CS 121 23wi Final Project Proposal

Due: Saturday Feb. 18th, 11:30PM

Student name(s): Mihir Borkar and Rupa Kurinchi-Vendhan

Student email(s): mborkar@caltech.edu and rkurinch@caltech.edu

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; you can find a template from last year here, which may be adjusted slightly before the Final Project is released, but gives you an idea of the breakdown. 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!

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

Our project will be working with a Jeopardy! database with information about 18 previous seasons. The database will include information about the players (such as their name, occupation, and which podium position they were in), information on questions from the games (question text and value), and information on responses from the games (including who selected a question, who answered it, its value, and how much was wagered). The end goal is an application which allows clients to interact with this data through providing queries to get information on past Jeopardy! games/players/questions and submitting requests for updating our database with corrections or more up-to-date information.

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:

Our Jeopardy! dataset (which we found here) spans 18 different seasons (Season 16 to Season 33), with roughly 230 games per season. Overall there are 7,855 contestants, 225,347 questions, and 32,061 question categories. This dataset is structured by 5 CSV files: contestants.csv, results.csv, locations.csv, questions.csv, and trend.csv. Based on the information provided within each of these files, we propose the following schema breakdown:

```
games_info(game_id, season, season_type)

podium_positions(game_id, player_id, seat_location)

contestant(player_id, first_name, last_name, hometown_city, hometown_state,
occupation)

responses(game_id, round_num, row_idx, column_idx, correct_respondent, value,
wager, dd_asker)

questions(game_id, round_num, row_idx, column_idx, category, value,
question_text, answer)
```

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. Consider clients as having the ability to search ("query") data from the command line, submit requests to insert/update data, etc. In 22wi, some students had a separate client .py file and admin .py file, with admins having admin-related features, such as approving requests, inserting/updating/deleting information, querying, etc.

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

From the client side, our application allows for submitting searches for data from the command line to query information about past Jeopardy! Games. Clients may also submit requests to update our database if they find errors or want to add more recent information.

Admin user(s) Answer:

From the admin side, our application manages the database itself. This includes inserting all information from the dataset, fulfilling searches, and approving client requests to update information. Because our database includes information from previous games, the admin will not need to insert/update/delete information otherwise.

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). To make this step easier, thinking about the menu options your application provides, as well as the queries you might want to support, is helpful to narrow down the information you'll want to model. At minimum, you must provide a general listing of attributes and possible types; you can provide these on a high-level (e.g. bullet points) or actual DDL with CREATE TABLE statements in an optional **`setup.sql`** file attached in your submission. The more you provide, the more we can help provide feedback. Include at least 4 tables in your response, as well as any design decisions you may run into in your schema breakdown.

Answer:

See **`setup.sql`** for more details and the current draft of the DDL.

Here are some other things to note:

- The responses table and the questions table have different numbers of rows so we may want to only keep questions/responses that appear in both tables.
- A question is uniquely identified by the game it is in, which round it is in, and which position it is in on the board. Therefore the primary key for questions/responses should be (game_id, round_num, row_index, col_index). We could potentially combine these into one long id (with the round_num, row_index, and col_index concatenated with the game_id). In all of the other tables the primary keys are easier to identify.
- One other thing to note is that in many of the other tables, players are referred to by the podium position they are playing at (left, middle, right), plus the id of the game they are playing in, so this, in addition to the player_id, can be useful to identify the player in the contestants table.

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:

Our group is hoping to include the following queries/questions in our Jeopardy! search engine. Each of the questions below have applications as potential input to our search engine from the user.

1. Which player won the most money through Jeopardy!? in season []?
For each game in the responses table, find the podium position of the player who answered correctly. If the season is specified, filter results by the given season. For that game, find the player_id in that position. Find the name corresponding to this player_id and add the value of the question to their total earnings. Find the maximum of these total earnings.

2. What is the average amount of money won per season? per player?
For each game in the responses table, sum the value of the question answered. If interested in results per season, combine results for games in the same season. If interested in results per player, find the podium position of the player who answered correctly. If the season is specified, filter results by the given season. For that game, find the player_id in that position. Find the name corresponding to this player_id and add the value of the question to their total earnings. Sort by player.

3. Is it better to answer questions that are easier and worth less money, or questions that are harder and worth more money?
This query is much harder to implement, but we are planning to calculate the expected-value payout for each value of question. For every question in the responses table, we will use the actual buzz outcomes to total the net dollar outcome and divide this by the number of responses for that question. For example, if a $400 question was answered incorrectly by the person who asked the question and a second contestant but answered correctly by the third contestant, the average value would be $166 (i.e. ($400 - $400 - $400)/3). If no one responds correctly to the chosen question, then the expected value is -$400. We will average the results of questions from the same round with the same value, and compare/report their expected values.

4. Which round is most indicative/influential of the final score, if any?
This is another higher-level question. For each game in the responses table, for each round, find the podium position of the player who answered correctly and add the value of the question to their total earnings. Find which player has the highest total earnings to see who won the game. Report their earnings per round, and compare which round contributed most to the total earnings of each winner.

* Note that [] indicates variables we would like to parametrize for queries.

Procedural SQL: You will need to implement at least 1 each of a UDF, procedure, and trigger in your project (we haven't seen triggers yet, so you don't need to provide one in your proposal). Otherwise, identify at least one UDF and one procedure here. For each:

1. What is the name of the function/procedure?
2. Why is it motivated in your database? Consider the differences discussed in Lecture 10 for UDFs, procedures, queries, temporary tables, and views. In your Final Project, we'll be

looking for you to demonstrate an understanding of appropriate design decisions here (and we're happy to help discuss any trade-offs)

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 demoed a while back).

Answers

UDF(s):

1. get_total_earnings
2. This UDF will take in a player_id or possibly player name as inputs and return the total earnings of that player. This, and similar UDFs, will allow database users to quickly find out statistics about players and games. One reason for implementing this as a UDF is to create some abstraction, so that it is easier to get the total earnings of a player, as this information might be needed in other more complex queries and we do not want to reimplement it every time we need it.

Procedure(s):

1. update_player_info
2. This procedure will take in some new player info that the client wants to update due to incorrect information as input and update that player information in the database. This input will probably include the player name or player_id as well as the values of the new information. We want to try to add an extra layer of security so that this sends a request that the database admin has to approve in order for the updates to be made. We are not sure about how to do this so would appreciate some guidance. This should be implemented as a procedure so that the table/schema information is hidden from the client user and we are not running explicit UPDATE statements in the Python application.

"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 potential way to extend our work is to implement a UI/UX design to allow for clients to input their queries or submit database update requests. This would look similar to most search engines, where there is a box for clients to enter searches. On a separate page, there may be another text box for providing information about suggested database changes, which asks for their name, the type of update they want to make, and the specific details of the update.

If time permits, a larger undertaking may be to implement Jeopardy! gameplay for clients to practice their skills with questions from previous games. This could be as simple as a simple trivia game where clients answer as many randomly chosen questions as possible to earn points, or as complicated as a full-scale Jeopardy! interface where players can choose their categories, earn points, and compete against other players.

Potential roadblocks

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

Answer:

Our database contains many rows of information, especially with 225,769 total questions over the 18 seasons. This may make obtaining results for queries slower and more time intensive than intended. Because we are working with a lot of data, it is also important to ensure that our queries are as efficient as possible, avoiding any correlated subqueries or unnecessarily complicated commands.

Collaboration

For projects with partners, this section is required (if not, you can leave it out, or note that you are decided yet). Both students should provide a brief summary of their planned workload distribution, method(s) of collaboration, and 1-2 points you are most interested in the project. You may also clarify any concerns/confidence for your partner work here. Feel free to provide a paragraph or bullet points; we're looking for you to have thought this out and discussed your plan for collaboration at this step.

Rupa's Answer:

I'm interested in the data cleaning process, so I will be formatting the information from the CSV files, setting up the schemas, and pulling the data into SQL tables. With this groundwork, I can pass off the database to Mihir to do his section of the project. I would also like to help brainstorm ways to answer the larger scale questions that could help inform gameplay for future contestants, such as which round of the game most influences the result.

Mihir's Answer:
I will be primarily focusing on developing the command-line Python application that the client will interact with, since I am interested in developing applications with SQL. Our main method of collaboration will be a chat on Discord and we could also set up a Github repository. As a long term goal, which may be outside of the scope of this project, I would like to develop a Jeopardy! game application that interacts with a SQL database to provide questions and maintain a leaderboard.

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:
What kinds of updates/inserts, given our Jeopardy! database schema, are the best to allow for a client side user to do?

How do we go about implementing added security such as when a user requests an UPDATE to the database via a procedure an admin has to approve it?

What kinds of things would usually be found in the "main menu" of the Python command-line application for this kind of project?

How are triggers usually used in this kind of application?

Have fun!

Optional Brainstorming/Sketches/Other Notes

This is an optional section you can provide any other brainstorming notes here that may help in the design phase of your database project (you may find it helpful to refer to the early design phase in your Final Project Reflection).
