Matthew Ryder

Hearthstone Battlegrounds

IST 659 Final Project

A picture containing cake, birthday, table, sitting

Description automatically generated

# **Summary**

I would like to design a database which holds all the information for the minions in Hearthstone’s popular battlegrounds game mode. The purpose of this database will be to best house all the information for the different minions. This will allow for the cards to be accessed easily by the application as well as allow for players to view the different available minions so that they could theorize before they enter a game. Some examples of these minions include the following:



Any other information can be found at <https://playhearthstone.com/en-us/battlegrounds?type=minion>

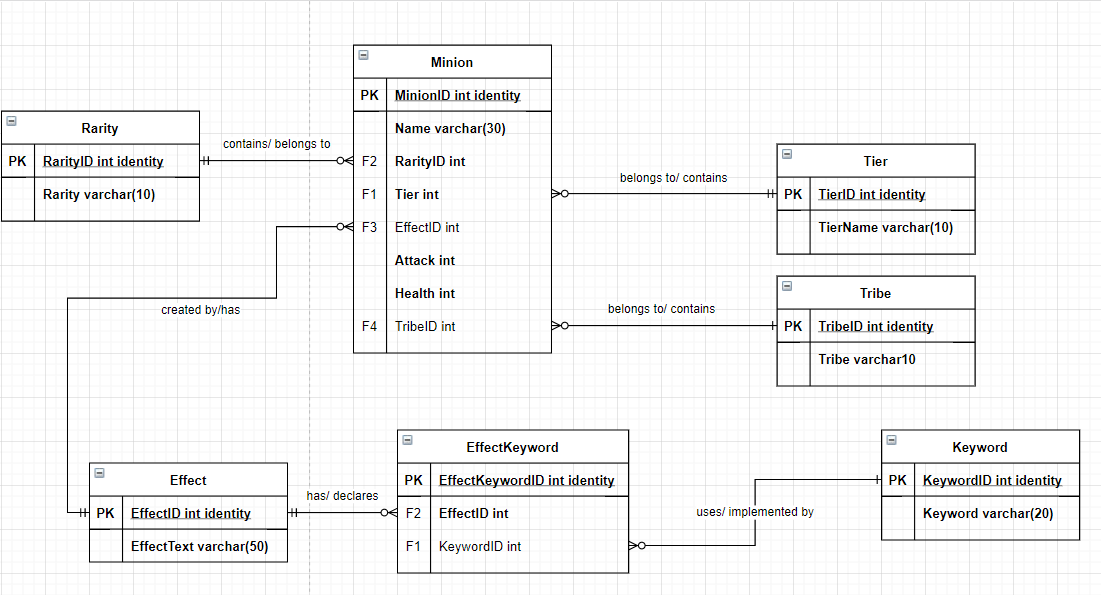
# **Business Rules**

* Every minion must have a name
* Every minion must have a rarity
  + The rarities will be constrained in our “Rarity” entity containing Basic, Common, Rare, Epic, and Legendary
* Every minion must have a tier
  + The tier will be constrained in our “Tier” entity containing tiers 1, 2, 3, 4, 5, and 6
* Minions have an effect; this is not required because some effects create minions which do not have their own effect
  + The bolder words on the cards show above are keywords, they help to implement the effects in plain text. Because an effect can contain 0 or many keywords, we have created an associative entity to join our keywords to their respective effects
  + Our “Keyword” entity will constrain our keywords to Battlecry, Deathrattle, Taunt, Divine Shield, Start of Combat, Charge, Magnetic, Reborn, Poisonous, Overkill, Immune, Discover, Windfury, Adapt

# **Stakeholders**

This database will be important to a few different groups of people. Primarily, it will be important to developers at ActivisionBlizzard as this is their game and their data. Secondly, it will be important to players and fans of Hearthstone since most players want to access the different minions that will be available to them during games and like to theorize so they are in the best position to perform well when they enter into a game. The final group this could be useful for, is third parties which would host tournaments or utilize gameplay data. Access to a well-maintained database will allow these groups to ensure they can access the correct information so that they do not share improper information due to our mistakes.

# **Logical Model**



# **Database Goals**

The goal for this database will be to include all our different minions so that an application can be developed around the data housed within it. This will allow the front facing application to collect all necessary information on our minions. We should also be able to query the database so that we can sort minions by tribe, keywords, or tier so that players and fans can see what a game may entail without having to play a game. Also, we will be able to efficiently exclude minions, as each game excludes 1 tribe.

# **Physical Database Design**

## Table Creation

All code below within this section creates table as they are defined in the Logical Model above, the only change was Effect.EffectText was changed from varchar(50) to varchar(500) as text was being cut off as a varchar(50).

create table Rarity (

RarityID int identity,

Rarity varchar(10) not null,

constraint pk\_rarity primary key (RarityID),

constraint u1\_rarity unique(Rarity)

)

create table Effect (

EffectID int identity,

EffectText varchar(500) not null,

constraint pk\_effect primary key (EffectID),

constraint u1\_effect unique(EffectText)

)

create table Tier (

TierID int identity,

TierText varchar(5) not null,

constraint pk\_tier primary key (TierID),

constraint u1\_tier unique(TierText)

)

create table Tribe (

TribeID int identity,

Tribe varchar(10) not null,

constraint pk\_tribe primary key (TribeID),

constraint u1\_tribe unique(Tribe)

)

create table Keyword (

KeywordID int identity,

Keyword varchar(20) not null,

constraint pk\_keyword primary key (KeywordID),

constraint u1\_keyword unique(Keyword)

)

create table EffectKeyword (

EffectKeywordID int identity,

EffectID int not null,

KeywordID int,

constraint pk\_effectkeyword primary key (EffectKeywordID),

constraint fk1\_effectkeyword foreign key (EffectID) references Effect(EffectID),

constraint fk2\_effectkeyword foreign key (KeywordID) references Keyword(KeywordID)

)

create table Minion (

MinionID int identity,

Name varchar(30) not null,

RarityID int not null,

TierID int not null,

EffectID int,

Attack int not null,

Health int not null,

TribeID int,

constraint pk\_minion primary key (MinionID),

constraint fk1\_minion foreign key (TierID) references Tier(TierID),

constraint fk2\_minion foreign key (RarityID) references Rarity(RarityID),

constraint fk3\_minion foreign key (EffectID) references Effect(EffectID),

constraint fk4\_minion foreign key (TribeID) references Tribe(TribeID),

constraint u1\_minion unique(name)

)

## Procedure Creation

Our addEffect procedure will be used to add a new data to the Effect table. The user will input the text for this new effect and as long as it does not already exist will add it to our table.

create procedure addEffect(@effecttext varchar(500))

as begin

insert into Effect(EffectText)

values (@effecttext)

return @@identity

end

go

Next, the addMinion procedure will help populate our Minion table. A user will manually enter the minion’s name, attack, and health. Then they will input the RarityID, TierID, EffectID, and TribeID, which must already exist in their respective tables. The reason I chose to manually lookup and enter the ID’s was to give myself an extra check of my previous manual data input.

create procedure addMinion(@name varchar(30), @rarity int, @tier int, @effect int, @attack int, @health int, @tribe int)

as begin

insert into Minion(Name, RarityID, TierID, EffectID, Attack, Health, TribeID)

values (@name, @rarity, @tier, @effect, @attack, @health, @tribe)

return @@identity

end

go

Lastly, the addEK procedure will populate the EffectKeyword table allowing us to populate our Effect and Keyword table.

create procedure addEK(@effect int, @keyword int)

as begin

insert into EffectKeyword(EffectID, KeywordID)

values (@effect, @keyword)

return @@identity

end

go

# Data Creation

First, I added data to my table which will act as constraints for minion table. The first few tables only have a few rows within them and their insert statements can be seen below with a quick check to make sure my tables were populated correctly.

-- Rarity: Basic, Common, Rare, Epic, Legendary

insert into Rarity(Rarity)

values ('Basic'), ('Common'), ('Rare'), ('Epic'), ('Legendary')

select \* from Rarity

-- Tier: 1-6

insert into Tier(TierText)

values ('One'), ('Two'), ('Three'), ('Four'), ('Five'), ('Six')

select \* from Tier

-- Tribe: Beast, Demon, Mech, Murloc, none, Pirate, Dragon, All

insert into Tribe(Tribe)

values ('Beast'), ('Demon'), ('Mech'), ('Murloc'), ('Pirate'), ('Dragon'), ('All')

select \* from Tribe

-- Keyword: Windfury, Deathrattle, Divine Shield, Battlecry, Taunt, Adapt, Poisonous, Overkill, Immune, Discover, Magnetic, Reborn, Charge

insert into Keyword(Keyword)

values ('Windfury'), ('Deathrattle'), ('Divine Shield'), ('Battlecry'), ('Taunt'), ('Adapt'), ('Poisonous'), ('Overkill'), ('Immune'), ('Discover'), ('Magnetic'), ('Reborn'), ('Charge')

select \* from Keyword

Then I utilized the procedures outlined in the previous section, here are a few samples of each in action.

DECLARE @newEffectID int

EXEC @newEffectID = addEffect 'Deathrattle: Give this minions Attack to a random friendly minion.'

DECLARE @newEffectID int

EXEC @newEffectID = addEffect 'Reborn At the end of your turn, give another random friendly minion +1 Attack.'

DECLARE @newEffectID int

EXEC @newEffectID = addEffect 'Whenever you summon a Murloc, gain +1 Attack.'

DECLARE @newMinionID int

EXEC @newMinionID = addMinion 'Kindly Grandmother', 2, 2, 19, 1, 1, 1

DECLARE @newMinionID int

EXEC @newMinionID = addMinion 'Metaltooth Leaper', 3, 2, 20, 3, 3, 3

DECLARE @newMinionID int

EXEC @newMinionID = addMinion 'Murloc Warleader', 4, 2, 21, 3, 3, 4

DECLARE @newMinionID int

EXEC @newMinionID = addMinion 'Spawn of NZoth', 2, 2, 24, 2, 2, null

DECLARE @newMinionID int

EXEC @newMinionID = addEK 57, null

DECLARE @newMinionID int

EXEC @newMinionID = addEK 58, 4

DECLARE @newMinionID int

EXEC @newMinionID = addEK 58, 5

DECLARE @newMinionID int

EXEC @newMinionID = addEK 59, 5

# Data Manipulation

I utilized update statement to correct error in data creation, here is what was used.

update Effect set EffectText = 'At the end of your turn give a friendly minion of each minion type +2/+1'

where EffectID = 90

update Effect set EffectText = 'Deathrattle: Summon the first 2 friendly Mechs that died this combat.'

where EffectID = 105

Our data is generally static so other manipulation was not necessary for this project.

# Data Questions

Three main goals of this database were to be able query minions by tier, tribe, and keyword. We can do this with the follow queries:

select Name from Minion where TierID = 1 –TierID will vary for any other tier

select Name from Minion join Tribe on Minion.TribeID = Tribe.TribeID where Tribe.Tribe = 'Beast' – Tribe.Tribe will vary for each tribe

select Name from Minion join Effect on Minion.EffectID = Effect.EffectID

join EffectKeyword on Effect.EffectID = EffectKeyword.EffectID

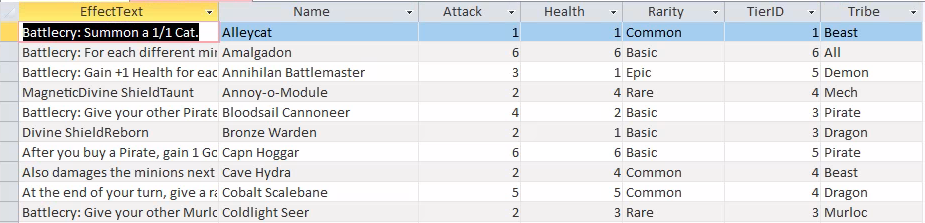
join Keyword on EffectKeyword.KeywordID = Keyword.KeywordID

where Keyword.Keyword = 'Battlecry' – Keyword.Keyword will change depending on our query

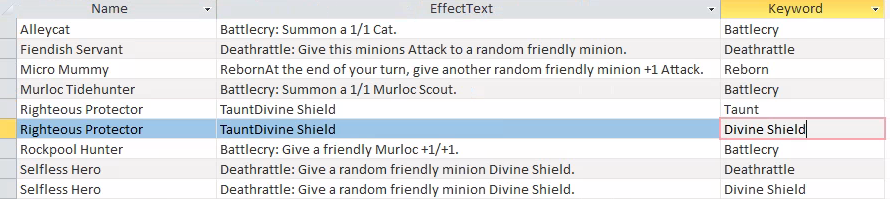
These queries will return the name of the Minion we want, and can be tailored to fit the scope of information we want presented in front of, but lay a good baseline for collecting all the information that we want out of this database.

# Implementation

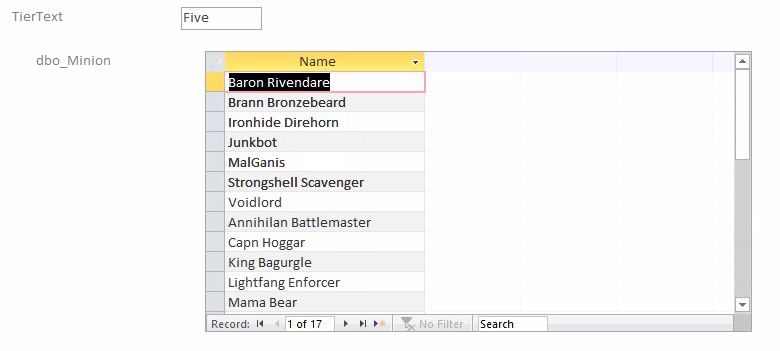
I utilized Access to best view my data. Here are some screenshots of what I visualized within access.



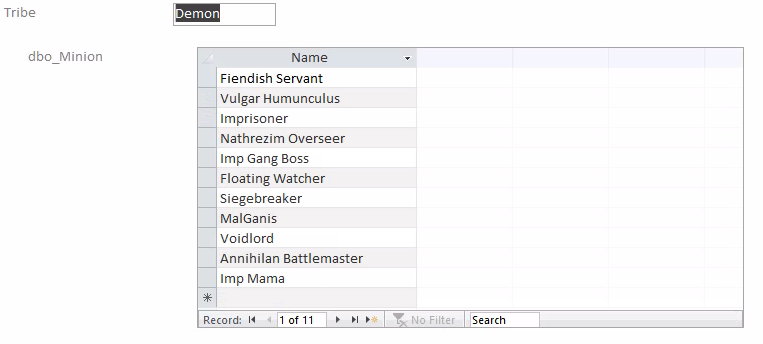
Here we can query our minions and get all their information



Here we are looking at the different keyword’s which associate with which minions and their effects.



Here we are grouping our minions by Tier so we can look at one tier at a time.



Lastly, we can do a similar query as the one above but instead sorted by tribe!

# Summary and Reflection

The goals of this database was to build a database which included all the minions in the Hearthstone Battlegrounds minion pool and to be able to find minions based on Tribe, Keyword, or Tier so that player could theorize without ever playing a game. With the UI implementation in Access the appropriate forms exist so that players could do exactly that, assuming they had access to Access. I was able to implement procedures for data entry with constraints making data entry as consistent as possible. I was able to answer my data question within Access and it allowed for a simple and straightforward way to query my data and display it in groups as I had hoped.

I think if I could have done things differently, I would have tested out different ways of setting up constraints to allow for null values. One example is, I run into an issue where a null in TribeID is not recognized by the Tribe entity, which made me wonder if I could have potentially required the ID and related and ID to null. Something I would have liked to try even though I could see there being issues with that as well. Overall, designing a database has shown me why having sound structure surrounding your data is so important. Without structure there are so many opportunities to create unnecessary mistakes. I caught myself making errors my data entry only to find my constraints catching those errors and keeping me on track. Also, prior to this course I never understood why IDs were used so frequently to associate entities within a database when you could instead input the “pretty” information. Now I understand that it constrains your data and enforces data quality.