**Magic the Gathering Database Report**

By Robert Anderson, Gabriel Davis, Kyle Sarre

**Motivations**

Magic the gathering, or MtG (which it will be called from this point on), is a popular trading card game owned by Wizards of the Coast LLC. An estimated 20 million players play MtG around the world. MtG has been printed in 11 different languages and is played competitively, with prize pools reaching around $250,000. Wizards of the Coast sponsors a considerable number of these tournaments.

MtG by design has incredible diversity in the types of decks that may be constructed. Our group wants to analyze decks used in the competitive scene of MtG to identify the popularity of select decks and cards by using a relational database. We believe that this information could be useful to MtG players for constructing new and exciting decks.

**Application Domain**

We identified four primary entities of interest for our database: cards, competitions, card representation (unique instances of decks that contained a specified card), and usage (an instance of a player using a given card in a given deck at a given competition). Many-to-many relationships exist between numerous entities. For instance, a given card could be played in many competitions, and a competition can have many cards played. Additionally, a card can exist in many decks, and many decks can contain a similar card. The usage table is an associative table that helps us resolve the many-to-many relationships between the entities in the system.

The following list elaborates on the attributes of our entities:

* Cards(card name, type, mana cost, white, blue, black, red, green, card description)
* Competitions(competition name, year)
* Card Representation(card name, deck name, copies of card)
* Usage(card name, deck name, competition name, year)

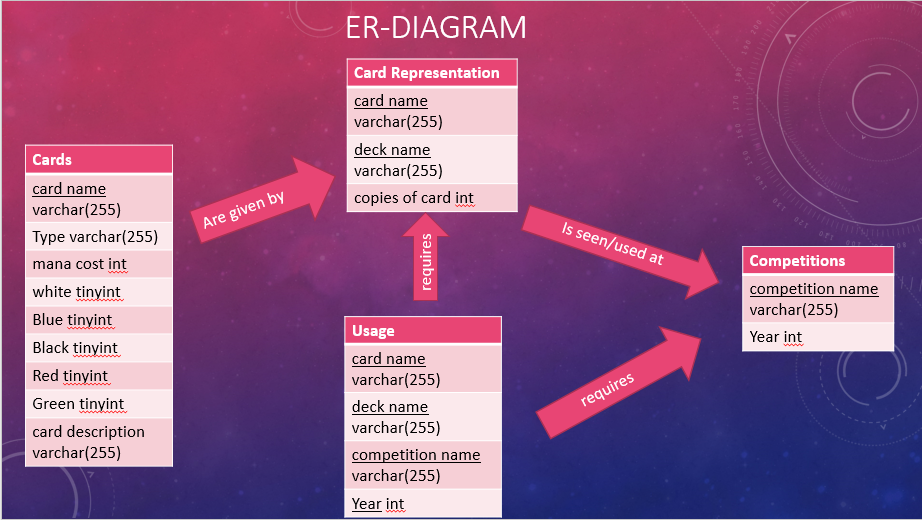
The following list elaborates on the assumptions made about the attributes in the system:

* Cards may have multiple types and have a subtype
  + example: | Steel Overseer (card name) | Artifact creature (multiple type) – construct (subtype) |
* For card representation, deck names incorporate the deck user’s name and the deck name itself

The following list elaborates on the constraints made about the attributes in the system:

* For cards, mana cost must be a non-negative number or null
* A usage must have a corresponding card representation
* Copies of card should be positive

It should be noted that Usage and Card Representation entities are weak entities. For a more complete description of the application domain, see the E-R diagram below.



**Database Design**

Our database schema:

* Cards(card name, type, mana cost, white, blue, black, red, green, card description)
* Competitions(competition name, year)
* Card Representation(card name, deck name, copies of card)
* Usage(card name, deck name, competition name, year)

We aimed to keep the relationships in 4BNF to reduce redundancy in the database. Cards is in 4BNF because the only dependency is between the primary attribute card name and the rest of the non-primary attributes. Competitions is in 4BNF because all its attributes are primary attributes (the same can be said of the Usage relation). Card Representation is in 4BNF because its only non-primary attribute “copies of card” has no dependencies on any other non-primary attribute. Hence the database is in 4BNF.

**Gathering the Data**

We utilized a Java API found at http://magicthegathering.io to gather card data for our card table. Additionally, we pulled data from <http://magic.wizards.com/en/articles/winning-decks> to populate the remainder of our tables. We wrote a parser to extract data from these files and place them into a formatted file that could then be loaded into our database using the “load data local infile” command.

**Database in Action**

Upon opening the bash shell, ensuring the project build is in the home folder, and giving scripts in the bin folder execution priveledges, we can initialize the database using “inject\_query -u$yourusername -p$yourpassword $yourdatabase db\_init.txt”. The inject\_query script will log a user into the mysql database and run the specified query. This instruction creates tables in our database to hold our records. We then navigate to the db\_queries folder and, using the inject\_query script, we can run any of the queries in the folder (examples of queries and their results can be viewed in the presentation accompanying this report).