MMORPG Database

Team Kelvin

Tyler Larican, Sean Miles, Elijah Edwards, Kylun Robbins

March 16th, 2020

Table of Contents

Version History

V1.0: Iteration 0
- Created the proposal overview with an approximate schedule.

V1.1: Iteration 1
- Added design document section containing the database's:
  - Entity-Relationship Diagram
  - Relational Data Model
- Added all relations, attributes, primary keys, foreign keys, and constraints.
- Added assumptions and explanations for design decisions.

V1.2: Iteration 2
- Added tooling assessment section containing the planned tools to use for development including the DBMS, UI, hosting service, version control, backend language, IDE, and tool for sample data generation.

V1.3: Iteration 3
- Added test data through manual insertion using SQL queries with MySQL.
- Added incorrect or invalid SQL queries to test database constraints.
- Added section explaining how the test data was acquired and a plan for generating sample data.

V1.4: Iteration 4
- Added an icon for the project on the front page.
- Updated SQL queries formatted as a table with an included purpose.
- Added explanation on how correctness of test data is ensured.
- Added explanation on how our sample data was randomly generated.
- Added normalization section in design document, explaining the form each table is in.
- Added UI section including screenshots and explanations of queries related to the functionality of the UI.
- Added project evaluation section including discussion on what went right, what went wrong, and potential changes we could make in the future.
- Added references section.

Proposal

Overview

Team Kelvin proposes a relational database for a simple Massively multiplayer online role-playing game (MMORPG). This database will be used by both developers and players of the game. It will hold data such as players, items, enemies, times, and locations. Players will be able to use queries such as:

- List the item that drops from enemy whose Mob_ID is “1234”
- List all mobs found in the location whose Location_ID is “BattleZone1”
- List all friends for the player whose Player_id is “1234”

These queries can be used by players to assist them in their gameplay. By offering an easy, built-in function for finding information on the game, players won’t have to go to other sources to find the information they need. Developers will be able to use queries such as:

- List the amount of times the mob whose Mob_ID is “1234” has killed players
- List the amount of players that are using the weapon whose Weapon_ID is “BasicSword1”
- List the most worn armor for players of class “Wizard”

These queries can be used by developers to assist in gaining information on how players are playing the game. This information such as timers and player inventory status can help with making decisions on changes and balances to the game.

Approximate Schedule

Table 1 shows the tentative schedule Team Kelvin will be following. The schedule is approximate and subject to change as the project goes on. These milestones are simply guidelines for members to reach for.

Table 1: Team Kelvin Milestone Schedule

Deliverable	Completion date	Plan post completion
Proposal	19-Jan-20	Prepare tooling document and presentation
Tooling	26-Jan-20	Prepare ER diagram and relation model
Design Documentation	02-Feb-20	Populate database and create sample queries
Populated Database	09-Feb-20	Create and test SQL statements
SQL Query Statements	16-Feb-20	Begin Normalization process
Normalization	23-Feb-20	Evaluate possible options for UI
User Interface Strategy	01-Mar-20	Prepare final deliverable and presentation
Documentation and Poster Presentation	16-Mar-20	Internal team retrospective

Work Distribution

All parts of the project will be worked on evenly by all members of Team Kelvin.

Design Document

Entities

The following are entity types chosen for the database:

- ACCOUNT: first name, last name, email, time played, # of characters
- CHARACTER: class, level, experience, name, playtime, health, strength, magika, money
- MOBS: mob ID, name, health, money to drop, exp to drop
- CONSUMABLES: item ID, description, heal total, strength buff, magic buff, item duration, item cost
- LOCATION: location ID, name, recommended location levell, # of players
- WEAPON: weapon ID, cost, name, type, strength given, magic given
- ARMOR: armor ID, cost, name, health given, strength given, magic given
- NPC: NPC ID, name, type
- QUEST: quest ID, name, money given, recommended quest level, exp given, list of mobs to kill, list of items to collect, type

Assumptions

1. An Account can have 0-7 Characters associated with it.
2. A Character can wear 0-5 pieces of Armor.
3. A Character can wield 0-2 weapons.
4. Combat occurs between 1 Character and 1 Mob at a time only (an attack hitting multiple Mobs occurs between each Mob separately).
5. Multiple Characters can interact with 1 NPC at once.

Design Decisions

As of now, the character has no inventory system, this suggests:

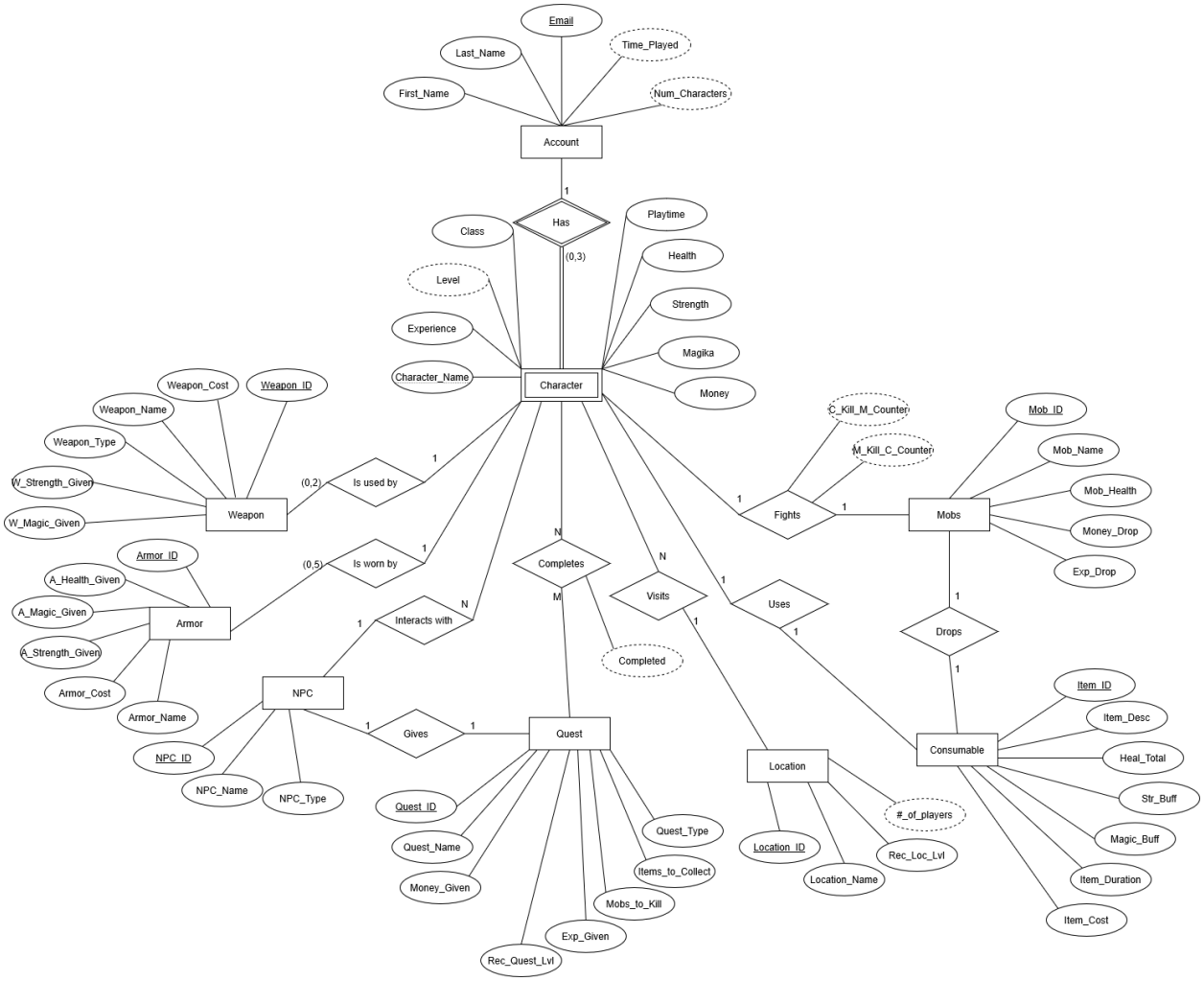
a. Consumables must be used as they are picked up
b. Characters can only swap armor
c. Characters can only swap weapons

Entity-Relationship Diagram

Figure 1: Game Database Entity-Relationship Diagram

Description

Figure 1 shows the Entity-Relationship Diagram for our MMORPG database. This diagram provides an outline of our data entities, the entities's attributes, and the relationships between different data entities.

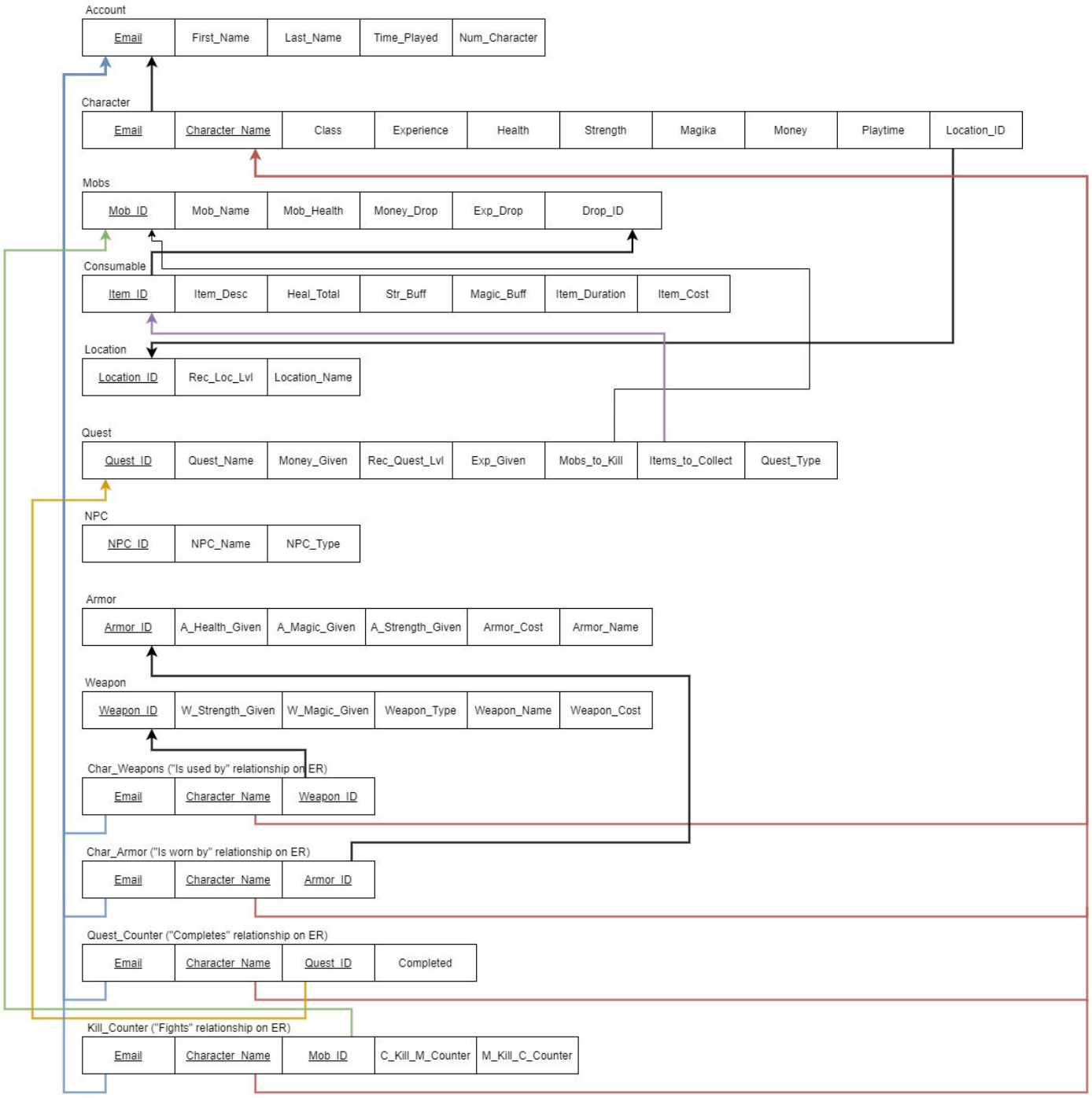


Relational Data Model

Figure 2: Game Database Relational Data Model

Description

Figure 2 shows the Relational Data Model for our MMORPG database. This model shows in more detail how the entities will interact with each other. This figure will include entities and relationships and their attributes, including primary keys and foreign keys.



Domain and Entity Integrity Constraints

The following section lists the domain and entity integrity constraints of each table's attributes. Bolded attributes represent a nullable attribute.

- Account(Email:d_{email}, First_Name:D_{string}, **Last_Name**:D_{string}, Time_Played:D_{time}, Num_Character:$D_{integer}$)
- Character(Email:d_{email}, Character_Name:D_{string}, Class:d_{class}, Experience:$D_{integer}$, Health:$D_{integer}$, Strength:$D_{integer}$, Magika:$D_{integer}$, Money:$D_{integer}$, Playtime:D_{time}, Location_ID:D_{string})
- Mobs(Mob_ID:D_{string}, Mob_Name:D_{string}, Mob_Health:$D_{integer}$, Money_Drop:$D_{integer}$, Exp_Drop:$D_{integer}$, Drop_ID:D_{string})
- Consumable(Item_ID:D_{string}, Item_Desc:D_{string}, **Heal_Total**:$D_{integer}$, **Str_Buff**:$D_{integer}$, **Magic_Buff**:$D_{integer}$, **Item_Duration**:D_{time}, **Item_Cost**:$D_{integer}$)
- Location(Location_ID:D_{string}, Rec_Loc_Lvl:$D_{integer}$, Location_Name:D_{string})
- Quest(Quest_ID:D_{string}, Quest_Name:D_{string}, Money_Given:$D_{integer}$, Rec_Quest_Lvl:$D_{integer}$, Exp_Given:$D_{integer}$, **Mobs_to_Kill**:D_{string}, **Items_to_Collect**:D_{string}, Quest_Type:d_{quest})
- NPC(NPC_ID:D_{string} NPC_Name:D_{string}, NPC_Type:d_{NPC})
- Armor(Armor_ID:D_{string}, A_Health_Given:$D_{integer}$, **A_Magic_Given**:$D_{integer}$, **A_Strength_Given**:$D_{integer}$, Armor_Cost:$D_{integer}$, Armor_Name:D_{string})
- Weapon(Weapon_ID:D_{string}, **W_Strength_Given**:$D_{integer}$, **W_Magic_Given**:$D_{integer}$, Weapon_Type:d_{weapon}, Weapon_Name:D_{string}, Weapon_Cost:$D_{integer}$)
- Char_Weapons(Email:d_{email}, Character_Name:D_{string}, Weapon_ID:D_{string})
- Char_Armor(Email:d_{email}, Character_Name:D_{string}, Armor_ID:D_{string})
- Quest_Counter(Email:d_{email}, Character_Name:D_{string}, Quest_ID:D_{string}, Completed:D_{bool})
- Kill_Counter(Email:d_{email}, Character_Name:D_{string}, Mob_ID:D_{string}, C_Kill_M_Counter:$D_{integer}$, M_Kill_C_Counter:$D_{integer}$)

The domains are described as follows:

- D_{string}: String
- $D_{integer}$: Non-Negative Integer
- D_{time}: H:M:S time
- D_{bool}: True/False
- $d_{email} \in D_{string}$: [name] @ [domain] format
- $d_{class} \in D_{string}$: Available classes in game
- $d_{quest} \in D_{string}$: Available quest types
- $d_{NPC} \in D_{string}$: Available NPC types

Normalization

- Account:
    - Email -> First_Name
    - Email -> Last_Name
    - Email -> Time_Played
    - Email -> Num_Character

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Location:
    - Location_ID -> Rec_Loc_Lvl
    - Location_ID-> Location_Name

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Player_Character:
    - Email -> ACC_Email
    - ACC_Email, Character_Name -> Class
    - ACC_Email, Character_Name -> Experience
    - ACC_Email, Character_Name -> Health
    - ACC_Email, Character_Name -> Strength
    - ACC_Email, Character_Name -> Magika
    - ACC_Email, Character_Name -> Money
    - ACC_Email, Character_Name -> Playtime
    - Location_ID -> LOC_Location_ID

  This table is in BCNF because all attributes are fully dependent on the primary keys.
- Consumable:
    - Item_ID -> Item_Desc
    - Item_ID -> Heal_Total
    - Item_ID -> Str_Buff
    - Item_ID -> Magic_Buff
    - Item_ID -> Item_Duration
    - Item_ID -> Item_Cost

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Mob:
    - Mob_ID -> Mob_Name
    - Mob_ID -> Mob_Health
    - Mob_ID -> Money_Drop
    - Mob_ID -> Exp_Drop
    - Item_ID -> Drop_ID

  This table is in BCNF because all attributes are fully dependent on the primary key.

- Quest:
    - Quest_ID -> Quest_Name
    - Quest_ID -> Money_Given
    - Quest_ID -> Rec_Quest_Lvl
    - Quest_ID -> Exp_Given
    - Mob_ID -> Mobs_to_Kill
    - Item_ID -> Items_to_Collect
    - Quest_ID -> Quest_Type

  This table is not in 1NF because a Quest can have more than one value within Mobs_to_Kill and Items_to_Collect. To fix this, two new tables should be created containing a Quest's Target Mobs and a Quest's Target Items. This would improve the design because one Quest ID could then be related to both a Quest Mobs and a Quest Items table.
- NPC:
    - NPC_ID -> NPC_Name
    - NPC_ID -> NPC_Type

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Armor:
    - Armor_ID -> A_Health_Given
    - Armor_ID -> A_Magic_Given
    - Armor_ID -> A_Strength_Given
    - Armor_ID -> Armor_Cost
    - Armor_ID -> Armor_Name

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Weapon:
    - Weapon_ID -> W_Strength_Given
    - Weapon_ID -> W_Magic_Given
    - Weapon_ID -> Weapon_Type
    - Weapon_ID -> Weapon_Name
    - Weapon_ID -> Weapon_Cost

  This table is in BCNF because all attributes are fully dependent on the primary key.
- Char_Weapon:
    - Email -> ACC_Email
    - Character_Name -> CHAR_Name
    - Weapon_ID -> WEAPON_ID

  This table is in BCNF because all attributes are fully dependent on the primary keys.
- Char_Armor:
    - Email -> ACC_Email
    - Character_Name -> CHAR_Name

    - Armor_ID -> ARMOR_ID

  This table is in BCNF because all attributes are fully dependent on the primary keys.
- Quest_Counter:
    - Email -> ACC_Email
    - Character_Name -> CHAR_Name
    - Quest_ID -> QUEST_ID

  This table is in BCNF because all attributes are fully dependent on the primary keys.
- Kill_Counter:
    - Email -> ACC_Email
    - Character_Name -> CHAR_Name
    - Mob_ID -> MOB_ID
    - ACC_Email, CHAR_Name, MOB_ID -> C_Kill_M_Counter
    - ACC_Email, CHAR_Name, MOB_ID -> M_Kill_C_Counter

  This table is in BCNF because all attributes are fully dependent on the primary keys.

Tooling Assessment

DBMS

MySQL will be our Database Management System because it is open source, free, and widely used. It also connects well with our other tools.

UI

We used Python to create our command-line UI.

Hosting

Amazon Web Services (AWS) will host our database because it is widely used in industry and free for students.

Additional Tools

Version Control

Git / GitHub will be used to collaborate and control versions.

Back-end Language

Python will hold our back-end because it connects well with MySQL and it is easy to learn and use.

IDE

PyCharm will be our Python IDE because it integrates well with GitHub, MySQL, and the premium version is free for students.

Data Generation

Mockaroo because we will need randomly generated data and it is free for the first 1,000 rows of data.

Database Creation

Table Creation Queries

SQL Statement	Purpose

`create table ACCOUNT`
`(`
`  Email varchar(50) not null,`
`  First_Name varchar(15) not null,`
`  Last_Name varchar(15) null,`
`  Time_Played time default '000:00:00' not null,`
`  Num_Character tinyint unsigned default 0 not null,`
`  constraint ACCOUNT_pk`
`  primary key (Email)`
`);`
	Create Account Table

`create table LOCATION`
`(`
`  Location_ID varchar(30) not null,`
`  Rec_Loc_Lvl tinyint unsigned default 1 not null,`
`  Location_Name varchar(30) not null,`
`  constraint LOCATION_pk`
`  primary key (Location_ID)`
`);`
	Create Location Table

`create table PLAYER_CHARACTER`
`(`
`  ACC_Email varchar(50) not null,`
`  Character_Name varchar(30) not null,`
`  Class varchar(7) not null`
`  check (Class in ('Warrior', 'Mage', 'Thief')),`
`  Experience int unsigned default 0 not null,`
`  Health int unsigned not null,`
`  Strength int unsigned not null,`
`  Magika int unsigned not null,`
`  Money int unsigned default 100 not null,`
`  Playtime time default '000:00:00' not null,`
`  LOC_Location_ID varchar(30) default 'StartingZone' not null,`
`  constraint CHARACTER_pk`
`  primary key (ACC_Email, Character_Name),`
`  index name (Character_Name),`
`  constraint CHAR_ACC_Email_fk`
`  foreign key (ACC_Email) references ACCOUNT (Email)`
`  on update cascade on delete cascade,`
`  constraint CHAR_LOC_LocID_fk`
`  foreign key (LOC_Location_ID) references LOCATION (Location_ID)`
`  on update cascade on delete cascade`
	Create Player Character Table

); 	

create table CONSUMABLE
(
    Item_ID varchar(20) not null,
    Item_Desc text not null,
    Heal_Total int signed null,
    Str_Buff int signed null,
    Magic_Buff int signed null,
    Item_Duration time null,
    Item_Cost mediumint unsigned null,
    constraint CONSUMABLE_pk
       primary key (Item_ID)
);
	Create Consumable Table

create table MOB
(
    Mob_ID varchar(20) not null,
    Mob_Name varchar(30) not null,
    Mob_Health smallint unsigned not null,
    Money_Drop mediumint unsigned not null,
    Exp_Drop int unsigned not null,
    Drop_ID varchar(20) not null,
    constraint MOB_pk
       primary key (Mob_ID),
    constraint MOB_CONSUME_Drop_fk
       foreign key (Drop_ID) references CONSUMABLE (Item_ID)
             on update cascade on delete cascade
);
	Create Mob Table

create table QUEST
(
    Quest_ID varchar(30) not null,
    Quest_Name tinytext not null,
    Money_Given mediumint unsigned not null,
    Rec_Quest_Lvl tinyint unsigned not null,
    Exp_Given int unsigned not null,
    Mobs_to_Kill varchar(20) not null,
    Items_to_Collect varchar(20) not null,
    Quest_Type varchar(7) not null
      check (Quest_Type in ('Slay', 'Collect')),
    constraint QUEST_pk
       primary key (Quest_ID),
    constraint QUEST_CONSUME_Collect_fk
       foreign key (Items_to_Collect) references CONSUMABLE
(Item_ID)
             on update cascade on delete cascade,
    constraint QUEST_MOB_Kill_fk
       foreign key (Mobs_to_Kill) references MOB (Mob_ID)
             on update cascade on delete cascade
);
	Create Quest Table

create table NPC
(
    NPC_ID varchar(20) not null,
    NPC_Name varchar(20) not null,
    NPC_Type varchar(10) not null,
    constraint NPC_pk
       primary key (NPC_ID)
);
	Create NPC Table

create table ARMOR
(
    Armor_ID varchar(30) not null,
    A_Health_Given int signed not null,
    A_Magic_Given int signed null,
    A_Strength_Given int signed null,
    Armor_Cost mediumint unsigned not null,
    Armor_Name tinytext not null,
    constraint ARMOR_pk
       primary key (Armor_ID)
);
	Create Armor Table

create table WEAPON
(
    Weapon_ID varchar(30) not null,
    W_Strength_Given int signed null,
    W_Magic_Given int signed null,
    Weapon_Type varchar(9) not null
      check (Weapon_Type in ('Staff', 'Wand', 'Dagger',
'Sword', 'Longsword')),
    Weapon_Name tinytext not null,
    Weapon_Cost mediumint unsigned not null,
    constraint WEAPON_pk
       primary key (Weapon_ID)
);
	Create Weapon Table

create table CHAR_WEAPON
(
      ACC_Email varchar(50) not null,
      CHAR_Name varchar(30) not null,
      WEAPON_ID varchar(30) not null,
      constraint CHAR_WEAPON_pk
      primary key (ACC_Email, CHAR_Name, WEAPON_ID),
      constraint CHAR_WEAPON_ACC_Email_fk
      foreign key (ACC_Email) references ACCOUNT (Email)
            on update cascade on delete cascade,
      constraint CHAR_WEAPON_CHAR_Name_fk
      foreign key (CHAR_Name) references PLAYER_CHARACTER
(Character_Name)
            on update cascade on delete cascade,
      constraint CHAR_WEAPON_WEAPON_ID_fk
      foreign key (WEAPON_ID) references WEAPON (Weapon_ID)
	Create Character Weapon Table

on update cascade on delete cascade);	
create table CHAR_ARMOR (ACC_Email varchar(50) not null, CHAR_Name varchar(30) not null, ARMOR_ID varchar(30) not null, constraint CHAR_ARMOR_pk primary key (ACC_Email, CHAR_Name, ARMOR_ID), constraint CHAR_ARMOR_ACC_Email_fk foreign key (ACC_Email) references ACCOUNT (Email) on update cascade on delete cascade, constraint CHAR_ARMOR_ARMOR_ID_fk foreign key (ARMOR_ID) references ARMOR (Armor_ID) on update cascade on delete cascade, constraint CHAR_ARMOR_CHAR_Name_fk foreign key (CHAR_Name) references PLAYER_CHARACTER (Character_Name) on update cascade on delete cascade);	Create Character Armor Table
create table QUEST_COUNTER (ACC_Email varchar(50) not null, CHAR_Name varchar(30) not null, QUEST_ID varchar(30) not null, Completed boolean not null default false, constraint QUEST_COUNTER_pk primary key (ACC_Email, CHAR_Name, QUEST_ID), constraint QUEST_COUNTER_ACC_Email_fk foreign key (ACC_Email) references ACCOUNT (Email) on update cascade on delete cascade, constraint QUEST_COUNTER_CHAR_Name_fk foreign key (CHAR_Name) references PLAYER_CHARACTER (Character_Name) on update cascade on delete cascade, constraint QUEST_COUNTER_QUEST_ID_fk foreign key (QUEST_ID) references QUEST(Quest_ID) on update cascade on delete cascade);	Create Quest Counter Table
create table KILL_COUNTER (ACC_Email varchar(50) not null, CHAR_Name varchar(30) not null, MOB_ID varchar(20) not null, C_Kill_M_Counter mediumint unsigned default 0 not null, M_Kill_C_Counter mediumint unsigned default 0 not null, constraint KILL_COUNTER_pk primary key (ACC_Email, CHAR_Name, MOB_ID),	Create Kill Counter Table

```
      constraint KILL_COUNTER_ACC_Email_fk
      foreign key (ACC_Email) references ACCOUNT (Email)
            on update cascade on delete cascade,
      constraint KILL_COUNTER_CHAR_Name_fk
      foreign key (CHAR_Name) references PLAYER_CHARACTER
(Character_Name)
            on update cascade on delete cascade,
      constraint KILL_COUNTER_MOB_ID
      foreign key (MOB_ID) references MOB (Mob_ID)
            on update cascade on delete cascade
);
```

Constraint Testing

The following SQL statements were specially written to test the four types of constraints (domain, referential integrity, entity integrity, key/uniqueness). Correctness is ensured in that all of the following SQL statements will fail if executed, as a constraint is failed on each statement.

SQL Statement	**Purpose**
`insert into NPC values ('01', 'Mark Jackson', 'Quest');` `insert into NPC values ('01', 'Mike Brown', 'Quest');`	Test that a two NPCs cannot have the same ID **Key/Uniqueness Constraint**
`insert into QUEST values ('7', 'Broken Quest 2', 500, 12, 10000, '09', 'FullHeal', 'Escort');`	Test that the quest type is of ('Slay', 'Collect') **Domain Constraint**
`insert into CHAR_WEAPON values ('edwareli@uw.edu', 'EliTheMage', '02');`	Test that an email must be valid in CHAR_WEAPON **Referential Integrity Constraint**
`insert into ACCOUNT values ('BadEmail@domain.C', 'John', 'Doe', default, 0);`	Test that an ACCOUNT's email is valid **Domain Constraint**
`insert into PLAYER_CHARACTER values ('GoodEmail@domain.com', 'GoodGuyJohn', 'Bard', 40000, 1000, 500, 10, 50000, '020:00:00', 'Skyrim');`	Test that a PLAYER_CHARACTER's class is of ('Warrior', 'Mage', 'Thief') **Domain Constraint**
`insert into WEAPON values ('15', 50, 0, 'Axe', 'Iron Axe', 1000);`	Test that a WEAPON's type is of ('Staff', 'Wand', 'Dagger', 'Sword', 'Longsword') **Domain Constraint**
`insert into Location values (null, 999, 'BadLocation');`	Test that a Location must have an Location_ID (cannot be null) **Entity Integrity Constraint**
`insert into QUEST values ('7', 'Broken Quest', 1000, 26, 10000, NULL, 'FullHeal', 'Collect');`	Test that both mob to kill and a item to collect are necessary in a quest **Domain Constraint**

Data Generation

The data was generated by manually writing all of the test data directly into MySQL with SQL statements. We took inspiration from the Elder Scrolls video game series for some of the manually created data (The Elder Scrolls Online).

Sample data for Account, Character, Char_Armor, Char_Weapon, Quest_Counter and Kill_Counter was randomly generated by Mockaroo.

UI

Our UI was implemented using Python and is a simple command-line interface. The user scenario we chose was the video game's developers. They would use this UI to analyze and view data from the game to assist in development decisions. The UI allows for the user to enter a SQL statement from the command line and the SQL statement is executed in the database. Our UI for the class demo was prepared with some use cases already prepared in functions to make the demo go smoothly. This would be replicated in our use case scenario because developers would need to run certain queries often to watch trends/change in data.

If we had more time, we would like to expand the UI to be able to be viewed by players of the game. This would provide information such as: the most popular weapons, the top players, the most popular quests or locations, et cetera. In addition, we would like to develop built in functions to detect bad/cheating accounts.

Figure 3: UI (Dashboard)
Figure 3 shows the UI that is available to developers when the file is run. This UI is the one used for the class demo with the option for a custom SQL query or a choice of 5 ready SQL queries.

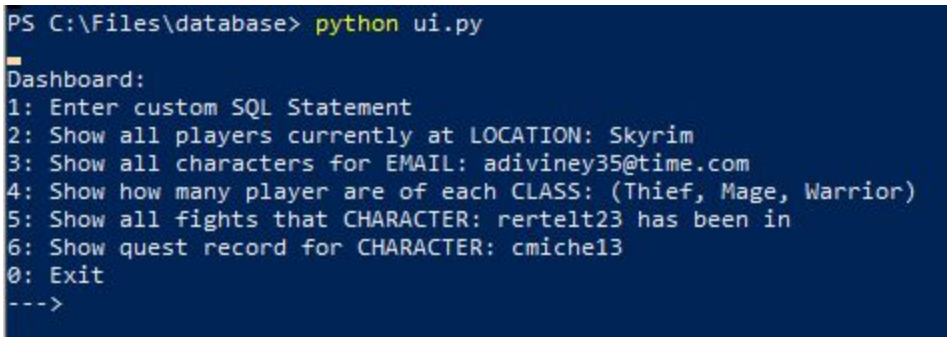


Figure 4: Code Snippet
Figure 4 shows two code snippets. The first is the input loop that takes input from the user and calls a function through a switch. The second section shows the function customSQL() that is called when the user inputs "1" for custom SQL input.

```
# Getting user input
while True:
    # UI Dashboard
    print("\nDashboard:")
```

```
    print("1: Enter custom SQL Statement")
    print("2: Show all players currently at LOCATION: Skyrim")
    print("3: Show all characters for EMAIL: adiviney35@time.com")
    print("4: Show how many player are of each CLASS: (Thief, Mage,
Warrior)")
    print("5: Show all fights that CHARACTER: rertelt23 has been in")
    print("6: Show quest record for CHARACTER: cmiche13")
    print("0: Exit")

    value = int(input('---> '))
    switcher[value]()
```

```
# Allows for custom SQL statement through command line input
def customSQL():
    retVal = input('Enter SQL ---> ')
    test = pd.read_sql(retVal, con=mydb)
    print(test)
```

Figure 5: Output

Figure 5 shows the output of a custom SQL query that was inputted form the UI.

```
---> 1
Enter SQL ---> SELECT * FROM LOCATION
       Location_ID  Rec_Loc_Lvl    Location_Name
0       Blackmarsh           80      Black Marsh
1         Cyrodiil            0         Cyrodiil
2        Cyrodiil2           30         Cyrodiil
3          Elsweyr           70          Elsweyr
4       Hammerfell           50       Hammerfell
5         HighRock           10        High Rock
6        Morrowind           20        Morrowind
7         Oblivion          100         Oblivion
8           Skyrim           40           Skyrim
9   SummersetIsles           90  Summerset Isles
10       Valenwood           60        Valenwood

Dashboard:
1: Enter custom SQL Statement
2: Show all players currently at LOCATION: Skyrim
3: Show all characters for EMAIL: adiviney35@time.com
4: Show how many player are of each CLASS: (Thief, Mage, Warrior)
5: Show all fights that CHARACTER: rertelt23 has been in
6: Show quest record for CHARACTER: cmiche13
0: Exit
--->
```

Demo

The demo showcases these 8 queries to show how a game developer may use our database to help retrieve useful information about the events taking place in game. The data would then be used to help make decisions on future development.
We felt more complicated (i.e. join, nested) queries would complicate the demo and be too specific for our data. We were confident that our queries provided the most common and practical use case scenarios. However, we do see the value and functionality of game developers creating certain complicated queries to check specific data.

SQL Statement	**Purpose**
SELECT ACC_Email, Character_Name **FROM** PLAYER_CHARACTER, LOCATION **WHERE** LOC_Location_ID='Skyrim'	This statement selects all players currently at the location “Skyrim”. This could easily be changed to access a different location. This statement is useful to see the most popular and least popular locations.
SELECT * **FROM** PLAYER_CHARACTER **WHERE** ACC_Email='adiviney35@time.com’	This statement shows all the characters for a given account. This is useful for bug/error tickets by accounts to see all the different characters connected to their account.
SELECT COUNT(Character_Name) **FROM** PLAYER_CHARACTER **WHERE** Class='Thief' **SELECT** COUNT(Character_Name) **FROM** PLAYER_CHARACTER **WHERE** Class='Mage’ **SELECT** COUNT(Character_Name) **FROM** PLAYER_CHARACTER **WHERE** Class='Warrior'	These three statements display the number of characters who are playing as class ‘Thief’, ‘Mage’, and ‘Warrior’. This was done in our demo and provides insight on what classes are being used the most which could alter balancing decisions for developers. For example, in our database ‘Thief’ has the most users by a sizable amount which could signal a “nerf” to the ‘Thief’ class or new content for the class.
SELECT * **FROM** KILL_COUNTER **WHERE** CHAR_Name='rertelt23'	This statement displays all “fights” that involve the character named ‘rertelt23’. This is useful to see how a certain character is performing against certain mobs or how low level characters are performing against certain mobs.
SELECT QUEST_ID, Completed **FROM** QUEST_COUNTER **WHERE** CHAR_Name='cmiche13'	This statement selects all quests that are currently assigned to character ‘cmiche13’. This is useful for displaying an in-game journal for characters of

	current and completed quests. In addition, it is useful for debugging glitches where certain quests are not completing correctly.

Connecting to the Database

Our database is hosted with AWS using MySQL.

To access our database follow these steps:

1. Download MySQL Workbench
2. In MySQL Workbench go to the "Database" tab
3. Select "Manage Connections"
4. Select "New"
5. Login with:
    a. Host-name: videogamedb.coollmbh4cdk.us-west-2.rds.amazonaws.com
    b. Port: 3306
    c. Username: admin
    d. Password: (emailed to professor and graders on 03/06/20)

Project Evaluation

Successes

Overall, the project went over smoothly. All project members communicated often online and met up in-person before each iteration to plan and design for the upcoming work. Frequent communication among our team allowed us to swiftly decide how we wanted to divide the work evenly among team members. Each team member specialized in a certain area, receiving help or feedback from other team members when needed, allowing for efficient completion of work. In the early stages of planning, we wanted to ensure that the scope of our project was realistic for the time we were given, which ended up working well as we did not plan for anything too small, or anything too large.

Issues

When generating the random data in Mockaroo, our team ran into some issues involving missing data and inconsistencies due to Mockaroo's limitations. Some Accounts were not linked to the correct number of Characters, and Account playtime's were not adding up with Character playtime's. These problems occurred because the numbers were derived from Account's number of characters and playtime, while if it was derived from Character, the problem would be solved. Additionally, not every character had a Kill or Quest Counter. This problem occurred because Kill Counter is a cartesian product of Character x Mob, which results in more rows than Mockaroo allows in a single generation. Mockaroo also does not have a time value, so we struggled a bit in formatting that. Apart from Mockaroo, our team did not encounter other obstacles during the development of this project.

Future Changes

Upon evaluation of our project, some future changes we could potentially make to improve our project would be:

- Adding a Character Inventory so a character can carry un-equipped Armor and Weapons, rather than equipping on pickup, and Consumables for later use, rather than immediate use on pickup.
- Normalization of the Quest table, allowing each Quest ID to be related to a Quest Mobs table and a Quest Items table.
- Improvement of and hosting for the UI.
- More test data and more types of data (different NPCs, different weapons, etc.)
- Adding a Shop table operated by an NPC that the Player can purchase weapons, armor, or consumables from using money.

References

The Elder Scrolls Online [Video game]. (2014). Rockville, MD: Bethesda Softworks.