Funds Management

There are several tables in the schema for managing funds. They are the Account, AccountJournal, JournalTransaction, AccountAudit, AccountBalanceHistory, and AccountingAgent tables.

The accounts are stored using a standard double-entry accounting system. There are four types of accounts. Assets, Liabilities, Income, and Expenses. Accounting terminology calls Asset and Expenses *left-hand* accounts, and Liability and Income *right-hand* accounts. In the double-entry system, it must always be the case that the sum of all left-hand accounts equals the sum of all right-hand accounts. All accounts start at zero.

Accounts are modified by transactions (or journal transactions). Each transaction will modify multiple accounts; the sum of changes to left hand accounts must equal the sum of changes to right hand accounts in order to preserve balance.

For each account, there will be an entry in the Account table, which will store the *current* balance of the account. We envision the following accounts in the system:

Type	**Name**	**How many**	**Description**
Asset	Cage	1	Actual funds on hand
Asset	Receivables	1 per CCard provider	Amount owed to us by this provider for CCard debit transactions
Liability	Player Bankroll	1 per player	Money in players bankroll, exclusive of amounts currently in play.
Liability	In Play	1 per active table	Money on the table in a particular active game
Liability	Tournament Pool	1 per active tournament	The prize pool for a particular active tournament
Liability	Payables	1 per CCard provider	Amount owed to this provider for CCard refund transactions
Liability	Checks Payable	1	Amount that players have requested to cash out via check but not yet drafted
Income	Rake	1	Drop from live games
Income	Tournament Fees	1	Entry fees from tournaments
Expense	Promotions	1	
Expense	Rebates	1	
Expense	BadChargebacks	1	

It should always be the case that Assets – Liabilities – Income + Expense = 0.

Account Table. The accountType field identifies the accounts type – Asset, Liability, Income, Expense. The accountClass field identifies the account class – Player Bankroll, Receivable, In Play, etc. Balances are generally stored as a positive number, because storing Income and Liabilities as negative numbers is often confusing and leads to programming errors. The accountFactor is 1 for Assets / Expense accounts, and –1 for Income / Liability accounts, so that you can do queries which sum accountFactor*accountBalance and get the right answer. When an account is closed (player closes their account, tournament is finished, table session ended, etc) the isClosed flag is set.

JournalTransaction Table. For each transaction that modifies account balances, you must create an entry in the JournalTransaction table and two or more entries in the AccountJournal table. The JournalTransaction table identifies the time and type of the transaction, as well as who is making the transaction (acctAgentId) and on behalf of what activity (relevantId).

AccountJournal Table. Each transaction in JournalTransaction will have two or more entries in AccountJournal. The amount field identifies the amount (positive or negative) by which the account balance is changed. All journal entries in a given transaction must sum to zero (after multiplying the amount by the associated accounts accountFactor.)

AccountAudit Table. Periodically, the accounts will be audited to determine that all transactions sum to zero, and that all accounts have a balance equal to their starting value (zero) plus the sum of all transactions applied to that account. Both full audits and incremental audits (audit changes since last audit) will be supported.

AccountBalanceHistory Table. When an audit is performed, audited balances are stored in the AccountBalanceHistory table. This allows us to efficiently perform incremental audits.

AccountingAgent Table. Every agent that is allowed to update account balances (credit card monitor, cashier, chip runner, dealer, employees) will have an entry in the AccountingAgent table.

Transaction Type	Affects Accounts	Meaning of relevantId
Buy Chips	Increase Player Bankroll{player} Increase Receivable{provider}	
Cash Out (CCard)	Increase Payable{provider} Decrease Player Bankroll{player}	
Cash Out (Check)	Increase Checks Payable Decrease Player Bankroll{player}	
Write check	Decrease Cage Decrease Checks Payable	
Settle Incoming	Decrease Receivable{provider} Increase Cage	

Settle Outgoing	Decrease Payable{provider} Decrease Cage	
Player Sit Down	Decrease Player Bankroll{player} Increase In Play{table}	tableSessionId
Player Stand Up	Decrease In Play{table} Increase Player Bankroll{player}	tableSessionId
Tournament Entry	Decrease Player Bankroll{player} Increase Tournament Pool{tournament} Increase Tournament Fees	tournamentId
Tournament Payout	Increase Player Bankroll{player} Decrease Tournament Pool{tournament}	tournamentId
Empty Drop Box	Increase Drop Decrease In Play{table}	tableSessionId
Pay Bonus	Increase Promotions Increase Player Bankroll{player}	

Transactions involving the Accounts table must be performed at an isolation level of Read Stability or above. Auditing transactions provide a challenge. A mechanism for allowing auditing to be done at a low isolation level is important, as the audit queries may be long-running and we don't want auditing queries to interfere with normal system operation, but at the same time we need to be able to identify which transactions were made after the beginning of the audit and ignore those. If the transaction Ids are generated by a sequence generator that doesn't guarantee in-order generation, this will further complicate the auditing procedure.

Game Management

The game management section of the database includes abstractions for Table, Game, TableSession (live game), and Tournament.

A Game describes a single type of game, with specified rules, stakes, structure, etc, such as "Real Money 10-20 Texas Holdem", "Play Money 3-6 7 Card Stud", "Real Money Pot Limit 2-3-5 Texas Holdem", or "No Limit Texas Holdem Tournament with XYZ structure." The rules for how various games and structures are dealt exist outside of the database.

A Table describes a virtual table where a game can be played. Each table has an In Play account associated with it. Tables can be created at will and torn down if no game is in play at that table.

A TableSession is an instance of a Game being played at a Table. Table sessions can be private or public; if they are private, they are associated with a PlayerGroup.

Table Table. Each Table has an associated dealerAgentId which will be used to make accounting transactions for that table. Tables have names, for use in reports and in the client UI. If a table has not been torn down, the isActive field will be set. Generally there will be no need to tear down tables; unused tables can be held in reserve.

TableSession Table. For each live game in progress, there will be an active TableSession. After a game is torn down, the isActive field will be set to false but the TableSession record will remain in the table. Each TableSession is associated with a Table, a Game, and an account which represents the money in play at that table. If the system crashes, the TableSession table can be used to reconstruct the set of currently active games.

Game Table. The Game table identifies all the types of games that the system can support. The gameType field identifies the underlying game – HoldEm, Stud, Stud Hi-Low-8, etc. The gameStructure field identifies the betting options – spread limit, fixed limit, pot limit, no limit. The gameStakes field identifies the stakes, and the meaning of this will depend on the structure. For fixed limit games, it will be the size of the small bet. The isRealMoney and isTournament fields have the obvious meaning. The gameName field will be the name displayed to the user.

Tournament Table. Each tournament is associated with a Game and an account which represents the tournament pool for this tournament.

PlayerSession Table. For each player session, a record in the PlayerSession table is created. When a player logs on, the system looks for a session for that player which is marked active; if it finds one, the player is reconnected. The isActive field identifies whether a session is active or not.

CurrentlyPlaying Table. The CurrentlyPlaying table identifies who is currently playing in the system, associating player sessions with table sessions. This is where the dealers will maintain each players chip stack. Each hand, the dealer will update the CurrentlyPlaying table to update the chip stacks. If the system crashes, the CurrentlyPlaying table can be used to reconstruct the current state of the system.

CurrentlyWaiting Table. The CurrentlyWaiting table identifies who is waiting for each active table session and in what order. If the system crashes, the CurrentlyWaiting table can be used to reconstruct the current state of the waiting lists.

TableAction Table. The TableAction table records events which change the contents of the CurrentlyPlaying table other than hand completions. When a player sits down, stands up, changes seats, or buys more chips, an entry is created in the TableAction table. Defined action types are: Sit, Stand, BuyChips, and ChangeSeats.

PlayMoneyAccounts Table. For games which are played with play money, the end-of-hand settlement is identical (update CurrentlyPlaying, write entries to Hand and HandAction) except for the taking of a rake. When table action is executed (such as sit

down, stand up, etc), the PlayMoneyAccounts table is updated instead of using the Accounts and JournalTransaction tables.

Hand, HandAction, PlayerHand tables. For each hand played, there will be an entry written to the Hand table and multiple records written to the PlayerHand and HandAction table. For each player dealt in, the players cards and their seat number are written to the PlayerHand table. Each distinct action, in order, will have an entry in the HandAction table. Action types include ante(amount), blind(amount), check, bet(amount), raise(amount), call(amount), callAllIn(amount), raiseAllIn(amount), dealHoleCards(nCards), dealUpCards(nCards), dealCommonCards(nCards), exposeCards, muckCards. The dealer bots will wait until the conclusion of the hand to write all the hand result entries.

Tournaments. When a tournament is started, a separate table session will be created for each table in the tournament. The tournament dealers or tournament director must arrange for players to move from table to table as tables are broken. If a table session is part of a tournament, the tournamentId will be filled in. The TournamentPlayer table identifies who has played in each tournament and what their outcome was.

Other Tables

CreditCard Table. Credit card numbers should be encrypted before being placed in the database. Only the cashier should know the private key required to decrypt credit card numbers.

PlayerPassword Table. The PlayerPassword table is not readable by any user. Password checking and changing is done through stored procedures. The Login stored procedure will check the password, and will create a PlayerSession if the password is correct. The ChangePassword stored procedure will require knowing the correct current password. There will also be a stored procedure for resetting the password which floor personnel can use; this will require the entry of a special reset password.

Table Constraints

The following table summarizes constraints enforced by the database, either through referential integrity checks, triggers, or permissions. The following abbreviations are used for common constraints:

- NoDel – Row may not be deleted. Enforced by trigger. This may be relaxed in the future if we find we need to archive old data such as hand histories, but for the time being, there are a lot of things we don't want deleted. Note that this is a protection against both bad program logic and against an attacker gaining access to the database.
- NoMod – Row may not be modified once created.

- InactiveNoMod – If a row is marked inactive (the isActive field is false or the isClosed field is true), the row may not be modified at all. Enforced by trigger.
- InactiveRequires(P) – If a row is being marked inactive, the condition P must be true. Enforced by trigger.
- InactiveAction(A) – When the row is marked inactive, the action A is executed.
- Check(P) – A CHECK constraint enforces the condition P.

Table	**Constraint**
Account	NoDel InactiveNoMod InactiveRequires (accountBalance=0) Check(balance >= 0)
AccountJournal	NoDel Check(amount >= 0)
JournalTransaction	NoDel Check(transactionType is valid)
AccountAudit	NoDel
AccountBalanceHistory	NoDel Check(balance >= 0)
AccountingAgent	
Employee	
Table	NoDel InactiveNoMod InactiveRequires (no active table sessions on this table)
TableSession	NoDel InactiveNoMod InactiveRequires (table account balance = 0) InactiveRequires (no players currently playing at this table)
Game	NoDel, NoMod
Tournament	NoDel InactiveNoMod InactiveRequires(tournament account balance = 0) InactiveAction(privateGroup = null)
Hand	NoDel
TournamentPlayer	NoDel
HandAction	NoDel, NoMod
PlayerHand	NoDel, NoMod
TableAction	NoDel, NoMod
CurrentlyPlaying	
CurrentlyWaiting	
PlayerSession	NoDel, InactiveNoMod
PlayerGroup	

Player	NoDel, InactiveNoMod InactiveRequires(account balance = 0) InactiveAction(set account inactive)
CreditCard	NoDel, NoMod.
CreditCardTransaction	NoDel
PlayerInfo	NoDel

Defined Users

The following users will be defined in the database:

- Owner. This is the user which owns all tables and some stored procedures. No one may log in as owner. However, certain stored procedures may run as Owner and be executed by other users.
- Cashier. The cashier may make transactions involving the exchange of chips for money.
- Dealer. The dealer may make transactions involving the CurrentlyPlaying table, including the exchange of bankroll for chips in play (chip runner functions), write hand histories, and transfer the contents of the drop box to the cage.
- Floorman. The floorman may create and break down tables and table sessions, start tournaments, and manage waiting lists.
- User web agent. The user web agent can access tables sufficiently to retrieve a users account, transaction, and hand history. It can also execute the ChangePassword stored procedure.
- Employee web agent. The employee web agent can access tables sufficiently to retrieve user's account, transaction, and hand histories, as well as run the ResetPassword stored procedure.

Stored Procedures

All database actions will be handled by stored procedures, which will be protected so that they can only be run by authorized agents. For procedures that update account balances, the procedure will validate that the amounts sum to zero. Some procedures can run at the Cursor Stability isolation level; some will require Read Stability (including all that update the Accounts table.)

There will be stored procedures for all of the Account operations (DepositFunds, CashOutCredit, CashOutCheck, BuyChips, ReturnChips, TournamentEntry, TournamentPayout, EmptyDropBox, etc.) There will also be procedures to update the distribution of funds in the CurrentlyPlaying table at the end of the hand, to write the hand history to the database, to update the player statistics, etc. Session creation will be handled by a stored procedure that validates the password, checks to see if a session already exists, and creates the PlayerSession record.

Many common actions will require executing more than one stored procedure within the context of a transaction. For example, when recording the results of a hand, the dealer will write the hand history and will update the distributions of funds in the CurrentlyPlaying table. Each of these actions will be a separate stored procedure, but they can be executed together in the same transaction.