing side.
2. **Flexible Batting Orders:**
   * Teams send power hitters earlier when a big score is needed or to capitalize on field restrictions.
3. **Specialist Bowlers:**
   * Teams rely on death-over specialists and wrist spinners to contain runs.
4. **Fitness and Fielding Standards:**
   * Improved fielding efforts to save crucial runs in high-scoring games.
5. **Impact Players:**
   * Utilizing all-rounders who can contribute both with the bat and ball.
6. **Bowling Plans:**
   * Execution of yorkers and variations (e.g., slower balls) is emphasized to counter aggressive batting.
7. **Comparison between Runs Scored during Power Play (1 to 6 overs) & Death Overs (17 to 20 overs) and Runs Scored during Middle Overs (7 to 16 Overs):**

* Query

|  |
| --- |
| SELECT  t4.Season\_Year,  SUM(  CASE  WHEN t1.Over\_Id BETWEEN 1 AND 6 OR t1.Over\_Id BETWEEN 17 AND 20  THEN t2.Runs\_Scored  ELSE 0  END  ) AS Runs\_in\_PowerPlay\_DeathOvers,  SUM(  CASE  WHEN t1.Over\_Id BETWEEN 7 AND 16  THEN t2.Runs\_Scored  ELSE 0  END  ) AS Runs\_in\_MiddleOvers  FROM  ball\_by\_ball t1  JOIN  batsman\_scored t2  ON t1.Match\_Id = t2.Match\_Id  AND t1.Over\_Id = t2.Over\_Id  AND t1.Ball\_Id = t2.Ball\_Id  AND t1.Innings\_No = t2.Innings\_No  JOIN  matches t3  ON t1.Match\_Id = t3.Match\_Id  JOIN  season t4  ON t3.Season\_Id = t4.Season\_Id  WHERE  t4.Season\_Year BETWEEN 2012 AND 2016  GROUP BY  t4.Season\_Year  ORDER BY  t4.Season\_Year; |

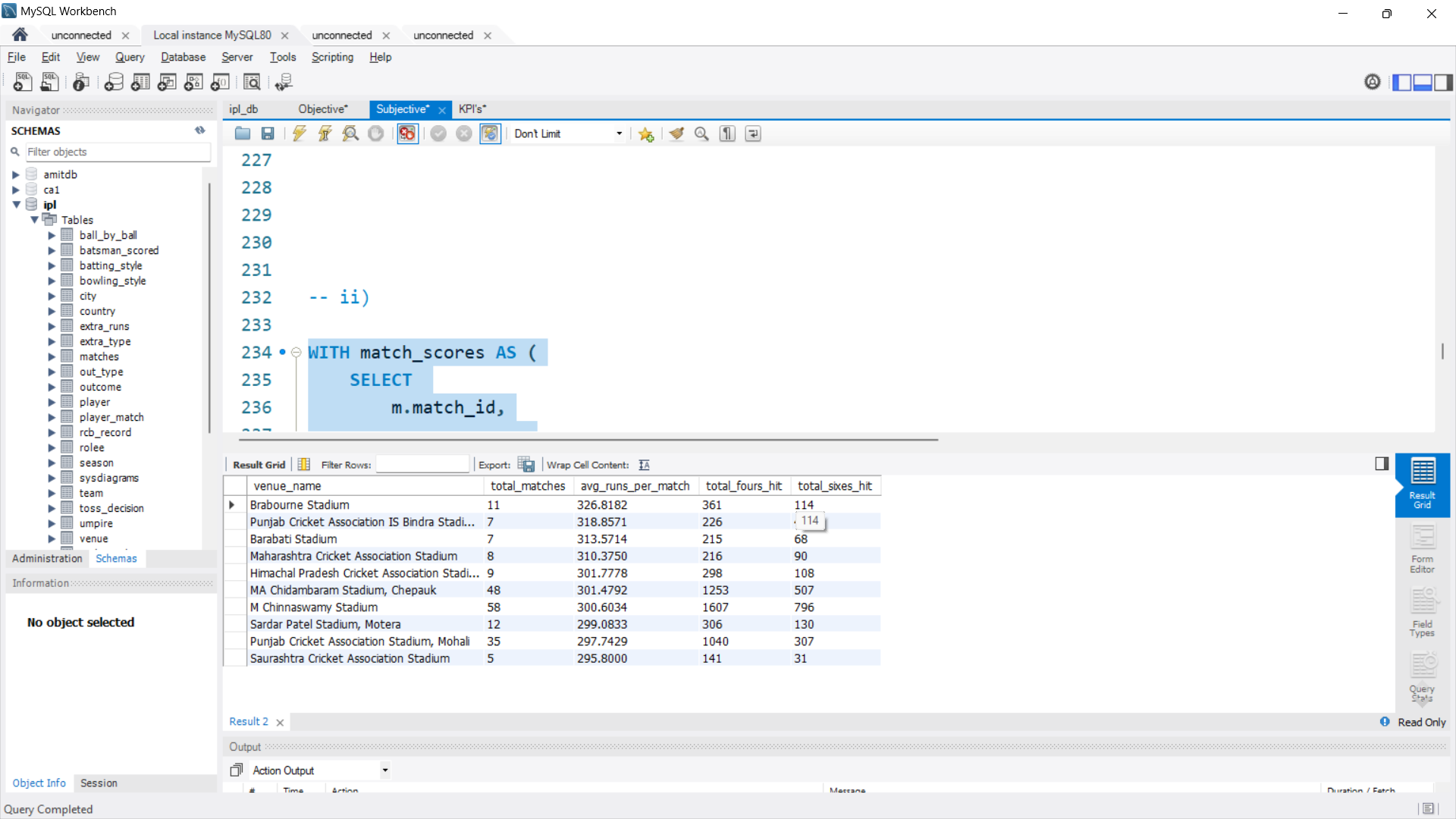
**Result:**

1. **Analyzing High-Scoring IPL Matches: Venue Performance and Contributing Factors.**

**Query:**

|  |
| --- |
| WITH match\_scores AS (  SELECT  m.match\_id,  v.venue\_name,  SUM(bs.runs\_scored) AS total\_runs,  COUNT(CASE WHEN bs.runs\_scored = 4 THEN 1 END) AS total\_fours,  COUNT(CASE WHEN bs.runs\_scored = 6 THEN 1 END) AS total\_sixes  FROM  matches m  JOIN ball\_by\_ball bb  ON m.match\_id = bb.match\_id  JOIN batsman\_scored bs  ON bb.match\_id = bs.match\_id  AND bb.over\_id = bs.over\_id  AND bb.ball\_id = bs.ball\_id  AND bb.innings\_no = bs.innings\_no  JOIN venue v  ON m.venue\_id = v.venue\_id  GROUP BY  m.match\_id, v.venue\_name  ),  venue\_analysis AS (  SELECT  venue\_name,  COUNT(match\_id) AS total\_matches,  AVG(total\_runs) AS avg\_runs\_per\_match,  SUM(total\_fours) AS total\_fours\_hit,  SUM(total\_sixes) AS total\_sixes\_hit  FROM  match\_scores  GROUP BY  venue\_name  )  SELECT  venue\_name,  total\_matches,  avg\_runs\_per\_match,  total\_fours\_hit,  total\_sixes\_hit  FROM  venue\_analysis  ORDER BY  avg\_runs\_per\_match DESC  LIMIT 10; |

**Result**:



1. Analyse the impact of home-ground advantage on team performance and identify strategies to maximize this advantage for RCB.

**ANS-**

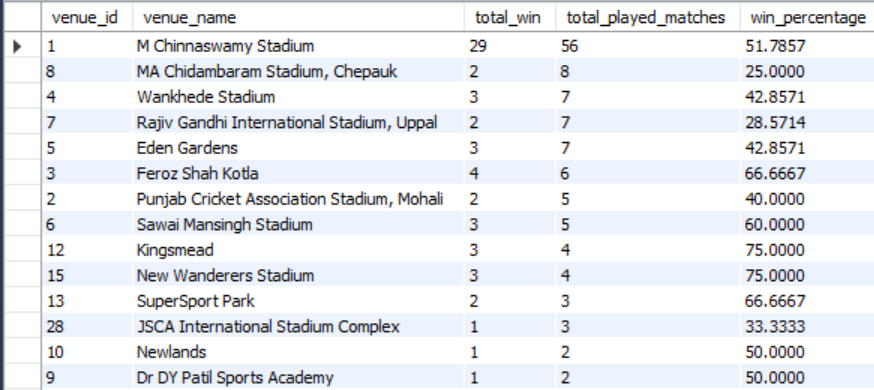
Key points on home ground.

* The concept of home-ground advantage in sports has a significant impact on team performance. Analysing this effect involves understanding how familiar conditions, fan support, and the psychological comfort of playing at home contribute to a team’s success. Here’s a breakdown of the impact and key factors involved in maximizing this advantage:
* **Pitch and Weather Conditions**: Teams are familiar with their home ground’s pitch behaviour, such as whether it Favors batting, spin, or pace.
* **Crowd Influence**: Home crowd support boosts player morale and can pressure the visiting team.
* **Win Rate**: Teams often have higher win rates at home compared to away matches. By analysing historical win rates, teams can evaluate the extent of their advantage when playing on their home turf.
* **Batting and Bowling Averages**: Batsmen and bowlers often perform differently at home versus away due to familiarity with pitch conditions. Analysing these averages can reveal whether players consistently perform better at home.
* **Fielding Metrics:** Teams tend to make fewer fielding errors at home due to familiarity with ground dimensions, sightlines, and lighting, which can reduce the likelihood of dropped catches or misfielding incidents.

**SQL Query to Analyze Home-Ground Advantage:**

|  |
| --- |
| WITH total\_win\_venue AS (  SELECT  m.venue\_id,  v.venue\_name,  COUNT(\*) AS total\_win  FROM  matches m  JOIN  venue v ON m.venue\_id = v.venue\_id  JOIN  team t ON t.team\_id = m.match\_winner  WHERE  t.team\_name = 'Royal Challengers Bangalore'  GROUP BY  m.venue\_id, v.venue\_name  ),  total\_played\_venue AS (  SELECT  venue\_id,  COUNT(\*) AS total\_played\_matches  FROM  matches  WHERE  team\_1 = '2' OR team\_2 = '2'  GROUP BY  venue\_id  )  SELECT  twv.\*,  tpv.total\_played\_matches,  (twv.total\_win / tpv.total\_played\_matches) \* 100 AS win\_percentage  FROM  total\_win\_venue twv  JOIN  total\_played\_venue tpv ON twv.venue\_id = tpv.venue\_id  ORDER BY  total\_played\_matches DESC; |

**Result:**



* Total played matches for every team are more in home ground compared to other venue.

It will increase the chance of winning the team.

* RCB does have not a great winning percentage in the home ground with 51.78%.
* **High-Scoring Games**: The stadium’s batting-friendly conditions lead to high scores, often requiring RCB to maintain a powerful batting lineup.
* **Conclusion**
* The home-ground advantage in cricket has a clear impact on team performance, often providing a boost through familiarity, fan support, and reduced fatigue. By strategically selecting condition-specific players, enhancing fan engagement, and preparing through data-driven analysis, teams can maximize their home-ground advantage. However, they must manage expectations and avoid complacency to ensure consistent performance.

1. Analyse the impact of home-ground advantage on team performance and identify strategies to maximize this advantage for RCB.

To conduct a visual and analytical analysis of **Royal Challengers Bangalore's (RCB)** past seasons' performance and potential reasons for not winning a trophy.

**1. Data Preparation**

You should focus on the following metrics in your analysis:

* **Win Percentage**: RCB's win percentage in each season.
* **Average runs conceded in death over of each team from the last 5 years**
* **Innings Stats**: How RCB performed while batting first vs chasing.
* **Team Performance**: Comparison of RCB’s team performance across different seasons.
* **Key Factors for Losing**: Analysis of matches lost (e.g., by narrow margins, poor batting, bowling, or fielding).

**2. Analytical Insights**

**a. Win Percentage Over Seasons**

Analyze RCB’s win percentage over the years. If you find a trend of low win percentages in the latter stages of seasons, this could point to fatigue, injuries, or tactical errors.

* **Insight**: If RCB has had poor performances in crucial matches (e.g., knockout stages), analyze the specific match conditions or opposition strengths.

**b. Batting and Bowling Analysis**

Track whether RCB consistently had a strong batting line-up but lacked bowling resources or vice versa.

* **Insight**: If RCB's batting has been strong but bowling has been weak, they may need to strengthen their bowling unit, especially in death overs or middle overs.

**c. Home-Ground Advantage**

Assess if RCB performed better at their home ground (M. Chinnaswamy Stadium) and whether the home-ground advantage contributed to their success.

* **Insight**: If RCB had better results at home but struggled away, it could suggest the need for a better overall squad for handling different pitch conditions.

**d. Injuries and Squad Depth**

Examine if injuries to key players (e.g., Virat Kohli, AB de Villiers, or any top bowler) affected RCB’s performance during critical phases.

* **Insight**: RCB may have suffered due to a lack of depth in the squad or failure to replace key players in case of injuries.

**e. Key Matches Lost**

Identify specific seasons where RCB lost matches by narrow margins (e.g., lost in the final or semi-final by a small margin).

* **Insight**: Analyze if there was a consistent pattern of losing close matches (e.g., poor fielding, last-over collapses, or tactical errors).

**f. Team Composition**

Compare RCB's team composition over the years. A mismatch in team balance (e.g., heavy reliance on a few top-order batsmen or a weak bowling attack) could be a key reason for their failure.

* **Insight**: If RCB was over-reliant on a few players like Virat Kohli or AB de Villiers, it might have led to inconsistency when they were unavailable or underperformed.
* **Win Percentage for RCB in Each Season**

WITH rcb\_performance AS (

SELECT

m.Season\_Id,

COUNT(\*) AS total\_matches,

SUM(CASE WHEN m.Match\_Winner = 2 THEN 1 ELSE 0 END) AS total\_wins -- Assuming Team\_Id '2' is RCB

FROM

matches m

WHERE

m.Team\_1 = 2 OR m.Team\_2 = 2 -- Matches involving RCB

GROUP BY

m.Season\_Id

)

SELECT

s.Season\_Year,

rp.total\_matches,

rp.total\_wins,

ROUND((rp.total\_wins / rp.total\_matches) \* 100, 2) AS win\_percentage

FROM

rcb\_performance rp

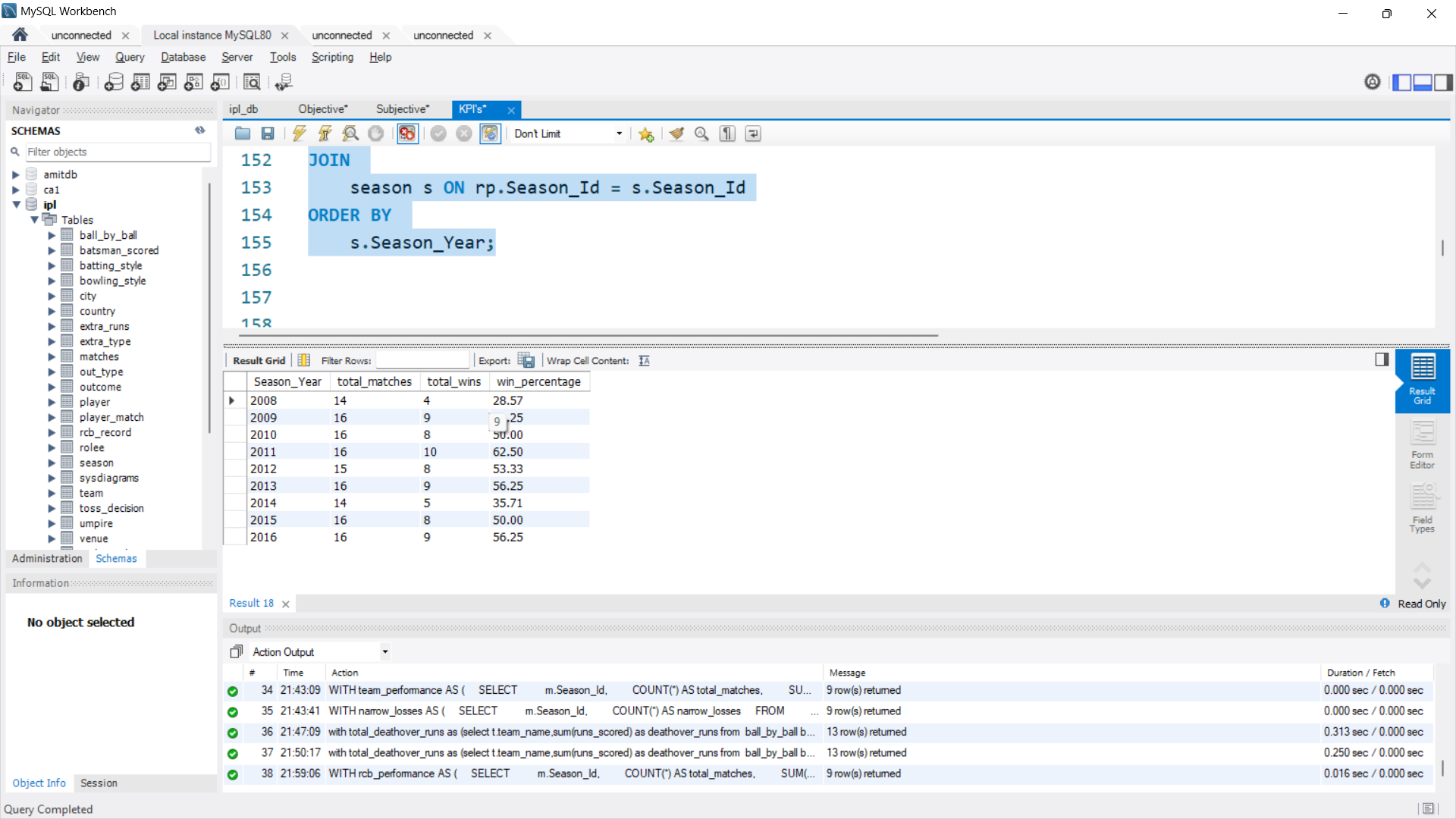
JOIN

season s ON rp.Season\_Id = s.Season\_Id

ORDER BY

s.Season\_Year;

**Result:**



**Visualization:**

* **Average runs conceded in death over of each team from the last 5 years**

with total\_deathover\_runs as (select t.team\_name,sum(runs\_scored) as deathover\_runs

from ball\_by\_ball bb join batsman\_scored bs on bb.match\_id=bs.match\_id and bb.over\_id=bs.over\_id

and bb.ball\_id=bs.ball\_id and bb.innings\_no=bs.innings\_no

join team t on t.team\_id=bb.team\_bowling

join matches m on m.match\_id=bb.match\_id

join season s on m.season\_id=s.season\_id

where bb.over\_id>=16

group by t.team\_name),

total\_matches as (select t.team\_name,count(distinct m.match\_id) as total\_matches

from matches, m join team t on m.team\_1=t.team\_id or m.team\_2=t.team\_id

join season s on m.season\_id=s.season\_id

group by t.team\_name)

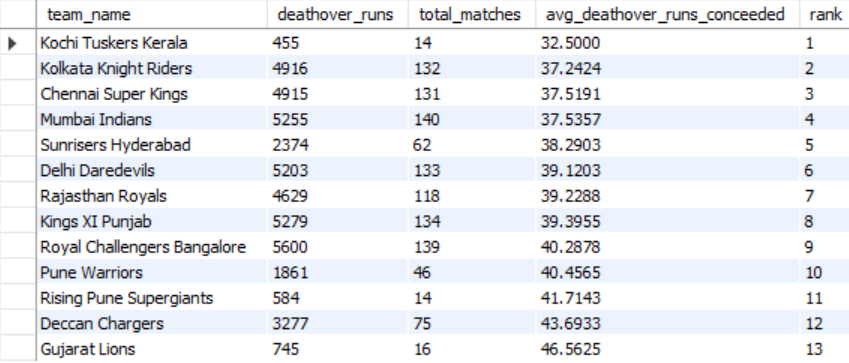
select tdr.\*,tm.total\_matches,(tdr.deathover\_runs/tm.total\_matches) as avg\_deathover\_runs\_conceeded,

dense\_rank() over(order by (tdr.deathover\_runs/tm.total\_matches) ) as 'rank'

from total\_deathover\_runs tdr join total\_matches tm on tdr.team\_name=tm.team\_name

order by 'rank';

Result:

****

* **RCB’s Performance While Batting First vs Chasing**

WITH rcb\_batting\_stats AS (

SELECT

m.Season\_Id,

SUM(CASE WHEN m.Team\_1 = 2 THEN 1 ELSE 0 END) AS matches\_batting\_first,

SUM(CASE WHEN m.Team\_2 = 2 THEN 1 ELSE 0 END) AS matches\_chasing

FROM

matches m

WHERE

m.Team\_1 = 2 OR m.Team\_2 = 2 -- Matches involving RCB

GROUP BY

m.Season\_Id

)

SELECT

s.Season\_Year,

rbs.matches\_batting\_first,

rbs.matches\_chasing

FROM

rcb\_batting\_stats rbs

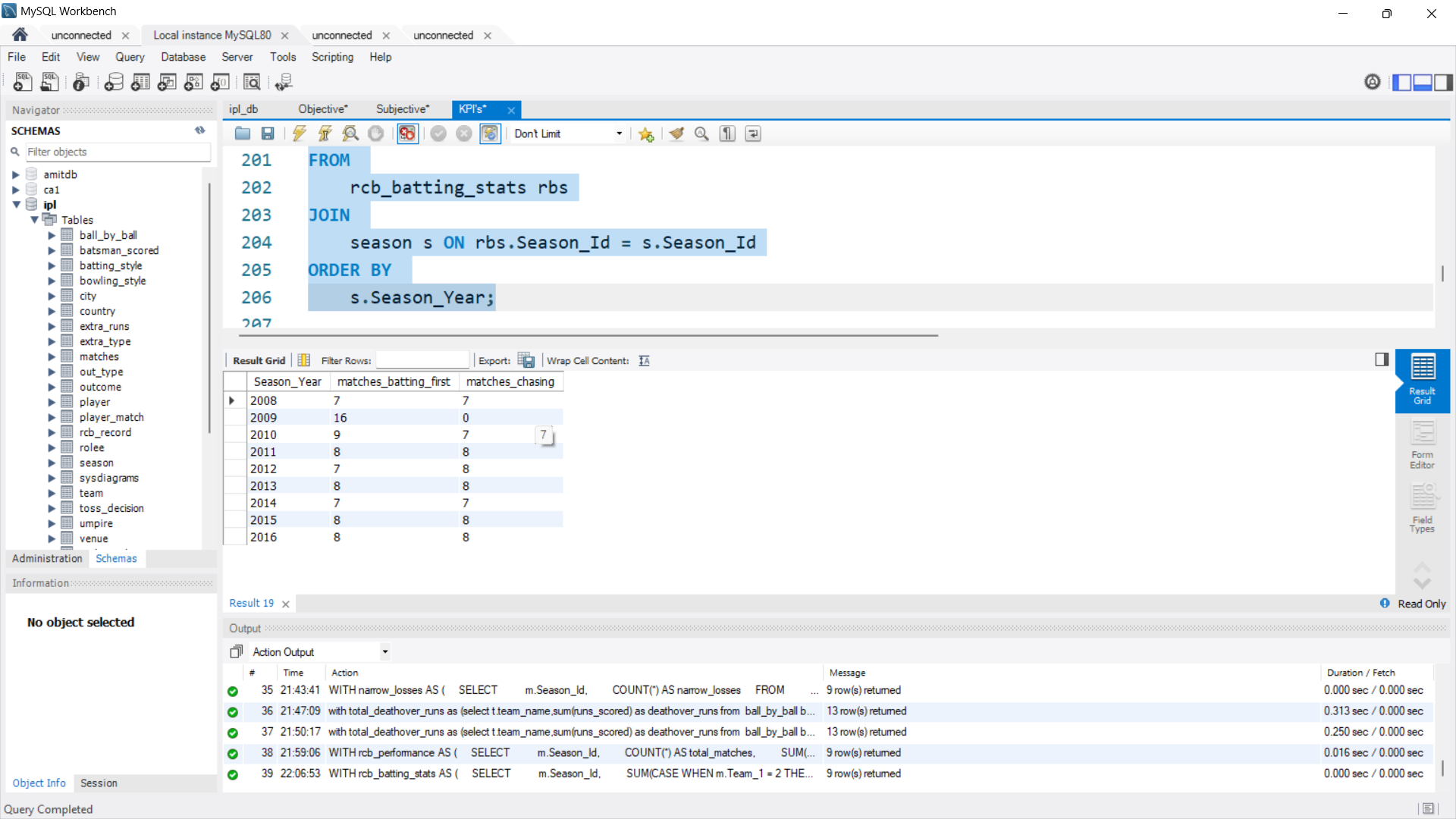
JOIN

season s ON rbs.Season\_Id = s.Season\_Id

ORDER BY

s.Season\_Year;

Result:



Visualization:

* **Comparison of RCB's Team Performance Across Seasons**

WITH rcb\_bowling AS (

SELECT

p.Player\_Name,

COUNT(\*) AS total\_wickets,

m.Season\_Id

FROM

wicket\_taken wt

JOIN

ball\_by\_ball bb ON wt.Match\_Id = bb.Match\_Id AND wt.Over\_Id = bb.Over\_Id AND wt.Ball\_Id = bb.Ball\_Id -- Join with ball\_by\_ball to get the bowler

JOIN

player p ON bb.Bowler = p.Player\_Id -- Get the bowler from ball\_by\_ball

JOIN

matches m ON wt.Match\_Id = m.Match\_Id

JOIN

player\_match pm ON p.Player\_Id = pm.Player\_Id

WHERE

pm.Team\_Id = 2 -- RCB team

GROUP BY

p.Player\_Name, m.Season\_Id

)

SELECT

s.Season\_Year,

SUM(rb.total\_wickets) AS total\_wickets\_taken

FROM

rcb\_bowling rb

JOIN

season s ON rb.Season\_Id = s.Season\_Id

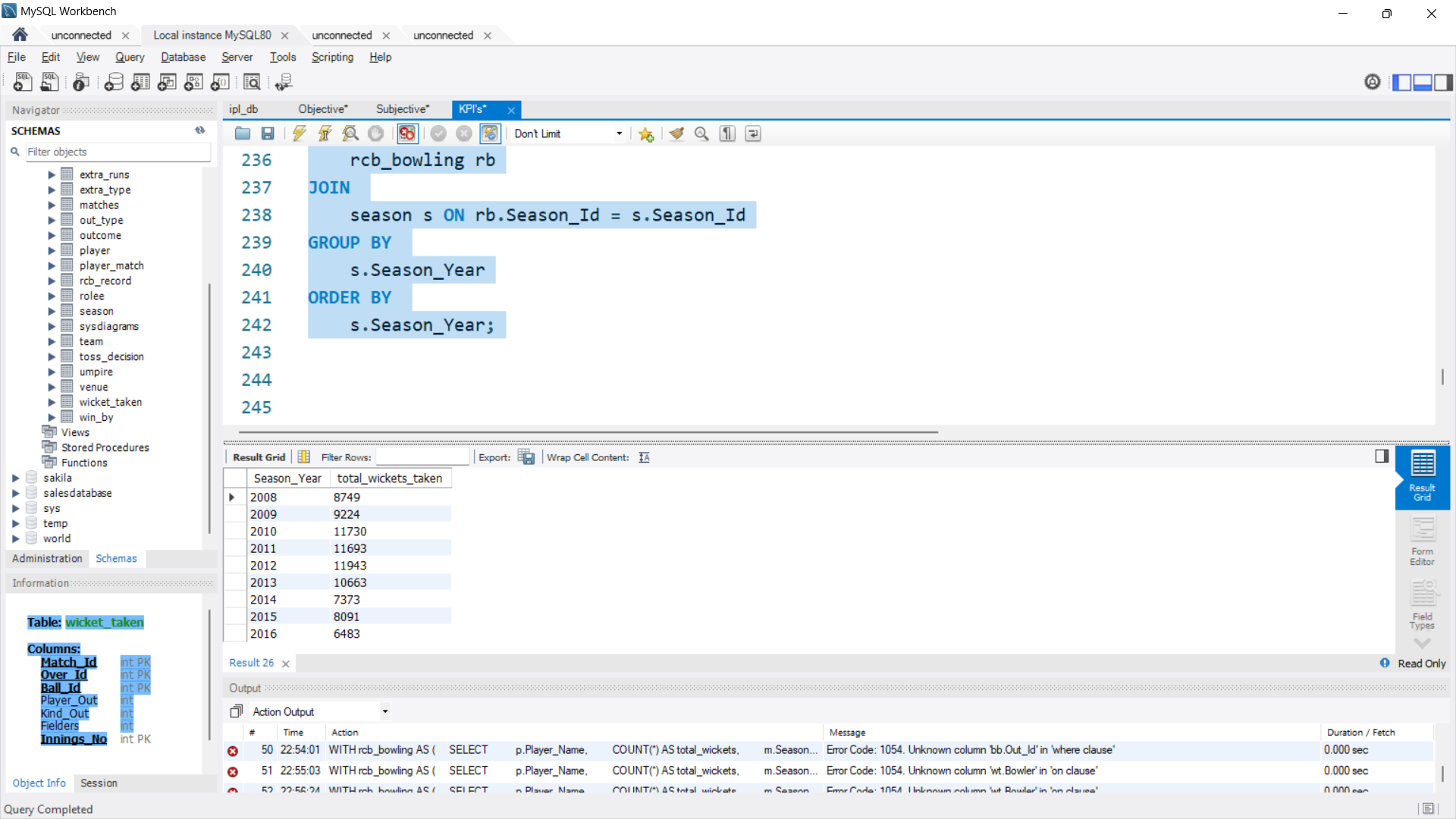
GROUP BY

s.Season\_Year

ORDER BY

s.Season\_Year;

**Result:**



* **Key Factors for Losing (Losses in Narrow Margins)**

WITH narrow\_losses AS (

SELECT

m.Season\_Id,

COUNT(\*) AS narrow\_losses

FROM

matches m

WHERE

(m.Team\_1 = 2 OR m.Team\_2 = 2) -- Matches involving RCB

AND m.Match\_Winner != 2 -- RCB lost

AND m.Win\_Margin < 20 -- Narrow margin loss (example: < 20 runs or wickets)

GROUP BY

m.Season\_Id

)

SELECT

s.Season\_Year,

nl.narrow\_losses

FROM

narrow\_losses nl

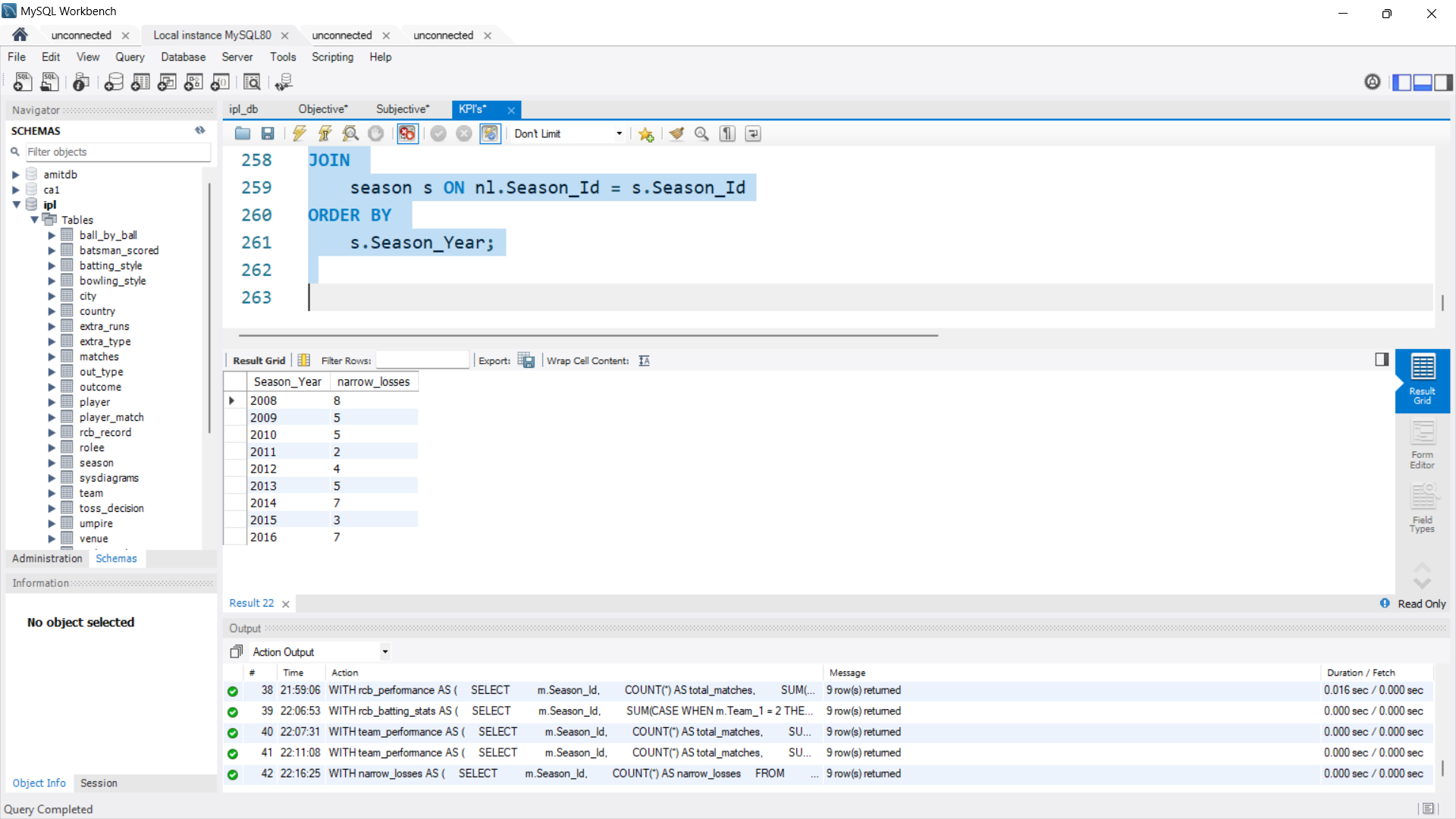
JOIN

season s ON nl.Season\_Id = s.Season\_Id

ORDER BY

s.Season\_Year;

Result:



* RCB’s death overs runs conceded average is 40.28 is very high compared to the top rank teams.
* **Conclusion:**
* Overdependency on key players.
* Inconsistence bowling lineup.
* bad home ground performance.

These are the potential reasons for RCB not to win the trophy.

10. How would you approach this problem, if the objective and subjective questions weren't given?

Ans-

If the objective and subjective questions weren’t provided, I’d approach analyzing RCB’s performance with a slightly different framework, focusing on broader performance evaluation and scenario-based analysis.

**Step 1: Understand the Context and Define Scope**

* **Objective Setting**: Without predefined goals, I’d focus on understanding RCB’s overall success and failure patterns in the IPL.
* **Scope of Analysis**: Emphasize team dynamics, and individual performances, and match scenarios to identify areas for improvement.

**Step 2: Perform Data Profiling and Exploration**

* **Database Exploration**: Review available tables, relationships, and data quality. Identify key tables like matches, ball\_by\_ball, player\_match, etc.
* **Data Patterns**: Understand recurring trends in team totals, player scores, and match outcomes.

**Step 3: Create Analytical Frameworks**

Instead of predefined KPIs, I’d derive new analytical frameworks:

1. **Game Situations Analysis**
   * **Performance Under Pressure**: Examine batting collapses or clutch bowling in high-pressure situations (e.g., final overs in tight games).
   * **Toss and Conditions**: Analyze how decisions (batting or fielding first) influenced match results based on toss outcomes and venues.
2. **Momentum Metrics**
   * **Impact of Powerplay**: Measure runs scored or wickets taken during powerplay overs to assess early momentum.
   * **Death Overs Efficiency**: Analyze batting strike rates and bowling economy in death overs (16–20).
   * **Middle Overs Consistency**: Evaluate stability in overs 7–15, focusing on partnerships and dot ball percentages.
3. **Player Role Optimization**
   * **Role-Based Impact**: Assess contributions of players in specific roles (e.g., finishers, strike bowlers, all-rounders).
   * **Match-Winning Performances**: Identify games where individual performances were decisive.
4. **Opponent-Specific Insights**
   * Examine RCB’s performance against specific teams to uncover potential rivalries or challenges.
   * Study head-to-head data to identify favourable matchups or frequent pitfalls.

**Step 4: Statistical Modeling for Deeper Insights**

* **Win Probability Models**: Develop models to predict match outcomes based on live match situations like wickets in hand or required run rate.
* **Clustering Analysis**: Group players based on their performance metrics to identify consistent contributors and outliers.

**Step 5: Formulate Hypotheses and Test Them**

* **Hypotheses**:
  1. RCB struggles with chasing high totals at specific venues.
  2. Over-reliance on top-order batting impacts middle-order contributions.
  3. Bowling economy in death overs correlates strongly with match outcomes.
* Test these using SQL queries and visual analytics.

**Step 6: Recommendations Based on Data-Driven Insights**

1. **Game Strategy**:
   * Prioritize partnerships and role clarity to avoid batting collapses.
   * Optimize bowler selection for specific venues based on historical performance.
2. **Squad Adjustments**:
   * Identify gaps in player roles and suggest acquisitions during auctions or team restructuring.
3. **Scenario-Specific Training**:
   * Focus on improving death-over performances, both in batting and bowling.

**Step 7: Advanced Visualization Techniques**

* Use **flow charts** to illustrate match progression and turning points.
* Create **radar charts** to compare batting and bowling strengths across teams.
* Build **interactive dashboards** to allow detailed exploration of venue-specific and player-specific performance.

This approach would provide actionable insights even without predefined objectives or questions.

* 1. In the "Match" table, some entries in the "Opponent\_Team" column are incorrectly spelt as "Delhi\_Capitals" instead of "Delhi\_Daredevils". Write an SQL query to replace all occurrences of "Delhi\_Capitals" with "Delhi\_Daredevils".

ANS-

There is no Match table and no column with Opponent\_team in any of the tables.

The available column with the team name is Team\_name in the table Team.

And there is no such name available like "Delhi\_Capitals"

If the table and column exist the query will be like this.

* **Query:**

SET SQL\_SAFE\_UPDATES = 0;

UPDATE team

SET Team\_Name = 'Delhi Capitals'

WHERE Team\_Name = 'Delhi Daredevils';

SET SQL\_SAFE\_UPDATES = 1; -- Re-enable safe updates if desired

* **Explanation**
* **UPDATE TEAMS**: Targets the Teams table for updates.
* **SET Team\_name = 'Delhi\_Daredevils'**: Replace the team name in the team table.
* **WHERE Team\_name= 'Delhi\_Capitals'**: Ensures that only rows with the value "Delhi\_Capitals".

After running this query, all instances of "Delhi\_Capitals" in Opponent\_Team will be corrected to "Delhi\_Daredevils.

* **Output**