CSAO593
DATABASE MANAGEMENT SYSTEM

PRAKALYA P V
192311418

Assignment –1

Real-Time Sports Event Tracking System:-

1). Create a database to track live sports events, including player statistics, scores, and game highlights.
Requirements:
Design tables for teams, players, games, scores, and play-by-play events.

2). Implement triggers to update real-time game statistics and scores.

3). Write SQL queries to generate reports on player performance, team standings, and historical game data.

4). Implement indexing to support high-frequency queries during live events. give answers with explanation and tabulations

1). To design a database to track live sports events with player statistics, scores, and game highlights, we'll create tables to model the entities: **Teams**, **Players**, **Games**, **Scores**, and **Play-by-Play Events**. This database will allow users to track individual game events in real time, monitor player statistics, and store final scores and highlights for each game.

DATABASE STRUCTURE:

This database design enables real-time tracking of live sports events, player statistics, scores, and game highlights. By structuring the database with tables for **Teams**, **Players**, **Games**, **Scores**, and **Play-by-Play Events**, we can efficiently manage, store, and analyse key aspects of sports games.

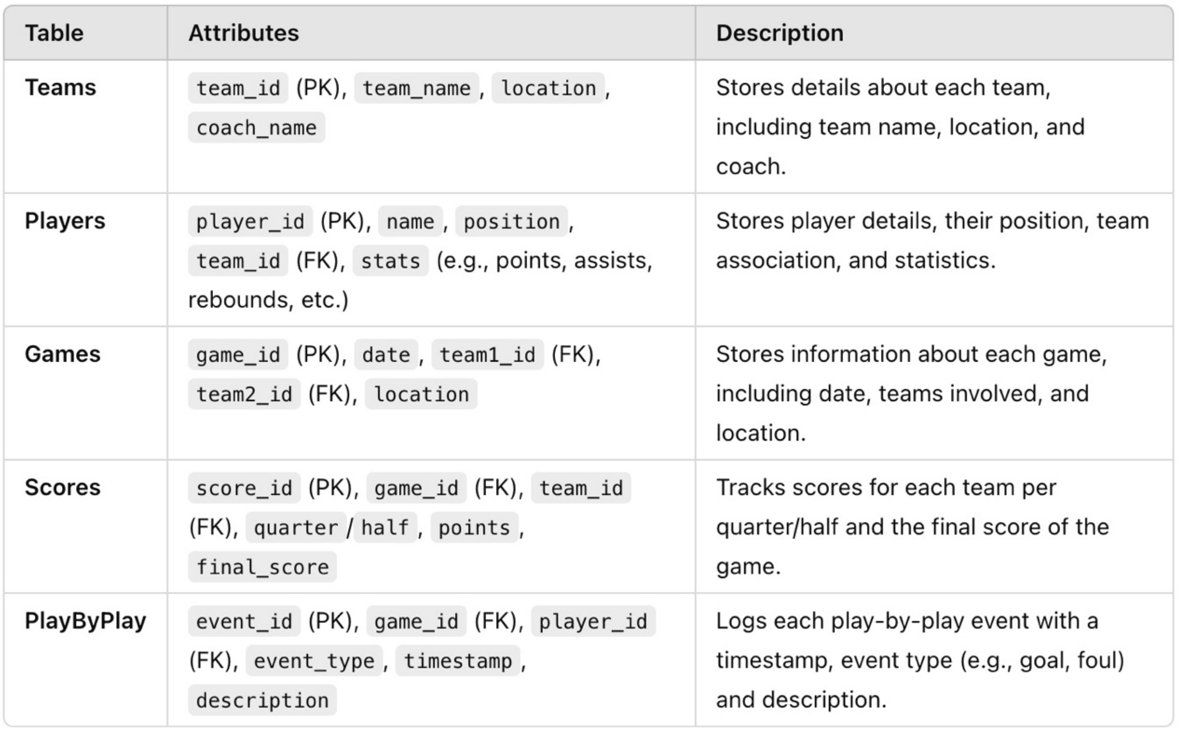


Table	Attributes	Description
Teams	`team_id` (PK), `team_name`, `location`, `coach_name`	Stores details about each team, including team name, location, and coach.
Players	`player_id` (PK), `name`, `position`, `team_id` (FK), `stats` (e.g., points, assists, rebounds, etc.)	Stores player details, their position, team association, and statistics.
Games	`game_id` (PK), `date`, `team1_id` (FK), `team2_id` (FK), `location`	Stores information about each game, including date, teams involved, and location.
Scores	`score_id` (PK), `game_id` (FK), `team_id` (FK), `quarter` / `half`, `points`, `final_score`	Tracks scores for each team per quarter/half and the final score of the game.
PlayByPlay	`event_id` (PK), `game_id` (FK), `player_id` (FK), `event_type`, `timestamp`, `description`	Logs each play-by-play event with a timestamp, event type (e.g., goal, foul) and description.

2). Triggers for Updating Real-Time Statistics:

```
DELIMITER $$

CREATE TRIGGER update_team_score
AFTER INSERT ON PlayByPlay
FOR EACH ROW
BEGIN
   -- Only update score if the event type is 'Point Scored'
   IF NEW.event_type = 'Point Scored' THEN
      UPDATE Scores
      SET points = points + NEW.points
      WHERE game_id = NEW.game_id AND team_id =
NEW.team_id
        AND quarter = NEW.quarter;  -- Update score per
quarter

      -- Update the final score for the team as well
      UPDATE Scores
      SET final_score = final_score + NEW.points
      WHERE game_id = NEW.game_id AND team_id =
NEW.team_id;
   END IF;
END$$

DELIMITER ;
```

- Player statistics are updated automatically in real time based on the play-by-play events.

Triggers to Update Team Score:

```
DELIMITER $$

CREATE TRIGGER update_team_score
AFTER INSERT ON PlayByPlay
FOR EACH ROW
BEGIN
   -- Only update score if the event type is 'Point Scored'
   IF NEW.event_type = 'Point Scored' THEN
      UPDATE Scores
      SET points = points + NEW.points
      WHERE game_id = NEW.game_id AND team_id =
NEW.team_id
        AND quarter = NEW.quarter;  -- Update score per quarter

      -- Update the final score for the team as well
      UPDATE Scores
      SET final_score = final_score + NEW.points
      WHERE game_id = NEW.game_id AND team_id =
NEW.team_id;
   END IF;
END$$

DELIMITER ;
```

- team scores are updated dynamically during the game, ensuring that the database always reflects the latest score per quarter and the final score.

These triggers facilitate a live sports tracking system that provides real-time updates without requiring manual intervention, making it easier to track game progression, individual player achievements, and team scores. This approach is highly beneficial for applications like sports analytics, live scoreboards, and sports broadcasting platforms.

3). SQL queries:

These SQL queries provide comprehensive reports on:

1. **Player Performance**: Summing individual player stats such as points, assists, and rebounds.
2. **Team Standings**: Showing standings by win-loss records or total points scored across all games.
3. **Historical Game Data**: Displaying historical records of game results and specific team performance.

These reports are valuable for team managers, analysts, and fans, enabling them to view current player stats, team standings, and historical game outcomes. They can also serve as a foundation for further analysis and insights into player and team performance trends over time.

a). on player performance.

```
SELECT p.player_id, p.name, p.position,
    SUM(CASE WHEN pbp.event_type = 'Point Scored' THEN
pbp.points ELSE 0 END) AS total_points,
    SUM(CASE WHEN pbp.event_type = 'Assist' THEN 1 ELSE 0
END) AS total_assists,
    SUM(CASE WHEN pbp.event_type = 'Rebound' THEN 1 ELSE
0 END) AS total_rebounds
FROM Players p
JOIN PlayByPlay pbp ON p.player_id = pbp.player_id
WHERE pbp.game_id = [specific_game_id]
GROUP BY p.player_id, p.name, p.position
ORDER BY total_points DESC;
```

b).on team standings.

```
SELECT t.team_name,
    SUM(CASE WHEN s.final_score > opp.final_score THEN 1
ELSE 0 END) AS wins,
    SUM(CASE WHEN s.final_score < opp.final_score THEN 1
ELSE 0 END) AS losses,
    SUM(s.final_score) AS total_points_scored,
    AVG(s.final_score) AS avg_points_per_game
FROM Teams t
JOIN Scores s ON t.team_id = s.team_id
JOIN Scores opp ON s.game_id = opp.game_id AND s.team_id <>
opp.team_id
GROUP BY t.team_name
ORDER BY wins DESC, avg_points_per_game DESC;
```

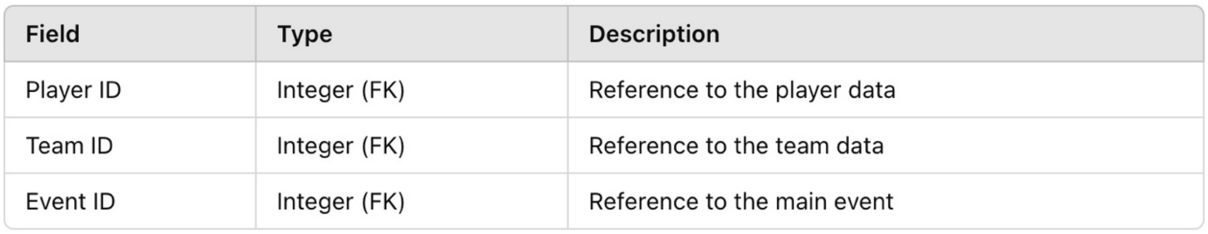
c).historical game data.

```
SELECT g.game_id, g.date,
    t1.team_name AS team1,
    t2.team_name AS team2,
    s1.final_score AS team1_score,
    s2.final_score AS team2_score
FROM Games g
JOIN Teams t1 ON g.team1_id = t1.team_id
JOIN Teams t2 ON g.team2_id = t2.team_id
JOIN Scores s1 ON g.game_id = s1.game_id AND g.team1_id =
s1.team_id
JOIN Scores s2 ON g.game_id = s2.game_id AND g.team2_id =
s2.team_id
ORDER BY g.date DESC;
```

4). 1. Primary Indexing Based on Event Time and Type

- **Explanation**: Since queries during live events typically center around real-time information (like scores, updates, and stats), it's efficient to use event time and type as primary keys. This allows queries to quickly access data related to the most recent updates.

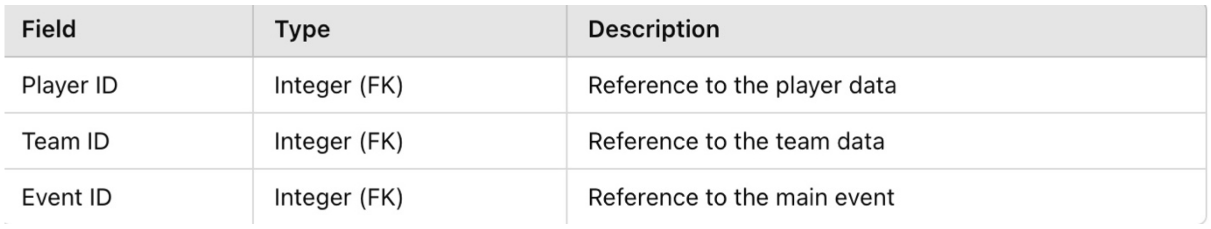
- **Structure**:
  - **Event ID** (primary key): Unique identifier for each event.
  - **Timestamp**: Record of when the update occurred.
  - **Event Type**: Type of live update (e.g., goal scored, foul, timeout).



Field	Type	Description
Player ID	Integer (FK)	Reference to the player data
Team ID	Integer (FK)	Reference to the team data
Event ID	Integer (FK)	Reference to the main event

2. Secondary Indexing on Key Attributes (e.g., Player or Team)

- **Explanation**: Fans and analysts often focus on specific players or teams. A secondary index allows for quick lookups based on player names or team names.
- **Structure**:
  - **Player ID/Team ID**: Foreign key pointing to the player or team involved.
  - **Event ID**: Cross-reference with the main event data.



Field	Type	Description
Player ID	Integer (FK)	Reference to the player data
Team ID	Integer (FK)	Reference to the team data
Event ID	Integer (FK)	Reference to the main event

Conclusion:-

implementing efficient indexing strategies is crucial to handle high-frequency queries during live events. By leveraging primary indexing on time and event type, secondary indexing on players or teams, composite indexing for combined queries, and full-text indexing for unstructured data, the system can quickly retrieve relevant updates and statistics. Additionally, using a time-series database for frequent metrics ensures optimized storage and retrieval for time-dependent data. Together with caching and query rate limiting, these indexing techniques provide a robust foundation for supporting rapid, real-time queries, delivering a seamless experience for users tracking live events. This approach not only enhances query performance but also helps manage system load, ensuring stability and responsiveness even under high demand.