Database Project

Aarsh Walavalkar	**220013**	**aarshaw22@iitk.ac.in**
Harshit Srivastava	**220444**	**harshits22@iitk.ac.in**
Nilay Agarwal	**220714**	**nilayag22@iitk.ac.in**
Purav Jangir	**220837**	**puravj22@iitk.ac.in**
Tanush Goel	**221131**	**tanushg22@iitk.ac.in**

CS315 Final Project

Abstract

This project presents an **IPL Query Optimization System** that enables users to query IPL data from 2008 to 2024. The system provides a menu-driven interface to execute analytical queries. It supports operations like player statistics, team head-to-head analysis, venue-based performance metrics, and seasonal award predictions. To facilitate optimized query processing, we transformed raw CSVs into normalized relational tables, following 3NF and BCNF principles.

1 Motivation and Problem Statement

The Indian Premier League (IPL), with its extensive historical data from 2008 to 2024, presents a valuable opportunity for structured analysis. However, the raw CSV format of the data hinders efficient querying and exploration. This project addresses the need for a query-optimized system by transforming unstructured IPL data into a set of BCNF-normalized relational tables. A terminal-based interface is developed to enable users to execute complex SQL queries and derive insights interactively. The system showcases the practical application of database design, normalization, and performance-optimized query execution using real-world sports data.

2 Methodology

The raw IPL data from Kaggle, comprising `ball-by-ball.csv` and `matches.csv`, is first cleaned and preprocessed. Functional dependencies are identified to decompose the data into BCNF-compliant relational tables with clearly defined primary and foreign keys. A terminal-based interface is built to support dynamic query execution, enabling users to retrieve match-wise, player-wise, and team-wise statistics efficiently. Query performance is enhanced through indexing and query optimization techniques.

2.1 Initial Datasets

These files include redundant and repetitive data not suitable for efficient querying or relational design.

ball_by_ball.csv

Attribute	Description
match_id	Match reference ID
inning	Inning number (1 or 2)
batting_team	Team batting during delivery
bowling_team	Team bowling during delivery
over, ball	Ball position within the over
batter, bowler, non_striker	Player names involved
batsman_runs	Runs scored off the bat
extra_runs	Extra runs (wide, no-ball, etc.)
total_runs	Total runs including extras
extras_type	Type of extra (if any)
is_wicket	1 if dismissal occurred, else 0
player_dismissed	Name of dismissed player
dismissal_kind	Mode of dismissal (e.g., caught, bowled)
fielder	Fielder involved in dismissal (if any)

Table 1: Attributes in `ball_by_ball.csv`

matches.csv

Attribute	Description
id	Unique match ID
season	IPL season/year
city	Match location
date	Match date
match_type	Match classification (e.g., League, Final)
player_of_match	Star performer (MVP)
venue	Stadium name
team1, team2	Participating teams
toss_winner	Team that won the toss
toss_decision	Toss decision (bat/field)
winner	Match winner
result	Result type (e.g., normal, tie, no result)
result_margin	Margin of victory
target_runs, target_overs	Chasing target values (if applicable)
super_over	Indicator if super over was played
method	Method of win (e.g., DLS)
umpire1, umpire2	On-field umpire names

Table 2: Attributes in `matches.csv`

2.2 Normalized CSVs

The raw IPL datasets were transformed into a set of relational tables, each designed to satisfy Boyce-Codd Normal Form (**BCNF**). Below are the normalized relations along with their keys and normalization proofs:

- ***teams***(team_id, team_name)
- ***players***(player_id, player_name)
- ***venue***(venue_id, venue_name, city)
- ***umpire***(umpire_id, umpire_name)
- ***match***(match_id, season, date, match_type, venue_id)
- ***match_teams***(match_id, team_id1, team_id2)
- ***toss***(match_id, toss_winner_id, toss_decision)
- ***match_result***(match_id, winner_id, result, result_margin, target_runs, target_overs, super_over, method)
- ***umpire_match***(match_id, umpire1_id, umpire2_id)
- ***player_of_match***(match_id, player_id)
- ***delivery***(match_id, inning, over, ball, batter_id, bowler_id, non_striker_id, batting_team_id, bowling_team_id, batsman_runs, extra_runs, total_runs)
- ***dismissals***(match_id, inning, over, ball, dismissal_kind, fielder_id)

1. Team

Attributes: *team_id*, *team_name*
FDs: *team_id* $\rightarrow$ *team_name*
Justification: Each team has a unique identifier (*team_id*), which determines the team name. No partial or transitive dependencies exist. Hence, the relation is in BCNF.

2. Player

Attributes: *player_id*, *player_name*
FDs: *player_id* $\rightarrow$ *player_name*
Justification: The player ID uniquely determines the name. Since the determinant is a candidate key, BCNF is satisfied.

3. Venue

Attributes: *venue_id*, *venue_name*, *city*
FDs: *venue_id* $\rightarrow$ *venue_name*, *city*
Justification: *venue_id* uniquely determines both *venue_name* and *city*. As the only non-trivial dependency is on the key, the relation is in BCNF.

4. Umpire

Attributes: *umpire_id*, *umpire_name*
FDs: *umpire_id* → *umpire_name*
Justification: Unique *umpire_id* fully determines the name. No violation of BCNF.

5. Match

Attributes: *match_id*, *season*, *date*, *match_type*, *venue_id*
FDs: *match_id* → all other attributes
Justification: Each match ID uniquely determines all other attributes. Since the determinant is the key, BCNF holds.

6. Match_Teams

Attributes: *match_id*, *team_id1*, *team_id2*
FDs: (*match_id* → (team_id1, team_id2) – (no transitive dependencies)
Justification: The composite key (*match_id*, *team_id*) has no partial or transitive dependencies. Fully normalized.

7. Toss

Attributes: *match_id*, *toss_winner_id*, *toss_decision*
FDs: *match_id* → *toss_winner_id*, *toss_decision*
Justification: Toss details are determined per match. No partial or transitive dependencies. Satisfies BCNF.

8. Match_Result

Attributes: *match_id*, *winner_id*, *result*, *result_margin*, *target_runs*, *target_overs*, *super_over*, *method*
FDs: *match_id* → all other attributes
Justification: Each match has exactly one result. All result-related info is functionally dependent on *match_id*. BCNF holds.

9. Umpire_Match

Attributes: *match_id*, *umpire1_id*, *umpire2_id*
FDs: *match_id* → *umpire1_id*, *umpire2_id*
Justification: A match is officiated by a fixed pair of umpires. Both IDs are functionally dependent on *match_id*. BCNF satisfied.

10. Player_Of_Match

Attributes: *match_id*, *player_id*
FDs: *match_id* → *player_id*
Justification: Every match has exactly one Player of the Match. No other dependency exists. BCNF satisfied.

11. Delivery

Attributes: *match_id*, *inning*, *over*, *ball*, *batter_id*, *bowler_id*, *non_striker_id*, *batting_team_id*, *bowling_team_id*, *batsman_runs*, *extra_runs*, *total_runs*
FDs: (*match_id*, *inning*, *over*, *ball*) $\rightarrow$ all other attributes
Justification: Each delivery is uniquely identified by the composite key. No other functional dependency violates BCNF.

12. Dismissals

Attributes: *match_id*, *inning*, *over*, *ball*, *dismissal_kind*, *fielder_id*
FDs: (*match_id*, *inning*, *over*, *ball*) $\rightarrow$ all other attributes
Justification: Dismissals are logged per ball. All attributes are fully dependent on the composite key. BCNF holds.

3 Implementation and Results

The database is built using BCNF-normalized tables loaded from preprocessed IPL CSVs. A Python-based terminal interface executes structured queries over this schema. We present a set of representative queries along with their relational logic and measured processing times. This demonstrates the system's ability to support efficient, real-time exploration of IPL data through optimized query execution.

3.1 Query Definitions

Query No.	Description
1–4	Projection: list of all teams, players, umpires, and venues
5	Match summary using multiple joins
6	Individual player stats (runs, wickets, catches, matches)
7	Player of the Match stats
8	Head-to-head stats between two teams
9	Team stats: wins, losses, no-results, abandoned
10	Player's team history across seasons
11	Venue-wise win stats (bat first vs. second)
12	Venue-wise average 1st innings score
13	Total number of 4s and 6s per season
14	Avg. powerplay score for a team per season
15	Avg. powerplay wickets for a team per season
16	Season summary: Orange/Purple Cap, most 4s/6s, winner, runner-up
17	Umpire match count
18	Most fifties and hundreds per season
19	Most 5-wicket hauls per season
20	Bowler vs Batter performance
21–22	Update team/venue name in CSV

Table 3: List of Queries Implemented

3.2 Relational Logic

Query 1–4: Entity Projection Queries

Objective: List all records of a type (e.g., all players, teams, venues).
Relational Algebra:

$$\pi_{\text{team\_name}}(Team), \quad \pi_{\text{player\_name}}(Player), \quad \pi_{\text{umpire\_name}}(Umpire), \quad \pi_{\text{venue\_name}}(Venue)$$

Processing Time:

1. 0.0155 sec
2. 0.0860 sec
3. 0.0138 sec
4. 0.0240 sec

Query 5: Match Summary

Objective: Display full match details by joining multiple tables.
Relational Algebra:

$$\pi_{\substack{\text{match\_id, season, date, venue,}\\ \text{team1, team2, umpire1, umpire2,}\\ \text{winner, pom}}} \left(\begin{array}{l} \text{Match} \bowtie \text{Venue} \bowtie \text{MatchTeams} \bowtie \text{Teams} \\ \bowtie \text{UmpireMatch} \bowtie \text{Umpire} \\ \bowtie \text{MatchResult} \bowtie \text{PlayerOfMatch} \bowtie \text{Player} \end{array} \right)$$

Processing Time: 0.3114 sec

Query 6: Player Stats

Objective: Calculate a player's cumulative runs, wickets, catches, and total matches.
Relational Algebra:

$$\gamma_{\substack{\text{SUM(batsman\_runs),}\\ \text{COUNT(wickets),}\\ \text{COUNT(catches)}}} \left(\begin{array}{l} \sigma_{\text{batter\_id=pid}}(\text{Delivery}) \\ \cup\ \sigma_{\text{bowler\_id=pid}}(\text{Delivery}) \\ \cup\ \sigma_{\text{fielder\_id=pid} \wedge \text{dismissal\_kind='caught'}}(\text{Dismissals}) \end{array} \right)$$

Processing Time: 0.0809 sec

Query 7: Player of the Match Records

Objective: List all matches where a selected player was awarded Player of the Match.
Relational Algebra:

$$\sigma_{player\_id=pid}(PlayerOfMatch) \bowtie Match \bowtie Venue$$

Processing Time: 0.0154 sec

Query 8: Head-to-Head Between Two Teams

Objective: Compare two teams across wins, toss wins, and matches played.
Relational Algebra:

$$\gamma_{\begin{array}{l}\text{team\_id,}\\ \text{COUNT(wins),}\\ \text{COUNT(tosses)}\end{array}} \left(\begin{array}{ll} \sigma & (\text{team1} = \text{T1} \wedge \text{team2} = \text{T2}) \\ & \vee \\ & (\text{team1} = \text{T2} \wedge \text{team2} = \text{T1}) \end{array} \quad (\text{MatchTeams} \bowtie \text{MatchResult} \bowtie \text{Toss}) \right)$$

Processing Time: 0.0067 sec

Query 9: Team-Wise Match Outcome Stats

Objective: Compute matches won, lost, no result, abandoned by a team.
Relational Algebra:

$$\gamma_{team\_id, COUNT(win), COUNT(loss), COUNT(no\_result)}(MatchTeams \bowtie MatchResult)$$

Processing Time: 0.0035 sec

Query 10: Player's Team History

Objective: Determine teams a player played for in each season.
Relational Algebra:

$$\pi_{season, team\_id}(\sigma_{batter\_id=pid \vee bowler\_id=pid \vee fielder\_id=pid}(Delivery \bowtie Match))$$

Processing Time: 0.1037 sec

Query 11: Venue Win Stats (Bat First vs Second)

Objective: Count matches won at a venue when batting first vs second.
Relational Algebra:

$$\gamma_{venue\_id, COUNT(bat\_first\_wins), COUNT(chase\_wins)}(Match \bowtie MatchResult \bowtie Toss)$$

Processing Time: 0.0085 sec

Query 12: Venue-Wise 1st Innings Average

Objective: Compute average first innings score for a venue.
Relational Algebra:

$$\gamma_{venue\_id, AVG(total\_runs)}(\sigma_{inning=1}(Delivery \bowtie Match))$$

Processing Time: 0.0184 sec

Query 13: 4s and 6s Per Season

Objective: Count the total 4s and 6s hit each season.
Relational Algebra:

$$\gamma_{season,SUM(fours),SUM(sixes)}\ (\sigma_{batsman\_runs=4 \vee batsman\_runs=6}(Delivery \bowtie Match))$$

Processing Time: 0.0814 sec

Query 14: Avg. Powerplay Score per Season

Objective: Calculate average score in powerplay overs (0–5).
Relational Algebra:

$$\gamma_{season,AVG(total\_runs)}(\sigma_{over<6 \wedge batting\_team\_id=tid}(Delivery \bowtie Match))$$

Processing Time: 0.0736 sec

Query 15: Avg. Powerplay Wickets per Season

Objective: Calculate average wickets taken in powerplay per season.
Relational Algebra:

$$\gamma_{season,AVG(wickets)}(\sigma_{over<6 \wedge bowling\_team\_id=tid \wedge is\_wicket=1}(Delivery \bowtie Match \bowtie Dismissals))$$

Processing Time: 0.1153 sec

Query 16: Season Awards Summary

Objective: Identify top players (Orange Cap, Purple Cap, 4s, 6s) and winner/runner-up for each season.
Relational Algebra (Conceptual):

$$\gamma_{\substack{\text{player\_id},\ \text{season},\\ \text{MAX(runs)},\\ \text{MAX(wickets)},\\ \text{MAX(fours)},\\ \text{MAX(sixes)}}}\ (\text{Delivery})$$

Processing Time: 5.80 sec

Query 17: Matches Judged by Umpire

Objective: Count matches officiated by a specific umpire.
Relational Algebra:

$$COUNT(*)(\sigma_{umpire\_id=uid}(UmpireMatch))$$

Processing Time: 0.0079 sec

Query 18: Most 50s and 100s per Season

Objective: Count fifties and hundreds scored by each player per season.
Relational Algebra:

$$\gamma_{season,player\_id,COUNT(fifty),COUNT(hundred)}(GroupBy_{batter\_id,season,match\_id}(Delivery) \text{ where } runs \geq 50)$$

Processing Time: 0.2560 sec

Query 19: 5-Wicket Hauls per Season

Objective: Identify bowlers with most 5-wicket hauls per season.
Relational Algebra:

$$\gamma_{season,bowler\_id,COUNT(*)}(GroupBy_{match\_id}(\sigma_{is\_wicket=1}(Delivery)) \text{ where wickets } \geq 5)$$

Processing Time: 0.2853 sec

Query 20: Bowler vs Batter Head-to-Head

Objective: Compare a specific bowler's performance against a specific batter.
Relational Algebra:

$$\sigma_{bowler\_id=bid \wedge batter\_id=pid}(Delivery \bowtie Dismissals) \rightarrow \text{Aggregate stats: runs, wickets, dot balls}$$

Processing Time: 0.0164 sec

4 Code Link

- Link to code encoded in:
  `https://drive.google.com/drive/folders/1MFMWhqg40gzDWWBUBGOc3S-HblGTUm5S?usp=sharing`.

5 Discussion and Limitations

The system effectively supports a wide range of analytical queries, from basic projections to complex multi-table joins, enabling granular insights such as player performance, head-to-head comparisons, and venue-specific trends. Query response times were consistently low due to normalization and indexing, validating the schema design and optimization strategies.

However, some limitations persist. The static nature of CSV-based data restricts real-time updates, requiring reprocessing for every data addition. Certain derived statistics (e.g., Orange/Purple Cap) required extensive aggregation and conditional logic, which could be optimized further using materialized views or triggers. Additionally, external factors like weather or toss decisions, which influence match outcomes, were not captured due to dataset constraints. Future extensions could include integration with APIs for live updates and support for advanced analytical queries using views or stored procedures.

Contributions

All five team members actively contributed to data preprocessing, schema design, implementation, and interface development. Tasks were collaboratively discussed and iterated upon throughout the project. Additionally, each member took primary ownership of a specific component:

220013 led the extraction, preprocessing, cleaning of raw CSV data, and database schema design.
220444 was primarily responsible for database schema design and normalization.
220714 focused on writing and optimizing queries.
220837 handled query optimization, performance analysis, and report compilation.
221131 developed the terminal-based user interface for query execution.