### **Project 3: IPL Analysis using SQL**

The Indian Premier League (IPL) is one of the most popular cricket leagues in the world. A dataset containing information about IPL matches and player statistics is available for analysis. As a data analyst with SQL expertise, your objective is to perform data analysis on the IPL dataset to gain insights into player performance, team dynamics, and match outcomes.

#### Segment 1: Database - Tables, Columns, Relationships

Seg1\_Q1) Identify the tables in the dataset and their respective columns.

```
# database.sqlite

1 -- Segment 1: Database - Tables, Columns, Relationships
2 -- - 1) Identify the tables in the dataset and their respective columns.
3
4 SELECT name FROM sqlite_schema WHERE type='table' ORDER BY name;
5
6 |

I name

Ball_by_Ball

Batsman_Scored

Batting_Style

City

Country
```

SELECT name FROM sqlite\_schema
WHERE type='table' ORDER BY name;

#### **Table Name**

	Ball b	Batsm	Battin	Bowlin		Count	Extra	Extra		Out T	Outco		Player		Seaso	Toss_	Umpir		Wicket	Win B	sqlite	svsdia
nam			g_Styl	g_Styl	City		Runs	Type	Match	ype	me	Player	_Matc	Rolee	n	Decisi		Venue	_Take	у	_sequ	grams
		ored	4	ť									- 11			on			n		ence	

#### Columns Details:

```
PRAGMA table_info(Ball_by_Ball);
PRAGMA table_info(Batsman_Scored);
PRAGMA table_info(Batting_Style);
PRAGMA table_info(Bowling_Style);
PRAGMA table_info(Country);
PRAGMA table_info(Country);
PRAGMA table_info(Extra_Runs);
PRAGMA table_info(Extra_Type);
PRAGMA table_info(Match);
PRAGMA table_info(Out_Type);
PRAGMA table_info(Out_Type);
PRAGMA table_info(Outcome);
PRAGMA table_info(Outcome);
```

```
PRAGMA table_info(Player_Match);
PRAGMA table_info(Rolee);
PRAGMA table_info(Season);
PRAGMA table_info(Team);
PRAGMA table_info(Toss_Decision);
PRAGMA table_info(Umpire);
PRAGMA table_info(Venue);
PRAGMA table_info(Wicket_Taken);
PRAGMA table_info(Win_By);
PRAGMA table_info(sqlite_sequence);
PRAGMA table_info(sysdiagrams);
```

	PRAGMA table_info(Ball_by_Ball);				
cid	name	type	notnull	dflt_value	pk
0	Match_Id	INTEGER	1	null	1
1	Over_ld	INTEGER	1	null	2
2	Ball_ld	INTEGER	1	null	3
3	Innings_No	INTEGER	1	null	4
4	Team_Batting	INTEGER	1	null	0
5	Team_Bowling	INTEGER	1	null	0
6	Striker_Batting_Position	INTEGER	1	null	0
7	Striker	INTEGER	1	null	0
8	Non_Striker	INTEGER	1	null	0
9	Bowler	INTEGER	1	null	0

	PRAGMA table_info(Batsman_Scored);				
cid	name	type	notnull	dflt_value	pk
0	Match_Id	INTEGER	1	null	1
1	Over_ld	INTEGER	1	null	2
2	Ball_Id	INTEGER	1	null	3
3	Runs_Scored	INTEGER	1	null	0
4	Innings No	INTEGER	1	null	4

	PRAGMA table_info(Batting_Style);				
cid	name	type	notnull	dflt_value	pk
0	Batting_Id	INTEGER	1	null	1
1	Batting_hand	archar(200	1	null	0

		PRAGMA table_info(Bowling_Style);				
k	cid	name	type	notnull	dflt_value	pk
1	0	Bowling_Id	INTEGER	1	null	1
0	1	Bowling_skill	archar(200	1	null	0

	PRAGMA table_info(City);				
cid	name	type	notnull	dflt_value	pk
0	City_Id	INTEGER	1	null	1
1	City_Name	varchar(200)	1	null	0
2	Country_id	INTEGER	0	null	0
	PRAGMA table_info(Country);				
cid	name	type	notnull	dflt_value	pk
0	Country_Id	INTEGER	1	null	1
1	Country_Name	varchar(200)	1	null	0
	PRAGMA table_info(Extra_Runs);				
cid	name	type	notnull	dflt_value	pk
0	Match_Id	INTEGER	1	null	1
1	Over_Id	INTEGER	1	null	2
2	Ball_Id	INTEGER	1	null	3
3	Extra_Type_Id	INTEGER	1	null	0
4	Extra_Runs	INTEGER	1	null	0
5 Innings No		INTEGER	1	null	4

cid	name	type	notnull	dflt_value	р
0	Extra_Id	INTEGER	1	null	
1	Extra_Name	varchar(150)	1	null	(
	PRAGMA table_info(Match);				
cid	name	type	notnull	dflt_value	F
0	Match_Id	INTEGER	1	null	
1	Team_1	INTEGER	1	null	•
2	Team_2	INTEGER	1	null	-
3	Match_Date	datetime	1	null	-
4	Season_Id	INTEGER	1	null	•
5	Venue_Id	INTEGER	1	null	-
6	Toss_Winner	INTEGER	1	null	-
7	Toss_Decide	INTEGER	1	null	-
8	Win_Type	INTEGER	1	null	-
9	Win_Margin	INTEGER	0	null	•
10	Outcome_type	INTEGER	1	null	-
11	Match_Winner	INTEGER	0	null	-
12	Man_of_the_Match	INTEGER	0	null	
	PRAGMA table_info(Outcome);				
cid	name	type	notnull	dflt_value	F
0	Outcome_Id	INTEGER	1	null	
	Outcome_Type	varchar(200)	1	null	

cid	name	type	notnull	dflt value	r
0	Out Id	INTEGER	1	null	
1	Out_Name	varchar(250)	1	null	
	PRAGMA table_info(Player);				
cid	name	type	notnull	dflt_value	р
0 Player_ld		INTEGER	1	null	
1	Player_Name	varchar(400)	1	null	
2	DOB	datetime	0	null	
3	Batting_hand	INTEGER	1	null	Ī
4	Bowling_skill	INTEGER	0	null	
5	5 Country_Name		1	null	-
	PRAGMA table_info(Player_Match);				
cid	name	type	notnull	dflt_value	μ
0	Match_Id	INTEGER	1	null	
1	Player_Id	INTEGER	1	null	
2	Role_Id	INTEGER	1	null	-
3	Team_Id	INTEGER	0	null	
	PRAGMA table_info(Rolee);				
cid	name	type	notnull	dflt_value	р
0	Role_Id	INTEGER	1	null	
1	Role_Desc	varchar(150)	1	null	
	PRAGMA table_info(Season);				
cid	name	type	notnull	dflt_value	p
0	Season_Id	INTEGER	1	null	
1	Man_of_the_Series	INTEGER	1	null	Ī
2	Orange_Cap	INTEGER	1	null	Ī
3	Purple_Cap	INTEGER	1	null	-
4	Season Year	INTEGER	0	null	ŀ

_									
	able_info(Win_By);								
5									
i cid	name	type	notnull	dflt value	pk				
- ciu	Hallie	туре	Houlun	unt_value	pκ				
0	Win_ld	INTEGER		NULL					
1	Win_Type	varchar(200)		NULL					
	<u> </u>	·	.,						

41.4	PRAGMA table_info(sqlite_sequence);					
cid	name	type	notnull	dflt_value	pk	
0	name		0	null	0	
1	seq		0	null	0	
	PRAGMA table_info(sysdiagrams);					
cid	name	type	notnull	dflt value	pk	
0	name	nvarchar(128)	1	null	0	
1	principal id	INTEGER	1	null	0	
2	diagram_id	INTEGER	1	null	1	
3	version	INTEGER	0	null	0	
4	definition	BLOB	0	null	0	
	DDACMA table info/Toom).					
cid	PRAGMA table_info(Team);	tuno	notnull	dflt value	nle	
0	Team Id	type INTEGER	nothuii 1	null	pk 1	
1	Team Name	varchar(450)	1	null	0	
1	ream_Name	varcilar(450)	1	IIIII	U	
	PRAGMA table_info(Toss_Decision);					
cid	name	type	notnull	dflt_value	pk	
0	Toss_Id	INTEGER	1	null	1	
1	Toss_Name	varchar(50)	1	null	0	
-2.4	PRAGMA table_info(Umpire);			101 .1 .	- 1	
cid	name	type	notnull	dflt_value	pk	
0	Umpire_Id	INTEGER	1	null	1	
1	Umpire_Name	varchar(350)	1	null	0	
2	Umpire_Country	INTEGER	1	null	0	
	PRAGMA table_info(Venue);					
cid	name	type	notnull	dflt_value	pk	
0	Venue_Id	INTEGER	1	null	1	
1	Venue_Name	varchar(450)	1	null	0	
2	City_Id	INTEGER	0	null	0	
	PRAGMA table_info(Wicket_Taken);					
cid	name	type	notnull	dflt_value	pk	
0	Match_Id	INTEGER	1	null	1	
1	Over_Id	INTEGER	1	null	2	
2	Ball_Id	INTEGER	1	null	3	
3	Player_Out	INTEGER	1	null	0	
4	Kind_Out	INTEGER	1	null	0	
5	Fielders	INTEGER	0	null	0	
6	Innings_No	INTEGER	1	null	4	
	PRAGMA table_info(Win_By);					
cid	name	type	notnull	dflt_value	pk	
0	Win_ld	INTEGER	1	null	1	

Seq1\_Q2) Determine the number of rows in each table within the schema.

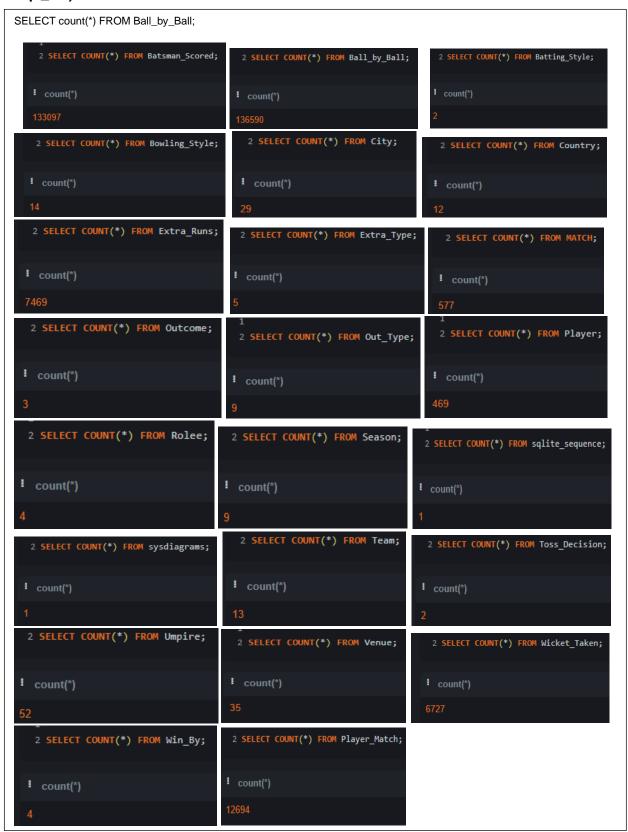


Table-name	no.of rows			
Ball_by_Ball	136590			
Batsman_Scored	133097			
Batting_Style	2			
Bowling_Style	14			
City	29			
Country	12			
Extra_Runs	7469			
Extra_Type	5			
Match	577			
Out_Type	9			
Outcome	3			
Player	469			
Player_Match	12694			
Rolee	4			
Season	9			
Team	13			
Toss_Decision	2			
Umpire	52			
Venue	35			
Wicket_Taken	6727			
Win_By	4			
sqlite_sequence	1			
sysdiagrams	1			

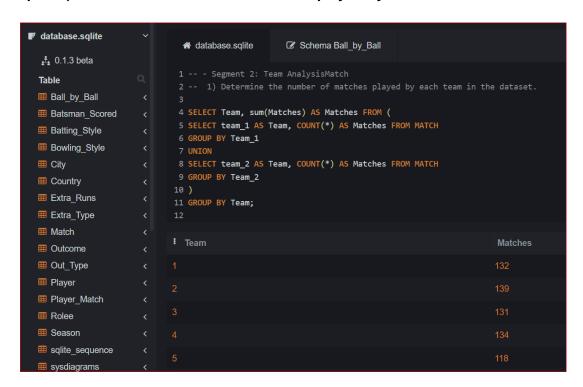
## Seg1\_Q3) Handle any missing or inconsistent values in the dataset.

In given database NULL values in all schema is less than 5%, can be neglected.

We can use imputation technique like Mean, Median, Mode to remove Null from database.

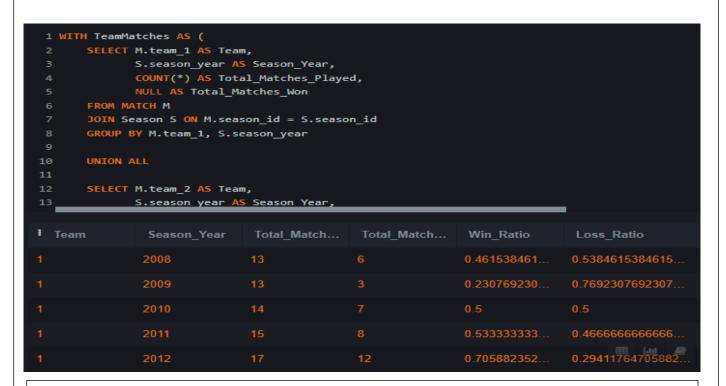
## **Segment 2: Team Analysis**

Seq2\_Q1) Determine the number of matches played by each team in the dataset.



ı	Matches	
1	132	
2	139	
3	131	
4	134	
5	118	
6	133	Select Team, sum(Matches) as Matches From (
7	140	Select team_1 as Team, COUNT(*) as Matches FROM
8	75	Match
9	7	GROUP BY team_1
10	23	UNION
11	62	Select team_2 as Team, COUNT(*) as Matches FROM
12	7	Match
13	16	GROUP BY team_2

#### Seg2\_Q2) Analyse the win-loss ratio for each team in IPL history



#### Note:

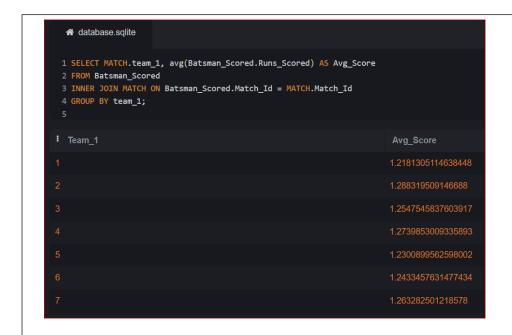
Table Match contain team1, team2, Win\_Type, Match\_Winner, Season column and now i want to calculate Team, Toatal\_match\_played, Total\_Match\_team\_won and win ratio and Loss ratio.

#### i have to write different quires then how i can combine all ????????

To combine all the queries into a single query, you can use Common Table Expressions (CTEs). CTEs allow you to create temporary result sets that you can reference multiple times within a single query. Here's how you can rewrite the previous queries using CTEs to get all the required information in one query

```
WITH TeamMatches AS (
 SELECT M.team_1 AS Team,
     S.season_year AS Season_Year,
     COUNT(*) AS Total_Matches_Played,
     NULL AS Total_Matches_Won
 FROM Match M
 JOIN Season S ON M.season_id = S.season_id
 GROUP BY M.team_1, S.season_year
 UNION ALL
 SELECT M.team_2 AS Team,
     S.season_year AS Season_Year,
     COUNT(*) AS Total_Matches_Played,
     NULL AS Total_Matches_Won
 FROM Match M
 JOIN Season S ON M.season_id = S.season_id
 GROUP BY M.team_2, S.season_year
TeamWins AS (
 SELECT M.match_winner AS Team,
    S.season_year AS Season_Year,
     NULL AS Total_Matches_Played,
     COUNT(*) AS Total_Matches_Won
 FROM Match M
 JOIN Season S ON M.season_id = S.season_id
 WHERE M.Win_Type <> 3 AND M.Win_Type <> 4
 GROUP BY M.match_winner, S.season_year
)
SELECT Team,
   Season_Year,
   SUM(Total_Matches_Played) AS Total_Matches_Played,
   IFNULL(SUM(Total_Matches_Won), 0) AS Total_Matches_Won,
   (IFNULL(SUM(Total_Matches_Won), 0) * 1.0 / SUM(Total_Matches_Played)) AS Win_Ratio,
   ((SUM(Total_Matches_Played) - IFNULL(SUM(Total_Matches_Won), 0)) * 1.0 / SUM(Total_Matches_Played)) AS
Loss_Ratio
FROM
 SELECT * FROM TeamMatches
 UNION ALL
 SELECT * FROM TeamWins
) AS TeamStats
GROUP BY Team, Season_Year;
```

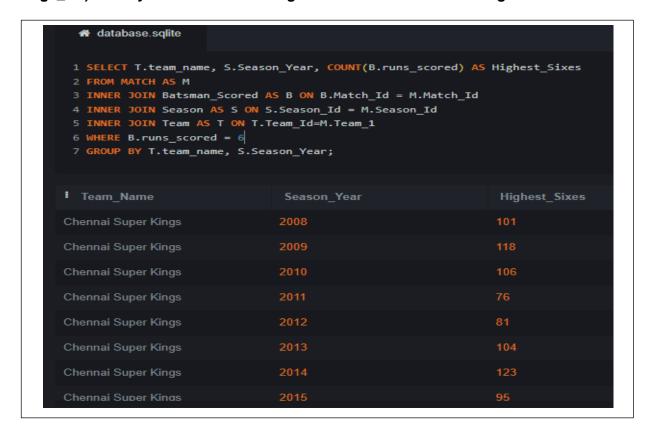
Seg2\_Q3) Calculate the average total runs scored by each team in all matches.



select Match.team\_1, avg(Batsman\_Scored.Runs\_Scored) as Avg\_Score
From Batsman\_Scored
INNER JOIN Match ON Batsman\_Scored.Match\_Id = Match.Match\_Id
group by team\_1;

Team	Avg_Score	
1	1.218131	
2	1.28832	
3	1.254755	
4	1.273985	
5	1.23009	
6	1.243346	
7	1.263283	
8	1.236554	
9	1.186922	
10	1.131369	
11	1.21329	
12	1.304886	
13	1.297496	

#### Seg2\_Q4) Identify the team with the highest number of sixes in a single season.



```
SELECT T.team_name, S.Season_Year, Count(B.runs_scored) AS Highest_Sixes
FROM Match AS M
INNER JOIN Batsman_Scored AS B ON B.Match_Id = M.Match_Id
INNER JOIN Season AS S ON S.Season_Id = M.Season_Id
INNER JOIN Team AS T ON T.Team_Id=M.Team_1
WHERE B.runs_scored = 6
GROUP BY T.team_name, S.Season_Year;
```

#### Seg2\_Q5) Determine the team that has won the most IPL titles.

```
2 SELECT T.team_name, COUNT(M.match_winner) AS Winning_count
3 FROM (
4 SELECT CASE WHEN team_1 = match_winner THEN team_1 END AS team_id, match_winner
5 FROM MATCH
6 ) AS M
7 JOIN Team AS T ON T.team_id = M.team_id
8 WHERE M.team_id IS NOT NULL
9 GROUP BY T.Team_Id;
10
11

! Team_Name Winning_count

Kolkata Knight Riders 37

Royal Challengers Bangalore 41

Chennai Super Kings 50

Kings XI Punjab 31
```

Team_Name	Winning_count	
Kolkata Knight Riders	37	
Royal Challengers Bangalore	41	
Chennai Super Kings	50	
Kings XI Punjab	31	
Rajasthan Royals	35	
Delhi Daredevils	30	
Mumbai Indians	39	
Deccan Chargers	13	
Kochi Tuskers Kerala	3	
Pune Warriors	6	
Sunrisers Hyderabad	17	
Rising Pune Supergiants	2	
Gujarat Lions	4	

```
SELECT T.team_name, count(M.match_winner) AS Winning_count
FROM (
    SELECT CASE WHEN team_1 = match_winner THEN team_1 END AS team_id, match_winner
    FROM Match
) AS M
JOIN Team AS T ON T.team_id = M.team_id
WHERE M.team_id IS NOT NULL
GROUP BY T.Team_Id;
```

### **Segment 3: Player Performance Analysis**

Seq3\_Q1) Identify the top five players with the most runs scored in IPL history.



Player_Name	Total_Runs	
SK Raina	43101	
MS Dhoni	41667	
RG Sharma	41431	
V Kohli	40563	
KD Karthik	39424	

Select 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 Batsman\_Scored bs on pm.match\_id=bs.Match\_Id
GROUP by p.player\_name
order by Total\_Runs DESC
limit 5;

#### Seq3\_Q2) Determine the average strike rate for batsmen who have played at least 50 matches.

```
1 -- Determine the average strike rate for batsmen who have played at least 50 matches.

2
3 SELECT p.player_name, AVG((bs.Runs_scored/bs.ball_id)*100) AS Strike_Rate
4 FROM Player p
5 JOIN Player_Match pm ON p.Player_Id = pm.Player_Id
6 JOIN Batsman_Scored bs ON pm.Match_Id = bs.Match_Id
7 GROUP BY p.Player_Name
8 HAVING COUNT(DISTINCT pm.Match_Id) >= 50
9 ORDER BY Strike_Rate DESC;
10

1 Player_Name
Strike_Rate

AB de Villiers
36.52096738406922

CH Gayle
36.51898734177215

DA Miller
35.994071564683466

Mandeep Singh
35.837056106610135

MM Sharma
35.836225827017074
```

```
SELECT p.player_name, AVG((bs.Runs_scored/bs.ball_id)*100) as Strike_Rate
From Player p
join Player_Match pm ON p.Player_Id = pm.Player_Id
Join Batsman_Scored bs On pm.Match_Id = bs.Match_Id
GROUP by p.Player_Name
HAVING COUNT(DISTINCT pm.Match_Id) >= 50
order by Strike_Rate Desc;
```

#### Seq3\_Q3) Identify the top three bowlers with the most wickets in a single season.

```
1 -- Identify the top three bowlers with the most wickets in a single season.

2
3 SELECT p.player_name, s.Season_Year, COUNT(w.player_out) AS Most_Wicket_Taker
4 FROM Wicket_Taken w
5 JOIN MATCH m ON m.match_id = w.match_id
6 JOIN Player_Match pm ON pm.match_id = w.match_id
7 JOIN Player p ON p.player_id = pm.player_id
8 JOIN Season s ON s.Season_Id = m.Season_Id
9 GROUP BY p.player_name
10 ORDER BY Most_Wicket_Taker DESC LIMIT 3 ;
11

i Player_Name

Season_Year

Most_Wicket_Taker

RG Sharma

2008

1708

SK Raina

2008

1678

KD Karthik

2008

1652
```

```
SELECT p.player_name, s.Season_Year, COUNT(w.player_out) AS Most_Wicket_Taker FROM Wicket_Taken w

JOIN Match m ON m.match_id = w.match_id

JOIN Player_Match pm ON pm.match_id = w.match_id

JOIN Player p ON p.player_id = pm.player_id

JOIN Season s ON s.Season_Id = m.Season_Id

GROUP By p.player_name

Order by Most_Wicket_Taker DESC limit 3;
```

Note: The data is wrong to calculate. In the "Wicket Taken" table column, Player\_out contains information about the batsman out and not the wicket-taking baller.

## Segment 4: Segment 4: Match Analysis

Seg4\_Q1) Determine the total number of matches played in each season.

```
1 -- Determine the total number of matches played in each season.

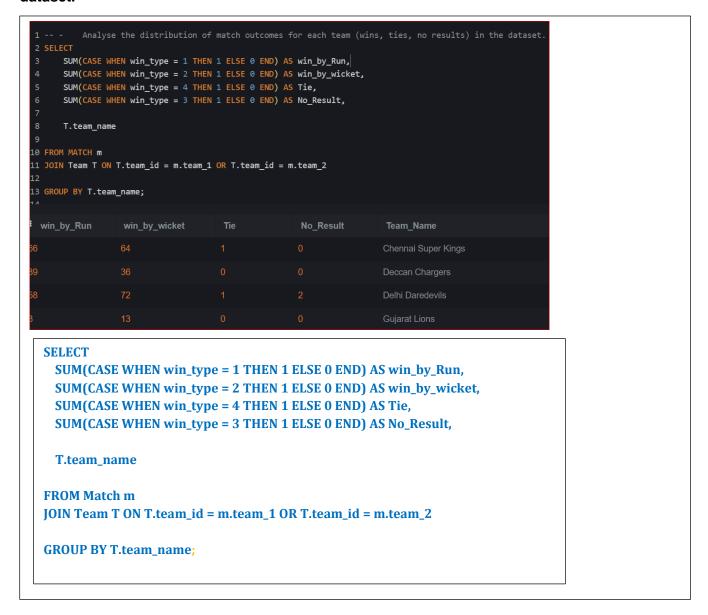
2
3 SELECT s.season_year , COUNT(m.match_id) AS Total_Match_Played
4 FROM MATCH m
5 JOIN Season s ON s.season_id=m.season_id
6 GROUP BY s.season_year;

7
2
i Season_Year Total_Match_Played
2008 58
2009 57
2010 60
2011 73
2012 74
2013 76
2014 60
2015 59
```

SELECT s.season\_year , count(m.match\_id) as Total\_Match\_Played From Match m JOIN Season s ON s.season\_id=m.season\_id GROUP by s.season\_year;

Season_Year	Total_Match_Played
2008	58
2009	57
2010	60
2011	73
2012	74
2013	76
2014	60
2015	59
2016	60

Seg4\_Q2) Analyse the distribution of match outcomes for each team (wins, ties, no results) in the dataset.



Seg4\_Q3) Calculate the average winning margin (runs or wickets) for all matches.



#### Seg4\_Q4) Identify the top three venues with the highest average runs scored per match.

```
2 SELECT v.venue_name, MAX(avg_runs_scored) AS highest_average_runs
       SELECT m.Venue_Id, AVG(bs.runs_scored) AS avg_runs_scored
       FROM Batsman Scored bs
       JOIN MATCH m ON bs.Match_Id = m.match_id
       GROUP BY m.Venue_Id, bs.match_id
 8 ) AS venue_avg_runs
 9 JOIN Venue v ON venue_avg_runs.Venue_Id = v.Venue_Id
10 GROUP BY v.venue_name LIMIT 3;
! Venue Name
                                                  highest_average_runs
Barabati Stadium
                                                  1.6833333333333333
Brabourne Stadium
                                                  1.6625
Buffalo Park
                                                  1.2488038277511961
```

```
SELECT v.venue_name, MAX(avg_runs_scored) AS highest_average_runs
FROM (
    SELECT m.Venue_Id, AVG(bs.runs_scored) AS avg_runs_scored
    FROM Batsman_Scored bs
    JOIN Match m ON bs.Match_Id = m.match_id
    GROUP BY m.Venue_Id, bs.match_id
) AS venue_avg_runs
JOIN Venue v ON venue_avg_runs.Venue_Id = v.Venue_Id
GROUP BY v.venue_name limit 3;
```

Venue_Name	highest_average_runs		
Barabati Stadium	1.683333333		
Brabourne Stadium	1.6625		
Buffalo Park	1.248803828		

Seg4\_Q5) Determine the team that has won the most matches by a narrow margin (less than 10 runs or 2 wickets).

```
Determine the team that has won the most matches by a narrow margin
      m.match_winner AS team_id,
      t.team_name,
      COUNT(CASE WHEN m.win_margin < 10 AND m.win_type = 1 THEN 1
                WHEN m.win_margin < 2 AND m.win_type = 2 THEN 1 ELSE 0 END) AS narrow_margin_wins
 9 FROM MATCH m
10 JOIN Team t ON m.match_winner = t.team_id
11 GROUP BY m.match_winner, t.team_name
12 ORDER BY narrow_margin_wins DESC;
! team_id
                                Team_Name
                                                                narrow_margin_wins
                                Mumbai Indians
                                Chennai Super Kings
                                Royal Challengers Bangalore
                                Kolkata Knight Riders
                                Kings XI Punjab
```

```
m.match_winner AS team_id,
t.team_name,
count(CASE WHEN m.win_margin < 10 AND m.win_type = 1 THEN 1
WHEN m.win_margin < 2 AND m.win_type = 2 THEN 1 ELSE 0 END) AS
narrow_margin_wins
FROM Match m
JOIN Team t ON m.match_winner = t.team_id
GROUP BY m.match_winner, t.team_name
ORDER BY narrow_margin_wins DESC;
```

team_id	Team_Name	narrow_margin_wins	
7	Mumbai Indians	80	
3	Chennai Super Kings	79	
2	Royal Challengers Bangalore	70	
1	Kolkata Knight Riders	68	
4	Kings XI Punjab	63	
5	Rajasthan Royals	63	
6	Delhi Daredevils	56	
11	Sunrisers Hyderabad	34	
8	Deccan Chargers	29	
10	Pune Warriors	12	
13	Gujarat Lions	9	
9 Kochi Tuskers Kerala		6	
9	Kochi Tuskers Kerala	6	

## **Segment 5: Player Performance Comparison**

Seg5\_Q1) Compare the batting performance of players in home matches versus away matches and identify any significant differences.

Seg5_Q2) Analyse the bowling performance of players against left-handed batsmen versus right-handed batsmen and identify any performance variations.	

Seg5_Q3) Identify the players who have shown consistent improvement in their performance metrics over multiple IPL seasons.

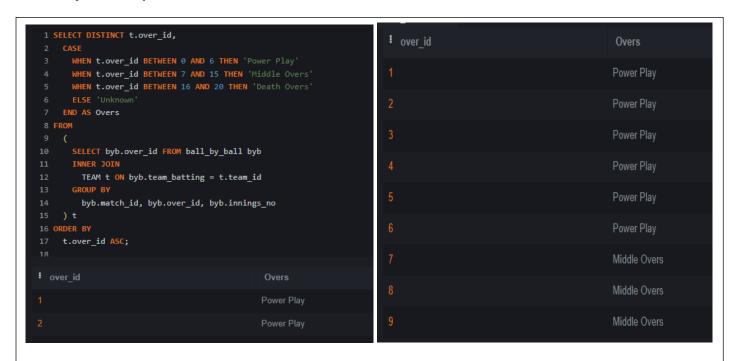
#### **Segment 6: Team Dynamics and Strategy**

## Seg6\_Q1) Analyse the relationship between a team's batting order and their overall run rate in matches

```
SELECT T.Team_Name, B.Striker_Batting_Position, AVG(Total_Runs/Over)*100 AS Run_Rate
FROM (
SELECT M.Match_Id,BB.Team_Batting, BB.Striker_Batting_Position,
SUM(BS.Runs_Scored) AS Total_Runs, count(BB.over_id) as Over
FROM Ball_by_Ball BB
JOIN Batsman_Scored BS ON BS.Match_Id = M.Match_Id
JOIN Match M ON BB.Match_Id = M.Match_Id
GROUP BY M.Match_Id, BB.Striker_Batting_Position
) AS B
JOIN Team T ON T.Team_Id = B.Team_Batting
GROUP BY T.Team_Name, B.Striker_Batting_Position
```

Seg6_Q2) Determine the effectiveness of teams in successfully chasing targets in different match scenarios (e.g., high target, low target, tight finish).

Seg6\_Q3) Identify the teams that have shown the most effective use of power play overs and analyse its impact on their match results.



```
SELECT DISTINCT t.over_id,
CASE
 WHEN t.over_id BETWEEN 0 AND 6 THEN 'Power Play'
 WHEN t.over_id BETWEEN 7 AND 15 THEN 'Middle Overs'
 WHEN t.over_id BETWEEN 16 AND 20 THEN 'Death Overs'
 ELSE 'Unknown'
END AS Overs
FROM
 SELECT byb.over_id FROM ball_by_ball byb
 INNER JOIN
  TEAM t ON byb.team_batting = t.team_id
 GROUP BY
  byb.match_id, byb.over_id, byb.innings_no
) t
ORDER BY
t.over_id ASC;
```

Seg6\_Q4) Analyse the distribution of match outcomes (wins, losses, ties) based on the team batting first or second. Identify any patterns or trends that could provide insights into successful match strategies for teams.

Seg6\_Q5) Which IPL season had the highest overall run rate? Analyze the factors contributing to the high-scoring matches and the impact on viewership and team strategies

```
1 SELECT
2 S.season_year,
3 SUM(BS.Runs_Scored) / (COUNT(BB.over_id) / 6) AS Run_Rate
4 FROM
5 Batsman_Scored BS
6 JOIN MATCH M ON BS.Match_Id = M.Match_Id
7 JOIN Ball_by_Ball BB ON BS.Match_Id = BB.Match_Id
8 JOIN Season S ON S.season_id = M.season_id
9 GROUP BY S.Season_Year
10 ORDER BY Run_Rate DESC;
11

i Season_Year Run_Rate

2016 7

2015 7

2014 7

2017 7

2018 7
```

Season_Year	Run_Rate	
2016	7	
2015	7	
2014	7	
2013	7	
2012	7	
2011	7	
2010	7	
2008	7	
2009	6	

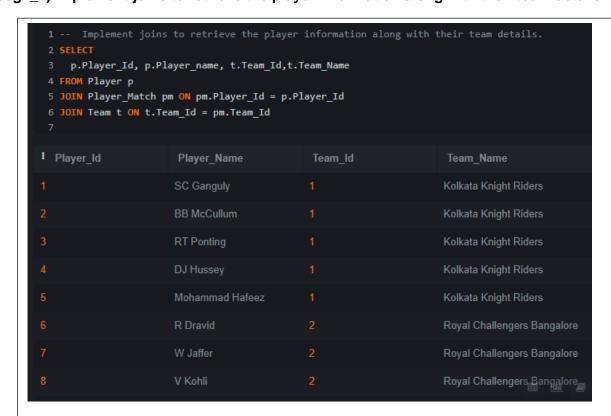
```
SELECT
S.season_year,
SUM(BS.Runs_Scored) / (COUNT(BB.over_id) / 6) AS Run_Rate
FROM
Batsman_Scored BS
JOIN Match M ON BS.Match_Id = M.Match_Id
JOIN Ball_by_Ball BB ON BS.Match_Id = BB.Match_Id
JOIN Season S ON S.season_id = M.season_id
GROUP BY S.Season_Year
order by Run_Rate DESC;
```

#### **Segment 7: SQL Concepts**

Seg7\_Q1) Use subqueries to find the players who have scored more than 500 runs in a single season.

```
1 SELECT DISTINCT (p.player_name),sum(bs.Runs_Scored) AS Run_scored
 2 FROM Player p
 3 JOIN Player_Match pm ON p.player_id = pm.player_id
 4 JOIN MATCH m ON pm.match_id = m.Match_Id
 5 JOIN Season s ON m.season_id = s.season_id
 6 JOIN Batsman_Scored bs ON pm.match_id = bs.match_id
 7 WHERE s.season_year = (season_year)
 8 GROUP BY p.player_name, p.player_id
 9 HAVING SUM(bs.runs_scored) > 500;
! Player_Name
                                                    Run_scored
A Ashish Reddy
                                                    8752
                                                    3409
                                                    1689
A Chopra
                                                    900
A Flintoff
A Kumble
A Mishra
                                                    31759
SELECT DISTINCT (p.player_name),sum(bs.Runs_Scored) AS Run_scored
FROM Player p
JOIN Player_Match pm ON p.player_id = pm.player_id
JOIN Match m ON pm.match_id = m.Match_Id
JOIN Season s ON m.season_id = s.season_id
JOIN Batsman_Scored bs ON pm.match_id = bs.match_id
WHERE s.season_year = (season_year)
GROUP BY p.player_name, p.player_id
HAVING SUM(bs.runs_scored) > 500;
```

Seg7\_2) Implement joins to retrieve the player information along with their team details.



```
SELECT
p.Player_Id, p.Player_name, t.Team_Id,t.Team_Name
FROM Player p
JOIN Player_Match pm ON pm.Player_Id = p.Player_Id
JOIN Team t ON t.Team_Id = pm.Team_Id
```

Seg7\_Q3) Utilise aggregate functions to calculate the average strike rate for each team.

```
-- - Utilise aggregate functions to calculate the average strike rate for each team.
      t.Team_Id,
      t.Team_Name,
      SUM(bs.Runs_Scored) AS total_runs,
      COUNT(bs.Ball_Id) AS total_balls_faced,
      (SUM(bs.Runs_Scored) * 100.0 / COUNT(bs.Ball_Id)) AS average_strike_rate
8 FROM
      Batsman_Scored bs
      MATCH m ON m.Match_Id = bs.Match_Id
I Team_ld Team_Name
                                             total_balls_faced
                               total_runs
                                                                 average_strike_rate
            Kolkata Knight Ri...
                                                                 121.58323400874056
            Royal Challenger...
                               38610
                                                                 126.35816206309727
            Chennai Super Ki...
            Kings XI Punjab
                                                                 128.09893307468477
                                                                 124.48905350882995
            Delhi Daredevils
                               37643
```

```
SELECT t.Team_Id, t.Team_Name,
SUM(bs.Runs_Scored) AS total_runs,
COUNT(bs.Ball_Id) AS total_balls_faced,
(SUM(bs.Runs_Scored) * 100.0 / COUNT(bs.Ball_Id)) AS average_strike_rate
FROM Batsman_Scored bs
JOIN
Match m ON m.Match_Id = bs.Match_Id
JOIN
Team t ON t.Team_Id = m.Team_1 OR t.Team_Id = m.Team_2
GROUP BY t.Team_Id, t.Team_Name;
```

Team_Id	Team_Name	total_runs	total_balls_faced	average_strike_rate
1	Kolkata Knight Riders	36723	30204	121.583234
2	Royal Challengers Bangalore	40854	31678	128.9664752
3	Chennai Super Kings	38610	30556	126.3581621
4	Kings XI Punjab	39621	30930	128.0989331
5	Rajasthan Royals	33568	27313	122.9011826
6	Delhi Daredevils	37643	30238	124.4890535
7	Mumbai Indians	40393	32727	123.4240841
8	Deccan Chargers	21832	17656	123.6520163
9	Kochi Tuskers Kerala	3630	3083	117.7424586
10	Pune Warriors	12556	10755	116.7456997
11	Sunrisers Hyderabad	17702	14282	123.946226
12	Rising Pune Supergiants	3967	3087	128.5066408
13	Gujarat Lions	4823	3685	130.8819539

## Seg7\_Q4) Apply window functions to rank the teams based on their total runs scored in a season.

```
1 -- Apply window functions to rank the teams based on their total runs scored in a season
 2 SELECT Team Id, Team Name, total runs,
         RANK() OVER (ORDER BY total_runs DESC) AS rank
 5 SELECT t.Team_Id, t.Team_Name, SUM(bs.Runs_Scored) AS total_runs
    FROM Team t
     JOIN MATCH m ON t.Team_Id = m.Team_1 OR t.Team_Id = m.Team_2
     JOIN Batsman_Scored bs ON m.Match_Id = bs.Match_Id
     GROUP BY t.Team_Id, t.Team_name
10 ) AS total_runs;
! Team Id
                      Team Name
                                             total runs
                      Royal Challengers Ban...
                      Kings XI Punjab
                      Chennai Super Kings
                                             37643
                      Delhi Daredevils
                      Kolkata Knight Riders 36723
```

```
SELECT Team_Id, Team_Name, total_runs,

RANK() OVER (ORDER BY total_runs DESC) AS rank

FROM (

SELECT t.Team_Id, t.Team_Name, SUM(bs.Runs_Scored) AS total_runs

FROM Team t

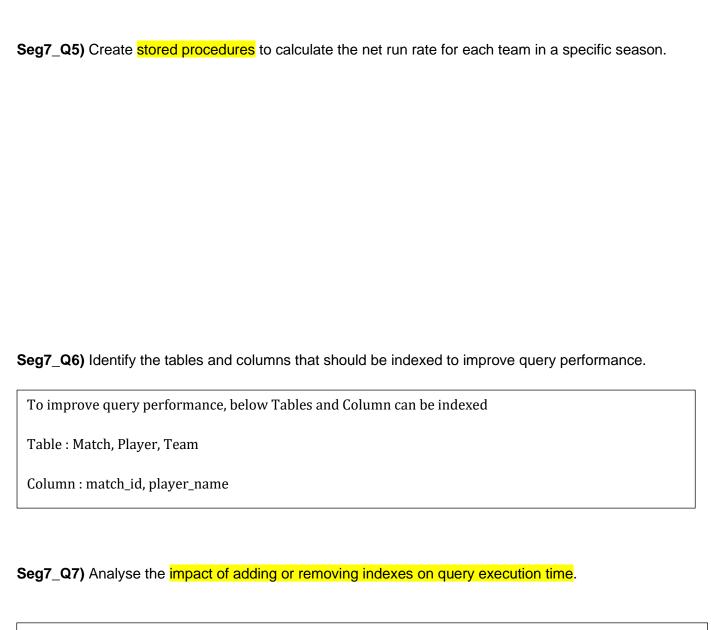
JOIN Match m ON t.Team_Id = m.Team_1 OR t.Team_Id = m.Team_2

JOIN Batsman_Scored bs ON m.Match_Id = bs.Match_Id

GROUP BY t.Team_Id, t.Team_name

) AS total_runs;
```

Team_Id	Team_Name	total_runs	rank
2	Royal Challengers Bangalore	40854	1
7	Mumbai Indians 40393		2
4	Kings XI Punjab 39621		3
3	Chennai Super Kings	38610	4
6	Delhi Daredevils	37643	5
1	Kolkata Knight Riders	36723	6
5	Rajasthan Royals	33568	7
8	Deccan Chargers	21832	8
11	Sunrisers Hyderabad	17702	9
10	Pune Warriors 12556		10
13	Gujarat Lions 4823		11
12	Rising Pune Supergiants	3967	12
9	Kochi Tuskers Kerala	3630	13



Indexing makes columns faster to query by creating pointers to where data is stored within a database. Imagine you want to find a piece of information that is within a large database. To get this information out of the database the computer will look through every row until it finds it. If the data you are looking for is towards the very end, this query would take a long time to run.

An index is a structure that holds the field the index is sorting and a pointer from each record to their corresponding record in the original table where the data is actually stored. Indexes are used in things like a contact list where the data may be physically stored in the order you add people's contact information but it is easier to find people when listed out in alphabetical order.

Indexes are meant to speed up the performance of a database, so use indexing whenever it significantly improves the performance of your database. As your database becomes larger and larger, the more likely you are to see benefits from indexing.

# Seg7\_Q8) Evaluate the performance improvement of queries after using common table expressions (CTEs).

A CTE (Common Table Expression) is a one-time result set that only exists for the duration of the query. It allows us to refer to data within a single SELECT, INSERT, UPDATE, DELETE, CREATE VIEW, or MERGE statement's execution scope.

A common table [deprecated]CTE is a powerful SQL construct that helps simplify queries. CTEs act as virtual tables (with records and columns) that are created during query execution, used by the query, and deleted after the query executes.

We can define CTEs by adding a WITH clause directly before the SELECT, INSERT, UPDATE, DELETE, or MERGE statement. The WITH clause can include one or more CTEs separated by commas.

# **Seg7\_Q9)** identify any potential bottlenecks in the database schema and suggest optimizations to mitigate them

- 1) In data base Win\_by instead win\_id , win\_type should be primary Key. No relevant of win\_type Id.
- 2) In the "Wicket Taken" table column, Player\_out contains information about the batsman out and not the wicket-taking baller. so, to analysis Bowler performance is difficult.

#### Evaluation pointers:

- The tasks are correctly identified and executed.
- The solution output matches the expected output.
- The query is optimised and syntactically correct.
- Proper aliases are used
- If required any, appropriate comments are written.
- The code is written concisely with appropriate indentations.

Project is Submitted by : AMIT DEVLEKAR