CS 121 Final Project Reflection

Due: March 16th, 11:59PM PST

Note: This specification is currently for 22wi, subject to minor changes for 23wi.

This reflection document will include written portions of the Final Project. Submit this as **`reflection.pdf`** with the rest of the required files for your submission on CodePost.

For ease, we have highlighted any parts which require answers in **blue**.

Student name(s): **Carl Cheng**

Student email(s): **carl@caltech.edu**

Part L0. Introduction

Answer the following questions to introduce us to your database and application; you may pull anything relevant from your Project Proposal and/or README.

Database/Application Overview

What application did you design and implement? What was the motivation for your application? What was the dataset and rationale behind finding your dataset?

Database and Application Overview Answer (3-4 sentences) :

My proposed database is an NFL Superbowl statistics database that contains the stats of all players who have played before in the Superbowl. I focused on the offensive statistics of each player per each Superbowl game. The 'player' table contains information about the set of all players who have played in a Superbowl, the 'game' table contains information about every Superbowl game, the 'team' table contains the team name, and the 'game_stats' table contains per-game offensive statistics for each player, as well as the team the player played for in that game. We use this data to create an application that displays an up-to-date leaderboard of the number of Superbowl wins each NFL team has. The admin application can also update the game table to add a new Superbowl, and the client application can view a variety of other NFL statistical queries that are of interest to NFL scouts and fans.

Data set (general or specific) Answer:

Data was sourced from these sources and preprocessed using Python scripts and spreadsheet applications.

https://www.kaggle.com/datasets/timoboz/superbowl-history-1967-2020

https://www.kaggle.com/datasets/mattop/nfl-superbowl-offensive-statistics-19662021

https://www.kaggle.com/datasets/kendallgillies/nflstatistics (to get player birthdays, colleges, positions)

***Client user(s) Answer*:**

The intended userbase are NFL scouts/head coaches who want to query and analyze important statistics of teams and players in the Superbowl, or NFL fans who want to see who the all-time best performing teams/players are in the Superbowl.

Admin user(s) Answer:

The intended administrator would be an NFL database administrator who maintains and updates the database every time a new Superbowl occurs. Specifically, they would update tables 'game' and 'game_stats'.

Part A. ER Diagrams

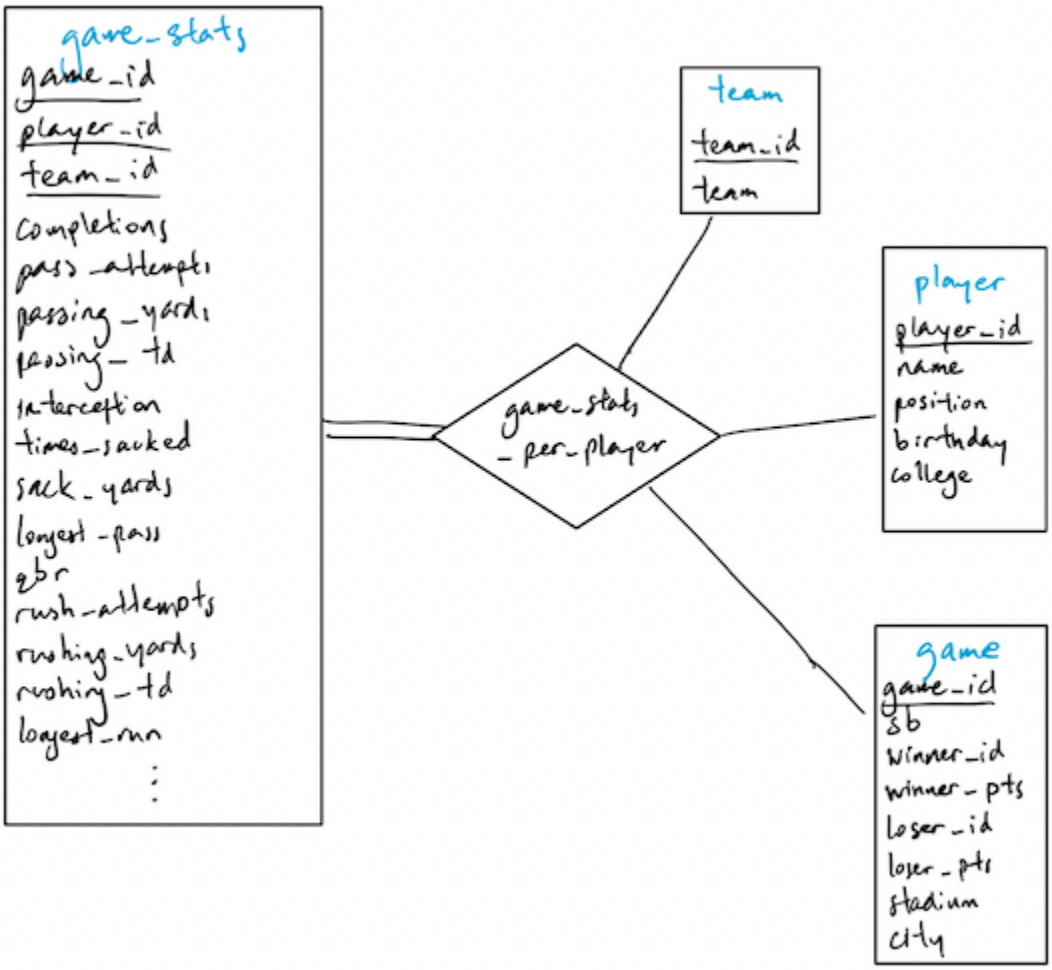
As we've practiced these past few weeks, the ER model is essential for designing your database application, and we expect you to iterate upon your design as you work through the ER and implementation steps. In this answer, you should provide a full ER diagram of your system. Your grade will be based on correct representation of the ER model as well as readability, consistency, and organization.

Notes: For this section **only**, we will allow (and encourage) students to share their diagrams on Discord (**#er-diagram-feedback**) to get feedback from other students on their ER diagrams given a brief summary of your dataset and domain requirements. This is offered as an opportunity to test your ER diagrams for accuracy and robustness, as another pair of eyes can sometimes catch constraints that are not satisfied or which are inconsistent with your specified domain requirements.

Requirements:

- Entity sets, relationship sets, and weak entity sets should be properly represented (also, do not use ER symbols not taught in class)
- Mapping cardinalities should be appropriate for your database schema, and in sync with your DDL
- Participation constraints should be appropriate and in sync with your DDL (total, partial, numeric)
- Use specialization where appropriate (e.g. *purchasers* and *travelers* inheriting from a *customers* specialization in A6)
- Do not use degrees greater than 3 in your relationships, do not use more than one arrow in ternary relationships.
- Use descriptive attributes appropriately
- Underline primary keys and dotted-underline discriminators
- Expectations from A6 still apply here
- Note: You do not need ER diagrams for views

ER Diagrams:


game_stats
game_id
player_id
team_id
completions
pass_attempts
passing_yards
passing_td
interception
times_sacked
sack_yards
longest_pass
qbr
rush_attempts
rushing_yards
rushing_td
longest_run
team
team_id
team
player
player_id
name
position
birthday
college
game_stats
_per_player
game
game_id
sb
winner_id
winner_pts
loser_id
loser_pts
stadium
city

Part B. DDL (Indexes)

As mentioned in Part B, you will need to add at least one index at the bottom of your **`setup.sql`** and show that it makes a performance benefit for some query(s).

Here, describe your process for choosing your index(es) and show that it is used by at least one query, which speeds up the performance of the same query on a version of the same table without that index. You may find lec14-analysis.sql and Lecture 14 slides on indexes useful for strategies to choose and test your indexes. **Remember that indexes are already created in MySQL for PKs and FKs, so you should not be recreating these.**

Index(es):

CREATE INDEX idx_pass_td ON game_stats(team_id, passing_td);

CREATE INDEX idx_rush_td ON game_stats(team_id, rushing_td);

CREATE INDEX idx_receiving_td ON game_stats(team_id, receiving_td);

Justification and Performance Testing Results:

Since a common query could be to query the number of touchdowns a team scored within a time period, or to simply query the total number of touchdowns a team has scored, I decided to make indexes on all types of touchdowns with team_id so queries like the ones aforementioned would be faster. Using a query that found the total number of passing and rushing touchdowns per team, without the index, the query took 0.0026 seconds, but with the index, the query took only 0.0006 seconds. This is clearly a massive improvement. Times were acquired using phpMyAdmin SQL query box.

A similar index could be made with (player_id, passing_td), etc… , to speed up queries related to player touchdowns instead of team touchdowns.

Part C. Functional Dependencies and Normal Forms

Requirements (from Final Project specification):

- Identify *at least 2 non-trivial functional dependencies* in your database
- Choose and justify your decision for the normal form(s) used in your database for at least 4 tables (if you have more, we will not require extra work, but will be more lenient with small errors). BCNF and 3NF will be the more common NF's expected, 4NF is also fine (but not 1NF).
    - Your justification will be strengthened with a discussion of your dataset breakdown, which we expect you to run into trade-offs of redundancy and performance.
- For two of your relations having at least 3 attributes (each) and at least one functional dependency, prove that they are in your chosen NF, using similar methods from A7.
    - If you have identified functional dependencies which are not preserved under a BCNF decomposition, this is fine
- Expectations from A7 still apply here.

Functional Dependencies:

player: {player_id} -> {name, position, birthday, college}

- Since the player table takes the form player(player_id, name, position, birthday, college), then clearly each player_id is associated with exactly one name, position, birthday, and college.

game_stats:

{game_id, player_id, team_id} -> {completions, pass_attempts, passing_yards, passing_td, interception, times_sacked, sack_yards, longest_pass, qbr, rush_attempts, rushing_yards, rushing_td, longest_run, receptions, receiving_yards, receiving_td, longest_reception, fumble, fumbles_lost}

- Each player in a specific game is associated with a specific set of game statistics depending on what the player did in the game. Since each player is on a team and two teams play in each game, then each combination of game_id, player_id, team_id is associated with a specific set of game stats: completions, pass_attempts, passing_yards, passing_td, interception, etc…

Normal Forms Used (with Justifications):

Our DDL for the 'superbowl' database is in 3NF.

Clearly, for the player table, all attributes depend on player_id, so player_id is a superkey since each player_id is associated with exactly one name, position, birthday, and college. We decided to keep all relevant player information with the player_id in one table since there are only 4 other attributes.

Similarly, for the team table, there is only one other attribute team that depends on the primary key team_id, so team_id is clearly a superkey since each team_id is associated with exactly one team. The

team table was one of our design choices to use an id for teams, as well as to have other tables reference team(team_id) to properly update and delete the team_ids, winner_ids, and loser_ids in other tables.

For the game table, the other attributes (sb, winner_id, winner_pts, loser_id, loser_pts, stadium, city, and state) depend on the primary key game_id since each game has exactly one of those attributes. Additionally, the foreign key constraints in game (winner_id and loser_id references team(team_id)) implies a one-to-many relationship between game and team, since each game can only have one winner and one loser but each team can play in multiple games. Thus, game must be in 2NF, and furthermore, since winner_pts and loser_pts depend only on winner_id and loser_id which only depend on game_id, then game must be in 3NF. This design for game ensures that all attributes are solely dependent on game_id, so it ensures game is in 3NF.

For the game_stats table, the offensive stats attributes (all attributes aside from game_id, player_id, and team_id) are all dependent on game_id, player_id, team_id, and the only other dependency is that player_id, team_id are dependent on game_id (since there is exactly one player_id, team_id combination for each game), so no non-key attributes are dependent on another non-key attribute. Thus, game_stats is in 3NF. The design choice of including all offensive stats in one table is so that a database administrator would not have to keep track of multiple tables when adding new superbowl games to the database, and because the offensive stats are pretty self-explanatory to a football watcher, so keeping all offensive stats keeps things relatively simpler. We also wanted to eliminate redundancy since if we split it into multiple tables, then we would have redundant stats per position (e.g. a quarterback, wide receiver, and runningback could have a rushing touchdown)

Thus, our database is in 3NF.

NF Proof 1:

We can show a relation is in 3NF by finding a candidate key for the relation and verifying that all non-primary attributes are functionally dependent on the candidate key. Let player_id be the candidate key. We will test it and its subsets (there are no subsets) that they are superkeys.

Clearly, for the table players, we have R(player_id, name, position, birthday, college) and F = {player_id → name&&position&&birthday&&college}, and by reflexivity player_id → player_id, so player_id → player_id&&name&&position&&birthday&&college and thus A → ABCDE for R(A, B, C, D, E), so all non-primary attributes are functionally dependent on the candidate key and R is in 3NF.

NF Proof 2:

We will do the same to show that the table game is in 3NF. Let game_id be the candidate key. We will test it and its subsets (there are no subsets) that they are superkeys.

For the table game, we have R(game_id, sb, winner_id, winner_pts, loser_id, loser_pts, stadium, city). For F, since winner_pts is dependent on how much the winner scored and loser_pts is dependent on how much the loser scored, then we have F = {winner_id → winner_pts, loser_id → loser_pts}. Additionally for F, there is one winner and one loser per superbowl, and each game_id represents a different superbowl, so we have F = {game_id → sb&&winner_id&&loser_id, winner_id → winner_pts, loser_id → loser_pts}. Finally, each superbowl takes place in a specific stadium and a

specific city, so we conclude with the set of functional dependencies F = {game_id → sb&&winner_id&&loser_id&&stadium&&city, winner_id → winner_pts, loser_id → loser_pts}. By transitivity, if game_id → winner_id and winner_id → winner_pts, then game_id → winner_pts. The same logic can be applied to loser_pts, so game_id → loser_pts as well. Thus, by union, game_id → {all attributes in R} and thus all non-primary attributes are functionally dependent on the candidate key and R is in 3NF.

Part G. Relational Algebra

Requirements (from Final Project specification, Part G):

- Minimum of 3 non-trivial queries (e.g. no queries simply in the form **`SELECT <x> FROM <y>`**)
- At least 1 group by with aggregation
- At least 3 joins (across a minimum of 2 queries)
- At least 1 update, insert, and/or delete
    - This may be equivalent to said SQL statements elsewhere (e.g. queries or procedural code), but are not required to be; in other words, you can write these independent of other sections
- Appropriate projection/extended projection use
- Computed attributes should be renamed appropriately
- Part of your grade will come from overall demonstration of relational algebra in the context of your schemas; obviously minimal effort will be ineligible for full credit; it is difficult to formally define "obviously minimal", but refer to A1 and the midterm for examples of what we're looking for
- Above each query, briefly describe what it is computing; we will use this to grade for correctness based on what the query is supposed to compute; lack of descriptions will result in deductions, since we have no idea otherwise of what the query is intended to do.

Below, provide each of your RA queries following their respective description.

1. Find all players who have played for the New England Patriots in the superbowl and count the number of games they have played in.

$$\text{NWE\_superbowl\_players\_counts} \leftarrow \Pi_{\text{name, num\_games}}({}_{\text{player\_id}}\mathcal{G}_{\text{count(*) as num\_games}}(\sigma_{\text{team.team = 'NWE'}}(\text{game\_stats} \bowtie \text{player} \bowtie \text{team})$$

2. Find the total number of passing touchdowns and rushing touchdowns each team has scored in the history of the Superbowl

$$\text{total\_touchdowns} \leftarrow \Pi_{\text{team, total\_pass\_tds, total\_rush\_tds}}({}_{\text{team\_id}}\mathcal{G}_{\text{sum(passing\_td) as total\_pass\_tds, sum(rushing\_td) as total\_rush\_tds}}(\text{game\_stats} \bowtie \text{team})$$

3. Update the player and game_stats relations to exclude players that have position OT.

$$\text{player} \leftarrow \text{player} - \sigma_{\text{position = 'OT'}}(\text{player})$$
$$\text{game\_stats} \leftarrow \text{game\_stats} - \Pi_{\{\text{attributes of game\_stats}\}}(\sigma_{\text{position = 'OT'}}(\text{game\_stats} \bowtie \text{player}))$$

Part L1. Written Reflection Responses

Challenges and Limitations

List any problems (at least one) that came up in the design and implementation of your database/application (minimum 2-3 sentences)

Answer:

The biggest challenge and the hardest part of the project was the initial preprocessing of the datasets. Many entries weren't filled out, so I had to allow for null values for nearly all attributes except primary keys. Additionally, all warnings when loading data are due to the processing of null values, and although I could've spent more time on the handling of null values such as providing a default value, I decided to focus my time more on the design of the DDL and ensuring the schema is 3NF, as well as the procedural SQL and command line applications, which were more fun and less tedious than writing Python scripts to preprocess the CSVs. The CSVs were mostly annoying to preprocess due to lack of consistency between the two datasets; for example, the player name field for both datasets had different formats; Last, First versus First Last, and the way they represented teams differed as well.

Future work

If you are particularly eager for a certain application (have your own start-up in mind?), it is easy to over-scope a final project, especially one that isn't a term-long project. You can list any stretch goals you might have "if you had the time" which staff can help give feedback on prioritizing (2-3 sentences).

Answer:

- Adding additional procedural SQL in setup-routines to give a view, procedure, and trigger for inserting entries into game_stats. This would be implemented for use in app-admin.py.
- Implement the remaining functions in setup-routines (such as completion_pct) for use in app-client.py.
- Implement the remaining procedures in setup-routines (such as update_game_location) for use in app-admin.py.

Collaboration (Required for Partners)

This section is required for projects which involved partner work. Each partner should include 2-3 sentences identifying the amount of time they spent working on the project, as well as their specific responsibilities and overall experience working with a partner in this project.

I was responsible for all of the work in this project. No partner or collaborators were involved.

OTHER COMMENTS

This is the first time CS 121 has had a Final Project, and we would appreciate your feedback on whether you would recommend this in future terms, as well as what you found most helpful, and what you might find helpful to change.

Answer:

Good project! Pretty intense/difficult without a partner though :(