JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Poker Room Database

Course Section: CS605.641.81
Fall, 2024

Prepared by

Koushik Parakulam
11/20/2024

Table of Contents

1. Introduction

Poker has always been the one gambling game I have always enjoyed engaging in, as the strategy of the game allows it to not be merely a game of chance. In recent years I have become more interested in the theoretical aspects of the game, the optimal way to play against certain players, and how to structure your style to seek out an advantage. This accompanied with my interest in machine learning always made me curious as to whether I could design a bot that could play games of poker, and then collect statistics on the data from those games to design strategies that could improve my own game. As such, in line with my greater interest to implement such a project, I decided to try to implement a poker room database in line with this purpose.

1.1. Scope and Purpose of Document

The purpose of the poker room database is not merely as a means of storage of obtained poker data, but to represent the complete design of all the game play logic, the evaluation of game states, determining when to produce records, etc. Thus it is structured in line with how an online poker room might be designed, holding player information, game information, the game account information, and populating the records in all the entities by itself simply by the user playing within the game.

As such, this document discusses the complete design, implementation, and conceptual workings in constructing a complete poker database. In doing, so we explain the functional requirements of our database, the way by which the database was designed, the construction of the entities to serve the functional requirements, and the integrity/security constraints placed within the design to secure the relations between the collected game data. After doing so, we will characterize the actual implementation of the database, the types of procedures and queries performed, hypothetical queries that could be performed to access specific information, etc.

Project Objective

The primary objective of this project is to implement a self-contained poker room database that will implement the logical design of game play, and produce records based on that game play while the games are being played by users of the database. The users of the database will be able to buy into and out of a game account, place bets on the games, and observe their past games, the money they have made, etc. In the background, the database will populate the games themselves, work out how the games are being played, keep track of the monetary/card transactions that are occurring, and query these results into their respective tables.

In order to implement the logical design of the database itself, we created a simpler modified version of poker, to make it easier to decipher game states. Since the bulk of our database design involved constructing the logical design of the game, we will discuss the rules and objective of the modified version of our poker game below.

The Rules of the Modified Poker Game

The poker game is initially defined by a maximum number of players and limit amount which determines the maximum amount a player can bet for the game. The players will start off with an arbitrary amount of money that they bought in for, and after there are two or more players, the game begins.

The game consists of an arbitrary number of rounds and each round has four or less subrounds (Pre-flop, Flop, Turn, and River). For a given round, each player receives a random card numbered one through ten, after which the subrounds begin.

Each of the subrounds consists of laying out a certain number of community cards from the deck consisting of numbers from one to ten. For both the player and community cards, the cards will have duplicates.

The number of cards for each subround is as follows,

Pre-Flop - 0 Cards
Flop - 3 Cards
Turn - 1 Cards
River - 1 Cards

On each subround, the players are comparing their cards to the community cards and either doing a ‘bet’ where they bet the limit amount of the game, ‘fold’, meaning they quit the subround and entire round losing all previous bets made, or if the total amount of money they have is less than the limit amount, can choose to go ‘All-In’ and make a bet less than the limit amount. When all the players have chosen to make their action, the Subround ends and goes to the next.

If at any point all but one player remains due to all the others folding, that player wins all the previous money put into the various subrounds of that round.

Otherwise, if the players remain until the end of all the Subrounds or if the remaining players in the subround have gone “All-In”, the winner of the game is determined based on the players cards and the community cards. Furthermore, in the event that the players have all gone All-In before the river, the remaining community cards are first placed, before the winner is determined.

Determining the Winner

The winner is determined by identifying the player with the highest even or odd total with regards to their card and the community cards.

For example, if the player has an odd numbered card, their total is determined by adding their card to all the odd community cards. If their card is even, their total is determined by adding their card to all the even numbered cards. For each player, all their total card values are totaled and the individual with the highest total wins all previous bets.

In the case of ties, the money is split amongst the different individual players.

2. System Requirements

The system requirements for our database is minimal, but there are still several requirements needed in order to effectively use our database.

2.1 Hardware Requirements

For the specifications of our database the hardware requirements are as follows:

Processor (CPU):

- Minimum: Dual-core processor

Memory (RAM):

- Minimum: 8 GB,

Disk Storage:

- Minimum: 5 GB however, more is always recommended as the database size increases with play

Network:

- For remote access a high speed internet connection is recommended.

2.2 Software Requirements

Operating System:

- Windows 10, 11, Server 2016, Server 2019
- Linux can also possibly used, but it was implemented with Windows

MySQL Version:

- The database was created through MySQL version MySQL 8.0
- It is necessary to have both the MySQL Workbench, Client, and Server installed.
- Ensure that mysqldump or other 3rd party backup software is installed

2.3 Functional Requirements

With regards to the poker room database, there are a whole host of functional requirements that it must serve. These functional requirements enable the tracking of monetary transactions, game-play behavior, and player statistics. The most important functional requirements are listed below:

Instill the Poker Game Play
By far the most important requirement is for the database to instill the entire logic of the poker game. This includes determining the creation of a game, round, and subround, determining what to do when a player makes a bet, folds, or goes all in, finding out the end of a round, determining winners based on the community/player cards, etc.

Register New Player Accounts
The database should allow for a new player to be added into the database and capture their basic personal information and their financial information. This would then allow them to buy into games and play through an intermediary game account. However, this functionality need not be directly referenced by the potential players of the game, but admins controlling the database.

Create New Game Accounts
Given an existing player, the database should enable the creation of a game account that keeps track of in-game player statistics. There should only be one active game account for a player at any given time, and when the player clears their game account, the account should close, and no longer be associated with the active game.

Allow Player to Buy Into a Game
The existing player should be allowed to buy into a game with the system checking whether there exists an open game, with open seats for the player, allowing for the game account to be created.

If a game is open and available, the amount of money the player wants to buy in for, should transfer from the player to the newly created game account.

Allow Player to Buy Out of a Game
Furthermore, the player should be able to buy-out of a game, transferring back any value earned within the game account, back to the player account. If the buy out clears the game account, the player should be quit out of the game, and their account should be nullified.

Allow Player to Place Bets
Within a game, players should be able to place bets during their turn, with the system checking for sufficient balance of the game account and updating the bet amount accordingly. They should not be allowed to place bets in the Round if they have folded, quit, or went all-in, and should not be able to re-bet in a Subround.

Pay Out Winning Players
Within a game, the bets of different players should be pooled by the system for each subround of a given round of a game. At the end of the round, the winning player should be identified by the system and pay out the winnings to the game account of the player.

In the case that there are multiple winners, a split pot should be made by the system and pay them out accordingly.

Obtain a Rake from Games
Within a round of a game, a portion of the pot at specific subrounds should be allocated for the “rake” by the system, i.e. the tax on the pot. This rake should then be transferred to the system register as profit for the poker system.

Track All Transactions
The system should track all the transactions including the rake, pay out, bets placed, buy-ins, and buyouts. The amount of the given transaction, the time of the transaction, and the individuals/accounts/game involved should all be recorded by the system.

Track Games, Rounds, and Subrounds
The system should keep track of individual games that are going on and have already been completed, along with the rounds that were played in the game, and the subrounds that occurred within each round.

The system should keep track of the cards seen within each of those subrounds, how they changed within each round, and timing of game, round, and subround start and end times. These statistics should be used by the database in determining when a Game, Round, and Subround has been completed, how the pay out should be made, and various other aspects of the game.

Track Player Hands
The hands associated with a player of a game account should be tracked by the system, along with the chip stack of the player and their position on the table.

Track Winners/Losers
The system should be able to obtain the winners and losers of a specific round, if not explicitly, through other information contained within the betting and pay-outs.

2.4 Database Requirements

MySQL workbench was the RDBMS used for this project, specifically version 8.0, as mentioned earlier.

3. Database Design Description

To address our prior functional requirements we designed our poker room database to capture relations between four distinct strong entity groups, the poker game, the player, the game account, and the system register. The game group would handle distinctions between the game itself, the game account would handle interactions between the player and the game, and finally the system register would interact with just the game.

The game group was designed under a strict hierarchical relationship with the poker game, as the parent at the top followed by the round, and the subround, with each consecutive entity being a child of the previous respectively. The game account acts as a central group that would interact with both the game/round and the player. Finally the system register would interact with the game solely through associations with the subround of a game.

In this sense, we consider the player and system register as isolated groups interacting with the rest of the database only through one relationship, either to the game account for

the player and game, or the subround for the system register. The relationships between these groups of strong entities are made through junction tables (associative entities) which act as intermediaries capturing the information of a group to group interaction.

To capture these relationships an ER diagram was used to create the necessary entities, the attributes needed for those entities to model our requirements, and the interacting relationships with these entities. We used a combination of reference entities, strong entities, and associative entities, and a majority of our relationships were non-identifying in nature, and minimized the use of composite keys.

3.1 Design Rationale

ER Notational Rationale
The database was designed and modeled using IE notation instead of Chen's notation and enhanced entity diagrams were not used in our database design. It was found necessary to use IE notation over Chen's notation as it enabled a more detailed overview of the relationships between tables, the attributes they contained, and the types of entities involved in each of the relationships. Chen's notation was deemed too cumbersome to model, as with the large number of attributes, the design would be muddled and hard to follow.

Artificial Primary Keys Rationale
Our design employed the heavy use of artificial primary keys (surrogate keys) along with non-identifying relationships. The choice to use artificial primary keys was to avoid having to determine whether there were any attributes that would remain unique regardless of the possible variations that were imputed into the database. As such, we wanted to ensure unique identification of every entity and not have to depend on a specific attribute having to always be unique. The exception for these were some of the associative entities which used key composites with no unique identifier other than the entities keys which it referenced.

Non-Identifying Relationships Rationale
The choice of using mostly non-identifying relationships was made to reduce the complexity of composite key migrations, and not having child entities hold large amounts of foreign keys, whenever possible. Furthermore, it was deemed that using non-identifying relationships would make it easier to normalize our database design to fit the second normal form. This is because in the second normal form, it is required that non-key attributes fully depend on the entire primary key. With large composite keys, it would be difficult to verify whether every non-key attribute depends on the entire composite, and thus, it was avoided whenever possible. The exception to this was with

associative entities which were always identifying in nature as the added composite complexity was easier to manage.

Reference/Associative Entities
Our design also employed the use of reference entities and associative entities to enforce integrity to our database design. Reference entities were employed heavily for data that would be repeatedly used by different entities, and would serve the purpose as a look up table. Associative entities on the other hand was used for many to many relationships to prevent redundant stores of repeating records between the entities. These associative entities were used as stores for transaction data within the game, tracking monetary and card transaction information.

3.2 ER Model

Our database E/R design used a total of eighteen total entity tables. Out of these eighteen, there were a total of seven strong entities, five associative entities, and six reference entities.

For our strong entities, our game group consisted of three entities game, round, and subround and there was simply one entity for the groups game_account, player, and system register, named Game_Account, Player, and System_Register.

Our reference entities consisted of five entities, All_Seat_Types, All_Card_Types, All_Subround_Types, All_Limit_Types, and All_Bet_Types.

Our associative entities consisted of Buy_In, Buy_Out, Make_Bet, Pay_Out, Rake_Out, Players_Cards, and Community_Cards.

3.2.1 Entities

Game (Strong Entity)

The parent of the game group of entities which records the parameters of the game including the maximum seat size of the game, the type of game of poker, the recorded time of opening, and closure of the game.

A new game record is created when there are no available games, and a player wants to play in a game, and it is closed when all the players leave the game.

The game entity's primary key is a foreign reference in round (direct child) and Game_Account.

Attributes: Game_ID (primary key), Limit_Type_ID, Max_Seats_ID (foreign key), Start_Time, End_Time.

Round (Strong Entity)

The direct child of the game entity and records all the rounds of a particular game, and the start/end of a round.

The first round record is created for a given game, when there are sufficient game accounts associated with a game (>=2 game accounts), and ends after a particular subround (river subround) is completed (in reference to the round), or if the subround ends prematurely due to either a player having won the particular round or all the players having left the game.

The round entity's primary key is a foreign reference in subround (direct child) and Players_Cards. The round entity is used by the Players_Cards entity to generate new card records for all players associated with a game, each new round. When a new round begins, all the players associated with the game, will have new card records (corresponding the the number of cards associated with the game type) generated.

Attributes: Round_ID (primary key), Game_ID (foreign key), Start_time, End_time

Subround (Strong Entity)

The direct child of the round entity and records the subround type (pre-flop, flop, turn, river) by reference, the start/end time of a given subround, and the pot amount, accumulated from players betting into the game.

The first subround record is created immediately after a round starts, and it ends after all the players have placed their bets, have won the current round, or if the players have left the game.

The subround entity's primary key is a foreign reference in Rake_Out, Pay_Out, Make_Bet, and Community_Cards.

The subround is used as reference by the Community_Cards entity in order to associate a given community card with a given subround type, and produce the number of community card records corresponding to the subround type.

Attributes: Sub_Round_ID (primary key), Round_ID (foreign key), Sub_Round_Type_ID (foreign key), Start_Time, End_Time

System_Register (Strong Entity)

A strong “isolated” entity, used for storing the poker rooms monetary balance. The System_Register’s balance is updated through rakes collected throughout the game. These rakes are a tax collected on each bet from the pot of the subround, and are then transacted to the System_Register through the Rake_Out associative entity.

As such, the System_Registers primary key is a foreign reference in only the Rake_Out entity.

Attributes: Register_ID (primary key), Account_Balance

Player (Strong Entity)

A strong “isolated” entity used for storing records of all the players joined onto the online poker site. Records of basic player information, financial information, and the account balance of the player from the poker site, is stored.

The player entity only interacts with the Game_Account entity through the associative entities Buy_In and Buy_Out. Thus, the player’s primary key is a foreign reference in only the Buy_In and Buy_Out associative entities.

Attributes: Player_ID (primary key), Name, Address, Age, Account_Creation_Date, Credit_Card_Number, Account_Balance.

Game_Account (Strong Entity)

The centralizing entity that interacts with the game tables and player tables. A new Game_Account record is generated through a transaction control procedure whereby for a given Buy_In transaction, the first open game is found, and if there are no open games, a new game record is generated. From here, a new Game_Account record is generated with the Buy_In amount as the chip stack, the Game_ID, and the time of creation. After this, the Buy_In transaction is generated with the newly created Game_Account_ID.

The Game_Account’s primary key is also stored as a foreign reference in Buy_In, Buy_Out, Player_Cards, Make_Bet, and Pay_Out entities. The Buy_Out entity uses the Game_Account_ID as reference to record transactions made by the player to transfer money back into their player account. As for the Player_Cards

entity, the Game_Account_ID is used to link the generated card records for a given round, to the player accounts associated with said round.

The Make_Bet entity references the Game_Account to associate player made bets for a given subround, and records outgoing transactions associated with the Game_Account balance when a bet is placed. The Pay_Out entity performs the same but in reverse recording incoming transactions associated with the Game_Account balance when the player wins a particular round.

Attributes: Game_Account_ID (Primary Key), Game_ID (Foreign Key), Chip_Stack, Time_Created, Time_Closed

Weak/Strong Associative Entities

Community_Cards (Strong Associative Entity)

Used for generating records of the community cards associated with a particular subround. These community cards are the cards that the players associated with the round use as reference to place bets. Based on the subround type, a corresponding number of community card records are generated.

There are five possible subround types; “Pre-flop”, “Flop”, “Turn”, and “River”. For each of these subround types, the number of Community_Cards records that are generated are zero, three, one, and one, as following the of our poker variation. In the event that a subround closes out due to a single active player being left after folds, all-ins, or quitting, this rule is broken, and all the community cards for that particular subround are generated in a sweep to allow comparisons to determine a winner.

The actual card types are generated through a reference to the All_Card_Types entity, whereby a random Card_Type_ID is chosen for the record.

Attributes: Community_Card_ID, Card_Type_ID (Primary Key, Foreign Key), Sub_Round_ID (Primary Key, Foreign Key)

Players_Cards

Used for generating records of the in game players cards for each new round of a particular game. For each new round, a card record for each of the players associated with that particular round is generated. When new players join an existing game, cards are generated for them as well, so they can participate in the game.

Attributes: Round_ID (Primary Key, Foreign Key), Game_Account_ID (Primary Key, Foreign Key), Card_Type_ID (Primary Key, Foreign Key)

Rake_Out

Used for recording transactions of the “rake” or tax collected on bets placed during the subrounds. The rake out amount is fixed at a 2.5% of any non all-in bet. Rake_Out transactions can also be made in other rare instances such as all the players leaving the game before a winner could be determined, in which case the pot is allocated to the system register.

Attributes: Sub_Round_ID (Primary Key, Foreign Key), Register_ID (Primary Key, Foreign Key), Rake_Out_Amount, Rake_Out_Time

Pay_Out

Used for recording transactions of the pay out money earned by a winning player(s) for the given subround.

Attributes: Sub_Round_ID (Primary Key, Foreign Key), Game_Account_ID (Primary Key, Foreign Key), Pay_Out_Amount, Pay_Out_time

Make_Bet

Used for recording transactions of the bet made by a player of a given Game_Account for a particular subround. With this, the Make_Bet records the bet type made by the player through a reference to the reference entity All_Bet_Types.

The primary bet types are ‘Bet’, ‘Fold’, and ‘All-In’.

A ‘Fold’ bet type is generated for a Make_Bet transaction record in the situation that the bet made by the Game_Account is equal to zero.

A ‘Bet’ bet type is generated for a Make_Bet transaction record in the situation that the bet made by the Game_Account is equal, greater, or less than the limit amount of the Game limit type.

A ‘All-in” bet type is generated for a Make_Bet transaction record in the situation that the bet made is equal to the total account balance of the Game_Account, but less than or equal to the limit type of the Game.

In the situation that a bet transaction is greater than the available chip stack balance of the Game_Account, or, less than zero, the transaction is rolled back and invalidated.

It is important to note that a player cannot make a bet of any size different that the game limit amount unless the maximum available money in their account is less than the limit. However, if a game account attempts to make such a transaction, the transaction is defaulted to the limit amount.

Attributes: Sub_Round_ID (Primary Key, Foreign Key), Game_Account_ID (Primary Key, Foreign Key), Amount, Time

Buy_In

Used for recording transactions of a player transacting money into an existing or new Game_Account record. In the event of a new Game_Account record, the transaction is halted until the creation of the Game_Account in order to store a reference to the Game_Account as a foreign key. If the Game_Account is pre-existing, the transaction is simply recorded as a new record.

Attributes: Player_ID (Primary Key, Foreign Key), Game_Account_ID (Primary Key, Foreign Key), Buy_In_Amount, Buy_In_Time.

Buy_Out

Used for recording transactions of a player transacting money out of an existing Game_Account record. Unlike the Buy_In entity, the Buy_Out entity can only record transactions of existing Player/Game_Account entity records, and will rollback in the event that either do not exist.

Attributes: Player_ID (Primary Key, Foreign Key), Game_Account_ID (Primary Key, Foreign Key), Buy_Out_Amount, Buy_Out_Time.

Reference Entities

All_Seat_Types

Stores a look up table of maximum seating sizes that is referenced by the Game entity for the purpose of specifying a limit of the size of a particular game. Based on the maximum seating, the size at which the Game will be full or open varies, preventing or allowing a new Game_Account to join an existing Game.

Example seat types are,

(6), (9), (12) …

Attributes: Max_Seats_ID (Primary Key), Max_Seats

All_Card_Types

Stores a look up table of the ten numbered cards used in our poker variation as playing cards, referenced by the Community_Cards and Players_Cards entities for generating records of the cards for a given Subround of the house and Round of the player, respectively.

Example card types are,

(1),(2),(3),(4)…

Attributes: Card_Type_ID (Primary Key), Card_Combination

All_Bet_Types

Stores a look up table of all the Bet types referenced and used by the Make_Bet entity for specifying the type of bet for a given betting record. This in turn is implicitly used for determining a possible winner of a given round/subround, the end of a round, the amount of players remaining in the round, etc.

Example bet types are,

(‘Bet’),(‘Fold’),(‘All-In) …

Attributes: Bet_Type_ID (Primary Key), Bet_Type

All_Limit_Types

Stores a look up table of all the limit bet amounts and used by the Game entity for the purpose of specifying the amount that a bet is during play.

Example limit types are,

(.1),(.5),(.10),(.20) …

Attributes: Bet_Type_ID (Primary Key), Bet_Type

All_Sub_Round_Types

Stores a look up table of all the Subround types referenced and used by the Subround entity for specifying the type of Subround the given round is in. This in turn is implicitly used for determining if the round has ended and a winner needs to be determined, whether a rake transaction needs to occur, the amount of community card records that need to be produced, etc.

Example Subround types are,

('Pre-Flop'), ('Flop'), ('Turn'), ('River') …

Attributes: Sub_Round_Type_ID (Primary Key), Sub_Round_Type

3.2.2 Relationships

Game and All_Limit_Types:

- Game has a many-to-one relationship with All_Limit_Types, where each game can have one and only one limit type (Limit_Type_ID), and zero or multiple games can share the same limit type. The game and All_Limit_Types entities have a non-identifying relationship.

Game and All_Seat_Types:

- Game has a many-to-one relationship with All_Seat_Types, where each game can have one and only one seat type (Max_Seats_ID), and zero or multiple games can share the same seat type. The Game and All_Seat_Types entities have a non-identifying relationship.

Game and Round:

- Game has a one-to-many relationship with round, where each game can have zero or multiple rounds, but each round is associated with one and only one game (Game_ID). The round entity is not necessarily generated immediately with the Game but rather generated once the number of Game_Accounts associated with the given game is >2, hence why the minimum cardinality is zero. The Game and round entities have a non-identifying relationship.

Round and Sub_Round:

- Round has a one-to-many relationship with Sub_Round, where each round can have one or multiple sub-rounds, but each sub-round belongs to one and only one round (Round_ID). The minimum cardinality for the

Sub_Round to the round entity is one as a Sub_Round is generated in parallel with the round entity, and thus, a round must have at least one Sub_Round. The round and Sub_Round entities have a non-identifying relationship.

Sub_Round and All_Sub_Round_Types:

- Sub_Round has a many-to-one relationship with All_Sub_Round_Types, where each sub-round has a one and only one sub-round type (Sub_Round_Type_ID), but zero or multiple sub-rounds can share the same type. The Round and All_Sub_Round_Types entities have a non-identifying relationship.

Game and Game_Account:

- Game has a one-to-many relationship with Game_Account, where each game can have zero or multiple game accounts, but each game account is associated with one and only one game (Game_ID). The **Game and Game_Account** entities have a non-identifying relationship.

Game_Account and Buy_In:

- Game_Account has a one-to-many relationship with Buy_In, where each game account can have one or multiple buy-in transactions, but each buy-in transaction is associated with a one and only one game account (Game_Account_ID). The minimum cardinality for Buy_In to Game_Account is one as there cannot be a Game_Account with no Buy_In transactions present. The Game and Buy_In entities have an identifying relationship.

Game_Account and Buy_Out:

- Game_Account has a one-to-many relationship with Buy_Out, where each game account can have zero or multiple buy-out transactions, but each buy-out transaction is associated with a one and only one game account (Game_Account_ID). Game_Accounts can be associated with no Buy_Out records thus, the minimum cardinality is zero. The Game and Buy_Out entities have an identifying relationship.

Player and Buy_In:

- Player has a one-to-many relationship with Buy_In, where each player can have zero or multiple buy-in transactions, but each buy-in transaction is associated with a one and only one player (Player_ID). The Player and Buy_In entities have an identifying relationship.

Player and Buy_Out:

- Player has a one-to-many relationship with Buy_Out, where each player can have zero or multiple buy-out transactions, but each buy-out transaction is associated with a one and only one player (Player_ID). The Player and Buy_Out entities have an identifying relationship.

Game_Account and Make_Bet:

- Game_Account has a one-to-many relationship with Make_Bet, where each game account can have zero multiple bets, but each bet is associated with a one and only one game account (Game_Account_ID). The Game_Account and Make_Bet entities have an identifying relationship.

Sub_Round and Make_Bet:

- Sub_Round has a one-to-many relationship with Make_Bet, where each sub-round can have zero or multiple bets, but each bet is associated with a one and only one sub-round (Sub_Round_ID). The Sub_Round and Make_Bet entities have an identifying relationship.

All_Bet_Types and Make_Bet:

- Make_Bet has a many-to-one relationship with All_Bet_Types, where each bet has a one and only one bet type (Bet_Type_ID), but zero or multiple bets can share the same bet type. The All_Bet_Types and Make_Bet entities have an non-identifying relationship.

Round and Players_Cards:

- Round has a one-to-many relationship with Players_Cards, where each round can have zero or multiple player card entries, but each player card entry is associated with one and only one round (Round_ID). The Round and Players_Cards entities have an identifying relationship.

Game_Account and Players_Cards:

- Game_Account has a one-to-many relationship with Players_Cards, where each game account can have zero or multiple player card entries, but each player card entry is associated with one and only one game account (Game_Account_ID). The Game_Account and Players_Cards entities have an identifying relationship.

All_Card_Types and Players_Cards:

- Players_Cards has a many-to-one relationship with All_Card_Types, where each player card entry has a one and only one card type (Card_Type_ID), but zero or multiple player card entries can share the same card type. All_Card_Types and Players_Cards have an identifying relationship.

Sub_Round and Community_Cards:

- Sub_Round has a one-to-many relationship with Community_Cards, where each sub-round can have zero or multiple community card entries, but each community card entry is associated with a one and only one sub-round (Sub_Round_ID). The Sub_Round and Community_Cards entities have an identifying relationship.

All_Card_Types and Community_Cards:

- Community_Cards has a many-to-one relationship with All_Card_Types, where each community card entry has a single card type (Card_Type_ID), but multiple community card entries can share the same card type. The All_Card_Types and Community_Cards entities have an identifying relationship.

System_Register and Rake_Out:

- System_Register has a one-to-many relationship with Rake_Out, where each system register can have zero or multiple rake-out transactions, but each rake-out transaction is associated with a one and only one system register (Register_ID). The System_Register and Rake_Out entities have an identifying relationship.

Sub_Round and Rake_Out:

- Sub_Round has a one-to-many relationship with Rake_Out, where each sub-round can have zero or multiple rake-out transactions, but each rake-out transaction is associated with a one and only one sub-round (Sub_Round_ID). The Sub_Round and Rake_Out entities have an identifying relationship.

Sub_Round and Pay_Out:

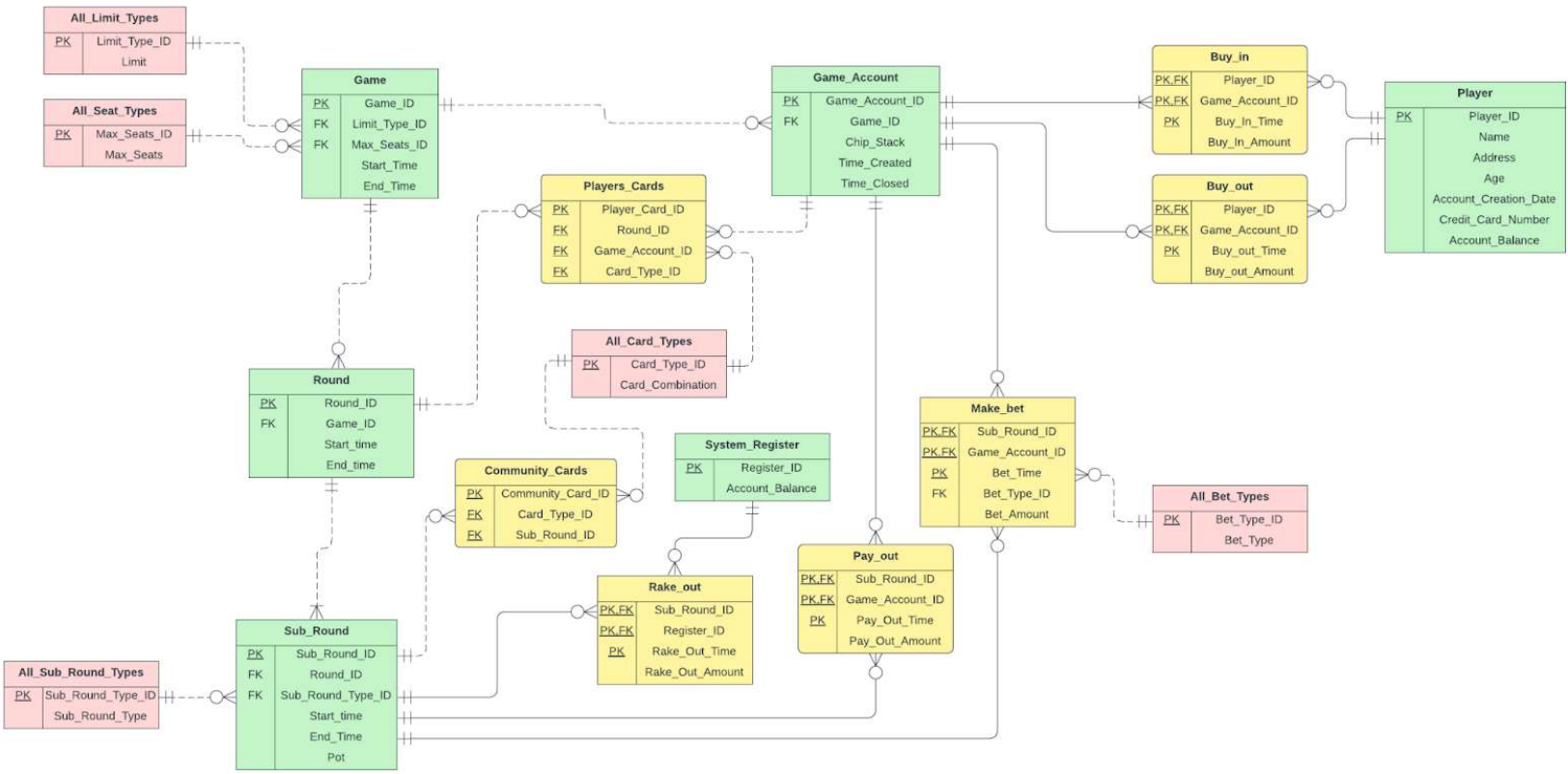
- Sub_Round has a one-to-many relationship with Pay_Out, where each sub-round can have zero or multiple pay-out transactions, but each pay-out transaction is associated with a one and only one sub-round (Sub_Round_ID). The Sub_Round and Pay_Out entities have an identifying relationship.

Game_Account and Pay_Out:

- Game_Account has a one-to-many relationship with Pay_Out, where each game account can have zero or multiple pay-out transactions, but each pay-out transaction is associated with a one and only one game account (Game_Account_ID). The Game_Account and Pay_Out entities have an identifying relationship.

3.2.3 ER Diagram

Our E/R diagram can be seen below with the strong entities in green, the associative entities in yellow, and the reference entities in red. Furthermore, all solid lines represent identifying relationships with the dashed being non identifying in nature.



3.3 Relational Model

Below we will discuss the data dictionaries, integrity constraints, operational rules, and major operations of our database.

3.3.1 Data Dictionary

GAME_ACCOUNT

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Game_Account_ID	Game Account Identifier	INT	32 bit	Primary Key	Y	Positive integers
Game_ID	Game Identifier	INT	32 bit	Foreign Key	N	Positive integers
Chip_Stack	Chip Stack	DECIMAL (7,3)	64 bit	-	N	Rounded Floats
Time_Created	Time Account Created	TIMESTAMP (6)	64 bit	-	Y	DateTime
Time_Closed	Time Account Closes	TIMESTAMP (6)	64 bit	-	N	DateTime

SUBROUND

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Subround_ID	Subround Identifier	INT	32 bit	Primary Key	Y	Positive integers
Round_ID	Round Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Sub_Round_Type_ID	Sub Round Type Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Pot	Accumulated Money	DECIMAL (7,3)	32 bit	-	Y	Rounded Floats
Start_Time	Start Time	TIMESTAMP (6)	64 bit	-	Y	DateTime
End_Time	End Time	TIMESTAMP (6)	64 bit	-	N	DateTime or NULL

ROUND

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Round_ID	Round Identifier	INT	32 bit	Primary Key	Y	Positive integers
Game_ID	Game Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Start_Time	Start Time	TIMESTAMP (6)	64 bit	-	Y	DateTime
End_Time	End Time	TIMESTAMP (6)	64 bit	-	N	DateTime or NULL

GAME

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Game_ID	Game Identifier	INT	32 bit	Primary Key	Y	Positive integers
Limit_Type_ID	Limit Type Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Max_Seats_ID	Max Seats Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Start_Time	Start Time	TIMESTAMP (6)	64 bit	-	Y	DateTime
End_Time	End Time	TIMESTAMP (6)	64 bit	-	N	DateTime or NULL

PLAYER

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Player_ID	Player Identifier	INT	32 bit	Primary Key	Y	Positive integers
Player_Name	Player Name	VARCHAR	50 chars	-	Y	Text
Address	Address	VARCHAR	50 chars	-	N	Text
Age	Age	INT	32 bit	Check (>=21)	Y	Positive integers
Account_Creation_Date	Account Creation Date	TIMESTAMP (6)	64 bit	-	Y	DateTime
Credit_Card_Number	Credit Card Number	BIGINT	64 bit	-	N	Positive integers
Account_Balance	Account Balance	DECIMAL (7,3)	64 bit	-	N	Rounded Floats

SYSTEM_REGISTER

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Register_ID	Register Identifier	INT	32 bit	Primary Key	Y	Positive integers
Account_Balance	Account Balance	DECIMAL (7,3)	64 bit	-	N	Rounded Floats

COMMUNITY_CARDS

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Community_Card_ID	Community Card Identifier	INT	32 bit	Primary Key	**Y**	Positive integers
Card_Type_ID	Card Type Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Sub_Round_ID	Sub Round Identifier	INT	32 bit	Foreign Key	Y	Positive integers

PLAYERS_CARDS

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Player_Card_ID	**Player Card Identifier**	**INT**	**32 bit**	Primary Key	**Y**	**Positive Integers**
Round_ID	Round Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Game_Account_ID	Game Account Identifier	INT	32 bit	Foreign Key	Y	Positive integers
Card_Type_ID	Card Type Identifier	INT	32 bit	Foreign Key	Y	Positive integers

BUY_IN

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Buy_In_Time	Buy In Time	DATETIME	64 bit	Composite Primary Key	Y	DateTime
Game_Account_ID	Game Account Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Player_ID	Player Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Buy_In_Amount	Buy In Amount	DECIMAL (7,3)	32 bit	Check(>0)	Y	Rounded Floats >0

BUY_OUT

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Buy_Out_Time	Buy Out Time	DATETIME	64 bit	Composite Primary Key	Y	DateTime
Game_Account_ID	Game Account Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Player_ID	Player Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Buy_Out_Amount	Buy Out Amount	DECIMAL (7,3)	32 bit	Check(>0)	Y	Rounded Floats >0

RAKE_OUT

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Rake_Time	Rake Time	DATETIME	64 bit	Composite Primary Key	Y	DateTime
Register_ID	Register Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Sub_Round_ID	Sub Round Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Rake_Amount	Rake Amount	DECIMAL (7,3)	32 bit	-	Y	Rounded Floats

PAY_OUT

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Pay_Out_Time	Pay Out Time	DATETIME	64 bit	Composite Primary Key	Y	DateTime
Sub_Round_ID	Sub Round Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Game_Account_ID	Game Account Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Pay_Out_Amount	Pay Out Amount	DECIMAL (7,3)	32 bit	-	Y	Positive or zero

MAKE_BET

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Bet_Time	Bet Time	DATETIME	64 bit	Composite Primary Key	Y	DateTime
Sub_Round_ID	Sub Round Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Game_Account_ID	Game Account Identifier	INT	32 bit	Composite Primary Key, Foreign Key	Y	Positive integers
Bet_Amount	Bet Amount	DECIMAL (7,3)	32 bit	Check(>=0)	Y	Rounded Floats

ALL_BET_TYPES

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Bet_Type_ID	Bet Type Identifier	INT	32 bit	Primary Key	Y	Positive integers
Bet_Type	Bet Type	VARCHAR	20 chars	Unique	Y	Text

ALL_SEAT_TYPES

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Max_Seats_ID	Max Seats Identifier	INT	32 bit	Primary Key	Y	Positive integers
Max_Seats	Maximum Seats	INT	32 bit	Check (>=0)	Y	Positive integers

ALL_LIMIT_TYPES

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Limit_Type_ID	Limit Type Identifier	INT	32 bit	Primary Key	Y	Positive integers
Limit	The Bet Amount	INT	32 bit	Check (>0)	Y	Positive integers

ALL_SUBROUND_TYPES

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Subround_Type_ID	Subround Type Identifier	INT	32 bit	Primary Key	Y	Positive integers
Subround_Name	Subround Name	VARCHAR	20 chars	Unique	Y	Text

ALL_CARD_TYPES

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Card_Type_ID	Card Type Identifier	INT	32 bit	Primary Key	Y	Positive integers
Card_Combination	Card Combination	INT	32 bit	Unique	Y	Integers Positive

3.3.2 Integrity Rules

Several different strategies were employed to maintain the integrity of our online poker database. With regards to mandatory fields, we restricted certain attributes to not being able to be null. This was automatically performed through the use of single/composite primary keys, but we also restricted several non key attributes.

For instance, all the attributes within the reference tables were set to be not null. This would ensure that any reference of the values in these tables were indeed present to prevent record imputation errors downstream. For the other entities, the time stamp attributes were set to be not null, but the end timestamp attributes were allowed to be null. This would ensure that the time of transactions were always recorded to make it easier to associate when these transactions occurred in case of a transactional conflict. However, the end times were by default always set null to allow us to see whether some entity was active or not. Another set of mandatory fields was within the amounts associated with the transactional associative entities, but also the pot attribute within the Subround entity. Furthermore, most of the foreign keys were set to not null, with the exception of the Game_Account entity which allowed it for the purpose of identifying disassociation with the Game entity after game account closure.The last group of mandatory fields could be found within the Player table which included name and age, with the name being not null to allow for player identification and the age field being not null to ensure that the Player was an adult.

The primary modes of data formatting were the use of VARCHAR for textual/numerical combination data, TIMESTAMP(6) for time stamping transactional information, and INT/DECIMAL(7,3) for integer and decimal based values. The TIMESTAMP(6) type was used for extra precision for transactional information to prevent conflicts with determining who was the first transaction of a particular entry.

For data conversion, we used check constraints and unique constraints primarily in the reference tables and associative entities to ensure non duplicated values were imputed, as well as, ensuring that the values imputed were within some specified range. For the Make_Bet transactions for instance, the bet_amount value was constrained to be equal to or above zero to ensure no negative bets would be placed and increase their account value for a bet. The Buy_In and Buy_Out transactions had a similar constraint but the amount had to be above zero, to prevent redundant transactions. The age attribute within Player was the only non-strong entity table that contained a check constraint to ensure that the player was over twenty-one.

Referential integrity was maintained through the use of foreign keys and specifically the delete/update rules for key references. For all non-reference entities whose keys were referenced by another table, we used the cascading rule for both deletion and updates.

This would ensure that upon an update/deletion to a given table, the tables referencing them would have their foreign keys match the new primary key, or their records would be deleted entirely. This in turns preserves referential integrity while at the same time making it easier to propagate changes or deletions across the children tables easily.

For the reference tables whose keys were referenced by another table, we used a restrict rule for both deletion and updates. This would prevent a deletion of a primary key within a reference table so long as there were records present referencing these keys. The choice to use the restrict rule here was made because changes/deletions to the imputed values within the reference tables were deemed to seldom occur, as the values in these tables form the basis of the rules for querying data for poker play, and are relatively static by nature. As such, allowing deletions/updates to occur so easily would cause major downstream impacts that could ruin the integrity of being able to query important values. For example, if a change were easily enabled to be made on the All_Card_Types table, this could affect the way a winner is determined, prevent pay out records from being generated, and affect the nature of the game. As such, the restrict rule would enforce a stricter level of referential integrity to make it harder to delete/update the static and important values within the reference tables.

All Entities Referential Relationships

Make_Bet

- References the Sub_Round_ID of the table Subround for the purpose of associating a bet with a specific subround.
- References the Game_Account_ID of the table Game_Account to link a bet with a specific game account.
- References the Bet_Type_ID of the table All_Bet_Types to specify the type of bet made.

Pay_Out

- References the Sub_Round_ID of the table Subround for associating a payout with a specific subround.
- References the Game_Account_ID of the table Game_Account to link a payout to a specific game account.

Rake_Out

- References the Register_ID of the table System_Register to link a rake transaction to a specific system register.

- References the Sub_Round_ID of the table Subround for associating a rake transaction with a specific subround.

Buy_Out

- References the Game_Account_ID of the table Game_Account to link a buy-out with a specific game account.
- References the Player_ID of the table Player for identifying which player is associated with the buy-out transaction.

Buy_In

- References the Game_Account_ID of the table Game_Account to link a buy-in with a specific game account.
- References the Player_ID of the table Player to identify the player associated with the buy-in transaction.

Players_Cards

- References the Round_ID of the table Round to associate players' cards with a specific game round.
- References the Game_Account_ID of the table Game_Account to link the cards to a particular player's game account.
- References the Card_Type_ID of the table All_Card_Types to specify the type of card held by a player.

Community_Cards

- References the Card_Type_ID of the table All_Card_Types to specify the type of community card.
- References the Sub_Round_ID of the table Subround to associate a community card with a specific subround.

Game

- References the Game_Type_ID of the table All_Game_Types to specify the type of game being played.
- References the Max_Seats_ID of the table All_Seat_Types to define the maximum number of seats in the game.

Round

- References the Game_ID of the table Game to associate a round with a specific game.

Subround

- References the Round_ID of the table Round to link a subround with a particular game round.
- References the Subround_Type_ID of the table All_Subround_Types to define the type of subround.

Game_Account

- References the Game_ID of the table Game to associate a game account with a specific game.

3.3.3 Operational Rules

Game, Round, Subround End_Time Update Hierarchy

1. The poker game group of entities have strict operational rules to enforce a hierarchy on when an non-null End_Time value can be inserted into their records.
2. A Round cannot update an End_Time until a given child Subround record has determined its winner and inserted its own End_Time, after which, the End_Time for the Round is immediately triggered.
3. A Game cannot update an End_Time within its record until the last Round has an inserted End_Time, and all the Game_Accounts associated with the Game have an inserted Time_Closed value.

   These prevent situations where a Game record with an End_Time having Round or Subround records that have no End_Time, maintaining a hierarchy of Subround, Round, and Game closure.

Insertion Constraints Based on Closed Records

1. Make_Bet, Pay_Out, and Rake_Out records can only be made for a Subround/Game_Account that has a null End_Time/Time_Closed value, respectively.
2. Players_Cards records can only be made for a Round/Game_Account that has a null End_Time/Time_Closed value, respectively.

3. Community_Cards records can only be made for a Subround that has a null End_Time value.
4. Buy_Out records can only be made for a Game_Account that has a null Time_Closed value. This ensures that in-game records are not produced for past games that are no longer in play.

Deletion Constraints Based on Open Records

1. The Game, Round, Subround, and Game_Account records cannot be deleted unless the End_Time or Time_Closed value for the given record is non null. Furthermore, the deletion hierarchy, similar to the update hierarchy, must be followed in that a parent cannot be deleted unless its direct children have closed.
2. The Community_Cards, Players_Cards, and Make_Bet records cannot be deleted unless the Round record associated with that record, either by direct reference (Players_Cards) or indirect reference through Subround (Community_Cards and Make_Bet), has an End_Time that is non null.
3. Buy_In/Buy_Out records cannot be deleted unless the Time_Closed value is non null within the referenced Game_Account record.

   These deletion constraints prevent deletions on records that are still open to prevent important records from being lost during in-game play or from the active game being lost.

Insertion Constraints Based on Monetary Transaction Records

For any transaction entity recording monetary transfer i.e. Pay_Out, Make_Bet, Buy_In, Buy_Out, and Rake_Out, a record can be produced if and only if the monetary amount for the given transaction is less than the transactor entity's balance, and greater than or equal to zero.

3.3.4 Operations

The following examples express the types of operations used by our database to perform some of our tasks.

Player Buys Into a Game

This operation occurs when a Buy_In record is attempting to be inserted, but the Game_Account_ID is null, thus indicating that a new Game_Account needs to be created, before creating the Buy_In record insertion.

1. Perform a *retrieve* on all the Game records that have a null End_Time attribute, indicating an open game, and *retrieve* the Max_Seats value with each open Game record.
2. For each Game open game record, *retrieve* and count the Game_Account_ID's associated with the Game that have a null Time_Closed value.
3. Verify that this count is less than the Max_Seats value.
4. If none is available, *insert* a new Game record containing default Game record parameters.
5. Given a Game record that is open and not full, or the newly created Game record, perform an *insert* of a new Game_Account record containing the corresponding open Game_ID, the buy-in amount as the Chip_Stack, and the time of creation.
6. After the Game_Account is created, trigger the *insert* of the attempted Buy_In record with and *update* Game_Account_ID with the newly generated Game_Account record, and the time of creation.

Player Makes a Bet

This operation occurs when a Make_Bet record is attempting to be inserted, and is queued before insertion.

1. *Retrieve* the Game record associated with the Game_Account of the queued insertion, and the Game record of the Subround of the queued insertion.
2. Verify that the Game_ID's are equivalent and that the Subround and Game are both open (null End_Time).
3. Obtain the bet amount associated with the attempted insertion, if the bet amount is zero, *retrieve* the Bet_Type_ID associated with a "Fold" Bet_Type and continue forward with verifying the insertion, with the bet amount being a value of zero.
4. Otherwise, *Retrieve* the Game_Account balance associated with the bet, and regardless of the inserted bet amount, *retrieve* the limit amount of the Game record associated with the Game_Account.
5. If the limit amount is greater than or equal to the Game_Account balance, set the bet type to "All-In" and continue forward with verifying the insertion with a bet amount equal to the value of the Chip_Stack associated with it.
6. If the limit amount is less than the Game_Account balance, set the bet type to "Bet" and continue forward with verifying the insertion with a bet amount equal to the limit amount
7. From here, *Retrieve* all records with the Game_Account_ID associated with the current Round. If any record contains a "Fold" or "All-In" bet type, prevent the insertion.
8. *Retrieve* all the records of the associated Subround. If the Game_Account_ID is already associated with a Make_Bet record, prevent the insertion.

9. Once all conditions are verified, execute an insertion to add the new Make_Bet record.

Generating Players Cards

Triggered at the start of a Round, after a new Round record has been inserted.

1. *Retrieve* the Game_ID and Game_Type associated with the newly inserted Round record.
2. *Retrieve* all the Game_Account records associated with the same Game_ID that have a null Time_Closed value.
3. For each Game_Account_ID previously obtained, *insert* a new Playes_Cards record with the Round_ID, Game_Account_ID, and *retrieve* a random Card_Type_ID.
4. For each Game_Account_ID, *insert* a singular record.

Performing a Monetary Transaction

Occurs for all general transactions associated with the Make_Bet, Buy_In, Buy_Out, Pay_Out, and Rake_Out tables.

1. Verify that the amount associated with the potential monetary transaction record insert meets the monetary transaction operational constraint.
2. *Retrieve* the balance of the transactor and transactee.
3. *Update* the transactors balance by subtracting the balance by the amount of the potential transaction insert.
4. *Update* the transactee balance by adding the amount of the potential transaction insert.
5. After the updates are successful, allow the insert to the transaction record.

Starting a New Game

A new game record does not immediately start a new game as enough players have to be associated with the game in order for play to begin.

1. Upon an insert of two Game_Account records referencing the same game whereby the Game record has a null End_Time, *insert* a new Round record referencing the same Game_ID.
2. After the Round record is generated, *insert* Players_Cards records for each Game_Account associated with the Game_ID. After generating new card records for each of the players, *retrieve* the Subround_Type_ID associated with the subround name “Pre-Flop”, initialize the pot value to zero, and *insert* a new Subround record referencing the Round_ID.

3.4 Security

In order to maintain the security of our database several control procedures are imposed on the users to prevent malicious attacks or unintended misimputations. This is achieved through first assigning roles to different users of the database schema, Players and Admin, and assigning varying levels of privilege in manipulating the data.

The Admin are simply the individuals who maintain the schema, and thus are granted full privilege to the database to make any necessary changes, correction, etc. The Players are the primary users of the data itself, however, they were granted no privilege to make direct changes to the database. All interactions between the Player and the poker room occurs only through procedures and thus, the Players cannot make insertions, deletions, updates, or even selections on any part of the data without a procedural intermediary. The purpose of this was to not only clean and filter the actions that the user wanted to make before the database was affected but also to filter records based on who they were. Since each user carried hidden personal information i.e. credit card numbers, account balances, their in-game player card information, etc, it was decided that even Select access had to be restricted and filtered, allowing only the information corresponding to their Game_Account/Player records to be seen.

Below is an example of our security measure set up by first assigning the roles ‘Poker_Player’ and ‘Poker_Admin’.

```
CREATE ROLE 'Poker_Admin'@'localhost';
CREATE ROLE 'Poker_Player'@'localhost';

-- Admin can make changes to anything related to the schema
GRANT ALL PRIVILEGES ON poker_room.* TO 'Poker_Admin'@'localhost';

-- Only Procedural Access for Player
GRANT EXECUTE ON PROCEDURE poker_room.PlayerPokerRecords TO 'Poker_Player'@'localhost';
GRANT EXECUTE ON PROCEDURE poker_room.SafeBuyIn TO 'Poker_Player'@'localhost';
GRANT EXECUTE ON PROCEDURE poker_room.SafeBuyOut TO 'Poker_Player'@'localhost';
GRANT EXECUTE ON PROCEDURE poker_room.SafeMakeBet TO 'Poker_Player'@'localhost';
```

The procedures that the ‘Poker_Player’ user is enabled to interact with are, SafeMakeBet, SafeBuyIn, SafeBuyOut, for all the insertions and updates to the database allowing the user to create a new game account, place bets within their game, and buy out of a game. In order to view important information including betting history, card history, game account information, and player information the ‘Poker_Player’ interacts with the PlayerPokerRecords procedure.

To test the security constraints we granted ‘Poker_Player’ privileges to the user ‘BillyBob’ whose records are already present in the database and so he is aware of his Player_ID.

```
CREATE USER 'BillyBob'@'localhost' IDENTIFIED BY 'Bobismyname';
GRANT 'Poker_Player'@'localhost' TO 'BillyBob'@'localhost';
SET DEFAULT ROLE 'Poker_Player'@'localhost' TO 'BillyBob'@'localhost';


-- Activate automatic role activation on login
SET PERSIST activate_all_roles_on_login = ON;

-- Refresh privileges
FLUSH PRIVILEGES;
```

✓	904	08:11:38	CREATE ROLE 'Poker_Admin'@'localhost'	0 row(s) affected	0.016 sec
✓	905	08:11:38	CREATE ROLE 'Poker_Player'@'localhost'	0 row(s) affected	0.000 sec
✓	906	08:11:38	GRANT ALL PRIVILEGES ON poker_room.* TO 'Poker_Admin'@'localhost'	0 row(s) affected	0.016 sec
✓	907	08:11:38	GRANT EXECUTE ON PROCEDURE poker_room.PlayerPokerRecords TO 'Poker_Player'@'localhost'	0 row(s) affected	0.000 sec
✓	908	08:11:38	GRANT EXECUTE ON PROCEDURE poker_room.SafeBuyIn TO 'Poker_Player'@'localhost'	0 row(s) affected	0.000 sec
✓	909	08:11:38	GRANT EXECUTE ON PROCEDURE poker_room.SafeBuyOut TO 'Poker_Player'@'localhost'	0 row(s) affected	0.000 sec
✓	910	08:11:38	GRANT EXECUTE ON PROCEDURE poker_room.SafeMakeBet TO 'Poker_Player'@'localhost'	0 row(s) affected	0.000 sec
✓	911	08:11:38	CREATE USER 'BillyBob'@'localhost' IDENTIFIED BY 'Bobismyname'	0 row(s) affected	0.016 sec
✓	912	08:11:38	GRANT 'Poker_Player'@'localhost' TO 'BillyBob'@'localhost'	0 row(s) affected	0.000 sec
✓	913	08:11:38	SET DEFAULT ROLE 'Poker_Player'@'localhost' TO 'BillyBob'@'localhost'	0 row(s) affected	0.000 sec
✓	914	08:11:38	SET PERSIST activate_all_roles_on_login = ON	0 row(s) affected	0.016 sec
✓	915	08:11:38	FLUSH PRIVILEGES	0 row(s) affected	0.000 sec

After successfully assigning procedural access to ‘BillyBob’ he can sign in and connect to the server remotely and view his filtered records and start playing games. Here, he views his ‘Player Records’ and ‘Game_Account_Records’

```
C:\Users\koupa>mysql -u BillyBob -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CALL PlayerPokerRecords(1, null, 'Player Records');
ERROR 1046 (3D000): No database selected
mysql> USE poker_room;
Database changed
mysql> CALL PlayerPokerRecords(1, null, 'Player Records');
+-----------+-------------+------------+-----+----------------------------+--------------------+-----------------+
| Player_ID | Player_Name | Address    | Age | Account_Creation_Date      | Credit_Card_Number | Account_Balance |
+-----------+-------------+------------+-----+----------------------------+--------------------+-----------------+
|         1 | BillyBob    | 456 Oak St |  45 | 2024-11-19 07:46:38.710276 |   4532678956789123 |          79.600 |
+-----------+-------------+------------+-----+----------------------------+--------------------+-----------------+
1 row in set (0.05 sec)

Query OK, 0 rows affected (0.06 sec)

mysql> CALL PlayerPokerRecords(1, null, 'Game Account Records');
+-----------------+---------+------------+----------------------------+----------------------------+
| Game_Account_ID | Game_ID | Chip_Stack | Time_Created               | Time_Closed                |
+-----------------+---------+------------+----------------------------+----------------------------+
|               1 |    NULL |      0.000 | 2024-11-19 07:47:39.600190 | 2024-11-19 07:50:14.197623 |
|              13 |       2 |     20.000 | 2024-11-19 07:54:54.753739 | NULL                       |
+-----------------+---------+------------+----------------------------+----------------------------+
```

He also makes a buy in and places a bet into the game,

```
Query OK, 0 rows affected (0.04 sec)

mysql> CALL SafeBuyIn(1, 10);
Query OK, 1 row affected (0.06 sec)

mysql> CALL PlayerPokerRecords(1, null, 'Game Account Records');
+-----------------+---------+------------+----------------------------+----------------------------+
| Game_Account_ID | Game_ID | Chip_Stack | Time_Created               | Time_Closed                |
+-----------------+---------+------------+----------------------------+----------------------------+
|               1 |    NULL |      0.000 | 2024-11-19 07:47:39.600190 | 2024-11-19 07:50:14.197623 |
|              13 |       2 |     30.000 | 2024-11-19 07:54:54.753739 | NULL                       |
+-----------------+---------+------------+----------------------------+----------------------------+
2 rows in set (0.01 sec)

Query OK, 0 rows affected (0.03 sec)

mysql> CALL SafeMakeBet(1, 1);
ERROR 1644 (45000): No Reference Game for Game Account
mysql> CALL SafeMakeBet(13, 1);
Query OK, 1 row affected (0.10 sec)

mysql> CALL PlayerPokerRecords(1, 13, 'All Bet Records');
+-------------+--------------+----------+------------+---------------+
| Bet_Type_ID | Subround_ID | Round_ID | Bet_Amount | Subround_Name |
+-------------+--------------+----------+------------+---------------+
|           2 |            1 |        1 |      0.100 | Pre-Flop      |
|           2 |            2 |        1 |      0.100 | Flop          |
|           2 |            3 |        1 |      0.100 | Turn          |
|           2 |            4 |        1 |      0.100 | River         |
+-------------+--------------+----------+------------+---------------+
4 rows in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

However, when he attempts to make a direct Selection on anything else all his attempts fail.

```
mysql> CALL PlayerPokerRecords(1, 13, 'All Bet Records');
+-------------+-------------+----------+------------+---------------+
| Bet_Type_ID | Subround_ID | Round_ID | Bet_Amount | Subround_Name |
+-------------+-------------+----------+------------+---------------+
|           2 |           1 |        1 |      0.100 | Pre-Flop      |
|           2 |           2 |        1 |      0.100 | Flop          |
|           2 |           3 |        1 |      0.100 | Turn          |
|           2 |           4 |        1 |      0.100 | River         |
+-------------+-------------+----------+------------+---------------+
4 rows in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

mysql> SELECT * FROM Player;
ERROR 1142 (42000): SELECT command denied to user 'BillyBob'@'localhost' for table 'player'
mysql> SELECT * FROM Game_Account;
ERROR 1142 (42000): SELECT command denied to user 'BillyBob'@'localhost' for table 'game_account'
mysql> INSERT INTO Buy_In(Buy_In_Time,Game_Account_ID,Player_ID,Buy_In_Amount) VALUES (CURRENT_TIMESTAMP(6),13,1,1000);
ERROR 1142 (42000): INSERT command denied to user 'BillyBob'@'localhost' for table 'buy_in'
mysql>
```

Thus showing the successful implementation of the security protocol in filtering his output and restricting his access to the database.

3.5 Database Backup and Recovery

Backups to the poker room database were performed frequently through the use of mysql dump in order to dump the entire schema into a home backup directory. An example of this is shown below,

```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\koupa>mysqldump -u root -p poker_room > C:\Users\koupa\Desktop\Backup_SQL\poker_room_backup13.sql
Enter password: ****

C:\Users\koupa>
```

the schema then shows in the associated directory here,

3.6 Using Database Design or CASE Tool

In order to draw the ERD used for the poker database the LucidChart software was primarily used. This was chosen simply due to preference of design layout however, functionally it would have been better to use MySQL workbench instead. With this, the MySQL GUI was used for writing all the DDL's, DML's, triggers, stored procedures, and functions as well as performing testing. However, to test some of the user end testing, the command line was used.

3.7 Other Possible ER Relationships

The ER design process underwent several changes in order to better incorporate the functional relationships we wanted, and to simplify the query process.

Initially, our design involved the use of a central transactional entity that would store all the transactional records made between any of the entities in our database. This assumption was that by doing this, it would enable us to easily propagate the necessary changes to the balances of the Player, Game_Account, and System_Register. However, this was deemed completely unnecessary and it would make it much more difficult to ensure that our database design was properly normalized.

Other ER designs that were considered included the idea to use a separate Pot entity that would reference the Subround and interact with the Pay_Out, Make_Bets, Rake_Out, and System_Register. However, by doing so either we would have to create a new Pot record each time the Pot would change from a bet made, which would convolute the querying process, or we would have to use a one to one relationship with Subround and create a Pot record at the end of each Subround. However, this seemed to make the Pot entity completely redundant as this could simply be done by the Subround itself, removing an extra layer in the querying process.

4. Implementation Description

Our database was designed under the presumption that the population of the database would be a relatively self contained process. By this we mean that after the creation of the tables, and the insertions to populate the reference entities, and the players, all the records within the database could be generated through indirect interactions of the users through the four procedures they can manipulate.

This along with the fact that, the inherent design of the database to not only function as a means of storing and managing the data of the poker game, but to construct the game play itself, necessitated a complex interplay of functions, stored procedures, and triggers to properly query incoming data to evaluate the game state, and produce new records.

Below we describe the architecture of our implementation and the complete workflow of the poker room.

4.1 Data Dictionary

To construct our data dictionary we used the describe command for all of our entities as shown here, along with the successful output.

The resulting data dictionaries can be seen below:

All_Card_Types

Field	Type	Null	Key	Default	Extra
Card_Type_ID	int	NO	PRI	NULL	auto_increment
Card_Combination	int	NO	UNI	NULL	

All_Subround_Types

Field	Type	Null	Key	Default	Extra
Subround_Type_ID	int	NO	PRI	NULL	auto_increment
Subround_Name	varchar(20)	NO	UNI	NULL	

All_Seat_Types

Field	Type	Null	Key	Default	Extra
Max_Seats_ID	int	NO	PRI	NULL	auto_increment
Max_Seats	int	NO	UNI	NULL	

All_Bet_Types

Field	Type	Null	Key	Default	Extra
Bet_Type_ID	int	NO	PRI	NULL	auto_increment
Bet_Type	varchar(20)	NO	UNI	NULL	

All_Limit_Types

Field	Type	Null	Key	Default	Extra
Limit_Type_ID	int	NO	PRI	NULL	auto_increment
Limit_Type	decimal(7,3)	NO	UNI	NULL	

Player

Field	Type	Null	Key	Default	Extra
Player_ID	int	NO	PRI	NULL	auto_increment
Player_Name	varchar(50)	NO	UNI	NULL	
Address	varchar(50)	YES		NULL	
Age	int	NO		NULL	
Account_Creation_Date	timestamp(6)	NO		CURRENT_TIMESTAMP(6)	DEFAULT_GENERATED
Credit_Card_Number	bigint	YES		NULL	
Account_Balance	decimal(7,3)	NO		0.000	

System_Register

Field	Type	Null	Key	Default	Extra
Register_ID	int	NO	PRI	NULL	auto_increment
Account_Balance	decimal(7,3)	NO		NULL	

Game_Account

Field	Type	Null	Key	Default	Extra
Game_Account_ID	int	NO	PRI	NULL	auto_increment
Game_ID	int	YES	MUL	NULL	
Chip_Stack	decimal(7,3)	YES		NULL	
Time_Created	timestamp(6)	NO		NULL	
Time_Closed	timestamp(6)	YES		NULL	

Game

Field	Type	Null	Key	Default	Extra
Game_ID	int	NO	PRI	NULL	auto_increment
Limit_Type_ID	int	NO	MUL	NULL	
Max_Seats_ID	int	NO	MUL	NULL	
Start_Time	timestamp(6)	NO		NULL	
End_Time	timestamp(6)	YES		NULL	

Round

Field	Type	Null	Key	Default	Extra
Round_ID	int	NO	PRI	NULL	auto_increment
Game_ID	int	NO	MUL	NULL	
Start_Time	timestamp(6)	NO		NULL	
End_Time	timestamp(6)	YES		NULL	

Subround

Field	Type	Null	Key	Default	Extra
Subround_ID	int	NO	PRI	NULL	auto_increment
Round_ID	int	NO	MUL	NULL	
Subround_Type_ID	int	NO	MUL	NULL	
Start_Time	timestamp(6)	NO		NULL	
End_Time	timestamp(6)	YES		NULL	
Pot	decimal(7,3)	NO		NULL	

Buy_Out

Field	Type	Null	Key	Default	Extra
Buy_Out_Time	timestamp(6)	NO	PRI	NULL	
Game_Account_ID	int	NO	PRI	NULL	
Player_ID	int	NO	PRI	NULL	
Buy_Out_Amount	decimal(7,3)	NO		NULL	

Make_Bet

Field	Type	Null	Key	Default	Extra
Bet_Time	timestamp(6)	NO	PRI	NULL	
Subround_ID	int	NO	PRI	NULL	
Game_Account_ID	int	NO	PRI	NULL	
Bet_Type_ID	int	YES	MUL	NULL	
Bet_Amount	decimal(7,3)	YES		NULL	

Pay_Out

Field	Type	Null	Key	Default	Extra
Pay_Out_Time	timestamp(6)	NO	PRI	NULL	
Subround_ID	int	NO	PRI	NULL	
Game_Account_ID	int	NO	PRI	NULL	
Pay_Out_Amount	decimal(7,3)	NO		NULL	

Rake_Out

Field	Type	Null	Key	Default	Extra
Rake_Time	timestamp(6)	NO	PRI	NULL	
Register_ID	int	NO	PRI	NULL	
Subround_ID	int	NO	PRI	NULL	
Rake_Amount	decimal(7,3)	NO		NULL	

Community_Cards

Field	Type	Null	Key	Default	Extra
Community_Card_ID	int	NO	PRI	NULL	auto_increment
Card_Type_ID	int	NO	MUL	NULL	
Subround_ID	int	NO	MUL	NULL	

Buy_In

Field	Type	Null	Key	Default	Extra
Buy_In_Time	timestamp(6)	NO	PRI	NULL	
Game_Account_ID	int	NO	PRI	NULL	
Player_ID	int	NO	PRI	NULL	
Buy_In_Amount	decimal(7,3)	NO		NULL	

Players_Cards

Field	Type	Null	Key	Default	Extra
Player_Card_ID	int	NO	PRI	NULL	auto_increment
Round_ID	int	YES	MUL	NULL	
Game_Account_ID	int	NO	MUL	NULL	
Card_Type_ID	int	NO	MUL	NULL	

4.2 Advanced Features

The implementation of our database consists of fourteen stored procedures, fourteen triggers, and three functions. The procedure handles all the game logic workflow, producing the necessary records based on particular games states, handling user input, etc.

The triggers act as a way to monitor the integrity of the database ensuring that the operational rule constraints for deletion, updates, and inserts were properly performed, as well as triggering the call to the procedures to change the game state upon a given input.

The functions on the other hand were seldom necessitated as its limited scope reduced their requirement, though repeated subqueries within the procedures could have been constructed as functions to reduce redundancy with our procedures, but were not implemented. The functions we did use acted primarily as calculation boxes as well as conditional checkers.

Stored Procedures

- CloseRound
- CloseSubround
- DetermineWinner
- GeneratePlayerCards
- InsertCards
- PlayerLossGameClosure
- RoundStateHandler
- SafeBuyIn
- SafeBuyOut
- SafeMakeBet
- PokerPlayerRecords
- UpdateAccountBalance
- VerifyOpen
- VerifyTransaction

Stored Functions

- ComputeGameStats
- SumCommCards
- VerifyMakeBet

Triggers

- Make_Bet_Deletion_Trigger
- Community_Cards_Deletion_Trigger
- Players_Cards_Deletion_Trigger
- Buy_In_Deletion_Trigger
- Buy_Out_Deletion_Trigger
- Game_Deletion_Trigger
- Round_Deletion_Trigger
- Subround_Deletion_Trigger
- Close_Game_Trigger
- Start_Game_Trigger
- Close_Game_Account_Trigger
- Start_Round_Trigger
- Community_Cards_Trigger
- Make_Bet_Trigger

For the above advanced features it can be helpful to characterize them into three groups; control, primary, and secondary features. The control features consist of the features executable by the user consisting of,

- **SafeBuyIn**
- **SafeBuyOut**
- **SafeMakeBet**

The primary features are in turn the features that get called upon/triggered by the actions through the control features. Together these features create the records produced in the entire database, control the game logic, the transactional logic, etc. As such, we can think of the interplay of these features as cascades that propagate the changes to the database, with each control feature contributing to a separate cascade of changes.

The secondary features are the features that are outside of this scope that do not normally get enacted on by the user. These consist primarily of triggers of which all the deletion triggers, and a few of the insertion/update triggers are a part of. The purpose of these features is more so to ensure the integrity of the database, to be used by the admin to test the database, as well as potentially protect against malicious attacks.

To describe the features of our database we will consider in detail the cascades generated by each control procedure that in turn calls upon all the primary features of the database, as well as produce records for all the entity tables. In accessing these cascades we will assume the standard workflow of our database as described below.

The Standard Workflow

For our standard scenario we will imagine the workflow as follows:

- Some x number of individuals buy into a Game, either where the game record exists or is generated automatically.
- After the number of individuals is above a threshold, the game starts, the Round and Subround begin, and the players cards are handed out.
- The players make a series of bets, for the given subround, after which the subround ends, and a new one begins.
- Community cards are dealt out after the “flop” subround is reached, and betting continues.
- When all the players have bet, folded out, quit, or went all-in, or if the betting has continued to the river, the Round is ended, payouts to the winners are made, and a new round and subround begins.
- Once the players have decided to buy out of the game, the game ends.

Control/Primary Features

SafeBuyIn Cascade:

Players Buy-In, New Game is Created, Player Cards Generated

Mandatorily Activated Features

Procedures: SafeBuyIn, VerifyTransaction, UpdateAccountBalance
Functions: None
Triggers: None

Conditionally Activated Features

Procedures: GeneratePlayerCards, InsertCards
Functions: None
Triggers: Start_Game_Trigger, Start_Round_Trigger

Players Buy-In

SafeBuyIn

The SafeBuyIn procedure is a control procedure to enable the continuation or rejection of a buyin record by the user, updating the account balances, and/or generating a new game account record.

To do this, it first takes the player id and amount they are trying to buy in for, and then verify through the verify through the VerifyTransactor procedure that the player is in fact a existing id within the player table, that amount they are transacting for is not null, and that the amount they are buying in for is within the balance of their player account.

VerifyTransactor

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `VerifyTransactor`(
    IN Transactor_ID INT,
    IN Amount FLOAT8,
    IN transactor_name VARCHAR(50)
)
BEGIN
    DECLARE current_balance FLOAT8;

    INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered VerifyTransactor'));
    -- Check if transactor ID and amount are provided
    IF Transactor_ID IS NULL OR Amount IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'transactor_ID or transaction amount cannot be null';
    END IF;

    CASE transactor_name
        WHEN 'Player' THEN
            IF Amount <= 0 THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A negative/zero transaction cannot be made';
            END IF;
            SELECT Account_Balance INTO current_balance
            FROM Player
            WHERE Player_ID = Transactor_ID;

        WHEN 'Game_Account' THEN
            IF Amount < 0 THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A negative transaction cannot be made';
            END IF;
            SELECT Chip_Stack INTO current_balance
            FROM Game_Account
            WHERE Game_Account_ID = Transactor_ID;

        WHEN 'Subround' THEN
            SELECT Pot INTO current_balance
            FROM Subround
            WHERE Subround_ID = Transactor_ID;
        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid table identifier';
    END CASE;

    IF current_balance IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ID of Transactor does not exist';
    END IF;

    -- Don't need to check for Game_Account
    IF current_balance < Amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient balance for Transaction';
    END IF;

END
```

Here, we can see that the VerifyTransactor is a conditional procedure that is associated with all the strong entities involved in transactional exchange, and called upon with the argument of which transactor entity is involved. Given this information, for any transactor that violates any of the conditions, an error is thrown to rollback the transaction associated with it.

Once verified, the player id is checked against all the records in the BuyIn table, and if a associated game account id is found, whose end time within the game account table is null, the balance of the associated player and game account records are updated by the buy in, and a BuyIn record is generated.

UpdateAccountBalance

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateAccountBalance`(
    IN Transactor_ID INT,
    IN Transactee_ID INT,
    IN Amount DECIMAL(7, 3),
    IN transaction_entity_name VARCHAR(50)
)
BEGIN
    DECLARE p_Game_ID INT;
    DECLARE p_Chip_Stack DECIMAL(7, 3);
    DECLARE p_pot DECIMAL(7, 3);

    CASE transaction_entity_name
        WHEN 'Buy_In' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered UpdateAccountBalance for Buy_In'));
            UPDATE Player
            SET Account_Balance = Account_Balance - Amount
            WHERE Player_ID = Transactor_ID;

            UPDATE Game_Account
            SET Chip_Stack = Chip_Stack + Amount
            WHERE Game_Account_ID = Transactee_ID;

        WHEN 'Buy_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered UpdateAccountBalance for Buy_In'));
            SET p_Chip_Stack = (SELECT Chip_Stack FROM Game_Account WHERE Game_Account_ID = Transactor_ID
LIMIT 1);
            IF p_Chip_Stack - Amount = 0 THEN
                INSERT INTO Debug_Temp (message)
                VALUES (CONCAT('Player Cleared out, checking whether we should close game'));
                SET p_Game_ID = (SELECT Game_ID FROM Game_Account WHERE Game_Account_ID = Transactor_ID
LIMIT 1);
                CALL PlayerLossGameClosure(p_Game_ID, Transactor_ID);
            END IF;

            UPDATE Game_Account
            SET Chip_Stack = Chip_Stack - Amount
            WHERE Game_Account_ID = Transactor_ID;

            UPDATE Player
            SET Account_Balance = Account_Balance + Amount
            WHERE Player_ID = Transactee_ID;

        WHEN 'Pay_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered UpdateAccountBalance for Pay_Out'));
            SET p_pot = (Select Pot from Subround Where Subround_ID = Transactor_ID);
            INSERT INTO Debug_Money (message)
            VALUES (CONCAT('Update Pot Pay_Out: ', p_pot - Amount));
            UPDATE Subround
            SET Pot = Pot - Amount
            WHERE Subround_ID = Transactor_ID;

            SET p_Chip_Stack = (SELECT Chip_Stack FROM Game_Account WHERE Game_Account_ID = Transactee_ID
LIMIT 1);
            IF p_Chip_Stack + Amount = 0 THEN
                SET p_Game_ID = (SELECT Game_ID FROM Game_Account WHERE Game_Account_ID = Transactee_ID
LIMIT 1);
                CALL PlayerLossGameClosure(p_Game_ID, Transactee_ID);
            END IF;

            UPDATE Game_Account
            SET Chip_Stack = Chip_Stack + Amount
            WHERE Game_Account_ID = Transactee_ID;

        WHEN 'Rake_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered UpdateAccountBalance for Rake_Out'));
            SET p_pot = (Select Pot from Subround Where Subround_ID = Transactor_ID);
            INSERT INTO Debug_Money (message)
            VALUES (CONCAT('Update Pot Rake_Out: ', p_pot - Amount));
            UPDATE Subround
            SET Pot = Pot - Amount
            WHERE Subround_ID = Transactor_ID;

            UPDATE System_Register
            SET Account_Balance = Account_Balance + Amount
            WHERE Register_ID = Transactee_ID;

        WHEN 'Make_Bet' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered UpdateAccountBalance for Make_Bet'));
            UPDATE Game_Account
            SET Chip_Stack = Chip_Stack - Amount
            WHERE Game_Account_ID = Transactor_ID;
            SET p_pot = (Select Pot from Subround Where Subround_ID = Transactee_ID);

            UPDATE Subround
            SET Pot = Pot + Amount
            WHERE Subround_ID = Transactee_ID;

        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Transaction Entity identifier';

    END CASE;

END
```

Similar to the VerifyTransactor procedure the UpdateAccountBalance procedure acts as a conditional procedure for all the specific transaction types and makes the corresponding updates to each entity referenced in the transaction. With this, the procedure calls upon the PlayerLossGameClosure procedure in the event of a buyout transaction, which is involved with determining if a game needs to be closed due to a complete loss of all players referenced to the game.

In the case that either the game account is not null or a existing record does not exist, all the records within the game table is checked to see whether a open game exists which is not full, i.e. the number of game accounts associated with that game record is not equivalent to the max seats limit of the game. If a game is found, the game id is used along with the buy-in amount to generate a new game account, create a new BuyIn record, and update the account balances of the players. Otherwise, a new game record is generated and the same process as before is repeated to create a new game account.

The SafeBuyIn procedure code can be visualized below,

SafeBuyIn

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `SafeBuyIn`(
    IN p_Player_ID INT,
    IN p_Buy_In_Amount DECIMAL(7, 3)
    )
BEGIN
    DECLARE open_Game_ID INT;
    DECLARE max_seats INT;
    DECLARE current_seat_count INT;
    DECLARE rand_max_seats_ID INT;
    DECLARE rand_limit_type_ID INT;
    DECLARE p_Game_Account_ID INT DEFAULT NULL;

    DECLARE game_cursor CURSOR FOR
        SELECT Game.Game_ID, All_Seat_Types.Max_Seats
        FROM Game
        JOIN All_Seat_Types ON Game.Max_Seats_ID = All_Seat_Types.Max_Seats_ID
        WHERE Game.End_Time IS NULL;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET open_Game_ID = NULL;

    INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered SafeBuyIn'));
    CALL VerifyTransactor(p_Player_ID, p_Buy_In_Amount, 'Player');
    SET p_Game_Account_ID = (SELECT BI.Game_Account_ID FROM Buy_IN BI WHERE BI.Player_ID = p_Player_ID
AND
    EXISTS(SELECT * FROM Game_Account GA WHERE GA.Time_Closed IS NULL AND GA.Game_Account_ID =
BI.Game_Account_ID) LIMIT 1);


    IF p_Game_Account_ID IS NOT NULL THEN
        -- Check if the Game_Account_ID exists in Game_Account table and is open (Time_Closed is NULL)
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Game account exists'));
        IF (SELECT COUNT(*) FROM Game_Account
            WHERE Game_Account_ID = p_Game_Account_ID
            AND Time_Closed IS NULL) > 0 THEN

            -- Insert the Buy_In record
            CALL UpdateAccountBalance(p_Player_ID, p_Game_Account_ID, p_Buy_In_Amount, 'Buy_In');
            INSERT INTO Buy_In (Buy_In_Time, Game_Account_ID, Player_ID, Buy_In_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Game_Account_ID, p_Player_ID, p_Buy_In_Amount);


        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid or Closed Game_Account_ID';
        END IF;

    ELSE
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Game account DOES NOT exist'));
        -- Open cursor for finding an open game
        OPEN game_cursor;

        -- Loop through open games
        read_game: LOOP
            FETCH game_cursor INTO open_Game_ID, max_seats;

            -- If there is an open game, check current seat count
            IF open_Game_ID IS NOT NULL THEN
                SELECT COUNT(*)
                INTO current_seat_count
                FROM Game_Account
                WHERE Game_ID = open_Game_ID AND Time_Closed IS NULL;

                -- Check if the game is not full
                IF current_seat_count < max_seats THEN
                    LEAVE read_game;
                END IF;
            ELSE
                LEAVE read_game;
            END IF;
        END LOOP read_game;
        CLOSE game_cursor;

        -- If no open game was found, create a new one
        IF open_Game_ID IS NULL THEN
            SELECT Limit_Type_ID INTO rand_limit_type_ID
            FROM All_Limit_Types ORDER BY RAND() LIMIT 1;

            SELECT Max_Seats_ID INTO rand_max_seats_ID
            FROM All_Seat_Types ORDER BY RAND() LIMIT 1;

            INSERT INTO Game (Limit_Type_ID, Max_Seats_ID, Start_Time)
            VALUES (rand_limit_type_ID, rand_max_seats_ID, CURRENT_TIMESTAMP(6));
            SET open_Game_ID = LAST_INSERT_ID();
        END IF;

        -- Insert a new Game_Account for the player
        INSERT INTO Game_Account (Game_ID, Chip_Stack, Time_Created)
        VALUES (open_Game_ID, 0, CURRENT_TIMESTAMP(6));
        SET p_Game_Account_ID = LAST_INSERT_ID();

        -- Insert the Buy_In record
        CALL UpdateAccountBalance(p_Player_ID, p_Game_Account_ID, p_Buy_In_Amount, 'Buy_In');
        INSERT INTO Buy_In (Buy_In_Time, Game_Account_ID, Player_ID, Buy_In_Amount)
        VALUES (CURRENT_TIMESTAMP(6), p_Game_Account_ID, p_Player_ID, p_Buy_In_Amount);


    END IF;
END
```

New Game is Created

Start_Game_Trigger

If a new game account is generated, upon the insertion of the game account record, the Start_Game_Trigger is set off. This trigger essentially determines whether the game associated with the new game account record has greater than two game accounts associated with it, and that it does not have any children rounds already present.

```
DELIMITER //
CREATE TRIGGER Start_Game_Trigger
AFTER INSERT ON Game_Account
FOR EACH ROW
    BEGIN
        DECLARE Game_End_Time INT;
        DECLARE Game_Round_Counts INT DEFAULT 0;
        DECLARE Open_Round_ID INT;
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered the Start_Game_Trigger'));

        -- Count the associated Games Round records to verify it is 0
        SELECT COUNT(*) INTO Game_Round_Counts FROM Round WHERE Game_ID = NEW.Game_ID LIMIT 1;

        IF Game_Round_Counts = 0 THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('New Round being created'));
            SET Game_End_Time = (SELECT End_Time FROM Game WHERE Game_ID = NEW.Game_ID);
            IF (SELECT COUNT(*) FROM Game_Account
                WHERE (Game_ID = NEW.Game_ID) AND (NEW.Time_Closed IS NULL) AND (Game_End_Time IS NULL))
>= 2
                THEN
                    INSERT INTO Round (Game_ID, Start_Time, End_Time)
                    VALUES (NEW.Game_ID, CURRENT_TIMESTAMP(6), NULL);
            END IF;
        -- Obtain a round record that is open, and give the new game_account their cards
        ELSE
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Round already exists, just give them new cards'));
            SELECT Round_ID INTO Open_Round_ID FROM Round WHERE Game_ID = NEW.Game_ID AND End_Time IS
NULL LIMIT 1;
            CALL GeneratePlayerCards(Open_Round_ID);
        END IF;

    END//
```

Seeing the code for the trigger above, we can see that in the event that the game does not have any rounds associated with it, a new round record is generated, if and only if, the number of game accounts associated with the game is greater than two. In the event that the game already has an active round record, the GeneratePlayerCards procedure is called to generate a new card for the account.

Start_Round_Trigger

```
DELIMITER //
CREATE TRIGGER Start_Round_Trigger
AFTER INSERT ON Round
FOR EACH ROW
    BEGIN
        DECLARE Curr_Subround_Type_ID INT;
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered the Start_Round_Trigger; Giving cards to players'));
        CALL GeneratePlayerCards(NEW.Round_ID);

        SELECT Subround_Type_ID INTO Curr_Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name =
'Pre-Flop';

        INSERT INTO Subround (Round_ID, Subround_Type_ID, Start_Time, End_Time, Pot)
            VALUES (NEW.Round_ID, Curr_Subround_Type_ID, CURRENT_TIMESTAMP(6), NULL, 0);
    END//
```

In the event that a new round record is being generated for the associated game, the Start_Round_Trigger calls the GeneratePlayerCards to produce new card records for each of the game accounts within the Players_Cards table. Only after producing these card records, does a new subround record get generated, initiating the start of the game.

Player Cards Generated

GeneratePlayerCards

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GeneratePlayerCards`(
    IN p_Round_ID INT
)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE Curr_Game_Account_ID INT;
    DECLARE Curr_Game_ID INT;
    DECLARE Curr_Card_Type_ID INT;

    DECLARE GA_G_IDs CURSOR FOR
        SELECT GA.Game_Account_ID, R.Game_ID
        FROM Game_Account GA
        JOIN Round R ON GA.Game_ID = R.Game_ID
        WHERE R.Round_ID = p_Round_ID AND GA.Time_Closed IS NULL
        AND NOT EXISTS (SELECT * FROM Players_Cards PC WHERE PC.Game_Account_ID = GA.Game_Account_ID AND
PC.Round_ID = p_Round_ID);

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('Entered GeneratePlayerCards'));
    OPEN GA_G_IDs;

    GA_G_IDs_Loop: LOOP

        FETCH GA_G_IDs INTO Curr_Game_Account_ID, Curr_Game_ID;
            IF done THEN
                LEAVE GA_G_IDs_Loop;
            END IF;
            CALL InsertCards(1, 'PC', p_Round_ID, Curr_Game_Account_ID);
    END LOOP;

    CLOSE GA_G_IDs;

END
```

Regardless of whether a new round was generated, or whether new players are added into a game, the GeneratePlayerCards procedure is called upon to produce records for these players. This occurs by iterating through all the game accounts associated with the game of reference, and calling upon a InsertCards to make the card insertion to the Players_Cards table.

InsertCards

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertCards`(
    IN p_NumCards INT,
    IN p_identifier VARCHAR(2),
    IN p_ID_a INT,
    IN p_ID_b INT
)
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE Random_Card_ID INT;

    INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('Entered the InsertCards my identity is: ', p_identifier));
    WHILE i < p_NumCards DO

        SELECT Card_Type_ID INTO Random_Card_ID
        FROM All_Card_Types ORDER BY RAND() LIMIT 1;

        IF p_identifier = 'CC' THEN
            -- Insert the randomly selected card into Community_Cards
            INSERT INTO Community_Cards (Subround_ID, Card_Type_ID)
            VALUES (p_ID_a, Random_Card_ID);
        ELSE
            INSERT INTO Players_Cards (Round_ID, Game_Account_ID, Card_Type_ID)
            VALUES (p_ID_a, p_ID_b, Random_Card_ID);
        END IF;

        -- Increment the loop counter
        SET i = i + 1;
    END WHILE;
END
```

The InsertCards procedure works to insert both player cards and community cards based on the conditional passed to the procedure as an argument. Given this, a random card type within the All_Card_Types reference table, is obtained for the number of cards associated with the conditional. In the case of a players card, only one card is generated, however, for community cards, the number of cards is dependent on the subround type (zero for ‘pre-flop’, three for ‘flop’, and one for ‘turn’/’river’). The sampling of these cards are made with duplication thus two different players/community cards can have the same card types.

Example Scenario

To show the SafeBuyIn Cascade we will start by creating a new game record and allowing a single player to buy into the game as seen below.

```
USE poker_room;
-- Initial Buy In --

-- Create Game
INSERT INTO Game(Limit_Type_ID, Max_Seats_ID, Start_Time) VALUES (1, 1, CURRENT_TIMESTAMP(6));
SELECT * FROM Game;

-- One player buys in
CALL SafeBuyIn(1, 1);

-- Show the Buy Ins were successful
-- Player and Game_Account should be updated

SELECT * FROM Buy_In;
SELECT * FROM Player;
SELECT * FROM Game_Account;

-- Show that no game was created because the Game_Accounts Associated is <2
-- Show no Players Cards were generated
SELECT * FROM Game;
SELECT * FROM Round;
SELECT * FROM Subround;
SELECT * FROM Players_Cards;
```

Buy_In Record

Buy_In_Time	Game_Account_ID	Player_ID	Buy_In_Amount
2024-11-20 15:03:26.370738	1	1	1.000
NULL	NULL	NULL	NULL

Game_Account Record

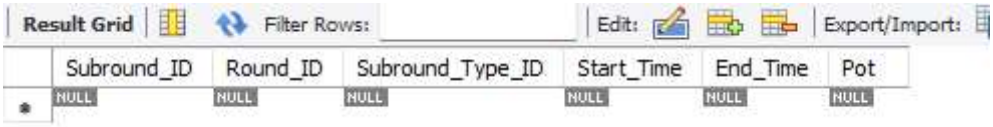
Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	1.000	2024-11-20 15:03:26.346243	NULL
NULL	NULL	NULL	NULL	NULL

Game Record

Game_ID	Limit_Type_ID	Max_Seats_ID	Start_Time	End_Time
1	1	1	2024-11-20 15:03:26.259687	NULL
NULL	NULL	NULL	NULL	NULL

Round Record

Subround



Players_Cards

Upon the initial buy-in of a single player, the BuyIn record is generated and the game account and player account is updated with the corresponding value bought in for. We see however, that the round, subround, and player cards records are still empty, as the threshold of players hasn't been reached.

From here, we can then create a new game account with a new buy in from another player,

```
-- Cascade the Start Game: Round, Subround, Player Cards --
-- Third Player Buys into the game, Show that the Game starts
-- First Round and Subround should be Generated
-- Player Cards Should be Generated
-- This is Set by the trigger Start_Game_Trigger and Start_Round_Trigger
CALL SafeBuyIn(2, 1);
SELECT * FROM Game_Account;
SELECT * FROM Game;
SELECT * FROM Round;
SELECT * FROM Subround;
SELECT * FROM Players_Cards;

-- Buy-In for the Rest of the Players
CALL SafeBuyIn(3, 1);
CALL SafeBuyIn(4, 1);
CALL SafeBuyIn(5, 1);
CALL SafeBuyIn(6, 1);
```

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	1.000	2024-11-18 19:10:20.503429	NULL
2	1	1.000	2024-11-18 19:15:22.705563	NULL
NULL	NULL	NULL	NULL	NULL

Game_ID	Limit_Type_ID	Max_Seats_ID	Start_Time	End_Time
1	1	1	2024-11-18 19:10:20.471864	NULL
NULL	NULL	NULL	NULL	NULL

Round_ID	Game_ID	Start_Time	End_Time
1	1	2024-11-20 15:09:16.331061	NULL
NULL	NULL	NULL	NULL

Subround_ID	Round_ID	Subround_Type_ID	Start_Time	End_Time	Pot
1	1	1	2024-11-20 15:09:16.331061	NULL	0.000
NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit:

Player_Card_ID	Round_ID	Game_Account_ID	Card_Type_ID
1	1	1	3
2	1	2	1
NULL	NULL	NULL	NULL

Immediately after inserting a new player a cascade is initiated by the trigger Start_Game_Trigger, generating a new round record, which in turn triggers Start_Round_Trigger, calling GeneratePlayerCards to create new player card records, and creating a new subround record.

Completing the rest of the buyins we get the filled game.

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	1.000	2024-11-20 15:03:26.346243	NULL
2	1	2.000	2024-11-20 15:09:16.331061	NULL
3	1	1.000	2024-11-20 15:18:14.611226	NULL
4	1	1.000	2024-11-20 15:18:14.637906	NULL
5	1	1.000	2024-11-20 15:18:14.659689	NULL
6	1	1.000	2024-11-20 15:18:14.678511	NULL
NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message	Duration / Fetch
1	15:03:26	USE poker_room	0 row(s) affected	0.000 sec
2	15:03:26	INSERT INTO Game(Limit_Type_ID, Max_Seats_ID, Start_Time) VALUES (1, 1, CURREN...	1 row(s) affected	0.016 sec
3	15:03:26	SELECT * FROM Game LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
4	15:03:26	CALL SafeBuyIn(1, 1)	1 row(s) affected	0.047 sec
5	15:03:26	SELECT * FROM Buy_In LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
6	15:03:26	SELECT * FROM Player LIMIT 0, 1000	16 row(s) returned	0.000 sec / 0.000 sec
7	15:03:26	SELECT * FROM Game_Account LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
8	15:07:23	SELECT * FROM Round LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
9	15:07:23	SELECT * FROM Subround LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
10	15:07:23	SELECT * FROM Players_Cards LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	15:09:16	CALL SafeBuyIn(2, 1)	1 row(s) affected	0.031 sec
12	15:09:16	SELECT * FROM Game_Account LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
13	15:09:16	SELECT * FROM Game LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
14	15:09:16	SELECT * FROM Round LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
15	15:09:16	SELECT * FROM Subround LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
16	15:09:16	SELECT * FROM Players_Cards LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
17	15:18:14	CALL SafeBuyIn(2, 1)	1 row(s) affected	0.031 sec
18	15:18:14	CALL SafeBuyIn(3, 1)	1 row(s) affected	0.031 sec
19	15:18:14	CALL SafeBuyIn(4, 1)	1 row(s) affected	0.031 sec
20	15:18:14	CALL SafeBuyIn(5, 1)	1 row(s) affected	0.016 sec
21	15:18:14	CALL SafeBuyIn(6, 1)	1 row(s) affected	0.016 sec

SafeMakeBet Cascade:

Player Bets, Subround Propagates, Winner is Determined, Round Ends

Mandatorily Activated Features (Unique)

Procedures: SafeMakeBet, VerifyOpen, RoundStateHandler
Functions: VerifyMakeBet, ComputeGameStats
Triggers: Make_Bet_Trigger,

Conditionally Activated Features (Unique)

Procedures: DetermineWinner, CloseRound, CloseSubround
Functions: SumCommCards
Triggers: Start_Subround_Trigger, Close_Game_Account_Trigger, Community_Cards_Trigger

Player Bets

SafeMakeBet

The SafeMakeBet procedure is a control procedure to enable the continuation or rejection of a makebet record, verifying/cleaning the bet, determining the bet type associated with the bet placed, and updating the account balances of the player once verified. The bet is first verified by calling upon VerifyMakeBet and VerifyOpen.

VerifyMakeBet

```
CREATE DEFINER=`root`@`localhost` FUNCTION `VerifyMakeBet`(
    p_Transactor_ID INT,
    p_Amount DECIMAL(7, 3),
    p_Game_ID INT,
    p_Round_ID INT,
    p_Subround_ID INT,
    check_condition VARCHAR(20)
) RETURNS int
    DETERMINISTIC
BEGIN

    DECLARE Game_Round_Counts INT DEFAULT 0;

    IF check_condition = 'Verify_Better' THEN

        IF p_Transactor_ID IS NULL OR p_Amount IS NULL OR
            NOT EXISTS (SELECT * FROM Game_Account GA WHERE GA.Game_Account_ID = p_Transactor_ID) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Transactor_ID or Transaction Amount Cannot be
Null';
        END IF;

    END IF;

    IF check_condition = 'Verify_Game' THEN

        IF p_Game_ID IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No Reference Game for Game Account';
        END IF;

        -- Check if the game even has started
        SELECT COUNT(*) INTO Game_Round_Counts FROM Round WHERE Game_ID = p_Game_ID LIMIT 1;
            IF Game_Round_Counts = 0 THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The Game has not started yet';
        END IF;

        IF p_Round_ID IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'All Rounds Are Currently Closed';
        END IF;

        IF p_Subround_ID IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'All Subrounds Are Currently Closed';
        END IF;

    END IF;

RETURN 1;
END
```

The VerifyMakeBet is a conditional procedure that verifies the better and the game associated with the better. First, the bettor is verified that they are indeed not null and exists within the game account records. Afterwards, SafeMakeBet has the assurance to pull the game, round, and subround ids to call upon VerifyMakeBet again, whereby the game id is checked to see if it is null, whether the game has indeed started i.e. has rounds and subrounds as children, and to see whether they are active and haven't closed.

After verifying these conditions VerifyOpen is called upon to send the user an error in the event that the subround/game account has a non null end time.

VerifyOpen

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `VerifyOpen`(
    IN Transactor_ID INT,
    IN Transactee_ID INT,
    IN transaction_entity_name VARCHAR(50)
)
BEGIN
    DECLARE Transactor_Open TIMESTAMP(6);
    DECLARE Transactee_Open TIMESTAMP(6);

    CASE transaction_entity_name
        WHEN 'Buy_In' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered VerifyOpen for Buy_In'));
            SELECT Time_Closed INTO Transactee_Open FROM Game_Account
                WHERE Game_Account_ID = Transactee_ID;
            IF Transactee_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Buy_In cannot be made on closed
Game_Accounts';
            END IF;

        WHEN 'Buy_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered VerifyOpen for Buy_out'));
            SELECT Time_Closed INTO Transactor_Open FROM Game_Account
                WHERE Game_Account_ID = Transactor_ID;
            IF Transactor_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Buy_Out cannot be made on closed
Game_Accounts';
            END IF;
            IF NOT EXISTS (
                SELECT 1 FROM Player
                WHERE Player_ID = Transactee_ID
                ) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Player does not exist';
            END IF;

        WHEN 'Pay_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered VerifyOpen for Pay_Out'));
            SELECT End_Time INTO Transactor_Open FROM Subround
                WHERE Subround_ID = Transactor_ID;
            IF Transactor_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Pay_Outs cannot be made on closed Subrounds';
            END IF;

        WHEN 'Rake_Out' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered VerifyOpen for Rake_Out'));
            SELECT End_Time INTO Transactor_Open FROM Subround
                WHERE Subround_ID = Transactor_ID;
            IF Transactor_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Rake_Out cannot be made on closed Subrounds';
            END IF;

        WHEN 'Make_Bet' THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered VerifyOpen for Make_Bet'));
            SELECT Time_Closed INTO Transactor_Open FROM Game_Account
                WHERE Game_Account_ID = Transactor_ID;
            IF Transactor_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Make_Bet cannot be made on closed
Game_Accounts';
            END IF;

            SELECT End_Time INTO Transactee_Open FROM Subround
                WHERE Subround_ID = Transactee_ID;
            IF Transactee_Open IS NOT NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Make_Bet cannot be made on closed Subrounds';
            END IF;

        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Transaction Entity identifier';
    END CASE;
END
```

The VerifyOpen procedure is used by all transactions to verify that the transactor and/or transactee has a non null end time. This is important to ensure that no potential transactions occur between closed records, (with the exception of Buy_Out).

The use of VerifyOpen and VerifyMakeBet is a bit of a redundancy as their functionalities directly cross over, however, in the case of betting transactions VerifyOpen was used simply as an error handler.

These checks validate the transactional validity of the bet made, but from here, the bet is then validated against the current round to see whether the bet placed violates any previous bet. In regards to our poker game, a player is only allowed to make one bet per subround and cannot place a bet within the same round after they have folded or went all in. Thus, the bet is checked across all previous records of the round to verify that this isn't violated.

After complete verification of the bet, a bet type is assigned based on the bet amount's value, and the amount the player bet is modified based on if it was greater than or less than the limit of the game. From here, if the bet type is determined to be of the type "bet" a rake is collected on the bet made from the pot, and the game account/pot balance of the subround is updated through the UpdateAccountBalance procedure. On the other hand, if the bet type is of the type "all-in" the game account is cleared of its balance with a small amount left behind to ensure that the game account is not prematurely deemed inactive. After this, all the updates are made and the corresponding bet record is inserted into the Make_Bet table.

The SafeMakeBet procedure can be seen below,

SafeMakeBet

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `SafeMakeBet`(
    IN p_Game_Account_ID INT,
    IN p_Bet_Amount DECIMAL(7, 3)
)
BEGIN
    DECLARE GA_Game_ID INT;
    DECLARE SR_Round_ID INT;
    DECLARE c_System_Register_ID INT;
    DECLARE Fold_All_In_Count INT DEFAULT 0;
    DECLARE Rake_Tax DECIMAL(7, 3);

    DECLARE Fold_Bet_Type_ID INT;
    DECLARE Bet_Bet_Type_ID INT;
    DECLARE All_In_Bet_Type_ID INT;

    DECLARE G_Limit DECIMAL(7, 3);
    DECLARE p_Chip_Stack DECIMAL(7, 3);
    DECLARE p_Subround_ID INT;
    DECLARE verifier INT;
    DECLARE p_Pot DECIMAL(7, 3);

    -- Verify that GA is not Null and that the Amount is within balance

    SET verifier = VerifyMakeBet(p_Game_Account_ID, p_Bet_Amount, NULL, NULL, NULL, 'Verify_Better');

    SELECT Game_ID INTO GA_Game_ID FROM Game_Account WHERE Game_Account_ID = p_Game_Account_ID;
    SELECT Round_ID INTO SR_Round_ID FROM Round WHERE Game_ID = GA_Game_ID AND End_Time IS NULL LIMIT 1;
    SELECT Subround_ID INTO p_Subround_ID FROM Subround WHERE Round_ID = SR_Round_ID AND End_Time IS NULL
LIMIT 1;

    SET verifier = VerifyMakeBet(NULL, NULL, GA_Game_ID, SR_Round_ID, p_Subround_ID, 'Verify_Game');

    CALL VerifyOpen(p_Game_Account_ID, p_Subround_ID, 'Make_Bet');

    -- Select IDs associated with Bet_Types
    SELECT Bet_Type_ID INTO Fold_Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'Fold';
    SELECT Bet_Type_ID INTO All_In_Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'All-In';
    SELECT Bet_Type_ID INTO Bet_Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'Bet';

    -- Find whether a previous record assoc with Game_Account_Id in the Round is Fold/All-in
    SELECT COUNT(*) > 0 INTO Fold_All_In_Count FROM Make_Bet
    JOIN Subround ON Make_Bet.Subround_ID = Subround.Subround_ID
    WHERE Subround.Round_ID = SR_Round_ID
    AND Make_Bet.Game_Account_ID = p_Game_Account_ID
    AND (Make_Bet.Bet_Type_ID = Fold_Bet_Type_ID OR Make_Bet.Bet_Type_ID = All_In_Bet_Type_ID)
    LIMIT 1;

    IF Fold_All_In_Count > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Game_Account is Associated with a Previous Fold/All-
in in the Same Round';
    END IF;

    IF EXISTS (SELECT 1 FROM Make_Bet WHERE Subround_ID = p_Subround_ID AND Game_Account_ID =
p_Game_Account_ID) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Game_Account is not allowed to make multiple bets per
Subround';
    END IF;

    SELECT Limit_Type INTO G_Limit FROM All_Limit_Types
    WHERE Limit_Type_ID = (
        SELECT Limit_Type_ID FROM Game
        WHERE Game_ID = GA_Game_ID);

    Set p_Chip_Stack = (SELECT Chip_Stack FROM Game_Account WHERE Game_Account_ID = p_Game_Account_ID);

    IF p_Bet_Amount = 0 THEN
        INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered SafeMakeBet: Fold'));
        INSERT INTO Make_Bet (Bet_Time, Subround_ID, Game_Account_ID, Bet_Type_ID, Bet_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Subround_ID, p_Game_Account_ID, Fold_Bet_Type_ID, 0);
    ELSEIF G_Limit >= p_Chip_Stack THEN
        INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered SafeMakeBet: All-In for ', p_Chip_Stack));
        -- For All-Ins to prevent closure of the Game_Account, a small fraction of money (.001) is left
in the account
        -- If the player loses, a negative payout transaction is made to the Game_account, closing their
account
        SET p_Chip_Stack = p_Chip_Stack - 0.001;
        CALL UpdateAccountBalance(p_Game_Account_ID, p_Subround_ID, p_Chip_Stack, 'Make_Bet');
        INSERT INTO Make_Bet (Bet_Time, Subround_ID, Game_Account_ID, Bet_Type_ID, Bet_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Subround_ID, p_Game_Account_ID, All_In_Bet_Type_ID,
p_Chip_Stack);


    ELSE

        CALL UpdateAccountBalance(p_Game_Account_ID, p_Subround_ID, G_Limit, 'Make_Bet');
        INSERT INTO Make_Bet (Bet_Time, Subround_ID, Game_Account_ID, Bet_Type_ID, Bet_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Subround_ID, p_Game_Account_ID, Bet_Bet_Type_ID, G_Limit);



        SELECT Pot INTO p_Pot FROM Subround S WHERE S.Subround_ID = p_Subround_ID;
        SET Rake_Tax = G_Limit * 0.025;

        -- Check that Pot hasn't cleared out from a Pay_Out already and its larger than the rake_tax
        IF p_Pot >= Rake_Tax THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered SafeMakeBet: Bet - RakeOut being performed'));
            -- Obtain a Rake_Tax on every bet for 2.5%
            SET c_System_Register_ID = (SELECT Register_ID FROM System_Register ORDER BY Register_ID ASC
LIMIT 1);
            CALL UpdateAccountBalance(p_Subround_ID, c_System_Register_ID, Rake_Tax, 'Rake_Out');
            INSERT INTO Rake_Out (Rake_Time, Register_ID, Subround_ID, Rake_Amount)
                VALUES (CURRENT_TIMESTAMP(6), c_System_Register_ID, p_Subround_ID, Rake_Tax);
        END IF;


    END IF;

END
```

Make_Bet_Trigger

Immediately after the insertion of the betting record, the Make_Bet_Trigger is enacted upon. The purpose of the Make_Bet_Trigger is to initiate a check on the current round state based on all the bets made previously in the current round, and whether the subround/round should close and if the winner needs to be determined. This is done by first retrieving the counts for the number of bets place within the subround, the number of game accounts linked with the round, the counts of the folds/all-ins of the entire round, and the counts of the folds/all-ins of all the subrounds with the exception of the current subround. These counts are then given and handled by the RoundStateHandler to evaluate the round state.

These counts are determined through the ComputeGameStats function which can be seen below,

ComputeGameStats

```
CREATE DEFINER=`root`@`localhost` FUNCTION `ComputeGameStats`(
    p_Subround_ID INT,
    p_Game_Account_ID INT,
    stats_case VARCHAR(20)

) RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE result INT DEFAULT 0;
    DECLARE inactive_accounts INT DEFAULT 0;
    DECLARE p_Game_ID INT DEFAULT 0;

    SET p_Game_ID = (SELECT Game_ID FROM Game_Account WHERE Game_Account_ID = p_Game_Account_ID);
    SELECT COUNT(*) INTO inactive_accounts FROM Game_Account GA WHERE
        GA.Game_ID IS NOT NULL AND GA.Game_ID = p_Game_ID AND GA.Time_Closed IS NOT NULL;

    -- All the Folds/Game_Account_Closures of the Entire Round AFTER the player has actually made a bet
in the Game
    IF stats_case = 'Fold_Count' THEN
        -- First get the players who are actively in the round and who have folded
        SELECT
            SUM(
                CASE
                    WHEN MB.Bet_Type_ID = (SELECT Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'Fold')
                    AND NOT EXISTS (
                            SELECT 1 FROM Game_Account GA WHERE GA.Game_Account_ID = MB.Game_Account_ID
                            AND GA.Time_Closed IS NOT NULL AND GA.Game_ID IS NOT NULL)
                    THEN 1 ELSE 0
                END
            )
            INTO result
            FROM Make_Bet MB JOIN Subround SR ON MB.Subround_ID = SR.Subround_ID
            JOIN Round R ON SR.Round_ID = R.Round_ID
            WHERE R.Round_ID = (
                SELECT SR.Round_ID FROM Subround SR
                WHERE SR.Subround_ID = p_Subround_ID
            );
        -- Secondly Add the number of total folds to the number of inactive accounts
        IF result IS NOT NULL THEN
            SET result = result + inactive_accounts;
        ELSE
            SET result = inactive_accounts;
        END IF;

    -- All the Folds/Game_Account_Closures of the Entire Round EXCEPT the Subround in Question
    ELSEIF stats_case = 'Outer_Fold_Count' THEN
        SELECT
            SUM(
                CASE
                    WHEN MB.Bet_Type_ID = (SELECT Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'Fold')
                    AND NOT EXISTS (
                            SELECT 1 FROM Game_Account GA WHERE GA.Game_Account_ID = MB.Game_Account_ID
                            AND GA.Time_Closed IS NOT NULL AND GA.Game_ID IS NOT NULL)
                    THEN 1 ELSE 0
                END
            )
        INTO result
        FROM Make_Bet MB
        JOIN Subround SR ON MB.Subround_ID = SR.Subround_ID
        JOIN Round R ON SR.Round_ID = R.Round_ID
        WHERE R.Round_ID = (
            SELECT SR.Round_ID
            FROM Subround SR
            WHERE SR.Subround_ID = p_Subround_ID
        )
        AND MB.Subround_ID != p_Subround_ID; -- Excludes the current subround

        IF result IS NOT NULL THEN
            SET result = result + inactive_accounts;
        ELSE
            SET result = inactive_accounts;
        END IF;

    ELSEIF stats_case = 'All_In_Count' THEN
        SELECT
            SUM(CASE
                WHEN MB.Bet_Type_ID = (SELECT Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'All-In')
AND
                NOT EXISTS (
                            SELECT 1
                            FROM Game_Account GA
                            WHERE GA.Game_Account_ID = MB.Game_Account_ID
                            AND GA.Time_Closed IS NOT NULL AND GA.Game_ID IS NOT NULL)
                THEN 1 ELSE 0 END)
        INTO result
            FROM Make_Bet MB
            JOIN Subround SR ON MB.Subround_ID = SR.Subround_ID
            JOIN Round R ON SR.Round_ID = R.Round_ID
            WHERE R.Round_ID = (
                SELECT SR.Round_ID FROM Subround SR
                WHERE SR.Subround_ID = p_Subround_ID
            );

    ELSEIF stats_case = 'Outer_All_In_Count' THEN
        SELECT
            SUM(CASE
                WHEN MB.Bet_Type_ID = (SELECT Bet_Type_ID FROM All_Bet_Types WHERE Bet_Type = 'All-In')
AND
                NOT EXISTS (
                            SELECT 1
                            FROM Game_Account GA
                            WHERE GA.Game_Account_ID = MB.Game_Account_ID
                            AND GA.Time_Closed IS NOT NULL AND GA.Game_ID IS NOT NULL)
                THEN 1 ELSE 0 END)
        INTO result
        FROM Make_Bet MB
        JOIN Subround SR ON MB.Subround_ID = SR.Subround_ID
        JOIN Round R ON SR.Round_ID = R.Round_ID
        WHERE R.Round_ID = (
            SELECT SR.Round_ID FROM Subround SR
            WHERE SR.Subround_ID = p_Subround_ID
        )
        AND MB.Subround_ID != p_Subround_ID; -- Exclude the current subround

    -- All Bets made in Subround
    ELSEIF stats_case = 'Subround_Bet_Count' THEN
        SELECT COUNT(*) INTO result FROM Make_Bet MB
        WHERE MB.Subround_ID = p_Subround_ID;

    -- All Game Accounts Closed or Open
    ELSEIF stats_case = 'Game_Accounts_Count' THEN
        SELECT COUNT(*) INTO result FROM Game_Account gA
        WHERE gA.Game_ID = (
            SELECT Game_ID FROM Game_Account WHERE Game_Account_ID = p_Game_Account_ID AND Game_ID IS NOT
NULL);

    -- Find the only non folded or non quit winner
    ELSEIF stats_case = 'Non_Folded_Winner' THEN
        SELECT MB.Game_Account_ID INTO result FROM Make_Bet MB
            JOIN Subround SR ON MB.Subround_ID = SR.Subround_ID
            JOIN All_Bet_Types BT ON MB.Bet_Type_ID = BT.Bet_Type_ID
            JOIN Game_Account GA ON MB.Game_Account_ID = GA.Game_Account_ID
            WHERE SR.Round_ID = (SELECT SR.Round_ID FROM Subround SR WHERE SR.Subround_ID =
p_Subround_ID)
                AND BT.Bet_Type != 'Fold'
                AND GA.Time_Closed IS NULL
            LIMIT 1;

    END IF;

    IF result IS NULL THEN
        SET result = 0;
    END IF;

    RETURN result;
END
```

The ComputeGameStats function is a conditional function that uses the given subround and game account to perform complex queries on determining the specific count specified by the conditional. For example, in finding the total folds of the given round, all the Make_Bet records from all the subrounds, given the current subround, needs be joined by the associated round, then filtered on the id that is of the type “fold” in the All_Bet_Types reference table, and is inclusive of all the players who have no become inactive having left the game in the middle of the round. These counts need to then be summed together and repeated for all the different betting types, leading to some of the most complex queries in designing the database.

RoundStateHandler

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `RoundStateHandler`(
    IN p_Subround_ID INT,
    IN p_Game_Account_ID INT,
    IN Game_Accounts_Count INT,
    IN Fold_Count INT,
    IN All_In_Count INT,
    IN Outer_Fold_Count INT,
    IN Outer_All_In_Count INT,
    IN Subround_Bet_Count INT
)
BEGIN

    DECLARE non_folded_winner_ID INT;

    DECLARE p_Subround_Pot DECIMAL(7, 3);
    DECLARE p_Subround_Type_ID INT;
    DECLARE p_Round_ID INT;
    DECLARE p_Register_ID INT;

    DECLARE Fold_ID INT;
    DECLARE Flop_ID INT;
    DECLARE Turn_ID INT;
    DECLARE River_ID INT;

    INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('Entered the Round State Handler'));
    SET Flop_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name = 'Flop');
    SET Turn_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name = 'Turn');
    SET River_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name = 'River');

    SET p_Subround_Type_ID = (SELECT Subround_Type_ID FROM Subround WHERE Subround_ID = p_Subround_ID);
    SET p_Subround_Pot = (SELECT Pot FROM Subround WHERE Subround_ID =p_Subround_ID);
    SET p_Round_ID = (SELECT Round_ID FROM Subround WHERE Subround_ID =p_Subround_ID);


    -- There is only one player who hasn't folded or left the game
    IF (Game_Accounts_Count - Fold_Count) = 1 THEN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered Only One Player hasnt folded'));
        -- Obtain only player who hasn't folded
        SET non_folded_winner_ID = ComputeGameStats(p_Subround_ID, p_Game_Account_ID,
'Non_Folded_Winner');

        -- Make Pay-Out
        CALL UpdateAccountBalance(p_Subround_ID, non_folded_winner_ID, p_Subround_Pot, 'Pay_Out');
        INSERT INTO Pay_Out (Pay_Out_Time, Subround_ID, Game_Account_ID, Pay_Out_Amount) VALUES
            (CURRENT_TIMESTAMP(6),p_Subround_ID, non_folded_winner_ID, p_Subround_Pot);

        -- Close Round; Start New Round
        CALL CloseRound(p_Subround_ID, p_Round_ID);

    -- All the players folded for some reason; as punishment, simply collect the winnings
    ELSEIF (Game_Accounts_Count - Fold_Count) = 0 THEN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Everyone folded'));

        SET p_Register_ID = (SELECT Register_ID FROM System_Register ORDER BY Register_ID ASC LIMIT 1);

        -- Rake Out Pot
        CALL UpdateAccountBalance(p_Subround_ID, p_Register_ID, p_Subround_Pot, 'Rake_Out');
        INSERT INTO Rake_Out (Rake_Time, Register_ID, Subround_ID, Rake_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Register_ID,p_Subround_ID, Rake_Tax);

        -- Close Subround, Round, New Round
        CALL CloseRound(p_Subround_ID, p_Round_ID);

    -- All the players have either folded or went all-in
    ELSEIF Game_Accounts_Count = (Fold_Count + All_In_Count) AND All_In_Count != 0 THEN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered All-In/Bet Mix Face-off'));
        -- Insert Remaining Comm cards
        IF p_Subround_Type_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name =
'Pre-Flop') THEN
            CALL InsertCards(5,'CC',p_Subround_ID, NULL);
        ELSEIF p_Subround_Type_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name
= 'Flop') THEN
            CALL InsertCards(2,'CC',p_Subround_ID, NULL);
        ELSEIF p_Subround_Type_ID = (SELECT Subround_Type_ID FROM All_Subround_Types WHERE Subround_Name
= 'Turn') THEN
            CALL InsertCards(1,'CC',p_Subround_ID, NULL);
        END IF;

        CALL DetermineWinner(p_Subround_ID);
        CALL CloseRound(p_Subround_ID, p_Round_ID);

    -- All the players have placed their bets and the Subround is the River (last subround)
    ELSEIF p_Subround_Type_ID = River_ID AND Subround_Bet_Count = (Game_Accounts_Count- (Outer_Fold_Coun
+ Outer_All_In_Count)) THEN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered Everyone Bet and Facing off on River'));

        CALL DetermineWinner(p_Subround_ID);
        CALL CloseRound(p_Subround_ID, p_Round_ID);

    -- All the players have placed their bets and the Subround is NOT the river
    -- Close Current Subround, and Start a New Subround
    -- Move Pot to new Subround type
    ELSEIF p_Subround_Type_ID != River_ID AND Subround_Bet_Count = (Game_Accounts_Count-
(Outer_Fold_Count+ Outer_All_In_Count)) THEN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered New Subround'));
        CALL CloseSubround(p_Subround_ID, p_Subround_Type_ID, p_Subround_Pot, p_Round_ID, Flop_ID,
Turn_ID, River_ID);
    END IF;
END
```

The RoundStateHandler is conditional procedure that evaluates the different counts generated by ComputeGameStats, to determine whether the round/subround needs to be closed after the winner has been determined, whether the subround needs to be closed and moved to the next subround, or whether betting should simply continue. This is determined by five major conditionals,

1. Only one player remains in the round, and the winner can be determined outright before round/subround closure.
2. No players remain for whatever reason, wherein the remaining pot balance is sent over to the System_Register as a rake out transaction.
3. There is a mixture of all-ins and folds before the “river” subround was reached, and thus, the rest of the community cards needs to be produced and then used in evaluating the winner through DetermineWinner, before closing the round/subround and potentially starting a new one.
4. There is a mixture of bets/folds/quits/all-ins bet types but all the players of the given subround were able to make their bets and the “river” subround type was reached, thus necessitating find the winner through DetermineWinner, before closing the round/subround and potentially starting a new one.
5. All the players who could make a bet i.e. haven’t folded, went all-in, or quit, did make a bet for the given subround, but none of the prior conditionals were met meaning the closure of the subround and the instantiation of a new one.

Subround Propagates

CloseSubround

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CloseSubround`(
    IN Open_Subround_ID INT,
    IN p_Subround_Type_ID INT,
    IN p_Subround_Pot DECIMAL(7, 3),

    IN Open_Round_ID INT,
    IN Flop_ID INT,
    IN Turn_ID INT,
    IN River_ID INT

)
BEGIN
    DECLARE Open_Game_ID INT;
    DECLARE Game_End_Time TIMESTAMP(6);
    DECLARE Round_End_Time TIMESTAMP(6);
    DECLARE Current_Subround_Name VARCHAR(20);
    INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered CloseSubround '));
    SET Open_Game_ID = (SELECT Game_ID FROM Round WHERE Round_ID = Open_Round_ID);
    SET Game_End_Time = (SELECT End_Time FROM Game WHERE Game_ID = Open_Game_ID);
    SET Round_End_Time = (SELECT End_Time FROM Round WHERE Round_ID = Open_Round_ID);
    SET Current_Subround_Name = (SELECT Subround_Name FROM All_Subround_Types WHERE Subround_Type_ID =
p_Subround_Type_ID);

    UPDATE Subround
    SET End_Time = CURRENT_TIMESTAMP(6)
    WHERE Subround_ID = Open_Subround_ID;

    IF (Game_End_Time IS NULL) AND (Round_End_Time IS NULL) THEN
        CASE Current_Subround_Name
            WHEN 'Pre-Flop' THEN
                INSERT INTO Subround (Round_ID, Subround_Type_ID, Start_Time, End_Time, Pot)
                    VALUES (Open_Round_ID, Flop_ID, CURRENT_TIMESTAMP(6), NULL, p_Subround_Pot);
            WHEN 'Flop' THEN
                INSERT INTO Subround (Round_ID, Subround_Type_ID, Start_Time, End_Time, Pot)
                    VALUES (Open_Round_ID, Turn_ID, CURRENT_TIMESTAMP(6), NULL, p_Subround_Pot);
            WHEN 'Turn' THEN
                INSERT INTO Subround (Round_ID, Subround_Type_ID, Start_Time, End_Time, Pot)
                    VALUES (Open_Round_ID, River_ID, CURRENT_TIMESTAMP(6), NULL, p_Subround_Pot);
        END CASE;
    END IF;
END
```

After all the bets have been placed for the given subround conditional (5) of the RoundStateHandler is reached and the subround needs to be propagated forward. To do this, the CloseSubround procedure is called whereby the given subround record's end

time is updated from null to the current time of closure, and a new subround record is inserted into the table depending on the subround type.

Community_Cards_Trigger

```
-- Creates Community Cards for each Subround on new Subround Insert
DELIMITER //
CREATE TRIGGER Community_Cards_Trigger
AFTER INSERT ON Subround
FOR EACH ROW
    BEGIN

        DECLARE Curr_Subround_Name VARCHAR(20);
        DECLARE Random_Card_ID INT;

        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered the Community_Cards_Trigger'));
        SELECT Subround_Name INTO Curr_Subround_Name FROM All_Subround_Types WHERE Subround_Type_ID =
NEW.Subround_Type_ID;
        IF Curr_Subround_Name != 'Pre-Flop' THEN
            IF Curr_Subround_Name = 'Flop' THEN
                CALL InsertCards(3, 'CC', NEW.Subround_ID, NULL);
            ELSE
                CALL InsertCards(1, 'CC', NEW.Subround_ID, NULL);
            END IF;
        END IF;
    END//
```

Upon the insertion of the new subround record, the Community_Cards_Trigger is activated. This in turn calls the InsertCards procedure to insert the number of community card records corresponding to the given subround.

Winner is Determined; Round Closes

At the point within the subrounds a bet is made that completes any of the other four conditionals of the RoundStateHandler. In the case that conditional (3) or (4) is met the DetermineWinner procedure is called. This procedure is involved with comparing each of the player's cards/generated community cards, calculating the sum totals, then identifying the winner(s) of the game. As per our modified rules of poker, if a player holds an even card, their sum total is determined by the sum of their card and all even community cards for the round, and vice versa if their card is odd.

SumCommCards

```
CREATE DEFINER=`root`@`localhost` FUNCTION `SumCommCards`(
    p_sum_param INT,
    p_Round_ID INT
) RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE Comm_Card_Total INT;
    -- Even Sum
    IF p_sum_param = 0 THEN
        SELECT SUM(AC.Card_Combination) INTO Comm_Card_Total
        FROM Subround SR
        JOIN Community_Cards CC ON CC.Subround_ID = SR.Subround_ID
        JOIN All_Card_Types AC ON AC.Card_Type_ID = CC.Card_Type_ID
        WHERE SR.Round_ID = p_Round_ID
          AND MOD(AC.Card_Combination, 2) = 0;
    -- Odd
    ELSE
        SELECT SUM(AC.Card_Combination) INTO Comm_Card_Total
        FROM Subround SR
        JOIN Community_Cards CC ON CC.Subround_ID = SR.Subround_ID
        JOIN All_Card_Types AC ON AC.Card_Type_ID = CC.Card_Type_ID
        WHERE SR.Round_ID = p_Round_ID
          AND MOD(AC.Card_Combination, 2) = 1;
END IF;

RETURN Comm_Card_Total;
END
```

To achieve this the SumCommCards function is called upon with the associated Round_ID of the game, and the conditional specifying whether the even/odd total should be calculated. A join on all the community card records of all the subrounds where the subround holds a reference to the current round, as well as a join on the Card_Types_ID from the All_Card_Types table where it matches the community cards reference id, is performed and filtered on the whether it is even/odd.

This is returned back to the DetermineWinner procedure which then performs a similar join but with the player card types associated with all the game accounts that have a bet type record corresponding to “All-In”/”Bet” and do not have non null end times. These are then totaled together with the previous community card even/odd totals, and the maximum value is found with the associated game account id. For all these winning accounts, the pot is subdivided amongst them, their account balances are changed, and payout records are generated.

In the event that individuals who had gone all-in had lost, these individuals would have a negative payout record made, whereby the 0.001 amount left in their account to prevent premature closure during the round, is now extracted back to trigger a closure of the game account downstream.

The complete DetermineWinner procedure can be seen below,

DetermineWinner

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `DetermineWinner`(
    IN p_Subround_ID INT
)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE max_total INT;
    DECLARE pot_value DECIMAL(7, 3);
    DECLARE num_winners INT DEFAULT 0;
    DECLARE payout_amount DECIMAL(7, 3);

    DECLARE p_Round_ID INT;
    DECLARE p_Even_Total INT;
    DECLARE p_Odd_Total INT;

    DECLARE winner_Game_Account_ID INT;
    DECLARE loser_Game_Account_ID INT;

    -- Set up cursors to interate through winners and all in losers
    DECLARE winners_cursor CURSOR FOR SELECT Game_Account_ID FROM TempWinners;
    DECLARE losers_cursor CURSOR FOR SELECT Game_Account_ID FROM TempLosers;

    -- Cursor Handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Compute Round Information
    SET p_Round_ID = (SELECT Round_ID FROM Subround WHERE Subround_ID = p_Subround_ID);
    SET p_Even_Total = SumCommCards(0, p_Round_ID);
    SET p_Odd_Total = SumCommCards(1, p_Round_ID);

     INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('Entered Determine Winner'));
    -- Create Temporary Table for Base Data
    DROP TEMPORARY TABLE IF EXISTS TempBaseData;
    CREATE TEMPORARY TABLE TempBaseData AS
    SELECT
        MB.Game_Account_ID,
        AC.card_combination,
        BT.Bet_Type
    FROM Make_Bet MB
    JOIN All_Bet_Types BT ON MB.Bet_Type_ID = BT.Bet_Type_ID
    JOIN Players_Cards PC ON MB.Game_Account_ID = PC.Game_Account_ID AND PC.Round_ID = p_Round_ID
    JOIN All_Card_Types AC ON PC.Card_Type_ID = AC.Card_Type_ID
    WHERE MB.Subround_ID = p_Subround_ID
      AND BT.Bet_Type IN ('All-In', 'Bet');

    -- Compute Adjusted Totals (player card + comm card)
    DROP TEMPORARY TABLE IF EXISTS TempAdjustedTotals;
    CREATE TEMPORARY TABLE TempAdjustedTotals AS
    SELECT
        Game_Account_ID,
        CASE
            WHEN MOD(card_combination, 2) = 0 THEN card_combination + p_Even_Total
            ELSE card_combination + p_Odd_Total
        END AS Adjusted_Total
    FROM TempBaseData;

    SET max_total = (SELECT MAX(Adjusted_Total) FROM TempAdjustedTotals);

    -- Identify Winners
    DROP TEMPORARY TABLE IF EXISTS TempWinners;
    CREATE TEMPORARY TABLE TempWinners AS
    SELECT Game_Account_ID
    FROM TempAdjustedTotals
    WHERE Adjusted_Total = max_total;

    -- Identify Losers (All-In but Not Winners) this is for negative payouts
    DROP TEMPORARY TABLE IF EXISTS TempLosers;
    CREATE TEMPORARY TABLE TempLosers AS
    SELECT BD.Game_Account_ID
    FROM TempBaseData BD
    WHERE BD.Bet_Type = 'All-In'
      AND BD.Game_Account_ID NOT IN (SELECT Game_Account_ID FROM TempWinners);

    -- SELECT * FROM TempAdjustedTotals;
    -- SELECT * FROM TempWinners;
    -- SELECT * FROM TempLosers;
    INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('There are ', num_winners, ' total winners, with a Max_Total of ', max_total));
    -- Count Winners and Compute Payout
    SELECT COUNT(*) INTO num_winners FROM TempWinners;
    SELECT Pot INTO pot_value FROM Subround WHERE Subround_ID = p_Subround_ID;
    SET payout_amount = pot_value / num_winners;

    -- Process all Winners
    OPEN winners_cursor;
    payout_loop: LOOP
        FETCH winners_cursor INTO winner_Game_Account_ID;
        IF done THEN
            LEAVE payout_loop;
        END IF;

        -- Update Balance and Record Payout
        CALL UpdateAccountBalance(p_Subround_ID, winner_Game_Account_ID, payout_amount, 'Pay_Out');
        INSERT INTO Pay_Out (Pay_Out_Time, Subround_ID, Game_Account_ID, Pay_Out_Amount)
        VALUES (CURRENT_TIMESTAMP(6), p_Subround_ID, winner_Game_Account_ID, payout_amount);
    END LOOP;
    CLOSE winners_cursor;

    -- Reset for Next Cursor so we can process losers
    SET done = FALSE;

    -- P@rocess all the Losers
    OPEN losers_cursor;
    negative_payout_loop: LOOP
        FETCH losers_cursor INTO loser_Game_Account_ID;
        IF done THEN
            LEAVE negative_payout_loop;
        END IF;
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('All In Loser'));
        -- Update Balance and Record Negative Payout
        CALL UpdateAccountBalance(p_Subround_ID, loser_Game_Account_ID, -0.001, 'Pay_Out');
        INSERT INTO Pay_Out (Pay_Out_Time, Subround_ID, Game_Account_ID, Pay_Out_Amount)
        VALUES (CURRENT_TIMESTAMP(6), p_Subround_ID, loser_Game_Account_ID, -0.001);
    END LOOP;
    CLOSE losers_cursor;

    DROP TEMPORARY TABLE IF EXISTS TempBaseData;
    DROP TEMPORARY TABLE IF EXISTS TempAdjustedTotals;
    DROP TEMPORARY TABLE IF EXISTS TempWinners;
    DROP TEMPORARY TABLE IF EXISTS TempLosers;

END
```

CloseRound

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CloseRound`(
    IN Open_Subround_ID INT,
    IN Open_Round_ID INT
)
BEGIN
        DECLARE Open_Game_ID INT;
        DECLARE Game_End_Time TIMESTAMP(6);
        INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered Round '));
        SET Open_Game_ID = (SELECT Game_ID FROM Round WHERE Round_ID = Open_Round_ID);
        SET Game_End_Time = (SELECT End_Time FROM Game WHERE Game_ID = Open_Game_ID);

        UPDATE Subround
        SET End_Time = CURRENT_TIMESTAMP(6)
        WHERE Subround_ID = Open_Subround_ID;

        UPDATE Round
        SET End_Time = CURRENT_TIMESTAMP(6)
        WHERE Round_ID = Open_Round_ID;

        IF Game_End_Time IS NULL THEN
            INSERT INTO Round (Game_ID, Start_Time, End_Time)
                    VALUES (Open_Game_ID, CURRENT_TIMESTAMP(6), NULL);
        END IF;
END
```

After the winner is determined, or if the winner was found outright or the pot was paid to the register through conditionals (1)/(2), the CloseRound procedure is called upon to end the given round. Given the current subround and round, both end times are updated to the current timestamp.

Close_Game_Account_Trigger

```
-- Sets the Game_ID NULL of all Game_Accounts that are closed with that Round
DELIMITER //
CREATE TRIGGER Close_Game_Account_Trigger
AFTER UPDATE ON Round
FOR EACH ROW
    BEGIN
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Entered the Close_Game_Account_Trigger'));
        IF NEW.End_Time IS NOT NULL THEN
            IF NOT EXISTS (SELECT * FROM Subround SR WHERE SR.Round_ID = NEW.Round_ID AND SR.End_Time IS
NULL) THEN
                INSERT INTO Debug_Temp (message)
                VALUES (CONCAT('Players has left game; their Game_ID is being nullified'));
                UPDATE Game_Account GA
                SET GA.Game_ID = NULL
                WHERE GA.Time_Closed IS NOT NULL AND GA.Game_ID = NEW.Game_ID;
            ELSE
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Game Closure Hierarchy Violated For Round,
Subround still active';
            END IF;
        END IF;
    END//
```

Upon this update yet another trigger, Close_Game_Account_Trigger, is activated. This trigger is made to set all the game accounts with a non null end time to have their game ids set to null, essentially deactivating them completely. This is performed to ensure that the account is no longer able to be referenced in future rounds when performing calculations of the number of total players and allows differentiation of accounts that quit during a given round, and those who were deactivated long prior.

Example Scenario (SafeMakeBet)

To show the complete SafeMakeBet Cascade we can continue with our previous example, wherein the six players have bought into the game, the game got started, player cards were distributed, and a round/subround record was generated.

To do this we will show a full round completion whereby all the players bet continuously for each subround until the “river” subround type. With regards to the RoundStateHandler this would test conditionals (5) and (4), as a new subround record would have to be generated after all bets are placed (5), and the round would have to end once the river has been hit as all the players are within the game (4).

To perform this we run each of the bets for each of the players as follows,

```
-- Pre-Flop
CALL SafeMakeBet(1, 1);
CALL SafeMakeBet(2, 1);
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Flop
CALL SafeMakeBet(1, 1);
CALL SafeMakeBet(2, 1);
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Turn
CALL SafeMakeBet(1, 1);
CALL SafeMakeBet(2, 1);
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);
```

```
-- River
CALL SafeMakeBet(1, 1);
CALL SafeMakeBet(2, 1);
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Show Player Cards, Community Cards, The Pay-Out, The Game_account to show winner got the money
-- Show all the bets placed in the round
-- Show that a new round is generated upon round closure and a new subround starts
SELECT * FROM Community_Cards;
SELECT * FROM Players_Cards;
SELECT * FROM Pay_Out;
SELECT * FROM Game_Account;
SELECT * FROM Make_Bet;
SELECT * FROM Subround;
SELECT * FROM Round;
```

It's important to note that as pointed out earlier the limit type id for the initialized game is equal to 1, meaning that the bet limit is 10 cents. However, regardless, even if the player tries to make a bet of any amount greater than or less than that (unless 0), the SafeMakeBet handles it to default to 10 cents unless their game account is forced to go all in. Thus, for this example setting the bet amounts to one does not make a difference.

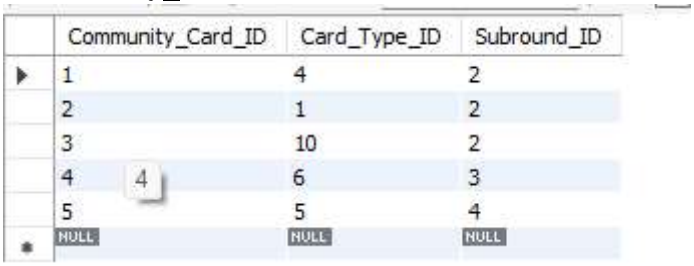
Make_Bet

Bet_Time	Subround_ID	Game_Account_ID	Bet_Type_ID	Bet_Amount
2024-11-20 15:26:08.839360	1	1	2	0.100
2024-11-20 15:26:08.892991	1	2	2	0.100
2024-11-20 15:26:08.925729	1	3	2	0.100
2024-11-20 15:26:08.953444	1	4	2	0.100
2024-11-20 15:26:08.980860	1	5	2	0.100
2024-11-20 15:35:30.125480	1	6	2	0.100
2024-11-20 15:35:41.193907	2	1	2	0.100
2024-11-20 15:35:41.223174	2	2	2	0.100
2024-11-20 15:35:41.251229	2	3	2	0.100
2024-11-20 15:35:41.278844	2	4	2	0.100
2024-11-20 15:35:41.304454	2	5	2	0.100
2024-11-20 15:35:41.329828	2	6	2	0.100
2024-11-20 15:35:41.354611	3	1	2	0.100
2024-11-20 15:35:41.381616	3	2	2	0.100
2024-11-20 15:35:41.406196	3	3	2	0.100
2024-11-20 15:35:41.432521	3	4	2	0.100
2024-11-20 15:35:41.457920	3	5	2	0.100
2024-11-20 15:35:41.484368	3	6	2	0.100

Bet_Time	Subround_ID	Game_Account_ID	Bet_Type_ID	Bet_Amount
2024-11-20 15:35:41.251229	2	3	2	0.100
2024-11-20 15:35:41.278844	2	4	2	0.100
2024-11-20 15:35:41.304454	2	5	2	0.100
2024-11-20 15:35:41.329828	2	6	2	0.100
2024-11-20 15:35:41.354611	3	1	2	0.100
2024-11-20 15:35:41.381616	3	2	2	0.100
2024-11-20 15:35:41.406196	3	3	2	0.100
2024-11-20 15:35:41.432521	3	4	2	0.100
2024-11-20 15:35:41.457920	3	5	2	0.100
2024-11-20 15:35:41.484368	3	6	2	0.100
2024-11-20 15:35:41.512612	4	1	2	0.100
2024-11-20 15:35:41.538856	4	2	2	0.100
2024-11-20 15:35:41.564169	4	3	2	0.100
2024-11-20 15:35:41.588230	4	4	2	0.100
2024-11-20 15:35:41.615789	4	5	2	0.100
2024-11-20 15:35:41.641524	4	6	2	0.100
NULL	NULL	NULL	NULL	NULL

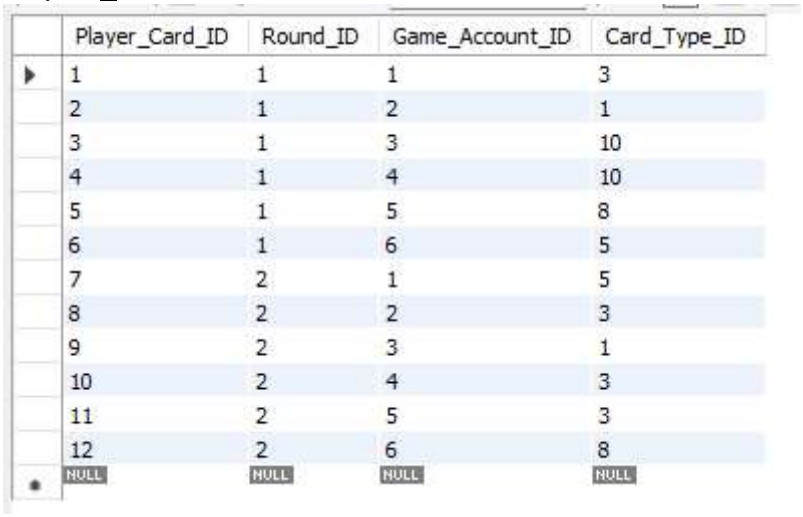
As we can see, the Make_Bet records are successfully generated for each subround, totaling to the six players who bet for each subround. Furthermore the bet type id can be seen to be of the value 2, corresponding to the bet type “bet”, and the bet amount is all defaulted to the limit of the game at 10 cents.

Community_Cards



Community_Card_ID	Card_Type_ID	Subround_ID
1	4	2
2	1	2
3	10	2
4	6	3
5	5	4
NULL	NULL	NULL

Players_Cards



Player_Card_ID	Round_ID	Game_Account_ID	Card_Type_ID
1	1	1	3
2	1	2	1
3	1	3	10
4	1	4	10
5	1	5	8
6	1	6	5
7	2	1	5
8	2	2	3
9	2	3	1
10	2	4	3
11	2	5	3
12	2	6	8
NULL	NULL	NULL	NULL

In a similar vein, five total community cards are created for each of the corresponding subrounds, successfully generating them.

Pay_Out

Pay_Out_Time	Subround_ID	Game_Account_ID	Pay_Out_Amount
2024-11-20 15:35:41.641524	4	3	1.220
2024-11-20 15:35:41.641524	4	4	1.220
NULL	NULL	NULL	NULL

In this round, two players won the game, player 3 and 4, and received a split pot pay out. Looking back at the player cards we see the reason why as they both had the card number 10, leading to the highest max total for both players.

Rake_Out

Rake_Time	Register_ID	Subround_ID	Rake_Amount
2024-11-20 15:26:08.875518	1	1	0.003
2024-11-20 15:26:08.911345	1	1	0.003
2024-11-20 15:26:08.939428	1	1	0.003
2024-11-20 15:26:08.967681	1	1	0.003
2024-11-20 15:26:08.996908	1	1	0.003
2024-11-20 15:35:30.164309	1	1	0.003
2024-11-20 15:35:41.209578	1	2	0.003
2024-11-20 15:35:41.238122	1	2	0.003
2024-11-20 15:35:41.265612	1	2	0.003
2024-11-20 15:35:41.291912	1	2	0.003
2024-11-20 15:35:41.317219	1	2	0.003
2024-11-20 15:35:41.342483	1	2	0.003
2024-11-20 15:35:41.367632	1	3	0.003
2024-11-20 15:35:41.394511	1	3	0.003
2024-11-20 15:35:41.419324	1	3	0.003
2024-11-20 15:35:41.445499	1	3	0.003
2024-11-20 15:35:41.470329	1	3	0.003
2024-11-20 15:35:41.496994	1	3	0.003
2024-11-20 15:35:41.367632	1	3	0.003
2024-11-20 15:35:41.394511	1	3	0.003
2024-11-20 15:35:41.419324	1	3	0.003
2024-11-20 15:35:41.445499	1	3	0.003
2024-11-20 15:35:41.470329	1	3	0.003
2024-11-20 15:35:41.496994	1	3	0.003
2024-11-20 15:35:41.526087	1	4	0.003
2024-11-20 15:35:41.552127	1	4	0.003
2024-11-20 15:35:41.576409	1	4	0.003
2024-11-20 15:35:41.603282	1	4	0.003
2024-11-20 15:35:41.628953	1	4	0.003
NULL	NULL	NULL	NULL

Since all the bets in the game were of the type “bet” all the players had a rake out transaction performed on their bets.

Game_Account

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	0.600	2024-11-20 15:03:26.346243	NULL
2	1	1.600	2024-11-20 15:09:16.331061	NULL
3	1	1.820	2024-11-20 15:18:14.611226	NULL
4	1	1.820	2024-11-20 15:18:14.637906	NULL
5	1	0.600	2024-11-20 15:18:14.659689	NULL
6	1	0.600	2024-11-20 15:18:14.678511	NULL
NULL	NULL	NULL	NULL	NULL

Notice that all players chip stack are equal except the winners who also are equal amongst themselves, and player 2 because they bought in twice. Four bets were made for each player, one for each subround, thus their chip stack only decreased by forty cents.

Subround

Subround_ID	Round_ID	Subround_Type_ID	Start_Time	End_Time	Pot
1	1	1	2024-11-20 15:09:16.331061	2024-11-20 15:35:30.125480	0.682
2	1	2	2024-11-20 15:35:30.125480	2024-11-20 15:35:41.329828	1.267
3	1	3	2024-11-20 15:35:41.329828	2024-11-20 15:35:41.484368	1.852
4	1	4	2024-11-20 15:35:41.484368	2024-11-20 15:35:41.641524	0.000
5	2	1	2024-11-20 15:35:41.641524	NULL	0.000
NULL	NULL	NULL	NULL	NULL	NULL

Round

Round_ID	Game_ID	Start_Time	End_Time
1	1	2024-11-20 15:09:16.331061	2024-11-20 15:35:41.641524
2	1	2024-11-20 15:35:41.641524	NULL
NULL	NULL	NULL	NULL

The records within subround and Round show that a new round was created after completion and a new subround was created at each stage, the pot was cleared after payout.

1 15:35:30	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.109 sec
2 15:35:41	CALL SafeMakeBet(1, 1)	1 row(s) affected	0.031 sec
3 15:35:41	CALL SafeMakeBet(2, 1)	1 row(s) affected	0.016 sec
4 15:35:41	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
5 15:35:41	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.031 sec
6 15:35:41	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.016 sec
7 15:35:41	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
8 15:35:41	CALL SafeMakeBet(1, 1)	1 row(s) affected	0.031 sec
9 15:35:41	CALL SafeMakeBet(2, 1)	1 row(s) affected	0.016 sec
10 15:35:41	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
11 15:35:41	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.031 sec
12 15:35:41	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.016 sec
13 15:35:41	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
14 15:35:41	CALL SafeMakeBet(1, 1)	1 row(s) affected	0.031 sec
15 15:35:41	CALL SafeMakeBet(2, 1)	1 row(s) affected	0.016 sec
16 15:35:41	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
17 15:35:41	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.032 sec
18 15:35:41	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.015 sec
19 15:35:41	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec

20 15:36:49	SELECT * FROM Community_Cards LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
21 15:37:06	SELECT * FROM Community_Cards LIMIT 0, 1000	5 row(s) returned	0.016 sec / 0.000 sec
22 15:37:06	SELECT * FROM Players_Cards LIMIT 0, 1000	12 row(s) returned	0.000 sec / 0.000 sec
23 15:37:06	SELECT * FROM Pay_Out LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
24 15:37:06	SELECT * FROM Rake_Out LIMIT 0, 1000	23 row(s) returned	0.000 sec / 0.000 sec
25 15:37:06	SELECT * FROM Game_Account LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
26 15:37:06	SELECT * FROM Make_Bet LIMIT 0, 1000	24 row(s) returned	0.000 sec / 0.000 sec
27 15:37:06	SELECT * FROM Subround LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
28 15:37:06	SELECT * FROM Round LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

SafeBuyOut Cascade:

Player Buys Out, Game Account Closes

Mandatorily Activated Features (Unique)

Procedures: SafeBuyOut
Functions: None
Triggers: None

Conditionally Activated Features (Unique)

Procedures: PlayerLossGameClosure
Functions: None
Triggers: None

Player Buys Out

SafeBuyOut

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `SafeBuyOut`(
    IN p_Game_Account_ID INT,
    IN p_Buy_Out_Amount DECIMAL(7, 3)

)
BEGIN
    DECLARE p_Player_ID INT default NULL;
    SET p_Player_ID = (SELECT Player_ID FROM Buy_In WHERE Game_Account_ID = p_Game_Account_ID LIMIT 1);
    INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Entered SafeBuyOut ', p_Chip_Stack));
    CALL VerifyTransactor(p_Game_Account_ID, p_Buy_Out_Amount, 'Game_Account');
    CALL VerifyOpen(p_Game_Account_ID, p_Player_ID, 'Buy_Out');
    CALL UpdateAccountBalance(p_Game_Account_ID, p_Player_ID, p_Buy_Out_Amount, 'Buy_Out');
    INSERT INTO Buy_Out (Buy_Out_Time, Game_Account_ID, Player_ID, Buy_Out_Amount)
            VALUES (CURRENT_TIMESTAMP(6), p_Game_Account_ID, p_Player_ID, p_Buy_Out_Amount);


END
```

The SafeBuyOut procedure is the last control procedure in manipulating and generating records to the database by the user. At any time within the game that a game account is participating in, a player can choose to buy out of the game, either partially or fully, returning some or all of their earnings within the game, back into their own accounts. To do this the player calls upon the SafeBuyOut procedure with their game account id and the amount wanting to buy out for. From here, the transactor i.e. the game account, is verified through VerifyTransactor, in the same way that the SafeBuyIn was verified, making sure the account id exists, is not null, and that the buyout amount is within balance. From here, the VerifyOpen procedure is called to ensure that the game account is not already closed, and that there exists a player record in the Buy_In table that corresponds to the game account in question upon initial buy in.

From here the UpdateAccountBalance procedure is called to update the balance of both the game/player accounts, and a buyout record is generated.

Game Account Closes

PlayerLossGameClosure

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `PlayerLossGameClosure`(
    IN p_Game_ID INT,
    IN p_Game_Account_ID INT
)
BEGIN
    DECLARE Total_Game_Accounts INT DEFAULT 0;
    DECLARE Open_Subround_ID INT;
    DECLARE Open_Round_ID INT;
    DECLARE Open_Subround_Pot DECIMAL(7, 3);
    DECLARE Remaining_Game_Account_ID INT;

    UPDATE Game_Account
    SET Time_Closed = CURRENT_TIMESTAMP(6)
    WHERE Game_Account_ID = p_Game_Account_ID;

    INSERT INTO Debug_Temp (message)
    VALUES (CONCAT('Entered the PlayerLossGameClosure '));

    -- Count all the closed Game_Accounts associated with the Game before account closure
    -- Not including the current Game account as it has yet to close
    SELECT COUNT(*) INTO Total_Game_Accounts FROM Game_Account
    WHERE Game_ID = p_Game_ID AND Time_Closed IS NULL AND Game_Account_ID != p_Game_Account_ID;


    -- If there are one or few players in the game,
    -- Make payout of the remaining pot, force buy_out, and end subround, round, and game
    IF Total_Game_Accounts <=1 THEN

        IF Total_Game_Accounts = 1 THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Everyone except 1 player has quit the game'));
            SELECT Game_Account_ID INTO Remaining_Game_Account_ID FROM Game_Account
            WHERE Game_ID = p_Game_ID AND Time_Closed IS NULL AND Game_Account_ID != p_Game_Account_ID;
        ELSE
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Everyone has quit the game'));
            SET Remaining_Game_Account_ID = p_Game_Account_ID;
        END IF;

        SELECT Round_ID INTO Open_Round_ID FROM Round
            WHERE Game_ID = p_Game_ID AND End_Time IS NULL;

        SELECT Subround_ID INTO Open_Subround_ID FROM Subround
            WHERE Round_ID = Open_Round_ID AND End_Time IS NULL;

        SET Open_Subround_Pot = (SELECT Pot From Subround WHERE Subround_ID = Open_Subround_ID);

        -- Sees whether there is an open Subround which has a remaining pot
        -- If so, then pay_out to the player is made to their game account and the entire game closes

        IF Open_Subround_ID IS NOT NULL AND (Open_Subround_Pot IS NOT NULL AND Open_Subround_Pot > 0)
THEN
            INSERT INTO Debug_Temp (message)
            VALUES (CONCAT('Remaining pot is going to the player'));
            CALL UpdateAccountBalance(Open_Subround_ID, Remaining_Game_Account_ID, Open_Subround_Pot,
'Pay_Out');
            INSERT INTO Pay_Out (Pay_Out_Time, Subround_ID, Game_Account_ID, Pay_Out_Amount)
            VALUES (CURRENT_TIMESTAMP(6), Open_Subround_ID, Remaining_Game_Account_ID,
Open_Subround_Pot);


        END IF;

        UPDATE Subround
        SET End_Time = CURRENT_TIMESTAMP(6)
        WHERE Subround_ID = Open_Subround_ID;

        UPDATE Round
        SET End_Time = CURRENT_TIMESTAMP(6)
        WHERE Round_ID = Open_Round_ID;

        UPDATE Game
        SET End_Time = CURRENT_TIMESTAMP(6)
        WHERE Game_ID = p_Game_ID;

        UPDATE Game_Account
        SET Time_Closed = CURRENT_TIMESTAMP(6)
        WHERE Game_Account_ID = Remaining_Game_Account_ID;
        INSERT INTO Debug_Temp (message)
        VALUES (CONCAT('Closing out the Game'));
    END IF;
END
```

In the event that the buy-out clears the game account completely, when the UpdateAccountBalance procedure is called by the SafeBuyOut procedure, the PlayerLossGameClosure procedure is called. This procedure performs three separate functions, with the first being to close the game account due to holding a balance that is zero. From here, the game id associated with the game account is retrieved and all the associated game accounts with a null end time (active players), are counted.

In the case that the active players are one or less, a closure of the entire subround, round, and game is initiated. Furthermore, if there is only one active player left, that player's id is obtained. Given this id, the pot of the subround is checked to see whether there exists any money left in the pot, if so, the last remaining player in the game receives a pay out of the pot to their game account. After this, both the game account and game are closed completely.

This process is used to ensure that in the case that a game loses all its players except one, the remaining player still gets back the money invested into the pot and is not left in a stagnant game.

Example Scenario (SafeBuyOut)

We will now continue with our example to test both the buyout functionality along with seeing how clearing out some of the members of the game affects game play. To do this we will have players 1 and 2, buyout of the game on the preflop, then have two more players fold, one player fold on the flop, and two on the turn. The RoundStateHandler should recognize that three players folded and that two left the game, making the last player remaining win the pot.

```
- Show Account Game Account Closes
ALL SafeBuyOut(1, 0.600);
ALL SafeBuyOut(2, 1.600);

ELECT * FROM Game_Account;
ELECT * FROM Buy_Out;
ELECT * FROM Player;
```

```
-- Pre-Flop
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Flop
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 0);

-- Turn
CALL SafeMakeBet(4, 0);
CALL SafeMakeBet(5, 0);
CALL SafeMakeBet(3, 1);
```

Game_Account

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	0.000	2024-11-20 15:03:26.346243	2024-11-20 16:01:39.291506
2	1	0.000	2024-11-20 15:09:16.331061	2024-11-20 16:01:39.327263
3	1	1.820	2024-11-20 15:18:14.611226	NULL
4	1	1.820	2024-11-20 15:18:14.637906	NULL
5	1	0.600	2024-11-20 15:18:14.659689	NULL
6	1	0.600	2024-11-20 15:18:14.678511	NULL
NULL	NULL	NULL	NULL	NULL

Buy_Out

Buy_Out_Time	Game_Account_ID	Player_ID	Buy_Out_Amount
2024-11-20 16:01:39.303701	1	1	0.600
2024-11-20 16:01:39.339387	2	2	1.600
NULL	NULL	NULL	NULL

As we can see, the buy out gets successfully recorded into the database and the game accounts get closed.

Pay_Out

Pay_Out_Time	Subround_ID	Game_Account_ID	Pay_Out_Amount
2024-11-20 15:35:41.641524	4	3	1.220
2024-11-20 15:35:41.641524	4	4	1.220
2024-11-20 16:05:41.239434	7	3	0.685
NULL	NULL	NULL	NULL

After then running all the subrounds, game account 3 is found to be the winner as he is the last one remaining who hadn't folded.

Game_Account

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	NULL	0.000	2024-11-20 15:03:26.346243	2024-11-20 16:01:39.291506
2	NULL	0.000	2024-11-20 15:09:16.331061	2024-11-20 16:01:39.327263
3	1	2.405	2024-11-20 15:18:14.611226	NULL
4	1	1.620	2024-11-20 15:18:14.637906	NULL
5	1	0.400	2024-11-20 15:18:14.659689	NULL
6	1	0.300	2024-11-20 15:18:14.678511	NULL
NULL	NULL	NULL	NULL	NULL

Bet, Account, Subround Query

Game_Account_ID	Bet_Type	Subround_ID	Bet_Amount	Subround_Name
3	Bet	5	0.100	Pre-Flop
4	Bet	5	0.100	Pre-Flop
5	Bet	5	0.100	Pre-Flop
6	Bet	5	0.100	Pre-Flop
3	Bet	6	0.100	Flop
4	Bet	6	0.100	Flop
5	Bet	6	0.100	Flop
6	Fold	6	0.000	Flop
4	Fold	7	0.000	Turn
5	Fold	7	0.000	Turn

1	16:35:07	CALL SafeBuyOut(1, 0.600)	1 row(s) affected	0.016 sec
2	16:35:07	CALL SafeBuyOut(2, 1.600)	1 row(s) affected	0.031 sec
3	16:35:07	SELECT * FROM Game_Account LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
4	16:35:07	SELECT * FROM Buy_Out LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
5	16:35:07	SELECT * FROM Player LIMIT 0, 1000	16 row(s) returned	0.000 sec / 0.000 sec
6	16:35:55	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.047 sec
7	16:35:55	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.031 sec
8	16:35:55	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.031 sec
9	16:35:55	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.032 sec
10	16:35:55	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
11	16:35:55	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.031 sec
12	16:35:55	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.031 sec
13	16:35:55	CALL SafeMakeBet(6, 0)	1 row(s) affected	0.016 sec
14	16:35:55	CALL SafeMakeBet(4, 0)	1 row(s) affected	0.016 sec
15	16:35:55	CALL SafeMakeBet(5, 0)	1 row(s) affected	0.000 sec
16	16:35:55	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.016 sec
17	16:35:55	SELECT * FROM Game_Account LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
18	16:35:55	SELECT * FROM Pay_Out LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
19	16:40:25	SELECT MB.Game_Account_ID, BT.Bet_Type, MB.Subround_ID, MB.Bet_A...	10 row(s) returned	0.000 sec / 0.000 sec

*Notice that the last bet of CALL SafeMakeBet(3,1) was not completed for the given round as all other players folded. This bet got carried over to the next round.

Since the game account players 5 and 6 are equal to or below 40 cents, both players will end up going all in if they bet continuously on the next round. As such, we can test how betting into the next round, continuously, will make the players go all-in and see how the game handles that.

```
-- Tests the Main conditional
-- All the players have either folde
-- Game_Accounts_Count = (Fold_Count
-- Pre-Flop
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Flop
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(6, 1);

-- Turn

CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(6, 1);

-- River
CALL SafeMakeBet(3, 1);
CALL SafeMakeBet(4, 1);
CALL SafeMakeBet(5, 1);
```

Game Account

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	NULL	0.000	2024-11-20 15:03:26.346243	2024-11-20 16:01:39.291506
2	NULL	0.000	2024-11-20 15:09:16.331061	2024-11-20 16:01:39.327263
3	1	3.570	2024-11-20 15:18:14.611226	NULL
4	1	1.220	2024-11-20 15:18:14.637906	NULL
5	NULL	0.000	2024-11-20 15:18:14.659689	2024-11-20 16:25:22.686326
6	NULL	0.000	2024-11-20 15:18:14.678511	2024-11-20 16:25:22.686326
NULL	NULL	NULL	NULL	NULL

Bet, Account, Subround Query

Game_Account_ID	Bet_Type	Subround_ID	Bet_Amount	Subround_Name
5	Bet	8	0.100	Pre-Flop
4	Bet	8	0.100	Pre-Flop
3	Bet	8	0.100	Pre-Flop
6	Bet	8	0.100	Pre-Flop
5	Bet	9	0.100	Flop
4	Bet	9	0.100	Flop
3	Bet	9	0.100	Flop
6	Bet	9	0.100	Flop
3	Bet	10	0.100	Turn
5	Bet	10	0.100	Turn
4	Bet	10	0.100	Turn
6	Bet	10	0.100	Turn
4	Bet	11	0.100	River
3	Bet	11	0.100	River
5	All-In	11	0.099	River
6	All-In	11	0.099	River

Pay_Out_Time	Subround_ID	Game_Account_ID	Pay_Out_Amount
2024-11-20 15:35:41.641524	4	3	1.220
2024-11-20 15:35:41.641524	4	4	1.220
2024-11-20 16:05:41.239434	7	3	0.685
2024-11-20 16:25:22.686326	11	3	1.565
2024-11-20 16:25:22.686326	11	5	-0.001
2024-11-20 16:25:22.686326	11	6	-0.001
NULL	NULL	NULL	NULL

As we can see, the game accounts 5 and 6 get closed out due to having lost the game as player 3 wins. However, we see the holdout transaction occur with the bet in order to keep the account active during the round, and the pay-out of that amount is taken back from the player after they have gone all in and lost, closing their account.

Secondary Features

As mentioned earlier, the secondary features are outside the scope of the other features in that they do not play a role in manipulating the database, but more so maintain the integrity of the database. Most of these features are highly similar in function and form and thus their details will be omitted.

The first set of these are the deletion constraint triggers meant to prevent the deletion of open records. As mentioned in our operational rules, it is necessary to ensure that if a deletion were to occur, which the user doesn't have the ability to do unless maliciously intentional, that the records associated with the deletion are not involved in an active game. Allowing this would yield highly problematic outcomes for the actual game and thus were constrained against.

Transactional Deletion Trigger Constraints on Open Records

- Make_Bet_Deletion_Trigger
- Community_Cards_Deletion_Trigger
- Players_Cards_Deletion_Trigger
- Buy_In_Deletion_Trigger
- Buy_Out_Deletion_Trigger

Example Trigger (Make_Bet_Deletion_Trigger):

```
DELIMITER //
CREATE TRIGGER Make_Bet_Deletion_Trigger
BEFORE DELETE ON Make_Bet
FOR EACH ROW
    BEGIN
        DECLARE p_Round_ID INT;

        SELECT Round_ID INTO p_Round_ID FROM Subround WHERE Subround_ID = OLD.Subround_ID;
        IF EXISTS(SELECT * FROM Round R WHERE R.Round_ID = p_Round_ID AND R.End_Time IS NULL) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Deletion Transaction Constraint Violated for Make
Bet, Round still Active';
        END IF;

    END//
```

The transactional deletion triggers consist of the same essential components, they are triggered when the corresponding tables record is attempting to be deleted, the transactor and/or transactee entities are checked to see whether they have closed, if not, the deletion is rolled back.

Game Hierarchy Deletion Trigger Constraints

- Game_Deletion_Trigger
- Round_Deletion_Trigger
- Subround_Deletion_Trigger

Example Trigger (Game_Deletion_Trigger):

```
DELIMITER //
CREATE TRIGGER Game_Deletion_Trigger
BEFORE DELETE ON Game
FOR EACH ROW
BEGIN

    IF OLD.End_Time IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Game Deletion Hierarchy Constraint Violated for Game,
Game is still active';

    ELSEIF EXISTS (SELECT * FROM Round R WHERE R.Game_ID = OLD.Game_ID AND R.End_Time IS NULL)
           OR EXISTS (SELECT * FROM Subround S JOIN Round R ON S.Round_ID = R.Round_ID
                      WHERE R.Game_ID = OLD.Game_ID AND S.End_Time IS NULL) THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Game Deletion Hierarchy Constraint Violated for
Game, Round/Subround(s) still active';
    END IF;
END//
```

These deletion triggers control the hierarchy of deletion order ensuring that no parent can be deleted before a child has closed. In this example, we see the Game_Deletion_Trigger set up such that only if all the rounds, and all the subrounds of all those rounds, have closed, can the game record be deleted from the database.

Other Features

PlayerPokerRecords

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `PlayerPokerRecords`(
    IN p_Player_ID INT,
    IN p_Game_Account_ID INT,
    IN p_record_type VARCHAR(50)
)
BEGIN
    CASE p_record_type
        WHEN 'Player Records' THEN
            SELECT * FROM poker_room.Player WHERE Player_ID = p_Player_ID;

        WHEN 'Game Account Records' THEN
            SELECT GA.* FROM Game_Account GA
            WHERE GA.Game_Account_ID IN (SELECT BI.Game_Account_ID FROM Buy_In BI WHERE BI.Player_ID =
p_Player_ID);

        WHEN 'Game Records' THEN
            SELECT G.Game_ID, AST.Max_Seats AS Max_Seat_Type, ALT.Limit_Type AS Limit_Type, G.Start_Time,
G.End_Time
            FROM Game G JOIN All_Seat_Types AST ON G.Max_Seats_ID = AST.Max_Seats_ID
            JOIN All_Limit_Types ALT ON G.Limit_Type_ID = ALT.Limit_Type_ID
            WHERE G.Game_ID = (SELECT GA.Game_ID FROM Game_Account GA WHERE GA.Game_Account_ID =
p_Game_Account_ID);

        WHEN 'All Bet Records' THEN
            SELECT MB.Bet_Type_ID, MB.Subround_ID, S.Round_ID, MB.Bet_Amount, AST.Subround_Name
            FROM Make_Bet MB
            JOIN Subround S ON MB.Subround_ID = S.Subround_ID
            JOIN All_Subround_Types AST ON S.Subround_Type_ID = AST.Subround_Type_ID
            WHERE MB.Game_Account_ID = p_Game_Account_ID;

        WHEN 'All Players Cards Records' THEN
            SELECT PC.Round_ID, PC.Card_Type_ID AS Player_Card, PC.Game_Account_ID FROM Players_Cards PC
            WHERE PC.Game_Account_ID = p_Game_Account_ID;

        WHEN 'All Community Cards Records' THEN
            SELECT S.Round_ID, CC.Subround_ID, CC.Card_Type_ID AS Community_Card
            FROM Community_Cards CC JOIN Subround S ON CC.Subround_ID = S.Subround_ID
            WHERE S.Round_ID IN (
                SELECT DISTINCT S.Round_ID
                FROM Subround S
                JOIN Make_Bet MB ON S.Subround_ID = MB.Subround_ID
                WHERE MB.Game_Account_ID = p_Game_Account_ID);
        WHEN 'Help' THEN
```

The last feature that fits neither of the groups mentioned earlier is simply the PlayerPokerRecords procedure, that we described briefly in the Security section. This procedure acts essentially as a view for specific records within the database, preventing the user from seeing not only certain tables, but from them seeing other records of other players. This is essential in preventing things like a player being able to see another player's cards, or letting them figure out someone else's primary key and logging into their game account.

4.3 Queries

For our queries we will describe a mixture of custom queries that could be used by admin when trying to obtain specific records, as well as queries performed by some of the procedures to query certain data for poker game operations.

4.3.1 All Bet Records

```
SELECT MB.Bet_Type_ID, MB.Subround_ID, S.Round_ID, MB.Bet_Amount, AST.Subround_Name
            FROM Make_Bet MB
            JOIN Subround S ON MB.Subround_ID = S.Subround_ID
            JOIN All_Subround_Types AST ON S.Subround_Type_ID = AST.Subround_Type_ID
            WHERE MB.Game_Account_ID = p_Game_Account_ID;
```

This query is used by the PlayerPokerRecords procedure to obtain all the bet records associated with all subrounds, rounds that the player played in. Given the records in the Make_Bet table, all the subrounds within the table that match the subround id of the record that contains the players game account id, are joined, and the respective columns are selected for the player to view.

We can see an example of this as below for a player whose game account id is 1,

Bet_Type_ID	Subround_ID	Round_ID	Bet_Amount	Subround_Name
2	2	1	0.100	Flop
1	6	2	0.000	Flop
2	10	3	0.100	Flop
2	14	4	0.100	Flop
2	18	5	0.100	Flop
2	22	6	0.100	Flop
2	26	7	0.100	Flop
2	30	8	0.100	Flop
2	34	9	0.100	Flop
2	38	10	0.100	Flop
2	42	11	0.100	Flop
2	1	1	0.100	Pre-Flop
2	5	2	0.100	Pre-Flop
2	9	3	0.100	Pre-Flop
2	13	4	0.100	Pre-Flop
2	17	5	0.100	Pre-Flop
2	21	6	0.100	Pre-Flop
2	25	7	0.100	Pre-Flop

✓	231	17:11:58	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
✓	232	17:11:58	CALL SafeMakeBet(1, 1)	1 row(s) affected	0.031 sec
✓	233	17:11:58	CALL SafeMakeBet(2, 1)	1 row(s) affected	0.016 sec
✓	234	17:11:58	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
✓	235	17:11:58	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.016 sec
✓	236	17:11:58	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.015 sec
✓	237	17:11:58	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
✓	238	17:11:58	CALL SafeMakeBet(1, 0)	1 row(s) affected	0.016 sec
✓	239	17:11:58	CALL SafeMakeBet(2, 1)	1 row(s) affected	0.016 sec
✓	240	17:11:58	CALL SafeMakeBet(3, 1)	1 row(s) affected	0.031 sec
✓	241	17:11:58	CALL SafeMakeBet(4, 1)	1 row(s) affected	0.016 sec
✓	242	17:11:58	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.031 sec
✓	243	17:11:58	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
✗	244	17:11:58	CALL SafeMakeBet(1, 1)	Error Code: 1644. Game_Account is Associated with a Previous Fold/All-in in the Same R...	0.000 sec
✓	245	17:13:06	CALL PlayerPokerRecords(Null, 1, 'All Bet Records')	41 row(s) returned	0.016 sec / 0.000 sec

*Note that the failed output at SafeMakeBet(1,1) was because I mistakenly bet for the same player after they folded.

As such, the player can obtain a view of all their bets but none of the bets of the other players.

4.3.2 All-In, Folds, and Bet Count

Suppose I as an admin wanted to collect some information about how many players have folded, went all in, and made bets during a specific game. The query I would perform to obtain this would be as shown below.

This would involve using the query below,

```
SELECT
    BT.Bet_Type AS BetType,
    COUNT(MB.Bet_Type_ID) AS Count
FROM Make_Bet MB
JOIN Game_Account GA ON MB.Game_Account_ID = GA.Game_Account_ID
JOIN All_Bet_Types BT ON MB.Bet_Type_ID = BT.Bet_Type_ID
WHERE GA.Game_ID = 1
GROUP BY BT.Bet_Type;
```

As an example, supposing an admin wanted to perform this query on game id 1, the output would look something like this,

	BetType	Count
▶	Bet	253
	Fold	3

✓	242	17:11:58	CALL SafeMakeBet(5, 1)	1 row(s) affected	0.031 sec
✓	243	17:11:58	CALL SafeMakeBet(6, 1)	1 row(s) affected	0.031 sec
✗	244	17:11:58	CALL SafeMakeBet(1, 1)	Error Code: 1644. Game_Account is Associated with a Previous Fold/All-in in the Same Round	0.000 sec
✓	245	17:13:06	CALL PlayerPokerRecords(Null, 1, 'All Bet Records')	41 row(s) returned	0.016 sec / 0.000 sec
✓	246	17:40:18	CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records')	11 row(s) returned	0.047 sec / 0.000 sec
✓	247	17:40:24	CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records')	11 row(s) returned	0.000 sec / 0.000 sec
✓	248	17:40:24	CALL PlayerPokerRecords(Null, 1, 'All Community Cards Records')	55 row(s) returned	0.016 sec / 0.000 sec
✓	249	17:47:25	SELECT BT.Bet_Type AS BetType, COUNT(MB.Bet_Type_ID) AS Count FROM Make_Bet MB JOIN Game...	2 row(s) returned	0.031 sec / 0.000 sec

4.3.3 All Players Cards and Community Cards

In a similar vein if a player was playing the game they would need to know what cards they were holding onto as well as what the community cards were in order to make a bet. However, it would be unwise to let them see all the cards of other players within the same game. Thus, we can query both card types based on the players game account id as seen below, restricting their view to only the cards and community cards that they were dealt.

```
SELECT PC.Round_ID, PC.Card_Type_ID AS Player_Card, PC.Game_Account_ID FROM Players_Cards PC
            WHERE PC.Game_Account_ID = p_Game_Account_ID;
```

```
SELECT S.Round_ID, CC.Subround_ID, CC.Card_Type_ID AS Community_Card
            FROM Community_Cards CC JOIN Subround S ON CC.Subround_ID = S.Subround_ID
            WHERE S.Round_ID IN (
                SELECT DISTINCT S.Round_ID
                FROM Subround S
                JOIN Make_Bet MB ON S.Subround_ID = MB.Subround_ID
                WHERE MB.Game_Account_ID = p_Game_Account_ID);
```

We can see an example of this being performed for game account id 1, through the PlayerPokerRecords procedure.

Round_ID	Player_Card	Game_Account_ID
1	5	1
2	1	1
3	6	1
4	1	1
5	5	1
6	6	1
7	5	1
8	2	1
9	8	1
10	3	1
11	1	1

Round_ID	Subround_ID	Community_Card
1	2	1
1	2	1
1	2	9
1	3	10
1	4	7
2	6	9
2	6	1
2	6	2
2	7	8
2	8	6
3	10	9
3	10	9
3	10	1
3	11	1
3	12	1
4	14	4
4	14	10
4	14	3
4	15	3
4	16	10

245 17:13:06 CALL PlayerPokerRecords(Null, 1, 'All Bet Records') 41 row(s) returned 0.016 sec / 0.000 sec
246 17:40:18 CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records') 11 row(s) returned 0.047 sec / 0.000 sec
247 17:40:24 CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records') 11 row(s) returned 0.000 sec / 0.000 sec
248 17:40:24 CALL PlayerPokerRecords(Null, 1, 'All Community Cards Records') 55 row(s) returned 0.016 sec / 0.000 sec

4.3.4 Obtain Bet, Type, Subround Name

A similar query was performed earlier in showing the functionalities of the overall poker game workflow. This query is different from the All Bet Types query that is specific to a player, however, this query can be used by admin wanting to obtain all the bet types placed, all amounts bet, and the subround names associated with them, for all game accounts for a specific round.

```sql
-- Show Complex Query of the round, its Subround Types, the bet types, the amounts in each bet
SELECT
    MB.Game_Account_ID,
    BT.Bet_Type,
    MB.Subround_ID,
    MB.Bet_Amount,
    AST.Subround_Name
FROM Make_Bet MB
JOIN All_Bet_Types BT ON MB.Bet_Type_ID = BT.Bet_Type_ID
JOIN Subround S ON MB.Subround_ID = S.Subround_ID
JOIN All_Subround_Types AST ON S.Subround_Type_ID = AST.Subround_Type_ID
WHERE MB.Subround_ID IN (
    SELECT Subround_ID
    FROM Subround
    WHERE Round_ID = 1
)
ORDER BY MB.Subround_ID ASC;
```

Game_Account_ID	Bet_Type	Subround_ID	Bet_Amount	Subround_Name
1	Bet	5	0.100	Pre-Flop
2	Bet	5	0.100	Pre-Flop
3	Bet	5	0.100	Pre-Flop
4	Bet	5	0.100	Pre-Flop
5	Bet	5	0.100	Pre-Flop
6	Bet	5	0.100	Pre-Flop
2	Bet	6	0.100	Flop
3	Bet	6	0.100	Flop
4	Bet	6	0.100	Flop
5	Bet	6	0.100	Flop
6	Bet	6	0.100	Flop
1	Fold	6	0.000	Flop
2	Bet	7	0.100	Turn
3	Bet	7	0.100	Turn
4	Bet	7	0.100	Turn
5	Bet	7	0.100	Turn
6	Bet	7	0.100	Turn

242	17:11:58	CALL SafeMakeBet(5, 1)	1 row(s) affected		0.031 sec
243	17:11:58	CALL SafeMakeBet(6, 1)	1 row(s) affected		0.031 sec
244	17:11:58	CALL SafeMakeBet(1, 1)	Error Code: 1644. Game_Account is Associated with a Previous Fold/All-in in the Same Round		0.000 sec
245	17:13:06	CALL PlayerPokerRecords(Null, 1, 'All Bet Records')	41 row(s) returned		0.016 sec / 0.000 sec
246	17:40:18	CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records')	11 row(s) returned		0.047 sec / 0.000 sec
247	17:40:24	CALL PlayerPokerRecords(Null, 1, 'All Players Cards Records')	11 row(s) returned		0.000 sec / 0.000 sec
248	17:40:24	CALL PlayerPokerRecords(Null, 1, 'All Community Cards Records')	55 row(s) returned		0.016 sec / 0.000 sec
249	17:47:25	SELECT BT.Bet_Type AS BetType, COUNT(MB.Bet_Type_ID) AS Count FROM Make_Bet MB JOIN Game...	2 row(s) returned		0.031 sec / 0.000 sec
250	17:53:10	SELECT MB.Game_Account_ID, BT.Bet_Type, MB.Subround_ID, MB.Bet_Amount, AST.Subroun...	22 row(s) returned		0.031 sec / 0.000 sec
251	17:53:15	SELECT MB.Game_Account_ID, BT.Bet_Type, MB.Subround_ID, MB.Bet_Amount, AST.Subroun...	22 row(s) returned		0.015 sec / 0.000 sec
252	17:53:42	SELECT MB.Game_Account_ID, BT.Bet_Type, MB.Subround_ID, MB.Bet_Amount, AST.Subroun...	22 row(s) returned		0.016 sec / 0.000 sec

4.3.5 Rank Players

Suppose an admin wants to collect information about who the best player is for a particular game. To do this, given a game id, all the game account ids can be extracted, and their respective pay out records can be counted for each player. In the case that the payout is -.001, this count will not be counted as this is a loss. From here, the players can be ranked by their counts. This query can be seen below.

```
USE poker_room;

SELECT
    GA.Game_Account_ID,
    COUNT(PO.Pay_Out_Amount) AS Win_Count
FROM Game_Account GA
LEFT JOIN Pay_Out PO ON GA.Game_Account_ID = PO.Game_Account_ID
WHERE GA.Game_ID = 1
  AND PO.Pay_Out_Amount != -0.001
GROUP BY GA.Game_Account_ID
ORDER BY Win_Count DESC;
```

Game_Account_ID	Win_Count
3	3
2	2
4	2
5	2
6	2
1	1

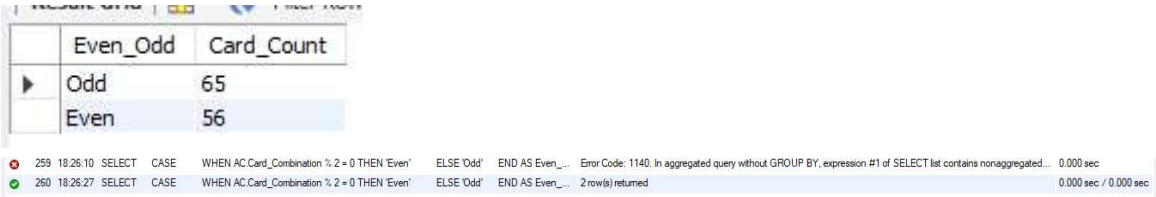
254 18:10:08 SELECT GA.Game_Account_ID, COUNT(PO.Pay_Out_Amount) AS Win_Count FROM Game_Account GA ... 6 row(s) returned 0.047 sec / 0.000 sec

4.3.6 Even/Odd Community Cards

Suppose that an admin receives a complaint from a player that they have been seeing an unusually high number of even/odd cards. The admin would need to then perform a query to obtain all the even and odd cards within the community/players cards table and obtain the counts of even/odd valued cards. This can be performed by the query seen below:

```
SELECT
    CASE
        WHEN AC.Card_Combination % 2 = 0 THEN 'Even'
        ELSE 'Odd'
    END AS Even_Odd,
    COUNT(*) AS Card_Count
FROM (
    SELECT Card_Type_ID FROM Community_Cards
    UNION ALL
    SELECT Card_Type_ID FROM Players_Cards
) AS All_Cards
JOIN All_Card_Types AC ON All_Cards.Card_Type_ID = AC.Card_Type_ID
GROUP BY Even_Odd;
```

In this, we perform a union all on all the Card_Type_ID's from both the community cards and players cards table, to consider the counts from both tables. From here, we join these values on the card combination found within the All_Card_Types table, and then group by the Even_Odd card combination value.



	Even_Odd	Card_Count
▶	Odd	65
	Even	56

✖	259	18:26:10	SELECT	CASE	WHEN AC.Card_Combination % 2 = 0 THEN 'Even'	ELSE 'Odd'	END AS Even_...	Error Code: 1140. In aggregated query without GROUP BY, expression #1 of SELECT list contains nonaggregated...	0.000 sec
✔	260	18:26:27	SELECT	CASE	WHEN AC.Card_Combination % 2 = 0 THEN 'Even'	ELSE 'Odd'	END AS Even_...	2 row(s) returned	0.000 sec / 0.000 sec

4.3.7 Aggressive Players

Suppose an admin wants to find the players who didn't make a single fold in any of rounds played by their game account ids. The way to do this would be to select all the distinct game account ids from make bet who has a fold, and then select those game account ids from all the game account ids in all the games. The query would look as seen below.

```
SELECT DISTINCT GA.Game_Account_ID
FROM Game_Account GA
WHERE GA.Game_Account_ID NOT IN (
    SELECT DISTINCT MB.Game_Account_ID
    FROM Make_Bet MB
    JOIN All_Bet_Types BT ON MB.Bet_Type_ID = BT.Bet_Type_ID
    WHERE BT.Bet_Type = 'Fold'
);
```

	Game_Account_ID
▶	10
	2
	6
	8
	11
	12
*	NULL

```
499  18:39:15  CALL SafeMakeBet(5, 0)                                          1 row(s) affected    0.016 sec
500  18:39:29  SELECT DISTINCT GA.Game_Account_ID FROM Game_Account GA WHERE GA.Game_Account_ID NOT IN ( ...  6 row(s) returned  0.000 sec / 0.000 sec
```

4.3.8 Obtain Pot Size Ranking

This query is more complex than it seems as the pot within the Subround is simply a placeholder for the pot for that given subround, but when the pay-out occurs the pot is cleared and so all subrounds that have ended, have their last pot size as zero. Thus to obtain the largest pot size ever seen we would need to obtain all the make bet records for each round, and then for all the bets that were not all-in or fold in bet type, subtract 2.5% of the bet, sum them together, and then rank them based on the size. This query can be seen below,

```
SELECT
    r.Round_ID,
    SUM(
        CASE
            WHEN abt.Bet_Type = 'All-In' THEN mb.Bet_Amount * .999
            WHEN abt.Bet_Type = 'Bet' THEN mb.Bet_Amount * 0.975
            ELSE 0
        END
    ) AS Total_Pot_Size
FROM Make_Bet mb
JOIN Subround sr ON mb.Subround_ID = sr.Subround_ID
JOIN Round r ON sr.Round_ID = r.Round_ID
JOIN All_Bet_Types abt ON mb.Bet_Type_ID = abt.Bet_Type_ID
GROUP BY r.Round_ID
ORDER BY Total_Pot_Size DESC;
```

Here, if the bet type is all-in we subtract .001 from the bet made due to the placeholder of .001 in their account, and if the bet type is of type bet, we strip the rake from the bet, then we join the records on their associated subrounds round, and group by the round id.

	Round_ID	Total_Pot_Size
▶	16	2.341401
	1	2.340000
	4	2.340000
	5	2.340000
	6	2.340000
	7	2.340000
	8	2.340000
	9	2.340000
	10	2.340000
	12	2.340000
	15	2.340000
	17	2.340000
	18	2.340000
	19	2.313429
	3	2.242500
	2	2.047500

5. CRUD Matrix

Now we will give a brief overview of our entities, functions/procedures/triggers, and show our CRUD matrix. We will only include the primary features that directly manipulate or aid in, manipulating the database.

5.1 List of Entity Types

E1: All_Card_Types
E2: All_Subround_Types
E3: All_Seat_Types
E4: All_Bet_Types
E5: All_Limit_Types
E6: System_Register
E7: Player
E8: Game
E9: Round
E10: Subround
E11: Game_Account
E12: Make_Bet
E13: Pay_Out
E14: Rake_Out
E15: Buy_Out
E16: Buy_In
E17: Players_Cards
E18: Community_Cards

5.2 List of Functions

Procedures

CloseRound (F1)

- Reads: Round, Game
- Updates: Subround, Round
- Inserts: Round

CloseSubround (F2)

- Reads: Round, Game, All_Subround_Types
- Updates: Subround
- Inserts: Subround

PlayerPokerRecords (F3)

- Reads: Player, Game_Account, Buy_In, Game, All_Seat_Types, All_Limit_Types, Make_Bet, Subround, All_Subround_Types, Players_Cards, Community_Cards

DetermineWinner (F4)

- Reads: Subround, Make_Bet, All_Bet_Types, Players_Cards, All_Card_Types, TempWinners, TempBaseData, TempAdjustedTotals, TempLosers
- Inserts: Pay_Out

GeneratePlayerCards (F5)

- Reads:Game_Account, Round, Players_Cards

InsertCards (F6)

- Reads: All_Card_Types
- Inserts: Community_Cards, Players_Cards

PlayerLossGameClosure (F7)

- Reads: Game_Account, Round, Subround, Game
- Updates: Game_Account, Subround, Round, Game
- Inserts: Pay_Out

RoundStateHandler (F8)

- Reads: All_Subround_Types, Subround, System_Register
- Inserts: Pay_Out, Rake_Out

SafeBuyIn (F9)

- Reads: Buy_In, Game_Account, Game, All_Seat_Types, All_Limit_Types
- Inserts: Game, Game_Account, Buy_In

SafeBuyOut (F10)

- Reads: Buy_In
- Inserts: Buy_Out

SafeMakeBet (F11)

- Reads: Game_Account, Round, Subround, Make_Bet, All_Bet_Types, All_Limit_Types, System_Register
- Inserts: Make_Bet, Rake_Out

UpdateAccountBalance (F12)

- Reads:
  Player, Game_Account, Subround, System_Register
- Updates:
  Player, Game_Account, Subround, System_Register

VerifyOpen (F13)

- Reads: Game_Account, Player, Subround

VerifyTransactor (F14)

- Reads: Player, Game_Account, Subround

Functions

ComputeGameStats (F15)

- Reads: Game_Account, Make_Bet, Subround, Round, All_Bet_Types

SumCommCards (F16)

- Reads: Subround, Community_Cards, All_Card_Types

VerifyMakeBet (F17)

- Reads: Game_Account, Round

Triggers

Start_Game_Trigger (F18)

- Reads: Game_Account, Game, Round
- Inserts: Round

Start_Round_Trigger (F19)

- Reads: All_Subround_Types
- Inserts: Subround

Close_Game_Account_Trigger (F20)

- Reads: Subround, Game_Account
- Updates: Game_Account

Community_Cards_Trigger (F21)

- Reads: All_Subround_Types

Make_Bet_Trigger (F22)

- Reads: Make_Bet, Subround, Round, Game_Account

CRUD Matrix

F/E	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17
F1	R	-	-	-	-	-	-	R	R	U	-	-	-	-	-	-	-
F2	-	R	-	-	-	-	-	R	R	U	-	-	-	-	-	-	-
F3	-	-	R	-	R	-	R	R	-	-	R	-	-	-	-	-	R
F4	R	-	-	R	-	-	-	-	-	R	-	R	C	-	-	-	R
F5	-	-	-	-	-	-	-	-	R	-	R	-	-	-	-	-	-
F6	R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C
F7	-	-	-	-	-	-	-	U	U	U	U	-	C	-	-	-	-
F8	-	R	-	-	-	R	-	-	-	R	-	-	C	C	-	-	-
F9	-	-	R	-	R	-	-	C	-	-	C	-	-	-	-	C	-
F10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	-	-
F11	-	-	-	R	R	-	-	-	R	R	R	C	-	C	-	-	-
F12	-	-	-	-	-	R	U	-	-	-	U	-	-	-	-	-	-
F13	-	R	-	-	-	-	R	-	-	R	R	-	-	-	-	-	-
F14	-	-	-	-	-	-	R	-	-	R	R	-	-	-	-	-	-
F15	-	-	-	R	-	-	-	-	-	-	R	-	-	-	-	-	-
F16	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-	-	R
F17	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-	-
F18	-	-	-	-	-	-	-	R	C	-	-	-	-	-	-	-	-
F19	-	C	-	-	-	-	-	-	-	C	-	-	-	-	-	-	-
F20	-	-	-	-	-	-	-	-	-	-	U	-	-	-	-	-	-
F21	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

6. Concluding Remarks

As the first database project I ever have done, this project was rather ambitious on my part as the complexity of designing a database that not only would store the data from the poker games and query relationships between them, constructing the complete logical design of the game was extremely challenging. This along with my utter unfamiliarity of SQL and its highly sensitive nature when it comes to the things you can and cannot do with functions, procedures, and triggers, made me rethink the design entirely. I come from a bioinformatics background and have recently shifted more into computer science and machine learning, but my familiarity in programming stems primarily from python, java, and C++. The

differences between these languages and SQL particularly through the MySQL Workbench was difficult to wrap my head around.

Furthermore, after finally designing the completed workflow, debugging the database also became a very challenging task. The lack of being able to use select statements within any procedure called by a trigger, and the utter lack of verbosity in the error statements of MySQL made debugging a grueling endeavor. Nonetheless, having never really used SQL before, this project really encompassed every basic SQL operation there is. Furthermore, constructing ERD's, figuring out how to normalize tables, understanding how to group the different entities together to make them queryable, were all things I learned from this project that I can now implement in any other future database project I might encounter.

With regards to the major strengths of my project, I think I am most proud of performing the logical design of the game, and designing a well grouped, partitioned entities that allowed all the records within the database to be generated by only the three control procedures that the users have access to. While constructing the logical design for the game play it became clear that query languages were not meant to perform a lot of the functions I was trying to make them do. When I first envisioned my design I thought about it in a general programming language paradigm, and the fact that I tried to implement these gameplay aspects of the game lends ears to my misconceptions of what the function of a database is. In hindsight if I were to do this again I would have used a general purpose language to perform all the logical functions of the game, and then remotely sent the actual changes needed to be made to the database thereafter. Nonetheless, being able to do so in SQL itself is something I am quite happy about.

With regards to the weaknesses of my project, there are several. The first is that it may have been better to use a larger corpus of attributes for the tables, to record more nuanced characteristics of the game. Originally, a larger quantity was used however, we were unable to normalize some of these attributes properly, and thus omitted them from the design. Things like recording the winners, losers,game play statistics of a player, were unable to be used as attributes which if we wanted to obtain would be difficult to query. The difficulty to query is also another aspect of the design that is weak, and the queries used were often convoluted and would likely be very inefficient in a real world setting. Handling all the edge cases of the poker game made the complexity of the design grow exponentially, and the call stack of procedures and functions from a single input from the user, makes the design difficult to understand. Furthermore, in retrospect I would have created more linkages between the entities to make it easier to query, and this would have helped the design immensely. Overall I can safely say I am extremely proud of the results, and being able to create a functional design especially as complicated as it was to make for me.

Appendices

Appendix A - DDL, INSERT, SELECT Statements

```
USE poker_room;

CREATE TABLE All_Card_Types (
    Card_Type_ID INT PRIMARY KEY AUTO_INCREMENT,
    Card_Combination INT NOT NULL,
    CONSTRAINT Card_Name_Unq
        UNIQUE (Card_Combination)
);
CREATE TABLE All_Subround_Types (
    Subround_Type_ID INT PRIMARY KEY AUTO_INCREMENT,
    Subround_Name VARCHAR(20)   NOT NULL,
    CONSTRAINT Subround_Name_unq
        UNIQUE (Subround_Name)
);

CREATE TABLE All_Seat_Types (
    Max_Seats_ID INT PRIMARY KEY AUTO_INCREMENT,
    Max_Seats INT NOT NULL,
    CONSTRAINT Max_Seats_non_negative
        CHECK (Max_Seats >= 2),
    CONSTRAINT Max_Seats_unq
        UNIQUE (Max_Seats)
);


CREATE TABLE All_Bet_Types (
    Bet_Type_ID INT PRIMARY KEY AUTO_INCREMENT,
    Bet_Type VARCHAR(20) NOT NULL,
    CONSTRAINT Bet_Type_unq
        UNIQUE (Bet_Type)
);

CREATE TABLE All_Limit_Types (
    Limit_Type_ID INT PRIMARY KEY AUTO_INCREMENT,
    Limit_Type DECIMAL(7, 3) NOT NULL,
    CONSTRAINT Limit_Type_unq
        UNIQUE (Limit_Type),
    CONSTRAINT Limit_Type_min
        CHECK (Limit_Type >= .1)
);
```

```sql
INSERT INTO All_Card_Types (Card_Combination) VALUES
    (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

INSERT INTO All_Subround_Types (Subround_Name) VALUES
    ('Pre-Flop'), ('Flop'), ('Turn'), ('River');

INSERT INTO All_Seat_Types (Max_Seats) VALUES
    (6), (9), (12);

INSERT INTO All_Bet_Types (Bet_Type) VALUES
    ('Fold'), ('Bet'), ('All-In');

INSERT INTO All_Limit_Types (Limit_Type) VALUES
    (.1),(.5),(1),(2),(5);


CREATE TABLE System_Register (
    Register_ID INT PRIMARY KEY AUTO_INCREMENT,
    Account_Balance DECIMAL(7, 3) NOT NULL
);

CREATE TABLE Player (
    Player_ID INT PRIMARY KEY AUTO_INCREMENT,
    Player_Name VARCHAR(50) NOT NULL,
    Address VARCHAR(50),
    Age INT     NOT NULL,
    Account_Creation_Date TIMESTAMP(6) NOT NULL DEFAULT CURRENT_TIMESTAMP(6),
    Credit_Card_Number BIGINT,
    Account_Balance DECIMAL(7, 3) NOT NULL DEFAULT 0.00,
    CONSTRAINT Adult_Constraint
        CHECK (Age >= 21),
    CONSTRAINT playername_unq
        UNIQUE(Player_Name)
);
```

```sql
INSERT INTO System_Register (Account_Balance)
VALUES
    (100.5);

INSERT INTO Player (Player_Name, Address, Age, Account_Creation_Date, Credit_Card_Number, Account_Balance)
VALUES
    ('BillyBob', '456 Oak St', 45, CURRENT_TIMESTAMP(6), 4532678956789123, 100),
    ('BobSmith', '456 Oak St', 45, CURRENT_TIMESTAMP(6), 4532678956789123, 2200.50),
    ('CathyBrown', '789 Pine St', 29, CURRENT_TIMESTAMP(6), 4532678945671234, 890.30),
    ('DavidWilliams', '101 Elm St', 32, CURRENT_TIMESTAMP(6), 4532678998765432, 5400.00),
    ('EmilyDavis', '202 Birch St', 27, CURRENT_TIMESTAMP(6), 4532678976543210, 120.00),
    ('FrankThompson', '303 Cedar St', 38, CURRENT_TIMESTAMP(6), 4532678965432109, 300.25),
    ('GraceLee', '404 Spruce St', 35, CURRENT_TIMESTAMP(6), 4532678932123456, 7200.80),
    ('HenryClark', '505 Ash St', 50, CURRENT_TIMESTAMP(6), 4532678912340987, 100.00),
    ('IsabellaMartinez', '606 Fir St', 26, CURRENT_TIMESTAMP(6), 4532678923456781, 180.60),
    ('JamesAnderson', '707 Redwood St', 42, CURRENT_TIMESTAMP(6), 4532678998761234, 2500.40),
    ('KarenWilson', '808 Hemlock St', 31, CURRENT_TIMESTAMP(6), 4532678912345678, 130.00),
    ('LeoScott', '909 Cypress St', 46, CURRENT_TIMESTAMP(6), 4532678954321987, 1600.90),
    ('MariaKing', '1010 Willow St', 28, CURRENT_TIMESTAMP(6), 4532678945678912, 540.50),
    ('NathanWright', '1111 Alder St', 54, CURRENT_TIMESTAMP(6), 4532678987654321, 470.80),
    ('OliviaWhite', '1212 Poplar St', 33, CURRENT_TIMESTAMP(6), 4532678932109876, 8900.20),
    ('PaulGreen', '1313 Magnolia St', 40, CURRENT_TIMESTAMP(6), 4532678923450981, 123.45);
```

```sql
CREATE TABLE Game (
    Game_ID INT PRIMARY KEY AUTO_INCREMENT,
    Limit_Type_ID INT NOT NULL,
    Max_Seats_ID INT NOT NULL,
    Start_Time TIMESTAMP(6) NOT NULL,
    End_Time TIMESTAMP(6),
    CONSTRAINT G_Limit_Type_ID_FK
        FOREIGN KEY (Limit_Type_ID) REFERENCES All_Limit_Types (Limit_Type_ID)
        ON DELETE RESTRICT ON UPDATE RESTRICT,
    CONSTRAINT G_Max_Seats_ID_FK
        FOREIGN KEY (Max_Seats_ID) REFERENCES All_Seat_Types (Max_Seats_ID)
        ON DELETE RESTRICT ON UPDATE RESTRICT
);

CREATE TABLE Round (
    Round_ID INT PRIMARY KEY AUTO_INCREMENT,
    Game_ID INT NOT NULL,
    Start_Time TIMESTAMP(6) NOT NULL,
    End_Time TIMESTAMP(6),
    CONSTRAINT R_Game_ID_FK
        FOREIGN KEY (Game_ID) REFERENCES Game (Game_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

```sql
CREATE TABLE Subround (
    Subround_ID INT PRIMARY KEY AUTO_INCREMENT,
    Round_ID INT NOT NULL,
    Subround_Type_ID INT NOT NULL,
    Start_Time TIMESTAMP(6) NOT NULL,
    End_Time TIMESTAMP(6),
    Pot DECIMAL(7, 3) NOT NULL,
    CONSTRAINT SR_Round_ID_FK
        FOREIGN KEY (Round_ID) REFERENCES Round (Round_ID)
        ON DELETE CASCADE ON UPDATE RESTRICT,
    CONSTRAINT SR_Subround_Type_ID_FK
        FOREIGN KEY (Subround_Type_ID) REFERENCES All_Subround_Types (Subround_Type_ID)
        ON DELETE RESTRICT ON UPDATE RESTRICT
);

CREATE TABLE Game_Account (
    Game_Account_ID INT PRIMARY KEY AUTO_INCREMENT,
    Game_ID INT,
    Chip_Stack DECIMAL(7, 3),
    Time_Created TIMESTAMP(6) NOT NULL,
    Time_Closed TIMESTAMP(6),
    CONSTRAINT GA_Game_ID_FK
            FOREIGN KEY (Game_ID) REFERENCES Game (Game_ID)
            ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Make_Bet (
    Bet_Time TIMESTAMP(6),
    Subround_ID INT,
    Game_Account_ID INT,
    Bet_Type_ID INT,
    Bet_Amount DECIMAL(7, 3),
    CONSTRAINT Make_Bet_Composite_PK
        PRIMARY KEY (Bet_Time, Subround_ID, Game_Account_ID),
    CONSTRAINT MB_Sub_Round_ID_FK
        FOREIGN KEY (Subround_ID) REFERENCES Subround (Subround_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT MB_Game_Account_ID_FK
        FOREIGN KEY (Game_Account_ID) REFERENCES Game_Account (Game_Account_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT MB_Bet_Type_ID_FK
        FOREIGN KEY (Bet_Type_ID) REFERENCES All_Bet_Types (Bet_Type_ID)
        ON DELETE CASCADE ON UPDATE RESTRICT,
    CONSTRAINT Bet_Amount_non_negative
        CHECK (Bet_Amount >= 0)
);
```

```sql
CREATE TABLE Pay_Out (
    Pay_Out_Time TIMESTAMP(6),
    Subround_ID INT,
    Game_Account_ID INT,
    Pay_Out_Amount DECIMAL(7, 3) NOT NULL,
    CONSTRAINT Pay_Out_Composite_PK
        PRIMARY KEY (Pay_Out_Time, Subround_ID, Game_Account_ID),
    CONSTRAINT PO_Sub_Round_ID_FK
        FOREIGN KEY (Subround_ID) REFERENCES Subround (Subround_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT PO_Game_Account_ID_FK
        FOREIGN KEY (Game_Account_ID) REFERENCES Game_Account (Game_Account_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Rake_Out (
    Rake_Time TIMESTAMP(6),
    Register_ID INT,
    Subround_ID INT,
    Rake_Amount DECIMAL(7, 3) NOT NULL,
    CONSTRAINT Rake_Out_Composite_PK
        PRIMARY KEY (Rake_Time, Register_ID, Subround_ID),
    CONSTRAINT RO_Register_ID_FK
        FOREIGN KEY (Register_ID) REFERENCES System_Register (Register_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT RO_Sub_Round_ID_FK
        FOREIGN KEY (Subround_ID) REFERENCES Subround (Subround_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
CREATE TABLE Buy_Out (
    Buy_Out_Time TIMESTAMP(6),
    Game_Account_ID INT,
    Player_ID INT,
    Buy_Out_Amount DECIMAL(7, 3) NOT NULL,
    CONSTRAINT Buy_Out_Composite_PK
        PRIMARY KEY (Buy_Out_Time, Game_Account_ID, Player_ID),
    CONSTRAINT BO_Game_Account_ID_FK
        FOREIGN KEY (Game_Account_ID) REFERENCES Game_Account (Game_Account_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT BO_Player_ID_FK
        FOREIGN KEY (Player_ID) REFERENCES Player (Player_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT Buy_Out_Amount_non_negative_or_zero
        CHECK (Buy_Out_Amount > 0)
);
```

```sql
CREATE TABLE Buy_In (
    Buy_In_Time TIMESTAMP(6),
    Game_Account_ID INT,
    Player_ID INT,
    Buy_In_Amount DECIMAL(7, 3) NOT NULL,
    CONSTRAINT Buy_In_Composite_PK
        PRIMARY KEY (Buy_In_Time, Game_Account_ID, Player_ID),
    CONSTRAINT BI_Game_Account_ID_FK
        FOREIGN KEY (Game_Account_ID) REFERENCES Game_Account (Game_Account_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT BI_Player_ID_FK
        FOREIGN KEY (Player_ID) REFERENCES Player (Player_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT Buy_In_Amount_non_negative_or_zero
        CHECK (Buy_In_Amount > 0)
);
CREATE TABLE Players_Cards (
    Player_Card_ID INT PRIMARY KEY AUTO_INCREMENT,
    Round_ID INT,
    Game_Account_ID INT NOT NULL,
    Card_Type_ID INT NOT NULL,
    CONSTRAINT PC_Round_ID_FK
        FOREIGN KEY (Round_ID) REFERENCES Round (Round_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT PC_Game_Account_ID_FK
        FOREIGN KEY (Game_Account_ID) REFERENCES Game_Account (Game_Account_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT PC_Card_Type_ID_FK
        FOREIGN KEY (Card_Type_ID) REFERENCES All_Card_Types (Card_Type_ID)
        ON DELETE CASCADE ON UPDATE RESTRICT
);

CREATE TABLE Community_Cards (
    Community_Card_ID INT PRIMARY KEY AUTO_INCREMENT,
    Card_Type_ID INT NOT NULL,
    Subround_ID INT NOT NULL,
    CONSTRAINT CC_Card_Type_ID_FK
        FOREIGN KEY (Card_Type_ID) REFERENCES All_Card_Types (Card_Type_ID)
        ON DELETE RESTRICT ON UPDATE RESTRICT,
    CONSTRAINT CC_Sub_Round_ID_FK
        FOREIGN KEY (Subround_ID) REFERENCES Subround (Subround_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

✔	1	23:25:22	USE poker_room	0 row(s) affected	0.000 sec
✔	2	23:25:22	CREATE TABLE All_Card_Types (Card_Type_ID INT PRIMARY KEY AUTO_INCREM...	0 row(s) affected	0.031 sec
✔	3	23:25:22	CREATE TABLE All_Subround_Types (Subround_Type_ID INT PRIMARY KEY AUTO...	0 row(s) affected	0.047 sec
✔	4	23:25:22	CREATE TABLE All_Seat_Types (Max_Seats_ID INT PRIMARY KEY AUTO_INCREME...	0 row(s) affected	0.031 sec
✔	5	23:25:22	CREATE TABLE All_Bet_Types (Bet_Type_ID INT PRIMARY KEY AUTO_INCREMENT,...	0 row(s) affected	0.016 sec
✔	6	23:25:22	CREATE TABLE All_Limit_Types (Limit_Type_ID INT PRIMARY KEY AUTO_INCREMEN...	0 row(s) affected	0.016 sec
✔	7	23:25:22	INSERT INTO All_Card_Types (Card_Combination) VALUES (1), (2), (3), (4), (5), (6), (7), (8...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.000 sec
✔	8	23:25:22	INSERT INTO All_Subround_Types (Subround_Name) VALUES ('Pre-Flop'), ('Flop'), ('Turn...	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.016 sec
✔	9	23:25:22	INSERT INTO All_Seat_Types (Max_Seats) VALUES (6), (9), (12)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
✔	10	23:25:22	INSERT INTO All_Bet_Types (Bet_Type) VALUES ('Fold'), ('Bet'), ('All-In')	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
✔	11	23:25:22	INSERT INTO All_Limit_Types (Limit_Type) VALUES (.1),(.5),(1),(2),(5)	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.016 sec
✔	12	23:25:22	CREATE TABLE System_Register (Register_ID INT PRIMARY KEY AUTO_INCREMEN...	0 row(s) affected	0.015 sec
✔	13	23:25:22	CREATE TABLE Player (Player_ID INT PRIMARY KEY AUTO_INCREMENT, Player_...	0 row(s) affected	0.031 sec
✔	14	23:25:22	INSERT INTO System_Register (Account_Balance) VALUES (100.5)	1 row(s) affected	0.016 sec
✔	15	23:25:22	INSERT INTO Player (Player_Name, Address, Age, Account_Creation_Date, Credit_Card_...	16 row(s) affected Records: 16 Duplicates: 0 Warnings: 0	0.016 sec
✔	16	23:25:22	CREATE TABLE Game (Game_ID INT PRIMARY KEY AUTO_INCREMENT, Limit_Ty...	0 row(s) affected	0.031 sec
✔	17	23:25:22	CREATE TABLE Round (Round_ID INT PRIMARY KEY AUTO_INCREMENT, Game_...	0 row(s) affected	0.031 sec
✔	18	23:25:22	CREATE TABLE Subround (Subround_ID INT PRIMARY KEY AUTO_INCREMENT, Ro...	0 row(s) affected	0.031 sec
✔	19	23:25:22	CREATE TABLE Game_Account (Game_Account_ID INT PRIMARY KEY AUTO_INCRE...	0 row(s) affected	0.032 sec
✔	20	23:25:22	CREATE TABLE Make_Bet (Bet_Time TIMESTAMP(6), Subround_ID INT, Game_Ac...	0 row(s) affected	0.031 sec
✔	21	23:25:22	CREATE TABLE Pay_Out (Pay_Out_Time TIMESTAMP(6), Subround_ID INT, Gam...	0 row(s) affected	0.062 sec
✔	22	23:25:23	CREATE TABLE Rake_Out (Rake_Time TIMESTAMP(6), Register_ID INT, Subround...	0 row(s) affected	0.047 sec

	#	Time	Action	Message	Duration / Fetch
✔	5	23:25:22	CREATE TABLE All_Bet_Types (Bet_Type_ID INT PRIMARY KEY AUTO_INCREMENT,...	0 row(s) affected	0.016 sec
✔	6	23:25:22	CREATE TABLE All_Limit_Types (Limit_Type_ID INT PRIMARY KEY AUTO_INCREMEN...	0 row(s) affected	0.016 sec
✔	7	23:25:22	INSERT INTO All_Card_Types (Card_Combination) VALUES (1), (2), (3), (4), (5), (6), (7), (8...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.000 sec
✔	8	23:25:22	INSERT INTO All_Subround_Types (Subround_Name) VALUES ('Pre-Flop'), ('Flop'), ('Turn...	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.016 sec
✔	9	23:25:22	INSERT INTO All_Seat_Types (Max_Seats) VALUES (6), (9), (12)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
✔	10	23:25:22	INSERT INTO All_Bet_Types (Bet_Type) VALUES ('Fold'), ('Bet'), ('All-In')	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
✔	11	23:25:22	INSERT INTO All_Limit_Types (Limit_Type) VALUES (.1),(.5),(1),(2),(5)	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.016 sec
✔	12	23:25:22	CREATE TABLE System_Register (Register_ID INT PRIMARY KEY AUTO_INCREMEN...	0 row(s) affected	0.015 sec
✔	13	23:25:22	CREATE TABLE Player (Player_ID INT PRIMARY KEY AUTO_INCREMENT, Player_...	0 row(s) affected	0.031 sec
✔	14	23:25:22	INSERT INTO System_Register (Account_Balance) VALUES (100.5)	1 row(s) affected	0.016 sec
✔	15	23:25:22	INSERT INTO Player (Player_Name, Address, Age, Account_Creation_Date, Credit_Card_...	16 row(s) affected Records: 16 Duplicates: 0 Warnings: 0	0.016 sec
✔	16	23:25:22	CREATE TABLE Game (Game_ID INT PRIMARY KEY AUTO_INCREMENT, Limit_Ty...	0 row(s) affected	0.031 sec
✔	17	23:25:22	CREATE TABLE Round (Round_ID INT PRIMARY KEY AUTO_INCREMENT, Game_...	0 row(s) affected	0.031 sec
✔	18	23:25:22	CREATE TABLE Subround (Subround_ID INT PRIMARY KEY AUTO_INCREMENT, Ro...	0 row(s) affected	0.031 sec
✔	19	23:25:22	CREATE TABLE Game_Account (Game_Account_ID INT PRIMARY KEY AUTO_INCRE...	0 row(s) affected	0.032 sec
✔	20	23:25:22	CREATE TABLE Make_Bet (Bet_Time TIMESTAMP(6), Subround_ID INT, Game_Ac...	0 row(s) affected	0.031 sec
✔	21	23:25:22	CREATE TABLE Pay_Out (Pay_Out_Time TIMESTAMP(6), Subround_ID INT, Gam...	0 row(s) affected	0.062 sec
✔	22	23:25:23	CREATE TABLE Rake_Out (Rake_Time TIMESTAMP(6), Register_ID INT, Subround...	0 row(s) affected	0.047 sec
✔	23	23:25:23	CREATE TABLE Buy_Out (Buy_Out_Time TIMESTAMP(6), Game_Account_ID INT, Pl...	0 row(s) affected	0.032 sec
✔	24	23:25:23	CREATE TABLE Buy_In (Buy_In_Time TIMESTAMP(6), Game_Account_ID INT, Play...	0 row(s) affected	0.046 sec
✔	25	23:25:23	CREATE TABLE Players_Cards (Player_Card_ID INT PRIMARY KEY AUTO_INCREMEN...	0 row(s) affected	0.016 sec
✔	26	23:25:23	CREATE TABLE Community_Cards (Community_Card_ID INT PRIMARY KEY AUTO_INC...	0 row(s) affected	0.031 sec

	Game_ID	Limit_Type_ID	Max_Seats_ID	Start_Time	End_Time
▶	1	1	1	2024-11-20 16:51:51.803685	NULL
	2	1	1	2024-11-20 18:31:11.483012	NULL
	3	1	3	2024-11-20 23:10:40.917553	NULL
	4	2	2	2024-11-20 23:10:41.065545	NULL
	5	3	1	2024-11-20 23:10:41.076450	NULL
*	NULL	NULL	NULL	NULL	NULL

Round_ID	Game_ID	Start_Time	End_Time
1	1	2024-11-20 16:51:52.120015	2024-11-20 16:52:08.557984
2	1	2024-11-20 16:52:08.557984	2024-11-20 17:10:01.257696
3	1	2024-11-20 17:10:01.257696	2024-11-20 17:11:30.909794
4	1	2024-11-20 17:11:30.909794	2024-11-20 17:11:41.856811
5	1	2024-11-20 17:11:41.856811	2024-11-20 17:11:43.062410
6	1	2024-11-20 17:11:43.062410	2024-11-20 17:11:44.208381
7	1	2024-11-20 17:11:44.208381	2024-11-20 17:11:45.421881
8	1	2024-11-20 17:11:45.421881	2024-11-20 17:11:46.274781
9	1	2024-11-20 17:11:46.274781	2024-11-20 17:11:47.913179
10	1	2024-11-20 17:11:47.913179	2024-11-20 17:11:49.274789
11	1	2024-11-20 17:11:49.274789	NULL
12	2	2024-11-20 18:31:11.527199	2024-11-20 18:32:10.148965
13	2	2024-11-20 18:32:10.148965	2024-11-20 18:32:30.748234
14	2	2024-11-20 18:32:30.748234	2024-11-20 18:33:05.989024
15	2	2024-11-20 18:33:05.989024	2024-11-20 18:35:24.848753
16	2	2024-11-20 18:35:24.848753	2024-11-20 18:35:26.020039
17	2	2024-11-20 18:35:26.020039	2024-11-20 18:35:27.272640
18	2	2024-11-20 18:35:27.272640	2024-11-20 18:35:28.086909

Card_Type_ID	Card_Combination
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
NULL	NULL

Subround_Type_ID	Subround_Name
2	Flop
1	Pre-Flop
4	River
3	Turn
NULL	NULL

Max_Seats_ID	Max_Seats
1	6
2	9
3	12
NULL	NULL

Bet_Type_ID	Bet_Type
3	All-In
2	Bet
1	Fold
NULL	NULL

Limit_Type_ID	Limit_Type
1	0.100
2	0.500
3	1.000
4	2.000
5	5.000
NULL	NULL

Register_ID	Account_Balance
1	101.781
NULL	NULL

Player_ID	Player_Name	Address	Age	Account_Creation_Date	Credit_Card_Number	Account_Balance
1	BillyBob	456 Oak St	45	2024-11-20 16:51:36.394868	4532678956789123	89.000
2	BobSmith	456 Oak St	45	2024-11-20 16:51:36.394868	4532678956789123	2188.500
3	CathyBrown	789 Pine St	29	2024-11-20 16:51:36.394868	4532678945671234	879.300
4	DavidWilliams	101 Elm St	32	2024-11-20 16:51:36.394868	4532678998765432	5389.000
5	EmilyDavis	202 Birch St	27	2024-11-20 16:51:36.394868	4532678976543210	109.000
6	FrankThompson	303 Cedar St	38	2024-11-20 16:51:36.394868	4532678965432109	291.590
7	GraceLee	404 Spruce St	35	2024-11-20 16:51:36.394868	4532678932123456	7192.800
8	HenryClark	505 Ash St	50	2024-11-20 16:51:36.394868	4532678912340987	96.000
9	IsabellaMartinez	606 Fir St	26	2024-11-20 16:51:36.394868	4532678923456781	171.600
10	JamesAnderson	707 Redwood St	42	2024-11-20 16:51:36.394868	4532678998761234	2498.400
11	KarenWilson	808 Hemlock St	31	2024-11-20 16:51:36.394868	4532678912345678	126.340
12	LeoScott	909 Cypress St	46	2024-11-20 16:51:36.394868	4532678954321987	1591.900
13	MariaKing	1010 Willow St	28	2024-11-20 16:51:36.394868	4532678945678912	540.500
14	NathanWright	1111 Alder St	54	2024-11-20 16:51:36.394868	4532678987654321	470.800
15	OliviaWhite	1212 Poplar St	33	2024-11-20 16:51:36.394868	4532678932109876	8900.200
16	PaulGreen	1313 Magnolia St	40	2024-11-20 16:51:36.394868	4532678923450981	123.450
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Round_ID	Game_ID	Start_Time	End_Time
1	1	2024-11-20 16:51:52.120015	2024-11-20 16:52:08.557984
2	1	2024-11-20 16:52:08.557984	2024-11-20 17:10:01.257696
3	1	2024-11-20 17:10:01.257696	2024-11-20 17:11:30.909794
4	1	2024-11-20 17:11:30.909794	2024-11-20 17:11:41.856811
5	1	2024-11-20 17:11:41.856811	2024-11-20 17:11:43.062410
6	1	2024-11-20 17:11:43.062410	2024-11-20 17:11:44.208381
7	1	2024-11-20 17:11:44.208381	2024-11-20 17:11:45.421881
8	1	2024-11-20 17:11:45.421881	2024-11-20 17:11:46.274781
9	1	2024-11-20 17:11:46.274781	2024-11-20 17:11:47.913179
10	1	2024-11-20 17:11:47.913179	2024-11-20 17:11:49.274789
11	1	2024-11-20 17:11:49.274789	NULL
12	2	2024-11-20 18:31:11.527199	2024-11-20 18:32:10.148965
13	2	2024-11-20 18:32:10.148965	2024-11-20 18:32:30.748234
14	2	2024-11-20 18:32:30.748234	2024-11-20 18:33:05.989024
15	2	2024-11-20 18:33:05.989024	2024-11-20 18:35:24.848753
16	2	2024-11-20 18:35:24.848753	2024-11-20 18:35:26.020039
17	2	2024-11-20 18:35:26.020039	2024-11-20 18:35:27.272640
18	2	2024-11-20 18:35:27.272640	2024-11-20 18:35:28.086909

Subround_ID	Round_ID	Subround_Type_ID	Start_Time	End_Time	Pot
1	1	1	2024-11-20 16:51:52.120015	2024-11-20 16:52:08.032636	0.582
2	1	2	2024-11-20 16:52:08.032636	2024-11-20 16:52:08.211918	1.167
3	1	3	2024-11-20 16:52:08.211918	2024-11-20 16:52:08.404221	1.752
4	1	4	2024-11-20 16:52:08.404221	2024-11-20 16:52:08.557984	0.000
5	2	1	2024-11-20 16:52:08.557984	2024-11-20 17:09:35.201816	0.582
6	2	2	2024-11-20 17:09:35.201816	2024-11-20 17:09:35.375803	1.070
7	2	3	2024-11-20 17:09:35.375803	2024-11-20 17:09:47.995069	1.558
8	2	4	2024-11-20 17:09:47.995069	2024-11-20 17:10:01.257696	0.000
9	3	1	2024-11-20 17:10:01.257696	2024-11-20 17:11:30.427009	0.582
10	3	2	2024-11-20 17:11:30.427009	2024-11-20 17:11:30.583401	1.167
11	3	3	2024-11-20 17:11:30.583401	2024-11-20 17:11:30.731546	1.752
12	3	4	2024-11-20 17:11:30.731546	2024-11-20 17:11:30.909794	-0.001
13	4	1	2024-11-20 17:11:30.909794	2024-11-20 17:11:41.394804	0.582
14	4	2	2024-11-20 17:11:41.394804	2024-11-20 17:11:41.546879	1.167
15	4	3	2024-11-20 17:11:41.546879	2024-11-20 17:11:41.698993	1.752
16	4	4	2024-11-20 17:11:41.698993	2024-11-20 17:11:41.856811	0.000
17	5	1	2024-11-20 17:11:41.856811	2024-11-20 17:11:42.603196	0.582
18	5	2	2024-11-20 17:11:42.603196	2024-11-20 17:11:42.752435	1.167

Game_Account_ID	Game_ID	Chip_Stack	Time_Created	Time_Closed
1	1	8.370	2024-11-20 16:51:51.860374	NULL
2	1	11.162	2024-11-20 16:51:52.120015	NULL
3	1	13.720	2024-11-20 16:51:52.286935	NULL
4	1	9.919	2024-11-20 16:51:52.317015	NULL
5	1	11.380	2024-11-20 16:51:52.337594	NULL
6	1	7.822	2024-11-20 16:51:52.360704	NULL
7	2	8.411	2024-11-20 18:31:11.488864	NULL
8	2	3.965	2024-11-20 18:31:11.527199	NULL
9	2	7.470	2024-11-20 18:31:11.550139	NULL
10	NULL	0.000	2024-11-20 18:31:11.567575	2024-11-20 18:35:29.043619
11	2	3.184	2024-11-20 18:31:11.586822	NULL
12	2	10.983	2024-11-20 18:31:11.603681	NULL
NULL	NULL	NULL	NULL	NULL

Bet_Time	Subround_ID	Game_Account_ID	Bet_Type_ID	Bet_Amount
2024-11-20 16:52:00.596251	1	1	2	0.100
2024-11-20 16:52:07.898093	1	2	2	0.100
2024-11-20 16:52:07.930149	1	3	2	0.100
2024-11-20 16:52:07.968650	1	4	2	0.100
2024-11-20 16:52:08.002011	1	5	2	0.100
2024-11-20 16:52:08.032636	1	6	2	0.100
2024-11-20 16:52:08.065074	2	1	2	0.100
2024-11-20 16:52:08.099378	2	2	2	0.100
2024-11-20 16:52:08.129996	2	3	2	0.100
2024-11-20 16:52:08.155345	2	4	2	0.100
2024-11-20 16:52:08.182260	2	5	2	0.100
2024-11-20 16:52:08.211918	2	6	2	0.100
2024-11-20 16:52:08.241340	3	1	2	0.100
2024-11-20 16:52:08.267871	3	2	2	0.100
2024-11-20 16:52:08.293786	3	3	2	0.100
2024-11-20 16:52:08.328363	3	4	2	0.100
2024-11-20 16:52:08.372943	3	5	2	0.100
2024-11-20 16:52:08.404221	3	6	2	0.100

Pay_Out_Time	Subround_ID	Game_Account_ID	Pay_Out_Amount
2024-11-20 16:52:08.557984	4	3	2.340
2024-11-20 17:10:01.257696	8	4	2.049
2024-11-20 17:11:30.909794	12	2	1.122
2024-11-20 17:11:30.909794	12	6	1.122
2024-11-20 17:11:41.856811	16	2	2.340
2024-11-20 17:11:43.062410	20	3	2.340
2024-11-20 17:11:44.208381	24	5	2.340
2024-11-20 17:11:45.421881	28	6	2.340
2024-11-20 17:11:46.274781	32	5	2.340
2024-11-20 17:11:47.913179	36	3	2.340
2024-11-20 17:11:49.274789	40	1	1.170
2024-11-20 17:11:49.274789	40	4	1.170
2024-11-20 18:32:10.148965	48	7	2.340
2024-11-20 18:32:30.748234	52	8	1.025
2024-11-20 18:32:30.748234	52	11	1.025
2024-11-20 18:33:05.989024	56	11	0.829
2024-11-20 18:33:05.989024	56	12	0.829

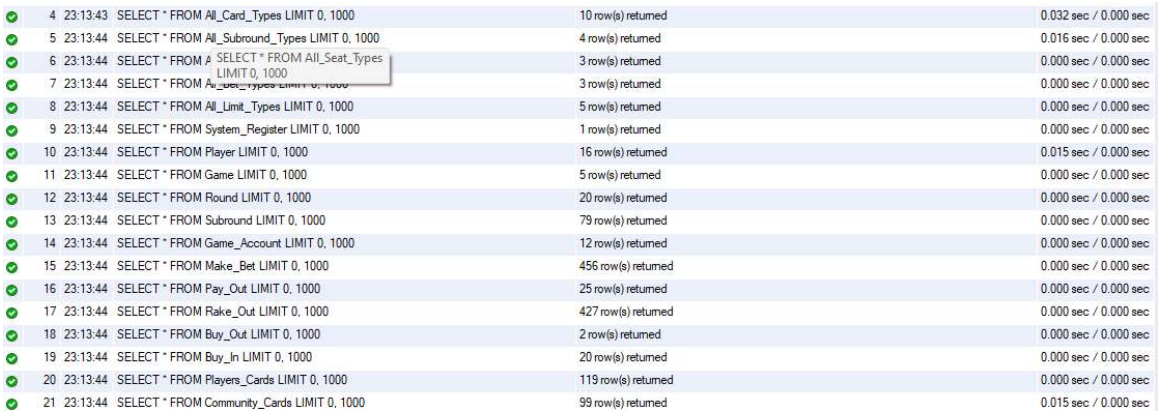
Rake_Time	Register_ID	Subround_ID	Rake_Amount
2024-11-20 16:52:00.667903	1	1	0.003
2024-11-20 16:52:07.913865	1	1	0.003
2024-11-20 16:52:07.953179	1	1	0.003
2024-11-20 16:52:07.986929	1	1	0.003
2024-11-20 16:52:08.019238	1	1	0.003
2024-11-20 16:52:08.050450	1	1	0.003
2024-11-20 16:52:08.083768	1	2	0.003
2024-11-20 16:52:08.115021	1	2	0.003
2024-11-20 16:52:08.142531	1	2	0.003
2024-11-20 16:52:08.168814	1	2	0.003
2024-11-20 16:52:08.198088	1	2	0.003
2024-11-20 16:52:08.227333	1	2	0.003
2024-11-20 16:52:08.255029	1	3	0.003
2024-11-20 16:52:08.280455	1	3	0.003
2024-11-20 16:52:08.309511	1	3	0.003
2024-11-20 16:52:08.352280	1	3	0.003
2024-11-20 16:52:08.391009	1	3	0.003
2024-11-20 16:52:08.417610	1	3	0.003

Buy_Out_Time	Game_Account_ID	Player_ID	Buy_Out_Amount
2024-11-20 18:34:15.844427	11	11	2.340
2024-11-20 18:34:15.870096	6	6	2.340
NULL	NULL	NULL	NULL

Buy_In_Time	Game_Account_ID	Player_ID	Buy_In_Amount
2024-11-20 16:51:51.879411	1	1	1.000
2024-11-20 16:51:52.133827	2	2	1.000
2024-11-20 16:51:52.270174	2	2	1.000
2024-11-20 16:51:52.302752	3	3	1.000
2024-11-20 16:51:52.327427	4	4	1.000
2024-11-20 16:51:52.348949	5	5	1.000
2024-11-20 16:51:52.372520	6	6	1.000
2024-11-20 17:10:29.970405	1	1	10.000
2024-11-20 17:10:29.987183	2	2	10.000
2024-11-20 17:10:30.003003	3	3	10.000
2024-11-20 17:10:30.026416	4	4	10.000
2024-11-20 17:10:30.047454	5	5	10.000
2024-11-20 17:10:30.065334	6	6	10.000
2024-11-20 18:31:11.515441	7	7	8.000
2024-11-20 18:31:11.540230	8	8	4.000
2024-11-20 18:31:11.558595	9	9	9.000
2024-11-20 18:31:11.576045	10	10	2.000
2024-11-20 18:31:11.594093	11	11	3.000

Player_Card_ID	Round_ID	Game_Account_ID	Card_Type_ID
1	1	1	5
2	1	2	2
3	1	3	7
4	1	4	1
5	1	5	2
6	1	6	5
7	2	1	1
8	2	2	4
9	2	3	9
10	2	4	10
11	2	5	9
12	2	6	1
13	3	1	6
14	3	2	9
15	3	3	1
16	3	4	3
17	3	5	10
18	3	6	9

	Community_Card_ID	Card_Type_ID	Subround_ID
▶	1	1	2
	2	1	2
	3	9	2
	4	10	3
	5	7	4
	6	9	6
	7	1	6
	8	2	6
	9	8	7
	10	6	8
	11	9	10
	12	9	10
	13	1	10
	14	1	11
	15	1	12
	16	4	14
	17	10	14
	18	3	14



Appendix B - Data Dictionary Index

Account_Balance	System_Register
Address	Player
Age	Player
Bet_Amount	Make_Bet
Bet_Time	Make_Bet
Bet_Type	All_Bet_Types
Bet_Type_ID	All_Bet_Types
Bet_Type_ID	Make_Bet

Buy_In_Amount	Buy_In
Buy_In_Time	Buy_In
Buy_Out_Amount	Buy_Out
Buy_Out_Time	Buy_Out
Card_Combination	All_Card_Types
Card_Type_ID	All_Card_Types
Card_Type_ID	Community_Cards
Card_Type_ID	Players_Cards
Chip_Stack	Game_Account
Community_Card_ID	Community_Cards
Credit_Card_Number	Player
End_Time	Game
End_Time	Round
End_Time	Subround
Game_Account_ID	Buy_In
Game_Account_ID	Buy_Out
Game_Account_ID	Game_Account
Game_Account_ID	Make_Bet
Game_Account_ID	Pay_Out
Game_Account_ID	Players_Cards
Game_ID	Game
Game_ID	Game_Account
Game_ID	Round
Limit_Type	All_Limit_Types
Limit_Type_ID	All_Limit_Types
Limit_Type_ID	Game
Max_Seats	All_Seat_Types
Max_Seats_ID	All_Seat_Types
Max_Seats_ID	Game
Pay_Out_Amount	Pay_Out
Pay_Out_Time	Pay_Out
Player_Card_ID	Players_Cards
Player_ID	Buy_In

Player_ID	Buy_Out
Player_ID	Player
Player_Name	Player
Pot	Subround
Register_ID	Rake_Out
Register_ID	System_Register
Rake_Amount	Rake_Out
Rake_Time	Rake_Out
Round_ID	Players_Cards
Round_ID	Round
Round_ID	Subround
Start_Time	Game
Start_Time	Round
Start_Time	Subround
Subround_ID	Community_Cards
Subround_ID	Make_Bet
Subround_ID	Pay_Out
Subround_ID	Rake_Out
Subround_ID	Subround
Subround_Type_ID	All_Subround_Types
Subround_Type_ID	Subround
Subround_Name	All_Subround_Types
Time_Closed	Game_Account
Time_Created	Game_Account

References

- Elmasri, R. and Navathe, S.B. (2016) Fundamentals of Database Systems. 7th Edition, Addison-Wesley, Boston