Mental Magic – Magic: The Gathering Database

Designed by

James Sior

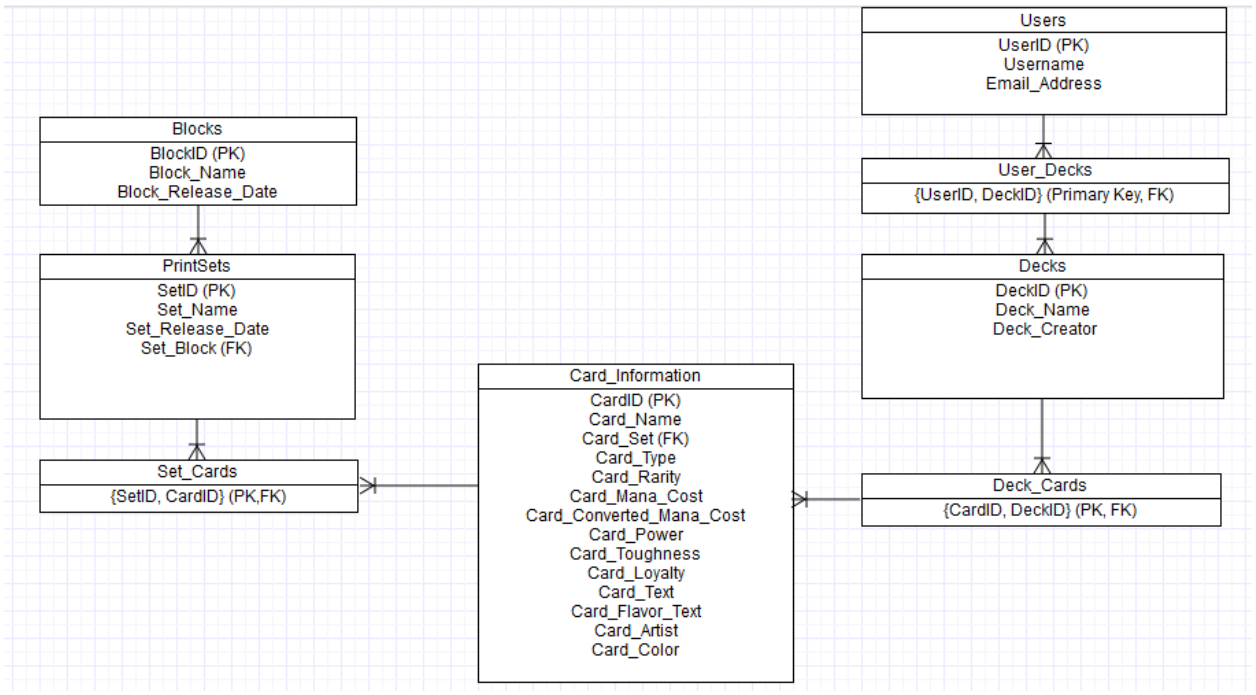
December 2, 2013

Table of Contents

Executive Summary

This document outlines our design and implementation of a database for Mental Magic. This database will serve as a central repository for all data related to Magic: The Gathering cards and decks. First, we will display our Entity-Relationship Diagram (ERD), which is used to provide a high-level description of the proposed database. We will cover the details of each table in the ERD in the following sections. These details include the purpose of each table along with design decisions made to meet Mental Magic's requirements. We will also provide standard SQL statements and sample data for each table. This will demonstrate how our design transfers to a concrete database. We will then provide examples of how our design will generate meaningful data through the use of fairly simple queries. Finally, we will explain the shortcomings of our design decisions and provide ideas for future database enhancements. As a technical note, this design is targeted for and tested on PostgreSQL 9.3.1.

ER-Diagram



This diagram provides a high-level view of the logical structure of this database. The process of implementing this design and its benefits are explained in the following sections.

Blocks Table

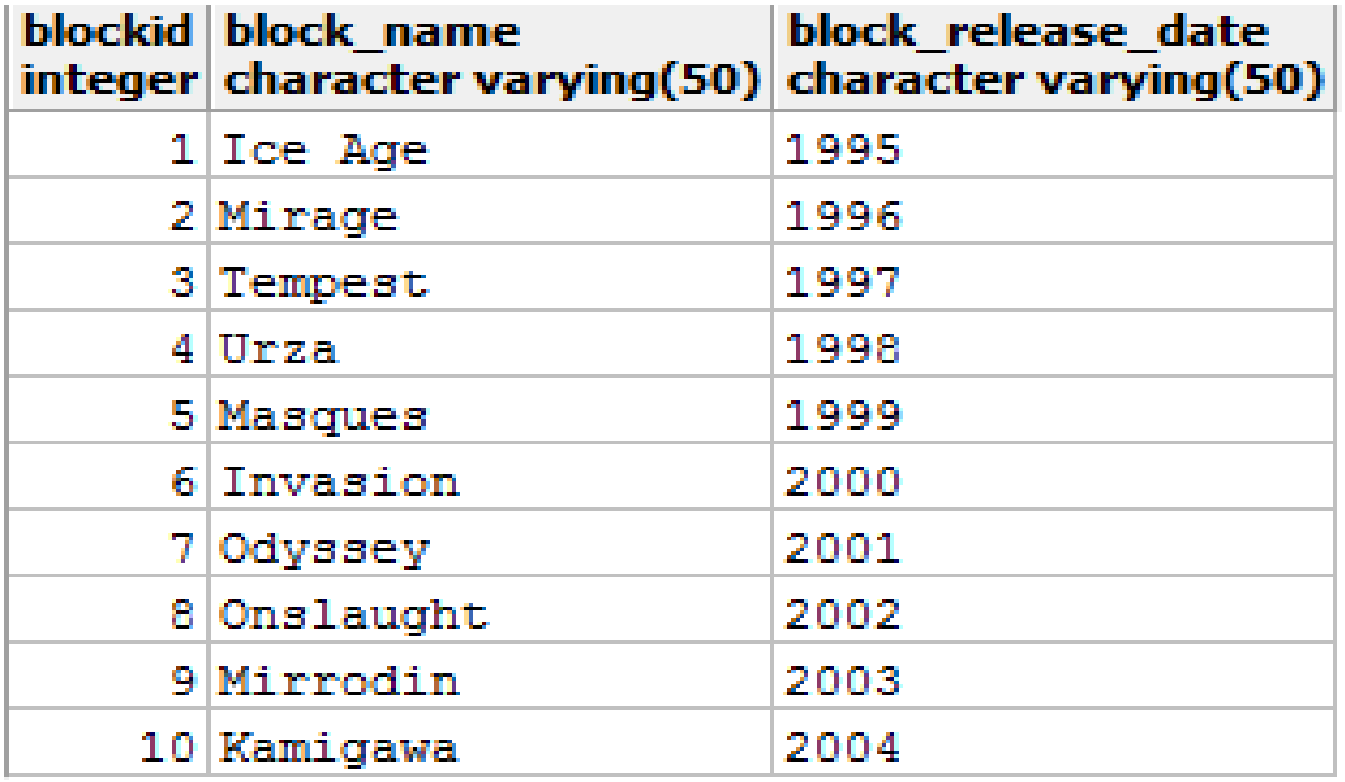
Magic: the Gathering cards are released in sets, and sets are in turn released by blocks. This table contains important information about blocks in Magic: the Gathering and allows users to query the database for information about individual blocks

```
CREATE TABLE Blocks (
   BlockID SERIAL,
   Block_Name varchar (50) NOT NULL,
   Block_Release_Date varchar (50) NOT NULL,
   UNIQUE (Block_Name),
   primary key (BlockID)
   );
```

Functional Dependencies:

BlockID → Block_Name, Block_Release_Date

Sample Data



blockid integer	block_name character varying(50)	block_release_date character varying(50)
1	Ice Age	1995
2	Mirage	1996
3	Tempest	1997
4	Urza	1998
5	Masques	1999
6	Invasion	2000
7	Odyssey	2001
8	Onslaught	2002
9	Mirrodin	2003
10	Kamigawa	2004

PrintSets Table

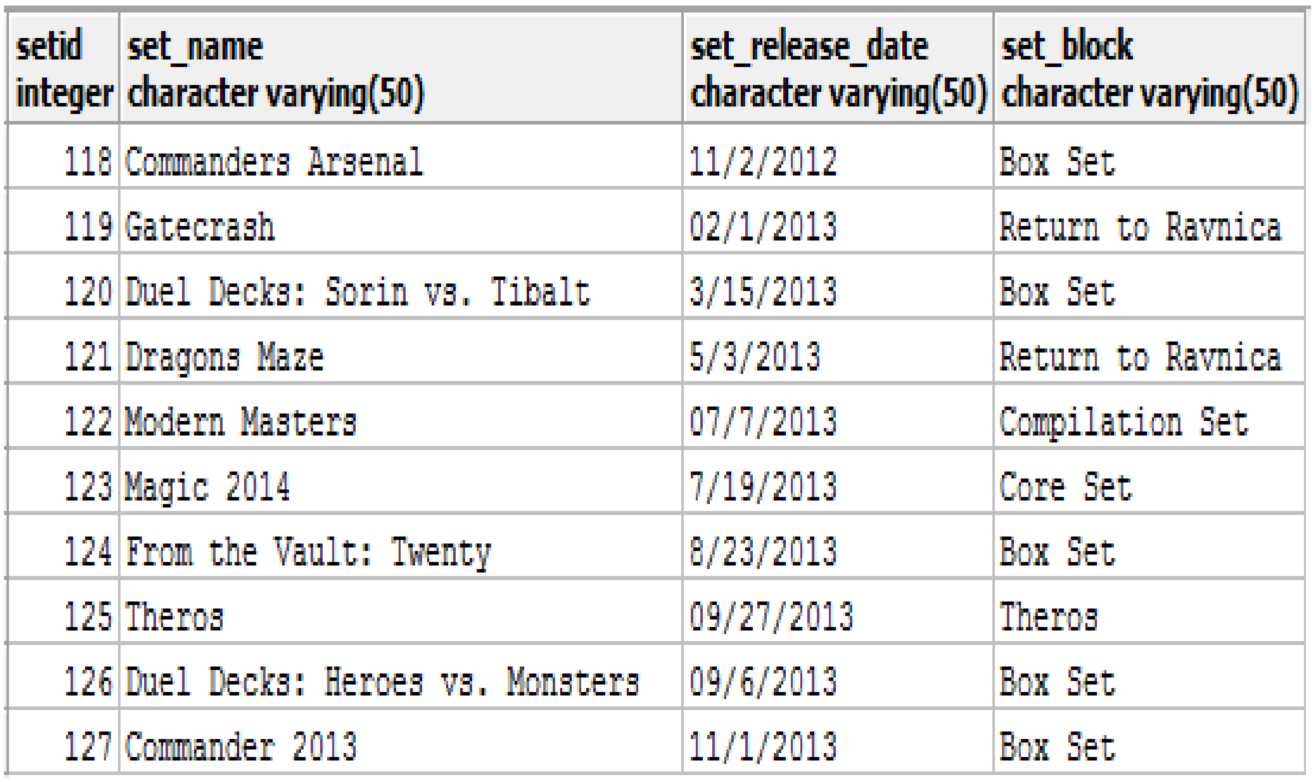
Magic: the Gathering cards are released in sets. Each set can contain numerous cards. This table contains information about sets and enables users to query the database for that information.

```
CREATE TABLE PrintSets (
   SetID SERIAL,
   Set_Name varchar (50) NOT NULL,
   Set_Release_Date varchar (50) NOT NULL,
   Set_Block varchar (50) NOT NULL references Blocks (Block_Name),
   UNIQUE (Set_Name),
   primary key (SetID)
   );
```

Functional Dependencies

SetID → Set_Name, Set_Release_Date, Set_Block

Sample Data



setid integer	set_name character varying(50)	set_release_date character varying(50)	set_block character varying(50)
118	Commanders Arsenal	11/2/2012	Box Set
119	Gatecrash	02/1/2013	Return to Ravnica
120	Duel Decks: Sorin vs. Tibalt	3/15/2013	Box Set
121	Dragons Maze	5/3/2013	Return to Ravnica
122	Modern Masters	07/7/2013	Compilation Set
123	Magic 2014	7/19/2013	Core Set
124	From the Vault: Twenty	8/23/2013	Box Set
125	Theros	09/27/2013	Theros
126	Duel Decks: Heroes vs. Monsters	09/6/2013	Box Set
127	Commander 2013	11/1/2013	Box Set

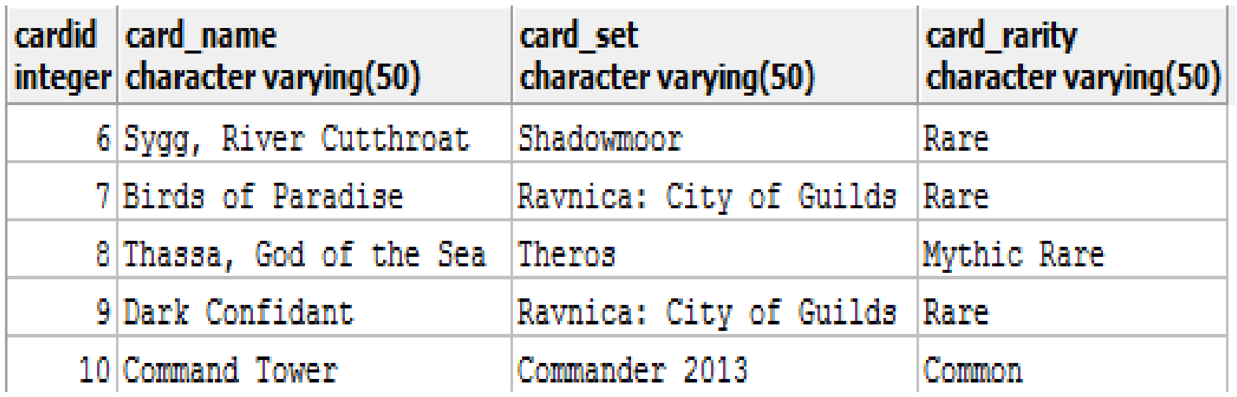
Cards Table

Cards are the backbone of Magic: the Gathering, and also the backbone of this database. Magic cards have many different properties. This table is quite large, but it contains useful data about cards and allows for interesting queries. Because this table contains many fields, not all will be shown on this page, but we will go into more detail about them later.

```
CREATE TABLE Cards (
    CardID SERIAL,
    Card_Name varchar (50) NOT NULL,
    Card_Set varchar (50) NOT NULL references PrintSets (Set_Name),
    Card_Type varchar (50) NOT NULL,
    Card_Rarity varchar (50) NOT NULL,
    Card_Mana_Cost varchar (50) NOT NULL,
    Card_Converted_Mana_Cost int NOT NULL,
    Card_Power int,
    Card_Toughness int,
    Card_Loyalty int,
    Card_Text varchar (500),
    Card_Flavor_Text varchar (500),
    Card_Artist varchar (50),
    Card_Color varchar (50) NOT NULL,
    primary key (CardID)
    );
```

Functional Dependencies
CardID → Card_Name, Card_Set, Card_Type, Card_Rarity, Card_Mana_Cost, Card_Converted_Mana_Cost, Card_Power, Card_Toughness, Card_Loyalty, Card_Text, Card_Flavor_Text, Card_Artist, Card_Color

Sample Data



cardid integer	card_name character varying(50)	card_set character varying(50)	card_rarity character varying(50)
6	Sygg, River Cutthroat	Shadowmoor	Rare
7	Birds of Paradise	Ravnica: City of Guilds	Rare
8	Thassa, God of the Sea	Theros	Mythic Rare
9	Dark Confidant	Ravnica: City of Guilds	Rare
10	Command Tower	Commander 2013	Common

Decks Table

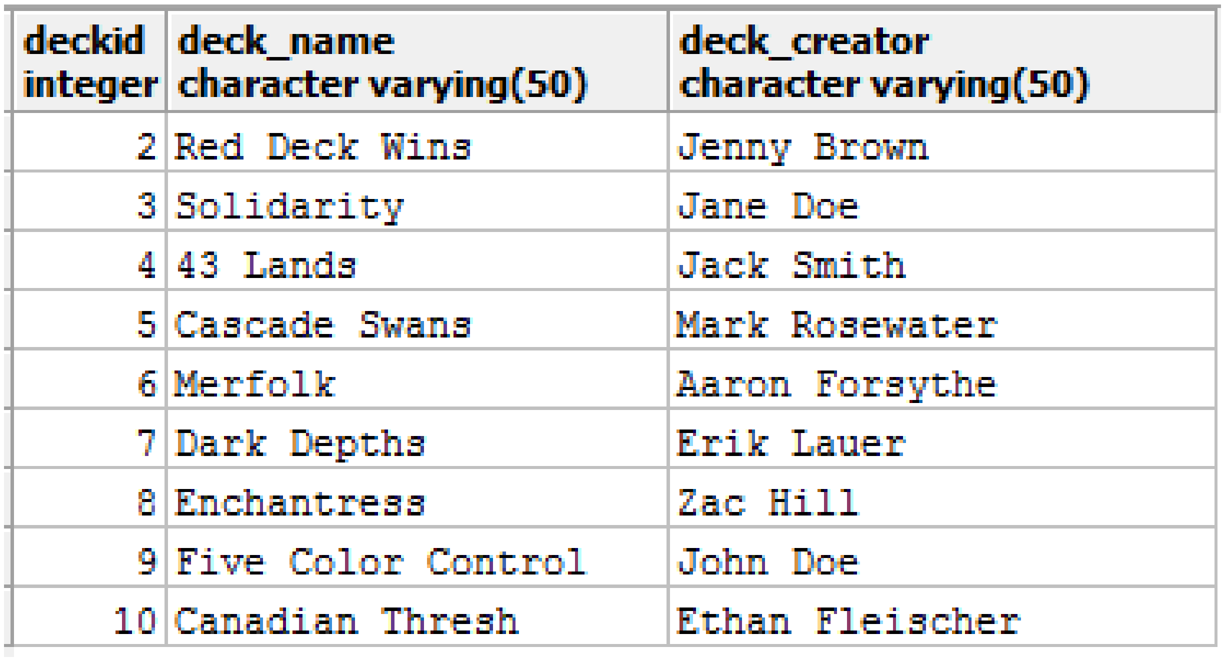
In this database design for Mental Magic, we wanted users to be able to both query the database for individual cards as well as decks. The decks table contains that information.

```
CREATE TABLE Decks (
    DeckID SERIAL,
    Deck_Name varchar (50) NOT NULL,
    Deck_Creator varchar (50) NOT NULL,
    primary key (DeckID)
    );
```

Functional Dependencies

DeckID → Deck_Name, Deck_Creator

Sample Data



deckid integer	deck_name character varying(50)	deck_creator character varying(50)
2	Red Deck Wins	Jenny Brown
3	Solidarity	Jane Doe
4	43 Lands	Jack Smith
5	Cascade Swans	Mark Rosewater
6	Merfolk	Aaron Forsythe
7	Dark Depths	Erik Lauer
8	Enchantress	Zac Hill
9	Five Color Control	John Doe
10	Canadian Thresh	Ethan Fleischer

Users Table

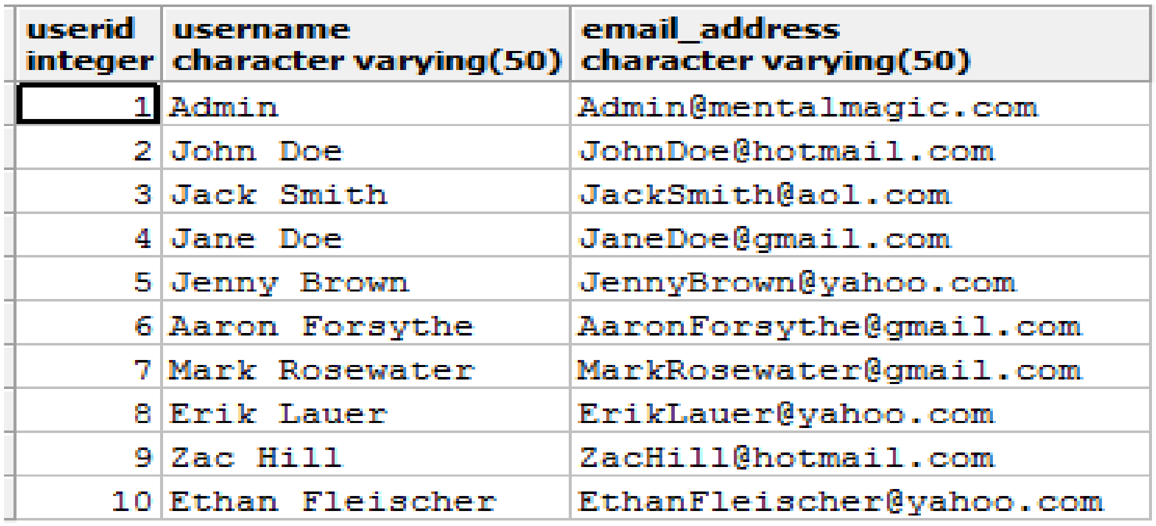
Mental Magic allows users to create usernames for themselves. They use these usernames when they create a deck. These usernames are stored in the Users table.

```
CREATE TABLE Users (
   UserID SERIAL,
   Username varchar (50) NOT NULL,
   Email_Address varchar (50) NOT NULL,
   UNIQUE (Username),
   primary key (UserID)
   );
```

Functional Dependencies

UserID → Username, Email_Address

Sample Data



userid integer	username character varying(50)	email_address character varying(50)
1	Admin	Admin@mentalmagic.com
2	John Doe	JohnDoe@hotmail.com
3	Jack Smith	JackSmith@aol.com
4	Jane Doe	JaneDoe@gmail.com
5	Jenny Brown	JennyBrown@yahoo.com
6	Aaron Forsythe	AaronForsythe@gmail.com
7	Mark Rosewater	MarkRosewater@gmail.com
8	Erik Lauer	ErikLauer@yahoo.com
9	Zac Hill	ZacHill@hotmail.com
10	Ethan Fleischer	EthanFleischer@yahoo.com

Set_Cards Table

This table is an associative entity that exists to break the many to many relationship between PrintSets and Cards – one set can contain many cards, but many cards can also belong to many sets. The Set_Cards Table fixes this problem by creating a one to many relationship between PrintSets and Set_Cards and a many to one relationship between Set_Cards and Cards.

```
CREATE TABLE Set_Cards (
   SetID int NOT NULL references PrintSets (SetID),
   CardID int NOT NULL references Cards (CardID),
   primary key (SetID, CardID)
   );
```

Deck_Cards Table

This table is an associative entity that exists to break the many to many relationship between Decks and Cards – one deck can contain many cards, but many cards can also belong to many decks. The Deck_Cards Table fixes this problem by creating a one to many relationship between Decks and Deck_Cards and a many to one relationship between Deck_Cards and Cards.

```
CREATE TABLE Deck_Cards (

        CardID int NOT NULL references Cards (CardID),

        DeckID int NOT NULL references Decks (DeckID),

        primary key (CardID, DeckID)

    );
```

User_Decks

This table is an associative entity that exists to break the many to many relationship between Users and Decks – one user can create many decks, but many decks can also belong to many users. The User_Decks Table fixes this problem by creating a one to many relationship between Users and User_Decks and a many to one relationship between User_Decks and Decks.

```
CREATE TABLE User_Decks (
        UserID int NOT NULL references Users (UserID),
        DeckID int NOT NULL references Decks (DeckID),
        primary key (UserID, DeckID)
         );
```

Views

This view can be used to find and display cards from specific sets and specific blocks. It is useful because it displays only the card names and the sets and blocks those cards belong in, so it is simple to search for any set or block you want.

```
CREATE VIEW Cards_Sets_Blocks AS
SELECT b.Block_Name,
   b.Block_Release_Date,
   s.Set_Name,
   s.Set_Block,
   s.Set_Release_Date,
   c.Card_Name,
   c.Card_Set
FROM Blocks b
INNER JOIN PrintSets s
ON b.Block_Name = s.Set_Block
INNER JOIN Cards C
ON C.Card_Set = s.Set_Block;
```

Example

```
SELECT *
FROM Cards_Sets_Blocks
WHERE Card_Set = 'Shadowmoor'
ORDER BY Card_Set DESC
LIMIT 5;
```

Security

Administrator Role

This database will only have two roles – a user role and an administrator role. The administrator role will have access to update, insert, select, and delete. This role is designed for a veteran database administrator who will manage the entire database.

```
REVOKE ALL PRIVILIGES ON Blocks FROM administrator;
REVOKE ALL PRIVILIGES ON PrintSets FROM administrator;
REVOKE ALL PRIVILIGES ON Cards FROM administrator;
REVOKE ALL PRIVILIGES ON Decks FROM administrator;
REVOKE ALL PRIVILIGES ON Users FROM administrator;
REVOKE ALL PRIVILIGES ON Set_Cards FROM administrator;
REVOKE ALL PRIVILIGES ON Deck_Cards FROM administrator;
REVOKE ALL PRIVILIGES ON User_Decks FROM administrator;

GRANT UPDATE, INSERT, SELECT, DELETE ON Blocks FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON PrintSets FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON Cards FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON Decks FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON Users FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON Set_Cards FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON Deck_Cards FROM administrator;
GRANT UPDATE, INSERT, SELECT, DELETE ON User_Decks FROM administrator;
```

Security, Continued

Users Role

The Users role will have read access on most tables. However, because Users should be able to create Decks, the User role will have write access to the Decks table. Users will also have write access to the Users table (for example, to change their email address).

```
REVOKE ALL PRIVILIGES ON Blocks FROM users;
REVOKE ALL PRIVILIGES ON PrintSets FROM users;
REVOKE ALL PRIVILIGES ON Cards FROM users;
REVOKE ALL PRIVILIGES ON Decks FROM users;
REVOKE ALL PRIVILIGES ON Users FROM users;
REVOKE ALL PRIVILIGES ON Set_Cards FROM users;
REVOKE ALL PRIVILIGES ON Deck_Cards FROM users;
REVOKE ALL PRIVILIGES ON User_Decks FROM users;

GRANT SELECT ON Blocks FROM users;
GRANT SELECT ON PrintSets FROM users;
GRANT SELECT ON Cards FROM users;
GRANT UPDATE, INSERT, SELECT ON Decks FROM users;
GRANT UPDATE, SELECT, ON Users FROM users;
GRANT SELECT ON Set_Cards FROM users;
GRANT SELECT ON Deck_Cards FROM users;
GRANT SELECT ON User_Decks FROM users;
```

Implementation Notes

- Although previously mentioned, it is important to note that this database is designed for and tested on PostgreSQL 9.3.1. If this database is implemented on any other version of postgreSQL, changes may have to be made. For more information on postgreSQL versions, see www.postgresql.org.
- As Magic: the Gathering is a rapidly evolving and changing game, new cards will have to be added to the database on a regular basis
- Although mentioned previously, the administrator role is extremely powerful and should only be granted to an individual who fully understand the intricacies of the database.

Known Problems

- In our implementation of this database, the Cards table is extremely large, containing 14 fields when most other tables have 2 or 3. We considered how we could break the Cards table down into multiple smaller tables, which would help normalize the database, but were unable to devise a way to do so in postgreSQL. Because the Cards table has so many rows compared to other tables, it is much more prone to INSERT, UPDATE, and DELETE anomalies. Although this is a serious flaw in implementation, we were unable to come up with an alternative solution.
- Because some of the fields in the Cards table are very long, such as Card_Text, the table will not always display the entirety of those rows. This is extremely annoying as a user. However, although we investigated ways to solve this issue, we were unable to find a solution. For now, the only way a user can see the full field is to extend the width of the columns.
- Although users can create Decks, and Decks contain cards, there is currently no way to directly relate users to cards. For example, if Sally makes the deck "Project X" with 60 cards in it, I can see Sally has made that deck, but I cannot see the cards in the deck. We were unsure how to implement this in postgreSQL without adding many tables which would increase the complexity of the database.

Future Enhancements

- Triggers should be added to ensure data consistency. This current implementation lacks triggers, so it does not contain full functionality.
- Additionally, stored procedures should be implemented to perform certain calculations. This would be especially helpful when Users are viewing the cards in the decks of other users – however, as mentioned above, that functionality does not exist in this implementation, and therefore adding stored triggers for them is impossible.
- More views should be added to allow users to look at data in various ways.
- More roles should potentially be added as the database grows or, at the very least, the addition of more administrators should be considered.
- If this design is to be implemented in a future version of postgreSQL, certain elements may need to be changed.