Magic the Gathering Database Project Report

Carlos A. Rios

Derek Jones

April 29, 2017

Abstract

Magic the Gathering is a popular trading-card game with over 30,000 cards available since of April 29, 2017. We took upon the task of creating a database capable of holding all these cards in MySQL, and allowing the user to be able to search cards by specific search fields and allow users to create decks that are stored into the database. The design and implementation of the database was created and revised so that each relation in our schema was decomposed into Boyce-Codd Normal Form (BCNF). Designing the schema was a complicated task as it required a large amount of decomposition of information, such as separating set name from set code to allow BCNF from the entity MTGSet and so forth. The vast amount of information found in the Magic the Gathering rules, card, color, color identity, types, set, and formats was a complicated task. But nonetheless, we were able to resolve this problem by creating a relational model that supports BCNF and allows user to be able to search from the database to find a card(s) that meet specific requirements provided by them and allow construction of decks by user based on format.

Table of Contents

Introduction

Our project for Database Management System involved creating a small real-world database application and we decided to create a database for the trading-card game “Magic the Gathering” using Java and MySQL. Magic the Gathering has over 30,000 card released, as of May 2017, and with such an immense collection of cards, it would be ideal to create a database that stores all the cards and a application that allows users to pull from the database based on search fields. Each individual card has the following information on them:

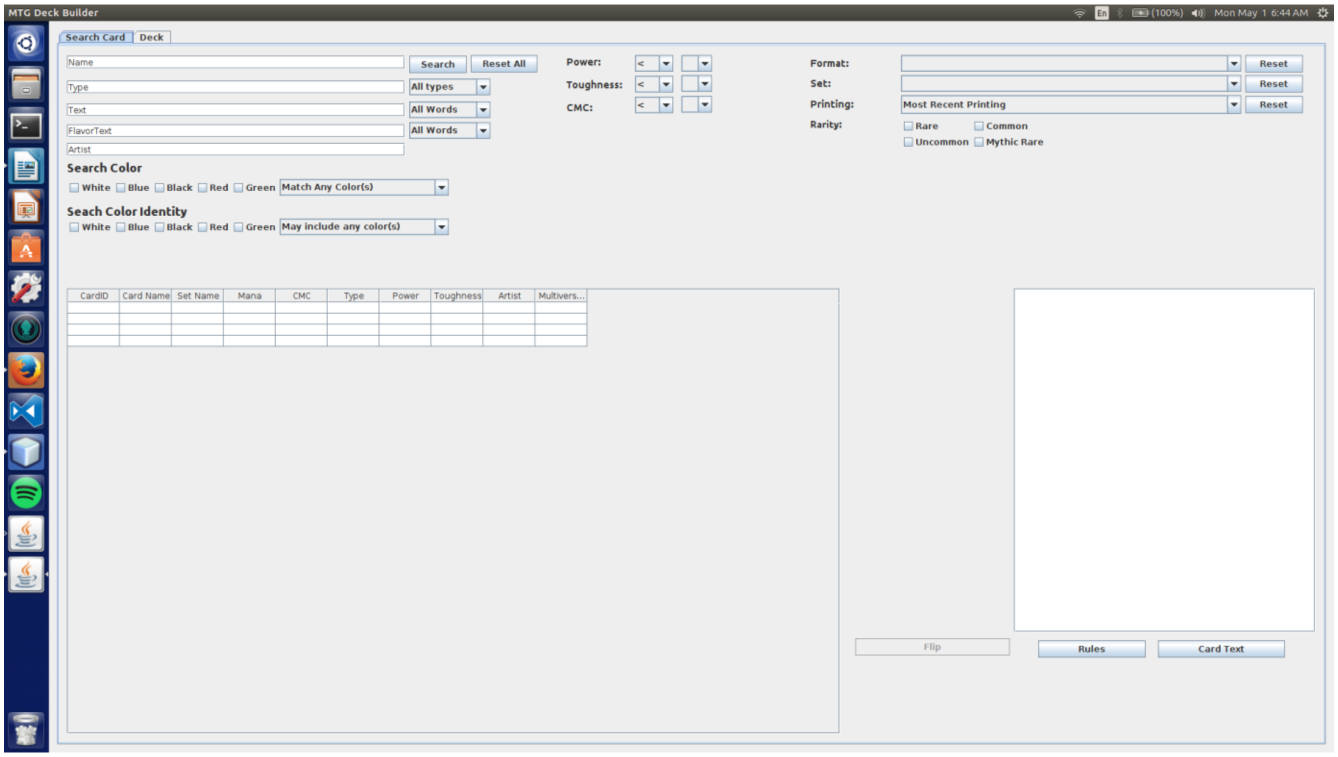
- Card Name (and the names of other card if layout is not standard)
- Mana Cost
- Converted Mana Cost (CMC)
- Colors
- Color Identity
- Types (includes Super Type, Type, and Sub Type)
- Text (includes Card Text, Flavor Text, and Ruling Text)
- Rarity
- Power
- Toughness
- Multiverse ID
- Artist
- Legalities
- Layout
- Loyalty
- Set Name

Though, there are more fields and information to an individual card, we decided that these were the necessities in order to create a database. With the help of MTGJson.com, we were able to obtain the JSON file, version 3.8.3, that included the most recent set, Amonkhet, that includes every set information and card information from those sets.[1] Every set includes the following information:

- Set Name
- Set Code
- Released Date
- Block
- Set Type

With the provided JSON file and Google's GSON, a JAR File with a library that can convert JSON into Java Object[2], we were able to parse the file and import the information into our database. To allow user an ease of access to the database, we also created a GUI, as seen below in Figure 1, with search fields that can be edit to fit the description of a card(s) that a user is looking for.

Figure 1- MTG Search GUI in Ubuntu 16.04



CardID	Card Name	Set Name	Mana	CMC	Type	Power	Toughness	Artist	Multivers...

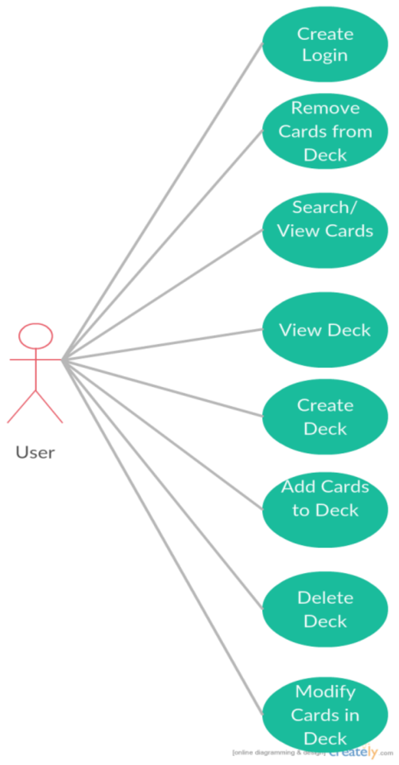
The process of creating the database proved difficult as the number of information and the normalization of such information challenged us with many design problem. In the next section, we will begin by introducing the product requirements, which list all actors and use cases by those actors. This will help us identity what it is that we will need in order to design the relational model and entity relationship (ER) diagram. By doing so, we can showcase what kind of problem we faced in designing such database and our approach to solve the problem.

Product Requirement

The actor, shown below Figure 2, is the User. In order to become a User of the system, they must first create a login account, which consist only of username and password. Afterwards, the user can do the following:

- Create New Login
- Search/View Cards
- View Deck
- Create Deck
- Delete Deck
- Add Cards to the Deck
- Remove Cards from the Deck
- Update Qty of Cards in the Deck

Figure 2 - Use Cases

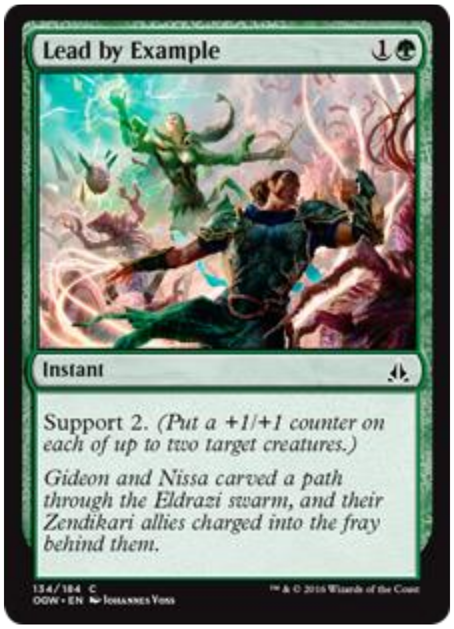


User can search/view cards based on the specified search fields, which are the following:

- Card Names
- Card Type
- Artist
- Card Text
- Flavor Text
- Color (Color and/or ColorIdentity)
- Cards Stats (CMC, Toughness, Power)
- Rarity
- Format
- Set
- Printing

All these fields narrow down the search result based only on what fields have been changed. Card name refers to the name of the card being searched for. Card type refers to type of the card, an example being a card that includes the type “Goblin” or “Legendary Creature Wizard”. Artist refers to the person who drew that card. Card text is the text on the card that is not a favor text and can be seen as what this can do if it has any special abilities. Flavor text refers to the lore/story of the card through some quote or description that is italicized. Color is simply the color of the card, usually based on the color symbol on the top right of the card. Card Stats are the cost it takes to cast the card and the power and toughness of the creature. Rarity refers to the ability of attaining the card from the set with common being the easiest to pull from a pack, followed by uncommon, then rare, and last is mythic rare being the rarest. Format refers to the format in which this card is legal or illegal to play in. Set refers to the set in which the card came from. The printing option refers to the printing of the card, which can be the original, reprints, newest card, or all the printings so far. An example of a card can be seen below as reference in Figure 3.

Figure 3 – MTG Card Image from Wizards of the Coast Gatherer[3]


Lead by Example
Instant
Support 2. (Put a +1/+1 counter on each of up to two target creatures.)
Gideon and Nissa carved a path through the Eldrazi swarm, and their Zendikari allies charged into the fray behind them.

View Deck cases provides us a way for users to be able to view their user created deck, if there any, which is an important component to deck building. Creating deck also helps user achieve this by allowing creation of the deck based on the Format they wish to construct in. Deleting deck helps user remove any unwanted decks they might have created. Adding card to the deck allows user to construct their decks by adding cards to it. Deleting card from the deck allows the removal of unwanted cards from the deck. And lastly, we have the ability to update the quantity of each card in the deck's mainboard and sideboard.

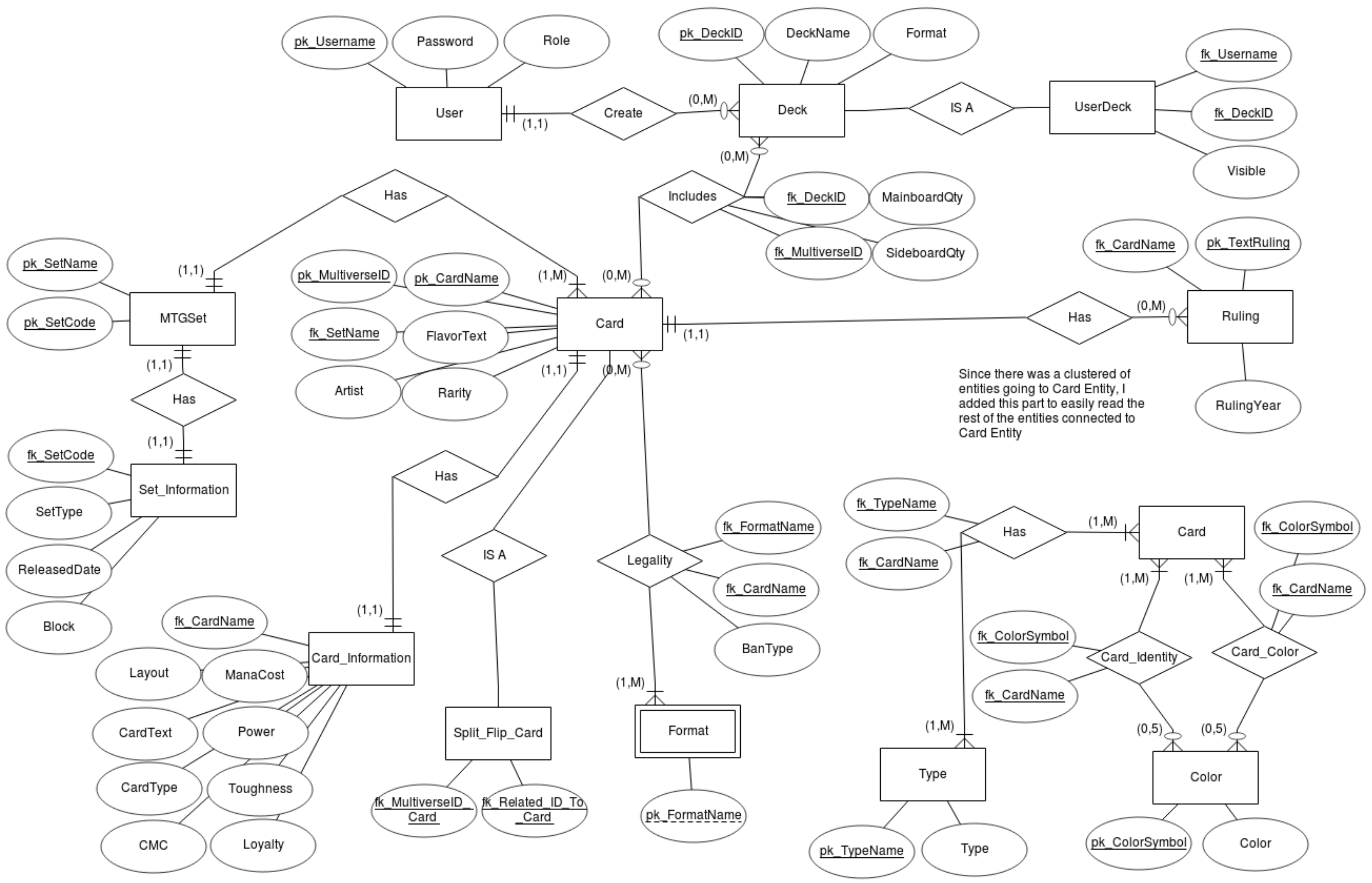
These use cases define the user actor in the database and in our application. These use cases grant the user the ability to search through the Magic the Gathering database filled with cards and allow them the ability to create a deck that is stored in the database. Currently, the only way to update the database is through running a Java file that parses the JSON file and inserts the information into their corresponding entities. The next section, we will go discuss E/R diagram, relational model, and BCNF verification of the database. More details on the use cases will be provided in the Design-Part 2 section of this report.

Design-Part 1: E/R Diagram, Relational Model, and BCNF Verification

E/R Diagram

The ER diagram for the database was created and organized in the following manner, as shown below in Figure 4:

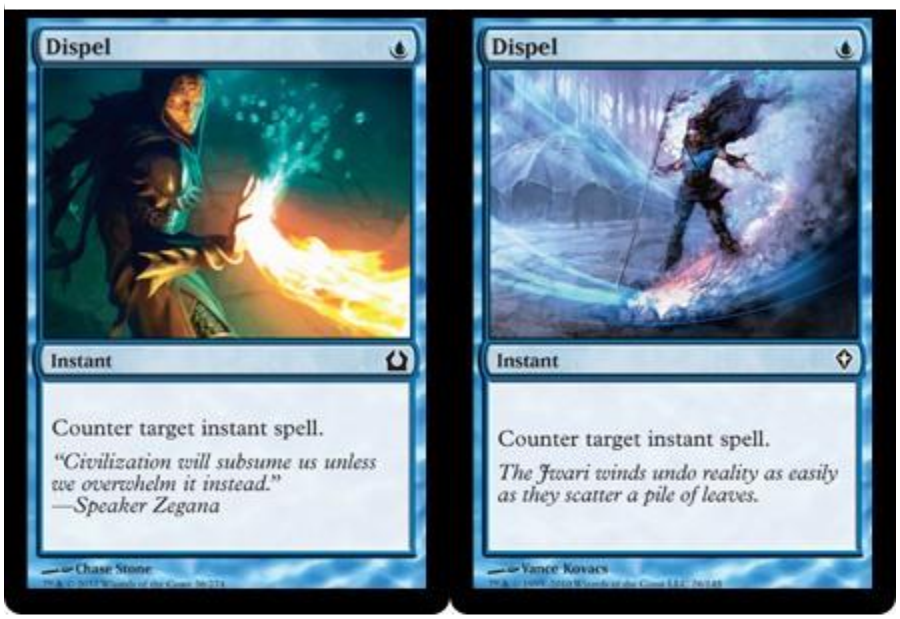
Figure 4 – ER Diagram



There are a total of 12 entities: MTGSet, User, Deck, UserDeck, Card, Ruling, Color, Type, Format, Split_Flip_Card, Card_Information, and Set_Information. Also, there a total of 5 entity relationship: Card_has_Card_Information, Card_legality_Format, Card_card_identity_Color, Card_card_color_Color, and Deck_has_Card. The reason for these entity relationships is that of the many-to-many relationship between each entity that share this relation. I will do my best to explain the purpose behind each entity and each entity relationship between each of the entities. We will begin by starting off with the Card entity and expand outwards as it is the core of the relational model as mostly every entity has a relationship with it.

The Card entity holds the multiverse ID, card name, rarity, and set name flavor text, and artist of the card. The multiverse ID, card name and set name, a foreign key from MTGSet entity, are the primary keys as every combination of multiverse ID, card name, and set name uniquely identifies a cards rarity, flavor text and artist. A cards rarity, flavor text and artist depend on the card and the set in which it was released in, which is identified by the combination of multiverse ID and card name. Below, an example, Figure 5, of two card that have the same card name, but were released in different sets.

Figure 5 – Dispel card from Return to Ravnica set (left) and Worldwake set (right)[4][5].


Dispel
Instant
Counter target instant spell.
"Civilization will subsume us unless we overwhelm it instead."
—Speaker Zegana
Chase Stone
Dispel
Instant
Counter target instant spell.
The Jwari winds undo reality as easily as they scatter a pile of leaves.
Vance Kovacs

The Card_Information entity holds the rest of information of the card. The information included are the name, layout, mana cost, card text, card type, CMC, power, toughness, and loyalty of the card. All of this information stays constant, regardless of whether the set is re-released in a different set as what change from a card is the rarity, flavor text (as seen above in Figure 5 the bottom quotes), and the artist (also seen above at the bottom left, next to the paintbrush symbol). As one can see, the Dispel card text is "Counter target instant spell" even though both cards were released under different artist, set and flavor text, the card itself still cost one blue mana (top right of the card), till holds the same type (Instant), and card text.. This is the core reason we had to separate this information from the Card entity as well as to normalize the relation.

The Split_Flip_Card entity holds information that describes a relationship of a card and other cards that are related to it. This kind of relationship exist in Magic the Gathering if a card's layout is an aftermath, flip, meld, double-faced, or split. This mean that one side of a card has unique values that is associated with the name of the card and the other side of the same card has different values and a card name. So, an example, card A, when flipped, becomes card B. Both card A and card B are unique yet they are related to each other. So, when associated by name, it will also include the other cards that the card is related to. For example, there exist a split card named "Turn" and the other half of the card is "Burn". So, it is important to store this relationship if the user ever wants to know the other half of the card as this is important in identifying color identity and card names on that card. The combination of the multiverse ID (current card) and the multiverse ID (other card or cards) is the primary key in this entity.

The MTGSet entity holds the name of the set and the code of the set. Combining these two result in the primary key of the MTGSet entity. Every Set has a name and a code associated with it. An example of this is the Magic the Gathering set “Magic 2015”, it has the name of “Magic 2015” and its code is “M15”. Thus, making it an ideal way to identify a set.

The Set_Information entity holds the rest of the set information which includes the set code as the primary key, set type, released date, and block name. As with the Card_Information entity, we separated the set type, released date, and block name to allow BCNF normalization. Therefore, by knowing the code of the set, one can find what kind of set released was it, which are expansion, box, commander, conspiracy, core, duel deck, from the vault, masterpiece, masters, planechase, premium deck, promo, reprint, starter, un, and vanguard. Also, the block name and the released of the set will also be revealed through identifying of the set code. Our previous problem with having the Set_Information and Set entity was that the set name was used as the primary key, but each non-primary key was also dependent on the set code, violating 3NF. Therefore, the separation was necessary.

The Rulings entity shares a one-to-many relationship with Card as a card could have zero to many rulings and that ruling is dedicated to one card only. It contains the card name, being a foreign key from Card entity, text ruling, and date of the ruling. The primary keys are both the card name and the text ruling as that is what makes them unique. The Ruling entity is necessary as text rulings in Magic the Gathering is very common and differs from card to card. They are meant to help the player understand the interaction and rules to the card, if any.

The Format entity is a weak entity that shares a many-to-many relationship with Card entity. The only value, which happens to be the primary key also, is Format name. Format name determines the cards that can be played in them as we will see in the next relationship.

Due to the playability of the card being determined by the format, we have created a entity relationship between Card and Format called Legality entity, or Format_Legality_Card entity. The purpose to this entity is that a card may be play in zero to many formats depending on the legality. Therefore, we must create a relationship that show whether a card is legal, limited, or banned within a format. This entity contains format name, card name, and ban type. Both format name and card name are foreign keys from Format and Card entity, respectively.

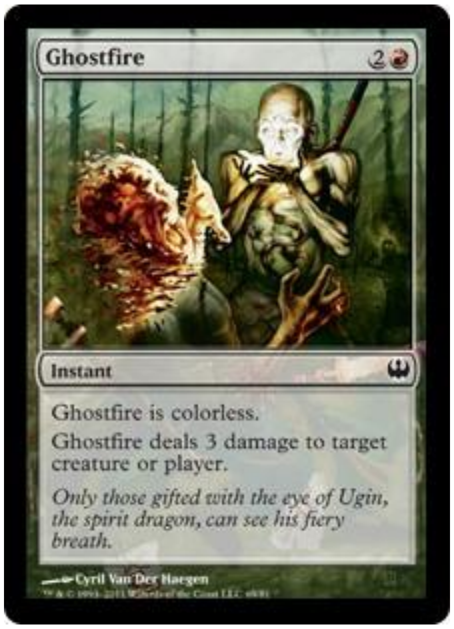
The Type entity also shares a many-to-many relationship with Card entity. Type entity has type name as a primary key and type. Type name is the type that describes the card, such as a goblin, wizard, instance, sorcery, and enchantment. The type is the kind of type the card is, which includes supertype, type, and subtype. These are important characteristics to a card for someone looking for specific archetypes.

A card can have several types to it, which is the reason why we have create a many-to-many relationship between Card and Type entity called Card_has_Type entity. In this relationship, our primary key is a combination of type name, from the Type entity, and card name, from the Card entity. For example, a card can have the card type “Legendary”, as a super type, “Creature”, as a type, and “Wizard”, as a subtype. Many cards shared these common types. Therefore, we felt as it would be a good idea to create a Type entity and a relationship between Type and Card.

The Color entities share a many-to-many relationship between Card entity. Color entity has color symbol and color name, with color symbol being the primary key of the entity. The main colors in Magic the Gathering are blue, black, white, green and red. The color symbols that correspond to the colors are “U” for blue, “B” for black, “W” for white, “G” for green and “R” for red.

A card has both color identity and color. These two differ from one another as a card’s color comes from either a special ruling text that declares it is or is not a specific color(s) and the color symbols, which are found on the top right of the card. One thing to note is that the specific ruling text overrides the color symbol. The card color identity comes from the color symbol on the top right and the color symbol(s) found within the card’s text. Color identity ignores special rulings found in the text. In Figure 6, found below, the card Ghostfire has null color or no color due to the ruling found in the text, but has the red color identity.

Figure 6- Ghostfire's color is null/colorless but its identity is red.[6]


Ghostfire
2
Instant
Ghostfire is colorless.
Ghostfire deals 3 damage to target creature or player.
Only those gifted with the eye of Ugin, the spirit dragon, can see his fiery breath.
Cyril Van Der Haegen

Therefore, we created two different entity relationships between Color and Card, yet they share the same concept. This concept is the identifying of the card based on color and color identity. In these entity relationships, the primary key consists of both color symbol, from the Color entity, and the card name, from the Card entity. The combination of these two helps identify a card’s color and color identity. Color identity is important in the format of Commander, also known as Elder Dragon Highlander (EDH). Just by knowing the card, we can identify it by its color or color identity. And vice-verse, we can identify colors by the card.

Now that we finish describing what a card is, we move onto discussing the entity relationship of the User-Deck creation by starting with User entity. In the User entity, it shares a one-to-many relationship between Deck entity. The user entity defined by the username, password, and role. The primary key of this entity consist of only the username as it is sufficient enough to uniquely identify a user of the system. Only one user can have a specific username. The only role the system supports at the moment is the “user” role. We have yet to implement an “admin” role that is responsible of creating “non-user decks” for others to see and manage the users and their decks. Users create decks, which will lead us to the Deck entity.

The Deck entity has a many-to-many relationship between Card entity and has a “is-a” relationship with UserDeck entity. Deck entity contains deck ID, which is the primary key of the entity, deck name, and format. Every deck is identified by a unique ID that auto increments for every entry. Each deck also has a name and format to identify the name of the deck and the format that the deck is intended to play in.

The UserDeck entity has an is-a relationship with Deck entity as the original intention of the application was to add public decks, which users could view. This entity includes username, from the User entity, deck ID, from the Deck entity, and the visibility of the deck. Primary key of UserDeck entity is the combination of username and deck ID. The purpose of this entity was to distinguish between user deck and public decks added by admin.

Lastly, we have the many-to-many entity relationship between Deck and Card entity. This entity relationship is called Deck_has_Card entity. The primary key for this entity is the combination of deck ID and multiverse ID. It also includes the mainboard quantity and sideboard quantity of the card, which is identified by the multiverse ID.

Relational Model

Relational Model
User (Username, Password, Role)
Deck (DeckID, DeckName, Format)
UserDeck(deckDeckID, userUsername, Visible)
Set(SetName, SetCode)
Card(MultiverseID, CardName, setSetName, FlavorText, Artist, Rarity)
Color(ColorSymbol, Color)
Type(TypeName, Type)
Format(FormatName)
Ruling(cardCardName, RulingText, RulingYear)
Card_Info(cardCardName, Layout, ManaCost, CardText, CardType, CMC, Power, Toughness, Loyalty)
Set_Information(setSetCode, SetType, ReleasedDate, BlockName)
Deck_has_Card(deckDeckID, cardMultiverseID, MainboardQty, SideboardQty)
Split_Flip_Card(cardCurrentCardMultiverseID, cardAssociatedMultiverseIDOnCard)
Format_Card(formatFormatName, cardCardName, BanType)
Card_Color(cardCardName, colorColorSymbol)
Card_ColorIdentity(cardCardName, colorColorSymbol)
Functional Dependencies
Username → Password, Role
DeckID → DeckName, Format
deckDeckID, userUsername → Visible
MultiverseID, CardName, setSetName → FlavorText, Artist, Rarity
ColorSymbol → Color
TypeName → Type
cardCardName, RulingText → RulingYear
cardCardName → Layout, ManaCost, CardText, CardType, CMC, Power, Toughness, Loyalty
setSetCode → SetType, ReleasedDate, BlockName
deckDeckID, cardMultiverseID → MainboardQty, SideboardQty
formatFormatName, cardCardName → BanType
All of our relations are in BNCF.

Design-Part 2: Use Case Realization

Use Case Name: Create a New Account

Steps:

1. The system asks the guest for a new password and unique username.
2. If the guest has provided valid input, a new account is added to the database.

SQL Procedures:

```
DELIMITER //
CREATE PROCEDURE spNewUser
(IN username VARCHAR(20), password VARCHAR(20), role VARCHAR(20))
BEGIN
        INSERT INTO user VALUES(username, password, role);
END //
CALL spNewUser(‘sampleusername’, ‘samplepassword’, ‘basic’);

DELIMITER //
CREATE PROCEDURE spGetDeckCount()
BEGIN
        SELECT u.Username, COUNT(*) FROM User u
        JOIN UserDeck ud ON u.Username = ud.UserName
        JOIN Deck d ON d.deckid = ud.deckid GROUP BY u.Username;
        END //
CALL spGetDeckCount();
```

Use Case Name: Search For a Card

Steps:

1. The user picks search criteria from name, color, mana cost, artist name, etc.
2. The user provides input for each criteria they selected.
3. Search returns matching cards from the DB in a sorted list.

SQL Procedures:

```
DELIMITER //
CREATE PROCEDURE spGetAllCards()
BEGIN
        SELECT c.cardname, ci.manacost, ci.power, ci.cardtype, ci.toughness, ci.cmc, s.setname,
        si.releaseddate, COUNT(*)
        FROM Card c
        JOIN Card_Information ci ON ci.cardname = c.cardname
        JOIN MTGSet s ON s.setname = c.setname
        JOIN Set_Information si ON si.setcode = s.setcode;
END //
CALL spGetAllCards();

DELIMITER //
CREATE PROCEDURE spParamSearch(IN _color VARCHAR(20), IN _toughness INT, IN _type
VARCHAR(20), IN _format VARCHAR(20) )
BEGIN
        SELECT c.cardName, COUNT(*) FROM Split_Flip_Card sfc
        JOIN Card c ON c.multiverseid = sfc.multiverseid_card
        JOIN Card_Information ci ON ci.cardName = c.cardName
        JOIN Card_Format cf ON cf.cardName = c.cardName
        JOIN Card_Type ct ON ct.cardName = c.cardName
        JOIN Type t ON t.typeName = ct.typeName
        JOIN Card_Color cc ON cc.cardName = c.cardName
        JOIN Color co ON co.colorSymbol = cc.colorSymbol
        WHERE (co.color = _color AND ci.toughness = _toughness AND t.type =_type
        AND cf.formatName = _format);
END //
CALL spParamSearch('blue',  4, 'exampletype', 'formatname');

DELIMITER //
CREATE PROCEDURE spGetRulings(IN _cardname VARCHAR 20)
BEGIN
        SELECT c.CardName, COUNT(*) FROM Ruling r
        JOIN Card c ON c.cardname = r.cardname
        WHERE c.cardname = _cardname;
END //
CALL spGetRulings('forest');
```

Use Case Name: Search for a Deck

Steps:

1. The user picks search criteria from deck name, name of the user who created the deck, and deck format name.
2. The user provides input for each criteria they selected.
3. Search returns matching decks from the DB in a sorted list.

SQL Procedures:

```
DELIMITER //
CREATE PROCEDURE spDeckSearch(IN _username VARCHAR(20), _decknameVARCHAR(20))
BEGIN
        SELECT d.deckname, u.username
        FROM deck d
        LEFT OUTER JOIN userdeck u on u.deckid = d.deckid
        WHERE (d.deckname = _deckname AND u.username = _username);
END //
CALL spDeckSearch(bob, bobsdeck);
```

Use Case Name: View a Deck

Steps:

1. The user requests to see a deck.
2. The system serves the deck page.

SQL Procedures:

```
DELIMITER //
```

```
CREATE PROCEDURE spGetDeck(IN _deckid INT)
BEGIN
        SELECT c.multiverseid, c.CardName, si.SetName, ci.ManaCost, ci.CMC,
        ci.Power, ci.Toughness, c.MultiverseID, dhc.MainboardQty, dhc.SideboardQty
        FROM Card c
        INNER JOIN Card_Information ci ON ci.CardName = c.CardName
        INNER JOIN Deck_has_Card dhc ON dhc.Card_ID = c.ID
        INNER JOIN MTGSet s ON s.SetName = c.SetName
        INNER JOIN Set_Information si ON si.setcode = s.setcode
        WHERE dhc.deckid = _deckid;
GROUP BY d.idDeck;
END //
CALL spGetDeck(5235);
```

Use Case Name: Create New Deck

Steps:

1. The system asks for a deck name (unique per user) and deck format.
2. The user is brought to the deck view for their new deck.

SQL Statements:

```
DELIMITER //
CREATE PROCEDURE spNewDeck
(IN _username VARCHAR(20), IN _decknameVARCHAR(20), _format VARCHAR(20))
BEGIN
        INSERT INTO Deck (DeckName, Format) VALUES (_deckname, _format);
        INSERT INTO UserDeck VALUES (_username, LAST_INSERT_ID(), true);
END //
CALL spNewDeck(‘bob’, ‘bobs deck’, ‘format name’);
```

Use Case Name: Add Cards to Deck

Steps:

1. From the deck view, registered user selects to modify their deck.
2. A dialog is presented for the registered user to quickly search for cards within the deck’s format and add them to the deck.

SQL Statements:

```
DELIMITER //
CREATE PROCEDURE spUpdateQuantity
(IN _iddeck INT, _idcard INT, _sideboardQty INT, _mainboardQty INT)
BEGIN
        UPDATE Deck_has_Card SET SideboardQty = _sideboardQty, SET MainboardQty =
        _mainboardQty)
        WHERE (Card_ID = _idcard AND idDeck = _iddeck);
END //
CALL spUpdateQuantity(15, 12309, 2, 3);
```

Test Plan and Records

MTGDB Test Plan

March 18, 2017

<u>Document Author(s):</u>

Derek Jones
Carlos A. Rios

Test ID	Description	Expected Results	Actual Results
1	Create Account	New account details added to DB	Pass
2	Search for Card	DB is queried with search params, results fetched and displayed on table	Pass (for querying entire card table only)
3	Search for Deck	DB is queried with deck name, results are displayed on the table	Fail (no decks have been included)
4	View Deck	DB is queried with specified deck ID, relevant deck details are fetched and displayed on the table	Fail (no decks have been included)
5	Create Deck	New deck details inserted into the deck and userdecks entities on the DB	Fails, MySQL FK error
6	Edit Deck	Cards are added or removed from associate with a particular deck; mainboard qty and sideboard qty are updated with new values to reflect this	Fail (no decks have been included)
7	Search for Users	DB user table is query, results are returned and placed on the table	(not yet implemented)
8	Admin Promotion	DB user table entry corresponding to some userID is updated to reflect the user’s new role	Pass
9	Import Card Data	If a new MTGJSON is published, the admin imports the new data into the DB, the old tables are dropped (except for users/decks)	(not yet implemented)

Version	1
Date	March 18, 2017
Change Description	Initial test plan

MTGDB Test Plan

April 29, 2017

<u>Document Author(s):</u>

Derek Jones
Carlos A. Rios

Test ID	Description	Expected Results	Actual Results
1	Create Account	New account details added to DB	Pass
2	Search for Card	DB is queried with search params, query results fetched and displayed on table	Pass
3	Search for Deck	DB is queried with specified deck name, Matching results are displayed on the table	Pass
4	View Deck	DB is queried with specified deck ID, relevant deck details are fetched and displayed on the table	Pass
5	Create Deck	New deck details inserted into the deck and userdecks entities on the DB	Pass
6	Edit Deck	Cards are added or removed from associate with a particular deck; mainboard qty and sideboard qty are updated with new values to reflect this	Pass

Version	2
Date	April 29, 2017
Change Description	Finalized version- removed admin cases (there is no more admin role), updated with final results of supported cases

Conclusion

After investing our efforts in building this Magic the Gathering Database application, we understand the importance of properly designing a database. When we started, neither of us had experience with databases. Using MySQL, we learned that implementing a database was relatively easy and straightforward. However, we ran into challenges when it came to database design. Before we really knew what we were doing we had made several design mistakes in the initial versions. Very early, we even planned on having comma separated lists stored in the database (a terrible idea). We also had problems with data redundancy. Because we were using real-world data (information for thousands of cards are stored in the database), we also got to experience the difficulties of working with a large amount of varied data. If our attribute for card names wasn't big enough, or we assumed some property that's usually an integer would always be an integer, there would always exist a card to break our assumptions and produce errors in the database and application.

Design flaws will surface when you try to apply your database. In our case, we'd end up having to go back to the drawing board several times to redesign the database. We would end up producing around fifteen revised versions of our ER model. In a real-world scenario, you may not be lucky enough to be able to always go back and restructure and rebuild the entire database. This was unlike other programming projects we've encountered. Usually, the simplicity of a program or the modular design of a program has enabled easy modification when problems were detected. Going into this project with the same attitude of quick coding, testing, and modifying until the application reaches some acceptable state led to early troubles. We realize now that a better attitude is to measure everything until you are sure the database design is correct before you start trying to create and apply your database.

References

[1]. MTGJson. https://mtgjson.com/

[2]. GitHub Google GSON. https://github.com/google/gson

[3]. Lead by Example card image. Wizards of the Coast Gatherer.
http://gatherer.wizards.com/Handlers/Image.ashx?multiverseid=407644&type=card

[4]. Dispel Return to Ravnica card image, Wizards of the Coast Gatherer.
http://gatherer.wizards.com/Handlers/Image.ashx?multiverseid=253566&type=card

[5]. Dispel Worldwake card image, Wizards of the Coast Gatherer.
http://gatherer.wizards.com/Handlers/Image.ashx?multiverseid=201562&type=card

[6]. Ghost Archenemy card image, Wizard of the Coast Gatherer.
http://gatherer.wizards.com/Handlers/Image.ashx?multiverseid=243486&type=card