Final Report

CS 3265: Database Management Systems

Baseball Stat Tracker

Lloyd Abernathy, Joshua Ludwig

Introduction

The purpose of our database application is simple. We wanted to create a large, historical, repository for baseball information and statistics. Our database contained every game's batting stats for all players since 1901 through 2021 during the regular season. The data was gathered from baseball-reference.com for every player with a plate appearance in a game. We wanted to use this data in our application to allow baseball lovers to find game by game statistics for their favorite players, as well as career stats. We organized our database to reflect these purposes.

Final Implementation

The final implementation of our application has several use cases. We accomplished all of the things we set out to do. We were able to implement the ability to sort the data by player name, as well as giving statistics by game date. We knew many baseball fans were also interested in all time leaders in many of the common stats. To accommodate for this we also implemented the ability to view all time leaders in many major statistics. We also included the ability to delete players from the database, as well as updating/inserting new games with plate appearances.

Database

Our database included 31 different attributes, and over 4.5 million rows. Each row represents a collection of stats for a single player during a single game. The normalization of our database was relatively simple. With the knowledge that each row represented a single game for a single player it was easy to test for functional dependencies. We first tested to make sure that there were no games for a single player that was split between two rows. This meant that the playerID taken together with the game date should be a functional dependency for the rest of the database (player_ID, game_date -> all other attributes). We also knew that every player had a

unique identifier that would return to us the player's name (player_ID, player_name). We tested many other functional dependencies but they were extraneous to the original candidate key.

Although our mega table was very close to 3NF, we still decided to break our tabel down into four separate tables. Our first table simply represented a player (player). One table represented common stats (common_player_stats_by_game), another table represented uncommon stats (uncommon_player_stats_by_game). Our last table represented a player's career stats (player_career_stats). We decided to break our table down this way to make cascading easier as well as returning relevant data.

To test our database we chose the player Seth Brown and simply returned the game date, the opposing team, and the result of the game, and displayed the results of that search. (see phpTest.php) We also had to delete several lines of data that caused database anomalies such as duplicates and specific games for specific players. (see project2.sql) Note: phpTest.php, searchDate.php, and searchName.php are just test files and are not connected to our final front-end application.

Advanced database implementations

We implemented several advanced SQL techniques to help with our final implementation. These advanced techniques can be found in several of our sql files (search_procedure.sql, project2.sql), and implemented and used through several php files (searchDate.php, searchName.php, deletePlayer.php, leaderboard.php. etc.). We used several stored procedures as well as triggers.

Stored procedures

We used stored procedures to implement our search functions. We created a procedure to return career stats for a player based on the player ID that was inputted. We used this procedure on the index page of our project (index.php). This is what allows the user to find the career stats of a player on our homepage.

We created another stored procedure to create new game entries in common_player_stats_by_game to format and correctly input 0s where input was not given (index.php). Finally, we created another stored procedure to use the player name to search for the player Id associated with the name. Although we didn't use this procedure in our final implementation of our application, we used this as the basis for our other search procedures. (see below)

```
-- Stored Procedure to get Career Stats based on player_id
DELIMITER //
DROP PROCEDURE IF EXISTS search_career //
CREATE PROCEDURE search_career(IN ID VARCHAR(20))
BEGIN
        DECLARE player_name_ TINYTEXT;
    DECLARE first_game_date VARCHAR(20);
    DECLARE last_game_date VARCHAR(20);
    DECLARE total_plate_app BIGINT UNSIGNED;
    DECLARE total_at_bat BIGINT UNSIGNED;
    DECLARE total_runs BIGINT UNSIGNED;
    DECLARE total_hits BIGINT UNSIGNED;
    DECLARE total_doubles BIGINT UNSIGNED;
    DECLARE total_triples BIGINT UNSIGNED;
    DECLARE total_homeruns BIGINT UNSIGNED;
    DECLARE total_RBIS BIGINT UNSIGNED;
    DECLARE total_strikeouts BIGINT UNSIGNED;
    DECLARE total_walks BIGINT UNSIGNED;
    DECLARE batting_average DECIMAL(4,3);

    SELECT player_name INTO player_name_ FROM player WHERE player_ID = ID LIMIT 1;
    SELECT game_date INTO first_game_date FROM common_player_stats_by_game WHERE player_ID = ID ORDER BY game_date LIMIT 1;
    SELECT game_date INTO last_game_date FROM common_player_stats_by_game WHERE player_ID = ID ORDER BY game_date DESC LIMIT 1;

    SELECT SUM(plate_app), SUM(at_bat), SUM(runs), SUM(hits), SUM(doubles), SUM(triples), SUM(homeruns), SUM(RBIS), SUM(strikeouts), SUM(base_on_balls)
    INTO total_plate_app, total_at_bat, total_runs, total_hits, total_doubles, total_triples, total_homeruns, total_RBIS, total_strikeouts, total_walks
    FROM common_player_stats_by_game WHERE player_ID = ID;

    SET batting_average = total_hits / total_at_bat;

    SELECT player_name_, first_game_date, last_game_date, total_plate_app, total_at_bat,
    total_runs, total_hits, total_doubles, total_triples, total_homeruns, total_RBIS, total_strikeouts, total_walks, batting_average;
END //
```

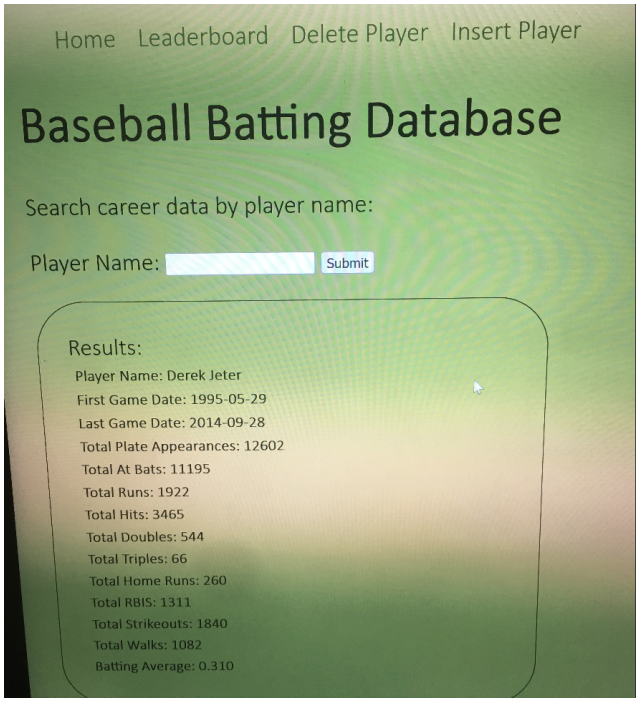
Triggers

We implemented a trigger to also allow for the cascading of our delete operation. When a player gets deleted from the database, our triggers then handle the deletion of the same player from the common player stats by game table as well as the uncommon player stats by game table. (see below)

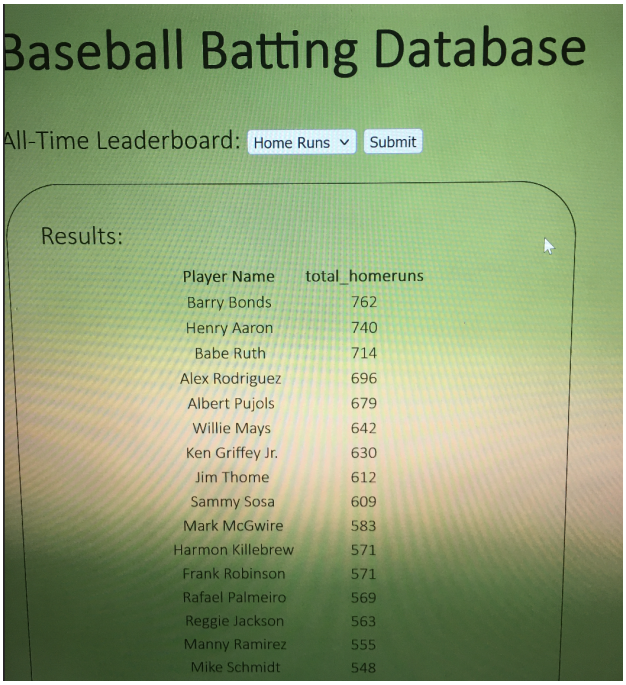
```
DROP TRIGGER IF EXISTS remove_from_common_table;
DELIMITER //
CREATE TRIGGER remove_from_common_table
AFTER DELETE
ON player
FOR EACH ROW
BEGIN
        DELETE FROM common_player_stats_by_game
    WHERE player_ID = OLD.player_ID;
END //
DELIMITER ;

DROP TRIGGER IF EXISTS remove_from_uncommon_table;
DELIMITER //
CREATE TRIGGER remove_from_uncommon_table
AFTER DELETE
ON player
FOR EACH ROW
BEGIN
        DELETE FROM uncommon_player_stats_by_game
    WHERE player_ID = OLD.player_ID;
END //
DELIMITER ;
```

Functionality walkthrough

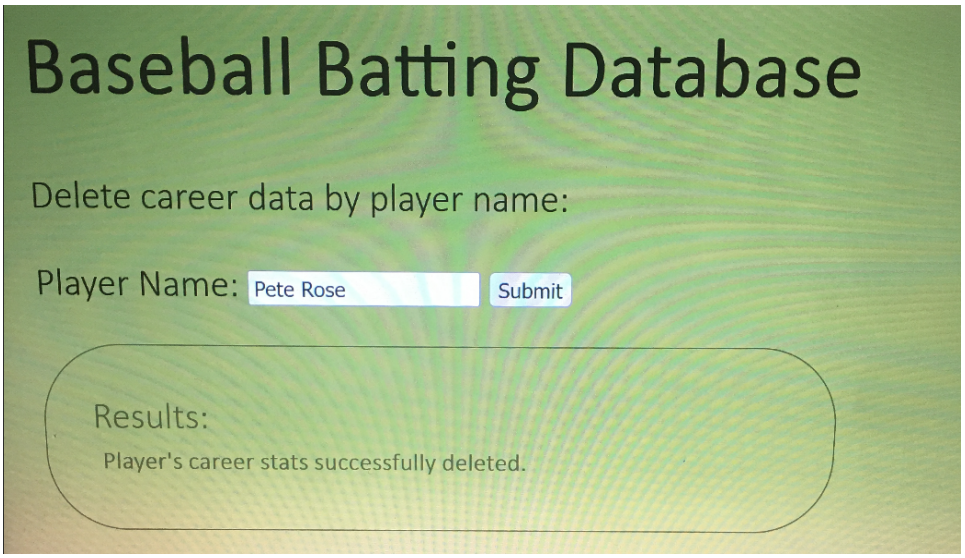
The homepage of our application includes a search for career stats for each player. To use this feature you will search for a name then select the player based off of the years played and the team played on. You then will be returned the common stats of that player throughout their career. For example if you were to search for Derek Jeter, our application will return his career statistics. (see below)



The leaderboard page of our application includes the top historical league leaders in four major stats. At-bats, hits, homeruns, and RBIs are all able to be displayed on the leaderboard page. For example if you select home runs you will see the list of all time leaders. (see below)



The delete player page will allow users to delete a player from the backend database based off of their name, and call the triggers to cascade that delete through our tables. It will then display whether or not the delete was successful. For example, to delete Pete Rose's stats from the database enter his name and submit. (see below)



Finally, the insert and update pages allow for new game entries to be created. On the insert page, the user inputs player name, game date, player team, opposing team, and game result such that a new game entry is created. Unfortunately, the update page only allows for updating plate appearances due to a shortage of time.

Summary

There were several roadblocks we experienced while working on this project. We both obviously had very busy schedules that we had to balance to allow for work to be done by the deadlines. We also had very little experience with PHP. Josh focused the most on the front end application, while Lloyd focused on the database normalization and final reporting. Both of us worked on creating and using advanced features in the database to return what was needed. If there were more time, more update functionality would be included in the front-end application.