CS 121 Final Project Proposal

Student name(s): Halle Blend, Eli Seiner

Student email(s): hblend@caltech.edu, eseiner@caltech.edu

For full credit, we are looking for a robust proposal demonstrating careful consideration of the motivation, design process, and implementation plan for your application; you should expect to spend 1-2 hours minimum on your proposal (finding a dataset may take the longest). For each part of this proposal that requires your response, you should have 2-3 sentences (bullet points are encouraged as well) justifying your answers. We strongly encourage you to include as much detail as you can in this proposal (this is your only assignment this week) and you can include any diagrams and SQL brainstorming (e.g. a start to your DDL) with your submission on Canvas. If you want to add any additional planning, you can include a diagram as a PDF/PNG/etc. and/or .sql files, though not required. You can also include a starter Python program with function stubs for a command-line implementation you will complete for the Final Project (without SQL integration).

The advantage of adding these in your proposal is to get you started in advance, and also to get feedback from the staff on your design and implementation so far.

Database/Application Overview

In this proposal, you will be "pitching" your project, in which you have some freedom in choosing the domain of with a structured set of requirements that bring together the course material in a single project, from design to implementation to tuning. Keep in mind that unlike CS 121 assignments, you are free to publish/share your project, which can be useful for internship or job applications. In terms of scope, you should shoot for 8-12 hours on this project, though you are free to go above and beyond if you choose.

First, what type of database application are you planning on designing? Remember that the focus of this project is the database DDL and DML, but there will be **a small Python command-line application** component to motivate its use and give you practice applying everything this term in an interactive program. Don't worry too much about implementation at this step, and you can jot down a few ideas if you have more than one. Just think about something you would like to build "if you had the data" which could be simulated with a command-line program in Python. Your command-line program will start with a main menu with usage options for different features in your application. Staff are here to help you with scoping!

Here's a list of some application ideas to get you started. These encompass most of the applications within the scope of this project we anticipate students might be interested in, but if there's a different application that meets the requirements for your DB schema and implementation, you are welcome to ask!

Proposed Database and Application Program Answer (3-4 sentences) :

Our proposed database is an NFL historical player stats database. **We found this database from https://www.kaggle.com/kendallgillies/nflstatistics?select=Basic_Stats.csv, which stores several csv files regarding historical NFL stats for players from 1924 to 2016. We are selecting five csv files to focus on in an overall category of offense in football: Basic Stats (essentially player information), Career Passing Stats, Career Rushing Stats, Career Receiving Stats, and Career Defensive Stats. We can apply this data by using our created dataset to create an application that makes it easier to compare different players across different eras in order to decide which players have strong enough stats to be considered for the hall of fame, an extremely prestigious award granted to only the most elite players .**

Next, where will you be getting your data? We will post a list of datasets to get you started, but you can find many here and here (look for ones in CSV file(s), which you will break into schemas similar to the Spotify assignment; we can also help students convert JSON to CSV if needed). You are also welcome to auto-generate your own datasets (e.g. with a Python script) and staff are more than happy to help you with the dataset-finding process, which can take longer than the rest of the starting design process.

Data set (general or specific) Answer:

Kaggle: NFL Statistics

https://www.kaggle.com/kendallgillies/nflstatistics/version/1?select=Basic_Stats.csv

- **Basic Stats**
- **Career Stats Passing**
- **Career Stats Receiving**
- **Career Stats Rushing**
- **Career Stats Defensive**

Who is the intended user base and what is their role? You will need to have at least one client usertype, and one admin. These different users may have different permissions granted for tables or procedural SQL.

***Client user(s) Answer*: The intended client user base are fans of football, people who need some database to store historical information about football (records, history, etc.), people who serve as scouts that need reliable information about player history, and people who serve on the hall of fame voting committee who need access to a database to easily compare players.**

Admin user(s) Answer: **The intended admin users are NFL statisticians, who have access to reliable NFL statistics and can continually update the tables, creating an updated database of all player's career statistics.**

Required Features

The full specification of the project will outline the requirements of the project, but you remember you should aim to spend 8-12 hours (it replaces the final exam and A8), depending on how far you want to go with it. The time spent on finding or creating a dataset will vary the most. For the proposal, you will need to brainstorm the following (you can change your decisions later if needed).

E-R Model: There will be an ER model component, but it is not required for the proposal.

DDL: What are possible tables you would have for your database? Remember to think about your application (e.g. command-line functions for user prompts to select, add, update, and delete data, either as a client or an admin user). We encourage you to include the attributes and their types as well at this step, though not required. Include at least 4 tables in your response, as well as any design decisions you may run into in your schema breakdown.

Answer: List of tables

Player (player_id, position, status, games_played, seasons_played)

- Player_id: VARCHAR(50) NOT NULL
- Position: CHAR(3)
- Status: CHAR(7)
    - CHECK status in ('Active', 'Retired')
- Games_played: INT
- Seasons_played: INT

Player_info (player_id, name, birth_place, birthday, college, height_inches, weight_lbs)

- Player_id: VARCHAR(50) NOT NULL
- Name: CHAR(30) NOT NULL
- Birth_place: CHAR(50)
- Birthday: DATE
- College: CHAR(25)
- Height_inches: INTEGER
- Weight_lbs: INTEGER

Defense: (player_id, year, curr_team, tot_tackles, solo_tackles, assist_tackles, sacks, safeties, pass_defended, ints, int_td, int_yards, yards_per_int)

- Player_id: VARCHAR(50) NOT NULL
- Year: YEAR NOT NULL
- Curr_team: VARCHAR(30)
- Tot_tackles: INTEGER
- Solo_tackles: INTEGER
- Assist_tackles: INTEGER
- Sacks: FLOAT
- Safeties: INTEGER
- Pass_defended: INTEGER
- Ints: INTEGER

- Int_td: INTEGER
- Int_yards: INTEGER
- Yards_per_int: FLOAT

Passing: (player_id, year, curr_team, pass_attempt, pass_complete, percent_complete, attempt_per_game, td_passes, interception, int_rate, passes_over_twenty, passes_over_forty, sacks, passer_rating)

- Player_id: VARCHAR(50) NOT NULL
- Year: YEAR NOT NULL
- Curr_team, VARCHAR(30)
- Pass_attempt: INTEGER
- Pass_complete: INTEGER
- Percent_complete: FLOAT
- Attempt_per_game: FLOAT
- Td_passes: INTEGER
- Interception: INTEGER
- Int_rate: FLOAT
- Passes_over_twenty: INTEGER
- Passes_over_forty: INTEGER
- Sacks: INTEGER
- Passer_rating: FLOAT

Rushing: (player_id, year, curr_team, rush_attempt, attempt_per_game, rush_yards, yards_per_carry, yards_per_game, rush_tds, rush_first_down, rush_over_twenty, rush_over_forty, rush_fumbles)

- Player_id: VARCHAR(50) NOT NULL
- Year: YEAR NOT NULL
- Curr_team: VARCHAR(30)
- Rush_attempt: INTEGER
- Attempt_per_game: FLOAT
- Rush_yards: INTEGER
- Yards_per_carry: FLOAT
- Yards_per_game: FLOAT
- Rush_tds: INTEGER
- Rush_first_down: INTEGER
- Rush_over_twenty: INTEGER
- Rush_over_forty: INTEGER
- rush_fumbles: INTEGER

Receiving: (player_id, year, curr_team, receptions, receiving_yards, yards_per_reception, yards_per_game, receiving_tds, reception_over_twenty, reception_over_forty, reception_first_down, receive_fumbles)

- Player_id: VARCHAR(50) NOT NULL
- Year: YEAR NOT NULL
- Curr_team: VARCHAR(30)
- Receptions: INTEGER
- Receiving_yards: INTEGER
- Yards_per_reception: FLOAT

- Yards_per_game: FLOAT
- Receiving_tds: INTEGER
- Reception_over_twenty: INTEGER
- Reception_over_forty: INTEGER
- Reception_first_down: INTEGER
- receive_fumbles: INTEGER

Queries: Part of the application will involve querying data in a way that would be useful for a client (e.g. searching for "Puzzle" games made in the last 5 years, ordered by year and price in a Video Game Store database, or finding all applicants who are pending for an adoption agency). Identify at least 3 queries that would make sense in a simple command-line application. In your answers, provide a brief description of the query or pseudocode, as well as the purpose in your application. These will likely be implemented as SQL queries within Python functions, wrapping your SQL queries (the methods of which will be taught in class). You are welcome (and encouraged) to add more, though not required.

Answers:

1. **Searching for all quarterbacks who passed for over 4,000 yards in a season who played for a certain team, ordered by year and number of passing yards.**
2. **Searching for the total number of rushing touchdowns scored by each team throughout their history, ordered by the number of touchdowns scored.**
3. **Searching for the total number of receptions longer than 30 yards for each player in their career ordered by the number of receptions.**
4. **Search for all players of any position (offensive or defensive) who played more than 500 career games, ordered by the number of games played.**

Procedural SQL: You will need to implement at least 1 each of a UDF, procedure, and trigger in your project. Identify at least one of each of these here. For each:

1. What is the name of the function/procedure/trigger?
2. Why is it motivated in your database? Remember for example that there are overheads for triggers, so they shouldn't be trivial.

Consider some examples in class/HW, such as logging DML queries, adding an extra layer of constraint-handling (e.g. the overdraft example from lecture), etc. For procedures, remember that these can be called in an application program written in a language like Python or Node.js, so these are especially useful to avoid writing queries in such application programs, *especially* SQL that performs **`INSERT/DELETE/UPDATE`** which you do not want to leave to an application user. Remember that you can also set permissions for different users to access tables and procedures. In your Python program, you can connect to your database as different users defined in your database (similar to the Node.js program El demo'd a while back).

Answers

UDF(s): We will create an eligible_for_hof udf that indicates whether a player, given their stats and their position, has strong enough career statistics to be eligible for the hall of fame. This function is motivated by the overall purpose of the database, which is to store information about players in order to select/decide which players should be inducted into the hall of fame.

Procedure(s): Because we plan on having a hall of fame materialized view, we plan on constructing a procedure that updates the hall of fame materialized view when new information (new players) are added or removed from the database. This would be extremely convenient, since we will have a specific location that is continuously updated that stores all hall of fame players regardless of position. If the player is already in the view then the information is updated to account for the new data.

Trigger(s): We can make an add and delete trigger to automatically update our tables (specifically our hall of fame materialized view) when we modify our database. We will use these triggers in our procedures when we add or remove data to our hall of fame materialized view. This is helpful so that we have an automatic update when changing our tables, saving running time.

Database Performance: At this point, you should have a rough feel for the shape and use of your queries and schemas. In the final project, you will need to add at least one index and show that it makes a performance benefit for some query(s). You don't need to identify what index(es) you choose right now (that comes with tuning) but you will need to briefly describe how and when you would go about choosing an index(es). Refer to the material on indexes if needed.

Performance Tuning Brainstorming: **We could create an index on player (player_id, year) for faster lookups on each player's stats during some year. This can make the queries run faster when looking at specific years and players, which we do in some of the queries.**

"Stretch goals"

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.

Answer:

One stretch goal could be to extend the application to creating a Fantasy Football league. This could include drafting players for teams and comparing them in different weeks to have a team win with the most collective points. Also, if we have enough time, we could add information about a player's success (how often they won the Super Bowl), since this is obviously correlated to how a player is perceived.

Potential roadblocks

List any problems (at least one) you think could come up in the design and implementation of your database/application.

Answer: **Because we attained the csv files from kaggle, we are not exactly sure how accurate the data is (though we should still be able to use the data as an example if it is not accurate). Additionally, we may encounter some challenges when we write our function because our database will involve so many different positions. While this will not add much difficulty, it will add much more complexity (way more cases to handle in our function, based on the position), which can potentially cause some headaches.**

Open Questions

Is there something you would like to learn how to implement in lecture? Any other questions or concerns? Is there anything the course staff can do to help accommodate these concerns?

Answer:
For the command line implementation, will there be resources available to help us implement the command line application? While both of us are comfortable with python, I'm not sure we have much experience with command line applications.
We are also unsure about the complexity of our tables. We include lots of stats to consider when choosing a player for the hall of fame, but is our table set up and the type of data we are looking at complex enough or too simple?

Have fun!