**Jeff Bertucci**

**Project Milestone 1**

**11/10/18**

**Project summary**

I want to create a database that logs game information from my in-person Dungeons and Dragons group and stores it all in one easy to access place. We have had a series of different adventures that were created by a variety of different players. At the current time, each of us as players, Game Masters (GMs) or both, hold only the information about our own stuff that we as individuals created. I want the database to be able to access which adventures were created by whom as well as which players (and their respective characters) engaged with those adventures. I would also like to include player demographic information (age, gender, location) as well as select character statistics from the game (Character Class, Race, highest level attained, current adventure status in game). One additional factor that I’d like to include are the unique magical items that characters have attained and which Game Master distributed them. In characterizing adventures, I would like to include who curated it, and which characters did the undertaking.

**Some specific data questions that should be answered are:**

Which Player has been given the most items? Related would be “Which GM gives out the most items?”

Which Characters have undergone the most adventures?

What type (character class/race) of character do certain players tend to play?

Which GM has had the deadliest adventure?

Which Player has made the most characters? Who has accumulated the most levels?

**Things that are currently out of scope, but I’d like to try to include in the future:**

More character data: Wealth and property, attribute statistics, number of kills, successes and failures (as determined by dice rolls), critical hits and misses. This data goes largely unrecorded, so getting enough of it to be relevant may not be possible in the short term.

Setting data: GMs create an expansive world filled with towns, people, enemies, and dungeons. I would love to include this extra data in a database, however, that would balloon my project into being more of an encyclopedia of descriptions rather than specific data points about those entities. It would also be an excessive amount of extra workload.

Security features: GMs may have some information that they would like to keep hidden from players until a certain event happens, but they may still want to keep it hidden. “Adventure name” being a standout as it may spoil some aspect of a newly ongoing adventure. I’d like for there to be different levels of access so that privileged data may remain hidden from the average user. I do not know if this one specifically is within the scope of this class and if it is, I will try to include it.

**Stakeholders**

My stakeholders for this project are the players and Game masters that have played with my group of friends. Players would interact with this database to determine interesting trends and facts that they were unaware of and shape decisions that they make at the creative level. Game Masters can use this information to shape their adventures into something enjoyable for each of their players by knowing what those players tend to enjoy doing. The database also serves as a type of historical log about the games that we friends played together. It is something we can use to look back on some of the good times we had. My friends are my best critics, so I want to create something that they will want to use.

**Entity Glossary**

**Game Master –** The player that runs the game, facilitates adventures, creates items

Characteristics – Name, Age, Gender, Location

**Player –** A person who plays the game, creates characters

Characteristics – Name, Age, Gender, Location

**Adventure –** The game that is facilitated by a GM, undertaken by characters

Characteristics – Start Date, End Date, Game Location, Adventure Name, Completion Status

**Character –** The avatar a player creates. Undertakes adventures, and collects items

Characteristics – Created Date, Name, Race, Class (type of character), background, Character Status, description, highest level attained

**Item –** Specific magical rewards characters receive by undertaking adventures, created by Game Master

Characteristics – Created Date, Item Name, Item Status, Item Description.

**Relation Glossary**

**Creates –** Players and Game Masters make things within the game based on statistics that guidebooks provide.

**Directs –** Game Masters run their adventures for players to play in.

**Undergoes –** Characters complete objectives to form a Game Master’s narrative

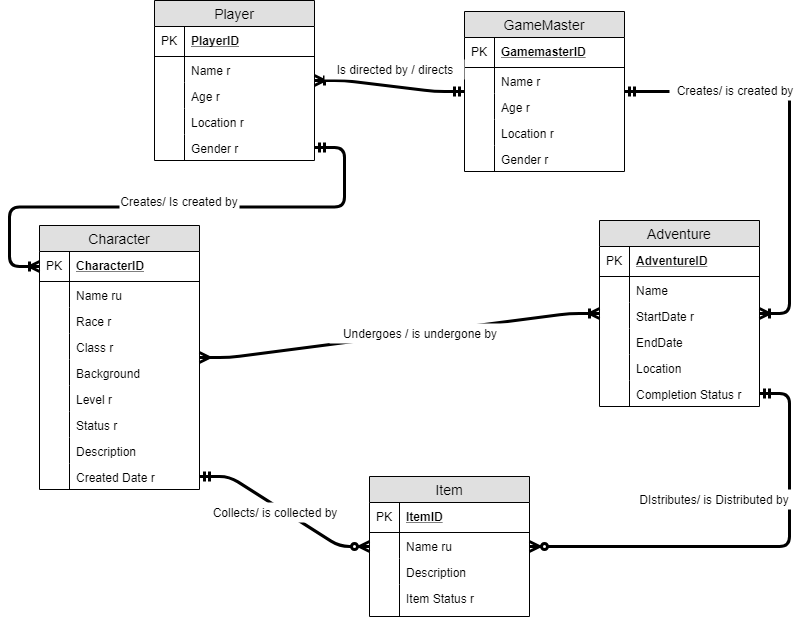
**Distributes –** Adventures are prepared such that items may be found by players

**Collects –** Players find magical items in their travels and add them to their inventory

**Business Rules**

1. A Game Master (GM) can create one to many adventures, an adventure can be run by one and only one GM.
2. A GM can direct many players, players can only have one GM at a time
3. An adventure can distribute zero to many items.
4. A Player can create one to many characters, a character can only have one player.
5. Characters can undertake many adventures, and adventures are undertaken by many characters.
6. Characters receive zero to many items.
7. Magical Items are unique.
8. Adventures, Characters, and Items all have specific required statuses.

**Conceptual Model**



For my conceptual model, each entity has is its own surrogate key. Items and Characters must have unique names in this system.

Sample data generated from our current campaign. This project should cover our past campaigns, and more detailed player information in addition to this data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Adventure** | **Adventure status** | **GM** | **Players** | **Characters** | **Race** |
| The Blood War | In Progress | Jimmy Roberts | Jeff Bertucci | Claire Ironstorm | Dwarf |
| The Blood War | In Progress | Jimmy Roberts | Kyle Shropshire | Alara | Halfling |
| The Blood War | In Progress | Jimmy Roberts | Kyle Knight | Golgoroth | Half Orc |
| The Blood War | In Progress | Jimmy Roberts | Tyler Duclos | Lethos of Vipers | Half Orc |
| The Blood War | In Progress | Jimmy Roberts | Sydney Hunsinger | Talise Seabrooke | Genasi |
| The Blood War | In Progress | Jimmy Roberts | Mallory Mcmahon | Amastacia Beverly Hills | Human |
| The Blood War | In Progress | Jimmy Roberts | Jack Hadley | Evelyn Clementine | Human |
| The Blood War | In Progress | Jimmy Roberts | Rory Bowden | Ada Kendral | Human |

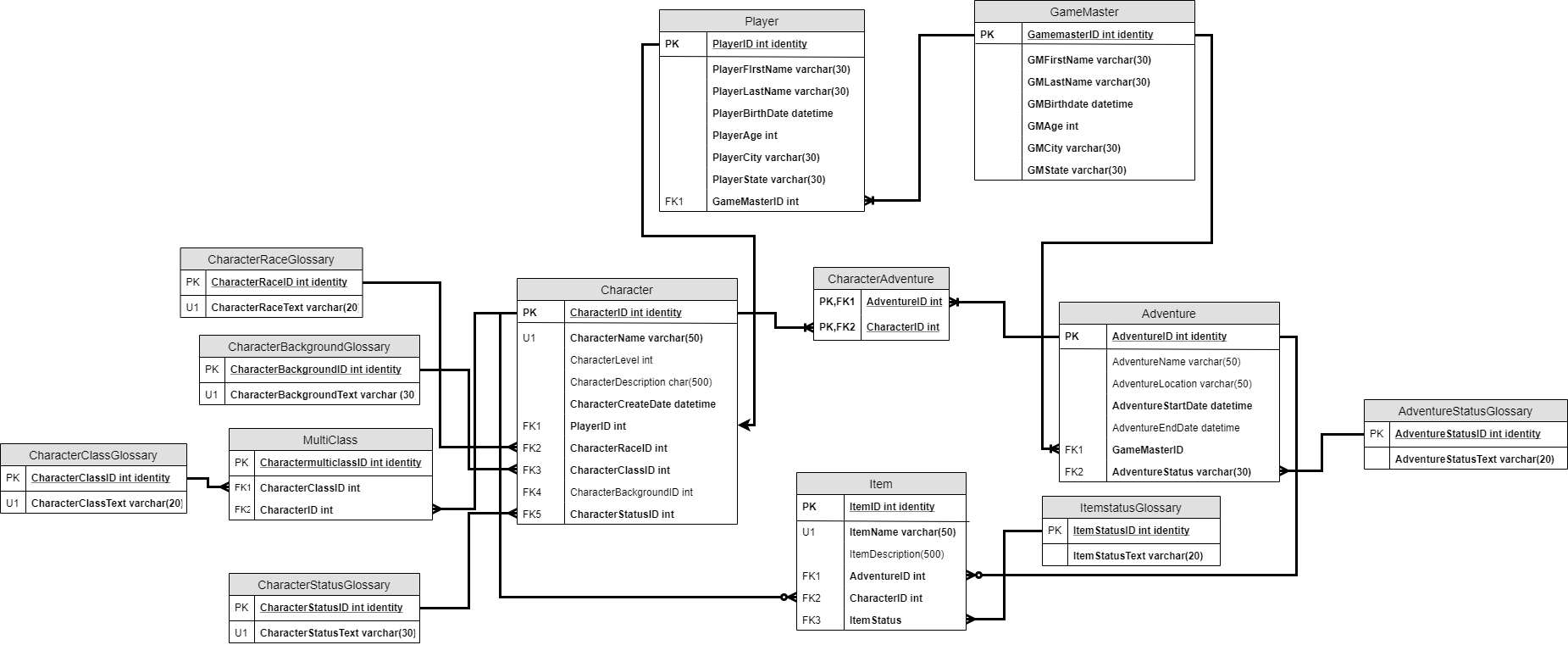
|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Level** | **Status** | **Items** |
| Sorcerer | 13 | Active | Twinclubs, Charisma stone |
| Rouge, fighter | 13 | Active | Duelist's cloak, Ring of Djinni Summoning, Djinn bound windblade |
| Barbarian | 5 | Missing in action | null |
| Barbarian | 13 | Active | Frostbite greataxe, ring of the living wall, earring of the red knight |
| Paladin | 13 | Active | Belt of Cloud Giant strength |
| Cleric | 13 | Active | Coin of the healer |
| Wizard | 13 | Active | ring of recall, envoy's veil, staff of charming |
| Bard | 13 | Active | Violin of the butterfly |

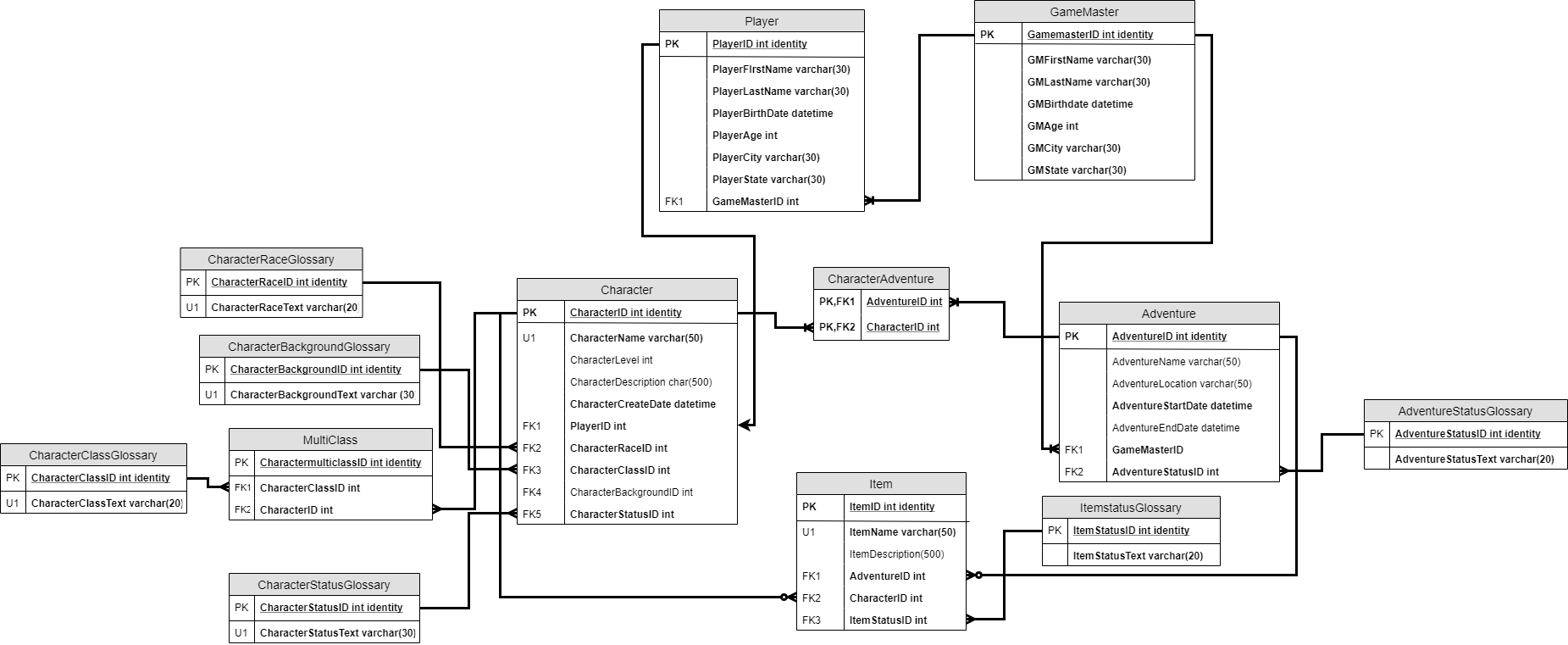
**Logical model Information**

For the logical model, each entity was given a specific generated surrogate key. This was done to keep everything consistent throughout. Although some entities have unique constraints that could have been used, using surrogate keys makes the whole system a bit more fool-proof, which I would want for this project because I could potentially have many users inputting information. For that same reason, I created several glossary tables to include look-ups to various finite game facts, as well as making tables for specific statuses players, adventures, and Items may have. One glossary table, CharacterClassGlossary, turned out to actually be a many to many relationship, as characters can have one or more classes on rare occasion, so I had to create a bridge table to allow for characters that had more than one class. I also needed to create a bridge table for the many to many relationship between the Character and Adventure tables. I first thought about using the Item table for that purpose, but it made more sense to me to have a definitive link between the entities since there may be instances where adventures do not distribute items to the characters. For most of my character input columns, I used the varchar data type. I left character level as an int data type because there is the potential for math to be done to it (finding a player’s total accumulated levels over time). Description text blocks were defined using the char data type to keep it a bit more organized if looking at multiple descriptions at once. Almost all inputs are set to be required, a few of the fields that potentially could give important information away can be set to null if desired. Specifically, this includes aspects of the Adventure entity, as that is where most privileged information may reside. The same idea applies to the ItemDescription field in the Item table, and to a lesser extent PlayerLevel. The idea between including birthdate and not age directly on the Player and GameMaster tables is that Birthdate is a constant that will not change and can be uniformly inputted with the datetime data type. Math can be done on the birthdate to get someone’s age, so it can be included, and it will correctly update over time. For Unique Constraints, all the text in the glossary tables needed to be unique. Characters and items cannot have the same name either per the business rule. One final note about character name: as characters may have aliases in game, they may not follow traditional First Name Last Name format, so I chose not to break that field up.

I apologize that the logical model had to be spread out across two pages, the glossary tables ballooned the model out a bit more than I would have liked.

**Logical Model**

****

****

**Project Milestone 2**

**12/07/18**

**Physical DataBase Design**

-- Dropping Bridge Tables

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_MultiClass')

BEGIN

DROP TABLE dd\_MultiClass

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_CharacterAdventure')

BEGIN

DROP TABLE dd\_CharacterAdventure

END

GO

--Dropping Input Tables

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_Item')

BEGIN

DROP TABLE dd\_Item

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_Adventure')

BEGIN

DROP TABLE dd\_Adventure

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_Character')

BEGIN

DROP TABLE dd\_Character

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_GameMaster')

BEGIN

DROP TABLE dd\_GameMaster

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_Player')

BEGIN

DROP TABLE dd\_Player

END

GO

--Dropping Glossary Tables

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_CharacterRace')

BEGIN

DROP TABLE dd\_CharacterRace

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_CharacterClass')

BEGIN

DROP TABLE dd\_CharacterClass

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_CharacterBackground')

BEGIN

DROP TABLE dd\_CharacterBackground

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_CharacterStatus')

BEGIN

DROP TABLE dd\_CharacterStatus

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_AdventureStatus')

BEGIN

DROP TABLE dd\_AdventureStatus

END

GO

IF EXISTS (Select \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'dd\_ItemStatus')

BEGIN

DROP TABLE dd\_ItemStatus

END

GO

--Dropping Functions

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_CharacterStatusLookup'))

DROP FUNCTION dbo.dd\_Characterstatuslookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_AdventureStatusLookup'))

DROP FUNCTION dbo.dd\_Adventurestatuslookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_ItemStatusLookup'))

DROP FUNCTION dbo.dd\_ItemStatusLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_RaceLookup'))

DROP FUNCTION dbo.dd\_RaceLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_ClassLookup'))

DROP FUNCTION dbo.dd\_ClassLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_BackgroundLookup'))

DROP FUNCTION dbo.dd\_BackgroundLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_PlayerLookup'))

DROP FUNCTION dbo.dd\_PlayerLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_GMIDLookup'))

DROP FUNCTION dbo.dd\_GMIDLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_GMNameLookup'))

DROP FUNCTION dbo.dd\_GMNameLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_CharacterLookup'))

DROP FUNCTION dbo.dd\_CharacterLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_AdventureLookup'))

DROP FUNCTION dbo.dd\_AdventureLookup

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_ItemLookup'))

DROP FUNCTION dbo.dd\_ItemLookup

GO

--Dropping Procedures

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_PlayerInput'))

DROP PROCEDURE dbo.dd\_PlayerInput

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MakeGM'))

DROP PROCEDURE dbo.dd\_MakeGM

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_AdventureInput'))

DROP PROCEDURE dbo.dd\_AdventureInput

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_CharacterInput'))

DROP PROCEDURE dbo.dd\_CharacterInput

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_AdventureParty'))

DROP PROCEDURE dbo.dd\_AdventureParty

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_ItemInput'))

DROP PROCEDURE dbo.dd\_ItemInput

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_LevelUp'))

DROP PROCEDURE dbo.dd\_LevelUp

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_LevelDown'))

DROP PROCEDURE dbo.dd\_LevelDown

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_EndAdventure'))

DROP PROCEDURE dbo.dd\_EndAdventure

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_ItemStatusUpdate'))

DROP PROCEDURE dbo.dd\_ItemStatusUpdate

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_Equip'))

DROP PROCEDURE dbo.dd\_Equip

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_CharacterStatusUpdate'))

DROP PROCEDURE dbo.dd\_CharacterStatusUpdate

GO

--Dropping Views

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostItemsRecieved'))

DROP VIEW dbo.dd\_MostItemsRecieved

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostItemsGiven'))

DROP VIEW dbo.dd\_MostItemsGiven

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostPlayedClasses'))

DROP VIEW dbo.dd\_MostPlayedClasses

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostPlayedRaces'))

DROP VIEW dbo.dd\_MostPlayedRaces

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostAdventures'))

DROP VIEW dbo.dd\_MostAdventures

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostLevels'))

DROP VIEW dbo.dd\_MostLevels

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_Deadliest'))

DROP VIEW dbo.dd\_Deadliest

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_MostCharacters'))

DROP VIEW dbo.dd\_MostCharacters

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id = OBJECT\_ID('dbo.dd\_GMAdventureReport'))

DROP VIEW dbo.dd\_GMAdventureReport

GO

--Creating Glossary Tables these will not updated frequently if at all and are only used for table reference

--The text on these tables need to be unique for lookup purposes

CREATE TABLE dd\_CharacterRace (

CharacterRaceID INT IDENTITY PRIMARY KEY,

CharacterRaceText VARCHAR(20) NOT NULL,

CONSTRAINT U1\_CharacterRace UNIQUE (CharacterRaceText)

)

CREATE TABLE dd\_CharacterClass (

CharacterClassID INT IDENTITY PRIMARY KEY,

CharacterClassText VARCHAR(20) NOT NULL,

CONSTRAINT U1\_CharacterClass UNIQUE (CharacterClassText)

)

CREATE TABLE dd\_CharacterBackground (

CharacterBackgroundID INT IDENTITY PRIMARY KEY,

CharacterBackgroundText VARCHAR(30) NOT NULL,

CONSTRAINT U1\_CharacterBackground UNIQUE (CharacterBackgroundText)

)

CREATE TABLE dd\_CharacterStatus (

CharacterStatusID INT IDENTITY PRIMARY KEY,

CharacterStatusText VARCHAR(30) NOT NULL,

CONSTRAINT U1\_CharacterStatus UNIQUE (CharacterStatusText)

)

CREATE TABLE dd\_AdventureStatus (

AdventureStatusID INT IDENTITY PRIMARY KEY,

AdventureStatusText VARCHAR(30) NOT NULL,

CONSTRAINT U1\_AdventureStatus UNIQUE (AdventureStatusText)

)

CREATE TABLE dd\_ItemStatus (

ItemStatusID INT IDENTITY PRIMARY KEY,

ItemStatusText VARCHAR(30) NOT NULL,

CONSTRAINT U1\_ItemStatus UNIQUE (ItemStatusText)

)

--Creating Input Tables

--For Players, I want the combination of first and last name to be unique so the same player can't be entered twice

--All information is required at this stage, birthdate was selected rather than age because it will give a more accurate result further in time

CREATE TABLE dd\_Player (

PlayerID INT IDENTITY PRIMARY KEY,

PlayerFirstName VARCHAR(30) NOT NULL,

PlayerLastName VARCHAR(30) NOT NULL,

PlayerBirthDate DATETIME NOT NULL,

PlayerCity VARCHAR(30) NOT NULL,

PlayerState VARCHAR(2) NOT NULL,

CONSTRAINT U1\_PlayerName UNIQUE (PlayerFirstName, PlayerLastName)

)

--The gamemaster table is actually just a a subtype of the player table, and it will only be used for reference purposes

CREATE TABLE dd\_GameMaster (

GameMasterID INT IDENTITY PRIMARY KEY,

PlayerID INT NOT NULL,

CONSTRAINT fk\_playerID FOREIGN KEY (PlayerID) REFERENCES dd\_Player(PlayerID)

)

--The character table, character names need to be unique, but they can be long. Their creation date defaults to the time they are entered into the system.

--Description and background can be null

CREATE TABLE dd\_Character (

CharacterID INT IDENTITY PRIMARY KEY,

CharacterName VARCHAR(50) NOT NULL,

CharacterDescription CHAR(500),

CharacterCreateDate Datetime DEFAULT getdate(),

PlayerID INT NOT NULL,

CharacterRaceID INT NOT NULL,

CharacterBackgroundID int NULL,

CharacterStatusID int NOT NULL DEFAULT 1, --Default will be set to active

CONSTRAINT U1\_CharacterName UNIQUE (CharacterName),

CONSTRAINT fk1\_playerID FOREIGN KEY (PlayerID) REFERENCES dd\_Player(PlayerID),

CONSTRAINT fk2\_CharacterRaceID FOREIGN KEY (CharacterRaceID) REFERENCES dd\_CharacterRace(CharacterRaceID),

CONSTRAINT fk3\_CharacterBackgroundID FOREIGN KEY (CharacterBackgroundID) REFERENCES dd\_CharacterBackground(CharacterBackgroundID),

CONSTRAINT fk4\_CharacterStatusID FOREIGN KEY (CharacterStatusID) REFERENCES dd\_CharacterStatus(CharacterStatusID)

)

--I originally planned to allow for null adventure names, this causes some confusion during implementation that I wanted to avoid

--Adventure Names need to be unique for a few functions to work

--StartDate and EndDate both default to the current time

--Its not neccessary to input a description or a location

CREATE TABLE dd\_Adventure (

AdventureID INT IDENTITY PRIMARY KEY,

AdventureName VARCHAR(50) NOT NULL,

AdventureLocation VARCHAR(50) NULL,

AdventureDescription CHAR(500) NULL,

AdventureStartDate DATETIME DEFAULT getdate() NOT NULL,

AdventureEndDate DATETIME DEFAULT getdate() NOT NULL,

GameMasterID int,

AdventureStatusID int NOT NULL DEFAULT 1, --Default will be set to "in progress"

CONSTRAINT U1\_AdventureName UNIQUE (AdventureName),

CONSTRAINT fk1\_AdventureStatus FOREIGN KEY (AdventureStatusID) REFERENCES dd\_AdventureStatus(AdventureStatusID),

CONSTRAINT fk2\_GameMasterID FOREIGN KEY (GameMasterID) REFERENCES dd\_GameMaster(GameMasterID)

)

--dd\_item will be a running glossary of the items collected

--Name needs to be unique, but its description can be left out

--Items can be attached to characters or adventures, but it is not necessary

CREATE TABLE dd\_Item (

ItemID INT IDENTITY,

ItemName VARCHAR(50) NOT NULL,

ItemDescription CHAR (500) NULL,

AdventureID int,

CharacterID int,

ItemStatusID int DEFAULT 1, --Default will be to "Unknown"

CONSTRAINT U1\_ItemName UNIQUE (ItemName),

CONSTRAINT fk1\_AdventureID FOREIGN KEY (AdventureID) REFERENCES dd\_Adventure(AdventureID),

CONSTRAINT fk2\_CharacterID FOREIGN KEY (CharacterID) REFERENCES dd\_Character(CharacterID),

CONSTRAINT fk3\_ItemStatusID FOREIGN KEY (ItemstatusID) REFERENCES dd\_ItemStatus(ItemStatusID)

)

--This is a bridge table that attaches characters to adventures

--A character can be attached to a single adventure only once

CREATE TABLE dd\_CharacterAdventure (

AdventureID INT,

CharacterID INT,

CONSTRAINT fk1\_CharAdventureID FOREIGN KEY (AdventureID) REFERENCES dd\_Adventure(AdventureID),

CONSTRAINT fk2\_AdventCharacterID FOREIGN KEY (CharacterID) REFERENCES dd\_Character(CharacterID),

CONSTRAINT u1\_CharacterAdventure UNIQUE (AdventureID, CharacterID)

)

--This is a bridge table that connects character to the classglossary,

--it also stores information regarding how many levels a character has in a given class

--A character can only be attached to a certain class once, but they can attach to as many different class as they want

CREATE TABLE dd\_MultiClass (

MultiClassID INT IDENTITY PRIMARY KEY,

CharacterClassID int,

CharacterID int,

LevelAmount int,

CONSTRAINT fk1\_CharacterClass FOREIGN KEY (CharacterClassID) REFERENCES dd\_CharacterClass(CharacterClassID),

CONSTRAINT fk2\_ClassCharacterID FOREIGN KEY (CharacterID) REFERENCES dd\_Character(CharacterID),

CONSTRAINT u1\_Multiclass UNIQUE (CharacterClassID, CharacterID)

)

GO

--Creating Functions

--This function will look up a corresponding CharacterStatusID based on the string input

CREATE FUNCTION dd\_CharacterStatusLookup (@name varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = CharacterStatusID FROM dd\_CharacterStatus

WHERE @name = CharacterStatusText

RETURN @returnValue

END

GO

--This function will look up a corresponding AdventureStatusID based on the string input

CREATE FUNCTION dd\_AdventureStatusLookup (@name varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = AdventureStatusID FROM dd\_AdventureStatus

WHERE @name = AdventureStatusText

RETURN @returnValue

END

GO

--This function will look up a corresponding ItemStatusID based on the string input

CREATE FUNCTION dd\_ItemStatusLookup (@name varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = ItemStatusID FROM dd\_ItemStatus

WHERE @name = ItemStatusText

RETURN @returnValue

END

GO

--This function will lookup a corresponding RaceID based on the string input

CREATE FUNCTION dd\_RaceLookup (@RaceText Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = CharacterRaceID FROM dd\_CharacterRace

WHERE @RaceText = CharacterRaceText

RETURN @returnValue

END

GO

--This function will lookup a corresponding ClassID based on the string input

CREATE FUNCTION dd\_ClassLookup (@ClassText Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = CharacterClassID FROM dd\_CharacterClass

WHERE @ClassText = CharacterClassText

RETURN @returnValue

END

GO

--This function will lookup a corresponding BackgroundID based on the string input

CREATE FUNCTION dd\_BackgroundLookup (@BackgroundText Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = CharacterBackgroundID FROM dd\_CharacterBackground

WHERE @BackgroundText = CharacterBackgroundText

RETURN @returnValue

END

GO

--This Function looks up corresponding PlayerID Using their Last Name

--For what I need Last Name is a good indicator to look up PlayerID, as they are all unique for my data set.

--If I were to expand operation, I would update it to be first and last name

CREATE FUNCTION dd\_PlayerLookup (@Name Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = PlayerID FROM dd\_Player

WHERE @Name = PlayerLastName

RETURN @returnValue

END

GO

--This Function Looks up a corresponding GMID For an inputted last Name

CREATE FUNCTION dd\_GMIDLookup (@GMName Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue =GameMasterID FROM dd\_GameMaster

JOIN dd\_Player ON dd\_GameMaster.PlayerID = dd\_Player.PlayerID

WHERE @GMName = PlayerLastName

RETURN @returnValue

END

GO

--This function looks up a gm name based on their id,

--only useful for when I want to display both players and gms at the same time

CREATE FUNCTION dd\_GMNameLookup (@gmid int)

RETURNS VARCHAR(30) AS

BEGIN

DECLARE @returnvalue VARCHAR(30)

SELECT @returnvalue = PlayerFirstName + ' ' + PlayerLastName FROM dd\_Player

WHERE PlayerID = @gmid

RETURN @returnValue

END

GO

/\*

Since Character names can get long, and users probably don't want to type out full names,

I used the CHARINDEX stored function here to find a CharacterName that is Close to What was input

Should work fine for nicknames, but it is designed to fail if more than one character are selected

\*/

CREATE FUNCTION dd\_CharacterLookup (@Name Varchar(50))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT = 0

DECLARE @returncount INT

SELECT @returncount = COUNT(CharacterID) FROM dd\_Character WHERE CHARINDEX(@Name, CharacterName) > 0

IF @returncount = 1

BEGIN

SELECT @returnvalue = CharacterID FROM dd\_Character WHERE CHARINDEX(@Name, CharacterName) > 0

END

RETURN @returnValue

END

GO

--This Function will look up an AdventureID based on a select string input

--For convenience sake, like dd\_characterLookup, the full Adventure name is not necessary as input,

--If more than one adventure is selected, the function is designed to fail procedures that use it

CREATE FUNCTION dd\_AdventureLookup (@Name Varchar(50))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT = 0

DECLARE @returncount INT

SELECT @returncount = COUNT(AdventureID) FROM dd\_Adventure WHERE CHARINDEX(@Name, AdventureName) > 0

IF @returncount = 1

BEGIN

SELECT @returnvalue = AdventureID FROM dd\_Adventure WHERE CHARINDEX(@Name, AdventureName) > 0

END

RETURN @returnValue

END

GO

--This function will lookup the corresponding ItemID for a given Item Name, The whole Item Name must be input

CREATE FUNCTION dd\_ItemLookup (@Name Varchar(20))

RETURNS INT AS

BEGIN

DECLARE @returnvalue INT

SELECT @returnvalue = ItemID FROM dd\_Item

WHERE @Name = ItemName

RETURN @returnValue

END

GO

--Making Procedures

--This Procedure Will Add a new Player into the database

CREATE PROCEDURE dd\_PlayerInput

(@First varchar(30), @Last varchar(30), @Birth datetime, @City varchar(30), @State varchar(2)) AS

BEGIN

INSERT INTO dd\_Player (PlayerFirstName, PlayerLastName, PlayerBirthDate, PlayerCity, PlayerState)

VALUES (@first, @last, @birth, @city, @state)

END

GO

--This Procedure Will elevate a player to Game Master level

CREATE PROCEDURE dd\_MakeGM (@PlayerLastName varchar (20)) AS

BEGIN

DECLARE @playerid int

SELECT @playerid = dbo.dd\_PlayerLookup (@PlayerLastName)

INSERT INTO dd\_GameMaster (PlayerID) VALUES (@playerid)

END

GO

--This procedure allows a user to input a new adventure

CREATE PROCEDURE dd\_AdventureInput

(@GameMasterLastName varchar(20), @AdventureName Varchar(50), @AdventureLocation Varchar(50), @AdventureDescription varchar(500)) AS

BEGIN

INSERT INTO dd\_Adventure (GameMasterID, AdventureName, AdventureLocation, AdventureDescription)

VALUES (

dbo.dd\_GMIDLookup(@GameMasterLastName),

@AdventureName,

@AdventureLocation,

@AdventureDescription

)

END

GO

--This Procedure allows a user to input a new character

CREATE PROCEDURE dd\_CharacterInput

(@PlayerLastName Varchar(20), @CharacterName varchar(30), @CharacterRace varchar(20), @CharacterBackground varchar(20), @CharacterDescription Char(500)) AS

BEGIN

INSERT INTO dd\_Character (PlayerID, CharacterName, CharacterRaceID, CharacterBackgroundID, CharacterDescription)

VALUES (

dbo.dd\_PlayerLookup (@PlayerLastName),

@CharacterName,

dbo.dd\_RaceLookup (@CharacterRace),

dbo.dd\_BackgroundLookup (@CharacterBackground),

@CharacterDescription)

END

GO

--This procedure will assign a character to an adventure essentially "placing them into a party"

--Both @charactername and @adventurename can be partially input and it should return the correct result

CREATE PROCEDURE dd\_AdventureParty (@characterName varchar(30), @adventureName varchar(50)) AS

BEGIN

DECLARE @adventureid int = dbo.dd\_AdventureLookup(@AdventureName)

DECLARE @characterID int = dbo.dd\_CharacterLookup(@CharacterName)

IF @adventureid = 0

SELECT 'Please Refine Adventure Name'

IF @characterID = 0

SELECT 'Please Refine Character Name'

IF @adventureID > 0 AND @CharacterID > 0

BEGIN

INSERT INTO dd\_CharacterAdventure (AdventureID, CharacterID)

VALUES (@adventureid, @characterid)

END

END

GO

--This Procedure allows a user to input new items, assign them to adventures, and equip them to characters if desired

--ItemDescription, AdventureID and CharacterID can both be null,

--Future Procedures will allow users to update item status as well as who is equipping the item

CREATE PROCEDURE dd\_ItemInput (@itemname varchar(50), @itemdescription varchar(500), @adventurename varchar(50), @charactername varchar(20)) AS

BEGIN

DECLARE @adventureid int = dbo.dd\_AdventureLookup(@AdventureName)

DECLARE @characterID int = dbo.dd\_CharacterLookup(@CharacterName)

IF @adventureid = 0

SELECT 'Please Refine Adventure Input'

IF @adventureid = 0

SELECT 'Please Refine Character Input'

IF @adventureid > 0 AND @characterID > 0

BEGIN

INSERT INTO dd\_Item (ItemName, ItemDescription, AdventureID, CharacterID)

VALUES (@itemname, @itemdescription, @adventureid, @characterid)

END

END

GO

--These Two Functions allow Users to Update dd\_MultiClass,

--It is designed so it can only be manipulated one character level at a time as it is the most common situation

--This procedure will level up a character's class once

--It will create an entry in dd\_MultiClass, if a character does not already have a level in the selected class

CREATE PROCEDURE dd\_LevelUp (@CharacterName varchar(30), @CharacterClass varchar(20)) AS

BEGIN

DECLARE @MultiClass int = 0

DECLARE @characterlevel int

DECLARE @characterid int

DECLARE @characterclassID int

SELECT @characterid = dbo.dd\_CharacterLookup (@CharacterName)

SELECT @characterclassID = dbo.dd\_ClassLookup (@CharacterClass)

SELECT @characterlevel = LevelAmount FROM dd\_MultiClass WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

SELECT @MultiClass = MultiClassID FROM dd\_MultiClass WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

IF @characterid = 0

SELECT 'Please Refine Character Input'

ELSE

BEGIN

IF @MultiClass = 0

INSERT INTO dd\_MultiClass (CharacterClassID, CharacterID, LevelAmount) VALUES (@CharacterClassID, @CharacterID, 1)

ELSE

UPDATE dd\_MultiClass SET LevelAmount = 1+@characterlevel WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

SELECT 'Character Updated: ', CharacterName FROM dd\_Character WHERE CharacterID = @characterid

SELECT CharacterClassText, LevelAmount FROM dd\_Character

JOIN dd\_MultiClass ON dd\_Character.CharacterID = dd\_MultiClass.CharacterID

JOIN dd\_CharacterClass ON dd\_MultiClass.CharacterClassID = dd\_CharacterClass.CharacterClassID

WHERE dd\_Character.CharacterID = @CharacterID

END

END

GO

--This procedure will level down a character's class once

--It will delete the class entry for that character if the level amount reaches zero

--And it wont do anything if a character doesn't have the class to begin with

CREATE PROCEDURE dd\_LevelDown (@CharacterName varchar(30), @CharacterClass varchar(20)) AS

BEGIN

DECLARE @MultiClass int = 0

DECLARE @characterlevel int

DECLARE @characterid int

DECLARE @characterclassID int

SELECT @characterid = dbo.dd\_CharacterLookup (@CharacterName)

SELECT @characterclassID = dbo.dd\_ClassLookup (@CharacterClass)

SELECT @characterlevel = LevelAmount FROM dd\_MultiClass WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

SELECT @MultiClass = MultiClassID FROM dd\_MultiClass WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

IF @characterid = 0

SELECT 'Please Refine Character Input'

ELSE

BEGIN

IF @CharacterLevel = 1

DELETE dd\_MultiClass Where CharacterID = @characterID AND CharacterClassID = @CharacterClassID

ELSE

UPDATE dd\_MultiClass SET LevelAmount = @characterlevel-1 WHERE CharacterID = @characterID AND CharacterClassID = @CharacterClassID

SELECT 'Character Updated: ', CharacterName FROM dd\_Character WHERE CharacterID = @characterid

SELECT CharacterClassText, LevelAmount FROM dd\_Character

JOIN dd\_MultiClass ON dd\_Character.CharacterID = dd\_MultiClass.CharacterID

JOIN dd\_CharacterClass ON dd\_MultiClass.CharacterClassID = dd\_CharacterClass.CharacterClassID

WHERE dd\_Character.CharacterID = @CharacterID

END

END

GO

--This Batch of Procedures will update the statuses of various elements

--This procedure will end a current adventure, changing its status and defaulting the enddate to time of execution

CREATE PROCEDURE dd\_EndAdventure (@adventurename varchar(50), @status varchar(20)) AS

BEGIN

DECLARE @statusid int = dbo.dd\_AdventureStatusLookup (@status)

DECLARE @adventureid int = dbo.dd\_AdventureLookup (@adventurename)

IF @adventureid = 0

SELECT 'Please Refine Adventure Input'

ELSE

BEGIN

UPDATE dd\_Adventure SET AdventureStatusID = @statusid WHERE AdventureID = @adventureid

UPDATE dd\_Adventure SET AdventureEndDate = GETDATE() WHERE AdventureID = @adventureid

SELECT 'Adventure Updated:'

SELECT AdventureName, AdventureStartDate, AdventureEndDate, AdventureStatusText FROM dd\_Adventure

JOIN dd\_AdventureStatus ON dd\_AdventureStatus.AdventureStatusID = dd\_Adventure.AdventureStatusID

WHERE AdventureID = @adventureID

END

END

GO

--This Procedure will Update an Item's Status to statuses that are not "Equipped"

CREATE PROCEDURE dd\_ItemStatusUpdate (@Iname varchar (30), @status varchar(20)) AS

BEGIN

DECLARE @ItemID int = dbo.dd\_ItemLookup (@Iname)

DECLARE @ItemstatID int = dbo.dd\_ItemStatusLookup (@status)

IF @status = 'Eqipped'

SELECT 'Use Equip procedure'

ELSE

BEGIN

UPDATE dd\_Item SET ItemStatusID = @ItemstatID WHERE ItemID = @ItemID

SELECT 'Item Updated:', ItemName, ItemStatusText FROM dd\_Item

JOIN dd\_ItemStatus ON dd\_Item.ItemStatusID = dd\_ItemStatus.ItemStatusID

WHERE ItemID = @ItemID

END

END

GO

--This Procedure will Update an item to "equipped" status and update which character is doing the equipping

CREATE PROCEDURE dd\_Equip (@itemname varchar(50), @charname varchar (30)) AS

BEGIN

DECLARE @itemID int = dbo.dd\_ItemLookup (@itemname)

DECLARE @characterID int = dbo.dd\_CharacterLookup (@charname)

DECLARE @statusID int = dbo.dd\_ItemStatuslookup ('Equipped')

IF @characterid = 0

SELECT 'Please Refine Adventure Input'

ELSE

BEGIN

UPDATE dd\_Item SET CharacterID = @characterID WHERE ItemID = @itemid

UPDATE dd\_Item SET ItemStatusID = @statusID WHERE ItemID = @itemid

SELECT CharacterName, ItemName, ItemStatustext FROM dd\_Character

LEFT JOIN dd\_Item ON dd\_Item.CharacterID = dd\_Character.CharacterID

LEFT JOIN dd\_ItemStatus ON dd\_Item.ItemStatusID = dd\_ItemStatus.ItemStatusID

WHERE dd\_Character.CharacterID = @characterID

END

END

GO

--This Procedure will update a character's current adventuring status

CREATE PROCEDURE dd\_CharacterStatusUpdate (@charname varchar (30), @status varchar(20)) AS

BEGIN

DECLARE @characterID int = dbo.dd\_CharacterLookup (@charname)

DECLARE @charstatID int = dbo.dd\_CharacterStatusLookup (@status)

IF @characterid = 0

SELECT 'Please Refine Character Input'

ELSE

BEGIN

UPDATE dd\_Character SET CharacterStatusID = @charstatID WHERE CharacterID = @characterID

SELECT 'Character Updated:', CharacterName, CharacterStatusText FROM dd\_Character

JOIN dd\_CharacterStatus ON dd\_Character.CharacterStatusID = dd\_CharacterStatus.CharacterStatusID

WHERE CharacterID = @characterID

END

END

GO

--Aggregate Functions for data questions Views created for all,

--the ORDER BY statement to be used with the views were commented out after crafting the SELECT statement

--Which Player has been given the Most items?

--Utilizes the COUNT function to figure out how many items a character has,

--Then adds each character together per player

CREATE VIEW dd\_MostItemsRecieved AS

SELECT PlayerFirstName +' ' + PlayerLastName AS PlayerName,

COUNT(ItemID) AS NumberofItems FROM dd\_Player p

JOIN dd\_Character c ON p.PlayerID = c.PlayerID

JOIN dd\_Item i ON i.CharacterID = c.CharacterID

GROUP BY PlayerLastName, PlayerFirstName

GO

--Which GM has handed out the Most Items?

--COUNTS the number of items given per adventure, then ascribes that to each gm

CREATE VIEW dd\_MostItemsGiven AS

SELECT

PlayerFirstName +' ' + PlayerLastName AS GameMasterName,

COUNT(ItemID) AS ItemsDistributed

FROM dd\_Player p

JOIN dd\_GameMaster g ON p.PlayerID = g.PlayerID

JOIN dd\_Adventure a ON a.GameMasterID = g.GameMasterID

JOIN dd\_Item i ON i.AdventureID = a.AdventureID

GROUP BY PlayerLastName, PlayerFirstName

GO

--Which Characters have undergone the Most Adventures,

--not very interesting now, but will be more interesting as the database increases

--Counts the characters that have been in the most adventures

CREATE VIEW dd\_MostAdventures AS

SELECT TOP(5)

CharacterName,

COUNT(AdventureID) AS NumberofAdventures

FROM dd\_Character c

JOIN dd\_CharacterAdventure ca ON ca.CharacterID = c.CharacterID

GROUP BY CharacterName

GO

--Which Class of Characters do players tend to play

--Aggregates the total number of character levels achieved per class type

CREATE VIEW dd\_MostPlayedClasses AS

SELECT

CharacterClassText,

SUM(LevelAmount) AS NumberofCharacterLevels,

COUNT(CharacterID) AS NumberofCharacters

FROM dd\_CharacterClass cc

JOIN dd\_MultiClass mc on cc.CharacterClassID = mc.CharacterClassID

GROUP BY CharacterClassText

GO

--Which Race of Characters do Players tend to play

--Aggregates number of characters that have each Race type

CREATE VIEW dd\_MostPlayedRaces AS

SELECT

CharacterRaceText,

COUNT(CharacterID) AS NumberofCharacterRaces

FROM dd\_CharacterRace cr

JOIN dd\_Character c on cr.CharacterRaceID = c.CharacterRaceID

GROUP BY CharacterRaceText

GO

--Which GM has had the deadliest Adventure?

--Counts characters that have the deceased status and aggregates them based on adventure

--Also displays gm that was responsible for that adventure

CREATE VIEW dd\_Deadliest AS

SELECT PlayerLastName, AdventureName, CharacterStatusText, COUNT(c.CharacterID) AS CharacterStatus FROM dd\_Character c

JOIN dd\_CharacterStatus cs ON c.CharacterStatusID = cs.CharacterStatusID

JOIN dd\_CharacterAdventure ca ON c.CharacterID = ca.CharacterID

JOIN dd\_Adventure a ON ca.AdventureID = a.AdventureID

JOIN dd\_GameMaster gm ON a.GameMasterID = gm.GameMasterID

JOIN dd\_Player p ON gm.PlayerID = p.PlayerID

WHERE CharacterStatusText = 'Deceased'

GROUP BY playerLastName, AdventureName, CharacterStatusText

GO

--Which Player has Made the Most Characters?

--Counts number of characters per player

CREATE VIEW dd\_MostCharacters AS

SELECT PlayerFirstName + ' ' + PlayerLastName AS PlayerName, COUNT(CharacterID) AS NumberofCharacters FROM dd\_Player p

JOIN dd\_Character c ON p.PlayerID = c.PlayerID

GROUP BY PlayerFirstName + ' ' + PlayerLastName

GO

--Which Player has Accumulated the most levels?

--SUMs total level amounts for all characters associated with each player

CREATE VIEW dd\_MostLevels AS

SELECT PlayerFirstName + ' ' + PlayerLastName AS PlayerName, SUM(LevelAmount) AS TotalLevels FROM dd\_Player p

JOIN dd\_Character c ON p.PlayerID = c.PlayerID

JOIN dd\_MultiClass mc ON c.CharacterID = mc.CharacterID

GROUP BY PlayerFirstName + ' ' + PlayerLastName

GO

/\*

This view is a handful of joins, but it will show nearly all the data in the database cultivated down to

a single easy to read report.

\*/

CREATE VIEW dd\_GMAdventureReport AS

SELECT [dbo].[dd\_GMNameLookup] (DD\_GameMaster.GameMasterID) AS 'Game Master', AdventureName, PlayerFirstName + ' ' +PlayerLastName AS 'Player Name', CharacterName, CharacterRaceText, CharacterClassText, LevelAmount

FROM dd\_Character

JOIN dd\_CharacterAdventure ON dd\_CharacterAdventure.CharacterID = dd\_character.CharacterID

JOIN dd\_Adventure ON dd\_Adventure.AdventureID = dd\_CharacterAdventure.AdventureID

JOIN dd\_GameMaster ON dd\_Adventure.GameMasterID = dd\_GameMaster.GameMasterID

JOIN dd\_CharacterRace ON dd\_CharacterRace.CharacterRaceID = dd\_Character.CharacterRaceID

JOIN dd\_Player ON dd\_Player.PlayerID= dd\_Character.PlayerID

JOIN dd\_MultiClass ON dd\_MultiClass.CharacterID = dd\_Character.CharacterID

JOIN dd\_CharacterClass ON dd\_MultiClass.CharacterClassID = dd\_CharacterClass.CharacterClassID

GO

**Data Creation**

--Populating Glossary Tables, these wont be Changed all too frequently if ever

--These are fairly straightforward

--Character Status

INSERT INTO dd\_CharacterStatus (CharacterStatusText)

VALUES ('Active'), ('Retired'), ('Missing'), ('Deceased')

GO

--Adventure Status

INSERT INTO dd\_AdventureStatus (AdventureStatusText)

VALUES ('In Progress'), ('Completed'), ('Failed'), ('Prepared')

GO

--Item Status

INSERT INTO dd\_ItemStatus (ItemStatusText)

VALUES ('Equipped'), ('Stored'), ('Undiscovered'), ('Lost')

GO

--Character Race

INSERT INTO dd\_CharacterRace (CharacterRaceText)

VALUES ('Dwarf'), ('Elf'), ('Halfling'), ('Human'), ('DragonBorn'), ('Gnome'), ('Half-Elf'),

('Half-Orc'), ('Tiefling'), ('Genasi'), ('Leonin'), ('Aasimar'), ('Tabaxi'), ('Goblin'), ('Firbolg')

GO

--Character Class

INSERT INTO dd\_CharacterClass (CharacterClassText)

VALUES ('Barbarian'), ('Bard'), ('Cleric'), ('Druid'), ('Fighter'), ('Monk'), ('Paladin'),

('Ranger'), ('Rouge'), ('Sorcerer'), ('Warlock'), ('Wizard'), ('Arcanist'), ('CardCaster')

GO

--CharacterBackground

INSERT INTO dd\_CharacterBackground (CharacterBackgroundText)

VALUES ('Acolyte'), ('Charlatan'), ('Criminal'), ('Entertainer'), ('Folk Hero'), ('Guild Artisan'), ('Hermit'),

('Noble'), ('Outlander'), ('Sage'), ('Sailor'), ('Soldier'), ('Urchin')

GO

--Initially Populating Data insertion Tables

--Populating dd\_Player, All data required

INSERT INTO dd\_Player (PlayerFirstName, PlayerLastName, PlayerBirthDate, PlayerCity, PlayerState)

VALUES

('Jeff', 'Bertucci', '06/22/1993', 'Marlborough', 'MA')

--Populating dd\_GameMaster

--Only a select number of players have GameMastered games

INSERT INTO dd\_GameMaster (PlayerID)

VALUES

(1)

--Populating dd\_Adventure,

--StartDate and EndDate are both defaulted to get the current datetime at time of execution,

--The thought process being that when one inserts the data, the campaign will have just started,

--Hence the default Status text being "In Progress"

--Users can change the enddate and status text by completing the campaign using a function

INSERT INTO dd\_Adventure (AdventureName, AdventureLocation, AdventureDescription, AdventureStartDate, GameMasterID, AdventureStatusID)

VALUES

('The Blood War', null, 'Dungeons have sprung up across the countryside', '06/22/2018', 3, 1) --End Date if not added Defaults to date added to database

INSERT INTO dd\_Adventure (AdventureName, AdventureLocation, AdventureDescription, AdventureStartDate, AdventureEndDate, GameMasterID, AdventureStatusID)

VALUES

('20M40Y', 'Zirkonia', 'A mysterious vortex has opened up', '04/13/2018', '05/06/2018', 2, 2)

--Populating dd\_Character, CharacterCreateDate defaults to the time data was input,

--CharacterBackground and characterdescription are optional

INSERT INTO dd\_Character (CharacterName, PlayerID, CharacterRaceID, CharacterBackgroundID, CharacterStatusID, CharacterDescription)

VALUES

('Gerome Garrick', 3, 6, null, 1, 'A hyper kid who only wants to get stronger')

--Populating dd\_CharacterAdventure, attaches characters to adventures

INSERT INTO dd\_CharacterAdventure (CharacterID, AdventureID)

VALUES

(7, 1)

--Populating dd\_MultiClass

INSERT INTO dd\_MultiClass (CharacterID, CharacterClassID, LevelAmount)

VALUES

(1, 6, 5)

--Populating dd\_Item, ItemDescription is optional,

--I don't have a good library of unequipped items, but that will grow with the database,

--ItemStatus defaults to "Equipped" as that is the most common status for items,

--Procedures may be used to change this

INSERT INTO dd\_Item (CharacterID, AdventureID, ItemName, ItemDescription)

VALUES

(7, 1, 'Twincast Clubs', 'Two clubs magic''ed together into oneish'

**Data Manipulation**

For Data manipulation, I crafted procedures using update and delete. Those procedures are listed with the other database objects on the first section. The statements below show them in action.

--These Execute statements are an example of the procedures that were created

--The purpose is to manipulate current lists with new or updated information

EXECUTE dd\_PlayerInput 'John', 'Smith', '07/04/1997', 'Phillidelphia', 'PA' --Adds Player Data

EXECUTE dd\_MakeGM 'Smith' --Upgrades player to GameMaster Status only gamemasters can make adventures

EXECUTE dd\_AdventureInput 'Smith','John''s Great Wacky Fun Time', 'here', 'He finally did it' --Adds an adventure

EXECUTE dd\_CharacterInput 'Smith', 'JohnSmith', 'Human', 'Soldier', 'John''s Wild and Crazy Guy' --Inputs a character

EXECUTE dd\_AdventureParty 'JohnS','John''s'--Links a Character with an adventure

EXECUTE dd\_LevelUp 'JohnS', 'Fighter' --Levels up a character once

EXECUTE dd\_LevelDown 'JohnS', 'Fighter' --Levels down a character once

EXECUTE dd\_ItemInput 'John''s Thing', 'It is John''s thing', 'John''s', 'JohnS' --adds an Item

EXECUTE dd\_EndAdventure 'John''s', 'Failed' --Ends an adventure with a given status

EXECUTE dd\_ItemStatusUpdate 'John''s Thing', 'stored' --Updates an Item's status

EXECUTE dd\_Equip 'John''s', 'JohnS' --Equips an item to a specific character

EXECUTE dd\_CharacterStatusUpdate 'JohnS', 'Deceased' --Updates a character to a certain status

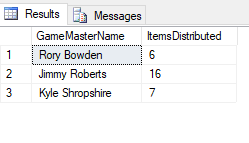
**Answering Data Questions**

For data questions, I created views to see them, the code for those views is listed in the first section of this part. Below are the select statements and resulting screenshots

--Which GM has handed out the Most Items?

--COUNTS the number of items given per adventure, then ascribes that to each gm

SELECT \* FROM dd\_MostItemsGiven

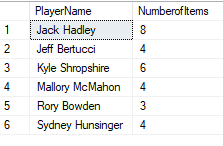


--Which Player has been given the Most items?

--Utilizes the COUNT function to figure out how many items a character has,

--Then adds each character together per player

SELECT \* FROM dd\_MostItemsRecieved

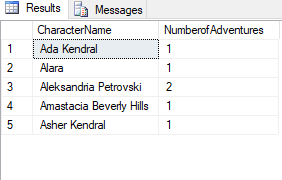


--Which Characters have undergone the Most Adventures,

--not very interesting now, but will be more interesting as the database increases

--Counts the characters that have been in the most adventures

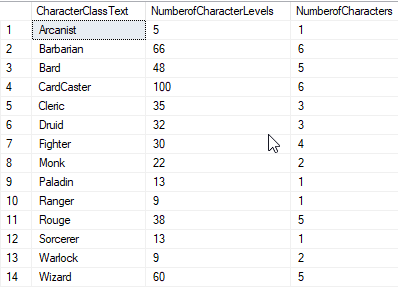
SELECT \* FROM dd\_MostAdventures



--Which Class of Characters do players tend to play

--Aggregates the total number of character levels achieved per class type

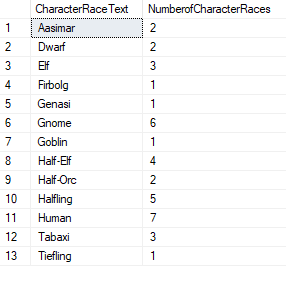
SELECT \* FROM dd\_MostPlayedClasses



--Which Race of Characters do Players tend to play

--Aggregates number of characters that have each Race type

SELECT \* FROM dd\_MostPlayedRaces

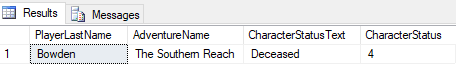


--Which GM has had the deadliest Adventure?

--Counts characters that have the deceased status and aggregates them based on adventure

--Also displays gm that was responsible for that adventure

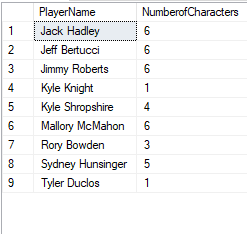
Select \* FROM dd\_Deadliest



--Which Player has Made the Most Characters?

--Counts number of characters per player

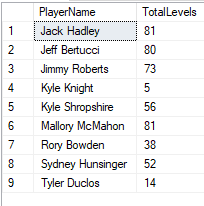
SELECT \* FROM dd\_MostCharacters



--Which Player has Accumulated the most levels?

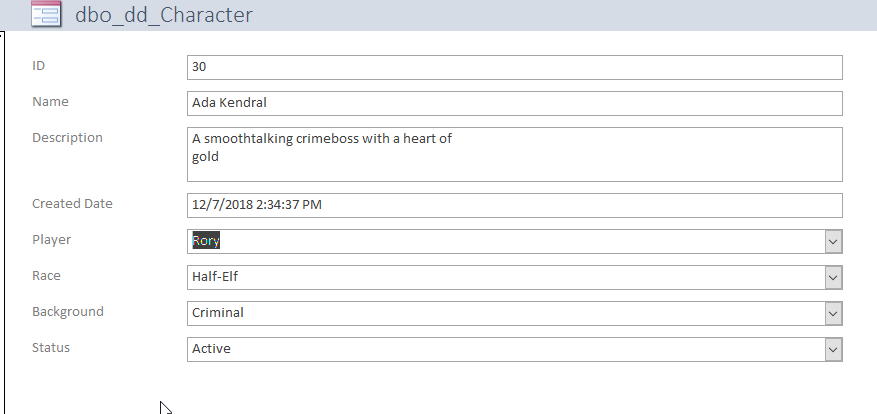
--SUMs total level amounts for all characters associated with each player

SELECT \* FROM dd\_MostLevels



**Implementation**

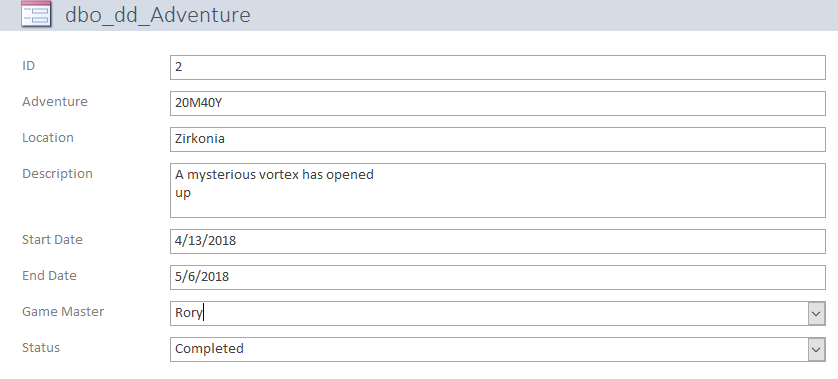
**Character Form**

****

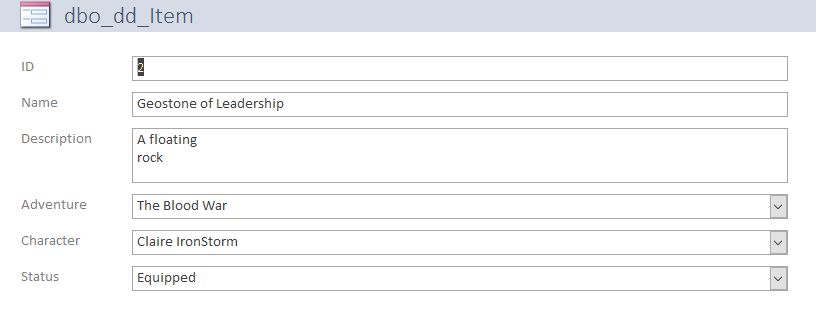
**Player Form**

****

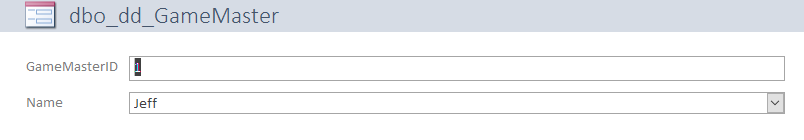
**Adventure Form**

****

**Item Form**

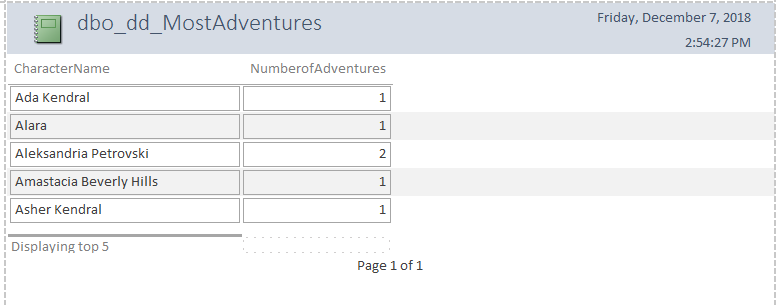
****

**Game Master Form**

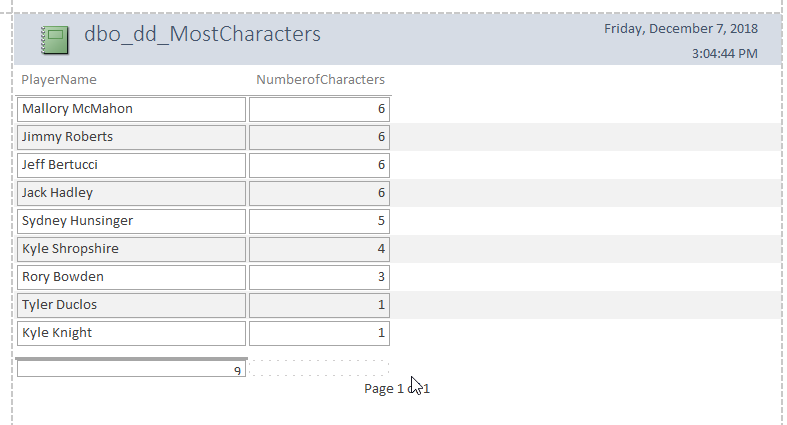
****

**Data Questions**

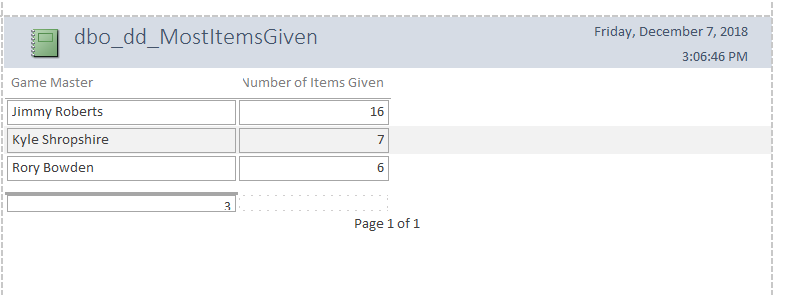
**Most Adventures per Character**

****

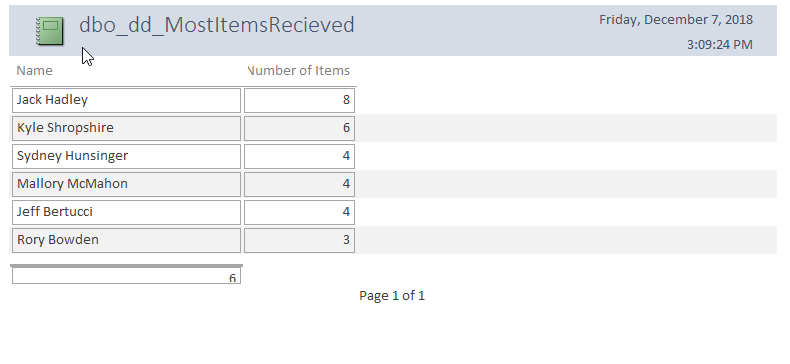
**Most Characters per Player**

****

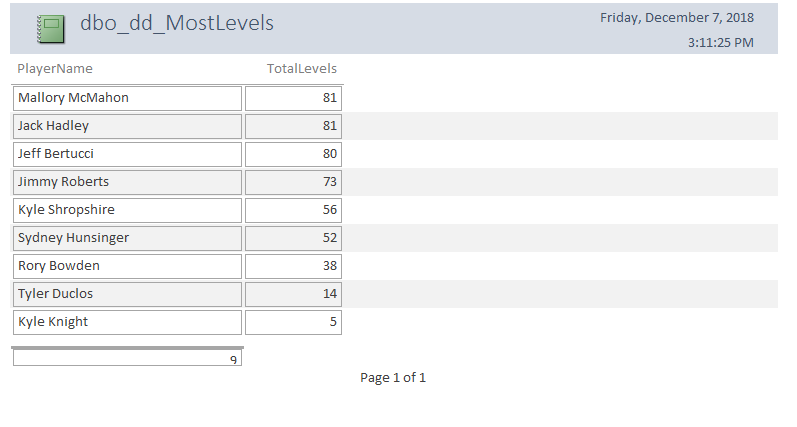
**Most Items Given by Game Master**

****

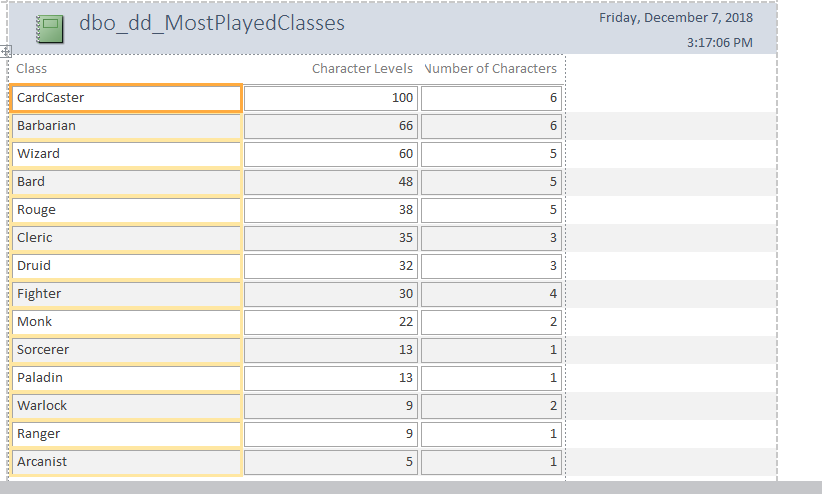
**Player who received the most items**

****

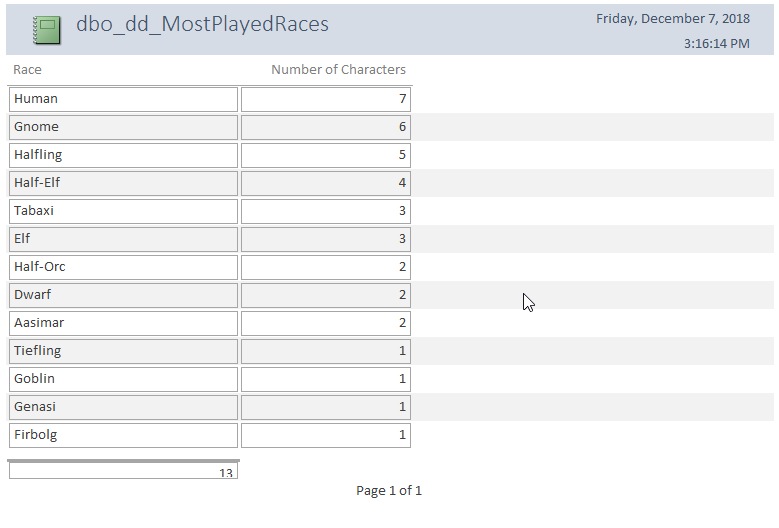
**Player Who has acquired the most levels**

****

**Classes that are played the most**

****

**Races that are played the most**

****

**Reflection**

At the start of this project, I was extremely overwhelmed. I had never done anything like this before and I didn’t have the slightest grasp on SQL coding. I had assumed this would be a painful and grueling process. However, when I sat down to get this done, I loved doing the work. It was a little much at times, but I never felt like I couldn’t handle the task at hand. It was very methodical and in many ways calming. The way I described it was organizing clutter into something nice and put together, and looking at my end products felt incredible. I am incredibly proud of the work I accomplished, especially considering that this is all new.

If I were to do this project again, I would certainly change a few things. I might cut down on the initial number of columns in tables. That is definitely where I got bogged down the most, trying to fill out everything on a table. Additionally, I would take extra careful steps during the table creation process. At the beginning of part 2, I had accidentally made a column the wrong type (an int instead of char). At the end of my coding I realized my mistake, but I had to reinsert everything back into my tables and the identities were all different all over my database. That was certainly a hurdle.

I also had to diverge from a couple guidelines that I set out in part one of the project. The biggest one being that I condensed the GameMaster Table down to just two columns, rather than essentially a duplicate of the player column. Since Gamemasters can play in games that they are not specifically GameMastering, that change made sense and it cut down on duplication of data. The other change that I had to make from my original design was that per my business rules, adventures did not have to be named, this was done to maintain a level of confidentiality that was ultimately not worth pursuing since Game Masters could simply use a filler name. Not having the name only served to create more problems and complications when it came to manipulating the data.

Going forward, I know that I have gained invaluable experience, and doing this project makes me seriously consider doing something like this when I get into my professional career. And really, no matter what I do with my career, these are the first building blocks for sure on what I want to do moving forward and I appreciate immensely the experience that I had.