

Sports Intelligence: Leveraging SQL Databases for Performance Tracking and Strategic Planning

Harini Murugan
Engineering Science in Data Science
University At Buffalo
Buffalo, New York ,USA
UBID - harinimu - (Team 44)

NithishhKrishna KS
Engineering Science in Data Science
University At Buffalo
Buffalo, New York ,USA
UBID - nithishh - (Team 44)

Abstract—This paper explains the process of designing, implementing, and optimizing queries for a Cricket Statistics Database using PostgreSQL. The database stores cricket-statistics information, including teams, players, match formats, stadiums, matches, and player statistics. The project covers Entity-Relationship (E/R) modeling, schema design, data insertion, and SQL queries (including JOIN, GROUP BY, and Subqueries). The system enables statistical analysis of players and teams, making it valuable for sports analysts, coaches and team managers. In addition, a web application interface is developed to allow users to interact with the database in a more intuitive and visual way using Streamlit.

Index Terms—Cricket Statistics, PostgreSQL, Database Design, SQL Queries, Sports Analytics, Query Optimization, Streamlit.

I. INTRODUCTION

Cricket is one of the most popular sports worldwide, generating a significant amount of statistical data. Traditional spreadsheet-based methods struggle to handle the complex relationships between teams, players, and matches. This paper introduces a Cricket Statistics Database, built using PostgreSQL, that efficiently stores, manages, and retrieves match-related data. The database is designed to answer key cricket-related queries, such as top run-scorers per match, player performance at different stadiums, and team performance trends. We used structured schema design, E/R modeling, and query optimization to ensure efficient data retrieval. To enhance usability, we also developed a web application that allows users to explore and interact with the database through a simple and intuitive interface.

II. PROBLEM STATEMENT

The Cricket Statistics Database is created to aid various participants in the cricket community. Coaches and team managers can make use of the database to monitor player performance, analyze both individual stats and team information, and formulate game strategies based on past data(past patterns or trends). Sports analysts can take advantage of the database to conduct comparative studies between teams, investigate player

trends, and produce insights that impact match predictions and assessments of team performance. Furthermore, effective oversight plays an important role in the management and upkeep of this database. Organizations overseeing sports leagues, such as the ICC and BCCI, are responsible for updating match results, ensuring accurate record-keeping of player statistics info, and maintaining uniform data throughout the seasons.

III. ER - MODEL

A. Entity-Relationship (E/R) Model

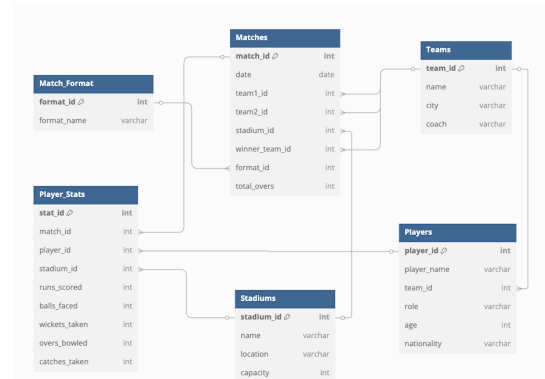


Fig. 1. ER Diagram

The database consists of the following entities: Teams store details about each team, including team_id, name, city, and coach. Players store individual player details, including player_id, player_name, team_id, role, age, and nationality. Stadiums store information about cricket venues, including stadium_id, name, location, and capacity. Match_Format defines different formats of the game, such as T20, ODI, and Test, using format_id and format_name. Matches store details of each match, including match_id, date, team1_id, team2_id, stadium_id, format_id, winner_team_id, and total_overs. Player_Stats store individual player performance in matches, including stat_id, match_id, player_id, stadium_id, runs_scored, balls_faced, wickets_taken, overs_bowled, and catches_taken.

IV. DATABASE ATTRIBUTES AND RELATIONS

A. List of Relations and Their Attributes

The Cricket Statistics Database consists of multiple relations, each designed to store and manage structured cricket-related data efficiently. The main relations and their attributes are:

- **Match_Format** (**format_id**, **format_name**)
- **Teams** (**team_id**, **name**, **city**, **coach**)
- **Players** (**player_id**, **player_name**, **team_id**, **role**, **age**, **nationality**)
- **Stadiums** (**stadium_id**, **name**, **location**, **capacity**)
- **Matches** (**match_id**, **date**, **team1_id**, **team2_id**, **stadium_id**, **winner_team_id**, **format_id**, **total_overs**)
- **Player_Stats** (**stat_id**, **match_id**, **player_id**, **stadium_id**, **runs_scored**, **balls_faced**, **wickets_taken**, **overs_bowled**, **catches_taken**)

Each relation has a primary key that uniquely identifies each row, and relevant foreign keys establish relationships between tables.

B. Primary Keys and Foreign Keys

Primary Keys: Each table has a primary key (PK) that ensures uniqueness.

- **Match_Format:** **format_id** (Unique match format identifier)
- **Teams:** **team_id** (Each team has a unique identifier)
- **Players:** **player_id** (Each player is uniquely identified)
- **Stadiums:** **stadium_id** (Each stadium is uniquely identified)
- **Matches:** **match_id** (Each match must be uniquely identified)
- **Player_Stats:** **stat_id** (Unique identifier for each player's performance entry)

Foreign Keys: Foreign keys (FKs) establish relationships between different tables.

- **Players.team_id** → **Teams.team_id**
- **Matches.team1_id**, **team2_id**, **winner_team_id** → **Teams.team_id**
- **Matches.stadium_id** → **Stadiums.stadium_id**
- **Matches.format_id** → **Match_Format.format_id**
- **Player_Stats.match_id** → **Matches.match_id**
- **Player_Stats.player_id** → **Players.player_id**
- **Player_Stats.stadium_id** → **Stadiums.stadium_id**

C. Detailed Attribute Descriptions

Match_Format Table			
Attribute	Data Type	Purpose	Nullable
format_id	INT (PK)	Match format ID	No
format_name	VARCHAR	Format type (T20, ODI, etc.)	No

TABLE I
MATCH_FORMAT TABLE - ATTRIBUTE DESCRIPTIONS

Teams Table			
Attribute	Data Type	Purpose	Nullable
team_id	INT (PK)	Unique team identifier	No
name	VARCHAR	Team name	No
city	VARCHAR	City where team is based	No
coach	VARCHAR	Team coach	Yes

TABLE II
TEAMS TABLE - ATTRIBUTE DESCRIPTIONS

Players Table			
Attribute	Data Type	Purpose	Nullable
player_id	INT (PK)	Unique player identifier	No
player_name	VARCHAR	Name of the player	No
team_id	INT (FK)	Associated team	No
role	VARCHAR	Player role (Batsman, Bowler)	No
age	INT	Age of the player	No
nationality	VARCHAR	Player's nationality	No

TABLE III
PLAYERS TABLE - ATTRIBUTE DESCRIPTIONS

Stadiums Table			
Attribute	Data Type	Purpose	Nullable
stadium_id	INT (PK)	Unique stadium identifier	No
name	VARCHAR	Stadium name	No
location	VARCHAR	Stadium location	No
capacity	INT	Seating capacity	No

TABLE IV
STADIUMS TABLE - ATTRIBUTE DESCRIPTIONS

Matches Table			
Attribute	Data Type	Purpose	Nullable
match_id	INT (PK)	Unique match identifier	No
date	DATE	Match date	No
team1_id	INT (FK)	First competing team	No
team2_id	INT (FK)	Second competing team	No
stadium_id	INT (FK)	Match stadium	No
winner_team_id	INT (FK)	Winning team	Yes
format_id	INT (FK)	Match format	No
total_overs	INT	Total overs played	No

TABLE V
MATCHES TABLE - ATTRIBUTE DESCRIPTIONS

Player_Stats Table			
Attribute	Data Type	Purpose	Nullable
stat_id	INT (PK)	Player performance ID	No
match_id	INT (FK)	Match reference	No
player_id	INT (FK)	Player reference	No
stadium_id	INT (FK)	Stadium reference	No
runs_scored	INT	Runs scored by player	No
balls_faced	INT	Balls faced by player	No
wickets_taken	INT	Wickets taken by player	No
overs_bowled	INT	Overs bowled	Yes
catches_taken	INT	Catches taken	No

TABLE VI
PLAYER_STATS TABLE - ATTRIBUTE DESCRIPTIONS

D. Foreign Key Actions on Deletion

To maintain referential integrity, specific actions are taken when referenced primary keys are deleted:

- **Players.team_id** → **Teams.team_id**: *SET NULL* (Players remain, but their team is removed)
- **Matches.team1_id, team2_id** → **Teams.team_id**: *CASCADE* (If a team is removed, its matches are deleted)
- **Matches.stadium_id** → **Stadiums.stadium_id**: *SET NULL* (Match exists but without a stadium reference)
- **Player_Stats.match_id** → **Matches.match_id**: *CASCADE* (If a match is removed, all related player stats are removed)
- **Player_Stats.player_id** → **Players.player_id**: *CASCADE* (If a player is deleted, all their stats are deleted)

This ensures data integrity and prevents orphaned records in the database.

V. USERS AND ADMINISTRATORS OF THE DATABASE

This Cricket Statistics Database is designed to work for multiple stakeholders within the cricketing system. Coaches and team managers will use the system to track player performance, refine team strategies, and assess individual player contributions. Sports analysts will utilize it to analyze trends, compare team performances, and generate insights that inform match strategies and player selection. On the administrative side, the sports league organizations such as the ICC and BCCI will look over the updates to match results, ensuring that player statistics are recorded properly. Data analysts will play an important role in data cleaning and processing, anomaly(irregularities) detection, and generating comprehensive reports that aid decision-making for stakeholders in the team.

VI. IMPORTANCE OF A DATABASE OVER EXCEL

Excel spreadsheets are limited when dealing with large datasets and complex relationships. They lack high level of data integrity while performing tasks, making them prone to human errors such as duplicate entries, incorrect formulas, and accidental deletions. Excel also struggles with multi-user access, making collaboration not very efficient. SQL databases provide structured storage with relational integrity, ensuring data accuracy and efficient query execution [1]. SQL queries allow users to retrieve, manipulate, and analyze data efficiently and unlike Excel, databases can handle large volumes of historical and real-time data without performance degradation.

VII. DATA POPULATION AND IMPORT

The dataset was generated using real-life cricket player names, team details, and stadium information. A single CSV

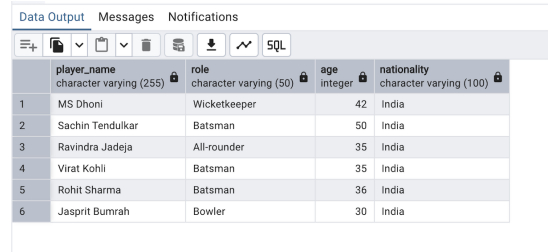
file was created containing all relevant cricket statistics. This file was loaded into a temporary staging table in PostgreSQL using the COPY command for efficient bulk import. Structured SQL queries were then used to populate the normalized relational tables from this staging table, preserving the relationships between players, matches, teams, and stadiums. Data validation and deduplication were applied during this transformation phase to maintain consistency and eliminate redundancy. This method ensured seamless integration of cricket-related data into the database and optimized query performance for analytical purposes.

VIII. QUERY EXECUTION AND OPTIMIZATION

Query performance in PostgreSQL can be enhanced through effective execution plans and indexing strategies, as analyzed in previous research [4][5]. In this section, we detail the execution and optimization of various SQL queries designed to extract meaningful insights from the Cricket Statistics Database. The queries are categorized into simple and complex types, each serving distinct analytical purposes.

1. Select Distinct(player_name), role, age, nationality From Players where team_id = 1;

The query retrieves distinct player names, roles, ages, and nationalities from the Players table where team_id = 1, which corresponds to India. It ensures that only players from Team India are selected, listing their details without duplicate names. The result includes six Indian players with their respective roles and ages



	player_name character varying (255)	role character varying (50)	age integer	nationality character varying (100)
1	MS Dhoni	Wicketkeeper	42	India
2	Sachin Tendulkar	Batsman	50	India
3	Ravindra Jadeja	All-rounder	35	India
4	Virat Kohli	Batsman	35	India
5	Rohit Sharma	Batsman	36	India
6	Jasprit Bumrah	Bowler	30	India

Fig. 2. Query 1 - Output

2. The query retrieves details of T20 matches from the Matches table, including the match ID, date, team names, stadium name, and format. It joins the Teams table twice to fetch team1 and team2 names, the Stadiums table to get the stadium name, and the Match_Format table to filter only T20 matches. The result displays 142 T20 matches, listing teams, venues, and match dates.

Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X Cricket_Records/postgres@PostgreSQL

Cricket_Records/postgres@PostgreSQL

Query Query History

```

1 SELECT Matches.match_id, Matches.date, Teams1.name AS team1, Teams2.name AS team2,
2 FROM Matches
3 JOIN Teams AS Teams1 ON Matches.team1_id = Teams1.team_id
4 JOIN Teams AS Teams2 ON Matches.team2_id = Teams2.team_id
5 JOIN Stadiums ON Matches.stadium_id = Stadiums.stadium_id
6 JOIN Match_Format ON Matches.format_id = Match_Format.format_id
7 WHERE Match_Format.format_name = 'T20';

```

Data Output Messages Notifications

match_id	date	team1	team2	stadium	format_name
integer	date	character varying (255)	character varying (255)	character varying (255)	character varying (20)
1	18	2025-02-21	England	South Africa	Newlands, Cape Town
2	99	2022-03-17	Australia	South Africa	Gaddafi Stadium, Lahore
3	456	2021-11-23	New Zealand	South Africa	Kabul International Cricket Stadium, Kabul
4	469	2022-04-04	Afghanistan	South Africa	Newlands, Cape Town
5	191	2023-12-26	Pakistan	Australia	Wankhede Stadium, Mumbai
6	144	2022-03-26	England	Bangladesh	Gaddafi Stadium, Lahore
7	233	2024-06-12	New Zealand	Afghanistan	Lord's Cricket Ground, London
8	449	2024-05-16	West Indies	Bangladesh	R. Premadasa Stadium, Colombo
9	239	2021-03-21	New Zealand	Sri Lanka	Gaddafi Stadium, Lahore
10	353	2021-07-16	New Zealand	West Indies	Gaddafi Stadium, Lahore
11	420	2022-11-18	England	South Africa	Gaddafi Stadium, Lahore
12	321	2024-01-29	Sri Lanka	Afghanistan	Wankhede Stadium, Mumbai
13	170	2021-04-29	India	Bangladesh	Newlands, Cape Town
14	36	2022-11-05	Pakistan	Australia	R. Premadasa Stadium, Colombo

Fig. 3. Query 2 - Output

3. This SQL query calculates the total runs scored by each player and associates them with their respective teams. It retrieves data from Player_Stats, Players, and Teams tables using JOIN operations. The SUM() function aggregates the runs_scored column, grouping results by player name and team. The ORDER BY total_runs DESC clause ensures players are ranked in descending order based on total runs. This query helps analyze player performance and identify top scorers efficiently.

Cricket_Records/postgres@PostgreSQL

</

Fig. 4. Query 3 - Output

4. This SQL query retrieves the highest run-scorer for each match by selecting the player with the maximum runs in every game. It joins the Player_Stats table with Players to get player names and match details. A subquery finds the highest runs scored in each match, and the WHERE condition ensures only those players are selected. The results are then sorted by match_id to display top scorers in order. This query helps in identifying standout performances in individual matches efficiently.

Cricket_Records/postgres@PostgreSQL

Fig. 5. Query 4 - Output

5. This query inserts a new record into the Players table by specifying values for each of its attributes: player_id, player_name, team_id, role, age, and nationality. The purpose of this operation is to add a new player to the database, associating them with an existing team and defining their playing role and demographic details. This type of query is essential for maintaining an up-to-date roster of players and allows administrators or data analysts to extend the dataset as new players join teams or participate in matches.

Query Query History

```

1 INSERT INTO Players (player_id, player_name, team_id, role, age, nationality)
2 VALUES (10000, 'Demo Player', 1, 'All-rounder', 38, 'India');

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 63 msec.

Fig. 6. Query 5 - Output

6. This query removes a specific record from the Players table where the player_id is equal to 999. The DELETE statement is used to permanently eliminate data from a table based on a given condition. In this case, it ensures that a particular player—identified uniquely by their player_id—is removed from the database. This operation is commonly used during data cleanup, or simulating player retirements or transfers. It helps maintain data integrity by allowing administrators to manage and update the dataset as needed.

Query	Query History
1	DELETE FROM Players WHERE player_id = 10000;
Data Output	Messages Notifications
DELETE 1	
Query returned successfully in 68 msec.	

Fig. 7. Query 6 - Output

7. This query updates the role of a player named "Virat Kohli" in the Players table, setting it to 'All Rounder'. The UPDATE statement is used to modify existing records in a table, and in this case, it targets rows where the player_name matches a specific value. This type of query is useful when a player's role changes over time—such as transitioning from an batsman to a specialist all-rounder or to correct existing role data. It ensures the database remains accurate and reflects current player profiles for reliable analysis and reporting.

Query	Query History
1	UPDATE Players
2	SET role = 'All Rounder'
3	WHERE player_name = 'Virat Kohli';
4	
Data Output	Messages Notifications
UPDATE 19	
Query returned successfully in 58 msec.	

Fig. 8. Query 7 - Output

8. This query modifies the capacity value of a stadium named "Narendra Modi Stadium" in the Stadiums table, setting it to 100000. The purpose of this operation is to update the number of spectators that the stadium can accommodate, possibly due to renovation, expansion, or correction of an earlier entry. Using the UPDATE statement in this way helps ensure that infrastructure-related details are kept current in the database, which is essential for event planning, match allocation, and facility analysis.

Query	Query History
1	UPDATE Stadiums
2	SET capacity = 100000
3	WHERE name = 'Narendra Modi Stadium';
4	
Data Output	Messages Notifications
UPDATE 0	
Query returned successfully in 53 msec.	

Fig. 9. Query 8 - Output

IX. QUERY EXECUTION ANALYSIS

1. This query joins Player_Stats with Players and computes the top five run-scorers by summing runs_scored. According to the EXPLAIN ANALYZE visual output, the query performs a sequential scan on the Player_Stats table and uses a hash join to match players. The aggregation and sorting operations contribute significantly to the overall cost. To improve performance, indexing player_id in the Player_Stats table is recommended. For frequent use cases, this query could also benefit from a materialized view that pre-aggregates total runs..

```
SELECT p.player_name, SUM(s.runs_scored) AS total_runs
FROM Player_Stats s
JOIN Players p ON s.player_id = p.player_id
GROUP BY p.player_name
ORDER BY total_runs DESC
LIMIT 5;
```

#	Node
1.	→ Limit
2.	→ Sort
3.	→ Aggregate
4.	→ Nested Loop Inner Join
5.	→ Seq Scan on player_stats as s
6.	→ Memoize
7.	→ Index Scan using players_pkey on players as p Index Cond: (player_id = s.player_id)

Fig. 10. EA - visual output

2. This query retrieves player names from the Players table for individuals who have scored more than 50 runs in at least one match, using a correlated subquery with a WHERE EXISTS clause. According to the execution plan, the database performs a sequential scan on both the Players and Player_Stats tables, with a filter applied on runs_scored > 50. The planner then uses a hash inner join to match player_id from both tables. While functionally accurate, the presence of sequential scans on large tables can lead to performance bottlenecks. This could be improved by adding indexes on the player_id and runs_scored columns in the Player_Stats table, or by rewriting the query using a JOIN with DISTINCT to reduce overhead from correlated subquery evaluation

```
SELECT p.player_name, SUM(s.runs_scored) AS total_runs
FROM Player_Stats s
JOIN Players p ON s.player_id = p.player_id
GROUP BY p.player_name
ORDER BY total_runs DESC
LIMIT 5;
```

Graphical	Analysis	Statistics
#	Node	
1.	→ Limit	
2.	→ Sort	
3.	→ Aggregate	
4.	→ Nested Loop Inner Join	
5.	→ Seq Scan on player_stats as s	
6.	→ Memoize	
7.	→ Index Scan using players_pkey on players as p Index Cond: (player_id = s.player_id)	

Fig. 11. EA - visual output

3. This query retrieves match IDs along with the names of the two teams by joining the Matches table with the Teams table twice. The execution plan shows sequential scans and hash joins, which can be expensive on large datasets. Adding indexes on team1_id, team2_id, and team_id can significantly improve performance by reducing scan and join costs.

```
SELECT m.match_id, t1.name AS team1, t2.name AS team2
FROM Matches m
JOIN Teams t1 ON m.team1_id = t1.team_id
JOIN Teams t2 ON m.team2_id = t2.team_id;
```

Graphical	Analysis	Statistics
#	Node	
1.	→ Hash Inner Join Hash Cond: (m.team2_id = t2.team_id)	
2.	→ Hash Inner Join Hash Cond: (m.team1_id = t1.team_id)	
3.	→ Seq Scan on matches as m	
4.	→ Hash	
5.	→ Seq Scan on teams as t1	
6.	→ Hash	
7.	→ Seq Scan on teams as t2	

Fig. 12. EA - visual output

X. WEBSITE DEPLOYMENT

To enhance the usability and accessibility of the Cricket Statistics Database, a web-based interface was developed using Streamlit. This application connects directly to the PostgreSQL database and allows users to interact with the data through a clean and dynamic interface. The app supports multiple predefined queries, such as viewing top run-scorers, match summaries, player searches, and individual performance statistics. Each result is rendered as an interactive table, making it easier for users to explore key insights without needing to write SQL manually. The website serves as a user-friendly bridge between the database and end-users like analysts, coaches, and administrators

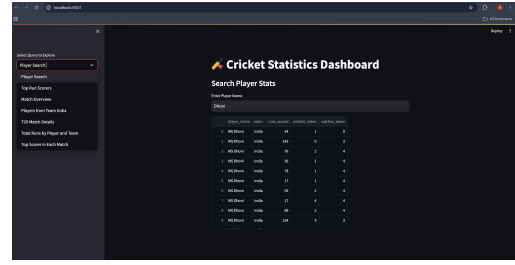


Fig. 13. Web-App using Streamlit

Web App link

XI. CONCLUSION AND FUTURE WORK

This paper demonstrated the design and implementation of a Cricket Statistics Database using PostgreSQL. The system effectively manages team and player statistics, supports complex SQL queries, and provides optimized retrieval of match statistics. The database enhances accessibility, efficiency, and reliability in handling large-scale cricket-related data, ensuring structured and accurate storage of match records, player performances, and team histories. It enables data-driven decision-making for coaches, analysts, and sports organizations, helping them track player performance, formulate strategies, and conduct comparative analysis based on historical data.

Future work includes the integration of a web-based front-end for user-friendly data visualization and interactive dashboards, enabling seamless exploration of cricket statistics. Additionally, incorporating machine learning models for predictive analytics can provide insights into player performance trends, team strategies, and match outcomes. Real-time data integration from external cricket APIs will enhance the database's functionality by ensuring live updates on matches, player statistics, and rankings. Furthermore, extending the system to support multi-dimensional data analysis, performance heatmaps, and anomaly detection can add significant value to cricket analytics, making it a comprehensive tool for sports professionals, analysts, and enthusiasts.

REFERENCES

- [1] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., & Neumann, T. (2020). How Good Are Query Optimizers, Really? Proceedings of the VLDB Endowment.
- [2] Stonebraker, M., & Rowe, L. A. (2021). The Design of POSTGRES. ACM SIGMOD International Conference on Management of Data.
- [3] Chaudhuri, S. (2022). An Overview of Query Optimization in Relational Systems. ACM SIGMOD Conference on Management of Data.
- [4] Jungmair, M., & Kemper, A. (2023). Designing an Open Framework for Query Optimization and Compilation. ACM SIGMOD Conference.
- [5] Abhayanand, & Rahman, M. M. (2024). Enhancing Query Optimization in Distributed Relational Databases: A Comprehensive Review. International Journal of Novel Research and Development.
- [6] Kim, J., & Park, H. (2023). Performance Analysis of SQL Query Optimization Techniques in Modern Relational Databases. Journal of Database Management.

- [7] Singh, A., & Verma, R. (2022). Advanced Indexing Strategies for Efficient Query Processing. International Conference on Data Science and Applications.
- [8] Wang, Y., & Zhang, L. (2021). Evaluating Query Execution Plans in PostgreSQL: A Case Study. Proceedings of the International Conference on Database Systems.