System Architecture

The following logical agents will participate in the system. Some may actually be performed by the same server process.

- Lobby Server. This is the main server to which clients will connect. It is responsible for creation and teardown of tables, starting table servers, and managing waiting lists.
- Table Servers. Each active game will be managed by a table server. Table servers will be created by the Lobby server. Table servers will manage the play of the game, recording the hand results to the database, updating the CurrentlyPlaying table at the end of each hand, collecting the rake, awarding the pot(s), and tabulating player statistics. Additional functions the dealer may perform include determining eligibility for a bad-beat or high-hand jackpot.
- Tournament Director. For each active tournament, there will be an agent that monitors the status of the tables, and breaks and combines tables appropriately.
- Audit Agent. The audit agent will run periodically in the background, and run various auditing queries to ensure the consistent state of the database and identify potentially suspicious behavior. Items to be audited include the balances of each account, the balance of each transaction, and win rates and loss rates for players (to detect dumping).
- Web Server. The web application will allow users and employees to manage and monitor the status of the card room without having to start the client application. Clients will be able to see account histories, hand histories, current status of the card room (number of players, who's logged on, list status, etc.) Employees will be able to access management reports, hand histories, current games, status of credit card transactions and cash outs, pay bonuses, freeze accounts, etc. It may be sensible to have multiple separate employee areas, each with their own password, to further reduce the risk of unauthorized access. As the employee section of the web site is probably the weakest link in the security chain, it is recommended that additional protections be put in place to prevent users from accessing the employee web site section, such as IP address restriction or requiring that the employee's browser present a certificate.
- Cashier. The cashier will process requests to buy chips via a credit card and requests for cash outs.
- Chip Runner. The chip runner will manage transfers from a player's account to and from his table bankroll (stored in the CurrentlyPlaying table.) The chip runner function may be performed by the table server or the lobby server.
- Credit Card Processor Gateway (CCPG). The CCPG will monitor the CreditCardTransaction table for new credit card transactions and post them to the credit card processor. When a response is received, it will update the status of the CreditCardTransaction table so that the cashier may update the client's account and notify the client.
- Clients. The clients will connect to the Lobby Server initially. When a session is requested with another server, such as the Cashier or a Table Server, the Lobby

server will notify the appropriate other server so it will accept the client's connection, and instruct the client to connect to the other server. This can be done by having the lobby server create a session password which it tells to both the client and target server over the secure connection.

Scaling

In the initial deployment, it is likely that the database server, the lobby server, and the table servers will run on the same system. As the traffic increases, the application will be distributed across multiple systems. The logical first partitioning is to move the database server to its own machine. Since many critical functions are implemented as stored procedures, the additional network latency cost of having the database server on another system will be low, as there will be fewer round-trips than if the application were executing multiple database operations per transaction.

The system can scale further by distributing the table servers across multiple systems. Since the table servers will be launched by the lobby server, there needs to be a secure mechanism for launching processes on other systems. Having a "launch server" on the remote system which registers with the lobby server and launches table servers when the lobby server requests would accomplish this.

As we approach the point where we reach the limit of active connections for a single process, the lobby server will also have to be distributed. The launch server could launch a lobby server on another node, and the main lobby server could dynamically load-balance clients to the new lobby server via a "Redirect" message. Since the waiting lists are stored in the database, multiple lobby servers could still share access to the waiting lists. However, having multiple lobby servers requires a great deal of additional synchronization and communication between agents, and so we will revisit the notion of multiple lobby servers at a later date.

Interprocess Communication

The client will connect to the lobby server. At any time, the lobby server may redirect the client to reconnect to a different lobby server. This mechanism can be used for both load balancing (deferred until a later version) and so that lobby servers can be restarted or upgraded without disconnecting clients that are currently playing.

Each table server should maintain a connection with the database server and with the lobby server. The client maintains a connection to a lobby server and to each table server at which they are playing or observing. When a client wants to sit down at a table, the lobby server will generate a random session password, and will forward that to both the table server and to the client, and the client can use that to claim his seat.

Communications between client and servers are encrypted at the transport layer. Communications between servers should also be encrypted, as otherwise the system is vulnerable to snooping attacks in the event another host on the server network is compromised. Having the server network behind a well-configured firewall reduces this risk somewhat. Alternately, the lobby server and table server could initially communicate via SSL so that the lobby server can dispense access keys to the table server, and then they could switch to a lower-overhead communication channel for less sensitive communications. If SSL decryption is to be offloaded to another host, then the server network MUST be behind a well-configured firewall.

Error Recovery

When a client is disconnected from a lobby or table server, the server should mark that client as disconnected, and await a reconnection. After a certain timeout interval, the server should assume the client is not coming back and begin the process of releasing resources in use by the client. Table servers can kick the user from the table, releasing the seat and returning the player's chips to the bankroll. Once a player has been kicked from all active table sessions, the lobby server can mark the record in the PlayerSession table as inactive.

When a client connects and attempts to log on, the lobby server will execute a stored procedure to log the client on and create a table session. If that player has an existing session, that procedure will instead return the id of the active session and an indication that a previous session has been recovered. The lobby server will then instruct the client which table servers to reconnect to. The lobby server will ensure that the same client does not log on twice through this session-recovery mechanism.

If the lobby server crashes, the clients will reconnect to the lobby server, which will discover that these are recovered sessions and accept the connection. The table servers will participate in a "heartbeat" protocol with the lobby server, so that the lobby servers can detect table server crashes. When a table server crashes, the lobby server restarts it, and directs clients to connect to the new server.

Live System Upgrades

It is desirable to be able to upgrade the server images (lobby server, table server) while the system is running and clients are playing. With the "Redirect" mechanism described before, the lobby server can launch the new lobby server, instruct all clients and table servers to connect to it, and then shut itself down, without the users noticing. Similarly, a table server can be upgrade by having the lobby server launch the new table server, tell the clients at that table to switch to the new table server at the end of the current hand, tell the table server to stop dealing at the end of the current hand, and the clients should never know that a switch has been performed.

Upgrades to the database may be more complicated if tables need to be reconfigured or stored procedures changed. Commercial databases have many features for supporting live upgrading of databases, but these all require support from the application.

Depending on the complexity of the upgrade, it may or may not be convenient to upgrade database code while the system is running.

Security Considerations

Communications between clients and server will be encrypted at the transport level. Communications between servers may or may not be encrypted at the transport level, depending on the network configuration. Additionally, sensitive information such as passwords or credit cards may be further encrypted with session keys, to reduce exposure to “man in the middle” attacks and to reduce the exposure if an SSL preprocessor system is employed at the server side. If there are multiple servers, it is important that the servers be behind a well-configured firewall to reduce the risk of system compromise and exposure to snooping attacks.

Important keys (such as passwords for database access) will be stored in an encrypted file, key server or smart card; the lobby server will retrieve the keys for other agents (table server, web server, cashier) and communicate the keys to the agent through a secure mechanism when the lobby server starts the agent.

Credit card information is particularly sensitive; the credit card numbers should be stored in the database encrypted with the public key of the cashier agent. Only the cashier will be able to decrypt the credit card numbers.

Development Methodology

Developing a secure, reliable, scalable database application differs from building a standard server application in a number of ways. The following principles should help to achieve this goal.

Short Transactions. In a database system, high concurrency is achieved by keeping transactions short, so that locks on critical resources can be released as soon as possible. This means collecting relevant data for the transactions and then executing all the database accesses together and then committing the transaction, rather than interleaving database access with program logic. Transactions should then be committed with an explicit commit; auto-commit should be disabled. Also, on many database systems, it is necessary to execute a commit even after simply reading data from the database, as the read may have acquired database locks which are held until commit time. A commit will release any locks held; unless you want to retain the lock, plan to modify the data you’ve just read, or use it to write new data, you should commit immediately after a read operation. The operating principle here is “Commit Early, Commit Often.”

Batched Transactions. Batching data operations and performing them together as described above also allows for recovery from involuntary transaction rollbacks. If the database server rolls back a transaction because a deadlock has been detected or a required resource is unavailable, it will complete the operation with a failure status indicating rollback. If the application has been notified that the transaction was involuntarily rolled back, the application should wait a small period of time, and then retry the transaction. If the program logic is separated from the database logic, this is easy to do.

Use of Triggers. Triggers can be a performance hazard if associated with tables that have frequent activity. However, for tables whose data doesn't change very often, such as the PlayerSession table (only changes when a player logs on or logs off), triggers can be used to maintain various consistency conditions. Examples:

- If a session is marked inactive, an UPDATE trigger can be used to efficiently render the record read-only.
- If a table is being broken down, an UPDATE trigger can ensure there is no active session on that table.
- If a table session is being broken down, an UPDATE trigger can ensure that the associated money-in-play account balance is zero.
- For tables like Account, records should never be deleted, only marked inactive. A DELETE trigger can prevent a record from being deleted.

Triggers can efficiently implement constraints such as these that can help to ensure that important operations are applied correctly to the database.

Use of Permissions. Commercial databases support a rich set of permissions that can be used to ensure that data is only modified or accessed by the appropriate agents or users. All important tables in the database, such as the Accounts table, the CurrentlyPlaying table, etc, will be protected in such a way that they can only be modified by a special user. All modifications to those tables will be done through stored procedures which run as that user, and execute access on the stored procedures can be reserved for agents which actually need to perform those modifications. This allows the dealer agents to, for example, move money from one account to another without being able to create or destroy money.

Each type of agent (dealer, chip runner, lobby server, user web application, employee web application, etc) will have a user id in the database access schema, so that permissions on particular operations can be reserved for those who actually need to perform them.

Use of Stored Procedures. All important database operations will be done through stored procedures. This has several advantages over inline execution of SQL code:

- SQL code is localized to the stored procedure, making maintenance easier.
- Stored procedures can implement and enforce invariants in the database, protecting the database against possibly unrecoverable corruption due to program error in the more-frequently-changing application code. This is also a useful design partition, making the overall system easier to correctly develop, modify, and maintain.

- Stored procedures can be compiled, making them more efficient, and since they execute in the server process, multiple operations can execute faster since only one round-trip between the client and server is required. Also, an SQL procedure call is a less expensive operation to prepare and execute than an ordinary SQL operation.
- Since stored procedures will perform higher-level operations, application code will be more readable.
- The possibility to migrate the application to another database platform is made easier, since there is less SQL code in the application.
- Stored procedures allow for important operations to be secured from misuse.

Password and Key Management. When a server is created, it must be apprised somehow of the password for its role in the system. Storing passwords in disk files is a serious security risk. A reasonable approach would be to have passwords for each of the various agents stored in a key server or a file that is encrypted with a master key. The master key would be communicated to the lobby server at startup by the operator or through a smart card. When the lobby server creates the various other servers, it will communicate to them their keys. The lobby server should checksum any applications it is launching and compare them with the known value before telling them their key.

Appropriate operational and security procedures should be developed and implemented to protect the physical integrity of systems, and to control access to passwords and security keys. Both external and internal threats should be considered in the design of these procedures.

Use of database as a backing store. Certain data retrieval operations may be cached by the server, if the server is confident the data will not be modified. For example, when a table server executes a “buy more chips” operation during a hand, the new chips will be recorded in the chipsBehind field of the appropriate player record. At the start of the next hand, the dealer will move those chips to the player’s table bankroll. Since the server performed the buy operation, it should know whether players have money behind or not and should not have to issue a query to find this out, but if the server crashes, the replacement server will find the data it needs in the database.

Testing and Deployment Methodology

Maintaining user confidence in the reliability and integrity of the system is very important – one bad experience posted on an Internet newsgroup can have a serious effect on the site’s reputation, not to mention the possibility of serious failures being reported in the mass media (e.g. Egghead credit cards, eBay crashes, etc.) To avoid deployment errors, a three-stage testing and deployment methodology should be employed. The following systems are required:

- Development systems. Development systems are reserved for development use. Only normal engineering testing should be done on development systems, during development.

- Staging systems. Prior to deployment of a new release of the application and/or database code, testing should be done on the staging server, which is reserved for testing. The staging server should be as close in configuration as possible to the deployment server, ideally identical to it. The testing lab will also require client machines, for testing the staging server under load. The client machines should, in aggregate, be faster than the staging server so that the server can be pushed to saturation.
- Deployment systems. Only once a release has passed through the full testing process can it be deployed to the deployment server.

The development process goes through the following stages:

1. Development. Code is developed on the development systems. When a release candidate is identified, the developer produces an installation kit, which includes all required system files, on appropriate install media, and an installation script (or instructions) which can be used to deploy on a clean base system *and* applied to a running system of known previous version.
2. Clean-slate regression testing. All testing must be done on the staging server. In this stage, the staging server disk is wiped and restored with a backup image to a baseline OS installation and an empty database, and then the software is installed from the installation media. Regression tests are run.
3. Incremental testing. The staging server data is wiped and restored with a backup image from the deployment server data. Any required incremental data transformation scripts are applied (if the schema has changed), and regression tests are run. The regression tests should include tests for correct behavior of the system, performance and resource usage under high load, and performance benchmarking.
4. Predeployment. The deployment server, software and data, is backed up.
5. Deployment. If the software has passed both clean-slate and incremental testing, then it can be deployed on the deployment server. If at any stage a test has failed, the process *must* restart at step one. Trying to “fix it” in the test environment is a very common cause of introducing errors into the production system.

Management and operational processes should be put into place to ensure that this process is followed, with exceptions made, if at all, only when authorized by high level management after a proper risk assessment.