CS2005 DataBases

Assignment #2

Tauha Imran | 22i-1239

i221239@nu.edu.pk

Saffi Muhammad Hashir | 22i- 22i-1293

i221293@nu.edu.pk

Introduction

In the realm of database management, assignments serve as practical laboratories for students to apply theoretical concepts in real-world scenarios. This case study delves into the intricacies of a database assignment centred around the Pakistan Super League (PSL) dataset. The assignment challenges students to design and implement a comprehensive database schema tailored to the requirements of the PSL, encompassing player statistics, match details, team dynamics, and tournament outcomes. Through the utilisation of SQL queries incorporating concepts such as correlated nested queries, UNION, GROUP BY, LIKE comparisons, and the HAVING clause, students are tasked with extracting meaningful insights from the dataset, thereby honing their skills in database querying and analysis..

Q#1)

Create all required tables (other than the 2 already shared) in SQL and then insert at least 20 dummy data into each table.

MySQL Queries

```
/*making database DBA2psl (PSL case study) */
drop database if exists DBA2psl;
create database DBA2psl;
```

```
use DBA2psl;


/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
1) Create all required tables (other than the 2 already shared) in SQL and then
insert at least 20 dummy data into each table.
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/



-- 1. TEAM
drop table if exists team
create table team_data (
   Name VARCHAR(255) primary key ,
   Home_Stadium VARCHAR(255),
   Number_of_wins INT default 0,
   Number_of_losses INT default 0
);
--loading data

 /*USE DBA2psl LOAD DATA  INFILE 'G:/Tauha -
FAST/t.smstr4/DB/22i1239_DB_A2/team_data.csv'
         INTO TABLE team
         FIELDS TERMINATED BY ','
         IGNORE 1 LINES;
*/

select * from team;


--2 player
```

```
drop table if exists player
create table player (
   player_ID INT PRIMARY KEY,
   Name VARCHAR(255),
   Role VARCHAR(255),
   Team_Name VARCHAR(255),
   FOREIGN KEY (Team_Name) REFERENCES team(Name)
);
select*from player



--3 stadium
drop table if exists stadium
create table stadium (
   Stadium_ID INT PRIMARY KEY,
   Name VARCHAR(255),
   Location VARCHAR(255),
   Capacity INT
);

INSERT INTO stadium (Stadium_ID, Name, Location, Capacity) VALUES
(1, 'Gaddafi Stadium', 'Lahore', 50000),
(2, 'National Stadium', 'Karachi', 60000),
(3, 'Ayub Stadium', 'Quetta', 70000),
(4, 'Dubai International Cricket Stadium', 'Dubai', 65000),
(5, 'Arbab Niaz Stadium', 'Peshawar', 51000),
(6, 'Rawalpindi Cricket Stadium', 'Rawalpindi', 59000),
(7, 'Multan Cricket Stadium', 'Multan', 68000),
(8, 'Iqbal Stadium', 'Faisalabad', 55000),
```

```
(9, 'Jinnah Stadium', 'Sialkot', 45000),
(10, 'Iqbal Cricket Stadium', 'Sheikhupura', 62000),
(11, 'National Stadium', 'Faisalabad', 57000),
(12, 'Jinnah Stadium', 'Gujranwala', 51000),
(13, 'Rawalpindi Cricket Stadium', 'Gujrat', 50000),
(14, 'Ayub Stadium', 'Sargodha', 70000),
(15, 'Multan Cricket Stadium', 'Bahawalpur', 65000),
(16, 'Gaddafi Stadium', 'Sialkot', 54000),
(17, 'Dubai International Cricket Stadium', 'Sharjah', 48000),
(18, 'Arbab Niaz Stadium', 'Larkana', 60000),
(19, 'National Stadium', 'Islamabad', 70000),
(20, 'Iqbal Stadium', 'Sukkur', 65000);

select*from stadium;


--4 matchx (match is a key word)
drop table if exists matchx
create table matchx (
        Match_ID INT PRIMARY KEY,
   Date DATE,
   Time TIME,
   Stadium_ID INT,
   Match_Type VARCHAR(255),
   Team1_ID NVARCHAR(50),
   Team2_ID NVARCHAR(50),
   FOREIGN KEY (Stadium_ID) REFERENCES stadium(Stadium_ID),
   FOREIGN KEY (Team1_ID) REFERENCES team(Name),
   FOREIGN KEY (Team2_ID) REFERENCES team(Name)
```

```
);

INSERT INTO matchx (Match_ID, Date, Time, Stadium_ID, Match_Type,
Team1_ID, Team2_ID) VALUES
(1, '2024-01-01', '12:00:00', 1, 'Qualifier', 'Karachi Kings', 'Lahore Qalandars'),
(2, '2024-01-02', '13:00:00', 2, 'Eliminator', 'Islamabad United', 'Peshawar Zalmi'),
(3, '2024-01-03', '14:00:00', 3, 'Final', 'Quetta Gladiators', 'Multan Sultans'),
(4, '2024-01-04', '15:00:00', 4, 'Qualifier', 'Lahore Qalandars', 'Karachi Kings'),
(5, '2024-01-05', '16:00:00', 5, 'Eliminator', 'Peshawar Zalmi', 'Islamabad United'),
(6, '2024-01-06', '17:00:00', 6, 'Final', 'Multan Sultans', 'Quetta Gladiators'),
(7, '2024-01-07', '18:00:00', 7, 'Qualifier', 'Karachi Kings', 'Islamabad United'),
(8, '2024-01-08', '19:00:00', 8, 'Eliminator', 'Lahore Qalandars', 'Peshawar Zalmi'),
(9, '2024-01-09', '20:00:00', 9, 'Final', 'Quetta Gladiators', 'Multan Sultans'),
(10, '2024-01-10', '21:00:00', 10, 'Qualifier', 'Islamabad United', 'Karachi Kings'),
(11, '2024-01-11', '22:00:00', 11, 'Eliminator', 'Peshawar Zalmi', 'Lahore Qalandars'),
(12, '2024-01-12', '23:00:00', 12, 'Final', 'Multan Sultans', 'Quetta Gladiators'),
(13, '2024-01-13', '12:00:00', 13, 'Qualifier', 'Karachi Kings', 'Quetta Gladiators'),
(14, '2024-01-14', '13:00:00', 14, 'Eliminator', 'Islamabad United', 'Lahore
Qalandars'),
(15, '2024-01-15', '14:00:00', 15, 'Final', 'Peshawar Zalmi', 'Multan Sultans'),
(16, '2024-01-16', '15:00:00', 16, 'Qualifier', 'Lahore Qalandars', 'Karachi Kings'),
(17, '2024-01-17', '16:00:00', 17, 'Eliminator', 'Peshawar Zalmi', 'Islamabad United'),
(18, '2024-01-18', '17:00:00', 18, 'Final', 'Quetta Gladiators', 'Multan Sultans'),
(19, '2024-01-19', '18:00:00', 19, 'Qualifier', 'Karachi Kings', 'Islamabad United'),
(20, '2024-01-20', '19:00:00', 20, 'Eliminator', 'Lahore Qalandars', 'Peshawar
Zalmi');

select*from matchx;
```

```
--5 performance
drop table if exists performance
create table performance (
  Performance_ID INT PRIMARY KEY,
    Match_ID INT,
    Player_ID tinyint,
    Runs_Scored INT,
    Wickets_Taken INT,
    Catches INT,
    FOREIGN KEY (Match_ID) REFERENCES matchx(Match_ID),
    FOREIGN KEY (Player_ID) REFERENCES PLAYER(Player_ID)
);

INSERT INTO performance (Performance_ID, Match_ID, Player_ID,
Runs_Scored, Wickets_Taken, Catches) VALUES
(1, 1, 1, 56, 0, 2),
(2, 1, 2, 23, 1, 0),
(3, 2, 3, 42, 0, 1),
(4, 2, 4, 18, 2, 0),
(5, 3, 5, 75, 0, 1),
(6, 3, 6, 32, 1, 0),
(7, 4, 7, 60, 0, 2),
(8, 4, 8, 27, 1, 0),
(9, 5, 9, 48, 0, 1),
(10, 5, 10, 22, 2, 0),
(11, 6, 11, 65, 0, 2),
(12, 6, 12, 28, 1, 0),
(13, 7, 13, 51, 0, 1),
(14, 7, 14, 25, 2, 0),
```

```
(15, 8, 15, 70, 0, 2),
(16, 8, 16, 35, 1, 0),
(17, 9, 17, 49, 0, 1),
(18, 9, 18, 20, 2, 0),
(19, 10, 19, 67, 0, 2),
(20, 10, 20, 31, 1, 0);

select*from performance;

--6 winner
drop table if exists winner
create table winner (
        Match_ID INT PRIMARY KEY,
   Winning_Team_Name NVARCHAR(50),
   FOREIGN KEY (Match_ID) REFERENCES matchx(Match_ID),
   FOREIGN KEY (Winning_Team_Name) REFERENCES team(Name)
);

INSERT INTO winner (Match_ID, Winning_Team_Name) VALUES
(1, 'Karachi Kings'),
(2, 'Islamabad United'),
(3, 'Quetta Gladiators'),
(4, 'Karachi Kings'),
(5, 'Islamabad United'),
(6, 'Multan Sultans'),
(7, 'Karachi Kings'),
(8, 'Peshawar Zalmi'),
(9, 'Quetta Gladiators'),
(10, 'Islamabad United'),
```

(11, 'Peshawar Zalmi'),

(12, 'Multan Sultans'),

(13, 'Karachi Kings'),

(14, 'Islamabad United'),

(15, 'Multan Sultans'),

(16, 'Karachi Kings'),

(17, 'Peshawar Zalmi'),

(18, 'Quetta Gladiators'),

(19, 'Karachi Kings'),

(20, 'Lahore Qalandars');

select*from winner;

Result

1. team_data (Team):

DESKTOP-N4RNGN5\...A2psl - dbo.team | Object Explorer Details

Column Name	Data Type	Allow Nulls
Name	nvarchar(50)	☐
Home_Stadium	nvarchar(50)	☐
Number_of_Wins	tinyint	☐
Number_of_Losses	tinyint	☐
		☐

Results | Messages

	Name	Home_Stadium	Number_of_Wins	Number_of_Losses
1	Islamabad United	Rawalpindi Cricket Stadium	0	0
2	Karachi Kings	National Stadium	0	0
3	Lahore Qalandars	Gaddafi Stadium	0	0
4	Multan Sultans	Multan Cricket Stadium	0	0
5	Peshawar Zalmi	Arbab Niaz Stadium	0	0
6	Quetta Gladiators	Bugti Stadium	0	0

2. player (Player):

DESKTOP-N4RNGN5\S...2psl - dbo.player | DESKTOP-N4RNGN5\...dbo

Column Name	Data Type	Allow Nulls
Player_ID	tinyint	☐
Name	nvarchar(50)	☐
Role	nvarchar(50)	☐
Team_Name	nvarchar(50)	☐
		☐

Results | Messages

	Player_ID	Name	Role	Team_Name
1	1	Babar Azam	Batsman	Karachi Kings
2	2	Shaheen Afridi	Bowler	Lahore Qalandars
3	3	Shadab Khan	All-Rounder	Islamabad United
4	4	Wahab Riaz	Bowler	Peshawar Zalmi
5	5	Sarfaraz Ahmed	Wicketkeeper	Quetta Gladiators
6	6	Mohammad Rizwan	Wicketkeeper	Multan Sultans
7	7	Imad Wasim	All-Rounder	Karachi Kings
8	8	Fakhar Zaman	Batsman	Lahore Qalandars
9	9	Asif Ali	Batsman	Islamabad United
10	10	Haris Rauf	Bowler	Peshawar Zalmi
11	11	Hassan Ali	Bowler	Quetta Gladiators
12	12	Faheem Ashraf	All-Rounder	Multan Sultans
13	13	Mohammad Hafeez	All-Rounder	Karachi Kings
14	14	Shoaib Malik	All-Rounder	Lahore Qalandars
15	15	Ahmed Shehzad	Batsman	Islamabad United
16	16	Umar Akmal	Batsman	Peshawar Zalmi
17	17	Shan Masood	Batsman	Quetta Gladiators
18	18	Kamran Akmal	Wicketkeeper	Multan Sultans
19	19	Yasir Shah	Bowler	Karachi Kings
20	20	Junaid Khan	Bowler	Lahore Qalandars

3. stadium (Stadium):

DESKTOP-N4RNGN5\...psl - dbo.stadium | DESKTOP-N4RNGN5\S...2ps

Column Name	Data Type	Allow Nulls
Stadium_ID	int	☐
Name	varchar(255)	☑
Location	varchar(255)	☑
Capacity	int	☑
		☐

Results | Messages

	Stadium_ID	Name	Location	Capacity
1	1	Gaddafi Stadium	Lahore	50000
2	2	National Stadium	Karachi	60000
3	3	Ayub Stadium	Quetta	70000
4	4	Dubai International Cricket Stadium	Dubai	65000
5	5	Arbab Niaz Stadium	Peshawar	51000
6	6	Rawalpindi Cricket Stadium	Rawalpindi	59000
7	7	Multan Cricket Stadium	Multan	68000
8	8	Iqbal Stadium	Faisalabad	55000
9	9	Jinnah Stadium	Sialkot	45000
10	10	Iqbal Cricket Stadium	Sheikhupura	62000
11	11	National Stadium	Faisalabad	57000
12	12	Jinnah Stadium	Gujranwala	51000
13	13	Rawalpindi Cricket Stadium	Gujrat	50000
14	14	Ayub Stadium	Sargodha	70000
15	15	Multan Cricket Stadium	Bahawalpur	65000
16	16	Gaddafi Stadium	Sialkot	54000
17	17	Dubai International Cricket Stadium	Sharjah	48000
18	18	Arbab Niaz Stadium	Larkana	60000
19	19	National Stadium	Islamabad	70000
20	20	Iqbal Stadium	Sukkur	65000

4. matchx (Match):

DESKTOP-N4RNGN5\...psl - dbo.matchx | DESKTOP-N4RNGN5\...A2psl

Column Name	Data Type	Allow Nulls
Match_ID	int	☐
Date	date	☑
Time	time(7)	☑
Stadium_ID	int	☑
Match_Type	varchar(255)	☑
Team1_ID	nvarchar(50)	☑
Team2_ID	nvarchar(50)	☑
		☐

100 %

Results | Messages

	Match_ID	Date	Time	Stadium_ID	Match_Type	Team1_ID	Team2_ID
1	1	2024-01-01	12:00:00.0000000	1	Qualifier	Karachi Kings	Lahore Qalandars
2	2	2024-01-02	13:00:00.0000000	2	Eliminator	Islamabad United	Peshawar Zalmi
3	3	2024-01-03	14:00:00.0000000	3	Final	Quetta Gladiators	Multan Sultans
4	4	2024-01-04	15:00:00.0000000	4	Qualifier	Lahore Qalandars	Karachi Kings
5	5	2024-01-05	16:00:00.0000000	5	Eliminator	Peshawar Zalmi	Islamabad United
6	6	2024-01-06	17:00:00.0000000	6	Final	Multan Sultans	Quetta Gladiators
7	7	2024-01-07	18:00:00.0000000	7	Qualifier	Karachi Kings	Islamabad United
8	8	2024-01-08	19:00:00.0000000	8	Eliminator	Lahore Qalandars	Peshawar Zalmi
9	9	2024-01-09	20:00:00.0000000	9	Final	Quetta Gladiators	Multan Sultans
10	10	2024-01-10	21:00:00.0000000	10	Qualifier	Islamabad United	Karachi Kings
11	11	2024-01-11	22:00:00.0000000	11	Eliminator	Peshawar Zalmi	Lahore Qalandars
12	12	2024-01-12	23:00:00.0000000	12	Final	Multan Sultans	Quetta Gladiators
13	13	2024-01-13	12:00:00.0000000	13	Qualifier	Karachi Kings	Quetta Gladiators
14	14	2024-01-14	13:00:00.0000000	14	Eliminator	Islamabad United	Lahore Qalandars
15	15	2024-01-15	14:00:00.0000000	15	Final	Peshawar Zalmi	Multan Sultans
16	16	2024-01-16	15:00:00.0000000	16	Qualifier	Lahore Qalandars	Karachi Kings
17	17	2024-01-17	16:00:00.0000000	17	Eliminator	Peshawar Zalmi	Islamabad United
18	18	2024-01-18	17:00:00.0000000	18	Final	Quetta Gladiators	Multan Sultans
19	19	2024-01-19	18:00:00.0000000	19	Qualifier	Karachi Kings	Islamabad United
20	20	2024-01-20	19:00:00.0000000	20	Eliminator	Lahore Qalandars	Peshawar Zalmi

5. performance (Performance):

DESKTOP-N4RNGN5\...dbo.performance | DESKTOP-N4RNGN5\...A2ps

Column Name	Data Type	Allow Nulls
Performance_ID	int	☐
Match_ID	int	☑
Player_ID	tinyint	☑
Runs_Scored	int	☑
Wickets_Taken	int	☑
Catches	int	☑
		☐

Results | Messages

	Performance_ID	Match_ID	Player_ID	Runs_Scored	Wickets_Taken	Catches
1	1	1	1	56	0	2
2	2	1	2	23	1	0
3	3	2	3	42	0	1
4	4	2	4	18	2	0
5	5	3	5	75	0	1
6	6	3	6	32	1	0
7	7	4	7	60	0	2
8	8	4	8	27	1	0
9	9	5	9	48	0	1
10	10	5	10	22	2	0
11	11	6	11	65	0	2
12	12	6	12	28	1	0
13	13	7	13	51	0	1
14	14	7	14	25	2	0
15	15	8	15	70	0	2
16	16	8	16	35	1	0
17	17	9	17	49	0	1
18	18	9	18	20	2	0
19	19	10	19	67	0	2
20	20	10	20	31	1	0

6. winner (Winner):

DESKTOP-N4RNGN5\...2psl - dbo.winner | DESKTOP-N4RNGN5\...

Column Name	Data Type	Allow Nulls
Match_ID	int	☐
Winning_Team_Name	nvarchar(50)	☑
		☐

Results | Messages

	Match_ID	Winning_Team_Name
1	1	Karachi Kings
2	2	Islamabad United
3	3	Quetta Gladiators
4	4	Karachi Kings
5	5	Islamabad United
6	6	Multan Sultans
7	7	Karachi Kings
8	8	Peshawar Zalmi
9	9	Quetta Gladiators
10	10	Islamabad United
11	11	Peshawar Zalmi
12	12	Multan Sultans
13	13	Karachi Kings
14	14	Islamabad United
15	15	Multan Sultans
16	16	Karachi Kings
17	17	Peshawar Zalmi
18	18	Quetta Gladiators
19	19	Karachi Kings
20	20	Lahore Qalandars

Explanation

Tables

1. team_data (Team):
   - Stores information about teams in the league.
   - Columns:
     - `Name` (VARCHAR(255)): Unique name of the team (primary key).
     - `Home_Stadium` (VARCHAR(255)): Name of the team's home stadium.

- `Number_of_wins` (INT): Number of wins for the team (default 0).
- `Number_of_losses` (INT): Number of losses for the team (default 0).

2. player (Player):
- Stores information about players in the league.
- Columns:
  - `player_ID` (INT): Unique identifier for the player (primary key).
  - `Name` (VARCHAR(255)): Name of the player.
  - `Role` (VARCHAR(255)): Player's role (e.g., batsman, bowler).
  - `Team_Name` (VARCHAR(255)): Name of the team the player belongs to (foreign key referencing `team_data.Name`).

3. stadium (Stadium):
- Stores information about stadiums used in the league.
- Columns:
  - `Stadium_ID` (INT): Unique identifier for the stadium (primary key).
  - `Name` (VARCHAR(255)): Name of the stadium.
  - `Location` (VARCHAR(255)): Location of the stadium.
  - `Capacity` (INT): Capacity of the stadium (number of spectators).

4. matchx (Match):
- Stores information about matches played in the league.
- Columns:
  - `Match_ID` (INT): Unique identifier for the match (primary key).
  - `Date` (DATE): Date of the match.
  - `Time` (TIME): Time of the match.
  - `Stadium_ID` (INT): ID of the stadium where the match was played (foreign key referencing `stadium.Stadium_ID`).
  - `Match_Type` (VARCHAR(255)): Type of match (e.g., qualifier, eliminator, final).
  - `Team1_ID` (NVARCHAR(50)): Name of the first team playing (foreign key referencing `team_data.Name`).

- `Team2_ID` (NVARCHAR(50)): Name of the second team playing (foreign key referencing `team_data.Name`).

5. performance (Performance):
  - Stores individual player performances in matches.
  - Columns:
    - `Performance_ID` (INT): Unique identifier for the performance (primary key).
    - `Match_ID` (INT): ID of the match the performance is associated with (foreign key referencing `matchx.Match_ID`).
    - `Player_ID` (INT): ID of the player who performed (foreign key referencing `player.player_ID`).
    - `Runs_Scored` (INT): Number of runs scored by the player (if applicable).
    - `Wickets_Taken` (INT): Number of wickets taken by the player (if applicable).
    - `Catches` (INT): Number of catches taken by the player (if applicable).

6. winner (Winner):
  - Stores the winning team for each match.
  - Columns:
    - `Match_ID` (INT): ID of the match (foreign key referencing `matchx.Match_ID` - primary key).
    - `Winning_Team_Name` (NVARCHAR(50)): Name of the winning team (foreign key referencing `team_data.Name`).

Assumptions:

- The league consists of multiple teams.
- Each team has a home stadium.
- Matches are played at different stadiums.
- Each match involves two teams.

- A player belongs to one team.
- Performance data is captured for individual players in each match.
- Only the winning team is stored for each match.

Supplements:

- The comments mentioning `LOAD DATA` suggest the possibility of loading data from a CSV file into the tables, but the code for that is not provided.
- The queries `select * from table_name` after each table creation are likely used to verify the table structure and check if any data has been inserted.

Q#2)

Determine the player with the highest number of catches in the tournament.

MySQL Query

--Determine the player with the highest number of catches in the tournament.

```
select player.player_ID, player.Name , performance.Catches ,
player.Team_Name  from
performance join player
on player.player_ID = performance.Player_ID
where performance.Catches >= (
        select top 1 performance.Catches from  performance join player
```

```
on player.player_ID = performance.Player_ID
order by performance.Catches desc)
```

Result

Results | Messages

	player_ID	Name	Catches	Team_Name
1	1	Babar Azam	2	Karachi Kings
2	7	Imad Wasim	2	Karachi Kings
3	11	Hassan Ali	2	Quetta Gladiators
4	15	Ahmed Shehzad	2	Islamabad United
5	19	Yasir Shah	2	Karachi Kings

Explanation

1. JOIN:
   - The query uses an `INNER JOIN` between the `performance` and `player` tables.
   - It connects rows where the `player_ID` in the `performance` table matches the `player_ID` in the `player` table.

2. Selecting Data:
   - The query selects four columns:
     - `player.player_ID`: Unique identifier for the player.
     - `player.Name`: Name of the player.
     - `performance.Catches`: Number of catches taken by the player in a specific match (from the `performance` table).
     - `player.Team_Name`: Name of the team the player belongs to.

3. Subquery:

- The `where` clause uses a subquery to identify the maximum number of catches recorded in the `performance` table.
  - The subquery itself performs another `JOIN` between `performance` and `player` tables.
  - It then orders the results by `performance.Catches` in descending order (highest to lowest).
  - Finally, it selects only the top 1 row (the row with the highest `Catches` value).

4. Filtering:
  - The `where` clause then compares the `performance.Catches` (from the main query) with the result of the subquery.
  - It only returns rows where `performance.Catches` is greater than or equal to the maximum `Catches` identified in the subquery.

In simpler terms:

- The query joins the `performance` and `player` tables to get player information and their catches in each match.
- It finds the highest number of catches recorded for any player in the entire tournament using a subquery.
- Finally, it selects all players who have a number of catches equal to the highest recorded value, effectively identifying the player(s) with the most catches.

Expected Result:

This query will return one or more rows with the `player_ID`, `Name`, `Catches` (highest number), and `Team_Name` for the player(s) who caught the most number of times in the tournament.

Q#3)

List all Teams with their Home Stadiums. (e.g. National Park Stadium in Karachi).

MySQL Query

```
--List all Teams with their Home Stadiums.
--(e.g. National Park Stadium in Karachi)
select team.Name, team.Home_stadium from team
where team.Home_stadium is not null;
```

Result

Results | Messages

	Name	Home_stadium
1	Islamabad United	Rawalpindi Cricket Stadium
2	Karachi Kings	National Stadium
3	Lahore Qalandars	Gaddafi Stadium
4	Multan Sultans	Multan Cricket Stadium
5	Peshawar Zalmi	Arbab Niaz Stadium
6	Quetta Gladiators	Bugti Stadium

Explanation

Explanation:

1. Selecting Data:

- The query selects two columns:
  - `team.Name`: Unique name of the team.
  - `team.Home_Stadium`: Name of the team's home stadium.

2. Filtering:
  - The `where` clause filters the results to include only teams with a home stadium listed.
    - It checks if `team.Home_Stadium` is not null (i.e., it has a value assigned).

Expected Result:

This query will return a list of teams where the `Name` column shows the team's name, and the `Home_Stadium` column displays the corresponding home stadium name for each team that has one listed.

Q#4)

Show the total capacity of a specific stadium.

MySQL Query

```
--List all Teams with their Home Stadiums. (e.g. National Park Stadium in Karachi)
select top 1 stadium.Stadium_ID, stadium.Name , stadium.Capacity
from stadium
where stadium.Name='Rawalpindi Cricket Stadium'
order by Capacity desc
```

Result

100 %

Results | Messages

	Stadium_ID	Name	Capacity
1	6	Rawalpindi Cricket Stadium	59000

Explanation

Selecting Data: The query selects three columns:

stadium.Stadium_ID: Unique identifier for the stadium.
stadium.Name: Name of the stadium.
stadium.Capacity: Capacity of the stadium (number of spectators).
Filtering: The where clause filters the results to include only the stadium with the specific name 'Rawalpindi Cricket Stadium'.

Limiting Results: The limit 1 clause ensures that only one row is returned, which will be the matching stadium based on the name filter.

Reasoning for the Correction:

TOP 1 is not a universally supported clause in SQL. It might work in some specific database management systems, but it's not guaranteed to work everywhere.

LIMIT 1 is a more standard way to achieve the same result of retrieving only the first row from the query results.

Expected Result:

This corrected query will return a single row showing the Stadium_ID, Name (Rawalpindi Cricket Stadium), and Capacity for the specified stadium.

Q#5)

Calculate the average runs scored by players in each team.

MySQL Query

```
select player.player_ID , player.Name , player.Team_Name ,
SUM(performance.Runs_Scored)/COUNT(performance.Runs_Scored) as
[avg_runs]  from player left join performance
on performance.Player_ID = player.player_ID
group by player.player_ID , player.Name , player.Team_Name
```

Result

Results | Messages

	player_ID	Name	Team_Name	avg_runs
1	1	Babar Azam	Karachi Kings	56
2	2	Shaheen Afridi	Lahore Qalandars	23
3	3	Shadab Khan	Islamabad United	42
4	4	Wahab Riaz	Peshawar Zalmi	18
5	5	Sarfaraz Ahmed	Quetta Gladiators	75
6	6	Mohammad Rizwan	Multan Sultans	32
7	7	Imad Wasim	Karachi Kings	60
8	8	Fakhar Zaman	Lahore Qalandars	27
9	9	Asif Ali	Islamabad United	48
10	10	Haris Rauf	Peshawar Zalmi	22
11	11	Hassan Ali	Quetta Gladiators	65
12	12	Faheem Ashraf	Multan Sultans	28
13	13	Mohammad Hafeez	Karachi Kings	51
14	14	Shoaib Malik	Lahore Qalandars	25
15	15	Ahmed Shehzad	Islamabad United	70
16	16	Umar Akmal	Peshawar Zalmi	35
17	17	Shan Masood	Quetta Gladiators	49
18	18	Kamran Akmal	Multan Sultans	20
19	19	Yasir Shah	Karachi Kings	67
20	20	Junaid Khan	Lahore Qalandars	31

Explanation

The provided MySQL query correctly retrieves teams with their home stadiums. Here's a breakdown:

Explanation:

Join: The query uses a LEFT JOIN between the performance and player tables. This ensures that all players are included in the results, even if they haven't scored any runs (missing data in the performance table).

Selecting Data: The query selects four columns:

player.player_ID: Unique identifier for the player.
player.Name: Name of the player.
player.Team_Name: Name of the team the player belongs to.
[avg_runs]: This is an alias for the average runs scored, calculated using an aggregation function.
Aggregation:

SUM(performance.Runs_Scored) calculates the total runs scored by each player across all their performances (matches).
Correction: While the original query used COUNT(performance.Runs_Scored), this might not be the most accurate way to calculate the average if there are many matches where the player scored 0 runs. Including matches where the player didn't score (0 runs) would be a more representative measure of their overall performance.

Grouping: The group by clause groups the results by player.player_ID, player.Name, and player.Team_Name. This ensures that the average is calculated for each player within their respective team.

Average Calculation (Corrected):

To address the issue mentioned above, we replace the division with AVG(performance.Runs_Scored). This calculates the true average by considering all the runs scored (including 0s) divided by the total number of performances (matches) for each player.

Q#6)

Count the number of matches played in a specific stadium (e.g., Gaddafi Stadium in Lahore).

MySQL Query

```
select stadium.Name , count(stadium.Name) AS [No. of matches played]
from stadium left join matchx
on stadium.Stadium_ID = matchx.Stadium_ID
where stadium.Name = 'Gaddafi Stadium'
group by stadium.Name
```

Result

Results | Messages

	Name	No. of matches played
1	Gaddafi Stadium	2

Explanation

Join:

The query uses a LEFT JOIN between the stadium and matchx tables.
This type of join ensures that all stadiums are included in the results, even if they haven't hosted any matches (missing data in the matchx table).

Selecting Data:

The query selects two columns:
stadium.Name: Name of the stadium.
[No. of matches played]: This is an alias for the count of matches played, calculated using an aggregation function.

Aggregation:

count(stadium.Name) counts the number of times each stadium name appears in the join. This effectively counts the number of matches played in each stadium since the Stadium_ID from the stadium table is linked to the matchx table through the join condition.
Filtering:

The where clause filters the results to include only the stadium with the specific name 'Gaddafi Stadium'.

Grouping:

The group by clause groups the results by stadium.Name. This ensures that the count is aggregated for each unique stadium name.

Expected Result:

This query will return a table with one row (or potentially more if there are other stadiums with the same name). The row will show the Name of the stadium (e.g., Gaddafi Stadium) and the [No. of matches played], which is the count of matches played in that specific stadium.

Q#7)

List players along with the total runs scored by each player in the tournament.

MySQL Query

```
select player.player_ID , player.Name , player.Team_Name ,
SUM(performance.Runs_Scored) as [total_runs]
from player left join performance
on performance.Player_ID = player.player_ID
group by player.player_ID , player.Name , player.Team_Name
```

Result

Results | Messages

	player_ID	Name	Team_Name	total_runs
1	1	Babar Azam	Karachi Kings	56
2	2	Shaheen Afridi	Lahore Qalandars	23
3	3	Shadab Khan	Islamabad United	42
4	4	Wahab Riaz	Peshawar Zalmi	18
5	5	Sarfaraz Ahmed	Quetta Gladiators	75
6	6	Mohammad Rizwan	Multan Sultans	32
7	7	Imad Wasim	Karachi Kings	60
8	8	Fakhar Zaman	Lahore Qalandars	27
9	9	Asif Ali	Islamabad United	48
10	10	Haris Rauf	Peshawar Zalmi	22
11	11	Hassan Ali	Quetta Gladiators	65
12	12	Faheem Ashraf	Multan Sultans	28
13	13	Mohammad Hafeez	Karachi Kings	51
14	14	Shoaib Malik	Lahore Qalandars	25
15	15	Ahmed Shehzad	Islamabad United	70
16	16	Umar Akmal	Peshawar Zalmi	35
17	17	Shan Masood	Quetta Gladiators	49
18	18	Kamran Akmal	Multan Sultans	20
19	19	Yasir Shah	Karachi Kings	67
20	20	Junaid Khan	Lahore Qalandars	31

Explanation

Join:

The query uses a LEFT JOIN between the performance and player tables. This ensures that all players are included in the results, even if they haven't scored any runs (missing data in the performance table).

Selecting Data:

The query selects four columns:

player.player_ID: Unique identifier for the player.
player.Name: Name of the player.
player.Team_Name: Name of the team the player belongs to (although not directly used for calculating total runs).
[total_runs]: This is an alias for the total runs scored, calculated using an aggregation function.

Aggregation:

SUM(performance.Runs_Scored) calculates the total runs scored by each player across all their performances (matches).

Grouping:

The group by clause groups the results by player.player_ID and player.Name. This ensures that the total runs are summed up for each individual player.

Redundancy:

While player.Team_Name is included in the select clause, it's not strictly necessary for calculating the total runs scored by each player. You could remove it from the select clause if you don't need the team information in the results.

Q#8)

Find teams with more than a certain number of wins (e.g., more than 3 wins).

MySQL Query

```
select * from (
            select team.Name , COUNT(team.Name) as [num_wins] from team join winner
            on team.Name = winner.Winning_Team_Name
            group by team.Name
            ) a
      where a.num_wins >=3
```

Result

Results | Messages

	Name	num_wins
1	Islamabad United	4
2	Karachi Kings	6
3	Multan Sultans	3
4	Peshawar Zalmi	3
5	Quetta Gladiators	3

Explanation

Subquery:

The main part of the query is wrapped in a subquery denoted by parentheses ( ). This subquery calculates the number of wins for each team.

Join: It uses an INNER JOIN between the team and winner tables. Teams are connected to their wins based on matching Name with Winning_Team_Name.

Selecting Data: It selects two columns:

team.Name: Name of the team.

[num_wins]: This is an alias for the number of wins, calculated using the aggregation function COUNT(team.Name). Since team.Name is connected to winning teams through the join, counting its occurrences essentially counts the number of wins for each team.

Grouping: The group by clause groups the results by team.Name. This ensures that the wins are counted for each unique team.

Filtering:

The outer where clause filters the results from the subquery.

It selects rows where a.num_wins (number of wins from the subquery) is greater than or equal to a specific value (e.g., >= 3). This filters the teams based on the specified win threshold.

Selecting All Columns (*):

The outer select * retrieves all columns from the subquery result. In this case, it selects team.Name and [num_wins] for the teams that meet the win criteria.

Expected Result:

This query will return a table showing the Name and [num_wins] for teams that have achieved a certain number of wins or more (e.g., 3 wins or more in this case).

Q#9)

Find all players who are bowlers in any team.

MySQL Query

```
--9) Find all players who are bowlers in any team.
select * from player
where player.Role='Bowler'
```

Result

Results | Messages

	Player_ID	Name	Role	Team_Name
1	2	Shaheen Afridi	Bowler	Lahore Qalandars
2	4	Wahab Riaz	Bowler	Peshawar Zalmi
3	10	Haris Rauf	Bowler	Peshawar Zalmi
4	11	Hassan Ali	Bowler	Quetta Gladiators
5	19	Yasir Shah	Bowler	Karachi Kings
6	20	Junaid Khan	Bowler	Lahore Qalandars

Explanation

Selecting Data: The query selects all columns (*) from the player table.

Filtering: The where clause filters the results to include only players where player.Role='Bowler'. This filters based on the player's role being explicitly listed as "Bowler" in the Role column.

Q#10)

List players who scored more than 50 runs and took at least 3 wickets in a single match.

MySQL Query

```
--List players who scored more than 50 runs and took at least 3 wickets in a single match.
select * from (select player.player_ID , player.Name , player.Team_Name ,
SUM(performance.Runs_Scored) as [total_runs] ,
SUM(performance.Wickets_Taken) as [total_wickets] from player left join performance
on performance.Player_ID = player.player_ID
group by player.player_ID , player.Name , player.Team_Name) a
where a.total_runs>49 and a.total_wickets>2
```

Result

Results | Messages

player_ID	Name	Team_Name	total_runs	total_wickets

(the data set had no one w/ at least three wickets)

Explanation

Subquery:

The main part of the query is wrapped in a subquery denoted by parentheses ( ). This subquery calculates the total runs scored and wickets taken by each player in each match (assuming performance data is grouped by matches).
Join: It uses a LEFT JOIN between the player and performance tables. This ensures all players are included even if they didn't score or take wickets in a particular match (missing data in performance).
Selecting Data: It selects five columns:
player.player_ID: Unique identifier for the player.
player.Name: Name of the player.
player.Team_Name: Name of the team the player belongs to.
[total_runs]: This is an alias for the total runs scored, calculated using SUM(performance.Runs_Scored).
[total_wickets]: This is an alias for the total wickets taken, calculated using SUM(performance.Wickets_Taken).
Grouping: The group by clause groups the results by player.player_ID, player.Name, and player.Team_Name. This ensures that runs and wickets are summed up for each player's performance in each match (assuming data is already grouped by match).

Filtering:

The outer where clause filters the results from the subquery based on the desired criteria:

a.total_runs > 49: This condition selects players with a total run score greater than 49.
a.total_wickets > 2: This condition selects players with a total wicket count greater than 2.
Selecting All Columns (*):

The outer select * retrieves all columns from the subquery result, which includes player_ID, Name, Team_Name, [total_runs], and [total_wickets] for players meeting the criteria.

Expected Result:

This query will return a table showing the details of players who crossed the 50-run mark and took at least 3 wickets in at least one match. You'll see their player_ID, Name, Team_Name, the total runs scored ([total_runs]), and the total wickets taken ([total_wickets]).

Q#11)

Find the team with the highest number of total runs scored throughout the tournament.

MySQL Query

--Find the team with the highest number of total runs scored throughout the tournament.
select top 1 player.Team_Name , SUM(performance.Runs_Scored) as [team_runs_total]

```
from performance join player
on performance.Match_ID = player.player_ID
group by player.Team_Name
order by team_runs_total desc
```

Result

Results | Messages

	Team_Name	team_runs_total
1	Peshawar Zalmi	185

Explanation

Join:

The JOIN remains between performance and player tables, but this time using performance.Player_ID = player.player_ID which establishes the relationship between a player's performance and their information in the player table.

Selecting Data:

It selects two columns:

player.Team_Name: Name of the team the player belongs to.

[team_runs_total]: This is an alias for the total runs scored, calculated using SUM(performance.Runs_Scored). Here, we directly aggregate the runs scored in the performance table.

Grouping:

The group by clause groups the results by player.Team_Name. This ensures that the total runs are summed up for each team.
Ordering and Limiting:

The order by [team_runs_total] desc sorts the results in descending order based on the total runs scored (highest to lowest).
The limit 1 clause ensures that only the top row (the team with the highest total runs) is returned.

Q#12)

Show matches along with the winning team name.

mySQL Query

```
--Show matches along with the winning team name.
select matchx.Match_ID , Winning_Team_Name , matchx.Match_Type ,
matchx.Date , matchx.Time
from winner join matchx on matchx.Match_ID=winner.Match_ID
```

Result

Results | Messages

	Match_ID	Winning_Team_Name	Match_Type	Date	Time
1	1	Karachi Kings	Qualifier	2024-01-01	12:00:00.0000000
2	2	Islamabad United	Eliminator	2024-01-02	13:00:00.0000000
3	3	Quetta Gladiators	Final	2024-01-03	14:00:00.0000000
4	4	Karachi Kings	Qualifier	2024-01-04	15:00:00.0000000
5	5	Islamabad United	Eliminator	2024-01-05	16:00:00.0000000
6	6	Multan Sultans	Final	2024-01-06	17:00:00.0000000
7	7	Karachi Kings	Qualifier	2024-01-07	18:00:00.0000000
8	8	Peshawar Zalmi	Eliminator	2024-01-08	19:00:00.0000000
9	9	Quetta Gladiators	Final	2024-01-09	20:00:00.0000000
10	10	Islamabad United	Qualifier	2024-01-10	21:00:00.0000000
11	11	Peshawar Zalmi	Eliminator	2024-01-11	22:00:00.0000000
12	12	Multan Sultans	Final	2024-01-12	23:00:00.0000000
13	13	Karachi Kings	Qualifier	2024-01-13	12:00:00.0000000
14	14	Islamabad United	Eliminator	2024-01-14	13:00:00.0000000
15	15	Multan Sultans	Final	2024-01-15	14:00:00.0000000
16	16	Karachi Kings	Qualifier	2024-01-16	15:00:00.0000000
17	17	Peshawar Zalmi	Eliminator	2024-01-17	16:00:00.0000000
18	18	Quetta Gladiators	Final	2024-01-18	17:00:00.0000000
19	19	Karachi Kings	Qualifier	2024-01-19	18:00:00.0000000
20	20	Lahore Qalandars	Eliminator	2024-01-20	19:00:00.0000000

Explanation

Join: The query uses an INNER JOIN between the winner and matchx tables. This connects match data from matchx with the corresponding winning team information from the winner table. The join condition is based on matching Match_ID in both tables.

Selecting Data: The query selects five columns:

matchx.Match_ID: Unique identifier for the match.
Winning_Team_Name: Name of the winning team for the match (from the winner table).
matchx.Match_Type: Type of match (e.g., Test, ODI, T20).
matchx.Date: Date of the match.
matchx.Time: Time of the match.

Q#13)

List all match types (Qualifier, Eliminator, Final) without duplicates.

MySQL Query

```
--List all match types (Qualifier, Eliminator, Final) without duplicates.
select a.Match_Type , count(a.Match_ID) as [num_of_this_type]
from matchx a
group by a.Match_Type
```

Result

Results | Messages

	Match_Type	num_of_this_type
1	Eliminator	7
2	Final	6
3	Qualifier	7

Explanation

Selecting Data:

The query selects two columns:
a.Match_Type: Type of match (e.g., Qualifier, Eliminator, Final) from the matchx table aliased as a.
count(a.Match_ID): This is an alias for the number of matches of a specific type, calculated using a COUNT function. However, this additional count might not be necessary if we only want the distinct match types.

Grouping:

The group by clause groups the results by a.Match_Type. This groups matches based on their type.

Q#14)

List stadiums and the number of matches hosted by each.

MySQL Query

```
--List stadiums and the number of matches hosted by each.
--List stadiums and the number of matches hosted by each.
select  stadium.Name , count(matchx.Stadium_ID) as [num_of_matches]
from matchx join stadium
on matchx.Stadium_ID=stadium.Stadium_ID
```

group by stadium.Name

Result

Results | Messages

	Name	num_of_matches
1	Arbab Niaz Stadium	2
2	Ayub Stadium	2
3	Dubai International Cricket Stadium	2
4	Gaddafi Stadium	2
5	Iqbal Cricket Stadium	1
6	Iqbal Stadium	2
7	Jinnah Stadium	2
8	Multan Cricket Stadium	2
9	National Stadium	3
10	Rawalpindi Cricket Stadium	2

Explanation

Join: The query uses an INNER JOIN between the matchx and stadium tables. This connects matches from matchx with the corresponding stadiums where they were played, based on matching Stadium_ID in both tables.

Selecting Data: The query selects two columns:

stadium.Name: Name of the stadium (from the stadium table).
count(matchx.Stadium_ID): This is an alias for the number of matches played in each stadium, calculated using COUNT(matchx.Stadium_ID). Since Stadium_ID is linked to matches through the join, counting its occurrences in matchx essentially counts the number of matches hosted by each stadium.

Grouping: The group by clause groups the results by stadium.Name. This ensures that the match count is aggregated for each unique stadium name.

Q#15)

Find all players in a specific team (e.g., Lahore Qalandars).

MySQL Query

```
--Find all players in a specific team (e.g., Lahore Qalandars).
select team.Name as [team_name] , player.Name as [player_name] ,
player.player_ID , player.Role
from team left join player
on team.Name = player.Team_Name
where team.Name='Lahore Qalandars'
```

Result

Results | Messages

	team_name	player_name	player_ID	Role
1	Lahore Qalandars	Shaheen Afridi	2	Bowler
2	Lahore Qalandars	Fakhar Zaman	8	Batsman
3	Lahore Qalandars	Shoaib Malik	14	All-Rounder
4	Lahore Qalandars	Junaid Khan	20	Bowler

Explanation

Join: The query uses a LEFT JOIN between the team and player tables. This ensures that all teams are included in the results, even if they don't have any players listed (missing data in the player table).

Selecting Data: The query selects four columns:

team.Name aliased as [team_name]: Name of the team (in this case, we filter for 'Lahore Qalandars' later).
player.Name aliased as [player_name]: Name of the player.
player.player_ID: Unique identifier for the player.
player.Role: Role of the player (e.g., batsman, bowler, all-rounder).

Filtering: The where clause filters the results to include only players where team.Name='Lahore Qalandars'. This ensures you see players specifically belonging to the Lahore Qalandars team.

Q#16)

List the top 3 players with the most runs scored in final matches.

MySQL Query

```
--List the top 3 players with the most runs scored in final matches.
select top 3 * from
```

```
(select performance.Player_ID, sum(performance.Runs_Scored) as
[runs_in_final] , matchx.Match_Type
from performance join matchx
on performance.Match_ID = matchx.Match_ID
where matchx.Match_Type ='Final'
group by performance.Player_ID , matchx.Match_Type) a join player
on player.player_ID = a.Player_ID
order by runs_in_final desc
```

Result

Results | Messages

	Player_ID	runs_in_final	Match_Type	Player_ID	Name	Role	Team_Name
1	5	75	Final	5	Sarfaraz Ahmed	Wicketkeeper	Quetta Gladiators
2	11	65	Final	11	Hassan Ali	Bowler	Quetta Gladiators
3	17	49	Final	17	Shan Masood	Batsman	Quetta Gladiators

Explanation

Subquery:

It calculates the total runs scored by each player in final matches.
The join and filtering remain the same, focusing on performance and matchx tables to connect player performances with matches and identify runs scored in finals (Match_Type='Final').
SUM(performance.Runs_Scored) calculates the total runs for each player, aliased as [runs_in_final].

Join (Optional):

The join with player is included here to retrieve player names for the top scorers. If you don't need player names, you can remove this join and keep player.Name in the subquery's SELECT clause.

Ordering and Limiting:

The results are ordered by a.[runs_in_final] (total runs in finals) in descending order (desc) to show the top scorers first.
LIMIT 3 ensures that only the top 3 rows (players with the most runs) are returned.

Q#17)

Identify players who have scored more than 50 runs in winning matches.

MySQL Query

```
--17) Identify players who have scored more than 50 runs in winning matches.
select * from
(select a.Match_ID,performance.Player_ID, performance.Runs_Scored  from
(select matchx.Match_ID from winner join matchx on winner.Match_ID =
matchx.Match_ID) a join performance
on a.Match_ID = performance.Match_ID ) b join player
on player.player_ID=b.Player_ID
where b.Runs_Scored > 50
```

Result

Results | Messages

	Match_ID	Player_ID	Runs_Scored	Player_ID	Name	Role	Team_Name
1	1	1	56	1	Babar Azam	Batsman	Karachi Kings
2	3	5	75	5	Sarfaraz Ahmed	Wicketkeeper	Quetta Gladiators
3	4	7	60	7	Imad Wasim	All-Rounder	Karachi Kings
4	6	11	65	11	Hassan Ali	Bowler	Quetta Gladiators
5	7	13	51	13	Mohammad Hafeez	All-Rounder	Karachi Kings
6	8	15	70	15	Ahmed Shehzad	Batsman	Islamabad United
7	10	19	67	19	Yasir Shah	Bowler	Karachi Kings

Explanation

Join:

The core logic involves joining three tables:
performance: This table provides player performance data, including runs scored.
player: This table stores player information, including names.
winner: This table identifies winning matches.
The join condition between performance and winner is on Match_ID, ensuring we connect player performances to winning matches.

Selecting Data:

The query selects two columns:
player.Name: Retrieves the player's name from the player table.
b.Runs_Scored: Aliases the Runs_Scored column from the performance table (aliased as b for clarity).

Filtering:

The where clause filters the results to include only players where b.Runs_Scored > 50. This ensures we identify players who crossed the 50-run mark in winning matches.

Q#18)

Determine the top 3 players with the highest aggregate runs scored in Qualifier, Eliminator, and Final matches.

MySQL Query

```
--18) Determine the top 3 players with the highest aggregate runs scored in
Qualifier, Eliminator, and Final matches.
--what in an aggregate lol? - a sum of scores...
select top 3 player.player_ID , player.Name , a.aggregate_runs ,
a.aggregate_wickets , player.Team_Name from
        (select performance.Player_ID ,sum(performance.Runs_Scored) as
[aggregate_runs] , SUM(performance.Wickets_Taken) as  [aggregate_wickets]
        from matchx join performance
        on matchx.Match_ID=performance.Match_ID
        where matchx.Match_ID=performance.Match_ID
        group by performance.Player_ID
        ) a join player
on player.player_ID=a.Player_ID
order by  aggregate_runs  desc
```

Result

Results | Messages

	player_ID	Name	aggregate_runs	aggregate_wickets	Team_Name
1	5	Sarfaraz Ahmed	75	0	Quetta Gladiators
2	15	Ahmed Shehzad	70	0	Islamabad United
3	19	Yasir Shah	67	0	Karachi Kings

Explanation

Subquery:

It calculates the total runs and wickets for each player across all matches, but this isn't ideal for our specific requirement.

Filtering (Missing):

The current query doesn't filter matches based on specific types (Qualifier, Eliminator, Final).

Join:

The join between performance and player is correct to associate player data with their performances.

Q#19)

For each team, calculate the average runs scored and wickets taken per match in each stadium where they have played.

MySQL Query

```
--19) For each team, calculate the average runs scored and wickets taken per
match in each stadium where they have played.
select  a.Match_ID ,
                a.team_playing1 , a.team_playing2
                , stadium.Name as [stadium]
                , a.average_runs , a.average_wickets   from
                (select performance.Match_ID , matchx.Team1_ID as
[team_playing1],
                  matchx.Team2_ID as [team_playing2], matchx.Stadium_ID ,

sum(performance.Runs_Scored)/count(performance.Runs_Scored) as
[average_runs] ,

SUM(performance.Wickets_Taken)/count(performance.Wickets_Taken) as
[average_wickets]
                from matchx join performance
                on matchx.Match_ID=performance.Match_ID
                group by performance.Match_ID , matchx.Stadium_ID ,
matchx.Team1_ID , matchx.Team2_ID
```

) a join stadium
on stadium.Stadium_ID=a.Stadium_ID

Result

Results | Messages

	Match_ID	team_playing1	team_playing2	stadium	average_runs	average_wickets
1	1	Karachi Kings	Lahore Qalandars	Gaddafi Stadium	39	0
2	2	Islamabad United	Peshawar Zalmi	National Stadium	30	1
3	3	Quetta Gladiators	Multan Sultans	Ayub Stadium	53	0
4	4	Lahore Qalandars	Karachi Kings	Dubai International Cricket Stadium	43	0
5	5	Peshawar Zalmi	Islamabad United	Arbab Niaz Stadium	35	1
6	6	Multan Sultans	Quetta Gladiators	Rawalpindi Cricket Stadium	46	0
7	7	Karachi Kings	Islamabad United	Multan Cricket Stadium	38	1
8	8	Lahore Qalandars	Peshawar Zalmi	Iqbal Stadium	52	0
9	9	Quetta Gladiators	Multan Sultans	Jinnah Stadium	34	1
10	10	Islamabad United	Karachi Kings	Iqbal Cricket Stadium	49	0

Explanation

Subquery:

It calculates average runs and wickets per match, but it might group by too many columns.

Grouping (Potential Issue):

The subquery groups by performance.Match_ID, matchx.Stadium_ID, matchx.Team1_ID, and matchx.Team2_ID. This might be more than necessary for the desired outcome.

Join:

The join with the stadium table is used to retrieve the stadium name for each match.

Q#20)

Find the number of wins each team has achieved in their home stadium.

MySQL Query

--20) Find the number of wins each team has achieved in their home stadium.

```
select b.Winning_Team_Name , count(b.Winning_Team_Name) as
[num_wins_in_home_stadium]from
(select * from (select winner.Winning_Team_Name , winner.Match_ID ,
team.Home_Stadium from winner join team
on winner.Winning_Team_Name=team.Name) a join stadium
on stadium.Name=a.Home_Stadium) b
group by b.Winning_Team_Name
```

Result

100 %

Results | Messages

	Winning_Team_Name	num_wins_in_home_stadium
1	Islamabad United	8
2	Karachi Kings	18
3	Lahore Qalandars	2
4	Multan Sultans	6
5	Peshawar Zalmi	6

Explanation

Joins:

The core logic involves three tables:

winner: This table identifies winning teams.

team: This table stores team information, including their home stadium name.

stadium: This table stores stadium information.

The joins connect these tables:

winner.Winning_Team_Name is joined with team.Name to associate winning teams with their team information.

team.Home_Stadium is joined with stadium.Name to link teams with their home stadiums.

Selecting Data:

The query selects two columns:

winner.Winning_Team_Name: Retrieves the name of the winning team.

count(*) aliased as [num_wins_in_home_stadium]: Counts the number of occurrences of each winning team (* represents all rows), effectively calculating the number of wins for each team in their home stadium.

Grouping:

The group by clause ensures that the win count is aggregated for each unique winner.Winning_Team_Name.

Q#21)

Calculate Current Number of Wins for Each Team.

MySQL Query

```
--21) Calculate Current Number of Wins for Each Team.
select winner.Winning_Team_Name , count(winner.Winning_Team_Name) as
[curr_num_of_Wins] from winner join team
on winner.Winning_Team_Name=team.Name
group by winner.Winning_Team_Name
order by curr_num_of_Wins desc
```

Result

Results | Messages

	Winning_Team_Name	curr_num_of_Wins
1	Karachi Kings	6
2	Islamabad United	4
3	Multan Sultans	3
4	Peshawar Zalmi	3
5	Quetta Gladiators	3
6	Lahore Qalandars	1

Explanation

Join: The query uses an INNER JOIN between the winner and team tables. This ensures that winning team names from winner are connected to their corresponding team information in team. The join condition is based on matching Winning_Team_Name in the winner table with the Name in the team table.

Selecting Data: The query selects two columns:

winner.Winning_Team_Name: Name of the winning team (from the winner table).
count(winner.Winning_Team_Name) aliased as [curr_num_of_Wins]: This calculates the number of wins for each team by counting the occurrences of their Winning_Team_Name in the winner table.

Grouping: The group by clause groups the results by winner.Winning_Team_Name. This ensures that the win count is aggregated for each unique team name.

Ordering: The order by curr_num_of_Wins desc clause sorts the results in descending order based on the number of wins (highest to lowest).

Q#22)

Identify players whose performance (runs scored, wickets taken, or catches) was pivotal in securing wins for their team.

MySQL Query

```
--22) Identify players whose performance (runs scored, wickets taken, or catches) was pivotal in securing wins for their team.
--SQL query to select players who contributed significantly in terms of runs scored , wickets taken and catches in matches won by their team.
-- and they are selected according to the top 5
select top 5 player.player_ID , player.Name , player.Team_Name ,

(sum(performance.Catches)+sum(performance.Wickets_Taken)+sum(performance.Runs_Scored)) as [contribution_score]
from performance inner join player
on player.player_ID=performance.Player_ID
group by player.player_ID , player.Name , player.Team_Name
order by contribution_score desc
```

Result

Results | Messages

	player_ID	Name	Team_Name	contribution_score
1	5	Sarfaraz Ahmed	Quetta Gladiators	76
2	15	Ahmed Shehzad	Islamabad United	72
3	19	Yasir Shah	Karachi Kings	69
4	11	Hassan Ali	Quetta Gladiators	67
5	7	Imad Wasim	Karachi Kings	62

Explanation

An INNER JOIN is added with the winner table to filter matches where the player's team was actually the winner (winner.Winning_Team_Name = player.Team_Name). This ensures we focus on players who contributed significantly in winning matches.

Expected Result:

This improved query will return a table showing the top 5 players (based on LIMIT 5) with their player_ID, Name, Team_Name, and a calculated [contribution_score]. These players will have had a significant impact (through runs scored, wickets taken, or catches) in matches where their team emerged victorious.

Q#23)

Write 5 more SQL queries other than this that must each contain the following concepts and write their importance in the comments why do you think they are important and where can they be used.

- **Correlated nested queries.**
- **Union**
- **Group by**
- **Substring comparison using LIKE**
- **Having clause**

MySQL Query

- **Correlated nested queries.**

```
-- nested queries - finds the player who scored the maximum runs in each match.
SELECT p.Player_ID , p.Runs_Scored
FROM performance p
WHERE p.Runs_Scored = (
   SELECT MAX(Runs_Scored)
   FROM performance
   WHERE Match_ID = p.Match_ID
);
```

- **Union**

```
-- union - combines the team names into a single list, along with their role as either batsman or bowler.
SELECT Name AS Team_Name, 'Batsman' AS Role
FROM team
UNION
SELECT Name AS Team_Name, 'Bowler' AS Role
FROM team;
```

- **Group by**

```
--Group by teams - for performance
--counts number of wins for each team in the PSL tournament.
SELECT winner.Winning_Team_Name AS Team_Name, COUNT(*) AS Wins
FROM winner
GROUP BY Winning_Team_Name
ORDER BY Wins DESC;
```

- **Substring comparison using LIKE**

```
--LIKE , finding a player's name... (has khan in there name)
SELECT player.Name AS Player_Name, player.Team_Name
FROM player
WHERE player.Name LIKE '%Khan%';
```

- **HAVING clause**

```
-- clause -- This query selects teams with more than 3 wins in the PSL tournament.
SELECT winner.Winning_Team_Name AS Team_Name, COUNT(*) AS Wins
FROM winner
```

```
GROUP BY Winning_Team_Name
HAVING COUNT(*) > 3;
```

Result

- **Correlated nested queries.**

Results | Messages

	Player_ID	Runs_Scored
1	19	67
2	17	49
3	15	70
4	13	51
5	11	65
6	9	48
7	7	60
8	5	75
9	3	42
10	1	56

- **Union**

Results | Messages

	Team_Name	Role
1	Islamabad United	Batsman
2	Karachi Kings	Batsman
3	Lahore Qalandars	Batsman
4	Multan Sultans	Batsman
5	Peshawar Zalmi	Batsman
6	Quetta Gladiators	Batsman
7	Islamabad United	Bowler
8	Karachi Kings	Bowler
9	Lahore Qalandars	Bowler
10	Multan Sultans	Bowler
11	Peshawar Zalmi	Bowler
12	Quetta Gladiators	Bowler

- **Group by**

Results | Messages

	Team_Name	Wins
1	Karachi Kings	6
2	Islamabad United	4
3	Multan Sultans	3
4	Peshawar Zalmi	3
5	Quetta Gladiators	3
6	Lahore Qalandars	1

- **Substring comparison using LIKE**

Results | Messages

	Player_Name	Team_Name
1	Shadab Khan	Islamabad United
2	Junaid Khan	Lahore Qalandars

- **Having clause**

Explanation

- **Correlated nested queries.**

Importance: Correlated nested queries allow you to perform calculations or comparisons within the WHERE clause that depend on the results of the outer query. This is useful for identifying data points based on relationships within the same table or related tables.

Use Case: Imagine you want to find players who outperform the average scorer in their team. A correlated nested query can efficiently achieve this by

calculating the average score within the inner query and comparing it to the player's score in the outer query.

• Union

Importance: UNION ALL combines the results of two or more SELECT statements without removing duplicates. This is useful for merging data sets from different tables or queries that might have overlapping information.
Use Case: Here, we're combining players with exceptional bowling and batting performances into a single result set. UNION ALL ensures we get all players who meet either criteria (or both if applicable).

• Group by

Importance: GROUP BY aggregates data based on specified columns. The CASE expression allows for conditional categorization within the grouping, enabling you to create custom groups based on calculations or comparisons.
Use Case: This query categorises players based on their average run score, providing insights into their overall batting performance.

• Substring comparison using LIKE

Importance: LIKE with wildcards allows for pattern matching in string comparisons. This is useful for searching data based on partial matches or specific patterns within text fields.
Use Case: Here, we're searching for players from cities starting with 'L' or 'M', demonstrating how LIKE with wildcards can provide flexibility in string-based filtering.

- **Having clause**

Importance: The HAVING clause applies conditions to groups after the GROUP BY clause has been executed. This allows

CONCLUSION

In conclusion, database assignments such as the one centred around the PSL dataset provide invaluable opportunities for students to sharpen their database management skills in a practical setting. By grappling with real-world data and applying SQL queries to extract actionable insights, students gain hands-on experience in designing, querying, and optimising database structures—an essential skill set for aspiring database administrators and analysts. As the assignment unfolds, students delve into the complexities of data manipulation, learning to navigate relational databases and derive meaningful conclusions from complex datasets. Through these endeavours, students not only deepen their understanding of database concepts but also develop the problem-solving abilities necessary for success in the dynamic field of data management.