**Team #3 Final Report**

**Chess Database**

Danny Luong - 007969221 | Luther Pedeglorio - 008503664

Nicholas Randhawa - 008457293 | Anahit Sarao - 008435583

Dennis Simon - 007742215

**The choice of database project:** Chess Game Database

**The choice of database engine**: MySQL

**DB application technologies, frameworks, languages, etc.:**

**The technology, frameworks, and language used:**

-QtCreator IDE

-Qt Frameworks

-C/C++ language

-Tcp Socket Communication

-MySQL

-MySQL++ Library

-HeidiSQL

-Debian Linux Server

**Final list of functionalities/operations:**

-Admin login to unlock insert/update/delete commands

-Sha256 encryption for login password

-Guest commands for login/help/query

-Supports specific queries like:

-lobby\_stats: queries the stats for a given lobby

-elo: queries the player information for a given elo/skill level

-punish: queries the player information for a given previous punishments

-winlongmatch: queries the player information for the winner of the longest match in history

**ERD and Relational Model**

REVISED_CMPE138_TEAM3_ERD.png

Figure 1 Final Design of Database Portion

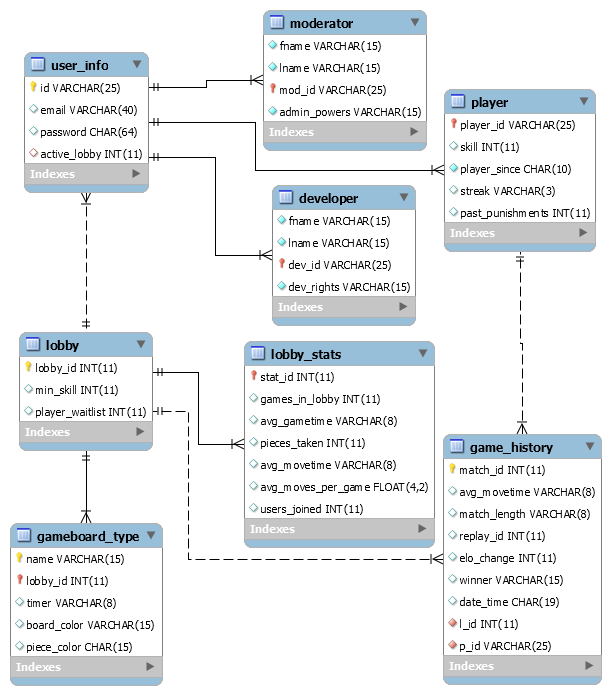


Figure 2: Relational Model/Table View

**Specification of each DB object**

We have a total of 8 tables in our database. 7 of them are from the regular entities on our EERD and 1 is from the weak entity. The first table we will talk about is user\_info. This table contains the information related to the user superclass. It has 4 attributes. The first is the primary key, id. Id is unique because it distinguishes one user from another. The other 3 attributes are not unique and those are email, password, and active\_lobby. A user can make an account on multiple emails, can have a password encrypted in sha256 as another user, and can be in a lobby with other users of course. Active\_lobby is a foreign key to lobby\_id, if user is not in a lobby the value is 0 but if they are in a lobby the lobby number will be displayed. Our user\_info superclass has 3 subclasses called player, moderator, and developer.

The moderator subclass table has just 4 columns. The first 2, lname and fname, display a moderator’s name. Mod\_id is the primary key that is also a foreign key to user so each mod will be identified separately. Admin\_powers have 2 string options of “full” or “limited” which grants an admin different rights in terms of keeping regulation in the chess game which is the sole purpose of the moderator in our chess schema.

Our developer table also has 4 columns. Just like moderator, the first 2 are for the first and last name of the developer. The next one is the unique dev\_id that references the id from player\_info to distinguish devs from each other. Our final column in the developer table is dev\_right which grants different programming rights to devs from QA to Developer.

Player is for players of the chess game and has 5 attributes, or columns. The first column is the primary key player\_id which is a unique varchar value for players. Other columns in player include skill, player\_since, past\_punishments, and streak. Skill will allow users to have a rank associated to them that can change with each game. Player\_since shows when a user joined the chess database as a player. Past\_punishments are an integer that shows how many times a player has been in trouble for cheating. Finally, streak will show the current run of wins or losses that a player has. The player table has a foreign key to game\_history.

The game\_history table has 9 columns. It contains a primary key of match\_id. Then it has some columns that tell statistics on the history of the desired game. There’s avg\_movetime, match\_length, and date\_time which are self explanatory. Date\_time is a varchar that displays when a game ended in a MM/DD/YYYY@HH:MM:SS format. Elo\_change is another column that is an integer and its role is to say if the player, linked to the game via the p\_id column referencing the player\_id, gained or lost rank following the match. L\_id is an integer that references the lobby\_id from lobby so each game\_history has the lobby it was played in. The winner column is a varchar that displays the id of the winning player. Finally, replay\_id is a unique integer column that contains a replay of the match for the player.

Our sixth table is titled lobby\_stats. This table has a primary key stat\_id that references the lobby\_id from lobby. The purpose of this table is to show the statistics for each lobby. Games\_in\_lobby is an integer value that shows the total number of games that have been played in that lobby. Avg\_gametime and avg\_movetime are varchar formatted columns that say how long the average game and move are, respectively, in an HH:MM:SS format. Pieces\_taken is a column displaying the total number of pieces that have been taken across all games in the lobby in an integer format. Users\_joined keeps track of the amount of users that have visit the lobby. The final column is avg\_moves\_per\_game and that is a float displaying how moves are made per game on average.

Our seventh table is lobby. It only has 3 attributes, but may be the most important table as 4 different tables reference its candidate key and primary key, lobby\_id which is a unique integer value. Each lobby has an integer player\_waitlist that will hold the count of all users waiting to get into a game. The third column is min\_skill as a player needs a minimum rank to get into the lobby in order to make matches more competitive.

Our eighth and final table is gameboard\_type which was a weak entity on our EERD. It was weak because each board has a unique name, but the name is not unique to the entire system. Instead, it is only unique to each lobby. In other words, a lobby may contain a board with the same name as a board in another lobby. Because of this, name and lobby\_id, referencing the id from the lobby table, make up the composite primary key together. Name is essentially a name of some theme for the board. Timer, piece\_color, and board\_color are varchar columns. Each board has a time limit per move and that’s where timer comes into play. For piece\_color, the board will give a user a default piece color which is why that one was needed. Finally, board\_color will store the color of each board type.

**Functional Dependencies and Normalization:**

Our database contains seven entities and one weak entity, to satisfy First Normal Form (1NF), each entity and weak entity is represented as a table with a primary key. In the case of the weak entity, the foreign key, in conjunction with the identifying primary key is used as composite key. Therefore, the database satisfies 1NF because each table has a primary key and all non-key columns depend on this primary key.

Seven of the eight tables satisfy Second Normal Form (2NF) because each of the non-key columns depends on the primary key. This means that there is no partial dependency. However, user\_info table is problematic because there are four columns, three of which are all unique among the rows, namely id, email and password. Id and email are composite key and yet password can depend on either id or email or both.This means that there is partial dependence.

This decision is done for compactness, as a better solution would have id and password, note how this table would simply have two columns, with the rest of the information about the user on a different table with id again as the primary key.

Again seven of the eight table s satisfy Third Normal Form (3NF) because they do not have transitive dependencies. This means that each non key column is directly functionally dependent on the primary key. The user\_info again is the problem because it fails 2NF criterias and therefore also fails 3NF criterias.

**Implicit/explicit Database Transactions and Application Log:**

All SQL statements were set to AutoCommit transactions. The Implicit\_Transactions function is turn OFF to allow atomicity of all transaction operations. This allows a better form of consistency, isolation and durability through automated transaction. The transaction and application log are stored within the var/log directory within the server.

**Final Design of Database Application Portion:**

The following technology, frameworks, and language were used:

-QtCreator IDE + Qt Frameworks were used for creating the simple client to send commands to the server

-C/C++ language was used to create the server and client.

-The server program was run on a Linux Debian server.

-TCP Socket Communication was used to communicate between the server and client

-Client<->Server<->Database model

**Database Access Technology**:

The project uses a MySQL database and utilizes the MySQL++ wrapper library for server to database communications to send requests and retrieve data from the database. HeidiSQL was a convenient open source frontend for MySQL to be able to visualize and manipulate the database directly.

**Sample Execution**:

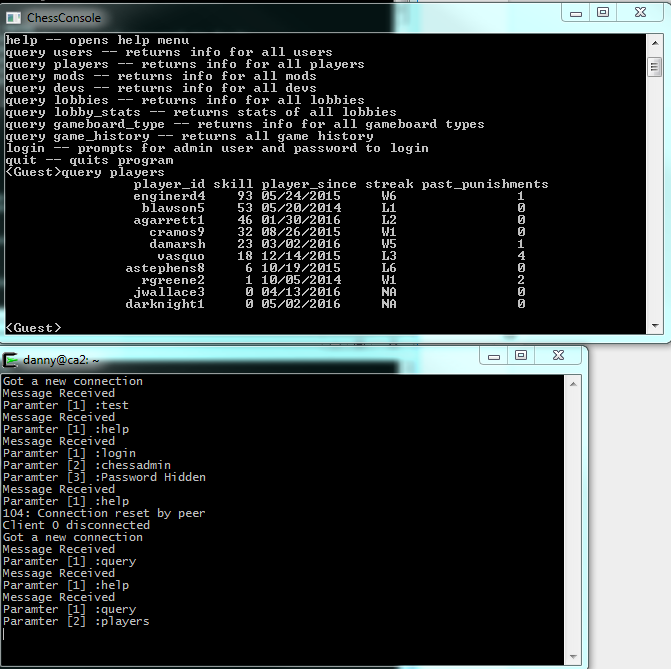


Figure 3: Sample query example

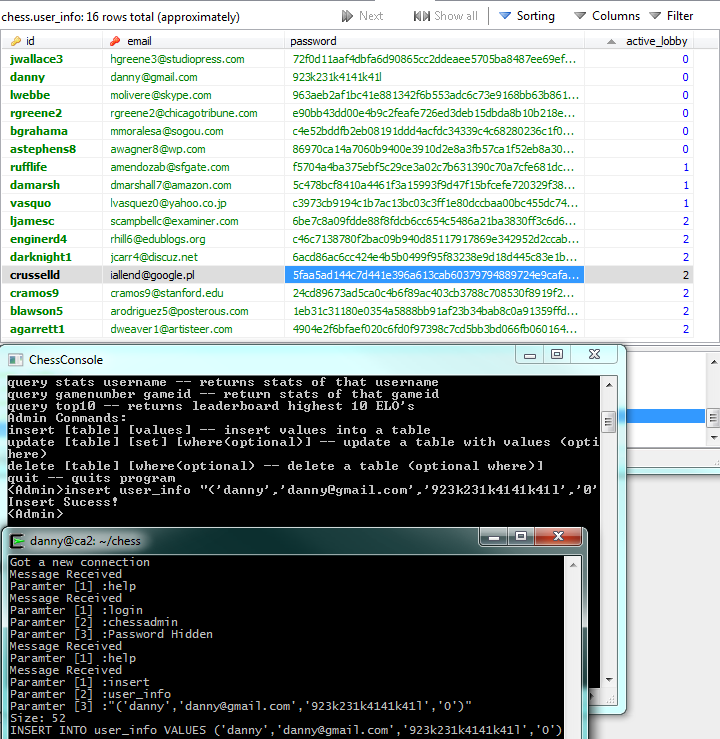


Figure 4: Admin insert example

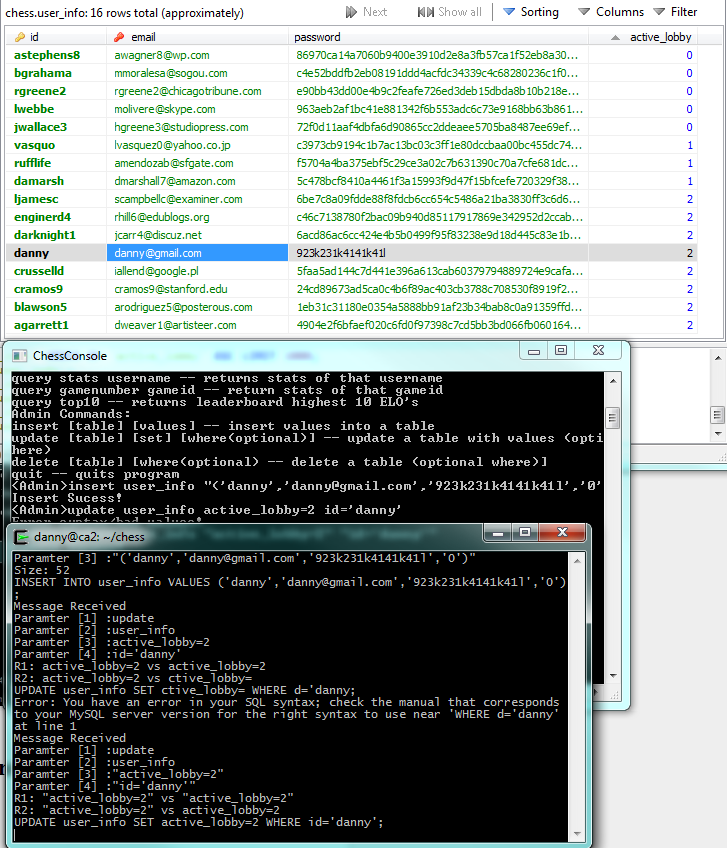


Figure 5: Admin update example

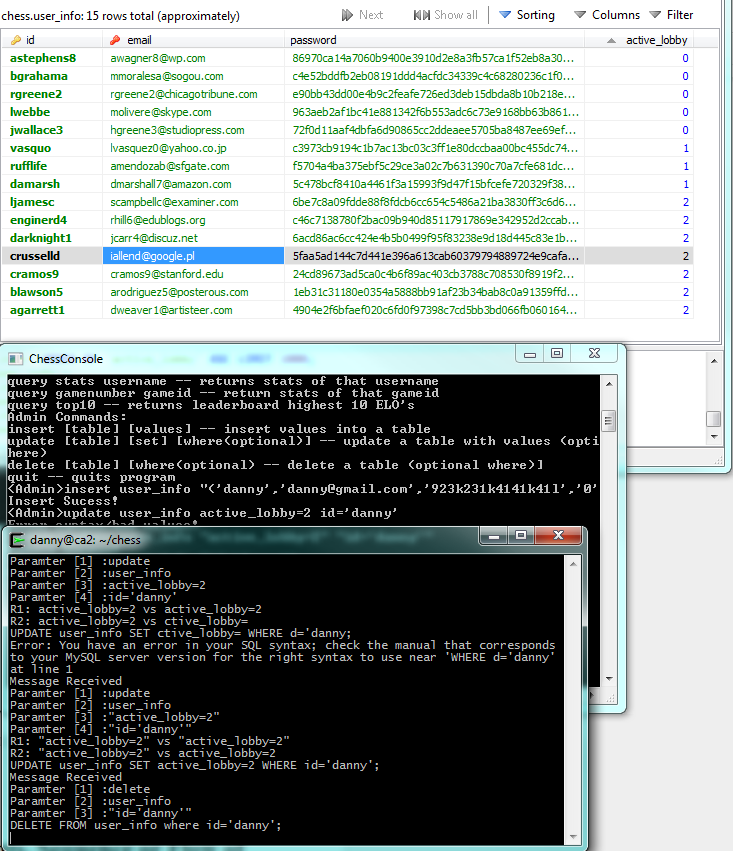
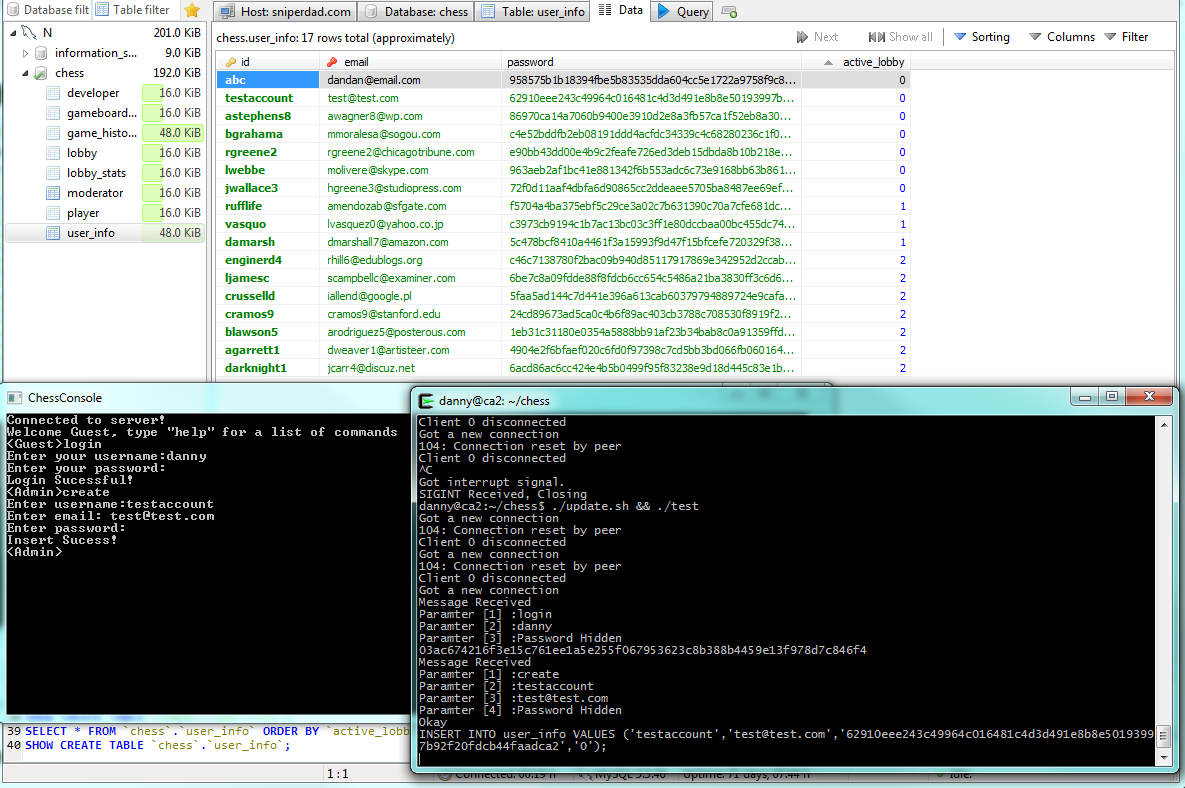


Figure 6: Admin delete example

Figure 7: Create account example

**Final Architecture, Components, Sequence or Flow of data:**

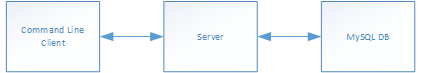


Figure 8: Basic architecture of application

**Any major design decisions and tradeoffs:**

For the design of the database application, we compared the use of simple command line client application versus a web based application. We decided on creating a simple command line client application to communicate with the server due to allow us to focus more on the database side of things. The command line client would be simple and the more complex queries and coding would be on the server side. Tradeoffs were to sacrifice a more visually pleasing application versus a simple command line client application that have the bare necessities to be able to query the database and have admin commands. For the password storage/hashing in the DB of user we had to decide between sha56 and md5 hashing and chose sha56 because we felt it was a better encryption method.

**Any major modifications from the proposal, ERD, EERD and why**

We made several changes to our ERD post-submission, including a few major changes:

* We removed the chess\_piece composite attribute, including all sub attributes connected to it. We did this because it was unnecessary to know the statistics for individual pieces during a game(like piece history, start/end positions, etc.) and it also over complicated the SQL database and made it cluttered/messy because there was no direct way to deal with composite attributes.
* We added a player\_id key attribute to the player entity because every other disjointed(developer, moderator, player) entity from user had its own key id attribute. Also because we needed it for a foreign key to match to game history.
* We added an active\_lobby attribute to the user entity in order to show if a user is in an active lobby at the present state.
* We removed the relationship from user to gameboard\_type(picks) because a user doesn’t actually pick a game board type, they join a lobby which has a certain game board type. They cannot change the game boards type they can only join a lobby with a certain game board type.
* We removed the grid\_number attribute from the gameboard\_type weak entity because gameboard\_type will always have the same grid numbers since all chessboards must have uniform grids.
* We added stats\_id to lobby\_statistics entity because the professor suggested that every entity should have a key attribute and we didn’t have one prior.

**Any unique designs you are proud of**

* Not unique, but we are particularly proud of our ERD because we revised it several times over the course of the semester to fit exactly what we wanted to do with our app/DB and spent quite a bit of time refining it even after the original submission.
* Our relational map because it’s generation and keys all match up with the relationships and our plotted design from the ERD meaning our DB functions properly.

**Final major milestones and task lists:**

Danny Luong

* Finished basic command line application client (4/15).
* Made an client .exe file for release (4/16).
* Created basic server code (4/20).
* Added TCP communication between client and server (4/20).
* Added admin access level via session class (4/22).
* Added message decoding into arguments (4/24).
* Coded some commands for client users (4/25).
* Coded SQL queries on server code (4/26).
* Added Sha256 hashing for login (4/28).
* Added admin commands to insert/delete/update (5/1).
* Tested guest commands between client and server for queries (5/3).
* Tested admin commands to insert/delete/update (5/4).
* Added some error handling for server (5/5).
* Added application logging (5/6).
* Worked on DB App part of report and presentation slides (5/7).

Luther Pedeglorio

* Worked on ERD with other group members (3/14).
* Specification of each DB object, e.g. normalization, etc (5/7).
* Did transaction for SQL code (4/30).
* Did logging for SQL server (4/30).
* Worked on project report(Specification, etc) (5/7).

Nicholas Randhawa

* Worked on ERD with other group members (3/14).
* Did ERD modifications post submission with Dennis (3/14-onwards).
* Created tables in SQL script (4/27).
* Populated sample data in SQL script (5/6).
* Coded queries into application (5/7).
* Created relational model from database tables along with Dennis.(5/7).
* Modified tables in SQL script to fit requirements/frontend app (4/28-onwards).
* Worked on project report section describing all database objects (5/7).

Dennis Simon

* Worked on project proposal(2/29).
* Helped create ERD with other group members(3/14).
* Did all ERD modifications along with Nick post submission.(3/14-onwards).
* Created DB tables in SQL script (4/27).
* Coded user queries into application (5/8).
* Fixed several issues where SQL script/relational model didn’t match up with ERD (5/1-onwards).
* Modified tables in SQL script to fit requirements/frontend app (4/28-onwards).
* Created relational model from database tables along with Nick (5/7).
* Worked on project report sections pertaining to sections of project worked on: ERD modifications, improvements, unresolved issues, the database, postmortem, etc (5/7).
* For project slides did user superclass and subclass slides/explanations for presentation (5/8).

Anahit Sarao

* Worked on ERD with other group members (3/14)
* Did transaction for SQL script (4/29).
* Did logging for SQL server (4/29).
* Testing and Debugging (5/1-5/8).
* Worked on project report sections(Transaction, Logging ) (5/7).
* Finalized project report and slides (5/9).

**Test plan execution**

Testing the database application involved including catch and error statements to see if a command was being interpreted correctly by the server and if the database rejected the server’s request for data. Sample data was loaded to test basic queries and the admin flag was tested by trying to use the insert/update/delete command as a guest and as an admin. The login system involved testing the hash function to check if the hashed input password matches the stored hashed password of the admin login. Queries were sent correctly and incorrectly to see if the database would appropriately accept and return a query or reject it.

**Any potential improvements**

Potential improvements to the database application could be adding a more visually pleasing GUI. With a more complex GUI the user could be able to visualize the data requested a lot better or export it with ease. On the server side, more commands and functionality in commands would also be a potential improvement to the application to accommodate for expandable features we add in the future. Adding a more detailed logging feature and better error handling is also a potential improvement.

**Project Postmortem**:

Some problems that came across in this project on the database application side was lack of knowledge on how to create a client and a server. Once this problem was solved we would have to tackle the problem of having the server communicate to the database using the appropriate functions, in this case with MySQL++ library functions. Learning how to code a server to handle client inputs through TCP and then handling the client requests to obtain data from the mysql database was a difficult task that involved looking up tutorials and documentation on functions. For the actual database, we ran into the issue of using composite attributes in MySQL because it is not directly implemented, to work around it we instead added the attributes into the entity as individual attributes separate from each other as was suggested online. This however made the DB cluttered/messy so we removed the composite attribute altogether(as reflected in ERD) because it was unnecessary anyways.. Another issue on the DB side was how we were going to handle storing passwords/password hashing so that we knew what value to list password as in the DB table for user. We decided to use sha256 hashing so we used a char value with a length of 64 in the table.