Baseball Database

Zachary Kellicutt, Patrick Delbarba, William Ordiway

Application background

Our group has created an application that allows users to insert and lookup statistics from the sport of baseball. This application will provide an interface where a user may query a baseball player's name and the player's statistics will be presented to the user. A team name can also be queried, which will bring up all the players and their stats for that team. Users will also be able to insert stats from each game for players. This application will effectively allow a team manager or baseball enthusiast to keep track of their team's player stats. As more games are played, new data can be inserted in the database, and of course, old data can always be looked up.

Data description

We will keep track of the following counting stats for each player:
Games played, plate appearances, at bats, runs, hits, doubles, triples, homeruns, RBIs, stolen bases, caught stealing, walks, strikeouts, games started, Innings pitched, hits allowed, runs allowed, earned runs, HR allowed, walks allowed, and strikeouts allowed.

From these we will be able to calculate all the basic non-counting stats (average, on-base percentage, slugging, earned-run average etc). These non-counting stats will be calculated on the fly from the counting stats that were brought up in the query.

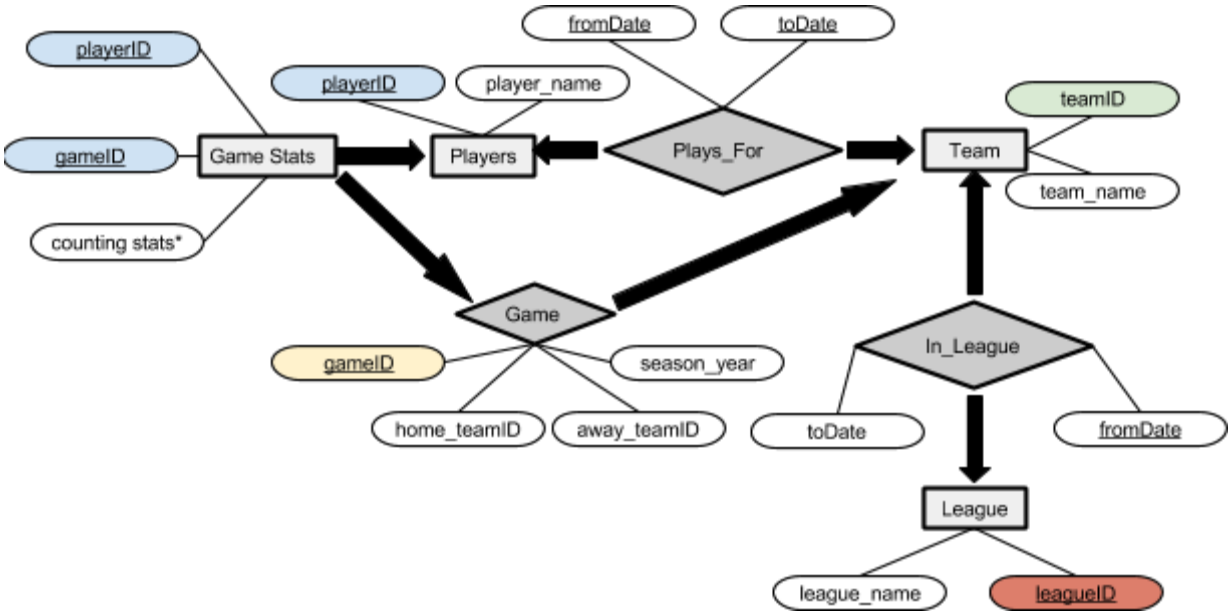
The tables will be organized in the following manner:

- Players: playerID, player_name
- Teams: teamID, team_name
- League: leagueID, league_name
- PlaysFor: playerID, fromDate, teamID, toDate
- In_League: teamID, fromDate, toDate, leagueID
- Game: gameID, home_teamID, away_teamID, seasonYear
- GameStats: gameID, playerID, countings stats*

Constraints

- Players can't play for multiple teams at the same time
- At least nine players must play for each team in a game
- Teams can't be in multiple leagues at the same time
- All attributes must have null assertions
- Teams can't have more than 40 players on roster at once

ER diagram



Functional dependencies

FDs from gamestats table. Candidate keys are in blue.

playerID ^ gameID → runs **playerID ^ gameID → at_bats** . . . **playerID ^ gameID → [other counting stats]**	The determinants playerID and gameID are candidate keys. Implies BCNF.

FDs from game table. Candidate key is in blue.

gameID → home_teamID	gameID is the candidate key and the

gameID → **away_teamID** **gameID** → **season_year**	determinate. Implies BCNF.

FDs from In_League table. Candidate keys are in blue.

teamID ^ fromDate → **leagueID** **teamID ^ fromDate** → **toDate**	The determinants playerID and gameID are candidate keys. Implies BCNF.

FDs from Plays_for table. Candidate keys are in blue.

playerID ^ fromDate → **teamID** **playerID ^ fromDate** → **toDate**	The determinants playerID and fromDate are candidate keys. Implies BCNF.

FD from Players table. Candidate key is in blue.

playerID → **player_name**	playerID is the candidate key and the determinate. Implies BCNF.

FD from Teams table. Candidate key is in blue.

teamID → **team_name**	teamID is the candidate key and the determinate. Implies BCNF.

FD from League table. Candidate key is in blue.

leagueID → League_name	leagueID is the candidate key and the determinate. Implies BCNF.

3NF || BCNF Schema

Having examined the functional dependencies implied by the relations in each table, and the fact that no functional dependencies exist but those, we know that not only does the database adhere to 3NF because there are no transitive dependencies, but it also adheres to BCNF as every determinate is a candidate key. Recognizing this demonstrates the absence of redundancy within the database.

Redundancy that could feasibly have existed within the database would have been the inclusion of calculated stats (RBI, Batting Average, etc…) within the game_stats table. This would have resulted in tuples determining the value of other tuples without being candidate keys (ex: Hits ^ at_bats → Batting_Average, by the formula Batting_Average = Hits/at_bats. But Hits and at_bats are not candidate keys). For that reason all calculated stats are generated upon query instead of stored in the database.

Indexes

SQLite is used for our database, so we are using a B+ tree index.

Triggers and assertions

The Trigger below enforces that in an instance where a user is adding game stats to the database that:

1. The game ID does point to a game in the game table (game stats added to DB after game created)

2. The player whose stats for that game is being entered must have played for the away team or the home team during the season that game took place.

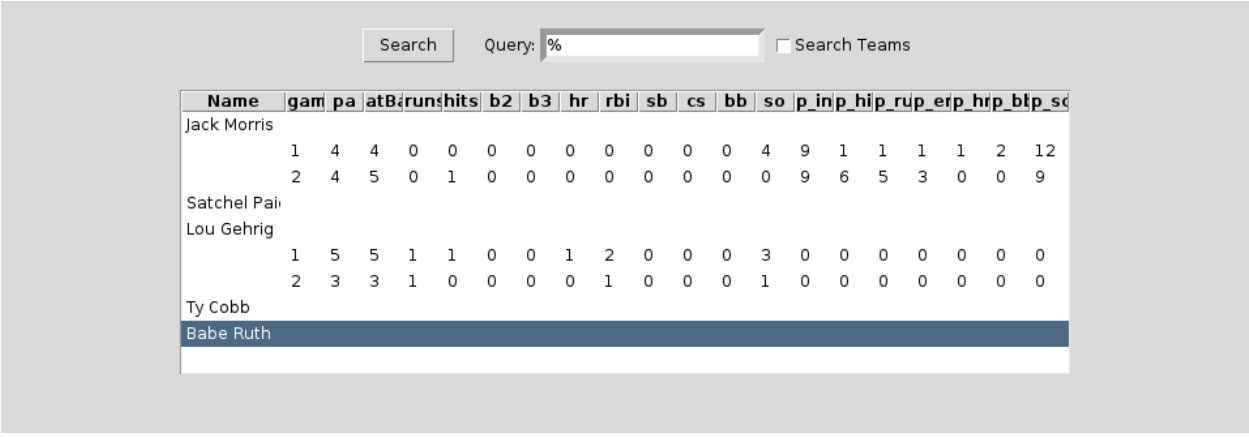
```
CREATE TRIGGER GameStatPlayerMustBeOnTeam
BEFORE INSERT ON GameStats
   BEGIN
      SELECT RAISE(FAIL, 'Player is not on a team  in Game')()
      WHERE NOT EXISTS(SELECT *
                        FROM Game_stats, Game, (SELECT * AS Plays_For_Relevant
                                                FROM Plays_For
                                                WHERE Game.seasonYear ≥ Plays_For.fromDate AND
                                                Game.seasonYear ≤ Plays_For.toDate)
                        WHERE new.playerID = Plays_For_Relevant.playerID AND new.gameID = Game.gameID AND
                        Plays_For_Relevant.teamID IN (Game.home_teamID, Game.away_teamID));
END;
```

The Assertion below, as part of the Plays_For table creation, enforces that in an instance where a user is adding players to a team that Teams can't have more than 40 players on roster at once.

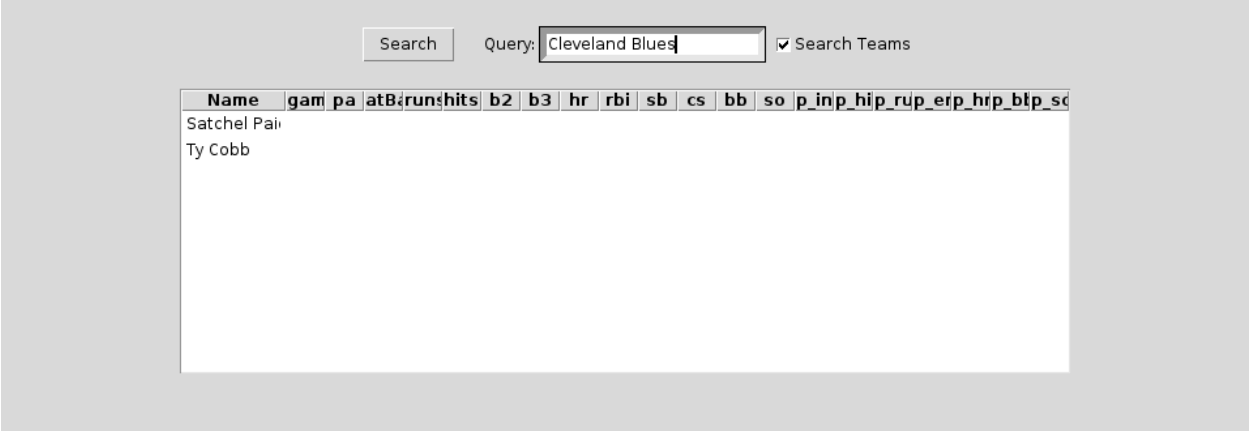
```
CREATE ASSERTION ApprorpietlySizedTeams
CHECK
( (SELECT COUNT (SELECT playerID
                 FROM Plays_For
                 WHERE Plays_For.toDate = YEAR(NOW())) < 40 )
```

Because all player's toDate will be the current year for a team they are currently on and this assertion exists on the addition of players to a team we only evaluate whether there are currently less than 40 players on the team [40 is permissible for the team, but then there is no room for this addition, so strictly less than 40 is necessary here]

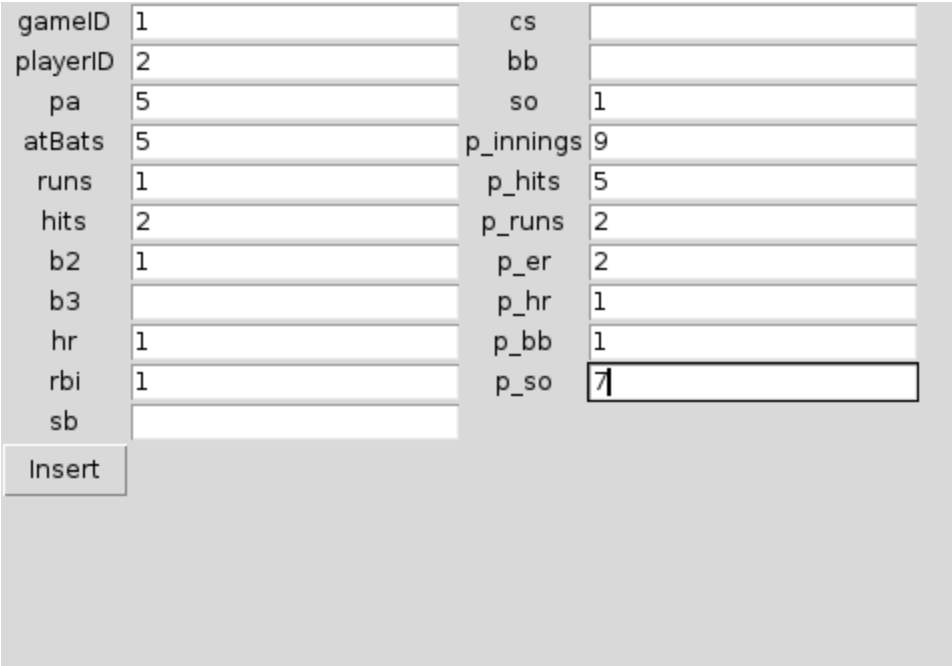
User Interface Screenshots



In the screenshot above, we see a query for all players in the database. The query brings up a list of all the players, and the user can double-click on each player to view his stats. In this case, both Jack Morris and Lou Gehrig have already been double-clicked.



In this screenshot, we see an example of a team query. The “Search Teams” checkbox is checked, so the user can search for a team name instead of a player name. In the results box, the list of players that currently play on the team are listed. As before, the players individual stats can viewed by double-clicking on the player names.



Finally, in this screenshot we see the data insertion interface. In this interface a player’s stats for any given game can be entered manually. Note that zeroes are automatically filled in the place of blanks.

Two Example Queries

SELECT * FROM player NATURAL JOIN gameStats WHERE playerName LIKE “Babe Ruth”

This searches the Player table for players with names similar to a search term then performs a natural join with gameStats to create a table with all players and starts from each of their games.

SELECT playerName FROM Player P, PlaysFor F, Team T WHERE P.playerID = F.playerID AND F.teamID = T.teamID AND T.teamName LIKE "Cleveland Blues"

This returns a list of all the names of all the players playing for a team defined by a search term.

Implementation

To interact with the database, a Python 3 desktop program was designed using TKinter. Given the application, Python was chosen due to its ease of use for rapid development as well as its string manipulation capabilities which were leveraged to handle querys. The application consists of two contexts each running on their own thread and both establish momentary database connections when needed to query or insert data. For the DMBS, SQLite 3 was used due to its simplicity and compatibility with source version control.

Team Member Role Descriptions

Zach:

- Team leader
- Baseball knowledge
- ER diagram/Tables

Patrick:

- SQL database
- Programming

William:

- Functional Dependencies/BNCF Proof
- Triggers
- Oral Presentation Generation and Organization
- Final Report Documentation Generation and Organization