

# CS2300 Project Phase II

DnD Digital Character Manager

Group: Evan Richard

## Problem Statement:

D&D is a fun fantasy role-playing game and is a great game to play with friends or to make new ones! One thing, however, is not so fun about the game. The mundane task of managing your inventory, managing the companions, loading the horses, calculating your travel speeds, eating, and drinking. Be you a player or dungeon master, taking care of these tasks can be time-consuming. Especially for the dungeon master, doing these tasks can leave your players feeling bored and overwhelmed. If only there were a simple way to take care of these tasks to maintain realism whilst also spending most of your time doing what you want to do, adventure and battle!

## Conceptual Database Design:

My database first consists of a “User” entity. This entity will store the hashed password and username of the user. This is important so that my application can recognize unique users and accurately store the characters that the user has created.

The “Character” entity is then identified as something a “User” can create, but every individual “Character” entity must be associated with at most one “User”. The Character entity stores a lot of pertinent information about the player’s character. Most importantly, here we have the CharacterId that acts as the primary key to uniquely identify each character. Another important attribute is the “IsCompanion” attribute. This is a boolean that will identify whether the created character is a companion or not. This is my solution to not having to create an entirely new entity for a companion, since a companion has all of the same attributes that a character would have. This goes along with the “OwnerId” attribute. This will be a nullable string attribute that will store the CharacterId inside of it only if it is a character. This is my solution to being able to quickly reference which companion is associated with what character. So, along with a companion having its own unique ID, it will have the stored ID of the character that “owns” it.

The EER diagram is then separated into two distinguishable parts, the “CanHave” and “MustHave” weak relationships. The purpose of these is to distinguish what a character MUST have to properly function as a playable entity and what a character CAN have but isn’t required for play. This is very important, why? Because not everyone finds it pertinent to add information like age or alignment to their characters, and it should not be required of the user to apply. More importantly, however, this information is not needed for the character to actually function in-game and is more along the lines of the “roleplay” part.

Starting with the “MustHave” section, we have the “STATS” weak entity. I’ve decided to make this a weak entity because the name of a stat alone is not enough to make it uniquely identifiable and should instead be uniquely identified depending on the character it is associated with. This entity will hold information relating to things that change quite often, including health, hit dice, inspiration, and exp.

Then we have the “Class” weak entity. Weak for the same reason as mentioned above, this will be responsible for holding the class of the player. Notice I’ve made this entity be able to be created more than once, hence the player can have more than one class. This is important for a specific mechanic called subclassing. The main idea of this application is to give the player freedom as well, and I don’t want my application to hinder what a player can do. So I say if a

player wants to have a dozen classes, then let them do so; it is a role-playing game after all. Information held here in attributes is associated with what a class would give your player. So things like proficiency bonuses, your total hit dice, max hit points, and spell slots are all stored here.

The “RACE” weak entity type is a basic entity that just stores what race your character is, along with your speed given by it. The reason nothing else is stored here is that everything else given by your race can be implemented or placed somewhere else in the DBMS, and there is no need to store it here. For example, you may be given some features from your race, which has its own place in the DBMS

We also have the skill entity here. This entity is responsible for holding your skills, so think of things like Deception or Investigation. It also gives the player the ability to add other skills for roleplaying purposes if they so choose. Stored here simply are the attributes pertinent to a skill, including its name, score, and whether you are proficient.

The “Saving\_Throw” weak entity is simply just like the skills, but stores your saving throws like “Dexterity” and keeps track of the score and if you’re proficient.

Finally, we have the “Ability” entity, which exists to give your character their abilities. So this will be a character’s Strength, Dexterity, Intelligence, and so on. Only the pertinent information about these abilities is stored here, such as the score and modifier.

Notice that all of these weak entities are connected by full participation double lines. This is because each entity MUST be associated with a character and a character MUST have at least 1 one of each of the aforementioned entity types. This rule does not exactly follow in the “CanHave” weak relationship, where a character doesn’t necessarily have to participate in the relationship with any of them.

Firstly, here we have the “Feat” entity, which is responsible for giving a player a feat. A feat is something that can mess with the character’s abilities, but is not necessarily something that a character must have to function. Hence, it was placed in this section.

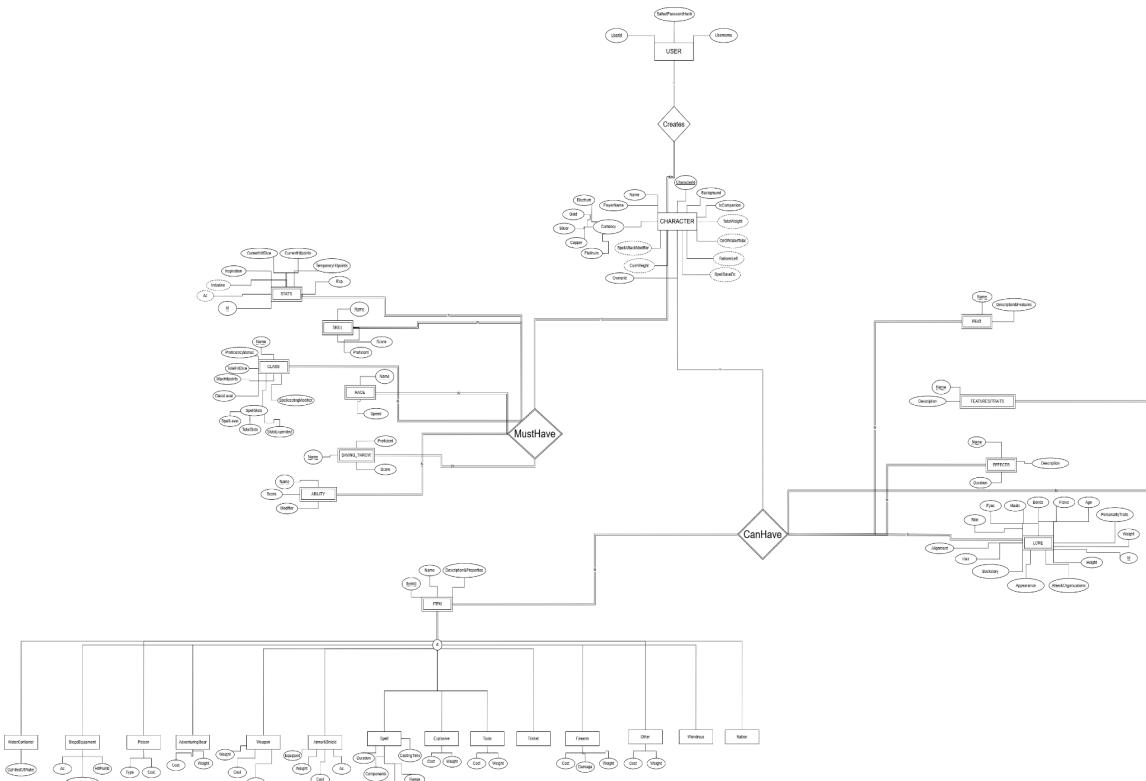
Next, we also have the “Features/Traits” weak entity, which is responsible for storing a character’s special features or traits that could be given to them through many things, like their race or class. A feature can also affect a character’s playstyle, and hence a string is stored that allows a player to describe exactly what the feature is and how it affects them. There are many different types, and infinitely many if you consider that this is a role-playing game. Hence, this entity is left pretty ambiguous, and what this entity could hold is left mostly up to the player.

The “EFFECTS” weak entity is kind of special compared to all of the others. Here is where status effects like unconscious, poisoned, bleeding, and more would be entered. Just like the previous entity, it has a pretty ambiguous description. This is because there are so many different types of possible status effects that could inhibit a player (not all of which are bad), it would be insane to store them all and keep track. Not even considering that, just like the last entity, a status effect could be very specific to a campaign, like imagine an effect called “Curse of the Ice Queen” or something. Anyway, a string-type attribute called “Duration” is left here so that the player can remember how long they have to have this effect for. The duration is a string because it could last days, hours, minutes, rounds, weeks, or even indefinitely.

The “LORE” entity holds all information that is related to a player’s physical, mental, and even spiritual bondage with the world. Obviously, this is placed here because you don’t need

this for every character. For example, if your companion is a literal horse, you might not feel the need to fill this out.

Finally and most importantly, the “ITEM” attribute. This is basically the inventory management system. Every item can be a multitude of different non-overlapping things. Every “ITEM,” however, will have its own ID, name, and description for ambiguity. An item can be a “WaterContainer” if it is something that holds oz of liquid that is important for the character’s survival and, more importantly, for the application to easily find this for later reference. The item can be siege equipment, a special item that a player can have that has its own AC, hit points, and immunities. It can be poison if it is well, a poison vial, for example. Adventuring gear is a lot, and most items, like a torch, are in your backpack. A weapon if it something that does damage, for example, a longsword. Armor or Shield, if it is something like a helmet that can be equipped and affects your AC. A spell, if it’s something like a fireball. Now this is an interesting implementation of my database. I decided to add spells as an item. This is just because it seemed like the easiest way and would make it easier to reference later. Explosives are things like bombs and dynamite, of course. A tool would be something like a pickaxe or shovel. Trinkets are a special form of item that can be found; they can have very specific properties and are left ambiguous for this reason. A firearm is something like a pistol or machine gun; they act similarly to a weapon type. We then had a wondrous item, which is similar to a trinket in that it is pretty ambiguous but can have incredible abilities. A ration is an item of course that is important for the player’s survival, and also important to have for the application to track. Finally, we have the “Other” item type, which, of course, is my way of countering ambiguity. This gives the player the ability to add items that may not fit into any of the other categories and are very specific to a campaign. For example, I can’t really come up with every item type if you are playing a DOOM or One Piece-themed DnD campaign.



# Logical Database Design:

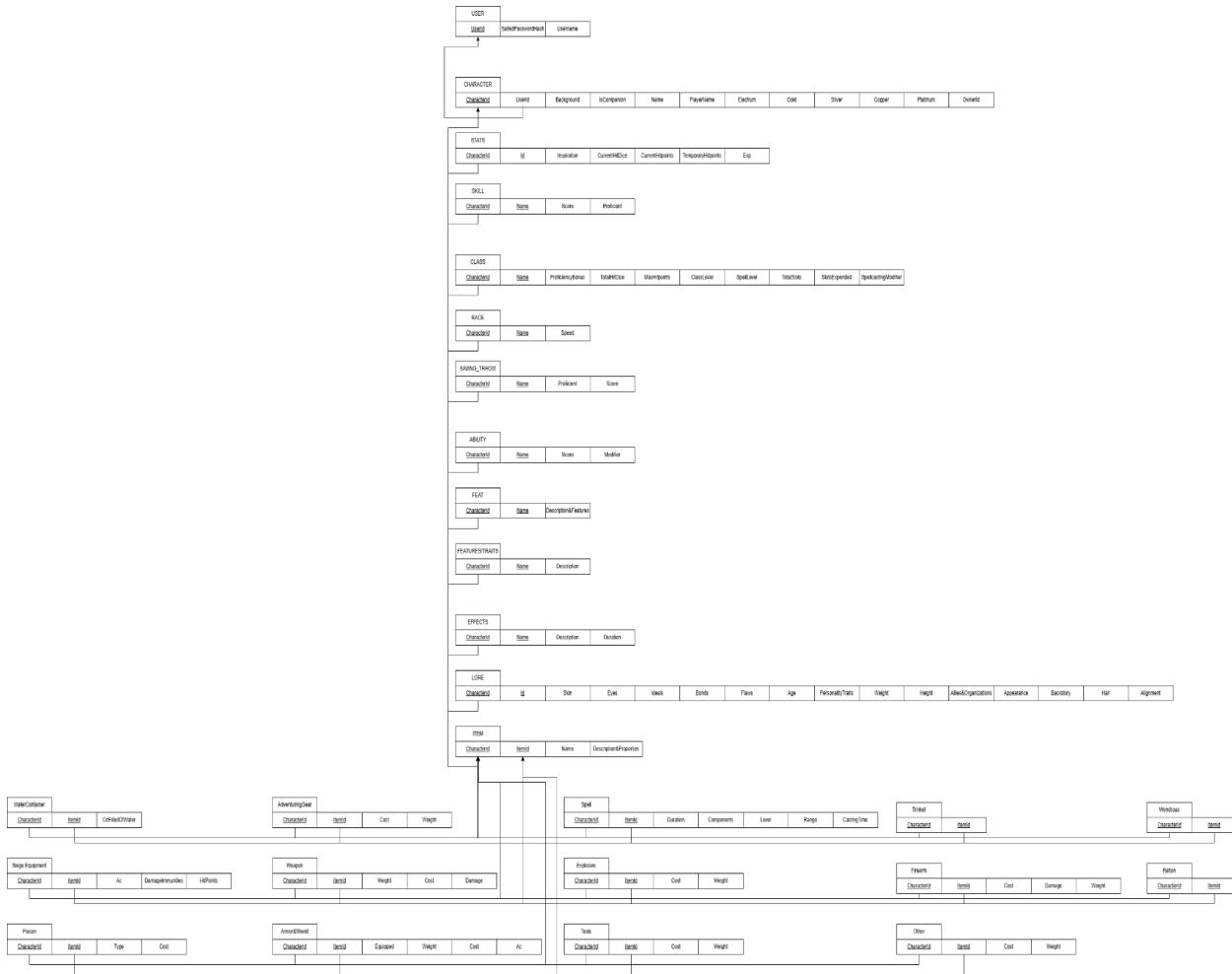


Table	Attribute	Type	Constraint
User	UserId	Integer	Primary Key
User	SaltedPasswordHash	String	NOT NULL
User	Username	String	NOT NULL
Character	CharacterId	Integer	Primary Key
Character	UserId	Integer	Foreign Key
Character	Background	String	

Character	IsCompanion	Boolean	NOT NULL
Character	Name	String	
Character	PlayerName	String	
Character	Electrum	Integer	>= 0
Character	Gold	Integer	>= 0
Character	Silver	Integer	>= 0
Character	Copper	Integer	>= 0
Character	Platinum	Integer	>= 0
Character	Onwerld	Integer	>= 0
Stats	CharacterId	Integer	Foreign Key
Stats	Id	Integer	Primary Key
Stats	Inspiration	Integer	>= 0
Stats	CurrentHitDice	Integer	>= 0

Stats	CurrentHitpoints	Integer	<= MaxHitpoints
Stats	TemporaryHitpoints	Integer	>= 0
Stats	Exp	Integer	>= 0
Skill	CharacterId	Integer	Foreign Key
Skill	Name	String	Primary Key
Skill	Score	Integer	NOT NULL
Skill	Proficient	Boolean	NOT NULL
Class	CharacterId	Integer	Foreign Key
Class	Name	String	Primary Key
Class	ProficiencyBonus	Integer	>= 0
Class	TotalHitDice	Integer	>= 0
Class	MaxHitpoints	Integer	>= 0
Class	ClassLevel	Integer	>= 0

Class	SpellLevel	Integer	>= 0
Class	TotalSlots	Integer	>= 0
Class	SlotsExpended	Integer	>= 0 AND <= TotalSlots
Class	SpellcastingModifier	Integer	>= 0
Race	CharacterId	Integer	Foreign Key
Race	Name	String	Primary Key
Race	Speed	Integer	>= 0

Saving_Throw	CharacterId	Integer	Foreign Key
Saving_Throw	Name	String	Primary Key
Saving_Throw	Proficient	Boolean	NOT NULL
Saving_Throw	Score	Integer	NOT NULL
Ability	CharacterId	Integer	Foreign Key
Ability	Name	String	Primary Key
Ability	Score	Integer	NOT NULL
Ability	Modifier	Integer	NOT NULL
Feat	CharacterId	Integer	Foreign Key
Feat	Name	String	Primary Key
Feat	Description&Features	String	
Features/Traits	CharacterId	Integer	Foreign Key
Features/Traits	Name	String	Primary Key
Features/Traits	Description	String	
Effects	CharacterId	Integer	Foreign Key
Effects	Name	String	Primary Key
Effects	Description	String	
Effects	Duration	String	

Lore	CharacterId	Integer	Foreign Key
Lore	Id	Integer	Primary Key

Lore	Skin	String	
Lore	Eyes	String	
Lore	Ideals	String	
Lore	Bonds	String	
Lore	Flaws	String	
Lore	Age	Integer	$\geq 0$
Lore	PersonalityTraits	String	
Lore	Weight	Integer	$\geq 0$
Lore	Height	String	
Lore	Allies&Organizations	String	
Lore	Appearance	String	
Lore	Backstory	String	
Lore	Hair	String	
Lore	Alignment	String	
Item	CharacterId	Integer	Foreign Key
Item	ItemId	Integer	Primary Key
Item	Name	String	
Item	Description&Properties	String	
WaterContainer	CharacterId	Integer	Primary Key
WaterContainer	ItemId	Integer	Primary Key

WaterContainer	OzFilledOfWater	Integer	$\geq 0$
SiegeEquipment	CharacterId	Integer	Primary Key
SiegeEquipment	ItemId	Integer	Primary Key

SiegeEquipment	Ac	Integer	
SiegeEquipment	Damagelmmunities	String	
SiegeEquipment	HitPoints	Integer	
Poison	CharacterId	Integer	Primary Key
Poison	ItemId	Integer	Primary Key
Poison	Type	String	
Poison	Cost	String	
AdventuringGear	CharacterId	Integer	Primary Key
AdventuringGear	ItemId	Integer	Primary Key
AdventuringGear	Cost	String	
AdventuringGear	Weight	Integer	$\geq 0$
Weapon	CharacterId	Integer	Primary Key
Weapon	ItemId	Integer	Primary Key
Weapon	Weight	Integer	$\geq 0$
Weapon	Cost	String	
Weapon	Damage	String	
Armor&Shield	CharacterId	Integer	Primary Key
Armor&Shield	ItemId	Integer	Primary Key

Armor&Shield	Equipped	Boolean	NOT NULL
Armor&Shield	Weight	Integer	$\geq 0$
Armor&Shield	Cost	String	
Armor&Shield	Ac	Integer	NOT NULL
Spell	CharacterId	Integer	Primary Key
Spell	ItemId	Integer	Primary Key
Spell	Duration	String	
Spell	Components	String	

Spell	Level	Integer	$\geq 0$
Spell	Range	String	
Spell	CastingTime	String	
Explosive	CharacterId	Integer	Primary Key
Explosive	ItemId	Integer	Primary Key
Explosive	Cost	String	
Explosive	Weight	Integer	$\geq 0$
Tools	CharacterId	Integer	Primary Key
Tools	ItemId	Integer	Primary Key
Tools	Cost	String	
Tools	Weight	Integer	$\geq 0$

Trinket	CharacterId	Integer	Primary Key
Trinket	ItemId	Integer	Primary Key
Firearm	CharacterId	Integer	Primary Key
Firearm	ItemId	Integer	Primary Key
Firearm	Cost	String	
Firearm	Damage	String	
Firearm	Weight	Integer	$\geq 0$
Other	CharacterId	Integer	Primary Key
Other	ItemId	Integer	Primary Key
Other	Cost	String	
Other	Weight	Integer	$\geq 0$
Wondrous	CharacterId	Integer	Primary Key
Wondrous	ItemId	Integer	Primary Key
Ration	CharacterId	Integer	Primary Key
Ration	ItemId	Integer	Primary Key

## Application Program Design:

```
//Beginning Stage Functions (Login and Listing)

Login()
    User = input for username
    Password = input for password
    If (query database for user and hashed password exists) then
        // Open_Next_Page for character listing
        List_Characters()
    Else
        Warn ("Your password and username don't match. Create account?")

Create_Account()
    User = input for username
    Password = input for password
    If (query database for user and hashed password exists) then
        Warn ("User already exists, please login")
    Else
        // Open_Next_Page for character listing
        List_Characters()

List_Characters()
    For (query all character that are not companions) do
        // list character names as clickable buttons
        If (character button clicked) then
            Query_Character()

Add_Character()
    // Load_Character_Creation_Page
    IsCompanion = toggle input
    If (IsCompanion) then
        CharacerId = input for character Id that this companion is associated with
        If (Query for CharacerId exists != true) then
            Warn ("Sorry that character does not exist")
            Return
    // Ask for input on attributes with buttons for races
```

```

    Add_Skill() // will automatically call to add atleast the
    required skills
        Add_Race() // character must have atleast one race
        Add_SavingThrow() // auto calls to give basic saving throws
        Add_Class() // character must have atleast one class
        Add_Ability() // auto call to add basic abilities
        Add_Lore() // not necessary but player may add
        Add_Effect() // not necessary but player may add
        Add_FeatureOrTrait() // not necessary but player may add
        Add_Feat() // not necessary but player can add
        Add_Item() // not necessary but player can add

    Finished = toggle button for character creation done
    If (Finished == true) then
        // make sure player meets the given restraints above and
    values are not null
        Query_Character() // query the new character

    Remove_Character()
        Id = user id input to delete
        If (query Id exist == true) then
            Execute character deletion

    Query_Character()
        // Query and List all attributes of the CHARACTER, STATS,
    SKILL, CLASS, RACE, SAVING_THROW, ABILITY, FEAT, FEATURES/TRAITS,
    EFFECTS, LORE, AND ITEM entities to their respective spots

        //call aggregation functions for listing now
        Get_Water_Total()
        Get_Ration_Total()
        Get_Total_Weight()
        Get_Spell_Attack_Modifer()
        Get_Initiative()
        Get_Ac()
        If (button for drink water pressed AND Get_Water_Total() >=
    request amount) then
            Drink_Water()
        Else
            Warn ("You don't have enough water")

        If (button for eat food pressed AND Get_Ration_Total() > 0)
    then
        Eat_Ration()

```

```
    Else
        Warn ("You don't have any available rations!")
```

### **//Character Creation Functions**

```
Add_Skill()
    // Input for name, score, and proficiency

Add_Race()
    // Input for name and speed given

Add_SavingThrow()
    // Input for name, score, and if proficient

Add_Class()
    // Input for name, proficiency bonus, total hit dice,
hitpoints, class level, spell level, slots, and slots expended

Add_Ability()
    // input for name, score, and modifier

Add_Lore()
    // input for attributes (lots...)

Add_Effect()
    // input attributes

Add_FeatureOrTrait()
    // input attributes

Add_Feat()
    // input attributes

Add_Item()
    // input attributes
```

### **//Character Maintenance Functions**

```
Drink_Water(oz)
    // should've been checked prior to being called
    WaterToBeDrinken = oz
    While (WaterToBeDrinken > 0) do
```

```

        Container = // Query database for WaterContainer above 0
OzFilledOfWater
        Container.OzFilledOfWater -= WaterToBeDrinken
        If (Container.OzFilledOfWater < 0) then
            WaterToBeDrinken =
max(abs(Container.OzFilledOfWater), 0)
        Else
            WaterToBeDrinken -= WaterToBeDrinken
Get_Water_Total()

Eat_Ration()
    // should've been checked before being called
    // Query for ration in database and delete it from database
    Get_Ration_Total()

//Storage Functions
LogOff()
    // save all attributes except those derived to database

//Aggregation Functions
Get_Water_Total()
    Water = 0
    For (query database for water container)
        Water += OzFilledOfWater
    Return Water

Get_Ration_Total()
    Food = 0
    For (query database for Ration)
        Food += 1
    Return Food

Get_Coin_Weight()
    // Query Database for Electrum, Gold, silver, Copper, and
Platinum
    // For each coin value add one pound to total weight and return

Get_Total_Weight()
    Weight = Get_Coin_Weight()
    Weight += Query database for Items that have a weight attribute
and add here
    Return Weight

Get_Spell_Attack_Modifer()

```

```
    Mod = query proficiency bonus + query spell casting ability
modifier
    Return Mod
```

```
Get_Initiative()
    Input = initiative is something rolled by player, put input
here
    Return Input
```

```
Get_Ac() // Ac can obviously be changed depending on features so is
left to be derived and changed by the player
    Ac = 10 + query for Dexterity Modifier
    Return Ac
```