PT1 Stage 2

UML Diagram Explanation:

For our database design, we have the entities User, Room, GameHistory, Transaction, and Skin.

The User entity keeps track of all general user information, including attributes such as userid(UID), password, and balance. This is kept as a separate entity because general user information needs to be grouped together into one entity and cannot be divided into individual attributes for other entities because those entities contain a lot more specific information, such as information relating to a specific Room. The User entity is connected to all other entities, and their individual relationships will be discussed later.

The Skin entity will store all skins for sale in our shop. Each user will be able to buy the same skin for the same price. Because many users can own the same skin and one user can own many skins, this will be a many-to-many relationship. The Inventory will be the relationship between the User and Skin entities, keeping track of what skins each user owns.

The GameHistory entity will store the history of all moves made by each user in every hand they play. There will be a unique ID for each hand, but if there are n users playing in the hand, there will be n tuples with the same HandID, but different UserIDs. Similarly if a user plays n hands, there will be n tuples with the same UserID but different HandIDs. Because of this, we use both HandID and UserID as the primary key. Additionally, because every game history belongs to a specific user and since a user can have multiple gamehistories (if they play multiple hands) users have a many-to-one relationship. The games played will be the relationship between the User and GameHistory entities, keeping track of what histories belong to each user.

The Room entity will keep track of all current rooms and their logs and chat logs. We want to keep track of open rooms, but we also don’t want to store closed rooms. The Log and ChatLog attributes are strings of the file path to the file that needs to be read/updated. For User and Room, since each room can have multiple users at once, but each individual user can only be in one room at a time, they have a one-to-many relationship. The current will be the relationship between the User and Room entities, keeping track of what room each user is currently in.

Transaction will store all currency changes made. This includes betting in a hand, winning pots, sending people money, and buying skins from the shop. When betting in a hand or winning a pot, the sender or receiver will respectively have UID as HandID. Since each transaction either involves a sender and a receiver (two parties), or a HandID and a user (one party), each transaction references one to two Users. Additionally, each user can have many different transactions so we are defining this as a many-one/two relationship. The affects balance will be the relationship between the User and Transaction entities, keeping track of what transactions affect each user.

Normalized Database(3NF):

User(UID, Password, Balance, CurrentSkin, RoomID)

- The User entity is 3NF as it meets the following criteria:
    - It matches 1NF
        - All attributes of this table contain atomic values
        - There can be no row which contains multiple values per attribute
    - It matches 2NF
        - User is in 1NF
        - All non key attribute values are dependent on the primary key (UID, RoomID) as their values are all specific and dependent on this composite key pair. In this case, we group attributes RoomID and UID to be our primary keys because the attribute RoomID by itself is not dependent on UID. Therefore, because a user is not constrained to only visit one room ever, grouping both attributes to be the primary key allows us to have entries of the user in any possible room, albeit one room at a time.
    - It matches 3NF
        - User is in 2NF
        - There are no transitive dependencies. No non key attribute in this table determines any other non key attribute value. They are all solely dependent on the primary key pair.

Room(RoomID, Log, ChatLog)

- The Room entity is 3NF as it meets the following criteria:
    - It matches 1NF
        - All attributes of this table contain atomic values
        - There can be no row which contains multiple values per attribute
    - It matches 2NF
        - Room is in 1NF
        - All non key attribute values are dependent on the primary key RoomID. The Room log and chat log are solely dependent and specific to the Room it is associated to
    - It matches 3NF
        - Room is in 2NF
        - There are no transitive dependencies as the chat log and log do not determine each other

GameHistory((HandID, UID), DateTime, buyin, blinds_level, …, result, balance)

- The GameHistory entity is 3NF as it meets the following criteria:
    - It matches 1NF
        - All attributes of this table contain atomic values
        - There can be no row which contains multiple values per attribute
    - It matches 2NF

        - GameHistory is in 1NF
        - All non-key attribute values are dependent on the primary key (HandID, UID) as their values are all specific and dependent on the game that is being played. Essentially, a specific instance of a game played tells us the in-game statistics. Additionally, because the HandID is not dependent on the game UID, we group these two attributes together as primary keys.
    - It matches 3NF
        - GameHistory is in 2NF
        - There are no transitive dependencies. No non key attribute in this table determines any other non key attribute value. They are all solely dependent on the primary keys.

Transaction(TransactionID, SenderUID, ReceiverUID, Amount, DateTime, Description)

- The Transaction entity is 3NF as it meets the following criteria:
    - It matches 1NF
        - All attributes of this table contain atomic values
        - There can be no row which contains multiple values per attribute
    - It matches 2NF
        - Transaction is in 1NF
        - All non key attribute values are dependent on the primary key TransactionID as their values are all specific and dependent on the transaction being completed
    - It matches 3NF
        - Transaction is in 2NF
        - There are no transitive dependencies. No non key attribute in this table determines any other non key attribute value. They are all solely dependent on the primary key.

Inventory(UID, SkinID)

- The Inventory relationship is 3NF as:
    - Contains only atomic values
    - No non key attributes
    - There are no transitive dependencies

Skin(SkinID, Image, Description):

- The Skin entity is 3NF as it meets the following criteria:
    - It matches 1NF
        - All attributes of this table contain atomic values
        - There can be no row which contains multiple values per attribute
    - It matches 2NF
        - Skin is in 1NF
        - All non key attribute values are dependent on our primary key SkinID because the image and description are dependent on what the skin is
    - It matches 3NF

- Skin is in 2NF
- There are no transitive dependencies. Image does not determine Description, and vice versa.

Logical Design/Relational Schema:

User(UID: VARCHAR(255), RoomID: VARCHAR(255) [FK to Room.RoomID], Password: VARCHAR(225), Balance: INT, CurrentSkin: VARCHAR(255) [FK to Skin.SkinID], (UIC, RoomID) [PK])

Room(RoomID: VARCHAR(255) [PK], Log: VARCHAR(255), ChatLog: VARCHAR(255))

GameHistory(
    HandID: VARCHAR(255),
    UID: VARCHAR(255),
    DateTime: DATETIME ,
    buyin: REAL,
    blinds_level: INT,
    init_stack: INT,
    position: INT,
    action_pre: VARCHAR(255),
    action_flop: VARCHAR(255),
    action_turn: VARCHAR(255),
    action_river: VARCHAR(255),
    all_in: BOOLEAN,
    cards: VARCHAR(255),
    board_flop: VARCHAR(255),
    board_turn: VARCHAR(255),
    board_river: VARCHAR(255),
    pot_pre: INT,
    pot_flop: INT,
    pot_turn: INT,
    pot_river: INT,
    ante: INT,
    blinds: INT,
    bet_pre: INT,
    bet_flop: INT,
    bet_turn: INT,
    bet_river: INT,
    result: VARCHAR(255),
    balance: INT,
    (HandID, UID) [PK]
)

Transaction(TransactionID: VARCHAR(255) [PK], SenderUID: VARCHAR(255) [FK to User.UserID], ReceiverUID: VARCHAR(255) [FK to User.UserID], Amount: INT, DateTime: DATETIME, Description: VARCHAR(255))

Skin(SkinID: VARCHAR(255) [PK], Image: VARCHAR(255), Description: VARCHAR(255))

Inventory(UserID: VARCHAR(255) [FK to User.UserID], SkinID: VARCHAR(255) [FK to Skin.SkinID])

Fix all suggestions for the previous stages (-2% for missing requirements from previous stages)

-1: From the proposal, it is unclear how the datasets will be utilized in the project
-1: Functionality section is missing discussion on CRUD operators

How we will apply the datasets into our project:

1. https://www.kaggle.com/datasets/murilogmamaral/online-poker-games

At a high level, we use the above dataset to demonstrate our product's ability to comprehensively simulate a gamesuite environment. Through populating tables with both auto-generated and dataset derived data, we show that our product's capable of supporting both in-game operations as well as other functionalities in our gamesuite environment. Populating our tables with pre-selected data essentially serves as a proof of concept to show that our product can track user relevant, game relevant, and other attribute relevant information (such as transactions).

We use our first dataset in the following manner:

- This dataset represents simulated Poker games in which 35 columns of in-game relevant data are stored (such as buy-in, blinds, balance, result, etc.)
- We use the information stored in this dataset to populate our GameHistory table, which essentially represents the game history from start to finish by tracking various in-game metrics
- By doing this, we are essentially "simulating" end-to end Poker games and verifying that when users are running games in real time, the relevant in-game data will be collected and stored properly

Our second dataset will be auto-generated but follows a similar pattern:

- Our auto-generated data can represent out-of-game functionality such as transaction history, user management, room management, and more.
- For example, we can auto generate relevant entries that we insert into our Skins, Users, Room, and/or Transaction table. By auto-generating the data that we add into our tables, we now not only simulate and test in-game operations, but additionally out-of-game functionalities such as transactions and user profile updates. Populating data into our

tables not only validates via simulation that our in/out of game functionality works, but allows us to generate more data (i.e., simulating transactions from existing user data adds entries to the transactions table), contributing to expanding our current dataset(s).

Functionality Section: (CRUD)

When users access our platform, they will be able to create their own account, create a new room to play with their friends, and create new transactions through actions such as playing games or making new transfers of currency to other users. Users can also delete their account and delete their rooms automatically when every user in that room leaves. Users can update their password and then update their balance through actions such as playing games, buying items, and sending/receiving currency. Finally, users can search for a new room to join, search through their past transactions, and the history of the games they played as well as the stats on those games.