

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 |

+-----+
| name |
+-----+
| Ball_by_Ball |
| Batsman_Scored |
| Batting_Style |
| Bowling_Style |
| City |
| Country |
```

SELECT name FROM sqlite_schema
WHERE type='table' ORDER BY name;

Table Name

name	Ball_by_Ball	Batsman_Scored	Batting_Style	Bowling_Style	City	Country	Extra_Runs	Extra_Type	Match	Out_Type	Outcome	Player_Match	Rolee	Season	Team	Toss_Decision	Umpire	Venue	Wicket_Taken	Win_By	sqlite_sequence	sysdiagrams
------	--------------	----------------	---------------	---------------	------	---------	------------	------------	-------	----------	---------	--------------	-------	--------	------	---------------	--------	-------	--------------	--------	-----------------	-------------

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(City);
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(Outcome);
PRAGMA table_info(Player);

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);						PRAGMA table_info(Batsman_Scored);					
cid	name	type	notnull	dflt_value	pk	cid	name	type	notnull	dflt_value	pk
0	Match_Id	INTEGER	1	null	1	0	Match_Id	INTEGER	1	null	1
1	Over_Id	INTEGER	1	null	2	1	Over_Id	INTEGER	1	null	2
2	Ball_Id	INTEGER	1	null	3	2	Ball_Id	INTEGER	1	null	3
3	Innings_No	INTEGER	1	null	4	3	Runs_Scored	INTEGER	1	null	0
4	Team_Batting	INTEGER	1	null	0	4	Innings_No	INTEGER	1	null	4
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(Batting_Style);						PRAGMA table_info(Bowling_Style);					
cid	name	type	notnull	dflt_value	pk	cid	name	type	notnull	dflt_value	pk
0	Batting_Id	INTEGER	1	null	1	0	Bowling_Id	INTEGER	1	null	1
1	Batting_hand	varchar(20)	1	null	0	1	Bowling_skill	varchar(20)	1	null	0

PRAGMA table_info(City);					
cid	name	type	notnull	dfit_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	dfit_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	dfit_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

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

PRAGMA table_info(Out_Type);					
cid	name	type	notnull	dfit_value	pk
0	Out_Id	INTEGER	1	null	1
1	Out_Name	varchar(250)	1	null	0
PRAGMA table_info(Player);					
cid	name	type	notnull	dfit_value	pk
0	Player_Id	INTEGER	1	null	1
1	Player_Name	varchar(400)	1	null	0
2	DOB	datetime	0	null	0
3	Batting_hand	INTEGER	1	null	0
4	Bowling_skill	INTEGER	0	null	0
5	Country_Name	INTEGER	1	null	0
PRAGMA table_info(Player_Match);					
cid	name	type	notnull	dfit_value	pk
0	Match_Id	INTEGER	1	null	1
1	Player_Id	INTEGER	1	null	2
2	Role_Id	INTEGER	1	null	0
3	Team_Id	INTEGER	0	null	0
PRAGMA table_info(Rolee);					
cid	name	type	notnull	dfit_value	pk
0	Role_Id	INTEGER	1	null	1
1	Role_Desc	varchar(150)	1	null	0
PRAGMA table_info(Season);					
cid	name	type	notnull	dfit_value	pk
0	Season_Id	INTEGER	1	null	1
1	Man_of_the_Series	INTEGER	1	null	0
2	Orange_Cap	INTEGER	1	null	0
3	Purple_Cap	INTEGER	1	null	0
4	Season_Year	INTEGER	0	null	0

PRAGMA table_info(sqlite_sequence);					
cid	name	type	notnull	dfit_value	pk
0	name		0	null	0
1	seq		0	null	0
PRAGMA table_info(sysdiagrams);					
cid	name	type	notnull	dfit_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
PRAGMA table_info(Team);					
cid	name	type	notnull	dfit_value	pk
0	Team_Id	INTEGER	1	null	1
1	Team_Name	varchar(450)	1	null	0
PRAGMA table_info(Toss_Decision);					
cid	name	type	notnull	dfit_value	pk
0	Toss_Id	INTEGER	1	null	1
1	Toss_Name	varchar(50)	1	null	0
PRAGMA table_info(Umpire);					
cid	name	type	notnull	dfit_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	dfit_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	dfit_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	dfit_value	pk
0	Win_Id	INTEGER	1	null	1
1	Win_Type	varchar(200)	1	null	0

```

1 -- Segment 1: Database - Tables, Columns, Relationships
2 -- - 1) Identify the tables in the dataset and their respective columns.
3
4 PRAGMA table_info(Win_By);
5

```

cid	name	type	notnull	dfit_value	pk
0	Win_Id	INTEGER	1	NULL	1
1	Win_Type	varchar(200)	1	NULL	0

Seq1_Q2) Determine the number of rows in each table within the schema.

SELECT count(*) FROM Ball_by_Ball;

```
1
2 SELECT COUNT(*) FROM Batsman_Scored;
count(*)
133097
```

```
1
2 SELECT COUNT(*) FROM Ball_by_Ball;
count(*)
136590
```

```
1
2 SELECT COUNT(*) FROM Batting_Style;
count(*)
2
```

```
1
2 SELECT COUNT(*) FROM Bowling_Style;
count(*)
14
```

```
1
2 SELECT COUNT(*) FROM City;
count(*)
29
```

```
1
2 SELECT COUNT(*) FROM Country;
count(*)
12
```

```
1
2 SELECT COUNT(*) FROM Extra_Runs;
count(*)
7469
```

```
1
2 SELECT COUNT(*) FROM Extra_Type;
count(*)
5
```

```
1
2 SELECT COUNT(*) FROM MATCH;
count(*)
577
```

```
1
2 SELECT COUNT(*) FROM Outcome;
count(*)
3
```

```
1
2 SELECT COUNT(*) FROM Out_Type;
count(*)
9
```

```
1
2 SELECT COUNT(*) FROM Player;
count(*)
469
```

```
1
2 SELECT COUNT(*) FROM Rolee;
count(*)
4
```

```
1
2 SELECT COUNT(*) FROM Season;
count(*)
9
```

```
1
2 SELECT COUNT(*) FROM sqlite_sequence;
count(*)
1
```

```
1
2 SELECT COUNT(*) FROM sysdiagrams;
count(*)
1
```

```
1
2 SELECT COUNT(*) FROM Team;
count(*)
13
```

```
1
2 SELECT COUNT(*) FROM Toss_Decision;
count(*)
2
```

```
1
2 SELECT COUNT(*) FROM Umpire;
count(*)
52
```

```
1
2 SELECT COUNT(*) FROM Venue;
count(*)
35
```

```
1
2 SELECT COUNT(*) FROM Wicket_Taken;
count(*)
6727
```

```
1
2 SELECT COUNT(*) FROM Win_By;
count(*)
4
```

```
1
2 SELECT COUNT(*) FROM Player_Match;
count(*)
12694
```

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.

database.sqlite0.1.3 beta

Table

Ball_by_Ball<Batsman_Scored<Batting_Style<Bowling_Style<City<Country<Extra_Runs<Extra_Type<Match<Outcome<Out_Type<Player<Player_Match<Rolee<Season<sqlite_sequence<sysdiagrams<

database.sqliteSchema Ball_by_Ball

1 -- - Segment 2: Team AnalysisMatch2 -- 1) Determine the number of matches played by each team in the dataset.34 SELECT Team, sum(Matches) AS Matches FROM (5 SELECT team_1 AS Team, COUNT(*) AS Matches FROM MATCH6 GROUP BY Team_17 UNION8 SELECT team_2 AS Team, COUNT(*) AS Matches FROM MATCH9 GROUP BY Team_210)11 GROUP BY Team;12

TeamMatches

1132

2139

3131

4134

5118

Team	Matches
1	132
2	139
3	131
4	134
5	118
6	133
7	140
8	75
9	7
10	23
11	62
12	7
13	16

Select Team, sum(Matches) as Matches From (
Select team_1 as Team, COUNT(*) as Matches FROM
Match
GROUP BY team_1
UNION
Select team_2 as Team, COUNT(*) as Matches FROM
Match
GROUP BY team_2
,

Seg2_Q2) Analyse the win-loss ratio for each team in IPL history

```
1 WITH TeamMatches AS (  
2     SELECT M.team_1 AS Team,  
3           S.season_year AS Season_Year,  
4           COUNT(*) AS Total_Matches_Played,  
5           NULL AS Total_Matches_Won  
6     FROM MATCH M  
7     JOIN Season S ON M.season_id = S.season_id  
8     GROUP BY M.team_1, S.season_year  
9  
10    UNION ALL  
11  
12    SELECT M.team_2 AS Team,  
13           S.season_year AS Season_Year,
```

!	Team	Season_Year	Total_Match...	Total_Match...	Win_Ratio	Loss_Ratio
1		2008	13	6	0.461538461...	0.5384615384615...
1		2009	13	3	0.230769230...	0.7692307692307...
1		2010	14	7	0.5	0.5
1		2011	15	8	0.533333333...	0.4666666666666...
1		2012	17	12	0.705882352...	0.29411764705882...

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.

database.sqlite

```
1 SELECT MATCH.team_1, avg(Batsman_Scored.Runs_Scored) AS Avg_Score
2 FROM Batsman_Scored
3 INNER JOIN MATCH ON Batsman_Scored.Match_Id = MATCH.Match_Id
4 GROUP BY team_1;
5
```

Team_1	Avg_Score
1	1.2181305114638448
2	1.288319509146688
3	1.2547545837603917
4	1.2739853009335893
5	1.2300899562598002
6	1.2433457631477434
7	1.263282501218578

```
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.

database.sqlite

```
1 SELECT T.team_name, S.Season_Year, COUNT(B.runs_scored) AS Highest_Sixes
2 FROM MATCH AS M
3 INNER JOIN Batsman_Scored AS B ON B.Match_Id = M.Match_Id
4 INNER JOIN Season AS S ON S.Season_Id = M.Season_Id
5 INNER JOIN Team AS T ON T.Team_Id=M.Team_1
6 WHERE B.runs_scored = 6
7 GROUP BY T.team_name, S.Season_Year;
```

Team_Name	Season_Year	Highest_Sixes
Chennai Super Kings	2008	101
Chennai Super Kings	2009	118
Chennai Super Kings	2010	106
Chennai Super Kings	2011	76
Chennai Super Kings	2012	81
Chennai Super Kings	2013	104
Chennai Super Kings	2014	123
Chennai Super Kings	2015	95

```
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.

```
1
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
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.

```
1 SELECT p.player_name, sum(bs.runs_scored) AS Total_Runs
2 FROM Player P
3 JOIN Player_Match pm ON p.player_id= pm.player_id
4 JOIN Batsman_Scored bs ON pm.match_id=bs.Match_Id
5 GROUP BY p.player_name
6 ORDER BY Total_Runs DESC
7 LIMIT 5;
```

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

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
```

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
```

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
-
```

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.

```

1 -- - Analyse the distribution of match outcomes for each team (wins, ties, no results) in the dataset.
2 SELECT
3     SUM(CASE WHEN win_type = 1 THEN 1 ELSE 0 END) AS win_by_Run,
4     SUM(CASE WHEN win_type = 2 THEN 1 ELSE 0 END) AS win_by_wicket,
5     SUM(CASE WHEN win_type = 4 THEN 1 ELSE 0 END) AS Tie,
6     SUM(CASE WHEN win_type = 3 THEN 1 ELSE 0 END) AS No_Result,
7
8     T.team_name
9
10 FROM MATCH m
11 JOIN Team T ON T.team_id = m.team_1 OR T.team_id = m.team_2
12
13 GROUP BY T.team_name;

```

win_by_Run	win_by_wicket	Tie	No_Result	Team_Name
66	64	1	0	Chennai Super Kings
39	36	0	0	Deccan Chargers
58	72	1	2	Delhi Daredevils
3	13	0	0	Gujarat Lions

```

SELECT
    SUM(CASE WHEN win_type = 1 THEN 1 ELSE 0 END) AS win_by_Run,
    SUM(CASE WHEN win_type = 2 THEN 1 ELSE 0 END) AS win_by_wicket,
    SUM(CASE WHEN win_type = 4 THEN 1 ELSE 0 END) AS Tie,
    SUM(CASE WHEN win_type = 3 THEN 1 ELSE 0 END) AS No_Result,

    T.team_name

FROM Match m
JOIN Team T ON T.team_id = m.team_1 OR T.team_id = m.team_2

GROUP BY T.team_name;

```

Seg4_Q3) Calculate the average winning margin (runs or wickets) for all matches.

```

1 SELECT avg(win_margin) AS Winning_Margin
2 FROM MATCH;
3
4
5

```

! Winning_Margin

17.350352112676056

```

SELECT avg(win_margin) as
Winning_Margin
From Match;

```


Seg4_Q4) Identify the top three venues with the highest average runs scored per match.

```
1
2 SELECT v.venue_name, MAX(avg_runs_scored) AS highest_average_runs
3 FROM (
4     SELECT m.Venue_Id, AVG(bs.runs_scored) AS avg_runs_scored
5     FROM Batsman_Scored bs
6     JOIN MATCH m ON bs.Match_Id = m.match_id
7     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).

```

1 -- Determine the team that has won the most matches by a narrow margin
2 -- (less than 10 runs or 2 wickets).
3
4 SELECT
5     m.match_winner AS team_id,
6     t.team_name,
7     COUNT(CASE WHEN m.win_margin < 10 AND m.win_type = 1 THEN 1
8             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
7		Mumbai Indians	80
3		Chennai Super Kings	79
2		Royal Challengers Bangalore	70
1		Kolkata Knight Riders	68
4		Kings XI Punjab	63

```

SELECT
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

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

```
1 SELECT T.Team_Name, B.Striker_Batting_Position, AVG(Total_Runs/Over)*100 AS Run_Rate
2 FROM (
3     SELECT M.Match_Id, BB.Team_Batting, BB.Striker_Batting_Position,
4     SUM(BS.Runs_Scored) AS Total_Runs, COUNT(BB.over_id) AS Over
5     FROM Ball_by_Ball BB
6     JOIN Batsman_Scored BS ON BS.Match_Id = M.Match_Id
7     JOIN MATCH M ON BB.Match_Id = M.Match_Id
8     GROUP BY M.Match_Id, BB.Striker_Batting_Position
9 ) AS B
10 JOIN Team T ON T.Team_Id = B.Team_Batting
11 GROUP BY T.Team_Name, B.Striker_Batting_Position
```

! Team_Name	Striker_Batting_Position	Run_Rate
Chennai Super Kings	1	92.20779220779221
Chennai Super Kings	2	89.23076923076924
Chennai Super Kings	3	90.625
Chennai Super Kings	4	88.70967741935483
Chennai Super Kings	5	90.56603773584906
Chennai Super Kings	6	90.74074074074075

```
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.

```
1 SELECT DISTINCT t.over_id,
2 CASE
3   WHEN t.over_id BETWEEN 0 AND 6 THEN 'Power Play'
4   WHEN t.over_id BETWEEN 7 AND 15 THEN 'Middle Overs'
5   WHEN t.over_id BETWEEN 16 AND 20 THEN 'Death Overs'
6   ELSE 'Unknown'
7 END AS Overs
8 FROM
9 (
10  SELECT byb.over_id FROM ball_by_ball byb
11  INNER JOIN
12    TEAM t ON byb.team_batting = t.team_id
13  GROUP BY
14    byb.match_id, byb.over_id, byb.innings_no
15 ) t
16 ORDER BY
17   t.over_id ASC;
```

over_id	Overs
1	Power Play
2	Power Play

over_id	Overs
1	Power Play
2	Power Play
3	Power Play
4	Power Play
5	Power Play
6	Power Play
7	Middle Overs
8	Middle Overs
9	Middle Overs

```
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
```

Season_Year	Run_Rate
2016	7
2015	7
2014	7
2013	7
2012	7
2011	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;
10
```

Player_Name	Run_scored
A Ashish Reddy	8752
A Chandila	3409
A Chopra	1689
A Flintoff	900
A Kumble	11627
A Mishra	31759
A Mithun	1617

```
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.

```
1 -- Implement joins to retrieve the player information along with their team details.
2 SELECT
3   p.Player_Id, p.Player_name, t.Team_Id,t.Team_Name
4 FROM Player p
5 JOIN Player_Match pm ON pm.Player_Id = p.Player_Id
6 JOIN Team t ON t.Team_Id = pm.Team_Id
7
```

!	Player_Id	Player_Name	Team_Id	Team_Name
1		SC Ganguly	1	Kolkata Knight Riders
2		BB McCullum	1	Kolkata Knight Riders
3		RT Ponting	1	Kolkata Knight Riders
4		DJ Hussey	1	Kolkata Knight Riders
5		Mohammad Hafeez	1	Kolkata Knight Riders
6		R Dravid	2	Royal Challengers Bangalore
7		W Jaffer	2	Royal Challengers Bangalore
8		V Kohli	2	Royal Challengers Bangalore

```
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.

```

1 -- - Utilise aggregate functions to calculate the average strike rate for each team.
2 SELECT
3     t.Team_Id,
4     t.Team_Name,
5     SUM(bs.Runs_Scored) AS total_runs,
6     COUNT(bs.Ball_Id) AS total_balls_faced,
7     (SUM(bs.Runs_Scored) * 100.0 / COUNT(bs.Ball_Id)) AS average_strike_rate
8 FROM
9     Batsman_Scored bs
10 JOIN
11     MATCH m ON m.Match_Id = bs.Match_Id

```

#	Team_Id	Team_Name	total_runs	total_balls_faced	average_strike_rate
1		Kolkata Knight Ri...	36723	30204	121.58323400874056
2		Royal Challenger...	40854	31678	128.96647515625986
3		Chennai Super Ki...	38610	30556	126.35816206309727
4		Kings XI Punjab	39621	30930	128.09893307468477
5		Rajasthan Royals	33568	27313	122.90118258704646
6		Delhi Daredevils	37643	30238	124.48905350882995

```

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,
3        RANK() OVER (ORDER BY total_runs DESC) AS rank
4 FROM (
5     SELECT t.Team_Id, t.Team_Name, SUM(bs.Runs_Scored) AS total_runs
6     FROM Team t
7     JOIN MATCH m ON t.Team_Id = m.Team_1 OR t.Team_Id = m.Team_2
8     JOIN Batsman_Scored bs ON m.Match_Id = bs.Match_Id
9     GROUP BY t.Team_Id, t.Team_name
10 ) AS total_runs;

```

Team_Id	Team_Name	total_runs	rank
2	Royal Challengers Ban...	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

```

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

Seg7_Q5) Create **stored procedures** to calculate the net run rate for each team in a specific season.

Seg7_Q6) Identify the tables and columns that should be indexed to improve query performance.

To improve query performance, below Tables and Column can be indexed

Table : Match, Player, Team

Column : match_id, player_name

Seg7_Q7) Analyse the **impact of adding or removing indexes on query execution time.**

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

