DUNGEONS & DRAGONS

DUNGEON MASTER DATABASE

CHIH YU, ELLEN TURNER, JUN H. PARK, MIDORI WILLIAMS

TABLE OF CONTENTS

INTRODUCTION

Team SQeaL intends to create a Dungeons and Dragons (D&D) database. D&D is a tabletop role-playing fantasy game, in which people gather around a table and go on an adventure with their imagination. This database is designed from the perspective of the game leader, story creator, and narrator: the Dungeon Master.

DATABASE APPLICATION

Every D&D game is run by a Dungeon Master (DM), who creates and explains the story and combat of an adventure including location and monsters that players getting attacked by. Therefore, due to the competence of the DM, the game could be very attractive or could be boring.

Team SQeaL decided to approach a database from the perspective of the DM, where a database can help improve the quality of the game by assisting the DM.

The DM must know and record:

- **Dungeon master store players data:**
  - Each players has a name, id, contact information such as email and phone number
- **Dungeon master creates game Campaign:**
  - The Campaign is an environment of one full story, which contains several episodes. Campaigns contain its name and owner of the campaign, who is the DM. Campaigns will contain a map for each campaign.
  - Campaign has monsters and Non-player-character that can be appear in a campaign
- **Each campaign contains episodes:**
  - Each Episode contains a name, number, and various of location from a map
  - Each Episode can contain specific monsters and non-playable-character for each episode.
- **Each map has various of location:**
  - Location has a name, number.
- **Dungeon master must describe monster:**
  - Monster has a name, number, level or powerfulness of the monster, rewards when it is defeated, and monster type.
- **Dungeon master must describe Non-player-character:**
  - Non-player-character (NPC) has a name and class (job).

Team SQeaL believes that providing a database system, which can store, retrieve, and update the given data above, will increase the efficiency of game play and simplify the DM role.

ENTITY

The database will hold all the information that DM must know for the game D&D. A list of entities and their attributes is provided below. Furthermore, based on these attributes with entity listed on this section, section Database Design will provide Entity Relationship diagram and Relational Model with constraints and keys.

Entity	PLAYER				
Attribute	Name	Player_id	Join_date	Email	Phone_number
Entity	CAMPAIGN				
Attribute	DungeonMaster	Campaign_name	Map_name		
Entity	MAP				
Attribute	Map_name	Map_number			
Entity	NON_PLAYER_CHARACTER				
Attribute	Npc_name	Npc_class			
Entity	EPISODE_NPC				
Attribute	Quest_reward	NPC_ID	Level		
Entity	EPISODE				
Attribute	Campaign_name	Location_name	Episode_name	Episode_num	
Entity	EPISODE_MONSTER				
Attribute	Quest_reward	Monster_id			
Entity	MONSTER				
Attribute	Monster_name	Monster_level	Monster_type	Drop_item	
Entity	CHARACTER				
Attribute	Level	Character_name	Passive_perception		
Entity	ITEM				
Attribute	Value	Item_count	Item_num	Item_name	

POSSIBLE FEATURES OF DATABASE QUERIES

The primary user of team SQeaL's D&D database would be the DM. Therefore, possible database queries would be only for DM.

Data that DM can retrieve, update, and delete through Team SQeaL's Database

- Retrieve/Update/Delete player information: player_id, name, join date, email and phone number.
- Retrieve/Update/Delete player's character information: character name, level, item character owned including item name, item value and total number of items a character owned, and passive perception (Definition of Passive perception: a number each character has, the number represents chances of noticing special events throughout adventure).
- Retrieve/Update/Delete information about the campaign that DM owned. Information would be campaign owner name, campaign name, and map of the entire campaign.
- Retrieve/Update/Delete list of all possible monsters, which include name, level, type, and drop items in a campaign and episodes.

- Retrieve/Update/Delete list of all NPC, which include name, class (job), and reward of complete quest from NPC, if quest is offered from NPC in a campaign and episodes.
- Retrieve/Update/Delete information about points of interest or landmarks of a map, include location name, and location number.

DELIVERABLES SCHEDULE

To Build a successful database for Dungeon Master and working as team, it is crucial to gather the information from each Team SQeaL member. Due to the time limits of this project, Team SQeaL had to schedule weekly meeting and emergency meeting.

General Weekly Meeting
Deliverables will be worked on a weekly basis, **Monday** and **Wednesday** before class, and **Fridays** at 11am. The Group leader (Midori) will structure tasks to be worked on, but it is the responsibility of all team members to come prepared and to have read the Iteration outline in the class notebook. Tasks will be assigned, but in general, everyone will contribute to all parts.

Accomplished Schedule

Date	Purpose/Goal
Jan 09, 2019	● Discussion about Iteration 0. ● Discussion about what is the group project's theme. ● Discussion about choose group leader.
Jan 14, 2019	● Focus the Project theme (Dungeons and Dragon). ● Research about what is D&D.
Jan 16, 2019	● Discussion about possible ideas based on research. ● List the attributes ● Finish the team contract form
Jan 17, 2019	● Person, who worked last, text to group through group message and turn in
Jan 25, 2019	● Weekly meeting start -Every Friday at 11am to 1pm -Library -Discuss about Entity Relationship diagram
Jan 30, 2019	● Complete Entity Diagram ● Modification of Relational Model ● Modification of project proposal
Jan 31, 2019	● Emergency Meeting is Possible ● Finish up the Entity Diagram

	● Relational Model ● Modification of project proposal
Feb 06, 2019	Discussion about Tooling and Big Picture Presentation ● Choose database software Options: -MySQL -SQL Server ● Database hosting Options: -Cloud -Personal -School Server ● Decision of UI (user_interface) Options: -Java -C# -Python -Javascript
Feb 16, 2019	Snow Storm, lost almost 10 days. No further discuss. ● Based on the discussion and Relationship Model, start to create database. ● Start to implement tables based on Relational Model.
Feb 22, 2019	Implement the user interface and connect with database ● Wordpress is not able to function due to the knowledge is beyond our capability. ● Deadline is getting closer, the team decided to complete the GUI parts on local host and connect to server.
Feb 27, 2019	Connect UI to UW student server ● Coding UI with html, css, php, and Javascript. ● PhpMyAdmin can't connect to UW student server.
Mar 6, 2019	PhpMyAdmin connected with Student Server with mysql provided from School. ● Implemented Insert/Update/Delete working with html and php. ● mysql table stored into PhpMyAdmin using myisam engine. ● Completed Foreign key constraint hard coding with php and insert and delete restriction implemented.
Mar 8, 2019	Complete the Iteration 3 Report

	● Require Confirm for final submission to Group Leader.
Mar 11, 2019	Start discuss about Iteration 4 ● Normalization ● Documentation (query statements, UI, and Server side code).
Mar 13, 2019	Task ● Finish Normalization ● Work on the Poster ○ Technical Content ○ Organization ○ Professionalism ○ Creativity ○ Approachable
Mar 15, 2019	Final Confirm Require for Iteration 4 Report ● Submit the Iteration 4 ○ Report ○ Code (query, UI, server side language) ● Finish the Poster ○ Get final confirm by Group Leader
Mar 17, 2019	Submit the Group Poster and Final report

Approximate Schedule for Future Tasks

This schedule is approximate schedule, which can be used to track on the tasks for the future.

- The dates are fixed since the meeting is weekly on **Friday** 11am to 1pm.
- Weekly meeting hours may extend if the progresses of the goal or purpose of the meeting did not meet expectation.
- Purpose and goal of the weekly meeting can be change due to work progression.
- Emergency meeting can be set up due to work progression.

Mar 18, 2019	Final Poster Presentation

Emergency Meeting

Emergency meeting can be set up depending on the progression of the weekly meeting. Emergency meetings will be announced 2 days in advance by the group leader. Since it is not an official meeting, valid excuse of absent is allowed.

Types of Emergency Meeting

- Virtual Meeting: Team will schedule a zoom meeting as needed on **Tuesday** or **Thursday** in order to complete the team assignment.
- Makers Room: Team leader will organize an emergency meeting on **Monday** and **Wednesday** after class as its needed.

Final Submission

- General Case: Iteration submissions will be completed by **Ellen**.

- Exceptional Case: The person who submits the assignment(s) will confirm with the team through text message.

DATABASE DESIGN

The following section covers the design of D&D database using Entity Relationship diagram and Relational Model based on Entity Relationship diagram, with appropriate assumptions of design and constraints.

ENTITY RELATIONSHIP DIAGRAM

The Entity Relationship Diagram (ERD) provides the conceptual design and basic outline for the data models. Team SQeaL designed a D&D ERD (Figure 1), from the perspective of a DM (game designer and storyteller/narrator) to help them manage the details of an ongoing game (campaign split into episodes) in a theoretical real world scenario, like that of a D&D club. All data kept is solely for the benefit of a DM, since there is already adequate support for players of the game wishing to manage their character design and play. The database design relies on the following set of assumptions made of what a DM would find useful in their game building, management, and cataloging, driven by insight from actual players of the game.

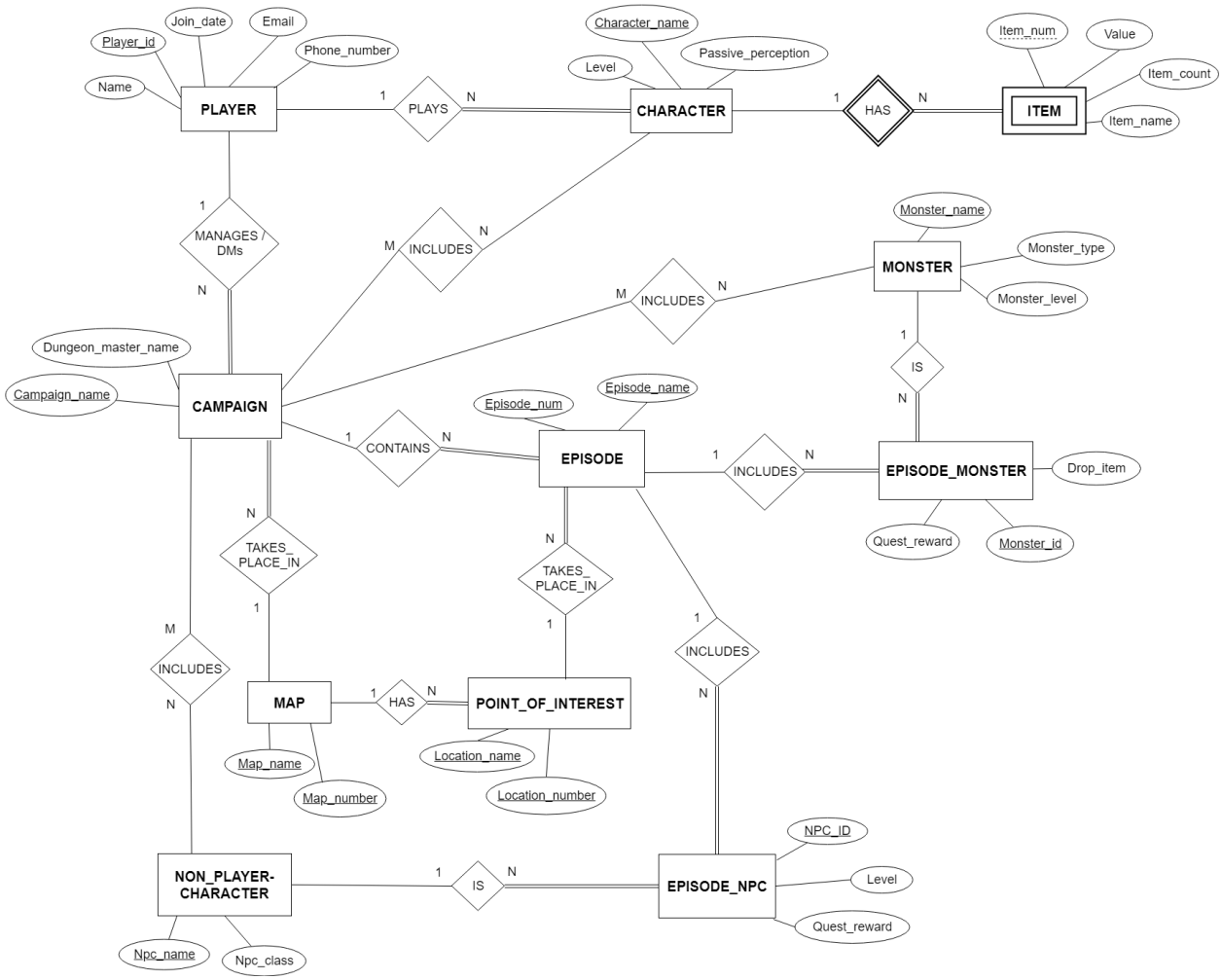


Figure 1: Dungeons and Dragon Entity Relationship Diagram

ASSUMPTION

- A player is any member of this scenario's D&D club.
- A user of this database is any player that wishes to manage a game as a DM.
- A player can be either a DM or a character (player character) in any one particular campaign, but cannot be both under the same campaign instance. A campaign DM does not play a character, but serves as the storyteller and game manager. As the DM of a campaign, they manage data associated with that campaign instance.
- A player may be as many characters as they want across multiple campaigns, but they can only be one character in any one particular campaign. A character can only be assigned to one player.
- A character instance may be involved in multiple campaigns, ie. reused by the owning player.
- A new campaign must have a player assigned to it as the DM and must include a "map" of the world the campaign takes place. A campaign can only have one map.
- A map tuple is a fantasy world that a campaign can take place in. Each map is associated with a set of points of interest (POI) where episodes of the campaign can take place at. A campaign episode cannot take place in POIs that are not part of the campaign's map. Each POI must be associated with a map.
- A map can be used by multiple campaigns for their story setting.
- An episode is a session played by current campaign players (ie, club members get together and the duration of the campaign played becomes an episode instance).
- Each episode is assigned the campaign they are part of and the POI they took place in. Each episode only takes place in one POI, though multiple episodes of the campaign may "revisit" the POI.
- Multiple campaigns may implement their games using the same map and set of POIs, but there is no relation between them.
- A campaign will include monsters and non-player characters (NPCs), from a list of all available monsters and NPCs, to be used in episodes throughout the campaign. For a Monster or NPC to be involved in an episode, they must be associated with the campaign first as part of the "world building." (ea. monsters may include trolls and robots, but the DM only wants trolls added in their current campaign, not robots, so only trolls will "show up" in future episodes).
- Once the DM has added monsters and NPCs to a campaign, they will create instances of them as they "appear" in episodes. For example, a campaign may have NPCs of type "villager"; if, during a certain episode, player characters interact with 4 villagers, then there will be 4 episode NPC tuples associated with that episode. This is because one villager may have different characteristics than another. This is true for all NPCs and monsters. They are separate entities because their attributes, and how player characters interact with them, are different.
- Episode NPCs and episode monsters are only associated with 1 episode, but all campaigns associate with the same master list of monsters and NPCs, resulting in a many-to-many relationship. If a DM wants to add a monster or NPC that is not already in the database, it must first be entered into the list of all available monsters and NPCs.
- During the game, a DM can make up items for a player character to "find" or receive as a gift from a NPC, and decides the item name, quantity and value. An item only belongs to one character.

CONSTRAINTS ENTITY RELATIONSHIP DIAGRAM

The list of constraints is based on Team SQeaL's Entity Relationship diagram. The list of constraints that would be cover on current section are Domain constraint, Entity Constraint, Referential constraint, and Key constraints based on ER diagram with its attributes. **The detail represent of foreign keys and primary keys will be label on on below Relational Model section.**

PLAYER

Player_id	Name	Phone_num	Email	Join_date

- **Player_id:** Integer, two digits, start at 01, Not NULL and duplicated.
- **Name:** String, length range of 20, Start character with Uppercase, Not NULL
- **Phone_num:** Integer digit of 10 digits. (U.S phone_number), Not NULL
- **Email:** varchar length range of 40, Can be NULL
- **Join_date:** String form of MM/DD/YYYY, length range of 10, Not NULL

CHARACTER

Player_id	**Name**	Level	Passive_Perception

- Player_id: Integer, two digits, start at 01, Not NULL and duplicated. Foreign key from PLAYER.
- **Name:** varchar, length range of 20, start character with uppercase, not NULL, primary key.
- Level: Integer, two digits starting at 01, not NULL
- Passive_Perception: Integer, two digits, not NULL

ITEM

Item_num	Item_name	Item_count	Value

- **Item_num:** Integer, three digits, start at 01, not NULL and duplicated, partial key.
- Item_name: String, length range of 20, not NULL
- Item_count: Integer, length range 30, can be NULL
- Value: varchar max length of 6, Tuple (plat, gold, silver, copper)

CAMPAIGN

Campaign_name	Campaign_map_name	Dm_name

- **Campaign_name:** String, length range 20, not NULL and duplicate. Primary key.
- Dm_name: String, length range 20, start character with uppercase, not NULL
- Campaign_map_name: String, length range 20, start character with uppercase, not NULL. Foreign key from MAP.

EPISODE

Campaign_name	**Episode_num**	Location_name

- Campaign_Name**:** String, length range 20, not NULL and duplicate. Foreign key from CAMPAIGN.
- **Episode_num:** Integer, two digits starting with 01, not NULL. Primary key.

- Location_name: String, length range 30, not NULL. Foreign key from POINT_OF_INTEREST.

MAP

Map_num	**Map_name**

- **Map_num:** Integer, start at 01, not NULL and duplicate. Candidate key.
- **Map_name:** String, length range 20, not NULL and duplicate. Primary key.

POINT_OF_INTEREST

Map_name	**Location_num**	**Location_name**

- Map_name: String, length range 20, not NULL and duplicate. Foreign key from MAP.
- **Location_num:** Integer, start at 01, not NULL and duplicate. Candidate key.
- **Location_name:** String, length range 30, not NULL. Primary key.

NON_PLAYER_CHARCTER

Npc_name	Npc_class

- **Npc_name:** String, length range 20, start character uppercase, not NULL and duplicate. Primary Key.
- Npc_class: String, length range 20, start character uppercase, not NULL

EPISODE_NPC

Npc_id	Episode_num	Npc_name	Quest_reward	Npc_level

- **Npc_id:** Integer, two digits starting with 01, not NULL and duplicate. Primary key.
- Episoside_num: Integer, two digits starting with 01, not NULL. Foreign key from EPISODE.
- Npc_name: String, length range 20, start character uppercase, not NULL and duplicate. Foreign key from NON_PLAYER_CHARACTER.
- Quest_reward: String, length range 20, not NULL
- Npc_level: Integer, two digits starting with 01, not NULL

MONSTER

Monster_level	**Monster_name**	Monster_type

- Monster_level**:** Integer, starting with 01, not NULL
- **Monster_name:** String, length range 20, not NULL
- Monster_type: String, length range 20, not NULL

EPISODE_MONSTER

Episode_num	Monster_name	**Monster_id**	Monster_level	Drop_item

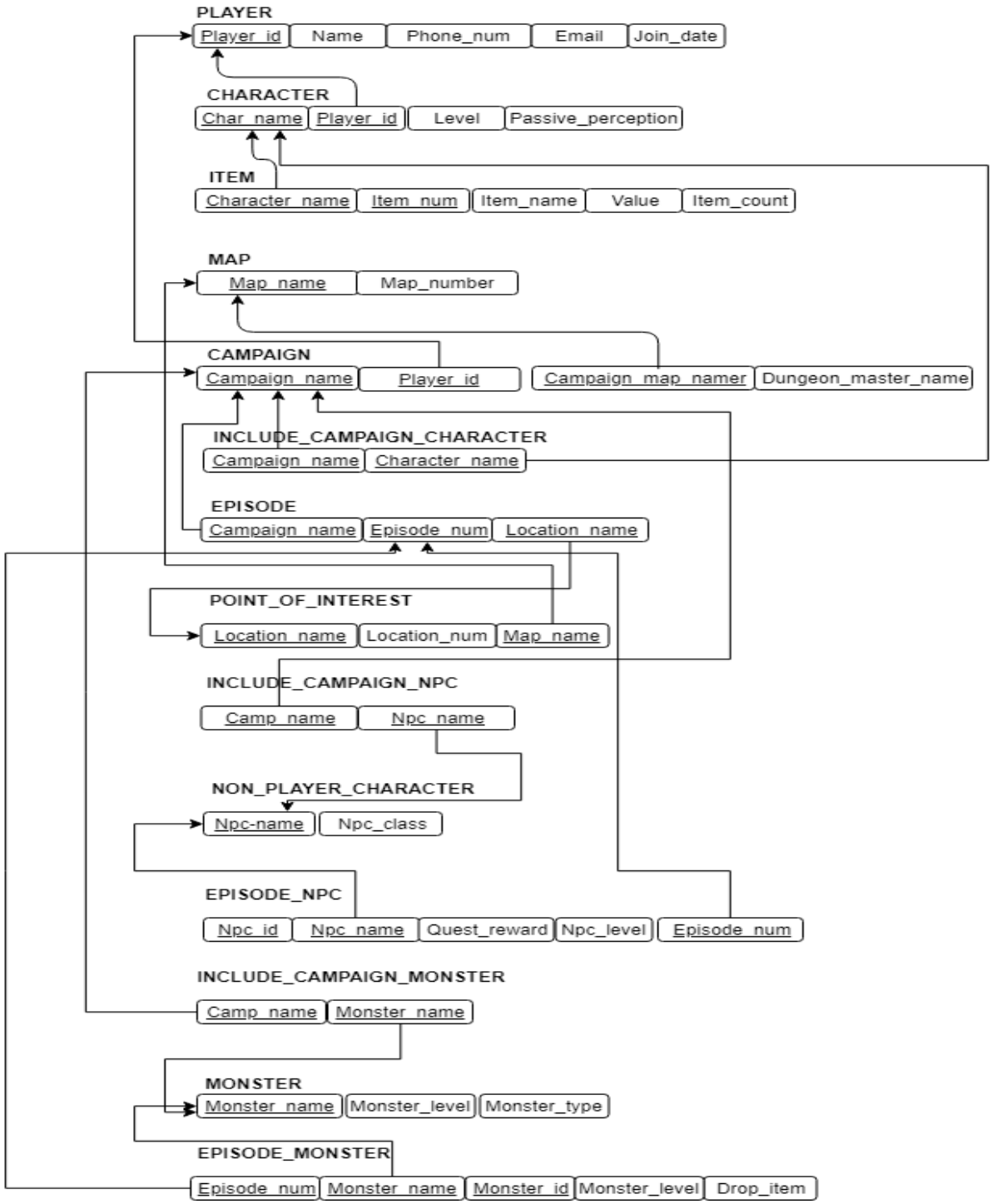
- Episode_num: Integer, two digit starting at 01, not NULL and duplicate. Foreign key from EPISODE.
- Monster_name**:** String, length range 20, not NULL. Foreign key from MONSTER.
- **Monster_id:** Integer, two digit starting at 01, not NULL and duplicate. Primary key.
- Monster_level: Integer, two digits starting at 01, not NULL

- Drop_item: String, length range 20, can be NULL

RELATIONAL MODEL

The Relational Model provides more concrete outline which contains relations, tuples, and attributes. Figure 2, D&D Relational Model will contains all relations, tuples, and attributes as form of table, which derived from Figure 1 with possible foreign keys and primary keys.

Figure 2: D&D Relational Model



PRIMARY KEY AND FOREIGN KEY

Based on the rules and step of "Relational Database Design by ER-to-Relational Mapping", Team SQeaL came out following primary keys and foreign keys for each relations.

RELATION	PRIMARY_KEY	FOREIGN_KEY
PLAYER	Player_id	
CHARACTER	Char_name	Player_id References PLAYER Player_id
ITEM	Item_num and Character_name	Character_name References CHARACTER char_name
MAP	Map_name	
CAMPAIGN	Campaign_name	Player_id ● References PLAYER Player_id Campaign_map_name ● References MAP Map_name
INCLUDE_CAMPAIGN_CHARACTER	● Campaign_name ● Character_name	Campaign_name ● References CAMPAIGN Campaign_name Character_name ● References CHARACTER Char_name
EPISODE	Epsiode_num	Campaign_name ● References CAMPAIGN Campaign_name Location_name ● References POINT_OF_INTEREST Location_name
POINT_OF_INTEREST	Location_name	Map_name ● References MAP Map_name
INCLUDE_CAMPAIGN_NPC	● Camp_name ● Npc_name	Camp_name ● References CAMPAIGN Campaign_name Npc_name ● References NON_PLAYABLE_CHARACTER Npc_name
NON_PLAYER_CHARACTER	Npc_name	
EPISODE_NPC	Npc_id	Npc_name ● References NON_PLAYER_CHARACTER Npc_name Episode_num ● References EPISODE

		Episode_num
INCLUDE_CAMPAIGN_MONSTER	● Camp_name ● Monster_name	Camp_name ● References CAMPAIGN Campaign_name Monster_name ● References MONSTER Monster_name
MONSTER	Monster_name	
EPISODE_MONSTER	Monster_id	Episode_num ● References EPISODE Episode_num Monster_name ● References MONSTER Monster_name

TOOLING

The following preliminary choice of tooling has been selected for the efficiency of implementation, based on the relative skill levels of each team member. Among the various of choices, team SQeaL had decided to chose the implementation tools based on its easiness and amount of available resources for learning the tools.

TOOL	**Team SQeaL choice**	**Reason**
Database Management System	**MySQL** managed through the **phpMyAdmin** administration package	● Free, cost-effective ● Supported through UW IT Connect ● Various resources helping learn about the DBMS. ● A GUI makes learning curve smaller.
Graphical User Interface/Website	**Graphic User Interface** HTML/CSS/JAVASCRIPT **Server Side Language** PHP 8	● Will be explain detail on Section “CREATE DATABASE”.
Hosting Service	UW Shared Web Hosting for students	● Integrates with existing Linux based student accounts that can be accessed through an SSH client ● Specific, detailed tutorials made setup from the Linux command line to linkage of related services an easy process. ● Free

Additional Tools	• IDE: Atom (html,css,js,php) • Google Drive • phpMyAdmin (Easy to import table Written in mySQL file)	• Easy to use • Appropriate IDE handling HTML/CSS/Javascript/PHP

The list of the tools to approach implementation of Dungeons and Dragons database may not be the best choice for people who has great experiences on database, however, team SQeaL is formed with people who are new to SQL through hosting its domain, therefore tools with great references with numerous tutorials was appropriate to team SQeaL.

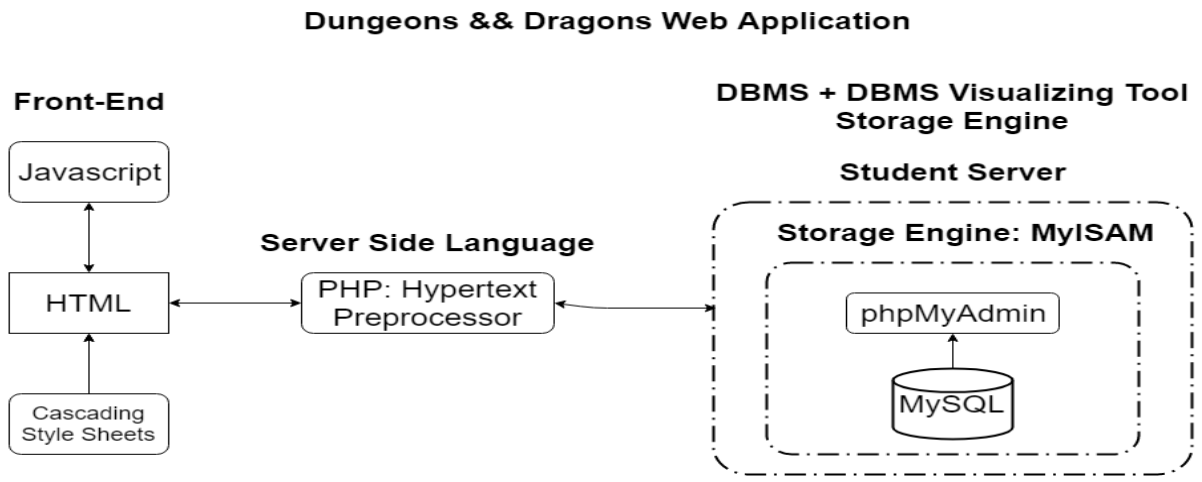
CREATE DATABASE

The initial tool that Team SQeaL had decided to use and implement was Java console application and host it into AWS server. However, discovering the existence of student server with pre-installed MySQL that can be connected to phpMyAdmin with student SSH, and UW-IT Connect provided instruction to connect phpMyAdmin to Front-end service called "Wordpress", the team decided to use "Wordpress" and host it into student server. However, due to complex of customizing wordpress and implement the customized HTML, CSS, Javascript, the team SQeaL decided to build Front-End and Back-End from scratch.

DUNGEONS AND DRAGONS WEB DESIGN

Team SQeaL had decided to build html contains css and interact with javascript for front end. It will be on .php written using Hypertext Preprocessor(computer program that modifies data to conform with the input requirement of another program) that interacts both GUI and database. The data will be store through phpMyAdmin that is connected to student server that runs tables written in mySQL. See Figure 3.1 for basic understand of team SQeaL web application.

Figure 3.1 Dungeons and Dragons Web Application Design



Storage engine that phpMyadmin and mySQL that school provides through SSH is MyISAM, which is different with storage engine InnoDB. Since the storage engine had give huge effect on team SQeaL entire project, the table Figure 3.2 will provide information about compare and Contrast the storage engine.

Figure 3.2: Storage Engine Compare and Contrast Between MyISAM and InnoDB

	MyISAM Storage Engine	**InnoDB Storage Engine**
Referential Integrity	Does not Provide	Provide
Transactions & Atomicity	Does not Support	Supports
Locking	Table-locking	Row-locking
References: https://dba.stackexchange.com/questions/1/what-are-the-main-differences-between-innodb-and-myisam		

Since, MyISAM does not provide referential Integrity and is the only storage engine that school provides to school's MySQL and phpMyAdmin, team SQeaL had to hard code to restrict insert and delete to maintain the relationship between tables using php.

Therefore, team SQeaL had to hard-code to restrict the insert and delete that can affect the referential integrity using PHP. The way that Team SQeaL handled will be provided on section "User Interface Implementation".

DATABASE TABLE IMPORTED TO PHPMYADMIN

Team SQeaL imported the following table to school's pre-installed phpMyAdmin provided by UW-ITConnect.

```
-- Dungeons and Dragons table creation

-- All the talbes are listed based on priorty insert



-- table for player

-- contain attributes of id, name, phone_num, Email, join_date, leave_date

-- Assume player in United States

-- NULL values: email

CREATE TABLE PLAYER

(
```

```
Player_id INT NOT NULL AUTO_INCREMENT,

Player_name VARCHAR(20) NOT NULL,

Player_phone CHAR(10) NOT NULL,

Player_email VARCHAR(30),

Player_join_date DATE NOT NULL,

Player_leave_date DATE,

-- Possible constraint?

PRIMARY KEY (Player_id),

CHECK (join_date < leave_date)

);


-- table for character

-- contain attributes of character_name, character_player_id(foreign) character_level

-- passive_perception

CREATE TABLE PCHARACTER

(

Character_name VARCHAR(15) NOT NULL,

Charac_player_id INT NOT NULL,

Charac_level INT NOT NULL CHECK(Charac_level > 0 AND Charac_level < 100),

Charac_passive_perception INT NOT NULL CHECK(Charac_passive_perception > 0 AND
Charac_passive_perception < 100),

-- constraints

PRIMARY KEY(Character_name),

FOREIGN KEY(Charac_player_id) REFERENCES PLAYER(Player_id) ON DELETE CASCADE ON UPDATE
CASCADE

);

-- Possibly have to create tigger for insert lvl and passive

-- table for ITEM

-- contain attributes of character_name, item_num(id), item_name value, item_count
```

```
-- Item_owner_character(foreign from character)

-- Item_value (plat, gold, silver, copper)

-- set constraint item max 99

CREATE TABLE ITEM

(

Item_id INT NOT NULL AUTO_INCREMENT,

Item_name VARCHAR(25) NOT NULL,

Item_owner_character VARCHAR(15) NOT NULL,

Item_value VARCHAR(6) NOT NULL CHECK (Item_value IN ('plat','gold','silver','copper')),

Item_count INT NOT NULL CHECK (Item_count > 0 AND Item_count < 99),

-- Constraints

PRIMARY KEY(Item_id),

FOREIGN KEY (Item_owner_character) REFERENCES PCHARACTER(Character_name) ON DELETE
CASCADE ON UPDATE CASCADE

);


-- table for MAP

-- contains attribute of map_number, map_name

-- Map_name varchar(25) change if you guys want

-- Simple contraints

CREATE TABLE MAP

(

Map_number INT NOT NULL,

Map_name VARCHAR(25) NOT NULL,

-- Constraint

PRIMARY KEY(Map_name),

CHECK (Map_number > 0)

);
```

```
-- table for campaign

-- contains attributes Campaign_name, player_id, camp_map_name, owner of campaign

CREATE TABLE CAMPAIGN

(

Campaign_name VARCHAR(25) NOT NULL,

Cam_map_name VARCHAR(25) NOT NULL,

Cam_master VARCHAR(20) NOT NULL,

Cam_player INT NOT NULL,

-- Constraints

PRIMARY KEY(Campaign_name),

FOREIGN KEY(Cam_map_name) REFERENCES MAP(Map_name) ON UPDATE CASCADE ON DELETE
CASCADE,

FOREIGN KEY (Cam_player) REFERENCES PLAYER(Player_id) ON UPDATE CASCADE ON DELETE
CASCADE

);


-- table for INCLUDE_CAMPAIGN_CHARACTER

-- Just contain primary keys of campaign and chacter_name

-- I personally think this is unnecessary

-- but we can use this table to just retrieve data of name of campaign and its name of involved charc

-- also may provide less condition to correlation with other tables since it only contains names (map,charc)

CREATE TABLE INCLUDE_CAMP_CHARC

(

Campai_name VARCHAR(25) NOT NULL,

Charc_name VARCHAR(15) NOT NULL,

PRIMARY KEY(Campai_name, Charc_name),

FOREIGN KEY (Campai_name) REFERENCES CAMPAIGN(Campaign_name) ON UPDATE CASCADE ON
DELETE CASCADE,
```

```
FOREIGN KEY (Charc_name) REFERENCES PCHARACTER(Character_name) ON UPDATE CASCADE ON
DELETE CASCADE

);


-- table for Point_of_interest

CREATE TABLE POINT_OF_INTEREST

(

Location_num INT NOT NULL AUTO_INCREMENT,

Location_name VARCHAR(15) NOT NULL,

Map_location_name VARCHAR(25) NOT NULL,

-- Constraints

PRIMARY KEY(Location_num),

UNIQUE KEY(Location_name),

FOREIGN KEY (Map_location_name) REFERENCES MAP(Map_name) ON DELETE CASCADE ON UPDATE
CASCADE

);

-- table for Episode

CREATE TABLE EPISODE

(

Episode_num INT NOT NULL AUTO_INCREMENT,

Campai_name VARCHAR(25) NOT NULL,

Episode_location VARCHAR(15) NOT NULL,

PRIMARY KEY(Episode_num),

FOREIGN KEY (Campai_name) REFERENCES CAMPAIGN(Campaign_name),

FOREIGN KEY(Episode_location) REFERENCES POINT_OF_INTEREST(Location_name) ON UPDATE
CASCADE ON DELETE CASCADE

);


-- table for NON_PLAYER_CHARACTEr
```

```
-- insert prior

CREATE TABLE NON_PLAYER_CHARACTER

(

Npc_name VARCHAR(15) NOT NULL,

Npc_class VARCHAR(20) NOT NULL CHECK (Npc_class IN ('merchant', 'mysterious', 'worrior', 'magician',
'traveler')),

PRIMARY KEY(Npc_name)

);

-- talbe for episode_npc

-- contains id, name, quest_reward, npc_level, epsidoe_num

-- Assume reward is only gold

-- Epi_num_npc ON DELETE CASCADE needs to be check

CREATE TABLE EPISODE_NPC

(

Npc_id INT NOT NULL AUTO_INCREMENT,

Epi_npc_name VARCHAR(15) NOT NULL DEFAULT 'NONE',

Epi_num_npc INT NOT NULL,

Npc_level INT NOT NULL,

Quest_reward INT NOT NULL DEFAULT 0,

-- Constraint

PRIMARY KEY(Npc_id),

FOREIGN KEY(Epi_npc_name) REFERENCES NON_PLAYER_CHARACTER(Npc_name),

FOREIGN KEY(Epi_num_npc) REFERENCES EPISODE(Episode_num),

CHECK(Npc_level > 0 AND Npc_level < 99),

CHECK(Quest_reward >= 0)

);


-- table include_camp_npc
```

```
-- Simply like INCLUDE CHARAC and CAMP
CREATE TABLE INCLUDE_CAMP_NPC
(
Camp_name VARCHAR(25) NOT NULL,
Npc_name VARCHAR(15) NOt NULL,
PRIMARY KEY(Camp_name, Npc_name),
FOREIGN KEY(Camp_name) REFERENCES CAMPAIGN(Campaign_name),
FOREIGN KEY(Npc_name) REFERENCES NON_PLAYER_CHARACTER(Npc_name)
);

-- table monster
CREATE TABLE MONSTER
(
Monster_name VARCHAR(25) NOT NULL,
Monster_level INT NOT NULL,
Monster_type VARCHAR(25) NOT NULL,
PRIMARY KEY(Monster_name),
CHECK(Monster_level > 0 AND Monster_level < 100)
);

-- table epsidoe monster
-- contain attribute ep_num, monster_name, monster_id, monster_lvel, drop item
CREATE TABLE EPISODE_MONSTER
(
Monster_id INT NOT NULL AUTO_INCREMENT,
Monster_name VARCHAR(25) NOT NULL,
Monster_level_epi INT NOT NULL,
Drop_item VARCHAR(20) NOT NULL DEFAULT 'NONE',
```

```
Monster_epi INT NOT NULL,
-- constraints
PRIMARY KEY(Monster_id),
FOREIGN KEY(Monster_name) REFERENCES MONSTER(Monster_name),
FOREIGN KEY(Monster_epi) REFERENCES EPISODE(Episode_num),
CHECK(Monster_level_epi > 0 AND Monster_level_epi < 100)
);

-- table inlucde monster _campaign
-- Monster_name_camp is monster name on camp
-- camp_name_mons is campaign name that monster is involved.
CREATE TABLE INCLUDE_CAMP_MONSTER
(
Camp_name_mons VARCHAR(25) NOT NULL,
Monster_name_camp VARCHAR(25) NOT NULL,
PRIMARY KEY(Camp_name_mons, Monster_name_camp),
FOREIGN KEY(Camp_name_mons) REFERENCES CAMPAIGN(Campaign_name),
FOREIGN KEY(Monster_name_camp) REFERENCES MONSTER(Monster_name)
);
-- End of Create tables --
```

DATABASE TABLE CONVERTED AFTER MYISAM STORAGE ENGINE

```
-- phpMyAdmin SQL Dump
-- version 4.8.5
-- https://www.phpmyadmin.net/
--
-- Host: vergil.u.washington.edu:6498
-- Generation Time: Mar 08, 2019 at 01:57 PM
-- Server version: 5.5.18
```

```
-- PHP Version: 7.2.4


SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";

SET AUTOCOMMIT = 0;

START TRANSACTION;

SET time_zone = "+00:00";



/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;

/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;

/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;

/*!40101 SET NAMES utf8mb4 */;


--

-- Database: `dungeon`

--


-- --------------------------------------------------------


--

-- Table structure for table `CAMPAIGN`

--


CREATE TABLE `CAMPAIGN` (

  `Campaign_name` varchar(25) NOT NULL,

  `Cam_map_name` varchar(25) NOT NULL,

  `Cam_master` varchar(20) NOT NULL,

  `Cam_player` int(11) NOT NULL
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


--
-- Dumping data for table `CAMPAIGN`
--


INSERT INTO `CAMPAIGN` (`Campaign_name`, `Cam_map_name`, `Cam_master`, `Cam_player`) VALUES
('Test Camp1', 'Raising Sun', 'Park', 1),
('Test Camp2', 'Raising Sun', 'Park', 2),
('Test Camp3', 'Raising Sun', 'Haram', 1);


-- --------------------------------------------------------


--
-- Table structure for table `EPISODE`
--


CREATE TABLE `EPISODE` (
  `Episode_num` int(11) NOT NULL,
  `Campai_name` varchar(25) NOT NULL,
  `Episode_location` varchar(15) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


--
-- Dumping data for table `EPISODE`
--


INSERT INTO `EPISODE` (`Episode_num`, `Campai_name`, `Episode_location`) VALUES
```

```
(1, 'Test Camp1', 'test interest'),
(2, 'Test Camp2', 'test interest');

-- --------------------------------------------------------

--
-- Table structure for table `EPISODE_MONSTER`
--

CREATE TABLE `EPISODE_MONSTER` (
  `Monster_id` int(11) NOT NULL,
  `Monster_name` varchar(25) NOT NULL,
  `Monster_level_epi` int(11) NOT NULL,
  `Drop_item` varchar(20) NOT NULL DEFAULT 'NONE',
  `Monster_epi` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `EPISODE_MONSTER`
--

INSERT INTO `EPISODE_MONSTER` (`Monster_id`, `Monster_name`, `Monster_level_epi`, `Drop_item`,
`Monster_epi`) VALUES
(1, 'weak undead', 10, 'bone', 2);

-- --------------------------------------------------------

--
```

```
-- Table structure for table `EPISODE_NPC`
--

CREATE TABLE `EPISODE_NPC` (
  `Npc_id` int(11) NOT NULL,
  `Epi_npc_name` varchar(15) NOT NULL DEFAULT 'NONE',
  `Epi_num_npc` int(11) NOT NULL,
  `Npc_level` int(11) NOT NULL,
  `Quest_reward` int(11) NOT NULL DEFAULT '0'
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `EPISODE_NPC`
--

INSERT INTO `EPISODE_NPC` (`Npc_id`, `Epi_npc_name`, `Epi_num_npc`, `Npc_level`, `Quest_reward`) VALUES
(3, 'Test NPC2', 2, 1, 1);

-- --------------------------------------------------------

--
-- Table structure for table `INCLUDE_CAMP_CHARC`
--

CREATE TABLE `INCLUDE_CAMP_CHARC` (
  `Campai_name` varchar(25) NOT NULL,
  `Charc_name` varchar(15) NOT NULL
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- ----------------------------------------------------------

--
-- Table structure for table `INCLUDE_CAMP_MONSTER`
--

CREATE TABLE `INCLUDE_CAMP_MONSTER` (
  `Camp_name_mons` varchar(25) NOT NULL,
  `Monster_name_camp` varchar(25) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `INCLUDE_CAMP_MONSTER`
--

INSERT INTO `INCLUDE_CAMP_MONSTER` (`Camp_name_mons`, `Monster_name_camp`) VALUES
('Test Camp2', 'weak undead');

-- ----------------------------------------------------------

--
-- Table structure for table `INCLUDE_CAMP_NPC`
--

CREATE TABLE `INCLUDE_CAMP_NPC` (
  `Camp_name` varchar(25) NOT NULL,
```

```
  `Npc_name` varchar(15) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `INCLUDE_CAMP_NPC`
--

INSERT INTO `INCLUDE_CAMP_NPC` (`Camp_name`, `Npc_name`) VALUES
('Test Camp2', 'Test NPC2');

-- --------------------------------------------------------

--
-- Table structure for table `ITEM`
--

CREATE TABLE `ITEM` (
  `Item_id` int(11) NOT NULL,
  `Item_name` varchar(25) NOT NULL,
  `Item_owner_character` varchar(15) NOT NULL,
  `Item_value` varchar(6) NOT NULL,
  `Item_count` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `ITEM`
--
```

```
INSERT INTO `ITEM` (`Item_id`, `Item_name`, `Item_owner_character`, `Item_value`, `Item_count`) VALUES
(1, 'Css Book', 'Super Worrior', 'plat', 1);

-- --------------------------------------------------------

--
-- Table structure for table `MAP`
--

CREATE TABLE `MAP` (
  `Map_number` int(11) NOT NULL,
  `Map_name` varchar(25) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `MAP`
--

INSERT INTO `MAP` (`Map_number`, `Map_name`) VALUES
(1, 'Raising Sun'),
(2, 'Victoria Island'),
(3, 'Sleep Wood');

-- --------------------------------------------------------

--
-- Table structure for table `MONSTER`
--
```

```
CREATE TABLE `MONSTER` (
  `Monster_name` varchar(25) NOT NULL,
  `Monster_level` int(11) NOT NULL,
  `Monster_type` varchar(25) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `MONSTER`
--

INSERT INTO `MONSTER` (`Monster_name`, `Monster_level`, `Monster_type`) VALUES
('Animal1', 1, 'animal'),
('weak undead', 10, 'undead');

-- --------------------------------------------------------

--
-- Table structure for table `NON_PLAYER_CHARACTER`
--

CREATE TABLE `NON_PLAYER_CHARACTER` (
  `Npc_name` varchar(15) NOT NULL,
  `Npc_class` varchar(20) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `NON_PLAYER_CHARACTER`
```

```
--

INSERT INTO `NON_PLAYER_CHARACTER` (`Npc_name`, `Npc_class`) VALUES
('Test NPC1', 'merchant'),
('Test NPC2', 'magician');

-- --------------------------------------------------------

--
-- Table structure for table `PCHARACTER`
--

CREATE TABLE `PCHARACTER` (
  `Character_name` varchar(15) NOT NULL,
  `Charac_player_id` int(11) NOT NULL,
  `Charac_level` int(11) NOT NULL,
  `Charac_passive_perception` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `PCHARACTER`
--

INSERT INTO `PCHARACTER` (`Character_name`, `Charac_player_id`, `Charac_level`,
`Charac_passive_perception`) VALUES
('Super Worrior', 4, 99, 99),
('Strong Pope', 1, 44, 44);
```

```
-- ----------------------------------------------------------


--
-- Table structure for table `PLAYER`
--


CREATE TABLE `PLAYER` (
  `Player_id` int(11) NOT NULL,
  `Player_name` varchar(20) NOT NULL,
  `Player_phone` char(10) NOT NULL,
  `Player_email` varchar(30) DEFAULT NULL,
  `Player_join_date` date NOT NULL,
  `Player_leave_date` date DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


--
-- Dumping data for table `PLAYER`
--


INSERT INTO `PLAYER` (`Player_id`, `Player_name`, `Player_phone`, `Player_email`, `Player_join_date`,
`Player_leave_date`) VALUES
(1, 'nemo', '2061231234', 'jhpp114@uw.edu', '2000-10-05', '0000-00-00'),
(2, 'Haram', '1231231234', 'ha@uw.edu', '2013-10-19', '0000-00-00'),
(3, 'Rider', '9879879876', 'rid@uw.edu', '2016-12-25', '0000-00-00'),
(4, 'Park', '2133215678', 'park@uw.edu', '1992-10-05', '0000-00-00');


-- ----------------------------------------------------------
```

```
--
-- Table structure for table `POINT_OF_INTEREST`
--

CREATE TABLE `POINT_OF_INTEREST` (
  `Location_num` int(11) NOT NULL,
  `Location_name` varchar(15) NOT NULL,
  `Map_location_name` varchar(25) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `POINT_OF_INTEREST`
--

INSERT INTO `POINT_OF_INTEREST` (`Location_num`, `Location_name`, `Map_location_name`) VALUES
(1, 'test interest', 'Victoria Island');

--
-- Indexes for dumped tables
--

--
-- Indexes for table `CAMPAIGN`
--
ALTER TABLE `CAMPAIGN`
  ADD PRIMARY KEY (`Campaign_name`),
  ADD KEY `Cam_map_name` (`Cam_map_name`),
  ADD KEY `Cam_player` (`Cam_player`);
```

```
--
-- Indexes for table `EPISODE`
--
ALTER TABLE `EPISODE`
  ADD PRIMARY KEY (`Episode_num`),
  ADD KEY `Campai_name` (`Campai_name`),
  ADD KEY `Episode_location` (`Episode_location`);

--
-- Indexes for table `EPISODE_MONSTER`
--
ALTER TABLE `EPISODE_MONSTER`
  ADD PRIMARY KEY (`Monster_id`),
  ADD KEY `Monster_name` (`Monster_name`),
  ADD KEY `Monster_epi` (`Monster_epi`);

--
-- Indexes for table `EPISODE_NPC`
--
ALTER TABLE `EPISODE_NPC`
  ADD PRIMARY KEY (`Npc_id`),
  ADD KEY `Epi_npc_name` (`Epi_npc_name`),
  ADD KEY `Epi_num_npc` (`Epi_num_npc`);

--
-- Indexes for table `INCLUDE_CAMP_CHARC`
--
```

```
ALTER TABLE `INCLUDE_CAMP_CHARC`
  ADD PRIMARY KEY (`Campai_name`,`Charc_name`),
  ADD KEY `Charc_name` (`Charc_name`);

--
-- Indexes for table `INCLUDE_CAMP_MONSTER`
--
ALTER TABLE `INCLUDE_CAMP_MONSTER`
  ADD PRIMARY KEY (`Camp_name_mons`,`Monster_name_camp`),
  ADD KEY `Monster_name_camp` (`Monster_name_camp`);

--
-- Indexes for table `INCLUDE_CAMP_NPC`
--
ALTER TABLE `INCLUDE_CAMP_NPC`
  ADD PRIMARY KEY (`Camp_name`,`Npc_name`),
  ADD KEY `Npc_name` (`Npc_name`);

--
-- Indexes for table `ITEM`
--
ALTER TABLE `ITEM`
  ADD PRIMARY KEY (`Item_id`),
  ADD KEY `Item_owner_character` (`Item_owner_character`);

--
-- Indexes for table `MAP`
--
```

```
ALTER TABLE `MAP`
  ADD PRIMARY KEY (`Map_name`);


--
-- Indexes for table `MONSTER`
--
ALTER TABLE `MONSTER`
  ADD PRIMARY KEY (`Monster_name`);


--
-- Indexes for table `NON_PLAYER_CHARACTER`
--
ALTER TABLE `NON_PLAYER_CHARACTER`
  ADD PRIMARY KEY (`Npc_name`);


--
-- Indexes for table `PCHARACTER`
--
ALTER TABLE `PCHARACTER`
  ADD PRIMARY KEY (`Character_name`),
  ADD KEY `Charac_player_id` (`Charac_player_id`);


--
-- Indexes for table `PLAYER`
--
ALTER TABLE `PLAYER`
  ADD PRIMARY KEY (`Player_id`);
```

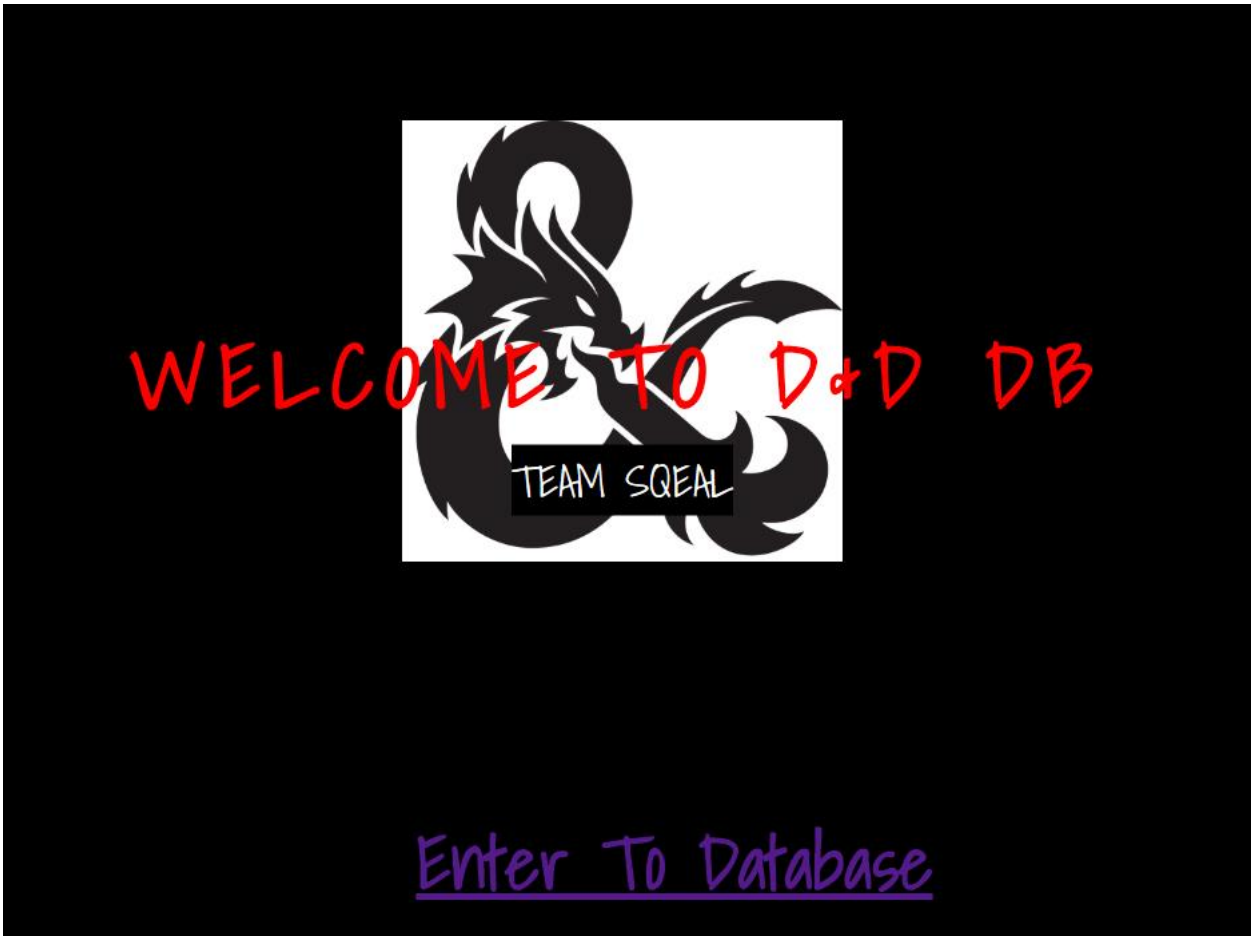
```
--
-- Indexes for table `POINT_OF_INTEREST`
--
ALTER TABLE `POINT_OF_INTEREST`
  ADD PRIMARY KEY (`Location_num`),
  ADD UNIQUE KEY `Location_name` (`Location_name`),
  ADD KEY `Map_location_name` (`Map_location_name`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `EPISODE`
--
ALTER TABLE `EPISODE`
  MODIFY `Episode_num` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;

--
-- AUTO_INCREMENT for table `EPISODE_MONSTER`
--
ALTER TABLE `EPISODE_MONSTER`
  MODIFY `Monster_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;

--
-- AUTO_INCREMENT for table `EPISODE_NPC`
--
ALTER TABLE `EPISODE_NPC`
```

```
  MODIFY `Npc_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;


--
-- AUTO_INCREMENT for table `ITEM`
--
ALTER TABLE `ITEM`
  MODIFY `Item_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;


--
-- AUTO_INCREMENT for table `PLAYER`
--
ALTER TABLE `PLAYER`
  MODIFY `Player_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;


--
-- AUTO_INCREMENT for table `POINT_OF_INTEREST`
--
ALTER TABLE `POINT_OF_INTEREST`
  MODIFY `Location_num` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
COMMIT;


/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

WEB APPLICATION IMPLEMENTATION

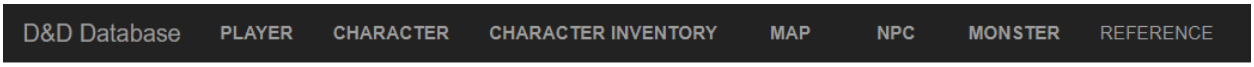
The user interface team SQeaL had created is from scratch using HTML, CSS, and Javascript.

USER WELCOMING INTERFACE



http://students.washington.edu/jhpp114/projec/index.php

The welcome page will provide a link to enter to user insert and delete page where allows user who would be dungeon master can manage the data that they want.

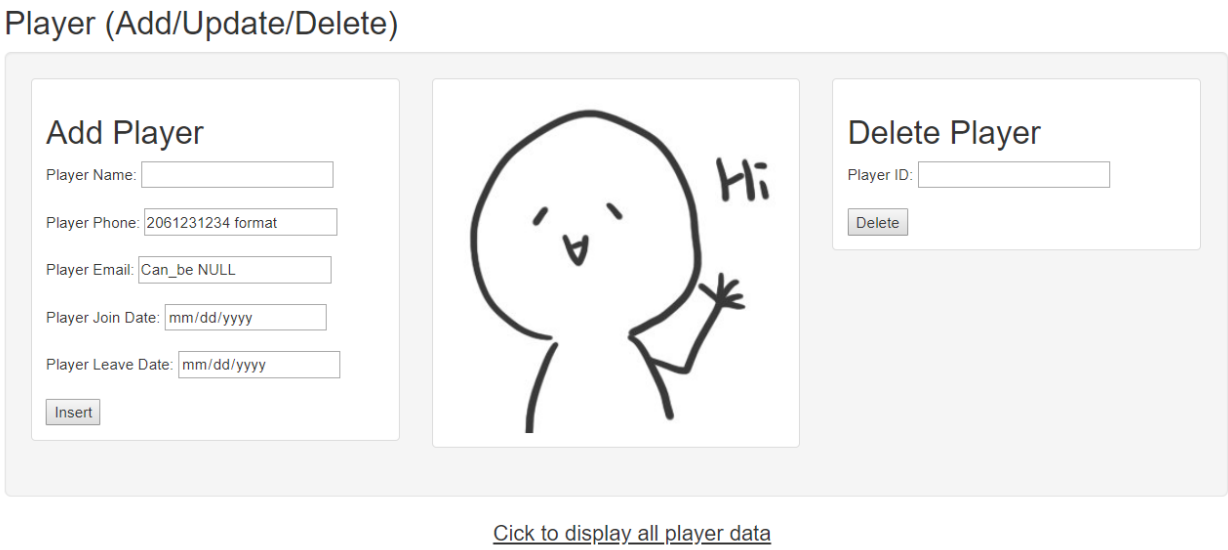


By going through link, on top of the user screen, the menu bar will be provided where user can navigate various link where where can insert and delete the data they put in the database.

INSERT DATA BY USER

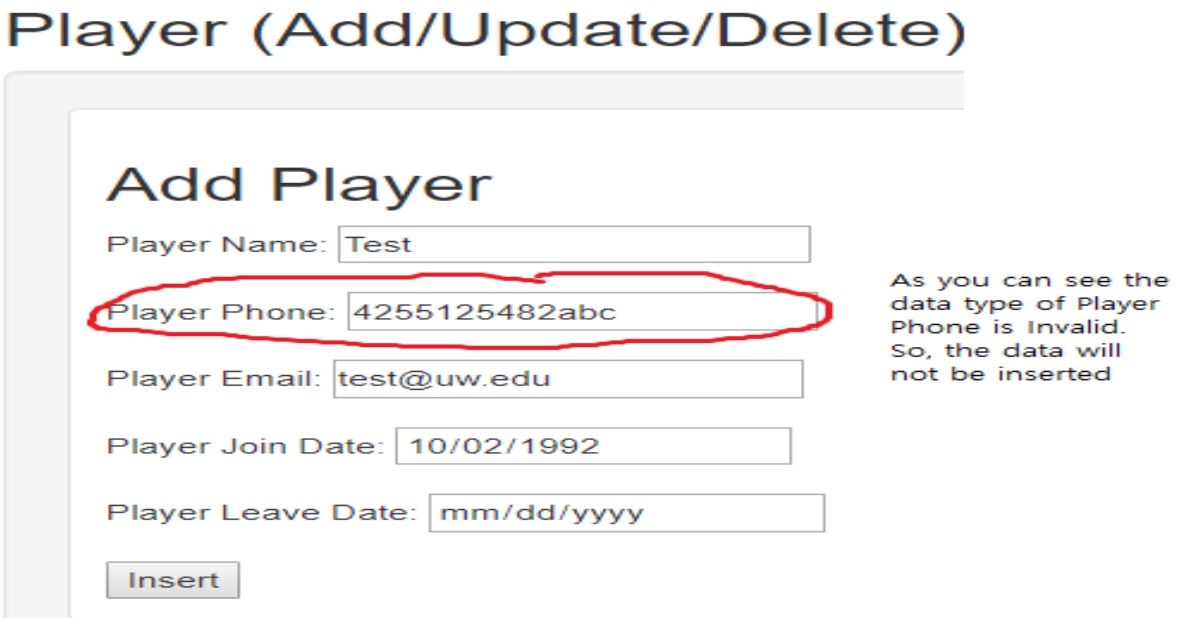
Since the theme and purpose of team SQeaL's database is to provide Dungen Master to manage the game Dungeons and Dragons. Therefore, Team SQeaL had approach that all the data must inserted by the dungeon master in order to be realistic. Figure 4.1 will provide the interface that Dungeon master can insert the data. Explanation about delete part will be explained on Section Delete Data By User.

Figure 4.1: INSERT USER INTERFACE



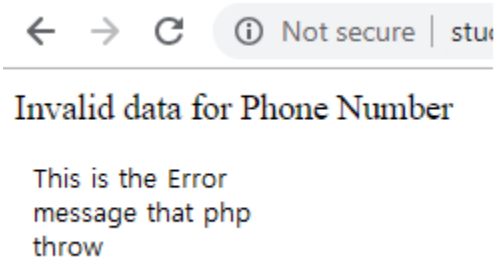
This is one of the insert and delete where dungeon master can add and delete player. Figure: 4.2 "Example using Player Insert" will provide explanation with image about how the data will be inserted using UI to actual database.

Figure 4.2: Example Using Player Insert



Since the data type that user entered is invalid (Red Circle), therefore from php code provided below will throw error message and automatically going back to the insert User Interface and disconnect the connection with database.

```php
if (empty($Player_phone)) {
  header("refresh:2; url=main.php");
  echo "Player_phone number can not be empty";
  echo "<br>";
  echo "Going back to main.html";
  echo "<br>";
  die();
} else if (!preg_match("/^\d{10}+$/", $Player_phone)) {
  header("refresh:2; url=main.php");
  echo "Invalid data for Phone Number";
  echo "<br>";
  die();
}
```



If the format of insert is correct then it will store data in phpMyAdmin by going through php and html, figure 4.3: Explanation Store Data Inserted From User(Front-end && Back-end)" will provide explanation on how it read and store data that is inserted from user to database.

Figure 4.3: Explanation Store Data Inserted From User (Front-End)

```php
<h2 id = "player">Add Player</h2>
  <form class="" action="playerInsert.php" method="post">
    Player Name: <input type="text" name="Player_name" value=""> <br><br>
    Player Phone: <input type="text" name="Player_phone" value="2061231234 format"> <br><br>
    Player Email: <input type="text" name="Player_email" value="Can_be NULL"> <br><br>
    Player Join Date: <input type="date" name="Player_join_date" value=""> <br><br>
    Player Leave Date: <input type="date" name="Player_leave_date" value="Can_be NULL"> <br><br>
    <input type="Submit" name="" value="Insert">
  </form>
/div>
```

From "Form" tag, the action that connected to is "playerInsert.php" where it sends the data that inserted by user to playerInsert.php file

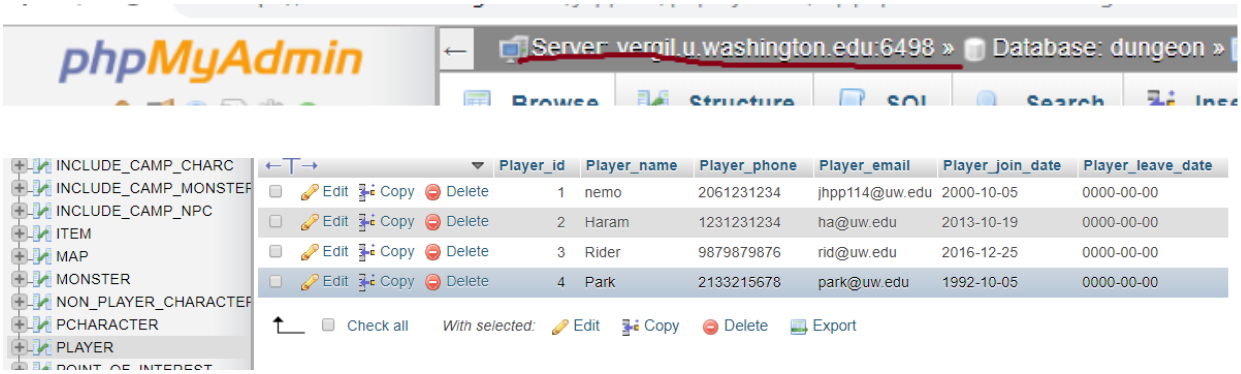
Figure 4.3 Explanation Store Data Inserted From User (Back-end)

```php
$mysqli_connection = new MySQLi('vergil.u.washington.edu', 'root', '[redacted]', '[redacted]', [redacted]);
if($mysqli_connection->connect_error){
   echo "Not connected, error: ".$mysqli_connection->connect_error;
}

$Player_name = $_POST['Player_name'];
$Player_phone = $_POST['Player_phone'];
$Player_email = $_POST['Player_email'];
$Player_join_date =$_POST['Player_join_date'];
$Player_leave_date = $_POST['Player_leave_date'];
```

```php
}
// insert the data
$sql = "INSERT INTO PLAYER (Player_name,Player_phone,Player_email,Player_join_date,Player_leave_date)
VALUES ('$Player_name', '$Player_phone', '$Player_email', '$Player_join_date', '$Player_leave_date')";
header("refresh:2; url=main.php");
if (!mysqli_query($mysqli_connection,$sql)) {
  echo 'Not Inserted';
  echo "<br>";
} else {
  echo 'Inserted Successfully';
  echo "<br>";
}
```

On back-end, it connect the php file with database hosting server and create variables to store user inserted data. then it is simply insert the data using the variables to the certain table.

Of course, due to the storage engine (MyISAM), team SQeaL had to write all the code to restrict the wrong input data. Because it is connected with the phpMyAdmin (where it contains MySQL Table), the data that is inserted from Front-end can be store in team SQeaL's database.



Through User Interface, these stored data can be displayed. The code level explanation will be provided in figure 4.4 "Explanation about read database and display through User Interface".

Cick to display all player data

Connected.

Player Number	Player Name	Player Phone Number	Player Email	Player Join Date	Player Leave Date
1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
3	Rider	9879879876	rid@uw.edu	2016-12-25	0000-00-00
4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

Figure 4.4: Explanation about Read Database and Display through User Interface

To display the data, following HTML, PHP, and Javascript code had been used.

```
<table>
  <tr>
    <th>Player Number</th>
    <th>Player Name</th>
    <th>Player Phone Number</th>
    <th>Player Email</th>
    <th>Player Join Date</th>
    <th>Player Leave Date</th>
  </tr>
  <?php
    $mysqli_connection = new MySQLi('vergil.u.washington.edu', 'root', [redacted], [redacted]);
    if($mysqli_connection->connect_error){
       echo "Not connected, error: ".$mysqli_connection->connect_error;
    }
    else{
       echo "Connected.";
    }
    // $conn = mysqli_connect("", "root", "", 'dungeon');
    // if ($conn->connect_error) {
    //   die("Connection failed:" . $conn->connect_error);
    // }
    $sql = "SELECT Player_id, Player_name, Player_phone, Player_email, Player_join_date, Player_leave_date FROM PLAYER";
    $result = $mysqli_connection->query($sql);
    if ($result->num_rows > 0) {
      while ($row = $result-> fetch_assoc()) {
        echo "<tr><td>". $row["Player_id"] . "</td><td>". $row["Player_name"] ."</td><td>". $row["Player_phone"] ."</td><td>". $row["Player_email"] ."</td><td>". $row["
        <td>". $row["Player_leave_date"] ."</td></tr>";
      }
      echo "</table>";
    } else {
      echo "0 result";
    }
    $mysqli_connection->close();
```

Connect the MySQL DBMS that is stored in phpMyAdmin

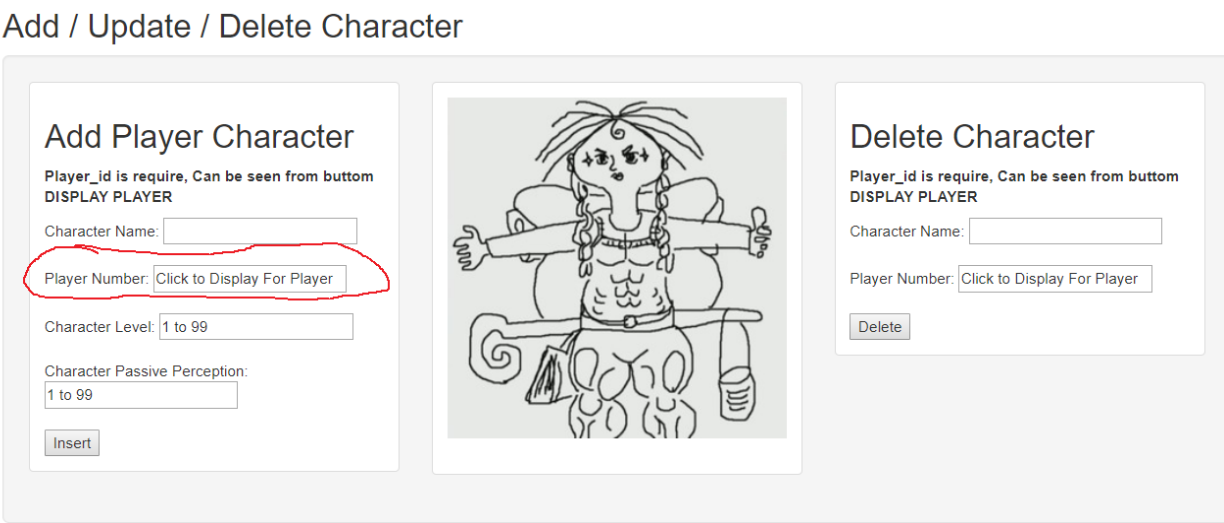
Where using Storage Engine MyISAM

Read all data that is stored in MySQL and display it to user using HTML and Javascript

INSERT WITH FOREIGN KEY RESTRICTION

Since the statement: FOREIGN KEY(Charac_player_id) REFERENCES PLAYER(Player_id) ON DELETE CASCADE ON UPDATE CASCADE **is not** a standard SQL language and the only storage engine provided from school (UW-ITConnect) is **MyISAM** (see **CREATE DATABASE section to have better understand**), team SQeaL had to give restrictions to user when certain inserts that can violate our Relational Model happens. Figure 4.5: **"Restriction on Insert Explanation"** will provide how can it handle the insert when its violate team SQeaL's relational model.

Figure 4.5: Restriction on Insert Explanation



The red circle on "Add Player Character", it requires valid player_id, which has to be automatically manage by .sql file however, since MyISAM is not supporting anything related to referential integrity, following code using php was required.

```php
if (!empty($Charac_player_id)) {
  $sqlResult = "SELECT Player_id FROM PLAYER";
  $result = $mysqli_connection->query($sqlResult);
  $fore = TRUE;
  while ($row = $result->fetch_assoc()) {
    if ($row["Player_id"] == $Charac_player_id) {
      $fore = TRUE;
      break;
    } else {
      $fore = FALSE;
    }
  }
}

if (!$fore) {
  header("refresh:3; url=main.php");
  echo "Foreign key Constraint";
  die();
}
```

Read Everything that is stored in Player Table at player database and check one by one.

The algorithm itself is O(n) at worst, which is bad algorithm.

Player Number	Player Name	Playe Numbe
1	nemo	20612
2	Haram	12312
3	Rider	98798
4	Park	21332

The Player Number 11 is not exist in the Player Database, so it will throw error message and disconnect the connect with database and return to the insert page

Add / Update / Delete Charac

Add Player Character

Player_id is require, Can be seen from buttom DISPLAY PLAYER

Character Name: Strong Elf

Player Number: 11

Character Level: 1

Character Passive Perception: 1

Insert

When User press the insert button even though the Player number is not exist, then the php will throw exception error

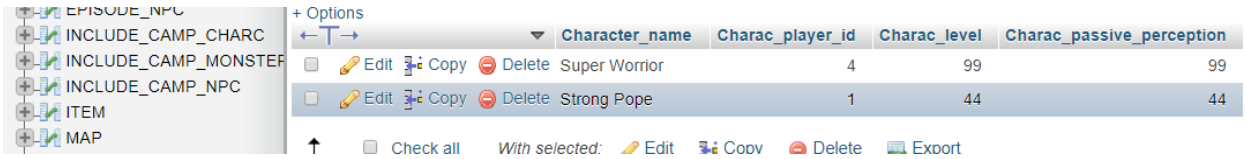
Not secure

Foreign key Constraint

There are also regular insert check for Character Level (between 1~99) and Character Passive Perception (between 1~99) all written in php.

If all the insert inputs are valid, then the following sql written in php will run and store it into database, which is the same way as Player Insert works.

```
}
$sql = "INSERT INTO PCHARACTER (Character_name, Charac_player_id, Charac_level, Charac_passive_perception)
VALUES ('$Character_name', '$Charac_player_id', '$Charac_level', '$Charac_passive_perception')";
header("refresh:3; url=main.php");
if (!mysqli_query($mysqli_connection,$sql)) {
  echo 'Not Inserted';
  echo "<br>";
} else {
  echo "Inserted";
  echo "<br>";
}
```



To display the data for character, the way of transfer and grab data is exactly same as Player but used different sql command.

```
<table>
  <tr>
    <th>Character Name</th>
    <th>Player ID</th>
    <th>Character Level</th>
    <th>Passive Perception</th>
  </tr>
  <?php
    $mysqli_connection = new MySQLi('vergil.u.washington.edu', 'root', '0505007pP', 'dungeon', 6498);
    if($mysqli_connection->connect_error){
       echo "Not connected, error: ".$mysqli_connection->connect_error;
    }
    else{
       echo "Connected.";
    }
    $sql = "SELECT Character_name, Charac_player_id, Charac_level, Charac_passive_perception FROM PCHARACTER";
    $result = $mysqli_connection->query($sql);
    if ($result->num_rows > 0) {
      while ($row = $result-> fetch_assoc()) {
        echo "<tr><td>". $row["Character_name"] . "</td><td>". $row["Charac_player_id"] ."</td><td>". $row["Charac_level"] ."</td><td>". $row["Charac_passive_perception"] ."</td></t
      }
      echo "</table>";
    } else {
      echo "0 result";
    }
    $mysqli_connection->close();
  ?>
</table>
```

Connected.

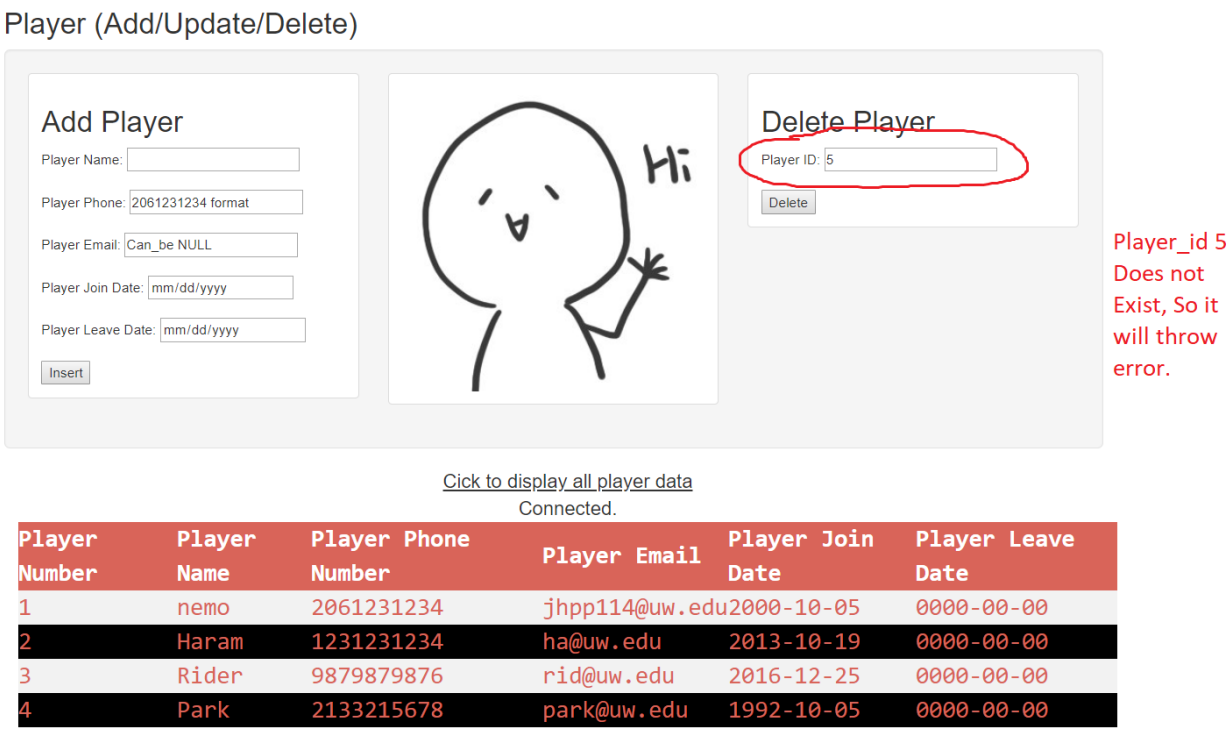
Character Name	Player ID	Character Level	Passive Perception
Super Worrior	4	99	99
Strong Pope	1	44	44

All other Tables: Item/Map/Campaign/Point Of Interest/NPC/Episode NPC/Monster/Monster In Episode is working the same way of transferring data that user inserted with checking constraints on back-end using php and store it in database and retrieve data to display back to user.

DELETE DATA BY USER (WITHOUT RESTRICTION)

Unlike insert, delete feature that team SQeaL proposed only requires Primary key, which can display to user. Because in real life situation, it is wrong to ask user to type user number which will work as primary key, but when it is deleting, team SQeaL wanted to get confirm by user by asking its primary key to delete it or not. Figure 4.6 will contain explanation on how delete feature works.

Figure 4.6: Explanation of Delete



Player Number	Player Name	Player Phone Number	Player Email	Player Join Date	Player Leave Date
1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
3	Rider	9879879876	rid@uw.edu	2016-12-25	0000-00-00
4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

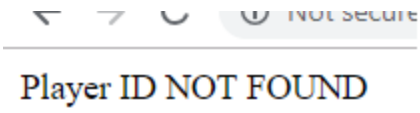
Notice that the Dungeon master is trying to delete a player who has player id 5, however, the database does not have player number 5, in that way following Back-end php code below will restrict the delete.

```php
if (!empty($Player_id)) {
  $sqlResult = "SELECT Player_id FROM PLAYER";
  $result = $mysqli_connection->query($sqlResult);
  $fore = TRUE;
  if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
      if ($row["Player_id"] == $Player_id) {
        $sql = "DELETE FROM PLAYER WHERE Player_id = $Player_id";
        $fore = TRUE;
        break;
      } else {
        $fore = FALSE;
      }
    }
  } else {
    $fore = FALSE;
  }
}

if (!$fore) {
  header("refresh:3; url=main.php");
  echo "Player ID NOT FOUND";
  die();
}
```

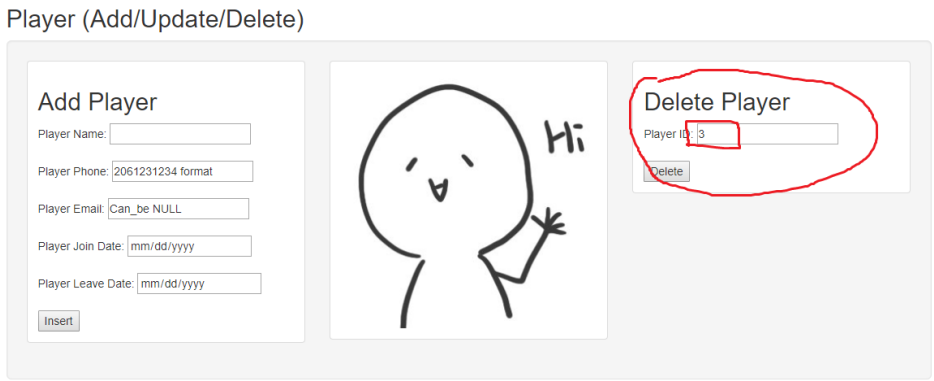
Read Player_id data from Player Database and if the the user input Player_id is not exist, then disconnect the server that is connected with database and return to delete page.

Also display the Error message to user

For Front-end part, the following message will be display.



Once, this message throw on display, it will automatically go back to delete interface.



Notice that the Player Id 3 exist on the database. Therefore, the delete will be success.

Cick to display all player data
Connected.

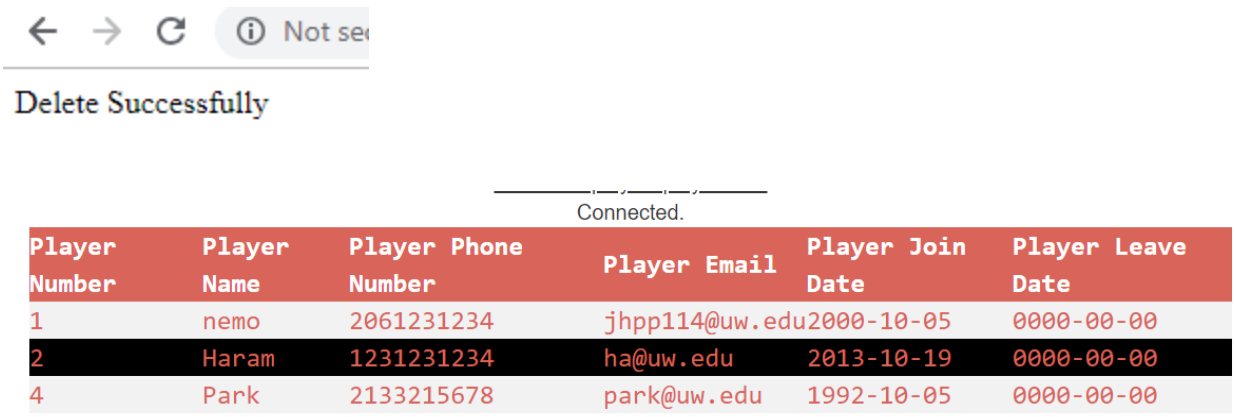
Player Number	Player Name	Player Phone Number	Player Email	Player Join Date	Player Leave Date
1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
3	Rider	9879879876	rid@uw.edu	2016-12-25	0000-00-00
4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

Since the delete input primary key is exist on the Player database, following php query will run on the back-end side.

```php
if (!empty($Player_id)) {
  $sqlResult = "SELECT Player_id FROM PLAYER";
  $result = $mysqli_connection->query($sqlResult);
  $fore = TRUE;
  if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
      if ($row["Player_id"] == $Player_id) {
        $sql = "DELETE FROM PLAYER WHERE Player_id = $Player_id";
        $fore = TRUE;
        break;
      } else {
        $fore = FALSE;
      }
    }
  } else {
    $fore = FALSE;
  }
}

if (!$fore) {
  header("refresh:3; url=main.php");
  echo "Player ID NOT FOUND";
  die();
}
```

On Back-end, Since, it passes the restriction written on php, the php sql will run and delete the data from database

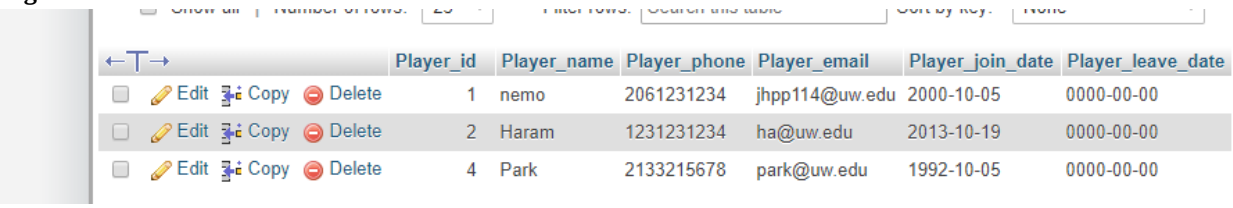
For the front-end following message on below will be display.



Player Number	Player Name	Player Phone Number	Player Email	Player Join Date	Player Leave Date
1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

Since, the data had been delete from database, on, display, Player_id 3 is disappeared. Figure 4.7 Prove of Delete will provide figure to prove that the data got deleted.

Figure 4.7: Prove of Delete



	Player_id	Player_name	Player_phone	Player_email	Player_join_date	Player_leave_date
Edit Copy Delete	1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
Edit Copy Delete	2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
Edit Copy Delete	4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

DELETE DATA BY USER (WITH RESTRICTION)

Team SQeaL had to handle all tables restriction for foreign key. For instance, it does not make sense that if the player owned a character and if Player get deleted but Character still exist. Due to the MyISAM engine, **which is not providing any of CASCADE,** team SQeaL had to hard code and provide restriction on delete based on the relational model. Figure 4.7 “Reject the Delete” will provide how it's going to work both front-end and back-end.

Figure 4.7: Reject the Delete

Player Number	Player Name	Player Phone Number	Player Email	Player Join Date	Player Leave Date
1	nemo	2061231234	jhpp114@uw.edu	2000-10-05	0000-00-00
2	Haram	1231231234	ha@uw.edu	2013-10-19	0000-00-00
4	Park	2133215678	park@uw.edu	1992-10-05	0000-00-00

Add / Update / Delete Character

Add Player Character

Player_id is require, Can be seen from buttom DISPLAY PLAYER

Character Name:

Player Number: Click to Display For Player

Character Level: 1 to 99

Character Passive Perception: 1 to 99

Insert

Delete Character

Player_id is require, Can be seen from buttom DISPLAY PLAYER

Character Name:

Player Number: Click to Display For Player

Delete

Notice that the Player_id 4 owned a character, therefore, by any chance, when Dungeon Master try to delete Player 4, the restriction has to be display and reject the delete (Since Cascade is not provided on MyISAM storage engine).

Cick to display all Character data

Connected.

Character Name	Player ID	Character Level	Passive Perception
Super Worrior	4	99	99
Strong Pope	1	44	44

Notice that the Player id 4 owns a character and

Player (Add/Update/Delete)

Add Player

Player Name:

Player Phone: 2061231234 format

Player Email: Can_be NULL

Player Join Date: mm/dd/yyyy

Player Leave Date: mm/dd/yyyy

Insert

Delete Player

Player ID: 4

Delete

Dungeon Master is trying to delete Player 4 from Player database then the following display message will be throw to the front-end.

Not secure | students.washi

Delete the Character First Before you Delete Player

What happens on back-end going to be explained through figure 4.8 Delete Restriction Back-end.

Figure 4.8 Delete Restriction Back-end

```php
$mysqli_connection = new MySQLi('vergil.u.washington.edu', 'root', '[redacted]', 'dungeon', [redacted]);
if($mysqli_connection->connect_error){
   echo "Not connected, error: ".$mysqli_connection->connect_error;
}

// select the Database
$Player_id = $_POST["Player_id"];
$Player_name = $_POST["Player_name"];

if (!empty($Player_id)) {
  $sqlResult = "SELECT Charac_player_id FROM PCHARACTER";
  $result = $mysqli_connection->query($sqlResult);
  $fore1 = TRUE;
  if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
      if ($row["Charac_player_id"] == $Player_id) {
        $fore1 = FALSE;
        break;
      } else {
        $fore1 = TRUE;
      }
    }
  } else {
    $fore1 = TRUE;
  }
}

if (!$fore1) {
  header("refresh:3; url=main.php");
  echo "Delete the Character First Before you Delete Player";
  die();
}
```

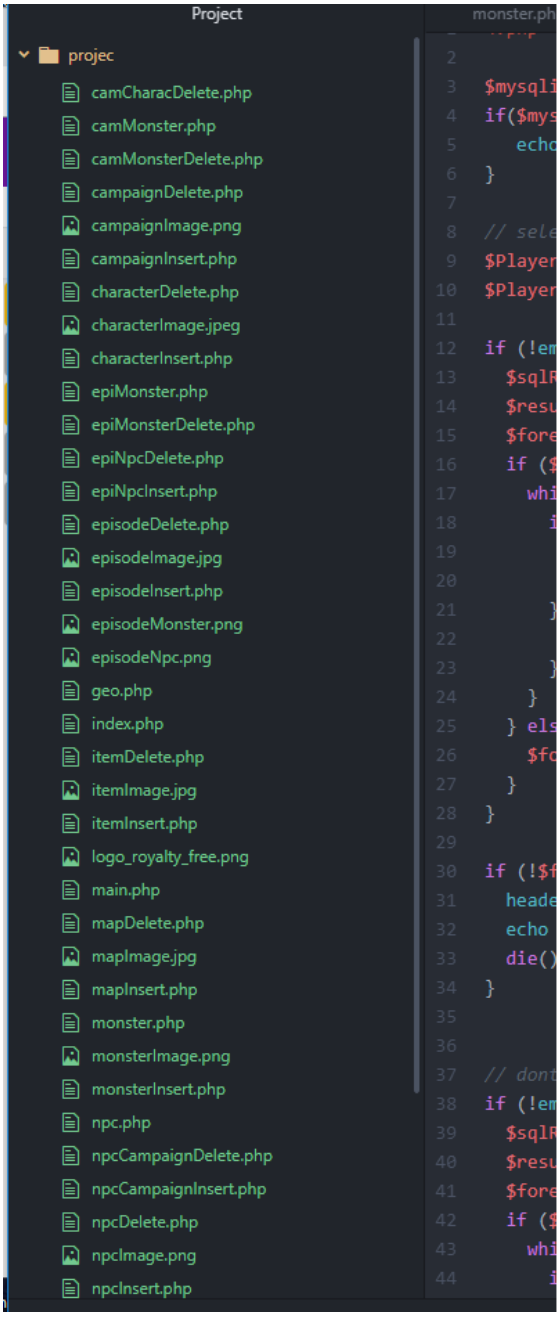
The Algorithm itself is O(n)

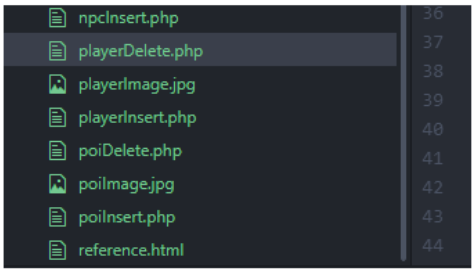
Connect with Student server which has MySQL that running in UW-ITConnect's provided running with MyISAM storage engine

It will run a loop see if the PCHARACTER database contains any Player_id that User inserted and if it does, then it will throw a message and disconnect the server. After that it will go back to delete page.

UPLOAD DATA FILE

Team SQeaL had uploaded the Dungeons and Dragons web application on student server.





This is Total Number of php files with imaged used for User Interface.

If I copy and paste all the code, it will be too many..

If you want the code file, please contact jhpp114@uw.edu because it contains password and netid information.

to visit the web, visit: http://students.washington.edu/jhpp114/projec

METHODOLOGY

Team SQeaL had to learn and put a lot of effort to build the website that host on the school student server and interacting with database. The following table is providing the methodology that we used to implement the web and explanation of why Team SQeaL needed to know with reference of URL. For Design of our web application, please go to Section “WEB APPLICATION IMPLEMENTATION”.

Methodology	Explanation	Reference
MySQL	MySQL on shared UW Hosting (Default on MyISAM storage engine and InnoDB does not exist)	https://itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/install-mysql/
phpMyAdmin	phpMyAdmin provides an esay-to-use web interface for most of the features of MySQL	https://itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/installing-phpmyadmin/
SSH(vergil.u.washington.edu)	Activating shared web hosting (Student Server)	https://itconnect.uw.edu/connect/web-publishing/shared-hosting/activating-shared-web-hosting/
SSH (Student Server)	Connect to server	https://itconnect.uw.edu/connect/web-publishing/shared-hosting/ssh/
Connecting to MySQL using PHP	allows uw-student server connect to MySQL using php	https://itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/connecting-to-mysql-using-php/
Learning about Storage Engine	Knowing and understand InnoDB and MyISAM engine that effects on MySQL	https://dba.stackexchange.com/questions/1/what-are-the-main-differences-between-innodb-and-myisam
HTML	providing User Interface	https://www.w3schools.com/html/
Bootsrap	CSS (Cascading Style Sheets).	https://getbootstrap.com/
php	Server-side Language	https://www.w3schools.com/php/default.asp

All the images that are used inside of Web application, please visit: http://students.washington.edu/jhpp114/projec/

1) Click link “Enter to database”
2) Click REFERENCE on the menu bar.

Other than those 4 url on Reference.html,

Other images had accomplished through tool “paint” and “aesprite” worked by Team member Jun Park’s cousin Minsu Lee.

UPDATE SECTION

This section provides the changes of our report based on iterations and its given feedbacks. The section will be divided based with subtitle with update iteration + number.

UPDATE FROM ITERATION 1

The feedback that team SQeaL had received was from Relational Model. The primary key in relational model were underline however, the foreign keys are not underlined. Figure 1.1 “Before Update” will provide the figure before update and Figure 1.2 “After Update” will provide the figure after update.

Figure 1: Before Update

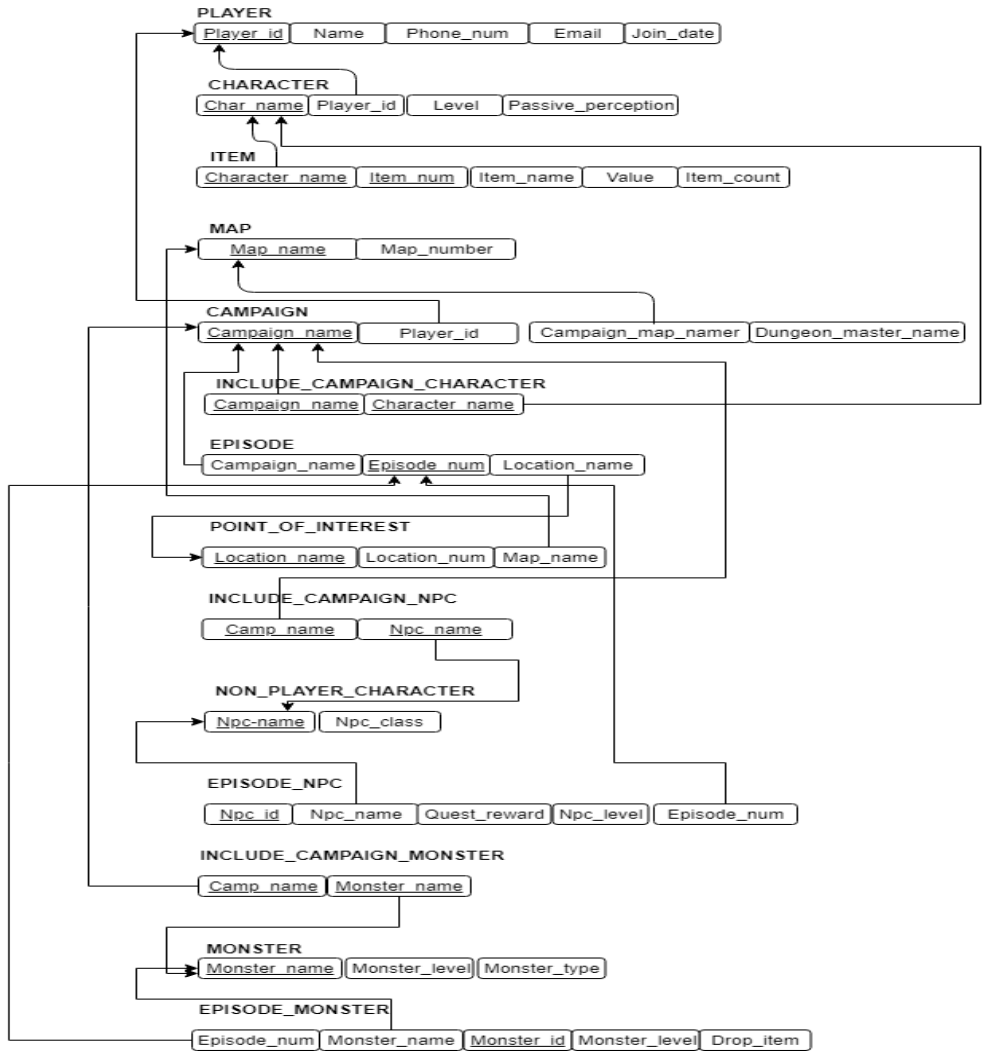
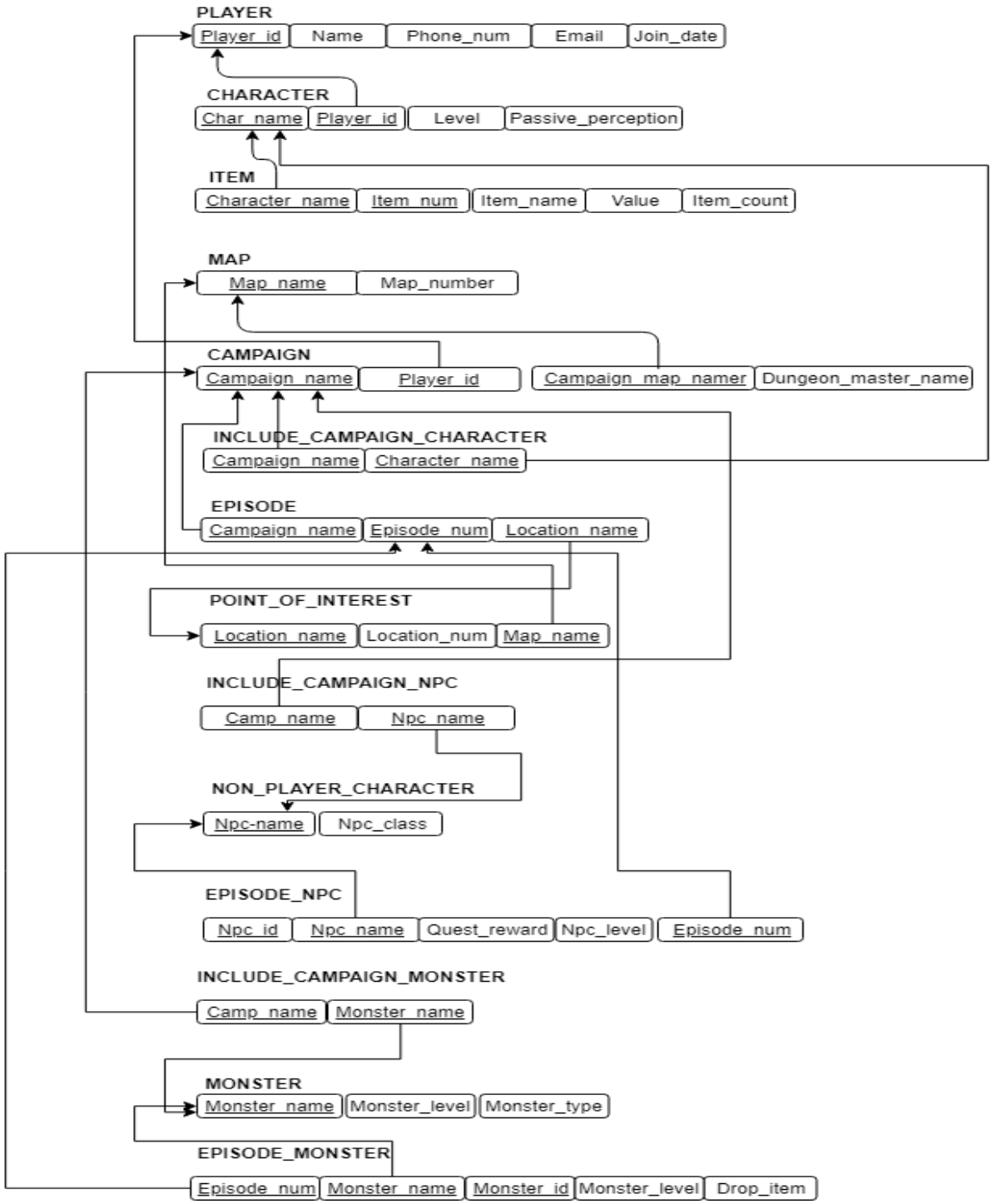


Figure 1.2 After Update



UPDATE FROM ITERATION 2

The updates that team SQeaL had made during Iteration 2 was mainly about tooling decision.

The initial plan that team SQeaL had will be shown below of this sentence, in figure 2.1 “Initial Tool Decision”

Figure 2.1 Initial Tool Decision

TOOL	Team SQeaL choice	Reason
Database Management System	MySQL	● Free, cost-effective ● Various resources helping learn about the DBMS.
User Interface	Java Console application	● Team SQeaL is not familiarize with UI. ● Team member familiarity. ● Provides library to connect with MySQL.
Database Hosting	AWS (Cloud Service)	● Amazon RDS (Relationship database) ● Free
Additional Tools	● Github ● IDE: Netbeans / Visual Studio	● Stores version updates ● Easy to use ● Team Familiarity.

However, team SQeaL had decided to use tools that are provided from school UW-ITConnect therefore, the tools had been changed by following in figure 2.2 “Second Tool Decision”.

Figure 2.2: Second Tool Decision

TOOL	Team SQeaL choice	Reason
Database Management System	MySQL managed through the phpMyAdmin administration package	● Free, cost-effective ● Supported through UW IT Connect ● Various resources helping learn about the DBMS. ● A GUI makes learning curve smaller.
Graphical User Interface/Website	Wordpress	● Offers a somewhat user friendly interface, needed since no one in the team has GUI building experience. ● Supported through UW’s student servers with documentation from IT Connect ● UW tutorials walked through process of linking to MySQL and student website hosted by UW’s

		servers.
Hosting Service	UW Shared Web Hosting for students	● Integrates with existing Linux based student accounts that can be accessed through an SSH client ● Fully supported and documented integration with MySQL and Wordpress ● Specific, detailed tutorials made setup from the Linux command line to linkage of related services an easy process. ● Free
Additional Tools	● Github ● IDE: any text editor, such as Notepad++, or code windows in Wordpress and MySQL	● Stores version updates ● Easy to use ● Team Familiarity.

After the tooling had changed, team SQeaL had put decent efforts to figure out how to customize our UI tool, which is "Wordpress". However, it was really hard to customize the template that "Wordpress" provides. Therefore, team SQeaL decided to build User Interface from scratch but stick with the hosting domain with student web-hosting and MySQL and PhpMyAdmin that school ITConnect team provided. In figure 2.3 "Final Tooling Decision" provides the last update for tooling decision that team SQeaL finally decided to use.

Figure 2.3: Final Tooling Decision

TOOL	**Team SQeaL choice**	**Reason**
Database Management System	**MySQL** managed through the **phpMyAdmin** administration package	● Free, cost-effective ● Supported through UW IT Connect ● Various resources helping learn about the DBMS. ● A GUI makes learning curve smaller.
Graphical User Interface/Website	**Graphic User Interface** HTML/CSS/JAVASCRIPT **Server Side Language** PHP 8	● Will be explain detail on Section "CREATE DATABASE".
Hosting Service	UW Shared Web Hosting for students	● Integrates with existing Linux based student

		accounts that can be accessed through an SSH client ● Specific, detailed tutorials made setup from the Linux command line to linkage of related services an easy process. ● Free
Additional Tools	● IDE: Atom (html,css,js,php) ● Google Drive ● phpMyAdmin (Easy to import table Written in mySQL file)	● Easy to use ● Appropriate IDE handling HTML/CSS/Javascript/PHP

UPDATE FROM ITERATION 3

The updates that team SQeaL had successfully extend the features of Dungeons and Dragons web application. The features are as known as queries that contains meaningful data as point of view for dungeon master.

Following Features had updated

- Allow user to type specific Campaign name to retrieve the episodes that is involved in the campaign name.
- Allow user to type specific Campaign name to retrieve players who are involved in the campangin.
- Allow user to type Campaign owner(Dungeon Master) to retrieve Campaign information.
- Allow user to type Campaign name with its episode number to retrieve monsters in that campaign and episode.
- Allow user to type Campaign name with episode number to retrieve Non Player Character associated with that campaign of the episode.
- Allow user to type Campaign name to retrieve number of dropped item from monster.
- Allow user to type Player id and name to retrieve Character information played by that player.
- Allow user to type dates in certain format to retrieve all players who join that date till now.
- Display all players who are not involved in any Campaign.

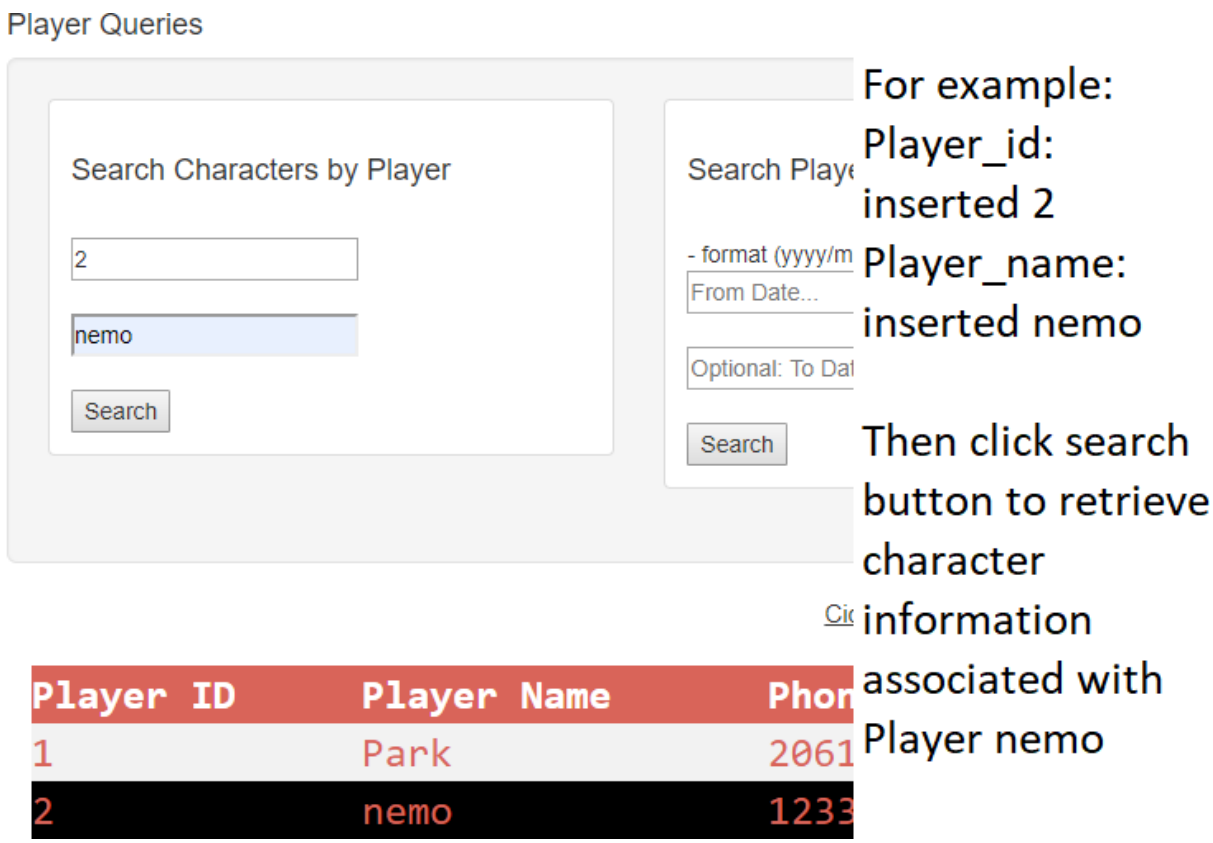
All of these extension that Team SQeaL had implemented can be seen in

http://students.washington.edu/jhpp114/projec/index.php

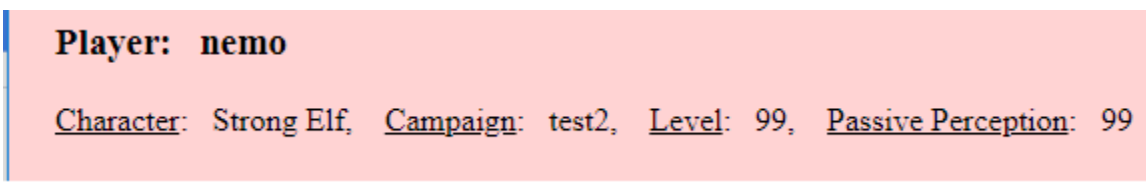
by going through Menu “DM Query”.

Figure 3.1 “Search Query” will explain the additional features with images.

Figure 3.1: Search Query



The following result will de display through user interface



The data that user retrieve has to be valid, otherwise it will just display 0 information. However, team SQeaL believe that these search features will improve the conveniency for Dungeon Master.

DISCUSS

This section will provide discussion about team SQeaL's

- Database design and its result.
- Evaluate the project (Dungeons and Dragon)
- Things that can be implemented better.
- How team SQeaL decided to populate the data for our database
- Testing the Dungeons and Dragons website.

DATABASE DESIGN WITH RESULT

Team SQeaL had approached the database design by start of drawing Entity Relationship Diagram, followed by Relational Model and implementation based on Relational Model. The database design, itself was great and team SQeaL has no doubt that the Creation of Table based on Relational Model is not wrong at all because I had tested in LocalHost where uses InnoDB storage engine and all the constraints (foreign key and Update cascade to delete cascade) works just team SQeaL had expected. However, student's PhpMyAdmin and MySQL that school provides their MySQL's engine is defaulted to MyISAM and InnoDB is set to NULL, the table.sql file goes into MyISAM storage engine without any option, which do not read any constraints and cascades. Eventually team SQeaL had to hard code to set the restriction on delete and insert.

EVALUATE PROJECT

Evaluate the amount of time that team SQeaL put on the project is at least full 9 days. The table "Time Spent" will provide the date and things that team SQeaL had worked on after decided to build Dungeons and Dragon web application from scratch.

Date	**Work**
March 02, 2019	● Understanding XAMPP ● Connect to Apache ● Learn about LocalHost ● Connect XAMPP LocalHost to PhpMyAdmin ● put database table created through MySQL into PhpMyAdmin (storage engine InnoDB for local host)
March 03, 2019	● Learn HTML ● Learn CSS ● Learn Javascript ● Learn php ● put the codes into localHost and connect the php to localHost
March 04, 2019	Implement UI ● Index.php ● main.php ● geo.php ● npc.php ● monster.php
March 05, 2019	Implement Back-end ● Connect the php file to LocalHost server ● Create form using HTML tag and connect to php file ● Implement query
March 06, 2019	Hosting to Student Server ● Download MySQL through student server ● Download PhpMyAdmin through student server ● Change the server connect to local to student

	server ● Host the project into student website.
March 07, 2019	Storage Engine Difference ● Learn difference between InnoDB and MyISAM storage engine. ● Check UW-ITConnect's MySQL Version (5.5) ● Check Storage Engine that UW-ITConnect's MySQL provides. ● Learned UW-ITConnect MySQL does not provide InnoDB (set to NULL), instead default it into MyISAM storage engine. ● Implement the constraints and rejections for insert and deletion, so the server can reject the data transfer from UI to database.
March 08, 2019	Continued Implement Back-End ● Keep working on Back-End implementation ● Finished it ● Modify User Interface
March 15, 2019	Clean the Code ● Erase unnecessary columns
March 16, 2019	Add More Meaningful SQL ● Implement it in Front-end ● Implement it in Back-end

THINGS WENT RIGHT

This section will provide things that went well during implementing the team SQeaL's final project. The things that went right for team SQeaL's project was implementing php, connecting the student server with PhpMyAdmin with MySQL and host it into student server. Team SQeaL was worrying about setting the restriction on insert and delete, however surprisingly it works and echo out the message

THINGS WENT WRONG

This section will provide the things that went wrong for team SQeaL's project. Team SQeaL had trouble due to storage engine and it was hard to fix the problem. The problem happened because the tool that team SQeaL decided was keep changed, it was hard to research about the tool that we are going to use. Due to lack of research about tool and its providing storage engine, team SQeaL had to identify and analyze the problems related to constraints and cascade, which evntually led team SQeaL to hardcode all the possible restrictions on both insert and delete.

EXTRA FEATURES COULD BE IMPLEMENTED

This section provides the extra features for team SQeaL's final project if we had extra time. There are few possible features that team SQeaL could implement, which are

- Log-in system, that allow to identify certain Dungeon Master
- Use InnoDB storage engine instead of MyISAM engine, which will be much more easy to implement since InnoDB understands referential integrity and MyISAM is not.
- Adopt the Objective Oriented Programming style on php codes, so one function can handle all other code.
- Read Dungeons and Dragons Dungeon Master guide and understand about the game.

Team SQeaL will focus on reading the guide book and understand the game and gain knowledges about dungeon master's position so it would be easy to implement the additional features.

POPULATE THE DATA TO DATABASE

Team SQeaL had decided to implement realistically based on our understanding of Dungeon Master's task, which is managing the game and players who are involved in the campaign. Therefore, populating the data into database will be inserted by the user using insert command through User Interface.

TESTING DUNGEONS AND DRAGONS WEB APPLICATION

Team SQeaL put a lot of time on testing the web application because the constraints about foreign key and update on cascade to delete on cascade was not allow in the storage engine that team SQeaL was using. Therefore, team SQeaL's solution to keep the relation among the tables was giving the restrictions to the web application user on server side.

Explanation with figures are provided in this report Section "WEB APPLICATION IMPLEMENTATION".

NORMALIZATION

The purpose of normalization is minimize the data redundancy, which can give effect on the memory on secondary storage, such as hard-disk. Since team SQeaL already has Entity relational diagram and its associated Relational Model, normalizing the database was essential. The following are all of the functional dependencies for the final table.

PLAYER table

Player_id -> Name

Player_id -> Phone_num

Player_id -> Email

Player_id -> Join_date

Since all the non-prime attributes are depending on the prim attribute (player_id), and there are no multi-value or composite value, PLAYER table is in third normal form.

CHARACTER table

Character_name -> Player_id

Character_name -> Charac_class

Character_name -> Charac_level

Character_name -> Charac_passive_perception

Since all attributes are fully dependent only in Character_name, and there are no multi-value or composite value, therefore the CHARACTER table is in third normal form.

ITEM table

Item_num -> Character_name

Item_num -> Item_name

Item_num -> Value

Item_num -> Item_count

All the attributes on Items are fully dependent by only Item_num, and there are no multi-value or composite value, therefore the ITEM table is in third normal form.

MAP table

Map_name - > Map_number

All the attributes on Map is fully dependent on only Map_name (prime attribute), and there are no multi-value or composite value, therefore the MAP table is in third normal form.

CAMPAIGN table

Campaign_name -> Player_id

Campaign_name -> Campaign_map_name

Campaign_name -> Cam_master

All of the attributes on CAMPAIGN are fully depends on only Campaign_name (prime attribute), and there are no multi-value or composite value, therefore CAMPAIGN table is in third normal form.

POINT_OF_INTEREST table

Location_num -> Location_name

Location_num -> Map_location_name

All the attributes are only depends on Location_num (prime attribute), therefore the POINT_OF_INTEREST table is in third normal form.

EPISODE table

Episode_num -> Campai_name

Episode_num -> Epsidoe_location

All the attributes are only dependent on prime attribute (Episode_num), and there are no multi-value or composite value, therefore the Episode table is in third normal form

NON_PLAYER_CHARACTER table

Npc_name -> Npc_class

All the attributes are only dependent on prime attribute (Npc_name), and there are no multi-value or composite value, therefore the NON_PLAYER_CHARACter is in third normal form.

EPISODE_NPC

Npc_id -> Epi_npc_name

Npc_id -> Epi_num_npc

Npc_id -> Npc_level

Npc_id -> Quest_reward

All the attributes are only dependent on prime attribute (Npc_id), and the table is already in 1NF and 2NF therefore the EPISODE_NPC is third normal form.

MONSTER table

Monster_name -> Monster_level

Monster_name -> Monster_type

All the attributes are only dependent on prime attribute (Monster_name) and the tables are already in first normal form and second normal form and there are no transitive dependency, therefore MONSTER table is in third normal form.

EPISODE_MONSTER table

Monster_id -> Monster_name

Monster_id -> Monster_level_epi

Monster_id -> Drop_item

Monster_id -> Monster_epi

All the attributes are only dependent on prime attribute (Monster_id) and the tables are already meet first normal form and second normal form. The table does not have transitive dependency, therefore EPISODE_MONSTER table is in the third normal form.

Team SQeaL's relational model is already form in third normal form. However, since the purpose of normalization is minimize the data redundancy. Therefore, one possible attribute that team SQeaL can remove from relational model is Monster level in Monster Table and Monster level in Episode Monster table. Although these two

attributes are different, but it is still unnecessary to ask user to insert monster level in monster table and ask user again to type monster level in episode monster.

SQL QUERY STATEMENT

Team SQeaL's project is combination of HTML combined with php mysqli::query statement. Team SQeaL **decided to copy just the sql part to simplify the report which means, the domain checks and value check will not be stated on this table,** since all the codes that had been used for project will be submitted separately.

The sql for **create database table is already on the report**,

Sections

- **"DATABASE TABLE IMPORTED TO PHPMYADMIN"**
- **"DATABASE TABLE CONVERTED AFTER MYISAM STORAGE ENGINE".**

	SQL Statement	**Purpose**
PLAYER	$sql = "SELECT Player_id, Player_name, Player_phone, Player_email, Player_join_date FROM PLAYER";	read the data just about player from database relation PLAYER that user inserted and display on User Interface.
	$Player_name = $_POST['Player_name']; $Player_phone = $_POST['Player_phone']; $Player_email = $_POST['Player_email']; $Player_join_date =$_POST['Player_join_date']; $sql = "INSERT INTO PLAYER (Player_name,Player_phone,Player_email,Player_join_date,Player_leave_date) VALUES ('$Player_name', '$Player_phone', '$Player_email', '$Player_join_date')";	Read the data that player inserted through User Interface and convert that that to php mysqli:sql and save it into database at PLAYER table.

	if (!empty($Player_id)) {
	 $sqlResult = "SELECT Charac_player_id FROM PCHARACTER";
	 $result = $mysqli_connection->query($sqlResult);
	 $fore1 = TRUE;
	 if ($result->num_rows > 0) {
	 while ($row = $result->fetch_assoc()) {
	 if ($row["Charac_player_id"] ==
		Due to the foreign key restriction, prevent the user to delete Player first even though the player has character.
		
		 Check if the Player has character and if not delete it from Player that is matching with Player_id from PLAYER table on the database.

	$Player_id) {
	$fore1 = FALSE;
	break;
	} else {
	$fore1 = TRUE;
	}
	}
	} else {
	$fore1 = TRUE;
	}
	}
	
	if (!empty($Player_id)) {
	$sqlResult = "SELECT Player_id FROM PLAYER";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Player_id"] == $Player_id) {
	$sql = "DELETE FROM PLAYER WHERE Player_id = $Player_id";
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	} else {
	$fore = FALSE;
	}
	}	

CHARACTER	$sql = "SELECT Character_name, Charac_player_id, Charac_class, Charac_level, Charac_passive_perception FROM PCHARACTER";	read the CHARACTER data from PCHARACTER table from database and display to user interface.
	$Character_name = $_POST["Character_name"]; $Charac_player_id = $_POST["Charac_player_id"]; $Charac_class = $_POST["Charac_class"]; $Charac_level = $_POST["Charac_level"]; $Charac_passive_perception =	Converting all user inserts to mysqli:sql read the data stored in PLAYER table to check if the Player had been inserted first, if not then reject the insert for Character.

	$_POST["Charac_passive_perception"];
	
	if (!empty($Charac_player_id)) {
	  $sqlResult = "SELECT Player_id FROM PLAYER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Player_id"] == $Charac_player_id) {
	  $fore = TRUE;
	  break;
	  } else {
	  $fore = FALSE;
	  }
	  }
	}	

	$sql = "INSERT INTO PCHARACTER (Character_name, Charac_player_id, Charac_class, Charac_level, Charac_passive_perception) VALUES ('$Character_name', '$Charac_player_id', '$Charac_class', '$Charac_level', '$Charac_passive_perception')";
	header("refresh:3; url=main.php");
	if (!mysqli_query($mysqli_connection, $sql)) {
	  echo 'Not Inserted';
	  echo "
	";
	} else {
	  echo "Inserted";
	  echo "
	";
	}
		Insert the data for Character that inserted from user from UI into database PCHARACTER table.

	if (!empty($Character_name)) {
	  $sqlResult = "SELECT Character_name FROM PCHARACTER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
		Delete the user inserted Character name from the database table. If the inserted character name does not exist on the table then simply disconnect the connection and return to the page.

	if ($row["Character_name"] == $Character_name) {
	$sql = "DELETE FROM PCHARACTER WHERE Character_name = $Character_name";
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	} else {
	$fore = FALSE;
	}
	}	

	if (!empty($Charac_player_id)) {
	$sqlResult1 = "SELECT Player_id FROM PLAYER";
	$result1 = $mysqli_connection->query($sqlResult1);
	$fore1 = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result1->fetch_assoc()) {
	if ($row["Player_id"] == $Charac_player_id) {
	$sql = "DELETE FROM PCHARACTER WHERE Charac_player_id = $Charac_player_id";
	$fore1 = TRUE;
	break;
	} else {
	$fore1 = FALSE;
	}
	}
	} else {
	$fore1 = FALSE;
	}
	}
		Loop through the matching Player_id that had assigned as foreign key in Character table and once it match then delete the foreign key from the database PCHARACTER table. If it fails to find it then simply return false.

ITEM	$sql = "SELECT Item_id, Item_name, Item_owner_character, Item_value, Item_count FROM ITEM";	Retrieve data inserted in ITEM table in the database and display it to user.
	if (!empty($Item_owner_character)) { $sqlResult = "SELECT Character_name FROM PCHARACTER";	foreign key restriction, loop through the inserted item_owner_character from CHARACTER and check if it is matching or not, if not disconnect the connection from database.

	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	while ($row = $result->fetch_assoc()) {
	if ($row["Character_name"] == $Item_owner_character) {
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	}	

	$sql = "INSERT INTO ITEM (Item_name, Item_owner_character, Item_value, Item_count) VALUES ('$Item_name', '$Item_owner_character', '$Item_value', '$Item_count')";	Insert data that had inserted from user through UI and put that data to ITEM table in Database.

	$Item_id = $_POST["Item_id"];
	
	if (!empty($Item_id)) {
	$sqlResult = "SELECT Item_id FROM ITEM";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Item_id"] == $Item_id) {
	$sql = "DELETE FROM ITEM WHERE Item_id = $Item_id";
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	} else {
	$fore = FALSE;
	}
	}
		Since the item that are inserted are verified that there is character owner, therefore simply delete it from database ITEM table that is matching with the item_id.

MAP	$sql = "SELECT Map_number, Map_name FROM MAP";	Retrieve all data from MAP table from database and display it to user through UI

	$Map_number = $_POST["Map_number"]; $Map_name = $_POST["Map_name"]; $sql = "INSERT INTO MAP (Map_number, Map_name) VALUES ('$Map_number', '$Map_name')";	Convert the User input to mysqli:sql and insert it into MAP table database.

	$Map_number = $_POST["Map_number"];
	$Map_name = $_POST["Map_name"];
	
	if (!empty($Map_name)) {
	  $sqlResult = "SELECT Cam_map_name FROM CAMPAIGN";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Cam_map_name"] == $Map_name) {
	  $fore = FALSE;
	  break;
	  } else {
	  $fore = TRUE;
	  }
	  }
	  } else {
	  $fore = TRUE;
	  }
	}
		Foreign key constraint, check give out restriction if map is trying to get deleted if CAMPAIGN is containing that map and CAMPAIGN still exist.

	if (!empty($Map_number)) {
	  $sqlResult = "SELECT Map_number FROM MAP";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Map_number"] == $Map_number) {
	  $sql = "DELETE FROM MAP WHERE Map_number =
		Check the input that user typed in and loop though the data in MAP table inside database and delete the matching data.

	$Map_number";
	$fore1 = TRUE;
	break;
	} else {
	$fore1 = FALSE;
	}
	}
	} else {
	$fore1 = FALSE;
	}
	}	

	if (!empty($Map_name)) {
	$sqlResult = "SELECT Map_name FROM MAP";
	$reuslt = $mysqli_connection->query($sqlResult);
	$fore2 = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Map_name"] == $Map_name) {
	$sql = "DELETE FROM MAP WHERE Map_name = $Map_name";
	$fore2 = TRUE;
	break;
	} else {
	$fore2 = FALSE;
	}
	}
	} else {
	$fore2 = FALSE;
	}
	}
		Loop through inserted Map_name by user from UI at Database MAP table and once it matches then delete the data.

CAMPAIGN	$sql = "SELECT Campaign_name, Cam_map_name, Cam_master, Cam_player FROM CAMPAIGN";	Read all the data that are store in CAMPAIGN data table and display it to user through UI

	$Campaign_name = $_POST["Campaign_name"];
	$Cam_map_name = $_POST["Cam_map_name"];
	$Cam_master = $_POST["Cam_master"];
	$Cam_player = $_POST["Cam_player"];
	
	// camp map name foreign key Constraint
	if (!empty($Cam_map_name)) {
		Foreign key constraint check, loop through MAP table and check if the Map is already inserted before insert the data to Campaign. If the map data is not inserted yet, then simply return false else return true.

	$sqlResult = "SELECT Map_name FROM MAP";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	while ($row = $result->fetch_assoc()) {
	if ($row["Map_name"] == $Cam_map_name) {
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	}	

	$sql = "INSERT INTO CAMPAIGN (Campaign_name, Cam_map_name, Cam_master, Cam_player) VALUES ('$Campaign_name', '$Cam_map_name', '$Cam_master', '$Cam_player')";	Insert the data that user had inserted through UI then store the data into CAMPAIGN table in the database.

	if (!empty($Campaign_name)) {
	$sqlResult = "SELECT Campai_name FROM EPISODE";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Campai_name"] == $Campaign_name) {
	$fore = FALSE;
	break;
	} else {
	$fore = TRUE;
	}
	}
	} else {
	$fore = TRUE;
	}
	}
	if (!$fore) {
	header("refresh:3; url=geo.php");
	echo "Delete the Episode First Before you Delete Campaign";
	die();
	}
		Delete restriction. Check all possible tables that contains the Campaign information before delete the Campaign. If there are some table that requires to have campaign information, then reject the delete and throw message.

	if (!empty($Campaign_name)) {
	  $sqlResult = "SELECT Campaign_name FROM CAMPAIGN";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Campaign_name"] == $Campaign_name) {
	  $sql = "DELETE FROM CAMPAIGN WHERE Campaign_name = $Campaign_name";
	  $fore1 = TRUE;
	  break;
	  } else {
	  $fore1 = FALSE;
	  }
	  }
	  } else {
	  $fore1 = FALSE;
	  }
	}
		Loop through all the data that are in the CAMPAIGN database and check the user inserted data, in this case Campaign_name is matching with the data that are stored in the CAMPAIGN database table, if it matches then delete.

	if (!empty($Cam_map_name)) {
	  $sqlResult = "SELECT Cam_map_name FROM CAMPAIGN";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore2 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Cam_map_name"] == $Cam_map_name) {
	  $sql = "DELETE FROM CAMPAIGN WHERE Cam_map_name = $Cam_map_name";
	  $fore2 = TRUE;
	  break;
	  } else {
	  $fore2 = FALSE;
	  }
	  }
	  } else {
	  $fore2 = FALSE;
	  }
	}
		Loop through all the data that are in the CAMPAIGN database and check the user inserted data, in this case Cam_map_name is matching with the data that are stored in the CAMPAIGN database table, if it matches then delete.

	if (!empty($Cam_player)) { $sqlResult = "SELECT Cam_player	Loop through all the data that are in the CAMPAIGN database and check

	FROM CAMPAIGN";
	$result = $mysqli_connection->query($sqlResult);
	$fore3 = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Cam_player"] == $Cam_player) {
	$sql = "DELETE FROM CAMPAIGN WHERE Cam_player = $Cam_player";
	$fore3 = TRUE;
	break;
	} else {
	$fore3 = FALSE;
	}
	}
	} else {
	$fore3 = FALSE;
	}
	}
		the user inserted data, in this case Cam_player is matching with the data that are stored in the CAMPAIGN database table, if it matches then delete.

	$sql = "SELECT Campaign_name, Character_name FROM CAMPAIGN, PCHARACTER WHERE Cam_player = Charac_player_id";	Simply display to user information where the campaign that has character.
POINT_OF_INTEREST	$sql = "SELECT Location_num, Location_name, Map_location_name FROM POINT_OF_INTEREST, MAP WHERE POINT_OF_INTEREST.Map_location_name = MAP.Map_name";	Display it to user information that point of interest (means like secret area of map) with its located MAP.

	if (!empty($Map_location_name)) {
	$sqlResult = "SELECT Map_name FROM MAP";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	while ($row = $result->fetch_assoc()) {
	if ($row["Map_name"] == $Map_location_name) {
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	}
		Before insert the Point_of_location, make sure the map_name that is in the MAP is exist, if it does not exist then return false if the map_name on Map exist then insert it into Point_OF_Interest data table in the database.

	$sql = "INSERT INTO POINT_OF_INTEREST(Location_name, Map_location_name) VALUES ('$Location_name', '$Map_location_name')";	

	if (!empty($Location_num)) {
	  $sqlResult = "SELECT Location_num FROM POINT_OF_INTEREST WHERE Location_num = $Location_num";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
	  // if ($reslut->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Location_num"] == $Location_num) {
	  $sql = "DELETE FROM POINT_OF_INTEREST WHERE Location_num = $Location_num";
	  $fore1 = TRUE;
	  break;
	  } else {
	  $fore1 = FALSE;
	  }
	  }
	  // } else {
	  // $fore1 = FALSE;
	  // }
	}
		Check if the location_num that is inserted by user is existing in the POINT_OF_INTEREST and if it exist then delete the matching data from database.

	if (!empty($Location_name)) {
	  $sqlResult = "SELECT Episode_location FROM EPISODE";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore3 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Episode_location"] == $Location_name) {
	  $fore3 = FALSE;
	  break;
	  }
	  }
	  } else {
	  $fore3 = TRUE;
	  }
	}
		Test before actually delete happen. Set up the rejection based on the foreign key in the relational model.
		
		Loop through location_name that is store in Episode table database and if the location_name is exist in the episode then reject the delete and throw message to user to delete episode first.

	if (!empty($Location_name)) {
	  $sqlResult = "SELECT Location_name FROM POINT_OF_INTEREST";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore2 = TRUE;
	  if ($result->num_rows > 0) {
	  while($row = $result->fetch_assoc()) {
	  if ($row["Location_name"] == $Location_name) {
	  $sql = "DELETE FROM POINT_OF_INTEREST WHERE Location_name = $Location_name";
	  $fore2 = TRUE;
	  break;
	  } else {
	  $fore2 = FALSE;
	  }
	  }
	  } else {
	  $fore2 = FALSE;
	  }
	}
		Loop through data for location name based on the user inserted and if the matching data is exist on the database then delete it.

EPISODE	$sql = "SELECT Episode_num, Campai_name, Episode_location FROM EPISODE";	Retrieve all the information about Episode from EPISODE table in the database.

	if (!empty($Campai_name)) {
	  $sqlResult = "SELECT Campaign_name FROM CAMPAIGN";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Campaign_name"] == $Campai_name) {
	  $fore = TRUE;
	  break;
	  } else {
	  $fore = FALSE;
	  }
	  }
	}
		Loop through all user inserted Campai_name which are stored in CAMPAIGN database and if that campaign does not exist, then reject the insert.

	$sql = "INSERT INTO EPISODE (Campai_name, Episode_location) VALUES ('$Campai_name', '$Episode_location')";	Insert the user inserted data into EPISODE table in the database.

	if (!empty($Episode_num)) {
	  $sqlResult = "SELECT Monster_epi FROM EPISODE_MONSTER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Monster_epi"] == $Episode_num) {
	  $fore = FALSE;
	  break;
	  } else {
	  $fore = TRUE;
	  }
	  }
	  } else {
	  $fore = TRUE;
	  }
	}
		Check foreign keys, loop through all episode_num that is assigned to monster table and reject the delete the episode if monster is in that episode.

	if (!empty($Episode_num)) {
	  $sqlResult = "SELECT Epi_num_npc FROM EPISODE_NPC";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore2 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Epi_num_npc"] == $Episode_num) {
	  $for2 = FALSE;
	  break;
	  } else {
	  $fore2 = TRUE;
	  }
	  }
	  } else {
	  $fore2 = TRUE;
	  }
	}
		Check foreign key, loop through every data that for Episode_num that are store in EPISODE_NPC, if certain epic is already assigned in certain episode then reject the delete and throw message to user to delete the npc first before delete episode.

	if (!empty($Episode_num)) {
	  $sqlResult = "SELECT Episode_num FROM EPISODE";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result-
		Loop through to search the matching input inserted from user and delete the data from episode once it matches.

	>fetch_assoc()) {
	  if ($row["Episode_num"] == $Episode_num) {
	  $sql = "DELETE FROM EPISODE WHERE Episode_num = $Episode_num";
	  $fore1 = TRUE;
	  break;
	  } else {
	  $fore1 = FALSE;
	  }
	  }
	  } else {
	  $fore1 = FALSE;
	  }
	}	

NON_PLAYER_CHARACTER	$sql = "SELECT Npc_name, Npc_class FROM NON_PLAYER_CHARACTER";	Read all the data that is stored in NON_PLAYER_CHARACTER and display it to user through UI.
	$Npc_name = $_POST["Npc_name"]; $Npc_class = $_POST["Npc_class"]; $sql = "INSERT INTO NON_PLAYER_CHARACTER (Npc_name, Npc_class) VALUES ('$Npc_name', '$Npc_class')";	Simply insert the user inserted data into NON_PLAYER_CHARACTER table in the database.

	if (!empty($Npc_name)) {
	  $sqlResult = "SELECT Epi_npc_name FROM EPISODE_NPC";
	  $reuslt = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Epi_npc_name"] == $Npc_name) {
	  $fore = FALSE;
	  break;
	  } else {
	  $fore = TRUE;
	  }
	  }
	  } else {
	  $fore = TRUE;
	  }
	}
		Foreign key purpose, read all the data that matches with User inserted Npc_name from EPISODE_NPC table, once it matches reject the delete and throw message to delete episode_npc before delete it from NPC table in database.

	if (!empty($Npc_name)) {
	  $sqlResult = "SELECT Npc_name FROM NON_PLAYER_CHARACTER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
	  if ($result->num_rows > 0) {
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Npc_name"] == $Npc_name) {
	  $sql = "DELETE FROM NON_PLAYER_CHARACTER WHERE Npc_name = $Npc_name";
	  $fore1 = TRUE;
	  break;
	  } else {
	  $fore1 = FALSE;
	  }
	  }
	  } else {
	  $fore1 = FALSE;
	  }
	}
		Loop through all the Npc_name store in NON_PLAYER_CHARACTER table in database and search the matching data with user inserted, if it matches then delete else return false.

EPISODE_NPC	$sql = "SELECT Npc_id, Epi_npc_name, Epi_num_npc, Npc_level, Quest_reward FROM EPISODE_NPC";	Get all the data store in EPISODE_NPC table in database.

	if (!empty($Epi_npc_name)) {
	  $sqlResult = "SELECT Npc_name FROM NON_PLAYER_CHARACTER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Npc_name"] == $Epi_npc_name) {
	  $fore = TRUE;
	  break;
	  } else {
	  $fore = FALSE;
	  }
	  }
	}
		Foreign key reject, if the user try to insert NON_PLAYER_CHARACTER into EPISODE_NPC, which means assign npc certain episode and if the NPC does not exist in NON_PLAYER_CHARACTER then reject the insert and throw message to user to insert the NPC first to NON_PLAYER_CHARACTER.

	if (!empty($Epi_num_npc)) {
	  $sqlResult = "SELECT Episode_num FROM EPISODE";
	  $result = $mysqli_connection-
		Foreign key reject, if user try to assign existing npc into certain Episode, loop through all episode store in episode database and check

	>query($sqlResult);
	$fore1 = TRUE;
	while ($row = $result->fetch_assoc()) {
	if ($row["Episode_num"] == $Epi_num_npc) {
	$fore1 = TRUE;
	break;
	} else {
	$fore1 = FALSE;
	}
	}
	}
		the episode is existing or not, if the episode does not exist then throw message to create the episode first and return false.

	$sql = "INSERT INTO EPISODE_NPC (Epi_npc_name, Epi_num_npc, Npc_level, Quest_reward) VALUES ('$Epi_npc_name', '$Epi_num_npc', '$Npc_level', '$Quest_reward')";	Simply insert the data into EPISODE_NPC table in database.

	if (!empty($Npc_id)) {
	$sqlResult = "SELECT Npc_id FROM EPISODE_NPC";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Npc_id"] == $Npc_id) {
	$sql = "DELETE FROM EPISODE_NPC WHERE Npc_id = $Npc_id";
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	} else {
	$fore = FALSE;
	}
	}
		Loop through the database table EPISODE_NPC and look for the matching data that player had inserted, if the data is matching then delete it from database table else simply return false

NPC_IN_CAMPAIGN	$sql = "SELECT Campai_name, Epi_npc_name FROM EPISODE_NPC, EPISODE WHERE Epi_num_npc = Episode_num";	Find the data where episode that contains npc, display campaign name and npc name.
MONSTER	$sql = "SELECT Monster_name, Monster_level, Monster_type FROM	Display all data that store in MONSTER table in the database

	MONSTER";	through user interface.
	$Monster_name = $_POST["Monster_name"]; $Monster_level = $_POST["Monster_level"]; $Monster_type = $_POST["Monster_type"]; $sql = "INSERT INTO MONSTER (Monster_name, Monster_level, Monster_type) VALUES ('$Monster_name', '$Monster_level', '$Monster_type')";	Convert the user input to mysqli:sql form and insert it into MONSTER table in database.
MONSTER_IN_EPISDOE	$sql = "SELECT Monster_epi, Monster_id, Monster_name, Monster_level_epi, Drop_item FROM EPISODE_MONSTER";	Display all the data that is stored in MONSTER_IN_EPSIODE in database.

	$Monster_epi = $_POST["Monster_epi"];
	$Monster_name = $_POST["Monster_name"];
	$Monster_level_epi = $_POST["Monster_level_epi"];
	$Drop_item = $_POST["Drop_item"];
	
	if (!empty($Monster_epi)) {
	  $sqlResult = "SELECT Episode_num FROM EPISODE";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore = TRUE;
	  while ($row = $result->fetch_assoc()) {
	  if ($row["Episode_num"] == $Monster_epi) {
	  $fore = TRUE;
	  break;
	  } else {
	  $fore = FALSE;
	  }
	  }
	}
		Foreign key constraint, loop through the user inserted Monster_episode and check in EPISODE table data, see if the user inserted data is matching with, (means check episode existence), if it does not exist then return false else return true

	if (!empty($Monster_name)) {
	  $sqlResult = "SELECT Monster_name FROM MONSTER";
	  $result = $mysqli_connection->query($sqlResult);
	  $fore1 = TRUE;
		Foreign key check, if the monster exist in the MONSTER table before assign the monster an episode, if the monster does not exist, then throw message to user to insert the monster first before insert monster

	while ($row = $result->fetch_assoc()) {
	if ($row["Monster_name"] == $Monster_name) {
	$fore1 = TRUE;
	break;
	} else {
	$fore1 = FALSE;
	}
	}
	}
		to certain episode.

	$sql = "INSERT INTO EPISODE_MONSTER (Monster_epi, Monster_name, Monster_level_epi, Drop_item) VALUES ('$Monster_epi', '$Monster_name', '$Monster_level_epi', '$Drop_item')";	Read all the data that user inserted through UI and sent the data to EPISODE_MONSTER table in database.

	if (!empty($Monster_id)) {
	$sqlResult = "SELECT Monster_id FROM EPISODE_MONSTER";
	$result = $mysqli_connection->query($sqlResult);
	$fore = TRUE;
	if ($result->num_rows > 0) {
	while ($row = $result->fetch_assoc()) {
	if ($row["Monster_id"] == $Monster_id) {
	$sql = "DELETE FROM EPISODE_MONSTER WHERE Monster_id = $Monster_id";
	$fore = TRUE;
	break;
	} else {
	$fore = FALSE;
	}
	}
	} else {
	$fore = FALSE;
	}
	}
		Loop through the EPISODE_MONSTER table in database and if the user inserted Monster_id is exist in the table then delete the data from database.

MONSTER IN CAMPAIGN	$sql = "SELECT Campai_name, Monster_name FROM EPISODE, EPISODE_MONSTER WHERE EPISODE_MONSTER.Monster_epi = EPISODE.Episode_num";	Simply display the Campaign name and its associated monster name to user through UI.

DM_QUERY		
Campaign Stats	$sql = "SELECT Campaign_name, Cam_map_name, Cam_master FROM CAMPAIGN";	Display information related to campaign that user may needed in order to use the search bar.
Search Episode by typing Campaign_name	$result = $mysqli_connection->query("SELECT Episode_num, Episode_location, Cam_map_name FROM EPISODE, CAMPAIGN WHERE Campaign_name = Campai_name AND Campaign_name = ('$Campaign_name')");	Retrieve the Episode number, Episode location, map name from database base on user type campaign name. This query is to provide convenience by visualize the data that user may needed to see about certain campaign and its associated geography information.
Search Player & Characters by typing Campaign_name	$result = $mysqli_connection->query("SELECT PLAYER.Player_name, PCHARACTER.Character_name, PCHARACTER.Charac_level, PCHARACTER.Charac_passive_perception FROM PLAYER, PCHARACTER, CAMPAIGN WHERE CAMPAIGN.Cam_player = PCHARACTER.Charac_player_id AND PCHARACTER.Charac_player_id = PLAYER.Player_id AND CAMPAIGN.campaign_name = ('$Campaign_name')");	Retrieve Character and Player information such as (player name, character name, character level, and character passive perception) based on the inserted campaign. This query is to provide convenience by visualize the data that user may need to see about certain Characters and player name associated in specific campaign.
Search Campaign by Dungeon Master	$result = $mysqli_connection->query("SELECT Campaign_name, Cam_map_name FROM CAMPAIGN WHERE Cam_master = ('$Cam_master')");	Retrieve campaign by typing the dungeon master name. This query is to provide convenience by visualizing the information by filtering out the data, so easy to see and manage the campaign owned by certain Dungeon Master.
Epsidoe Stats	$sql = "SELECT Campai_name, Episode_num, Episode_location, Cam_map_name, Cam_master FROM EPISODE, CAMPAIGN WHERE Campai_name = Campaign_name	Display information related to episode that user may needed in order to use the search bar.

	ORDER BY Campai_name, Episode_num ASC";	
Episode Stats	$result = $mysqli_connection->query("SELECT Monster_name, Monster_level_epi, Episode_location, Drop_item FROM EPISODE, EPISODE_MONSTER WHERE Monster_epi = Episode_num AND Campai_name = ('$Campai_name') AND Episode_num = ('$Episode_num')");	Allow user to search Monster_name, Monster_level, episode location and drop item by searching specific campaign name and episode number. This query is to provide convenience to user by visualize organized data of monster name to dropped item that existed or assigned to certain campaign with certain episode.
Show NPC and Location	$result = $mysqli_connection->query("SELECT Epi_npc_name, Npc_level, Episode_location, Quest_reward FROM EPISODE, EPISODE_NPC WHERE Epi_num_npc = Episode_num AND Campai_name = ('$Campai_name') AND Episode_num = ('$Episode_num')");	Allow user to type campaign name and episode number to search its associated Npc information. This query is to provide convenience to user by visualize organize or filter the information.
Show total dropped items and quest rewards by typing campaign name	$result = $mysqli_connection->query("SELECT COUNT(Quest_reward) AS quantity FROM EPISODE, EPISODE_NPC WHERE Epi_num_npc = Episode_num AND Campai_name = ('$Campai_name')");	Allow user to type campaign name to retrieve all items and quest reward can be gain in that campaign. This query is to provide convenience to user by filtering out the data by specific campaign name.
Player Query	$sql = "SELECT Player_id, Player_name, Player_phone, Player_email, Player_join_date FROM PLAYER";	Display information related to Player that user may needed in order to use the search bar.
Search Character By typing Player id and name	$result = $mysqli_connection->query("SELECT Character_name, Charac_level, Charac_passive_perception, Campaign_name FROM PLAYER, PCHARACTER, CAMPAIGN WHERE	Allow user to type player id and player name to retrieve all character information include campaign name. This query is to provide convenience to user by filtering out the data by specific player by inserting player id and player name.

	Player_id = Charac_player_id AND Cam_player = Charac_player_id AND Player_id = ('$Player_id') AND Player_name = ('$Player_name')");	
Search Players by Join Date Range	$result = $mysqli_connection->query("SELECT Player_name, Player_phone, Player_email, Player_join_date FROM PLAYER WHERE Player_join_date BETWEEN CAST(('$Player_join_date_fr') AS DATE) AND CAST(('$Player_join_date_to') AS DATE)");	Retrieve all the players who joined the game within certain range.
Display Players Not in any Campaign	$result = $mysqli_connection->query("SELECT DISTINCT Player_name, Player_phone, Player_email, Player_join_date FROM PLAYER WHERE Player_id NOT IN (SELECT Cam_player FROM CAMPAIGN)");	Retrieve all Players who are not involving in any campaign. This query will provide huge convenience to manage players by seeing who are not part of any game.

REFERENCES

This section is providing the reference of the tools that had been used to create this reports and implementation of team SQeaL's final project in MLA format.

Tools had been used for report

Information	**Reference**
Diagrams Figures	https://www.draw.io/
Professor's oneNote	Parsons, Erika. "CSS475_Notebook." *Microsoft OneDrive - Access Files Anywhere. Create Docs with Free Office Online.*, 27 Mar. 2016, 09:46PM, onedrive.live.com/redir?resid=14B3EF2D44442C5A%211048&authkey=%21AIoh4yZlikxsQnU&page=View&wd=target%28Project.one%7Cab7073bd-

	42bc-4b9e-9bf6-4d691a0a7789%2FExample%2BDocument%2B1%7C5807e5af-314e-4ef1-9e58-9e15d2f9bc49%2F%29.
Information about MyISAM and InnoDB	ilahn, ilhanilhan1, et al. "What Are the Main Differences between InnoDB and MyISAM?" *Database Administrators Stack Exchange*, StackExchange, 3 Jan. 2011, 20:46, dba.stackexchange.com/questions/1/what-are-the-main-differences-between-innodb-and-myisam.

Tool had been used for Final Project

Tools	**Reference**
MySQL	"Install MySQL." *IT Connect*, itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/install-mysql/.
phpMyAdmin	"Installing PhpMyAdmin." *IT Connect*, itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/installing-phpmyadmin/.
SSH(vergil.u.washington.edu)	"Activating Shared Web Hosting." *IT Connect*, itconnect.uw.edu/connect/web-publishing/shared-hosting/activating-shared-web-hosting/.
SSH (Student Server)	"Connecting with SSH." *IT Connect*, itconnect.uw.edu/connect/web-publishing/shared-hosting/ssh/.
Connecting to MySQL using PHP	"Connecting to MySQL Using PHP." *IT Connect*, itconnect.uw.edu/connect/web-publishing/shared-hosting/using-mysql-on-shared-uw-hosting/connecting-to-mysql-using-php/.
Learning about Storage Engine	ilahn, ilhanilhan1, et al. "What Are the Main Differences between InnoDB and MyISAM?" *Database Administrators Stack Exchange*, StackExchange, 3 Jan. 2011, 20:46, dba.stackexchange.com/questions/1/what-are-the-main-differences-between-innodb-and-myisam.
HTML	"HTML5 Tutorial." *HTML Tutorial*, www.w3schools.com/html/.
Bootsrap	Otto, Mark, and Jacob Thornton. "Bootstrap." · *The Most Popular HTML, CSS, and JS Library in the World.*, getbootstrap.com/.
php	"PHP 5 Tutorial." *PHP 5 Tutorial*, www.w3schools.com/php/default.asp.

ACCESS TO DUNGEONS & DRAGONS WEB APPLICATION

http://students.washington.edu/jhpp114/projec/