Cricket DB - IPL

Hemanth Vihari Kothapalli
Venkata Sai Nikhil Thodupunuri
Brian Heath
Chris Niemeyer

Abstract

As a sport that began in the late 16th century that has many unique rules and terminology, cricket offers good statistical information that lends itself to a compelling and dynamic database project.

Cricket datasets have many records, for example if we record each ball thrown then every match has at least 300 entries and over all there are 600 matches across 10 seasons of IPL. Aggregation queries on such datasets might be complex and need to be optimised.

Our platform provides Player, Team and Trivia portals for digging into cricket facts and statistics as well as a team build fantasy simulation tournament. Our Cricket DB is working on bringing the best possible insights into knowing and understanding the game better in an exciting way, as well as presenting our platform efficiently by utilizing database performance enhancements on the backend.

Introduction and project goals

We wanted to find an ongoing topic that has good robust existing datasets, be useful and fun to our portal users, and be compelling to all our team members. For our two Indian team members, cricket is practically the national sport in terms of popularity and offers intrigue to them as obligatory life long fans. For our remaining two American team members, it is a great opportunity to learn about a popular international sport as well as gain insight into a very prevalent aspect of Indian culture along with our teammates.

Coaches, managers, team captains, and fans get input from data analysts to understand the situation and game better. We'd like to help organize these data and insights, and make it fun to learn more about cricket.

Our study looks at the Indian Premier League (IPL) which consists of eight teams from cities around India. The IPL in recent years has ranked top 6 for most attended sports worldwide. There are generally 56 regular season games and 4 playoff games. There is a lot of scope for analysis, all the way from decision making to player analytics.

Our portal users will login with a password so that they can save information about their favorite player. This information can later be used to help generate a fantasy team for simulation games. Otherwise, the portal provides granular statistics and random fun facts about players and teams as well as general statistical trivia. Finally, we want visitors to use our portal to have fun, leveraging our algorithm to create a fantasy team. Players are added to a fan's roster and then evaluated as a group.

Data sources and technologies used

Our group chose to combine three data sources (*Appendix D for web links*). First, the Cric sheet dataset consists of 746 IPL matches structured ball-to-ball which is particularly useful for some of our more complex trivia queries. From this data

source, we crawled for player data and history, creating blob objects in HTML, parsed using Beautiful Soup. Secondly, we use IPL Cricket Data from Kaggle. And finally, we scraped ESPN data for additional profiles, player history and team images.

Our web app is a Flask Application, a web framework written in Python and that runs within a Python2 environment. We store our data in two databases. For basic player data, match information, and team information, we use MySQL (using MySQL-Connector hosted on AWS) since the relations between the three elements are well defined. For example, each match has balls served, those balls are bowled by players, and those players belong to a team. For more complex player data, for which we have differing values based on players according to how long they have played, their number or appearances, etc. we use NoSQL (using MongoDB hosted on EC2). We use HTML/Bootstrap and the "Ace" CSS template to create a professional website.

To assist with caching and optimization, we used Redis to help the performance of our queries, particularly for the "Team Build" aspect of the portal. Redis is a efficient single threaded cache; that uses bloom filter for faster retrievals. "Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster." (Wikipedia) Redis allowed us to only perform our calculations of players' batting and bowling skill one time before an unlimited number of fantasy team simulations.

We also use a Twitter feed on our Dashboard page to show current IPL news and streaming data, keeping our site relevant and contemporary. Our MySQL and Mongo were hosted on AWS/EC2 respectively.

Relational Schema

Player (ID, Name, Short_Name, DOB, Country_Name, Bat_Handedness, Bowl_Skill, Image_Pointer, Additional_Info)

Team (ID, Name, Home, Country)

Player_to_Team (Team_ID, Player_ID, Year)
FK- Team_ID references Team.ID, Player_ID references Player.ID

Match (Match_ID, Team_1, Team_2, Date, Year, Venue, City, Country, Toss_Winner, Match_Winner, Toss_Decision, Win_Type, Outcome_Type, Margin, Player_of_the_Match)
FK- Player_of_the_Match references Player.ID

Innings (Match_ID, Innings_No, Batting_Team, Bowling_Team)

Ball_to_Ball (Match_ID, Over_ID, Ball_ID, Innings_no, striker_ID, Non-striker_ID, Bowler_ID, Batsman_Score, Output_type)

Ball_Extra(Match_ID, Over_ID, Ball_ID, Innings_no, Extra_Type, Extras_runs)

**Note: Image_Pointer, Additional_Info of Player table will be stored in Mongo DB.
**Note: Query latency metrics will be pushed to Mongo.

Description of system architecture

Below are descriptions of our Cricket DB portal service web pages with information about what the respective pages accomplish:

Welcome Page:
You are invited to log into our Cricket DB web app with a username/password. We store credentials so we can also store you favorite team and player.

Dashboard Page:
Our dashboard page makes an eye grabbing first impression providing a variety of information, graphs and random rotating fun facts about IPL cricket. Offering a Twitter feed of recent cricket news keeps our portal up to date and dynamic as well as helps our Users find other userful cricket information on the web. This page also provides a fixture board with past, current, and future matches, a comparison of which teams generate the most runs and which teams generate the most wickets, the highest scoring batsmen at the present time, and finally a chart representing the number of runs scored by match day over the course of each season.

Player Page:
Our player page includes a profile of any player from the IPL. Simply type a name into the search bar and click enter to obtain a variety of statistical information, random facts along with an image of the player. Users can designate the player as their favorite on this page. Biographical information is provided

from the Mongo and MySQL Database and player images come from the Mongo Database. Both batting and bowling statistics are presented in a grid format at the bottom of the page. We also offer random queries about players shown as "Facts", which can be regenerated by refreshing the page.

Team Page:
Our team page presents a profile of a team with a dropdown search for all the available teams and the roster is populated from the SQL Database. An image associated with each team is loaded using the Mongo Database. Again, rotating queries/interesting facts about each team are randomly generated and shown on the center of the page to provide new content on any subsequent visit.

Trivia Page:
Our trivia page provides 8 static and 2 dynamic fun fact and generally presents some of our more complex queries. They are listed below with the first and seventh indicated as dynamic and the specific queries are shown at the end in (*Appendix E*):

- Who is the most successful batsman against a random bowler? (dynamic)
- How has winning the toss affected winning the match?
- Which out of town team has won the most matches in all cities where matches are played?
- Which player has the most player-of-the-match awards in each city?
- Which player has scored the greatest difference of away runs minus home runs?
- Which bowler allows the highest rate of extras in the league?
- Who's whose "bunny"? (Highest number of times a random batsman got out for the same bowler) (dynamic)
- What is the highest score of runs in an over?
- Who took more than 100 wickets in IPL?
- Who scored the most centuries in IPL?

Team Build "Battle" Page:
This page allows the portal user to select/"buy" 11 players as input provided a budget in order to form a team. The algorithm computes a bowling strength and a batting strength for each player and gives the player a rating. The simulation will not allow for repeat selections nor will allow you to start the simulation until a full team within budget is selected.

The simulation then creates a random fixture of matches amongst 8 IPL teams, including the portal User's team. The simulation calculates the team's scores and performance and returns the IPL winner. This enables the User to choose his best from the IPL players.

How we addressed required features

We combined several datasets in a compelling and useful portal, one of which need to be scraped and cleaned.

We generated complicated queries combining information from multiple tables and optimized the database performance by indexing, caching and carefully choosing join orders and selections.

We used complex architecture incorporating many current, popular and relevant technologies: Python(Flask), Angular, AWS/EC2, Mongo (NoSQL), and Redis.

Performance evaluation

Early in our project planning process, we observed in our ER diagram that some data was in 2NF. We manipulated our schema to ensure all our relations are in 3NF, and subsequently there is no redundancy in the tables after redistributing our data.

We also performed an optimization experiment on calculating a "hat-trick", which requires getting data for three consecutive bowls from the largest table, ball-to-ball. A hat-trick in cricket is when a bowler takes three wickets on consecutive deliveries, dismissing three different batsmen. To find the hat-trick, we join the "ballToBall" table with itself.
This table contains the data of all legal deliveries in IPL, which contains ~144,454 rows and is ~17.6 MB in size. The first query in our experiment used a full table scan, which takes extensive time to compute due to the cross product join. Clearly this is not ideal with an exponential blow up of 10^15 rows in the interim table.

Joining the table on successive ball deliveries and then filtering out the table entries on each step of join works a bit slow at ~120ms because our temp table doesn't have an index. With an index on Output type, the time reduces to ~32ms. (*See Appendix B & C*)

Therefore, choosing appropriate join orders with the smaller relation on the outside as well as indexing output type for bowl type offered improvement opportunities.

Furthermore, we improved performance with caching. Since the scores for player and the team remain constant as per the existing database set up, we have leveraged the idea of caching to additionally improve performance for the team build section of our portal.

We implemented this caching feature with Redis for the "Battle Page" where we need the scores of teams and in turn need scores/ratings of players.

Technical Challenges and How They were Overcome

The first challenge we faced was using complicated data sources. While Kaggle was an easy upsert into our production database (MySQL), the rest of our data came from unstructured web sources and required us to recursively scrape the various player profiles that we would need for our player page, roster on our team page, and most importantly our battle page.

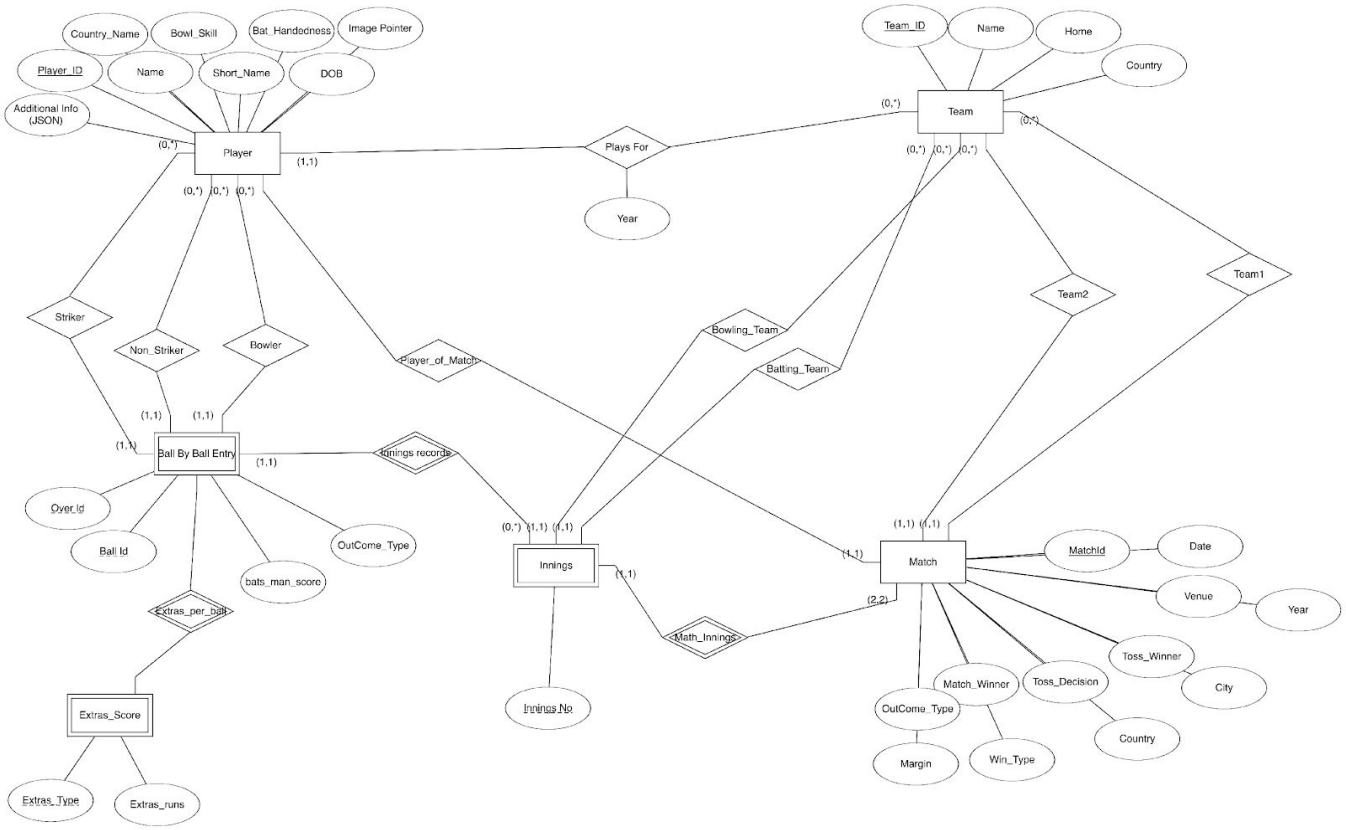
Next, we faced substantial difficulty joining the different datasets. For example, some of the players names formats varied from our different data sources and therefore needed to be cleaned during the scraping process. This required us to make decisions about how best to parse player names so that we could link, for example, one particular player's statistics with their image.

Cricket datasets have many records, every match has at least 300 entries and overall we have 600 matches. Aggregation queries on such datasets can be complex and need to be optimized. Carefully choosing join orders was essential to making our web application responsive.

Extra Credit Features

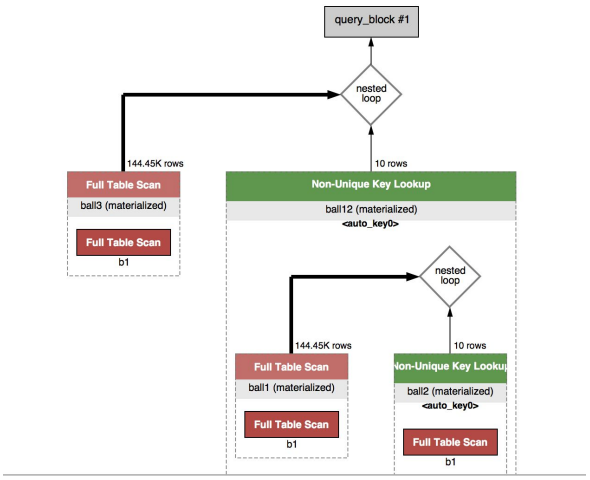
- Mongo DB (NoSQL) hosted with EC2 to store the crawled information because it is not structured.
- SQL for the relational data stored with AWS/RDS.
- Python (Flask) and Angular to provide
- Redis with bloom filter for caching of queries
- Twitter feed to show the current season of the IPL which happens at this time in the spring every year.

Appendix A: *Entity Relationship Diagram*

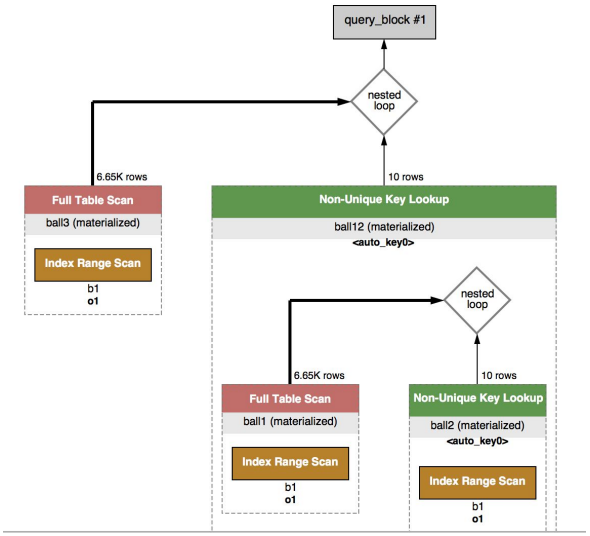


Appendix B

With only primary index, time is 80ms

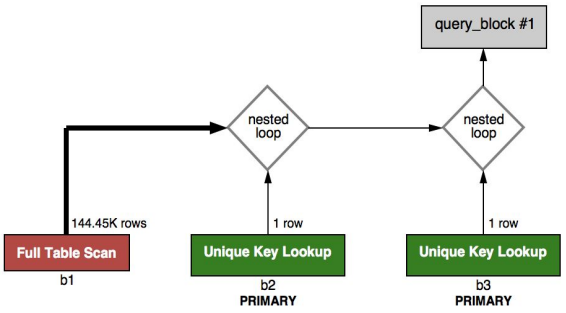


With primary index and Output type index, time is 37ms

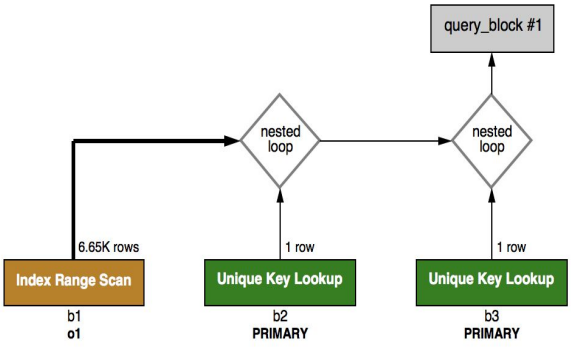


Appendix C

With only primary index, time is 80ms



With primary index and Output type index, time is 37ms



Appendix D

Data Repositories

I. IPL Cricket Data from Kaggle (this will be a direct upsert into MySQL on AWS)
   A. https://www.kaggle.com/raghu07/ipl-datatill-2016
   B. https://www.kaggle.com/raghu07/ipl-data-till-2017

II. Cric Sheets (the will be a direct upsert into MySQL on AWS)
   A. https://cricsheet.org/

III. ESPN Cricket Scraped Data (this will be a direct upsert to MySQL for profiles and match statistics and Mongo for Images)
   A. Player Profiles
   B. Player Stats
   C. Image

Appendix E

SQL Queries

I. Who is the best batsman against each bowler by striker rate.

```
SELECT p1.NAME AS Striker,
       p2.NAME AS Bowler,
       Max(av) AS Average
FROM   (SELECT strikerid,
               bowlerid,
                      Sum(batsmanscore)  /
Count(DISTINCT matchid) AS av
        FROM   ballToBall
        GROUP  BY strikerid,
                  bowlerid) AS a
       LEFT JOIN player p1
              ON strikerid = p1.id
       LEFT JOIN player p2
              ON bowlerid = p2.id
GROUP  BY bowlerid
ORDER  BY Max(av) DESC
```

II. How has winning the toss affected winning the match?

```
SELECT ( m2.c / m1.c ) * 100
FROM   (SELECT Count(*) AS c
        FROM   `match`) m1,
       (SELECT Count(*) AS c
        FROM   `match`
        WHERE  tosswinner = matchwinner) m2;
```

III. Which out-of-town team has won the most matches in all cities where matches are played?

```
SELECT city,
       name,
       Max(wins)
FROM   (SELECT m.city,
               t.name,
               Count(m.matchwinner) AS wins
        FROM   `match` m
               LEFT JOIN team t
                      ON m.matchwinner =
t.id
        WHERE  m.city <> t.home
        GROUP  BY m.city,
                  t.name) AS awayWins
GROUP  BY city;
```

IV. Which player has the most player-of-the-match awards in each city where matches are played?

```
SELECT poms.city,
       poms.name,
       Max(poms.pom)
FROM   (SELECT m.city,
               p.name,
               Count(p.name) AS pom
        FROM   `match` m
               LEFT JOIN player p
                      ON m.playerofthematch
= p.id
        WHERE  city IS NOT NULL
               AND p.name IS NOT NULL
        GROUP  BY m.city,
                  p.name) AS poms
GROUP  BY city;
```

V. Which player has scored the greatest difference of away runs minus home runs?

```
SELECT p.name, Abs(homeScore.score - awayScore.score)
FROM
(SELECT                                    b.strikerid
AS bman,
                Sum(b.batsmanscore) / Count(DISTINCT
b.matchid) AS score
        FROM   ballToBall b
               JOIN `match` m
                 ON b.matchid = m.`matchid`
               JOIN innings
                 ON `innings`.`matchid` = b.matchid
               JOIN team t
                 ON innings.`battingteam` = t.`id`
                    AND t.home = m.`city`
        GROUP  BY b.strikerid) AS homeScore
                     JOIN    (SELECT    b.strikerid
AS bman,
                                Sum(b.batsmanscore)  /
Count(DISTINCT b.matchid) AS score
             FROM   ballToBall b
                    JOIN `match` m
                      ON b.matchid = m.`matchid`
                    JOIN innings
                              ON `innings`.`matchid` =
b.matchid
                    JOIN team t
                           ON innings.`battingteam` =
t.`id`
                       AND t.home <> m.`city`
             GROUP  BY b.strikerid) AS awayScore
         ON homeScore.bman = awayScore.bman
       JOIN player p
         ON p.`id` = homeScore.bman
ORDER    BY  Abs(homeScore.score  -  awayScore.score)
DESC;
```

VI. Which bowler allows the highest number of extras in the league?

```
SELECT p.NAME,
               Sum(bowlerMatch.badcount)   /
Count(bowlerMatch.matchid) AS badAverage
FROM   (SELECT b.bowlerid AS bowlerID,
               b.matchid,
               Count(*)   AS badCount
        FROM   ballToBall b
               JOIN ballExtra be
                 ON be.matchid = b.matchid
                    AND be.overid = b.overid
                    AND be.ballid = b.ballid
                          AND be.inningsno =
b.inningsno
        GROUP  BY b.bowlerid,
                  b.matchid) AS bowlerMatch
       JOIN player p
         ON p.id = bowlerMatch.bowlerid
GROUP  BY bowlerMatch.bowlerid,
          p.NAME
ORDER      BY   Sum(bowlerMatch.badcount)   /
Count(bowlerMatch.matchid) DESC;
```

VII. Who's whose bunny? (Highest number of times a batsman got out for the same bowler)

```
SELECT p1.NAME  AS Batsman,
       p2.NAME  AS Bowler,
       Count(*) AS Times_out
FROM   ballToBall b,
       player p1,
       player p2
WHERE    outputtype  IN  (  'caught',  'lbw',
'bowled', 'stumped' )
       AND b.strikerid = p1.id
       AND b.bowlerid = p2.id
GROUP  BY bowlerid,
          strikerid
HAVING Count(*) > 5
ORDER  BY times_out DESC;
```

VIII. *What is the highest score of runs in an over?

```
SELECT b.matchid,
       b.overid,
       b.inningsno,
        Sum(b.batsmanscore)                AS
score,
         COALESCE(Sum(be.extraruns), 0)    AS
extra,
       Sum(b.batsmanscore)
         + COALESCE(Sum(be.extraruns), 0) AS
temp
FROM   ballToBall b
       LEFT JOIN ballExtra be
              ON be.matchid = b.matchid
                 AND be.overid = b.overid
                 AND be.ballid = b.ballid
                       AND be.inningsno =
b.inningsno
GROUP  BY b.matchid,
          b.overid,
          b.inningsno
HAVING Sum(batsmanscore) > 20
ORDER  BY score DESC;
```

IX. Who took more than 100 wickets in IPL?

```
SELECT p.NAME,
Count(outputtype) AS wickets
FROM   ballToBall b
JOIN player p
         ON p.id = b.bowlerid
WHERE   outputtype IN ( 'caught', 'bowled',
'lbw', 'stumped' )
GROUP  BY bowlerid
HAVING Count(outputtype) > 100
ORDER  BY wickets DESC;
```

X. Who scored the most centuries in IPL?

```
SELECT name,
       Count(*) AS n
FROM   (SELECT m.matchid,
               p.name,
               Sum(b.batsmanscore) AS score
        FROM   ballToBall b
               LEFT JOIN `match` m
                              ON b.matchid =
m.matchid
               LEFT JOIN player p
                      ON b.strikerid = p.id
        GROUP  BY b.strikerid,
                  b.matchid
        HAVING Sum(b.batsmanscore) > 100) AS
cents
GROUP  BY name
ORDER  BY Count(*) DESC
LIMIT  1;
```

XI Most wickets by player sorted on wicket

```
SELECT DISTINCT( T.NAME ),
               wickets
FROM   (SELECT p.NAME,
               p.id,
               Count(outputtype) AS wickets
        FROM   ballToBall b
               JOIN player p
                 ON p.id = b.bowlerid
        WHERE  outputtype IN ( 'caught',
'bowled', 'lbw', 'stumped',
                               'caught and
bowled', 'hit wicket' )
        GROUP  BY bowlerid
        ORDER  BY wickets DESC) T
       JOIN playerToTeam ptt
```

```
         ON T.id = ptt.playerid
WHERE  ptt.teamid = 1
ORDER  BY wickets DESC;
```

Most runs for the team sorted on runs

```
SELECT DISTINCT( T.NAME ),
               runs
FROM   (SELECT p.NAME,
               p.id,
               Sum(batsmanscore) AS runs
        FROM   ballToBall b
               JOIN player p
                 ON p.id = b.strikerid
        GROUP  BY strikerid) T
       JOIN playerToTeam ptt
         ON T.id = ptt.playerid
WHERE  ptt.teamid = 1
ORDER  BY runs DESC;
```

Percentage of wins across seasons per team

```
SELECT ( y.win / x.tot ) * 100 AS percentage
FROM   (SELECT Count(*) AS tot
        FROM   `match`
        WHERE  team1 = 1
                OR team2 = 1) x
       JOIN (SELECT Count(*) AS win
             FROM   `match`
             WHERE  matchwinner = 1) y
         ON 1 = 1
```

Getting lucky with Toss

```
SELECT ( y.win / x.tot ) * 100 AS percentage
FROM   (SELECT Count(*) AS tot
        FROM   `match`
        WHERE  team1 = 1
                OR team2 = 1) x
       JOIN (SELECT Count(*) AS win
             FROM   `match`
             WHERE  tosswinner = 1) y
         ON 1 = 1
```

Quantity of left hand batters

```
SELECT Count(lefthanded)
FROM  (SELECT DISTINCT p.NAME AS Lefthanded
       FROM   player p,
              playerToTeam ptt
       WHERE  p.id = ptt.playerid
              AND p.bathandedness =
'Left-hand bat') Lefthanded
```

Quantity of left and right hand batters

```
SELECT (SELECT Count(DISTINCT( playerid ))
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid
        WHERE  teamid = 3
               AND p.bathandedness LIKE (
'%Right-hand Bat%' )) AS righthanded,
       (SELECT Count(DISTINCT( playerid ))
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid
        WHERE  teamid = 3
               AND p.bathandedness LIKE (
'%Left-hand Bat%' ))  AS lefthanded
FROM   dual
```

Quantity of right hand hand bowlers

```
SELECT Count(DISTINCT( playerid ))
FROM   player p
       JOIN playerToTeam ptt
         ON p.id = ptt.playerid
WHERE  teamid = 1
       AND p.bowlskill LIKE ( '%Right%' );
```

Quantity of left hand hand bowlers

```
SELECT Count(DISTINCT( playerid ))
FROM   player p
       JOIN playerToTeam ptt
         ON p.id = ptt.playerid
WHERE  teamid = 1
       AND p.bowlskill LIKE ( '%Left%' );
```

Quantity of left/right hand bowlers

```
SELECT (SELECT Count(DISTINCT( playerid ))
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid
        WHERE  teamid = 1
               AND p.bowlskill LIKE (
'%Right%' )) AS righthanded,
       (SELECT Count(DISTINCT( playerid ))
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid
        WHERE  teamid = 1
               AND p.bowlskill LIKE (
'%left%' ))  AS lefthanded
FROM   dual
```

Count of players with age < and > 25.

```
SELECT Count(DISTINCT( NAME ))
FROM   (SELECT p.NAME,
               Year(Curdate()) - Year(dob)
AS Age,
               teamid
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid) T
WHERE  teamid = 1
       AND age < 25;


SELECT Count(DISTINCT( NAME ))
FROM   (SELECT p.NAME,
               Year(Curdate()) - Year(dob)
AS Age,
               teamid
        FROM   player p
               JOIN playerToTeam ptt
                 ON p.id = ptt.playerid) T
WHERE  teamid = 1
       AND age > 25;
```

Home Win percentage

```
SELECT ( y.home_win / x.home_tot ) * 100 AS
percentage
FROM   (SELECT Count(*) AS home_tot
        FROM   `match` m
        WHERE  m.city IN (SELECT home
                          FROM   team
                          WHERE  team.id =
12)) x
       JOIN (SELECT Count(*) AS home_win
             FROM   `match` m
             WHERE  matchwinner = 12
                    AND m.city IN (SELECT
home
                                   FROM
team
                                   WHERE
team.id = 12)) y
         ON 1 = 1;
```

Query Optimization Hat-tricks (exponential blow-up)

```
SELECT Count(*)
FROM   (SELECT b1.matchid      AS b1_matchID,
               b1.overid       AS b1_overID,
               b1.ballid       AS b1_ballID,
               b1.inningsno    AS b1_inningsNo,
               b1.strikerid    AS b1_strikerID,
               b1.nonstrikerid AS b1_nonStrikerID,
               b1.bowlerid     AS b1_bowlerID,
               b1.batsmanscore AS b1_batsmanScore,
               b1.outputtype   AS b1_outputType,
               b2.matchid      AS b2_matchID,
               b2.ballid       AS b2_ballID,
               b2.overid       AS b2_overID,
               b2.strikerid    AS b2_strikerID,
               b2.nonstrikerid AS b2_nonStrikerID,
               b2.bowlerid     AS b2_bowlerID,
               b2.batsmanscore AS b2_batsmanScore,
               b2.outputtype   AS b2_outputType,
               b2.inningsno    AS b2_inningsNo,
               b3.matchid      AS b3_matchID,
               b3.ballid       AS b3_ballID,
               b3.overid       AS b3_overID,
               b3.inningsno    AS b3_inningsNo,
               b3.strikerid    AS b3_strikerID,
               b3.nonstrikerid AS b3_nonStrikerID,
               b3.bowlerid     AS b3_bowlerID,
               b3.batsmanscore AS b3_batsmanScore,
               b3.outputtype   AS b3_outputType
        FROM   balltoball b1,
               balltoball b2,
               balltoball b3) temp
WHERE  b1_matchid = b2_matchid
       AND b1_matchid = b3_matchid
       AND b1_overid = b2_overid
       AND b1_overid = b3_overid
       AND b1_ballid + 1 = b2_ballid
       AND b1_ballid + 2 = b3_ballid
       AND b1_inningsno = b2_inningsno
       AND b1_inningsno = b3_inningsno
       AND b1_outputtype IN ( 'caught', 'lbw', 'bowled',
'caught and bowled',
                              'stumped'
                            )
       AND b2_outputtype IN ( 'caught', 'lbw', 'bowled',
'caught and bowled',
                              'stumped'
                            )
       AND b3_outputtype IN ( 'caught', 'lbw', 'bowled',
'caught and bowled',
                              'stumped' )
```

Query Optimization Hat-tricks (optimized)

```
SELECT Count(*)
FROM   (SELECT *
        FROM   (SELECT b1.matchid    AS b1_matchID,
                       b1.overid     AS b1_overID,
                       b1.ballid     AS b1_ballID,
                       b1.inningsno  AS b1_inningsNo,
                       b1.outputtype AS b1_outputType
                FROM   balltoball b1
                WHERE  b1.outputtype IN ( 'caught', 'lbw',
'bowled',
                                          'caught and
bowled',
                                          'stumped' )) AS
ball1
               JOIN (SELECT b1.matchid    AS b2_matchID,
                            b1.overid     AS b2_overID,
                            b1.ballid     AS b2_ballID,
                            b1.inningsno  AS b2_inningsNo,
                            b1.outputtype AS b2_outputType
                     FROM   balltoball b1
                     WHERE  b1.outputtype IN ( 'caught',
'lbw', 'bowled',
                                               'caught and
bowled',
                                               'stumped' ))
AS ball2
                 ON ball1.b1_matchid = ball2.b2_matchid
                    AND ball1.b1_overid = ball2.b2_overid
                    AND ball1.b1_ballid + 1 =
ball2.b2_ballid
                    AND ball1.b1_inningsno =
ball2.b2_inningsno) ball12
       JOIN(SELECT b1.matchid    AS b3_matchID,
                   b1.overid     AS b3_overID,
                   b1.ballid     AS b3_ballID,
                   b1.inningsno  AS b3_inningsNo,
                   b1.outputtype AS b3_outputType
            FROM   balltoball b1
            WHERE  b1.outputtype IN ( 'caught', 'lbw',
'bowled',
                                      'caught and bowled',
                                      'stumped' )) AS ball3
         ON ball12.b1_matchid = ball3.b3_matchid
            AND ball12.b1_overid = ball3.b3_overid
            AND ball12.b1_ballid + 2 = ball3.b3_ballid
            AND ball12.b1_inningsno = ball3.b3_inningsno
```