CS 411 Group 116 Project – Stage 2

Sruthi Kode (kode2), Samidha Sampat (ssamp2),
James Mallek (jmallek2), Kavya Moharana (kavyam3)

2. Explanation of Assumptions

We included six entities in our UML diagram –

User Entity:
We created a User as an entity rather than a relationship because the User is the person who is playing the game and has many characteristics that are all tied to that entity. Therefore, it does not make sense to model the User as an attribute because there are many characteristics associated with the User. This includes the UserId, Email, Encrypted Password, Date Joined, Most Guessed, and Average Guesses. If the User was an attribute, our diagram would be complex and hard to understand. The User is the "main character" of our system because they are the ones who are interacting with the system. The cardinality from User to Guesses shows that a user can make 0 to many guesses. In turn, each guess is connected to one user. The cardinality from User to Courses shows that a user makes a guess about a specific course. However, a course can have many guesses that pertain to it - 0 to many to be more precise. A user having a most guessed course means that they guess that single course at the highest frequency than other courses. Thus for the user, the guesses relationship can be seen as one-to-many and the most guessed relationship can be seen as many-to-one for the Courses.

Guess Entity:
For the Guess entity, we assume that each guess is for a specific course. We modeled Guess as an entity instead of an attribute of an entity because we need to store each individual guess to provide the user with statistics about their guesses. For example, users will be able to see their average amount of guesses and their most guessed course. There are several aspects of the guess that need to be stored, making it an entity. Each Guess is linked to the user who is making the guess, and a single user can make 0 to many guesses. Therefore the relationship between Guess and User is many-to-one. Each Guess is also linked to one course. The relationship between the guesses and courses is similarly many-to-one because several guesses can link to a single course, but each Guess can only link to one course at most. Additionally, Guess has a many-to-one relationship with DailyClass where there can be many guesses associated with a single DailyClass.

DailyClass Entity:
The Class of the Day is modeled as an entity because we need to store specific information about the chosen course and display statistics to the user based on that day's guesses. For instance, the user will be able to view the average number of guesses made across all users for that day and be given the college (or 'academic unit') that the Class of the Day falls under, prior to making guesses. After successfully completing the game, they will also be able to view the most commonly guessed wrong answer for that day. This entity has a one-to-many relationship with the Guess entity because each Class of the Day will receive many guesses. DailyClass also has a

many-to-one relationship with the Courses entity because there can be multiple days on which the Class of the Day will be the same course.

Courses Entity:
Courses are modeled as an entity instead of an attribute of an entity because they are the main subject matter of the application. We need to track several attributes of each course in order to provide the user with useful hints about them. Courses has a one-to-one relationship with CourseDescription because each individual course should only have one description. Courses has a many-to-one relationship with Departments because each course is only linked to one department, but a single department is linked to many courses. Courses has a flipped relationship with Guess because many guesses can be linked to a single course but only one course is linked to each guess. There is also a relationship between Courses and DailyClasses which is a 1 to many relationship because the Class of the Day is based on date. There can be multiple instances where a course is deemed the Class of the Day.

Departments Entity:
The Department is modeled as an entity rather than an attribute of the Courses entity because it provides information that is relevant to multiple courses. Our approach removes the need to store redundant information. This is both the name of the department and the college unit that the department is a part of. The college unit refers to the college or school that the department is housed in, such as Grainger Engineering or iSchool. Departments are only related to the Courses entity and it is a one to many relationship because one department typically has many courses.

4. Normalization of the database

User Entity:
UserId (A), Email (B), EncryptedPassword (C), DateJoined (D), MostGuessed (E), AvgGuesses (F)

- Candidate Keys: UserId (A), Email (B)
- (Minimal) Functional Dependencies:

  $A \rightarrow B, C, D, E, F$

  $B \rightarrow A, C, D, E, F$

The FD’s obey the 3NF conditions where the left-hand side are the unique superkeys and the right-hand side are the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are functionally dependent on the super keys.

Guess Entity:
GuessId (A), UserId (B), Date (C), CourseId (D), Value (E)

- Primary Key: GuessId (A)
- (Minimal) Functional Dependency:

  $A \rightarrow B, C, D, E$

The FD obeys the 3NF conditions where the left-hand side is the primary key and the right-hand side is all the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are fully functionally dependent on GuessId (A).

DailyClass Entity:
Date (A), CorrectCourseId (B), MostCommonWrong (C), TotalAvgGuesses (D)

- Primary Key: Date (A)
- (Minimal) Functional Dependency:

  A → B, C, D

The FD obeys the 3NF conditions where the left-hand side is the primary key and the right-hand side is all the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are fully functionally dependent on the Date (A).

Courses Entity:
CourseId (A), DepartmentId (B), Number (C), Name (D), Credits (E), GenEd (F)

- Primary Key: CourseId (A)
- (Minimal) Functional Dependency:

  A → B, C, D, E, F

The FD obeys the 3NF conditions where the left-hand side is the primary key and the right-hand side is all the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are fully functionally dependent on CourseId (A).

Departments Entity:
DepartmentId (A), Name (B), CollegeUnit (C)

- Primary Key: DepartmentId (A)
- (Minimal) Functional Dependency:

  A → B, C

The FD obeys the 3NF conditions where the left-hand side is the primary key and the right-hand side is all the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are fully functionally dependent on DepartmentId (A).

CourseDescription Entity:
CourseId (A), Description (B), Instructor (C)

- Primary Key: CourseId(A)
- (Minimal) Functional Dependency:

  A → B, C

The FD obeys the 3NF conditions where the left-hand side is the primary key and the right-hand side is all the dependent attributes. The entity is in 2NF because all attributes are atomic and all non-key attributes are fully functionally dependent on CourseId (A).

Each entity in the schema adheres to 3NF because there are no partial or transitive dependencies present and all attributes are dependent on the primary key or superkey. Thus all the functional dependencies are preserved and there is no information loss due from decomposition.

5. Logical Design (relational schema)

```
User(UserId: Varchar(31) [PK], Email: Varchar(63), EncryptedPassword:
Varchar(255), DateJoined: DATE, MostGuessed: Int [FK to Courses.CourseId],
AvgGuesses: Decimal)

Guess(GuessId: Int [PK], UserId: Varchar(31) [FK to User.UserId], Date: DATE
[FK to DailyClass.Date], CourseId [FK to Courses.CourseId], Value: INT)

DailyClass(Date: DATE [PK], CorrectCourseId: Int [FK to Courses.CourseId],
MostCommonWrong: Int, TotalAvgGuesses: Decimal)

Courses(CourseId: Int [PK], DepartmentId: Int [FK to
Departments.DepartmentId], Number: Int, Name: Varchar(255), Credits: Int,
GenEd: VarChar(63))

Departments(DepartmentId: Int [PK], Name: Varchar(63), CollegeUnit:
Varchar(63))

CourseDescription(CourseId: Int [PK], Description: Varchar(1023), Instructor:
Varchar(255))
```