Stored Procedures:

Top 5 Teams according to Average Points:
This is helpful especially for Users wanting to place “Over” bets.

```
DELIMITER //

CREATE PROCEDURE GetTop5AveragePoints()
BEGIN
   SELECT t.TeamName, AVG(TeamScores.PointsScored) AS AveragePoints
   FROM (
      SELECT HomeTeamID AS TeamID,
           CASE
              WHEN HomeTeamID = WinTeamID THEN WinTeamScore
              ELSE LoseTeamScore
           END AS PointsScored
      FROM Games
      UNION ALL
      SELECT AwayTeamID AS TeamID,
           CASE
              WHEN AwayTeamID = WinTeamID THEN WinTeamScore
              ELSE LoseTeamScore
           END AS PointsScored
      FROM Games
   ) AS TeamScores
   JOIN Teams t ON TeamScores.TeamID = t.TeamID
   GROUP BY t.TeamName
   ORDER BY AveragePoints DESC
   LIMIT 5;
END //

DELIMITER ;
```

Top 15 Teams according to Win Percentage:
This is helpful especially for Users wanting to place “Win/Loss” bets.

```
DELIMITER //

CREATE PROCEDURE GetTopTeamsByWinPercentage()
BEGIN
   WITH TotalGamesPerTeam AS (
      SELECT TeamID, COUNT(*) AS TotalGames
      FROM (
         SELECT HomeTeamID AS TeamID FROM Games
         UNION ALL
         SELECT AwayTeamID AS TeamID FROM Games
      ) AS AllGames
      GROUP BY TeamID
   ),
   TotalWinsPerTeam AS (
      SELECT WinTeamID AS TeamID, COUNT(*) AS Wins
      FROM Games
      WHERE WinTeamID IS NOT NULL
      GROUP BY WinTeamID
   )
   SELECT
      Teams.TeamID,
      TotalGamesPerTeam.TotalGames,
      COALESCE(TotalWinsPerTeam.Wins, 0) AS Wins,
      (COALESCE(TotalWinsPerTeam.Wins, 0) * 100.0 / TotalGamesPerTeam.TotalGames) AS
WinPercentage
   FROM Teams
   JOIN TotalGamesPerTeam ON Teams.TeamID = TotalGamesPerTeam.TeamID
   LEFT JOIN TotalWinsPerTeam ON Teams.TeamID = TotalWinsPerTeam.TeamID
   ORDER BY WinPercentage DESC
   LIMIT 15;
END //

DELIMITER ;
```

Trigger:

```
DELIMITER $$
CREATE TRIGGER trg_InsertUserBet
BEFORE INSERT ON UserBets
FOR EACH ROW
BEGIN
   DECLARE ExistingGameID INT;
   DECLARE NewGameID INT;

   -- Check if a GameID already exists in the Games table
   SELECT GameID INTO ExistingGameID
   FROM Games
   WHERE GameID = NEW.GameID;

   -- If no game exists, create a new GameID
   IF ExistingGameID IS NULL THEN
      -- Generate a new GameID (use MAX(GameID) + 1)
      SELECT COALESCE(MAX(GameID), 0) + 1 INTO NewGameID FROM Games;

      -- Insert the new game with only the GameID set
      INSERT INTO Games (GameID, HomeTeamID, AwayTeamID, GameDate, WinTeamID,
LoseTeamID, WinTeamScore, LoseTeamScore)
      VALUES (NewGameID, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

      -- Assign the newly created GameID to the UserBets row
      SET NEW.GameID = NewGameID;
   ELSE
      -- If the GameID exists, use the existing GameID
      SET NEW.GameID = ExistingGameID;
   END IF;
END$$
DELIMITER ;
```

The user enters the team names for both teams and the date. If a GameID exists with those, then need to do nothing. If a GameID does not exist with those requirements, then our query to insert a UserBet believes GameID is null. We have to add a new game with a new generated GameID to ensure that the user's bet is placed.

Transaction:

```
START TRANSACTION;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;


UPDATE UserBets
SET Status = %s
WHERE UserID = (SELECT UserID FROM UserInfo WHERE Username = %s)
AND GameID = (SELECT GameID
        FROM Games
        WHERE HomeTeamID = (SELECT TeamID FROM Teams WHERE
TeamName = %s)
        AND AwayTeamID = (SELECT TeamID FROM Teams WHERE TeamName
= %s)
        AND GameDate = %s
        LIMIT 1)
AND BetTypeID = (SELECT BetTypeID FROM BetTypes WHERE BetTypeName =
%s);


SELECT
ub.UserID,
ub.GameID,
ub.BetTypeID,
ub.Amount,
ub.Status,
ht.TeamName AS HomeTeamName,
at.TeamName AS AwayTeamName,
g.GameDate
FROM UserBets ub
JOIN Games g ON ub.GameID = g.GameID
JOIN Teams ht ON g.HomeTeamID = ht.TeamID
JOIN Teams at ON g.AwayTeamID = at.TeamID
WHERE ub.UserID = (SELECT UserID FROM UserInfo WHERE Username = %s)
   AND ub.Status = %s
   AND ub.Amount <= (
      SELECT AVG(Amount)
      FROM UserBets
WHERE UserID = (SELECT UserID FROM UserInfo WHERE Username = %s)
);
COMMIT;
```

```
START TRANSACTION;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
INSERT INTO UserBets (UserID, GameID, BetTypeID, Amount, Status)
SELECT
   (SELECT UserID FROM UserInfo WHERE Username = %s) AS UserID,
   %s AS GameID,
   (SELECT BetTypeID FROM BetTypes WHERE BetTypeName = %s) AS BetTypeID,
   %s AS Amount,
   %s AS Status
WHERE
   (SELECT UserID FROM UserInfo WHERE Username = %s) IS NOT NULL
   AND (SELECT BetTypeID FROM BetTypes WHERE BetTypeName = %s) IS NOT
NULL;

SELECT ho.*, bt.BetTypeName, ht.TeamName AS HomeTeamName, at.TeamName
AS AwayTeamName
FROM HistoricalOdds ho
JOIN Games g ON ho.GameID = g.GameID
JOIN Teams ht ON g.HomeTeamID = ht.TeamID
JOIN Teams at ON g.AwayTeamID = at.TeamID
JOIN BetTypes bt ON ho.BetTypeID = bt.BetTypeID
WHERE ht.TeamName = %s AND at.TeamName = %s;


COMMIT;
```

The first transaction requires Read Committed since we update a table and then immediately read from it to return the list of below average bets of that type. This is also the default isolation level for MySQL transactions.

We chose Repeatable Read for the second transaction because following the Insert there are only read processes. Repeatable Read is sufficient to fulfill all the parts of the transaction while being the most performant isolation level.