CS 121 Final Project Reflection

Due: March 18th, 11:59PM PST

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): Mihir Borkar, Rupa Kurinchi-Vendhan

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

Our project works 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. This will be useful for statisticians and game theorists to analyze historical data from Jeopardy! games.

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.

***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. Sample queries can provide players who won the most money overall or in a particular season, as well as how much money was earned on average per player or per season.

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 updating information to correct errors or add more recent games. Because our database includes information from previous games, the admin will not need to insert/update/delete information otherwise.

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.

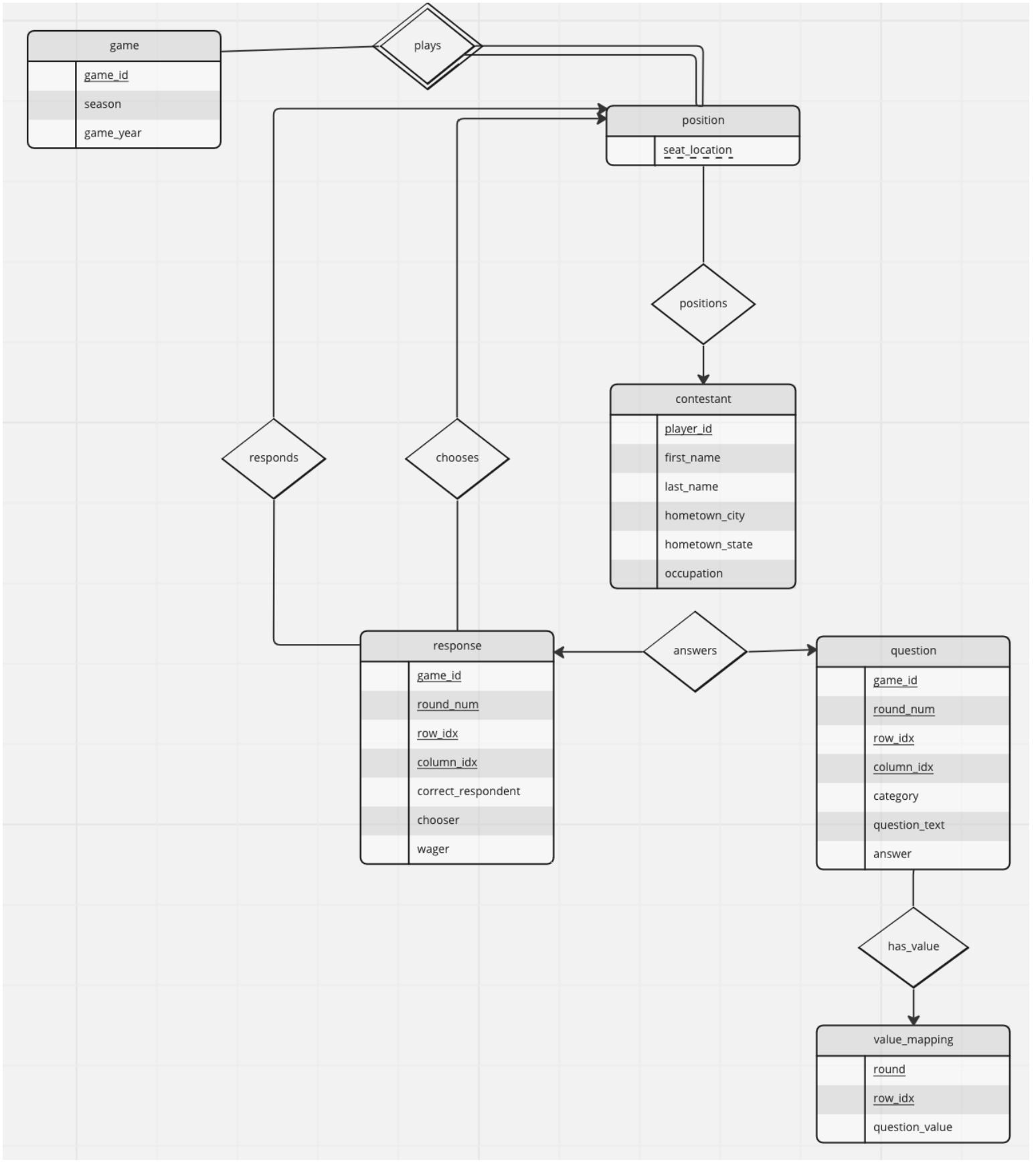
Notes: For this section **only**, we will allow (and encourage) students to share their diagrams on Discord (**#er-diagram-feedback**) to get feedback from other students on their ER diagrams given a brief summary of your dataset and domain requirements. This is offered as an opportunity to test your 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:

Note: See the next page. We did not include all components of this E-R in the DDL (we left out the responds, chooses, has_value, and answers tables), but the entities and relationships in the diagram are correct. We excluded the responds and chooses tables (which provide information on which seat locations chose or responded to questions), since they were not needed for the queries we were interested in . The answers table would contain information that is the same for each game (round, row_idx, column_idx) so it doesn't add anything useful . Finally, the has_value table could be implemented efficiently as a join if needed, since the value_mappings table is very small. As shown in the E-R, we decomposed our original questions table into two tables (questions and value_mappings) because of the FDs {game_id, round, row_idx, column_idx} → {category, question_text, answer} and {round, row_idx} → {question_value}, which otherwise lead to redundancy from partial dependencies. See the setup.sql file and Part C for more information.


game
game_id
season
game_year
plays
position
seat_location
positions
contestant
player_id
first_name
last_name
hometown_city
hometown_state
occupation
responds
chooses
response
game_id
round_num
row_idx
column_idx
correct_respondent
chooser
wager
answers
question
game_id
round_num
row_idx
column_idx
category
question_text
answer
has_value
value_mapping
round
row_idx
question_value

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_category ON questions (category);

Justification and Performance Testing Results:

Creating an index on the question category will improve the performance of many queries where we want to search for question information based on the question's category, and these queries are one of the most common question related searches. Our index makes the test query we chose about 4 times as fast and this was verified with repeated testing. Results from one test are shown below:

```
mysql> SELECT SUM(v.question_value) AS total_value
    ->         FROM questions AS q
    ->         INNER JOIN value_mapping as v
    ->         ON q.round = v.round AND q.row_idx = v.row_idx
    ->         WHERE q.category BETWEEN 'FOOD' AND 'GRIT';
+--------------+
| total_value |
+--------------+
|     7803000 |
+--------------+
1 row in set (0.16 sec)

mysql> EXPLAIN SELECT SUM(v.question_value) AS total_value
    ->         FROM questions AS q
    ->         INNER JOIN value_mapping as v
    ->         ON q.round = v.round AND q.row_idx = v.row_idx
    ->         WHERE q.category BETWEEN 'FOOD' AND 'GRIT';
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------------------------------+--------+----------+-------------+
| id | select_type | table | partitions | type   | possible_keys | key     | key_len | ref                                     | rows   | filtered | Extra       |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------------------------------+--------+----------+-------------+
|  1 | SIMPLE      | q     | NULL       | ALL    | NULL          | NULL    | NULL    | NULL                                    | 254176 |    11.11 | Using where |
|  1 | SIMPLE      | v     | NULL       | eq_ref | PRIMARY       | PRIMARY | 23      | jeopardydb.q.round,jeopardydb.q.row_idx |      1 |   100.00 | NULL        |
+----+-------------+-------+------------+--------+---------------+---------+---------+-----------------------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.01 sec)

mysql> CREATE INDEX idx_category ON questions (category);
Query OK, 0 rows affected (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT SUM(v.question_value) AS total_value
    ->         FROM questions AS q
    ->         INNER JOIN value_mapping as v
    ->         ON q.round = v.round AND q.row_idx = v.row_idx
    ->         WHERE q.category BETWEEN 'FOOD' AND 'GRIT';
+--------------+
| total_value |
+--------------+
|     7803000 |
+--------------+
1 row in set (0.04 sec)

mysql> EXPLAIN SELECT SUM(v.question_value) AS total_value
    ->         FROM questions AS q
    ->         INNER JOIN value_mapping as v
        ON q.round = v.round AND q.row         INNER JOIN value_mapping as v
    ->         ON q.round = v.round AND q.row_idx = v.row_idx
        WHERE q.category BETWEEN 'FOOD' AND 'GRIT';         ON q.round = v.round AND q.row_idx = v.row_idx
    ->         WHERE q.category BETWEEN 'FOOD' AND 'GRIT';
+----+-------------+-------+------------+--------+---------------+-------------+---------+-----------------------------------------+--------+----------+--------------------------+
| id | select_type | table | partitions | type   | possible_keys | key         | key_len | ref                                     | rows   | filtered | Extra                    |
+----+-------------+-------+------------+--------+---------------+-------------+---------+-----------------------------------------+--------+----------+--------------------------+
|  1 | SIMPLE      | q     | NULL       | range  | idx_category  | idx_category | 1019    | NULL                                    | 16460 |   100.00 | Using where; Using index |
|  1 | SIMPLE      | v     | NULL       | eq_ref | PRIMARY       | PRIMARY     | 23      | jeopardydb.q.round,jeopardydb.q.row_idx |      1 |   100.00 | NULL                     |
+----+-------------+-------+------------+--------+---------------+-------------+---------+-----------------------------------------+--------+----------+--------------------------+
2 rows in set, 1 warning (0.00 sec)
```

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:

{game_id} → {season, game_year}

{game_id, seat_location} → {player_id}

{player_id} → {first_name, last_name, hometown_city, hometown_state, occupation}

{game_id, round, row_idx, column_idx} → {correct_respondent, chooser, wager}

{game_id, round, row_idx, column_idx} → {category, question_text, answer}

{round, row_idx} → {question_value}

Normal Forms Used (with Justifications):

The games table is in BCNF, the contestants table is in BCNF, the positions table is in BCNF, the responses table is in BCNF, the questions table is in BCNF, the value_mapping table is in BCNF, and the plays table has no FDs associated with it.

We decided to keep separate games and contestants tables because we may need to add new contestants that have not yet played in a game or have not answered any questions in a game. This avoids any update, insert, or delete anomalies. This leads to both of these tables being in BCNF because they each only have one FD with the primary key on the left side associated with them..

We decided to have a positions table which stores how game_id and seat_location maps to player_id. This is because the responses data we were provided identifies who responded to a question based on their seat_location in that game. This leads to redundantly storing the game ids in the positions table, but this is worth it since we can easily identify players from the game_id and seat_location. This leads to this table being in BCNF because it only has one FD with the primary key on the left side associated with it.

Originally our questions table was in 1NF because it had the attributes game_id, round, row_idx, column_idx, category, question_text, question_value, and answer but there was a partial dependency because of the FD {round, row_idx} → {question_value}. This is because the round and the position on the game board determines a question's value. Therefore, our original questions table had lots of redundant information due to storing the question_value column. We split the questions table into two tables (questions and value_mapping) so that the two FDs {game_id, round, row_idx, column_idx} → {category, question_text, answer} and {round, row_idx} → {question_value} have separate tables associated with them. This makes both the resulting tables be in BCNF because these FDs involve a primary key on the left side. Although splitting into two tables makes it so that we have to look up the question_value in the value_mapping table, this is worth it because we get rid of a lot of redundant information in storing the question values for each question (there's 225k questions) and the value_mapping table has only 12 rows since there are 12 possible round + board position to question_value mappings.

NF Proof 1:

The games table is in BCNF because the only non-trivial FD associated with it is {game_id} → {season, game_year} and game_id is a candidate key (also the primary key) of the games table. A table is in BCNF if all of its non-trivial FDs have a candidate key on the left side.

NF Proof 2:

The questions table is in BCNF because the only non-trivial FD associated with it is {game_id, round, row_idx, column_idx} → {category, question_text, answer} and (game_id, round, row_idx, column_idx) is a candidate key (also the primary key) of the questions table. A table is in BCNF if all of its non-trivial FDs have a candidate key on the left side.

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.

Note: In the relational algebra expressions below, *name* is in the format "{first_name} {last_name" and *question_points* is output of a UDF which gives the earnings of the contestant who chooses a particular question. Our dataset is formatted such that contestant responses are based on player positions within the game. As a result, we needed to map player positions per game to the contestants themselves in order to calculate each player's total score. To calculate the total score per season, we calculated the total score per player per season, grouped seasons together. To calculate average scores, we extended this methodology to average scores by player and by season.

What are the total earnings of the players through Jeopardy?

responses_values ⟶ games ⋈ responses ⋈ value_mapping

$\Pi_{\text{name} \rightarrow \text{contestant, SUM(question\_points)} \rightarrow \text{total\_score}}(_{\text{first\_name, last\_name, SUM(question\_value)}}G(\rho_j(\text{responses\_values})$

$\bowtie_{\text{j.correct\_respondent = p.seat\_location } \Lambda \text{ j,chooser = p.seat\_locations } \Lambda \text{ j.game\_id = p.game\_id}} \rho_p(\text{positions}) \bowtie_{\text{p.player\_id = c.player\_id}}$

$\rho_c(\text{contestants}))$

What are the total earnings of the players in season 16?

responses_values ⟶ games ⋈ responses ⋈ value_mapping

$\Pi_{\text{name} \rightarrow \text{contestant, SUM(question\_points)} \rightarrow \text{total\_score}}(_{\text{first\_name, last\_name, SUM(question\_value)}}G(\sigma_{\text{g.season = 16}}(\rho_j(\text{responses\_values}) \bowtie$

$_{\text{j.correct\_respondent = p.seat\_location } \Lambda \text{ j,chooser = p.seat\_locations } \Lambda \text{ j.game\_Id = p.game\_id}} \rho_p(\text{positions}) \bowtie_{\text{p.player\_id = c.player\_id}}$

$\rho_c(\text{contestants})))$

What is the average amount of money won per player?

responses_values ⟶ games ⋈ responses ⋈ value_mapping

$cp \rightarrow positions \bowtie contestants$
$num\_games \rightarrow \Pi_{name \rightarrow contestant,\ COUNT\text{-}DISTINCT(game\_id) \rightarrow num}\ (_{first\_name,\ last\_name,\ COUNT\text{-}DISTINCT(game\_id)}\ G(cp))$
$\Pi_{name \rightarrow contestant,\ SUM(question\_points)\ /\ ng.num \rightarrow avg\_score}\ (_{first\_name,\ last\_name,\ num}\ G(\rho_j(responses\_values) \bowtie_{j.correct\_respondent = p.seat\_location\ \Lambda\ j,chooser = p.seat\_locations\ \Lambda\ j.game\_Id = p.game\_id}\ \rho_p(positions) \bowtie_{p.player\_id = c.player\_id}\ \rho_c(contestants) \bowtie_{name = ng.contestant}\ \rho_{ng}(num\_games))$

What is the average amount of money won per season?
$responses\_values \rightarrow games \bowtie responses \bowtie value\_mapping$
$cp \rightarrow positions \bowtie contestants$
$num\_games \rightarrow \Pi_{name \rightarrow contestant,\ cp.season \rightarrow season,\ COUNT\text{-}DISTINCT(game\_id) \rightarrow num}\ (_{first\_name,\ last\_name,\ COUNT\text{-}DISTINCT(game\_id)}\ G(cp))$
$player\_totals \rightarrow \Pi_{name \rightarrow contestant,\ ng.season \rightarrow season,\ SUM(question\_points)\ /\ ng.num \rightarrow avg\_score}\ (_{first\_name,\ last\_name,\ num}\ G(\rho_j(responses\_values) \bowtie_{j.correct\_respondent = p.seat\_location\ \Lambda\ j,chooser = p.seat\_locations\ \Lambda\ j.game\_Id = p.game\_id}\ \rho_p(positions) \bowtie_{p.player\_id = c.player\_id}\ \rho_c(contestants) \bowtie_{name = ng.contestant}\ \rho_{ng}(num\_games))$
$\Pi_{season,\ AVG(avg\_score) \rightarrow season\_avg\_score}\ (_{season,\ AVG(avg\_score)} G(player\_totals))$

Add a new contestant to the database.
$contestants \rightarrow contestants \cup \{(11375, "Buford", "Frink", "Pasadena", "CA", "author")\}$

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:

One of the biggest challenges our team faced when writing the queries for calculating the scores for each player was the formatting of the data. The dataset that we were working with lists contestant responses in terms of their seat location rather than their player id or name. To address this issue, we had to manipulate our queries to map these seat locations to the corresponding contestant based on the game that the response was from. Additionally, calculating the Jeopardy! scores was more involved than we initially assumed. Again, due to the structure of our responses data, we had to consider whether the contestant who correctly answered the question matched the contestant who selected the question in the first place, and whether a wager was made (in the case of a Daily Double). To make our queries more readable, we added a UDF to handle the casework for score calculation.

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:

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.

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.

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.

Partner 1 Name: Rupa Kurinchi-Vendhan
Partner 1 Responsibilities and Reflection:
I was responsible for setting up the schemas in the DDL and pulling the data into SQL tables. Additionally, I wrote the queries (and relational algebra expression) for calculating the total/average earnings per player/season, and created the UDFs for client searches, and procedures for admins to update our database. Overall, I spent 15 hours on this project, and I believe that the workload was distributed fairly between Mihir and I.

Partner 2 Name: Mihir Borkar
Partner 2 Responsibilities and Reflection:
I was responsible for the data clean-up of the CSVs/TXTs so that they could be imported into SQL, making the ER diagram, writing about the FDs, and editing the DDL. I also developed and debugged both the client and admin Python command-line applications and interfaced them with Rupa's UDFs/procedures. Overall, I believe I spent 15 hours on the project as well, and I believe the workload was distributed fairly between Rupa and I.

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:
We thoroughly enjoyed doing the work for this project, as it was an excellent way to tie together all the skills we gained over the course of the term with a fun, real-world application. Having work sessions in-class and optional 1-1s were extremely helpful, both as a way to decompress from the rigor of the term and as an opportunity to work with El directly and ask questions/workshop ideas. The guidelines for the project were also clearly spelled out and easy to follow. One of the most important parts of this project was selecting the dataset that we wanted to work with. We think it would be helpful to provide resources at the beginning of the project with links to datasets that students have used in the past in addition to the resources that were shared.