DAT602 – Milestone 2

Mark Christison

Nelson Marlborough Institute of Technology, New Zealand

# Contents

[Contents 2](#_Toc42532639)

[Introduction 4](#_Toc42532640)

[SQL 5](#_Toc42532641)

[DDL 5](#_Toc42532642)

[Inserted Test data 5](#_Toc42532643)

[Update Statements 6](#_Toc42532644)

[Delete Statements 6](#_Toc42532645)

[Procedures/Transactions 6](#_Toc42532646)

[Register a User 6](#_Toc42532647)

[User Log in 6](#_Toc42532648)

[User Log off 7](#_Toc42532649)

[Edit User 7](#_Toc42532650)

[Delete User 7](#_Toc42532651)

[User Creates Character 7](#_Toc42532652)

[User Deletes Character 8](#_Toc42532653)

[Select Character to play game 8](#_Toc42532654)

[Change character to play game with 8](#_Toc42532655)

[Online Characters 8](#_Toc42532656)

[Create Game 9](#_Toc42532657)

[Leave Game 9](#_Toc42532658)

[C# Application 10](#_Toc42532659)

[Support for multi-player game play 11](#_Toc42532660)

[Atomicity 11](#_Toc42532661)

[Consistency 12](#_Toc42532662)

[Isolation 12](#_Toc42532663)

[Durability 12](#_Toc42532664)

[InnoDB 12](#_Toc42532665)

[Conclusion 14](#_Toc42532666)

[References 15](#_Toc42532667)

# Introduction

This document carries on the work done previously in Milestone 1 where the business rules, storyboards and an outline for the procedures need to be created where laid out. This milestone revolved around completing the SQL required for making the database schema and the procedures needed for the game play. To ensure the correctness of the database and functionality of the build, a set of test data has been included in the SQL. This serves functionally as it provides the initial game dataset.

After the creation of the database was completed, a C# console application was created to interface with the database and test its functionality. The app has been written to accept dynamic input as to allow for complete testing of the database and to catch any and all possible errors in the code.

Additionally, as transactions are being used to interact with the database, note has been taken to ensure that those transactions have atomicity, consistency, isolation, durability (ACID) in the case of any errors or disasters occurring to the database.

# SQL

## DDL

In creating the procedures needed for the usages scenarios I have realised that the database requires a few changes. Most of the changes have happened to the primary keys on several tables DDL. While initially manually incrementing each data row that was entered to the table was fine, I knew that this needed to be smarter for adding extra data especially in game play. While it was an initial consideration to give these tables an ID column that was an integer, I have instead opted of more natural keys in the data.

The user table has a natural key in the username. Understandably, this might slow down the database if there were to be a large number of rows in the table, however, the size of the database for this problem, it should not grow large enough for that to become an issue. However, I do note it as a potential issue in the future that may require changes.

As with the user table, the skills table, item table, mine table, map table, and character table all have used the attribute that “names” them as a primary key.

An auto incremented integer primary key has been added to the chat table as there was no other logical way to reference every row from the data in the table already.

The other changes to the DDL include the addition of “On update cascade” and “on delete cascade” to the foreign key columns in several tables. This is to support the changes that could occur to the data across the database and to not orphan any data. An example of this is in table character on the column username. As one of the usage scenario’s is for a user changing their name, if the username column was not updated when the primary key in the user table was, then the data rows with the old user name would no longer reference the new user and all old data would be orphaned. Likewise, if a user deletes their account, all the characters that user had will be deleted as they are no longer needed by the database.

## Inserted Test data

To meet the new DDL, the test data has been updated. New test data has been included as well to more accurately reflect the game that is being created. All the concepts from the draft in milestone 1 have been fleshed out in the database. All game assets have been created and a set of different maps has been created.

The inserted data ensures that the tables still have the initial integrity that they had during the tests carried out for milestone 1.

## Update Statements

The initial update statements where intended as tests only and have therefore been commented out. The newly created procedures/transactions now encompass all the required updates for the game.

## Delete Statements

The delete statements have been removed as the created database now has a functional set of data and the deletion of the data would disable some of the functionality, such as the administrator procedures. As with the update statements, these have been commented out.

## Procedures/Transactions

The original usage scenarios as defined in milestone 1 have provided a template for constructing the procedures. Notable, several usage scenarios have been reduced from 2 or 3 scenarios to 1 procedure. As well, other newly discovered scenarios have been created into procedures that the game will require. Some of the scenarios have been split into 2 procedures to allow for user input as needed.

These transactions are similar in a sense to use cases. As such, while the do not follow the normal structure of a use case description, I will only talk about the success case for each scenario.

## Register a User

SQL Lines 400-424.

This procedure checks to see if the username or email passed as parameters already exist in the database. If not then this procedure inserts the parameters into the user table. It also catches exception cases where a username or email are already in the database.

## User Log in

SQL Lines 428-480.

This procedure initially sets variables locally from the database which match the same data row from the user table as the username parameter.

If there is a matching user in the database as the parameter, the password passed as a parameter is checked against the username data row. If that is the same, then a check is made on the Login in attempts count and the locked status of the user. If the user has less that 5 login attempts and the user is unlocked and their password matches the stored password, then they are logged in. Their online status is set to true.

The user locked out scenario from milestone 1 has been amalgamated into this procedure. Each failed log in attempt with a username that exists in the data base increments a counter. Once that counter hits 5 or more then the user row updates on the column is locked and the user is unable to log in, even if on future attempts their password is correct. An administrator must unlock their account.

## User Log off

SQL lines 484-503

This procedure can only be access if a user is already logged in from the app. It updates the user table and sets the online status to false.

## Edit User

SQL lines 507 – 542

This procedure checks if the current username parameter exists in the database, if it does it then checks if the new username parameter does not exist in the database. If it does not exist it then checks that the email parameter does not exists in the database. If all the checks return true, then the user table is updated on the row matching the username with all the input parameters.

## Delete User

SQL lines 546 - 567

If the username passed as a parameter exists in the user table then that row is deleted. Notably, this procedure can only be accessed once a user has logged into their account, thus not requiring other parameters as checks.

## User Creates Character

SQL lines 571 – 600

If the character name passed as a parameter does not exists in the character table, then an insert is done on the character table. Also, all the skills passed as parameters are inserted into the character skills table.

## User Deletes Character

SQL lines 605 - 649

This scenario is broken into 2 different procedures. This is because it requires input from the user based on data from the data base. The first procedure that is run returns all character names from the character table which match the username of the user.

The user then selects the character name that they want to delete. A check is done to make sure that this character exists and belongs to the player then it is deleted.

## Select Character to play game

SQL lines 653 - 679

This scenario uses the first procedure from the previous scenario. First it gets all the characters that belong to a user from the user table. Then based on the character name that the user selects, updates the character table where the character name matches the parameter character name (the one the user selected).

## Change character to play game with

SQL lines 683 - 709

This procedure is a new procedure that I have identified that adds extra functionality. A user can change their character instead of being forced to play only with the same character once logged in.

When this procedure is run, the database updates the character table and sets the active status of the character name matching the parameter character to false.

## Online Characters

SQL lines 713 - 730

This is another new procedure. This will show all the online characters that can be challenged to a game. This procedure will be used solely for the purposes of the GUI version of the game.

This procedure returns a set of character names who have the is active column set to true.

## Create Game

SQL lines 734 – 874

This scenario encompasses several of the originally identified scenario’s, game generates items, game generates mines, and create game.

This scenario requires 3 procedures. The first gets the online characters whose character name does not match the character name of the current character.

The second procedure gets all the maps that are available for play.

The third procedure creates the game. Based on the choice of map from the user, local variables are set to the user’s choice of map. Then the database checks to see if either the current character or the character chosen to play against are already playing a game. If they are not, then the game can be made. Both characters get inserts that create rows on the character map table where the character name and map name are the same as the parameter. The database then “spawns” the items and mines inside the game, on tiles that that map has. Check are made to ensure that mines do not spawn on the same tiles as items. Characters are placed on the home tile of the map. Inserts are made to the character tile table, putting both characters on the home tile of the map.

## Leave Game

SQL lines 789 – 926

Refers to storyboard 9. During game play a character can leave the game, saving the state of the character in the game and returning to the game menu.

After attending class on 8 June 2020 I realise that this might be too much for the requirements for this assignment. As such, I have concluded to not complete the logic for the rest of the procedures. However, they do follow similar lines of thinking.

# C# Application

The application to interface with the data base has been written in C#. The choice for this is because C# is an incredibly slick and modern language. The application has been made to accept dynamic input from the user through a range of menus. These menus can be navigated through using the command line(CLI).

By using an application that is dynamic, the CLI application more accurately represents what will be needed from the application from the 3rd milestone. As nothing is hard coded into the main class, testing can be more in-depth and realistic data can be added to the database through the app. As well, it more accurately represents how the GUI version of the game will be interacting with the game. Also, just more generally, it feels simply better not hard code in tests, as that what you are only testing what is hard coded in. Its also harder then to catch errors or exception cases.

Along with the standard packages that come with a C# console application, I am using the MySQL Data package that has been included from the NuGet package manager. This package provides methods for creating connections to a MySQL database and other useful methods for interacting with a MySQL.

The application has 2 classes. The main class contains all the input menus which the user interacts with. I would have liked to split this off into different classes if I had the time, however this has not happened. The second class is the test class which contains all the methods for making calls to the connected MySQL database.

The biggest benefit of breaking up the program into several different classes is that it allows code reuse. The test class will be able to be reused in milestone 3 for the GUI version of the game.

# Support for multi-player game play

To ensure that the database can support handling many requests from different users at or near the same time, all of the procedures have been structured as transactions. Transactions in SQL have 4 main properties that ensure that the database will work correctly even in the event of a power failure or other fault event or failure.

There are 3 main types of failure that can occur. A transaction failure is caused when there is a improper input to the database or a violation of consistency occurs. A system failure is when either the host or server connected to the database suffer a critical error and the operating system crashes. A media failure is when either the primary or secondary storage of

These 4 properties are knowing by the acronym ACID and are:

* Atomicity
* Consistency
* Isolation
* Durability

## Atomicity

Inside each transaction there can be many statements. If each of these statements where to run normally, they would execute sequentially for example selecting data, inserting new data, updating a table. If on the third query there was an error in the database, then only that query would not execute correctly.

However, Atomicity ensure that in all transactions it is either all or nothing. All transactions inside the query must execute without error or warning (or else exit with a handler for this conditions) or else the transaction will not be parsed.

In MySQL, this is handled by the adjusting the auto commit setting. Normally, this setting is true, meaning that every query is executed straight away. However, when a transaction is started, the auto commit setting is set to false. Only the Commit statement makes all the changes in the transaction permanent.

If the transaction encounters an error, the transaction can rollback to the previous state of the database before the transaction started. This cancels the changes that are made in the transaction.

## Consistency

Consistency ensures that the database is taken from a known and valid state, to another valid state. Transactions must only effect the data stored in allowable ways. If invalid data was to be entered in the database ie not following a constraint, then there would be issues in the child or parent tables of the rows entered in. Ensuring that data is input correctly into the database allows the same or other transactions or queries to be executed by the database engine without incurring fault.

If for whatever reason, there was data the was entered that was invalid then the database again would rollback to a consistent state.

## Isolation

Isolation ensures that only one transaction can occur at once. If 2 or more transaction where to try and interact with the data in a database, one transaction might delete or updated the data the another had already changed or deleted. To resolve this, the database management system will queue the transactions so that are preformed one after another. That way, only one transaction can interact with the database at once. However, to the transaction, it seems as though it is interacting with the database as it normally would as though no other transactions or users where also interacting with the database.

## Durability

Durability ensures that after the transaction has been committed, the changes made by the queries remain. While the database will use volatile memory in ram for temporary storage, durability enforces that once transactions are completed the changes are stored to secondary storage.

### InnoDB

The database engine that MySQL uses called InnoDB ensures that all transactions follow the ACID model. It includes features like Commit which ends a transaction, and rollback which undoes all the changes made by the transaction. It also includes a crash recovery feature. When a crash occurs, any queries executed outside of a made outside of a transaction are completed and saved. Any query executed inside of a transaction that does not commit is rolled back.

For this milestone, the database uses transactions that are managed by InnoDB for all user input queries that are executed. As such, the database engine enforces ACID on all the transactions that take place.

# Conclusion

This assignment was possibly one of the hardest yet most rewarding pieces of work that I have ever done. The structuring of first the SQL, then test the SQL, to writing the application that interfaces with the built database, to testing of the database was an elegant process to perform.

All the procedures required for the game to run have been built. I suspect that I may identify more procedures the need to be built for the creation of the GUI version of the game as required in milestone 3, however I am quite confident that I have captured most of the requirements for the full game.

The application in C# works effectively accepting dynamic input and allowed me to test it effectively after everything was made.

# References

MySQL 8.0 Reference Manual :: 15.1.1 Benefits of Using InnoDB Tables. (n.d.). Retrieved June 1, 2020, from https://dev.mysql.com/doc/refman/8.0/en/innodb-benefits.html

MySQL 8.0 Reference Manual :: 15.2 InnoDB and the ACID Model. (n.d.). Retrieved June 03, 2020, from https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html

Ian. (2017, October 08). What does ACID mean in Database Systems? Retrieved May, 2020, from https://database.guide/what-is-acid-in-databases/

MySql.Data 8.0.20. (n.d.). Retrieved June, 2020, from https://www.nuget.org/packages/MySql.Data