Nathan Greenlaw

CS 340

6/7/2017

Magic the Gathering Database

For this project, I used Magic the Gathering (MtG) to create a database. MtG is a trading card game with cards based on fantasy and real-world themes such as vampires or ancient Greece. These cards are released in sets and a player creates a deck of cards to play against other players. The game has a long history and therefore I focused on the physical materials of the game from the most recent sets as that was the best candidate for a database that was feasible. The physical components have many relationships and this allowed for an interesting database.

The entities I have in my database are:

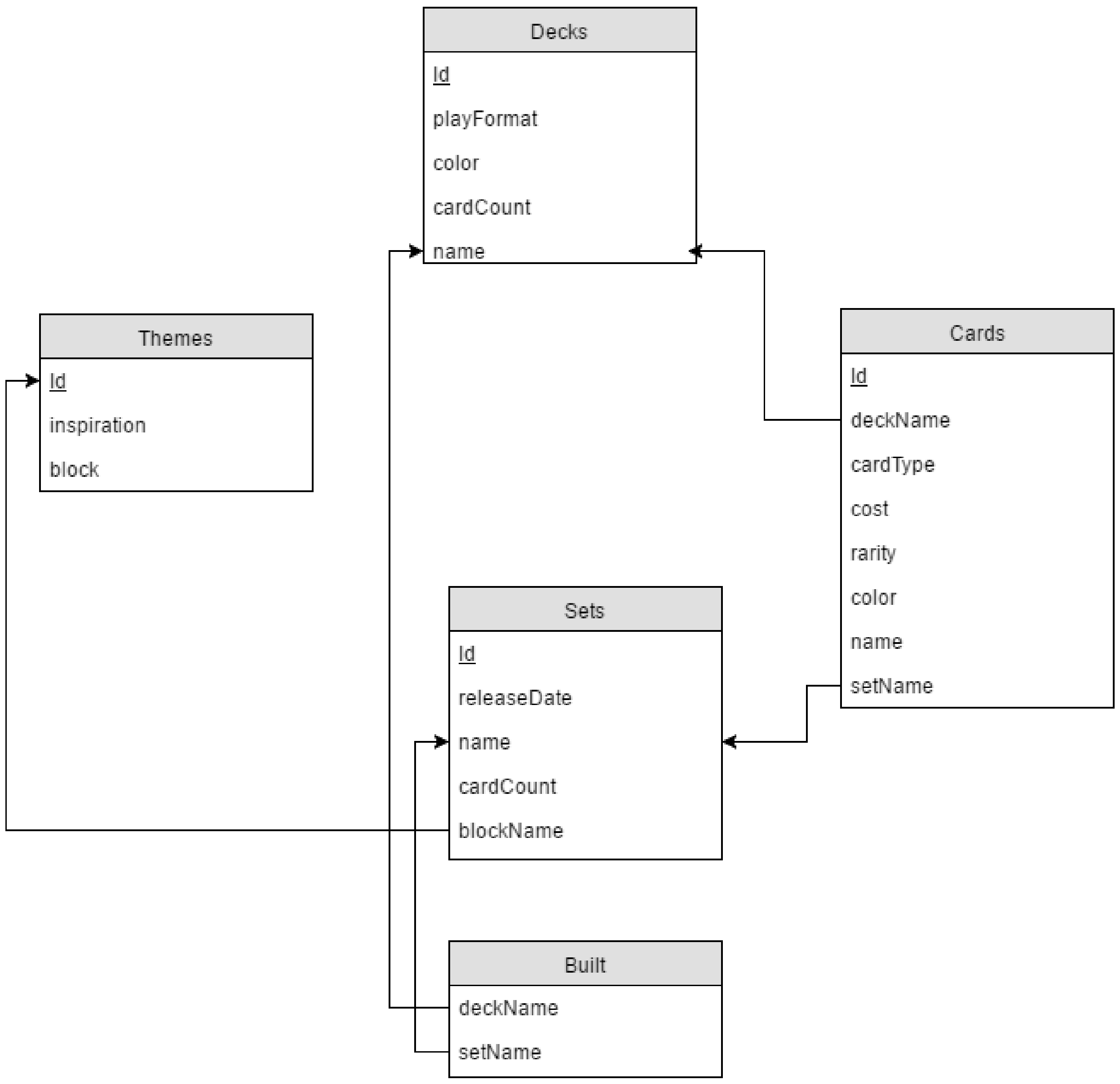
- Themes- These will have blocks (group of sets) and inspiration (real world basis for the block)
- Sets- These are the collections of cards that release. They will have a release date, name, card count and block they are from
- Decks- The collection of cards a player creates to play the game. They will have a name, card count, play format (there are different formats of playing the game) and color (decks have 1 to 5 colors from the set of green, blue, white, black and red)
- Cards- The main components of the game. They will have a name, type (cards can have many types), cost (integer amount of how much "mana" is needed to play the card), rarity (the rarity of the card in the set it comes from), color(same as decks), set name and deck name

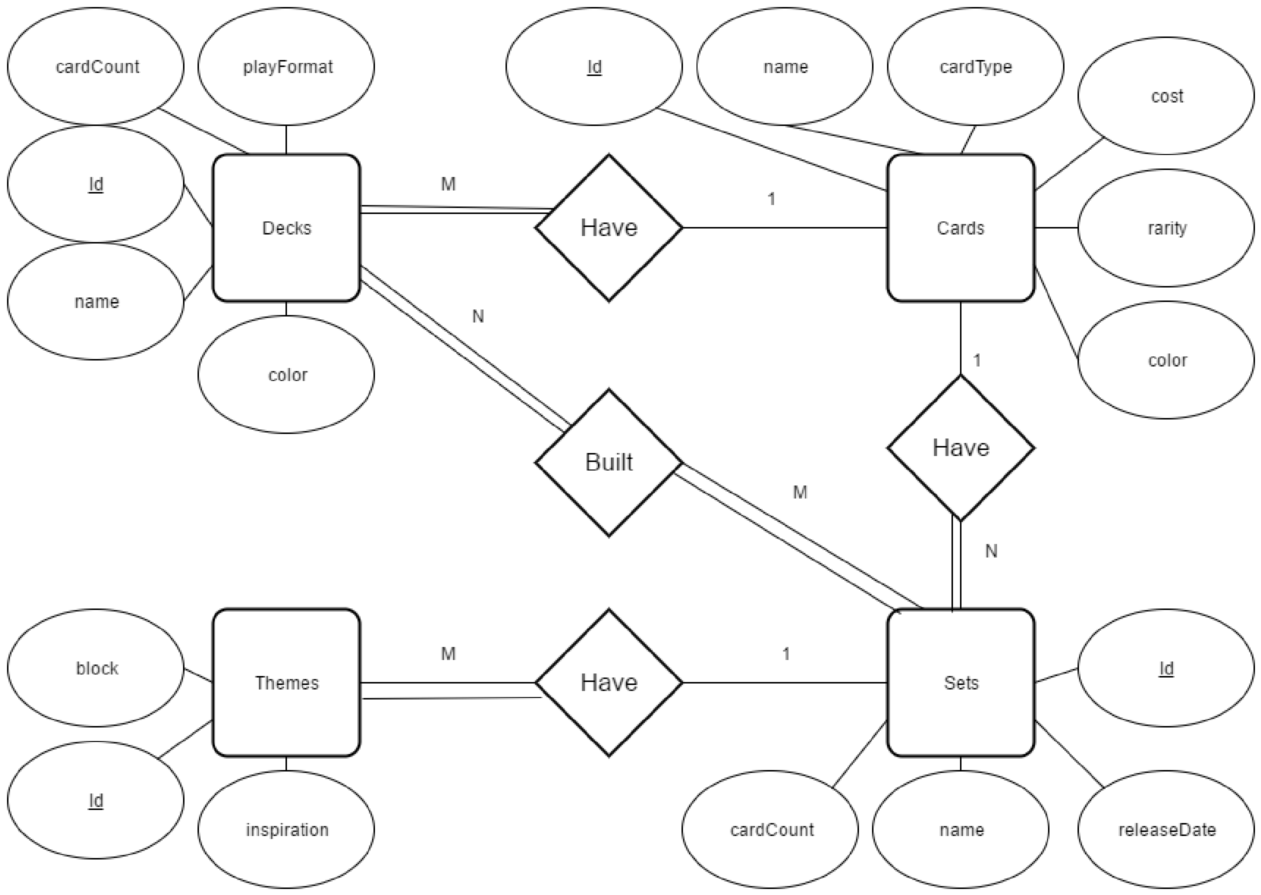
The relationships I have in my database are:

- Cards have sets- A card can only be from one set but a set can have many cards making this a one to many relationship.
- Cards are in decks- A card can only be in one deck but a deck can have many different cards making this a one to many relationship.
- Sets have decks built from them- Many decks can be built from many sets and many sets can have many decks making this a many to many relationship. (Represented by the built joint table)
- Sets have themes- A set can only have one theme but a theme can have multiple sets making this a one to many relationship.

ER and Schema

Here are the diagrams that I used for this database:


Decks
Id
playFormat
color
cardCount
name
Themes
Id
inspiration
block
Cards
Id
deckName
cardType
cost
rarity
color
name
setName
Sets
Id
releaseDate
name
cardCount
blockName
Built
deckName
setName



Create Tables Queries

Here are the queries that I used to create the tables:

```
/*Drop tables if they exist*/

DROP TABLE IF EXISTS `built`;
DROP TABLE IF EXISTS `cards`;
DROP TABLE IF EXISTS `sets`;
DROP TABLE IF EXISTS `decks`;
DROP TABLE IF EXISTS `themes`;


/*Create the tables*/

/*Deck table*/

CREATE TABLE decks(

        id int NOT NULL AUTO_INCREMENT,
	name varchar(255) NOT NULL,
	color varchar(255),
	cardCount int,
	playFormat varchar(255),
	PRIMARY KEY (id),
	UNIQUE KEY (name)
)ENGINE=Innodb;
/*Theme table*/
CREATE TABLE themes(
	id int NOT NULL AUTO_INCREMENT,
	block varchar(255) NOT NULL,
	inspiration varchar(255),
	PRIMARY KEY (id),
	UNIQUE KEY (block)
)ENGINE=Innodb;
/*Sets Table*/
CREATE TABLE sets(
	id int NOT NULL AUTO_INCREMENT,
	name varchar(255) NOT NULL,
	cardCount int,
	block varchar(255),
	releaseDate date,
	PRIMARY KEY (id),
        UNIQUE KEY (name),

        FOREIGN KEY (block)
                REFERENCES themes(block)
                ON DELETE CASCADE

)ENGINE=Innodb;

/*Cards table*/

        CREATE TABLE cards(

        id int NOT NULL AUTO_INCREMENT,

        name varchar(255) NOT NULL,

        cardType varchar(255),

        cost int,

        color varchar(255),

        rarity varchar(255),

        setName varchar(255),

        deckName varchar(255),

        PRIMARY KEY (id),

        UNIQUE KEY (name),

        FOREIGN KEY (setName)
                REFERENCES sets(name)
                ON DELETE CASCADE,

        FOREIGN KEY (deckName)
                REFERENCES decks(name)
                ON DELETE CASCADE

)ENGINE=Innodb;

/*Built table Set to Deck Many to Many relationship*/

CREATE TABLE built(
	setName varchar(255),

	deckName varchar(255),

	UNIQUE KEY (setName,deckName),

	FOREIGN KEY (setName)
		REFERENCES sets(name)
		ON DELETE CASCADE,

	FOREIGN KEY (deckName)
		REFERENCES decks(name)
		ON DELETE CASCADE

)ENGINE=Innodb;
```

<u>Adding, Deleting, Selecting and Filtering Queries</u>

Here are the queries I used to filter the cards and establish the many to many relationship of sets to decks:

```
-- decks

	-- adding to decks
	INSERT INTO decks (name, color, playFormat, cardCount)
		VALUES
		([name_input], [color_input], [playFormat_input], [cardCount_input]);

-- themes

	-- adding to themes
	INSERT INTO themes( block, inspiration )
	VALUES (
	[block_input], [inspiration_input]);

-- sets

	-- adding to sets
	INSERT INTO sets( name, cardCount, block, releaseDate )
	VALUES (
	[name_input], [cardCount_input], (

		SELECT block
		FROM themes
		WHERE block =  [block_input]
		),  [releaseDate_input]
	);
```

```
-- cards

        --adding to cards
        INSERT INTO cards( name, cardType, cost, color, rarity, setName, deckName )
        VALUES (
        [name_input], [cardType_input], [cost_input], [color_input], [rarity_input], (

                SELECT name
                FROM sets
                WHERE name = [setName_input]
                ), (

                SELECT name
                FROM decks
                WHERE name = [deckName_input]
                )
        );

        --filtering by type
                --filter options
                SELECT distinct cardType FROM cards

                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE cardType=[cardType_input]

        --filtering by cost
                --filter options
                SELECT distinct cost FROM cards

                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE cost=[cost_input]

        --filtering by color
                --filter options
                SELECT distinct color FROM cards

                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE color=[color_input]

        --filtering by rarity
                --filter options
                SELECT distinct rarity FROM cards
                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE rarity=[rarity_input]

        --filtering by setName
                --filter options
                SELECT distinct setName FROM cards

                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE setName=[setName_input]

        --filtering by deckName
                --filter options
                SELECT distinct deckName FROM cards

                --displaying the results
                SELECT name, cardType, cost, color, rarity, setName, deckName FROM `cards`
                WHERE deckName=[deckName_input]


-- built table

        -- adding to built
        INSERT INTO built( setName,deckName )
        VALUES (
        [setName_input],[deckName_input]);

        -- select all decks for a set
        SELECT b.setName, name as deck
        FROM built b
        INNER JOIN decks d ON d.name = b.deckName
        WHERE b.setName =  "Shadows over innistrad";

        -- select all sets for a given deck
        SELECT b.deckName, name as set_name
        FROM built b
        INNER JOIN sets s ON s.name = b.setName
        WHERE b.deckName =  "clues";

        -- delete rows from built by setName
        DELETE FROM built WHERE setName=[setName_input]

        -- delete rows from built by deckName
        DELETE FROM built WHERE deckName=[deckName_input]
```

Website

The website is not very complex in terms of functions or style. The tables are grouped with their respective adding or filtering options with the built table being the many to many relationship that has the options to add, filter or delete from the table. Once you perform a function you can click return to go back to the page.