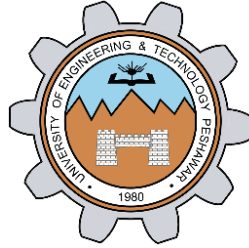


Project Report



Spring 2024

CSE-403 Database Management System

Submitted by:

Ali Asghar(21PWCSE2059)

Shahzad Bangash(21PWCSE1980)

Suleman Shah(21PWCSE1983)

Class Section: **C**

Submitted to:

Dr. Sadeeq Jan

Date:

25th June 2024

Department of Computer Systems Engineering
University of Engineering and Technology, Peshawar

Shooting Game Architecture with MySQL

Contents

1	Introduction	3
1.1	Project Overview	3
1.2	Keywords	3
1.3	Objectives	3
1.4	Softwares/Frameworks Used	4
1.5	Database Management System	4
1.5.1	Types of DBMS	4
1.5.2	RDBMS	4
1.5.3	NoSQL	4
1.5.4	MySQL	5
1.5.5	Database System and DBMS Workflow	5
1.6	Unity	6
1.6.1	Asset Workflow Overview of Unity	6
1.7	Odin	7
1.8	Draw.IO	8
1.8.1	Role of Draw.IO in this project	8
2	Logical Database Design	9
2.1	Business Rules	9
2.2	Conceptual Schema	10
2.3	Relational Schema	10
2.3.1	Relational Schema Sketch	10
3	Physical Database Design	12
3.1	Physical Database Structure	12
3.2	SQL Queries for Tables Creation	13
3.2.1	Achievement	13
3.2.2	Boss	13

3.2.3	Checkpoint	13
3.2.4	Enemy AI	13
3.2.5	Game Character	14
3.2.6	Item	14
3.2.7	Level	14
3.2.8	Level Item	14
3.2.9	Level Played	15
3.2.10	Pickup	15
3.2.11	Player	15
3.2.12	Quest	15
3.2.13	Store	16
3.2.14	Unlock Achievement Certificate	16
3.2.15	Weapon	16
4	Integrating MySQL with Unity	17
4.1	Introduction	17
4.2	Setting Up MySQL Connector/NET in Unity	17
4.3	Database Connection Code - MySQLDatabase class	18
4.4	Serializing and displaying the data	18
4.5	Role of Odin	18
5	Conclusion	21
5.1	Final Output	21
5.2	Future Work	22

Chapter 1

Introduction

1.1 Project Overview

This project involves integrating a Unity game with a MySQL database. A standard method for creating a game has always been a big challenge faced by game developers. In this project, we aim to create a standard game architecture method for creating a shooting game using Unity as Game Engine and MySQL as backend database for our game.

1.2 Keywords

MySQL, Unity, Odin, Database, Operating System(OS), DBMS, Relational Schema, Conceptual Schema.

1.3 Objectives

- Make a standardized shooting game architecture.
- Create a MySQL database for our game.
- Connect Unity with a MySQL database.
- Create a custom editor window to display and manage data inside Unity.

1.4 Softwares/Frameworks Used

- MySQL.
- Unity.
- Odin.
- Draw.IO.

1.5 Database Management System

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database[2].

1.5.1 Types of DBMS

DBMS can be classified into two types: Relational Database Management System (RDBMS) and Non-Relational Database Management System (NoSQL or Non-SQL)

1.5.2 RDBMS

Data is organized in the form of tables and each table has a set of rows and columns. The data are related to each other through primary and foreign keys.

1.5.3 NoSQL

Data is organized in the form of key-value pairs, documents, graphs, or column-based. These are designed to handle large-scale, high-performance scenarios.

1.5.4 MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. It is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons [1].

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

1.5.5 Database System and DBMS Workflow

The general workflow of a database system is that queries are passed to a DBMS which interpret it. After successful execution of a query, OS(Operating System) interacts with actual database and retrieves/add/delete data from database as shown in 1.1. Below is the more refined workflow of a database system.

Refined Workflow is:

- User/Application \rightarrow DBMS
- DBMS interprets and optimizes the query.
- DBMS requests the OS for data access.
- OS reads/writes data from/to disk.
- OS \rightarrow DBMS
- DBMS processes the data and returns results to User/Application.

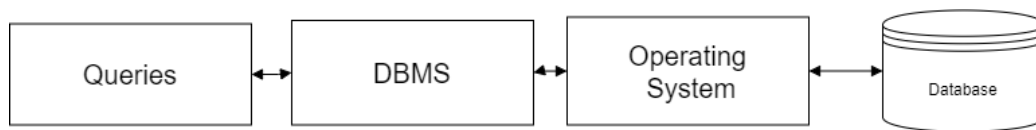


Figure 1.1: Database System Workflow

1.6 Unity

Unity is a cross-platform game engine initially released by Unity Technologies, in 2005. The focus of Unity lies in the development of both 2D and 3D games and interactive content. Unity now supports over 20 different target platforms for deploying, while its most popular platforms are the PC, Android and iOS systems[4].

Unity features a complete toolkit for designing and building games, including interfaces for graphics, audio, and level-building tools, requiring minimal use of external programs to work on projects.

1.6.1 Asset Workflow Overview of Unity

Importing assets into Unity involves placing files into the Assets folder, which the Unity Editor processes for use in projects. Key steps include adjusting import settings, understanding meta files, and utilizing the Asset Database. After importing, you can create your game by organizing assets into Scenes as GameObjects, scripting interactions, and optimizing downloads using the

Addressables system. Building the project involves exporting it to platform-specific binary files and possibly using build automation. Distribution depends on the target platform and can be facilitated by Unity's Cloud Content Delivery service. Finally, the loading process for users is determined by the project's programming and asset bundling strategy, enabling efficient initial downloads and continuous updates[3]. All this can be seen in figure1.2.

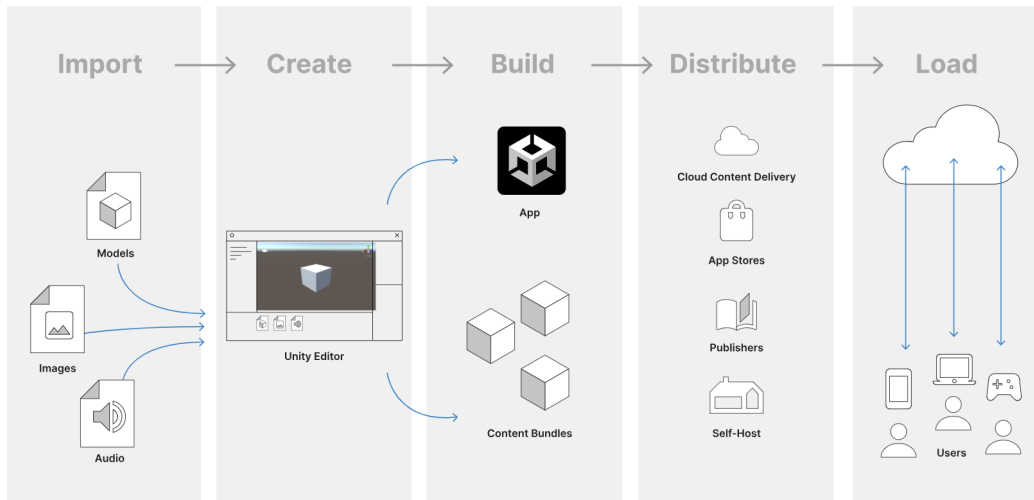


Figure 1.2: Asset Workflow Overview

1.7 Odin

Odin Inspector is a plugin for Unity that lets you enjoy all the workflow benefits of having a powerful, customized and user-friendly editor, without ever having to write a single line of custom editor code. Instead of writing and maintaining thousands of lines of custom editor code, developers can annotate their data structures with our 100+ building block attributes to quickly and easily build user-friendly editors for their entire team. Odin is additionally packed to the brim with a broad suite of editor utilities for many common tasks like creating custom editor windows and much, much more[5].

1.8 Draw.IO

draw.io, also known as Diagrams.net, is a customized tool for creating diagrams and charts. It offers existing automated layout options, or the user can design a custom layout for their preferences. The tool provides a wide range of shapes and hundreds of graphic components to let you create unique diagrams. Depending on your needs, draw.io can keep saved charts in the cloud or in network storage at a data center. It works both offline and on mobile devices. The service does not keep any of your data or claim ownership of your creations, ensuring that you are safe regardless of where or how you work[7].

1.8.1 Role of Draw.IO in this project

In this project, draw.io is used for making diagrams for conceptual and relational schema. Draw.io is a great tool and saves a lot of time when making a schema for a database.

Chapter 2

Logical Database Design

2.1 Business Rules

Following business rules were followed during the database design.

- A player can participate in multiple quests.
- A player can achieve multiple achievements.
- Stores contain multiple characters and weapons..
- Characters are associated with a store and have a price.
- Each quest can be assigned to multiple players.
- Each quest provides a reward and has a specific type.
- A player can play one or many levels.
- Each level can have multiple associated bosses and items.
- A player can unlock multiple achievements.
- The Unlock Achievement Certificate table tracks which achievements have been unlocked by which players.
- Bosses are associated with a specific level and each boss has unique abilities.
- Items can be classified as different types (e.g., Enemy AI, Checkpoint, Pickup).

2.2 Conceptual Schema

The database schema includes tables for players, quests, characters, stores, and other game entities. The schema is designed to handle various relationships between these entities.

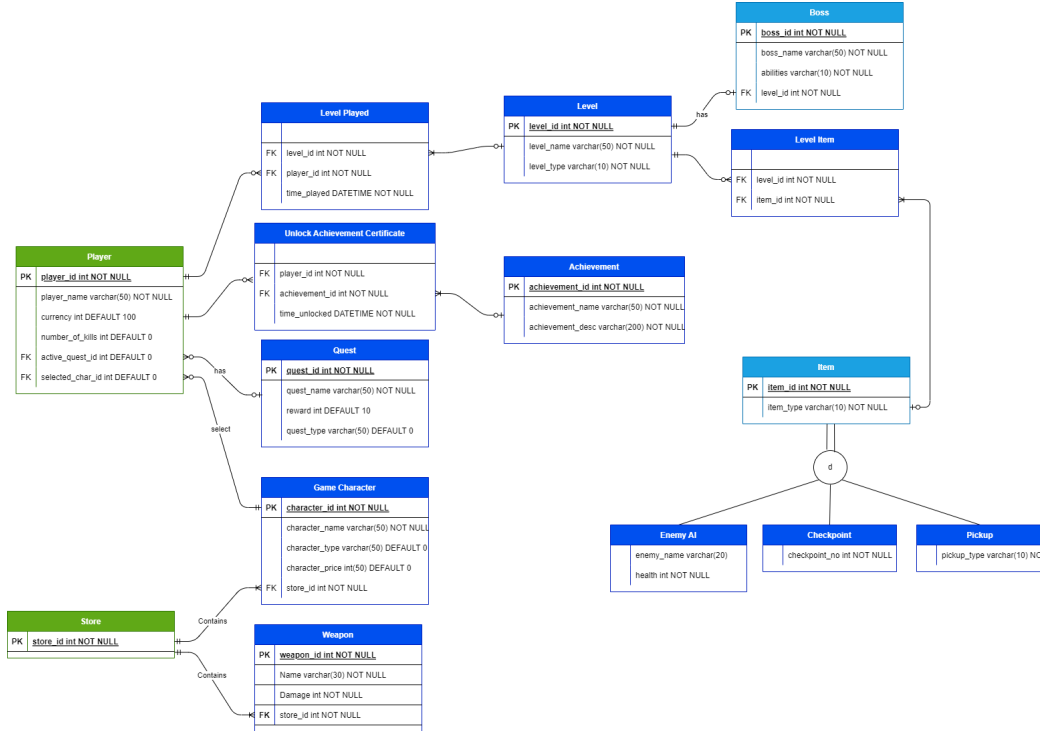


Figure 2.1: Database Conceptual Schema

2.3 Relational Schema

The following normalized relational schema is obtained from the conceptual schema from previous section.

2.3.1 Relational Schema Sketch

Sketch of Relational Schema is given below.

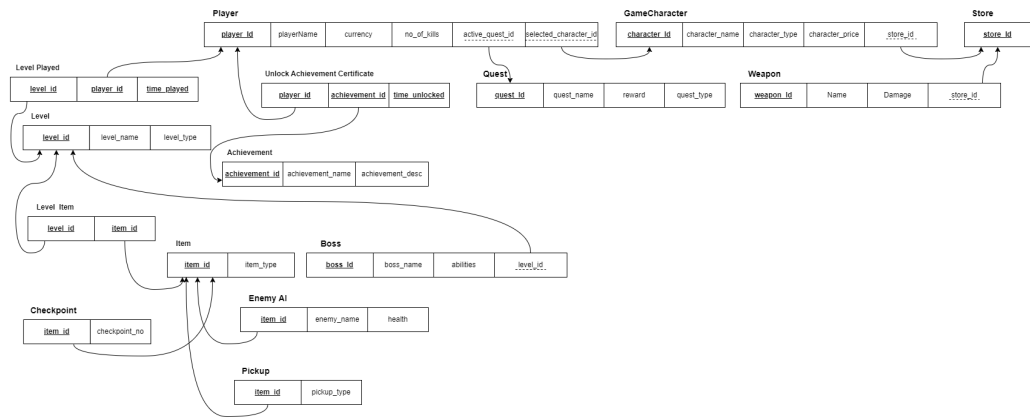


Figure 2.2: Database Relational Schema

Chapter 3

Physical Database Design

3.1 Physical Database Structure

After running successful SQL queries, following tables we created in the database. These tables are physically stored in the memory of the computer as shown in 1.1.

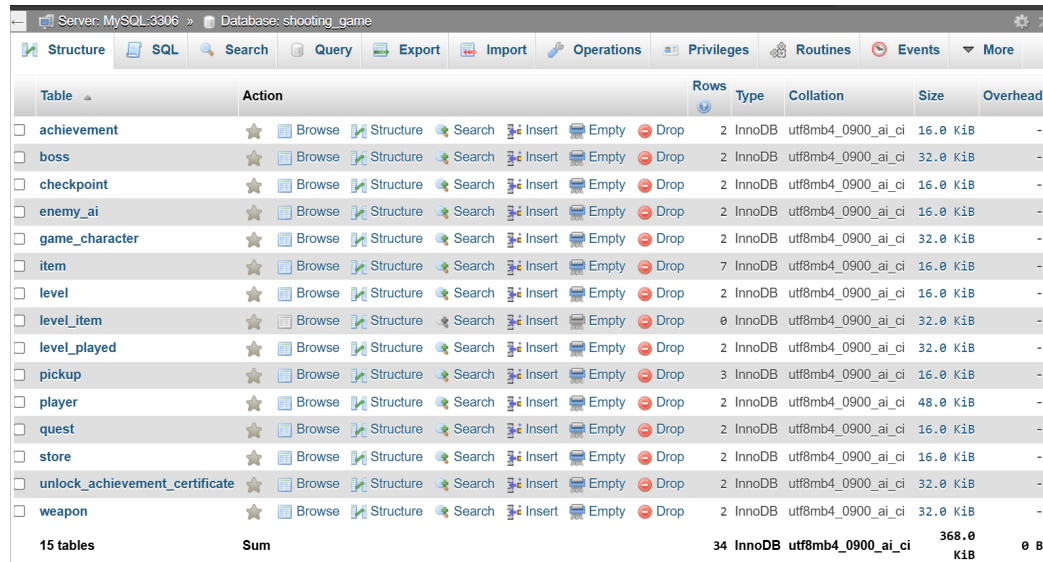


Table	Action	Rows	Type	Collation	Size	Overhead
achievement	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
boss	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
checkpoint	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
enemy_ai	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
game_character	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
item	★ Browse Structure Search Insert Empty Drop	7	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
level	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
level_item	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
level_played	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
pickup	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
player	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	48.0 KiB	-
quest	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
store	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
unlock_achievement_certificate	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
weapon	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_0900_ai_ci	32.0 KiB	-
15 tables	Sum	34	InnoDB	utf8mb4_0900_ai_ci	368.0 KiB	0 B

Figure 3.1: Database Physical Structure

3.2 SQL Queries for Tables Creation

Following queries were given to MySQL for tables creation.

3.2.1 Achievement

```
CREATE TABLE achievement(  
    'achievement_id' int NOT NULL,  
    'achievement_name' varchar(50) NOT NULL,  
    'achievement_desc' varchar(200) NOT NULL,  
    PRIMARY KEY ('achievement_id'));
```

3.2.2 Boss

```
CREATE TABLE boss(  
    'boss_id' int NOT NULL,  
    'boss_name' varchar(50) NOT NULL,  
    'abilities' varchar(50) NOT NULL,  
    'level_id' int NOT NULL,  
    PRIMARY KEY ('boss_id'),  
    KEY 'level_id' ('level_id'));
```

3.2.3 Checkpoint

```
CREATE TABLE checkpoint(  
    'item_id' int NOT NULL,  
    'checkpoint_no' int NOT NULL,  
    PRIMARY KEY ('item_id'));
```

3.2.4 Enemy AI

```
CREATE TABLE enemy_ai(  
    'item_id' int NOT NULL,  
    'health' int NOT NULL,  
    'enemy_name' varchar(20) DEFAULT NULL,
```

```
PRIMARY KEY ('item_id'));
```

3.2.5 Game Character

```
CREATE TABLE game_character(  
  'character_id' int NOT NULL,  
  'character_name' varchar(50) NOT NULL,  
  'character_type' varchar(50) DEFAULT NULL,  
  'character_price' int DEFAULT '0',  
  'store_id' int NOT NULL,  
  PRIMARY KEY ('character_id'),  
  KEY 'store_id' ('store_id'));
```

3.2.6 Item

```
CREATE TABLE item(  
  'item_id' int NOT NULL,  
  'item_type' varchar(10) NOT NULL,  
  PRIMARY KEY ('item_id'));
```

3.2.7 Level

```
CREATE TABLE level(  
  'level_id' int NOT NULL,  
  'level_name' varchar(50) NOT NULL,  
  'level_type' varchar(10) NOT NULL,  
  PRIMARY KEY ('level_id'));
```

3.2.8 Level Item

```
CREATE TABLE level_item(  
  'level_id' int NOT NULL,  
  'item_id' int NOT NULL,  
  PRIMARY KEY ('level_id', 'item_id'),  
  KEY 'item_id' ('item_id'));
```


3.2.9 Level Played

```
CREATE TABLE level_played(  
    'level_id' int NOT NULL,  
    'player_id' int NOT NULL,  
    'time_played' datetime DEFAULT NULL,  
    PRIMARY KEY ('level_id','player_id'),  
    KEY 'player_id' ('player_id'));
```

3.2.10 Pickup

```
CREATE TABLE pickup(  
    'item_id' int NOT NULL,  
    'pickup_type' varchar(20) DEFAULT NULL,  
    PRIMARY KEY ('item_id'));
```

3.2.11 Player

```
CREATE TABLE player(  
    'player_id' int NOT NULL,  
    'player_name' varchar(50) NOT NULL,  
    'currency' int DEFAULT '100',  
    'number_of_kills' int DEFAULT '0',  
    'active_quest_id' int DEFAULT '0',  
    'selected_char_id' int DEFAULT '0',  
    PRIMARY KEY ('player_id'),  
    KEY 'active_quest_id' ('active_quest_id'),  
    KEY 'selected_char_id' ('selected_char_id'));
```

3.2.12 Quest

```
CREATE TABLE quest(  
    'quest_id' int NOT NULL,  
    'quest_name' varchar(50) NOT NULL,  
    'reward' int DEFAULT '10',  
    'quest_type' varchar(50) DEFAULT '0',  
    PRIMARY KEY ('quest_id'));
```

3.2.13 Store

```
CREATE TABLE store(  
    'store_id' int NOT NULL,  
    PRIMARY KEY ('store_id'));
```

3.2.14 Unlock Achievement Certificate

```
CREATE TABLE unlock_achievement_certificate(  
    'player_id' int NOT NULL,  
    'achievement_id' int NOT NULL,  
    'time_unlocked' datetime DEFAULT NULL,  
    PRIMARY KEY ('player_id', 'achievement_id'),  
    KEY 'achievement_id' ('achievement_id'));
```

3.2.15 Weapon

```
CREATE TABLE weapon(  
    'weapon_id' int NOT NULL,  
    'name' varchar(30) NOT NULL,  
    'damage' int NOT NULL,  
    'store_id' int DEFAULT NULL,  
    PRIMARY KEY ('weapon_id'),  
    KEY 'store_id' ('store_id'));
```

Chapter 4

Integrating MySQL with Unity

4.1 Introduction

Integrating MySQL with Unity can provide robust backend support for multiplayer games, persistent data storage, and complex data queries. MySQL Connector/NET, a data provider for MySQL, enables Unity to communicate with MySQL databases.

4.2 Setting Up MySQL Connector/NET in Unity

- First of all, Download MySQL Connector/NET
- Download and install the appropriate version for your system.
- Add MySQL.Data.dll to Unity.
- Locate the MySql.Data.dll file from the installation directory (usually found in the assemblies or lib directory of the MySQL Connector installation).
- Copy MySql.Data.dll to the Assets/Plugins folder in your Unity project as shown in figure4.1.

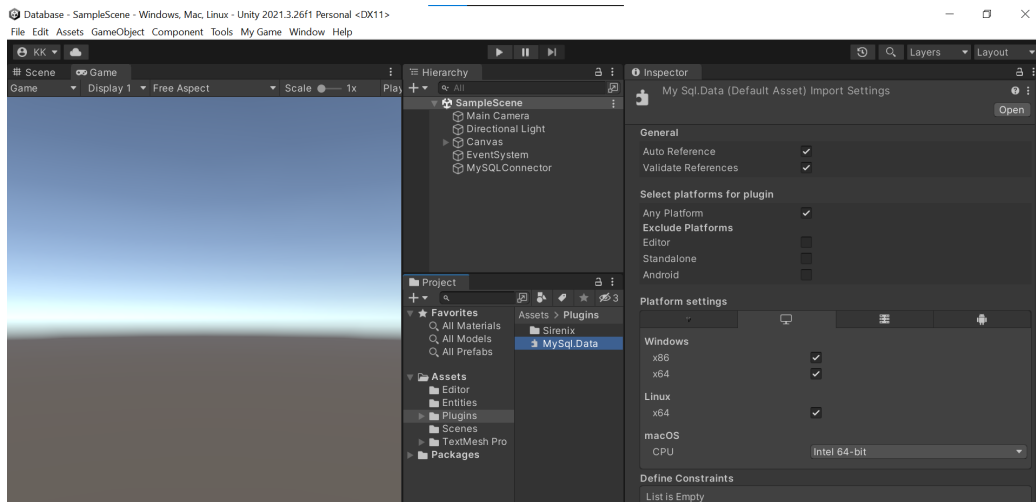


Figure 4.1: MySQL.Data.dll in Unity Plugins Folder for connecting MySQL with Unity

4.3 Database Connection Code - MySQLDatabase class

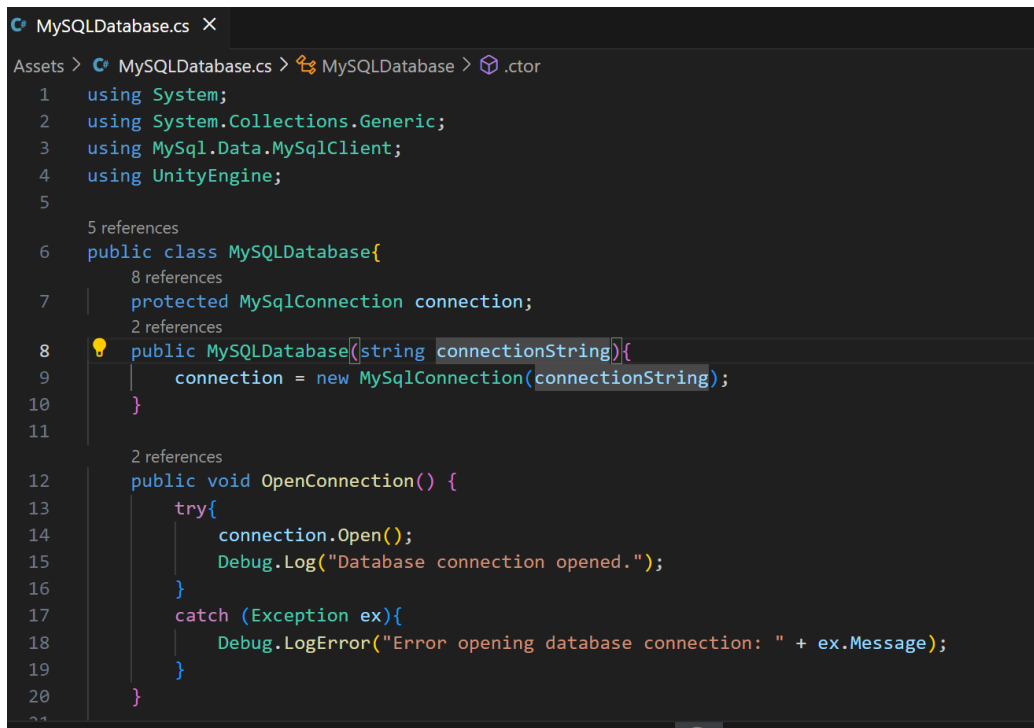
The code shown in figure4.4 shows how can we establish connection with MySQL from Unity c class. The database credentials are given in the string argument of the constructor.

4.4 Serializing and displaying the data

This is the last part of the implementation phase. In order to get data and show it in Unity[6], We have created a C class for each table. This class serializes the data and this data is shown on the editor window of Unity. Figure 4.3 Shows the code of Player class.

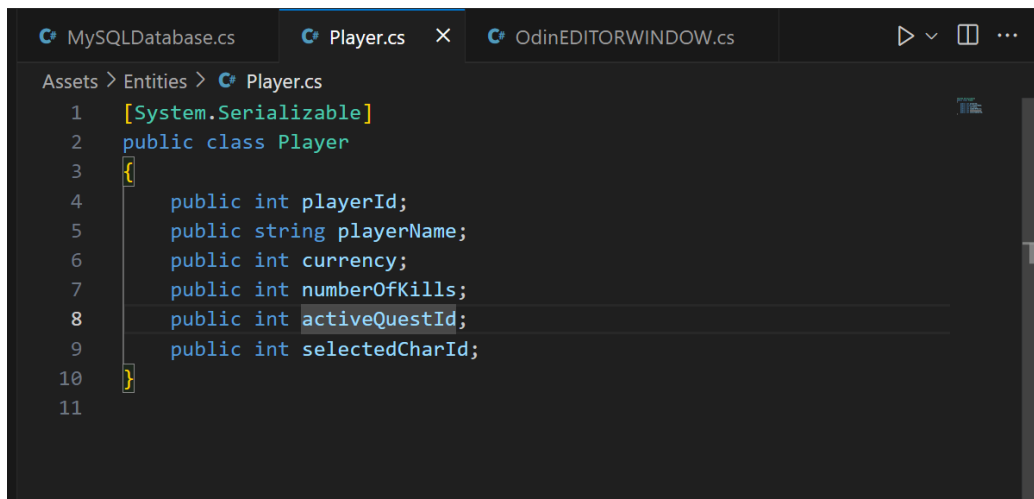
4.5 Role of Odin

Odin is a great tool for making Editor Windows and Serializing data in Unity. The following script demonstrate the use of Odin Library and MySQL-Database class to get data from MySQL and show it on Unity Editor



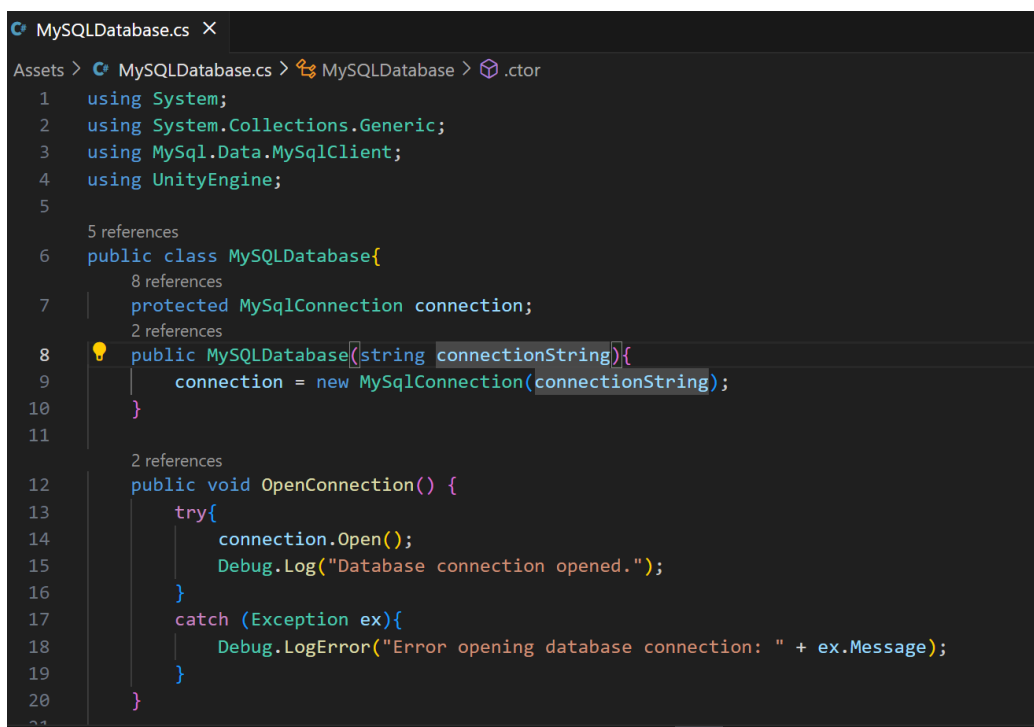
```
MySQLDatabase.cs X
Assets > MySQLDatabase.cs > MySQLDatabase > .ctor
1 using System;
2 using System.Collections.Generic;
3 using MySql.Data.MySqlClient;
4 using UnityEngine;
5
6 5 references
7 public class MySQLDatabase{
8     8 references
9     protected MySqlConnection connection;
10    2 references
11    public MySQLDatabase(string connectionString){
12        connection = new MySqlConnection(connectionString);
13    }
14
15    2 references
16    public void OpenConnection() {
17        try{
18            connection.Open();
19            Debug.Log("Database connection opened.");
20        }
21        catch (Exception ex){
22            Debug.LogError("Error opening database connection: " + ex.Message);
23        }
24    }
25 }
```

Figure 4.2: Database Connection Code - MySQLDatabase class



```
MySQLDatabase.cs Player.cs X OdinEDITORWINDOW.cs
Assets > Entities > Player.cs
1 [System.Serializable]
2 public class Player
3 {
4     public int playerId;
5     public string playerName;
6     public int currency;
7     public int numberOfKills;
8     public int activeQuestId;
9     public int selectedCharId;
10 }
11
```

Figure 4.3: Player class code for data serialization



```

MySQLDatabase.cs X
Assets > MySQLDatabase.cs > MySQLDatabase > .ctor
1  using System;
2  using System.Collections.Generic;
3  using MySql.Data.MySqlClient;
4  using UnityEngine;
5
6  5 references
   public class MySQLDatabase{
7      8 references
       protected MySqlConnection connection;
8      2 references
       public MySQLDatabase(string connectionString){
9          connection = new MySqlConnection(connectionString);
10     }
11
12     2 references
       public void OpenConnection() {
13         try{
14             connection.Open();
15             Debug.Log("Database connection opened.");
16         }
17         catch (Exception ex){
18             Debug.LogError("Error opening database connection: " + ex.Message);
19         }
20     }

```

Figure 4.4: Database Connection Code - MySQLDatabase class

Chapter 5

Conclusion

This project demonstrates how to integrate a Unity game with a MySQL database. The custom editor window provides an easy way to display and manage the data within the Unity Editor.

5.1 Final Output

Following are some of the snapshots of the output of this project.

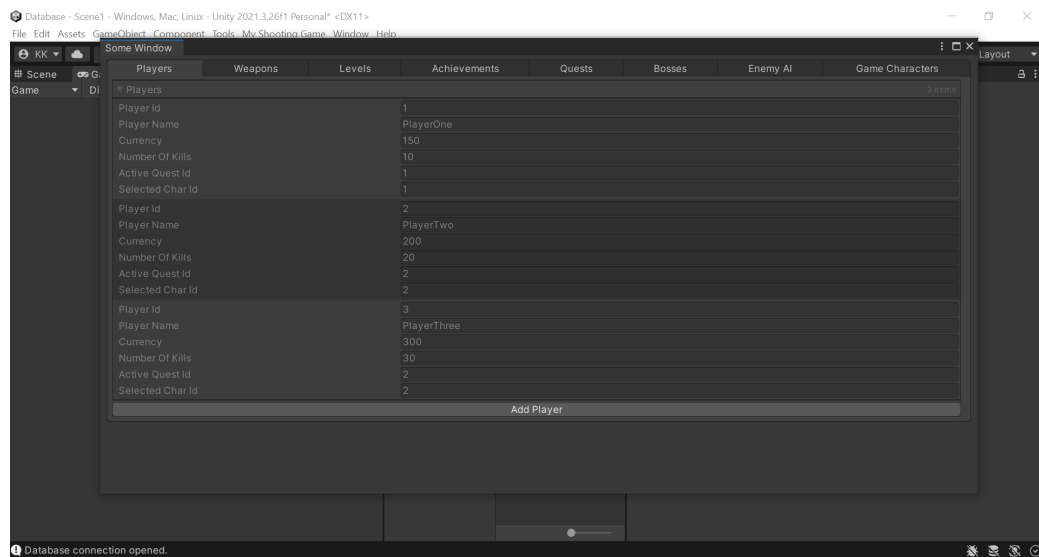


Figure 5.1: Output Snapshot 1

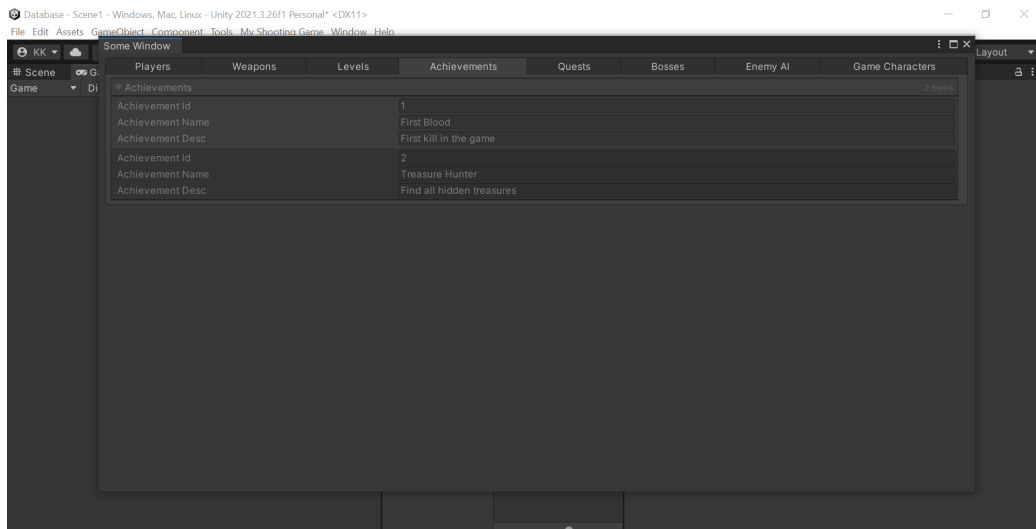


Figure 5.2: Output Snapshot 2

5.2 Future Work

Future work could include enhancing the database operations, adding more features to the custom editor window, and optimizing the performance of the database interactions.

References

- [1] <https://www.tutorialspoint.com/mysql/mysql-introduction.htm>
- [2] <https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1>
- [3] <https://docs.unity3d.com/Manual/AssetWorkflow.html>
- [4] [https://www.tutorialspoint.com/unity/unity_i*ntroduction*.htm](https://www.tutorialspoint.com/unity/unity_introduction.htm)
- [5] <https://odininspector.com>
- [6] <https://docs.unity3d.com/Manual/script-Serialization.html>
- [7] <https://medium.com/@hasaliedirisinghe/user-guide-to-draw-io-6a1a4d7b5d33>