mtgDB

Magic **T**he **G**athering Database Manager

—

Antonio Pelusi
Matricola - 182267

Panoramica

mtgDB è un Database Manager per il gioco di carte Magic The Gathering.

Consente all'utente di cercare, aggiungere, rimuovere, classificare e filtrare le carte a seconda di determinati parametri.

Nel dataset utilizzato sono presenti tutte le carte giocabili nel formato standard.

Istruzioni

Installazione dipendenze

- **MongoDB**: https://www.mongodb.com/try/download/community-edition
- **Python**: https://www.python.org/downloads/
- **PyMongo**: `pip install pymongo`
- **Pandas**: `pip install pandas`

Avvio

- Avviare MongoDB
- Muoversi nella directory `/mtgdb`
- Eseguire il programma: `python3 mtgdb.py`

Per l'utilizzo di **mtgDB**, si rimanda all'ultima sezione della documentazione (**Test guidato**).

Specifiche

Il programma, scritto in Python, utilizza PyMongo per stabilire una connessione con MongoDB.

Le fasi di inizializzazione del database e il caricamento del file JSON verranno gestite direttamente dal programma, quindi non è richiesta alcuna configurazione manuale di MongoDB.

Il programma viene eseguito su linea di comando. All'avvio (e dopo ogni operazione) verrà mostrato il menù delle funzionalità. L'utente, inserendo il numero relativo al comando scelto, potrà eseguire la funzione.

Il programma, per evitare eventuali errori, controllerà lo stato del database prima dell'esecuzione di una funzione.

Dataset

Il dataset utilizzato (`StandardCards.json`) è stato scaricato dal sito https://mtgjson.com/, e raccoglie tutti i dati delle carte legalmente giocabili nel formato standard di Magic The Gathering.

Le modifiche apportate al dataset verranno spiegate all'inizio della sezione funzioni, poiché effettuate direttamente all'interno delle query MongoDB.

Comandi

L'utente può interagire con il database tramite l'uso delle seguenti funzioni:

Comando	Descrizione
0	Esci da mtgDB
1	Carica il database
2	Conta le carte presenti nel database
3	Aggiungi una nuova carta
4	Rimuovi una carta esistente
5	Cerca una carta tramite il nome
6	Filtra le carte tramite il costo massimo di mana e il colore
7	Recupera le 5 carte più forti
8	Cerca una carta tramite parole presenti nella descrizione
9	Controlla se è possibile giocare una carta in un determinato formato
10	Effettua il drop del database

Processo di sviluppo

Il codice è strutturato in tre sezioni principali, **inizializzazione**, **funzioni** e **start**.

Nella sezione **inizializzazione** verrà stabilita la connessione con MongoDB, controllando se sia già presente il database di mtgDB.

Nella sezione **funzioni** sono presenti tutte le funzioni utilizzate da mtgDB per la gestione del dataset.

Nella sezione **start** partirà l'esecuzione del programma, in cui l'utente potrà chiamare la funzione desiderata.

Inizializzazione

In questa sezione viene inizializzata la connessione con MongoDB:

```
client = MongoClient("localhost", 27017)
```

Una volta creata la connessione, viene inizializzato il collegamento al database:

```
db = client["mtgdb"]
```

Stabilita la connessione con il database, si procede con l'inizializzazione della collection:

```
collection = db["cards"]
```

Questa collection verrà riempita attraverso l'apposita funzione (**0**) che si occuperà di caricare il file JSON all'interno di MongoDB.

Verrà quindi controllato se il database inizializzato era già presente, salvando il risultato nella variabile booleana `is_loaded`:

```
is_loaded = False
dbnames = client.list_database_names()
if 'mtgdb' in dbnames:
      is_loaded = True
```

Così facendo sarà possibile bloccare l'esecuzione delle funzioni nel caso il database non fosse stato ancora inizializzato, evitando di ricorrere in eventuali errori.

Funzioni

Prima dell'esecuzione di ogni funzione che contiene una interazione con MongoDB, viene controllato se esiste già il database, grazie all'uso della variabile `is_loaded` creata nella fase di inizializzazione.

Il database presenta una struttura che va modificata per renderla adatta al caso d'uso scelto. In particolare il set di carte è conservato in un oggetto di nome cards, ma sarebbe comodo disporre di tutte le carte singolarmente. Il comando per "estrarre" degli elementi da un array è `$unwind`, ma è applicabile soltanto ad un array.
Quindi viene in aiuto `$objectToArray`, il quale convertirà l'oggetto cards in una lista di oggetti, ognuno contenente due valori:

- ➔ **k**: contenente il nome della carta;
- ➔ **v**: contenente i dettagli della carta, compreso il nome.

Sarebbe quindi più corretto scartare il parametro k e tenere i dati contenuti in v direttamente nell'oggetto cards.
Per fare questo ho utilizzato `$map`, il quale prende due parametri:

- ➔ **input**: nel quale passo la lista cards, creata utilizzando `$objectToArray`;
- ➔ **in**: al quale passo `$$this.v`, che contiene il riferimento all'oggetto dell'array che sta venendo processato dalla funzione `$map`.

Ottenuta la lista di carte chiamata cards, è possibile finalmente effettuare un `$unwind` per estrarre le carte dalla lista cards.
Tutte queste operazioni vengono inserite in una `$project`.
Per avere una visione più pulita a livello di leggibilità, ho eseguito una `$project`, la quale ha il compito di nascondere il parametro "`_ID`", e mostrare solo i valori caratterizzanti di una carta, quali nome, costo della carta espresso in mana (l'unità di misura del costo delle carte in Magic The Gathering), i punti forza (power), i colori della carta (rosso, verde, blu, bianco, rosso, o qualunque combinazione di essi), tipo e descrizione (text).
Ho escluso dalla stampa finale valori non testuali, come il link all'artwork della carta.
Infine, come ultimo comando, ho effettuato un `$sort` che ordinerà gli item prima per nome e poi per costo del mana per coerenza tra le varie funzioni.

Di seguito il comando completo:

```
cards = collection.aggregate([
      {"$project":
            {"cards":
                  {"$map":
                  {
                        "input": {"$objectToArray": "$cards"},
                        "in": "$$this.v"
                  }
                  }
            }
      },
      {"$unwind": "$cards"},
      {"$project":
            {
                  "_id": 0,
                  "cards.name": 1,
                  "cards.convertedManaCost": 1,
                  "cards.power":1,
                  "cards.colors": 1,
                  "cards.type": 1,
                  "cards.text": 1
            }
      },
      {"$sort": {"cards.name": 1, "cards.convertedManaCost":1}},
])
```

Il comando PyMongo `aggregate` ritornerà tutte le carte in un oggetto di tipo CommandCursor, le quali verranno salvate nella variabile cards.

Questa query sarà la base di partenza per le restanti query sviluppate per mtgDB, e verrà identificata per il resto della documentazione con la dicitura **query base**.

Segue una spiegazione dettagliata delle funzioni sviluppate, divise in due categorie:

- ➔ **Funzioni di interfaccia**: funzioni che verranno eseguite esclusivamente per tenere un certo decoro estetico al programma;
- ➔ **Funzioni di interazione con MongoDB**: funzioni che implementano le query per MongoDB, eseguite con l'ausilio dei metodi messi a disposizione da PyMongo.

Funzioni di interfaccia:

print_logo()
Stampa il logo del programma. Viene eseguita solo all'avvio e alla chiusura del programma.

print_menu()
Stampa il menu dei comandi esistenti. Viene eseguita dopo ogni operazione effettuata.

clear_terminal()
Pulisce il contenuto del terminal prima di eseguire un nuovo comando. Permette di avere una interfaccia sempre pulita con il focus sul menù e sull'output della funzione precedentemente chiamata.
Funziona sia su tutti i sistemi operativi desktop grazie all'uso dell'operatore "||" e della libreria os presente in Python, grazie alla quale è possibile passare direttamente alla linea di comando il comando corretto tra `clear` (per Linux e MacOS) e `cls` per Windows.
Di seguito il comando completo:

```
os.system('cls||clear')
```

Funzioni di interazione con MongoDB:

1: **load_db()**
Si occupa del caricamento del dataset `StandardCards.json` all'interno di MongoDB.
Di seguito è riportato il codice per l'importazione del JSON, utilizzando la libreria json presente in python:

```
with open("StandardCards.json") as json_db:
      database = json.load(json_db)
      collection.insert_one(database)
```

2: **print_n_cards()**
Stampa il numero di carte presenti nel database.

Per farlo, aggiunge alla **query base** il comando `$group` per raggruppare tutte le carte utilizzando il parametro `$name`, per poi contarle attraverso il comando `$sum`:

```
cards = collection.aggregate([
      {"$project":
            {"cards":
                  {"$map":
                  {
                        "input": {"$objectToArray": "$cards"},
                        "in": "$$this.v"
                  }
                  }
            }
      },
      {"$unwind": "$cards"},
      { "$group": { "_id": "$name" , "count": { "$sum": 1 } } }
])
```

Successivamente stampa il valore "`count`":

```
cards.next()["count"]
```

3: **add_card(**name, mana, power, colors, type, text, legalities**)**
Prende in input i seguenti parametri:

- ➔ name: nome della carta;
- ➔ mana: costo in mana della carta;
- ➔ power: punti forza della carta;
- ➔ colors: lista dei colori della carta;
- ➔ type: tipo della carta
- ➔ text: descrizione della carta
- ➔ legalities: lista dei formati in cui:
  - ◆ se la carta è giocabile in quel formato, il valore sarà "Legal";
  - ◆ se la carta non è giocabile in quel formato, il valore sarà "Banned".

Poiché colors e legalities sono liste, nell'interfaccia grafica, l'utente dovrà inserire tutti gli oggetti della lista separandoli con una virgola "**,**", in modo tale da poter separare i vari parametri della lista con l'uso della funzione `split()` presente in Python.
Questa funzione, per aggiungere una carta alla collection, aggiunge il comando PyMongo `update_one` alla **query base** con al suo interno il comando MongoDB `$set`, al quale viene passato un dizionario chiamato name contenente l'insieme dei parametri della carta:

```
cards = collection.update_one({},
    {"$set":
        {"cards." + name:{
         "name": name,
         "convertedManaCost": float(mana),
         "power":power,
         "colors": colors,
         "type": type,
         "text": text,
         "legalities": legalities}
        }
    }
)
```

4: **remove_card(**name**)**
Similmente alla funzione **add_card()**, verrà utilizzato il comando PyMongo `update_one`, stavolta con il comando MongoDB `$unset`, al quale verrà passato il nome della carta.
Verrà quindi cercata la carta e rimossa dalla collection con la seguente query:

```
cards = collection.update_one({},{"$unset":{"cards." + name:""}})
```

5: **search_card(**name**)**
Cerca nel database una carta attraverso il nome immesso dall'utente nell'interfaccia testuale.
Se la trova, stampa i suoi dati (utilizzando Pandas per una migliore formattazione).
Per cercarla utilizza il comando `$match`, passandogli il nome della carta da cercare.

```
cards = collection.aggregate([
       {"$project":
           {"cards":
               {"$map":
                   {
                       "input": {"$objectToArray": "$cards"},
                       "in": "$$this.v"
                   }
               }
           }
       },
       {"$unwind": "$cards"},
       {"$match": {"cards.name": name}},
       {"$project":
           {
               "_id": 0,
               "cards.name": 1,
               "cards.convertedManaCost": 1,
               "cards.power":1,
               "cards.colors": 1,
               "cards.type": 1,
               "cards.text": 1
           }
       },
       {"$sort": {"cards.name": 1, "cards.convertedManaCost":1}},
   ])
```

Verrà poi stampata la carta (se trovata).

6: **filter_card(**mana, colors**)**
Filtra le carte che hanno costo in mana minore o uguale al valore inserito e che hanno la combinazione di colori inserita.
Per farlo, va aggiunta alla **query base** un `$match` sulle due condizioni precedentemente introdotte. Per aggiungere due condizioni al `$match` viene usato il comando `$end`, al quale verranno passate le condizioni da rispettare.
Di seguito la query completa:

```
cards = collection.aggregate([
   {"$project":
       {"cards":
           {"$map":
               {
                   "input": {"$objectToArray": "$cards"},
                   "in": "$$this.v"
               }
           }
       }
   },
   {"$unwind": "$cards"},
   {"$match": {"$and": [{"cards.convertedManaCost": {"$lte": float(mana)}},
                        {"cards.colors": colors}]}},
   {"$sort": {"cards.name": 1, "cards.convertedManaCost":1}},
   {"$project":
       {
           "_id": 0,
           "cards.name": 1,
           "cards.convertedManaCost": 1,
           "cards.power":1,
           "cards.colors": 1,
           "cards.type": 1,
           "cards.text": 1
       }
    }
])
```

Verrà poi stampata la lista di carte che rispettano le condizioni sopra espresse.

7: **filter_top()**
Recupera dalla collection le 5 carte più forti.
Per farlo, oltre alla **query base**, andranno ordinate le carte seguendo il parametro power in ordine decrescente:

```
cards = collection.aggregate([
     {"$project":
        {"cards":
            {"$map":
                {
                    "input": {"$objectToArray": "$cards"},
                    "in": "$$this.v"
                }
            }
        }
    },
    {"$unwind": "$cards"},
    {"$project":
        {
            "_id": 0,
            "cards.name": 1,
            "cards.convertedManaCost": 1,
            "cards.power":1,
            "cards.colors": 1,
            "cards.type": 1,
            "cards.text": 1
        }
    },
    {"$sort": {"cards.power":-1}},
    {"$limit": 5}
])
```

Verranno poi stampate le 5 carte recuperate attraverso la precedente query.

8: **find_description(**text**)**
Elenca tutte le carte che contengono la frase presente nella stringa text inserita dall'utente nella descrizione.
Per farlo, tornerà utile il comando `$regex`, con il quale si potrà utilizzare una espressione regolare per cercare una frase all'interno di una stringa. Questa `$regex` verrà poi inserita all'interno di un `$match`:

```
cards = collection.aggregate([
    {"$project":
        {"cards":
            {"$map":
                {
                    "input": {"$objectToArray": "$cards"},
                    "in": "$$this.v"
                }
            }
        }
    },
    {"$unwind": "$cards"},
    {"$match": {"cards.text": {"$regex" : text}}},
    {"$project":
        {
            "_id": 0,
            "cards.name": 1,
            "cards.convertedManaCost": 1,
            "cards.power":1,
            "cards.colors": 1,
            "cards.type": 1,
            "cards.text": 1
        }
    },
    {"$sort": {"cards.name": 1, "cards.convertedManaCost":1}}
])
```

Verrà poi stampata la lista di carte che rispettano le condizioni sopra espresse.

9: **check_legalities(**name, format**)**
Riceve in input un nome e un formato, e controlla se la carta inserita è legalmente giocabile nel formato inserito. Per farlo, controlla se nella lista legalities è presente una coppia chiave valore che ha per chiave il formato e come valore la stringa "*Legal*".

```
cards = collection.aggregate([
        {"$project":
            {"cards":
                {"$map":
                    {
                        "input": {"$objectToArray": "$cards"},
                        "in": "$$this.v"
                    }
                }
            }
        },
        {"$unwind": "$cards"},
        {"$match": {"cards.name": name}},
        {"$match": {"cards.legalities." + format: "Legal"}},
        {"$project":
            {
                "_id": 0,
                "cards.name": 1,
                "cards.convertedManaCost": 1,
                "cards.power":1,
                "cards.colors": 1,
                "cards.type": 1,
                "cards.text": 1
            }
        },
        {"$sort": {"cards.name": 1, "cards.convertedManaCost":1}}
])
```

Al termine, avvisa l'utente se la carta inserita è legalmente giocabile nel formato scelto.

10: **drop_db()**
Effettua il drop del database creato con la funzione **load_db()**.
Per farlo, basta utilizzare la funzione `drop_database()`, resa disponibile da PyMongo:

```
client.drop_database('mtgdb')
```

Oltre al drop del database, bisognerà aggiornare la variabile `is_loaded` (la quale mantiene lo stato del database, `True` se è presente un database, `False` altrimenti):

```
is_loaded = False
```

Start

Questa sezione fa da main al programma. Infatti implementa uno switch (attraverso un sistema `if-elif-else`) per controllare e chiamare la funzione scelta dall'utente.

Ogni funzione avrà il suo `if`, mentre l'`else` (che fa da caso *default* dello switch) si occuperà di avvisare l'utente che non vi è alcun comando associato al valore erroneamente inserito.

Indifferentemente dal caso dello switch, verrà pulito il terminale per dare un effetto "applicazione interattiva" (se pur con le limitazioni indotte dalla linea di comando), e verrà controllato lo stato del database (non verrà eseguita la funzione se il database non è stato precedentemente caricato).

Test guidato di mtgDB

Segue una lista di comandi inseribili in ordine per testare le funzionalità del programma sviluppato e i suoi punti critici.

Nel seguente esempio verrà utilizzata come carta di riferimento per le operazioni effettuate (aggiunta, ricerca, rimozione) una carta non presente nel database Standard, ovvero il **Black Lotus**, non che la carta più famosa e costosa dell'intero gioco di carte, stampata originariamente nel 1993.

Inoltre fare attenzione ad inserire correttamente le lettere maiuscole e minuscole, in quanto l'intero programma è case-sensitive.

- ➔ Seguire le istruzioni presenti nell'apposita sezione per l'installazione di MongoDB, di Python e delle librerie PyMongo e Pandas;
- ➔ Avviare MongoDB;
- ➔ Eseguire mtgdb.py. Verrà mostrato il logo e il menù;
- ➔ Essendo questo il primo utilizzo, va caricato il dataset. Utilizzare il comando **1**. Notare che, provando ad eseguire qualunque altro comando, verrà avvisato l'utente della mancanza di un database, invitandolo ad eseguire il comando **1**;
- ➔ Contare il numero di carte presenti sul database con il comando **2**. Inizialmente saranno presenti *1419* carte (ovvero quelle presenti nel dataset originale);

- ➔ Eseguire il comando **3** per inserire una nuova carta. Verrà richiesto interattivamente all'utente di inserire (in step separati) tutti i dati necessari, di seguito l'elenco:
  - ◆ Name (Stringa) - inserire la stringa "**Black Lotus**";
  - ◆ Mana cost (Numero float) - inserire il valore **5**;
  - ◆ Power (Numero intero) - inserire il valore **9**;
  - ◆ Colors (Lista di Caratteri) - inserire la stringa **R,G** virgola inclusa (che sta per **R**ed e **G**reen);
  - ◆ Type (Stringa) - inserire la stringa "**Artifact**";
  - ◆ Text (Stringa) - inserire la descrizione della carta "**Sacrifice Black Lotus: Add three mana of any one color.**";
  - ◆ Legalities (Lista di Stringhe) - inserire "**vintage**".
- ➔ Eseguire nuovamente il comando **2** per contare le carte. Ora saranno 1420;
- ➔ Eseguire il comando **5** per cercare la carta appena aggiunta. Inserire quindi il nome "**Black Lotus**". Sopra al menù verrà stampata la versione formattata testualmente della carta cercata (se presente);
- ➔ Eseguire il comando **6** per elencare le carte con un certo costo di mana minore o inferiore al valore inserito, e della combinazione di colori inserita. Inserire quindi prima il valore **5**, poi **R,G**. Verrà stampata la lista di carte (ordinata per nome e per costo in mana). In prima posizione troveremo il Black Lotus appena aggiunto;
- ➔ Eseguire il comando **7** per recuperare le migliori 5 carte a livello di forza (power). In prima posizione, ancora una volta, verrà mostrato il Black Lotus appena aggiunto;
- ➔ Eseguire il comando **8** per cercare una carta inserendo una parola (o una frase). Questa stringa verrà cercata all'interno delle descrizioni delle carte. Inserire la Stringa "**Add three mana**". Verranno mostrate le carte che presentano la frase Add three mana nella descrizione, tra cui il Black Lotus;
- ➔ Eseguire il comando **9** per controllare se la carta Black Lotus è giocabile nel formato standard. Inserire la Stringa "**Black Lotus**", e successivamente la Stringa "**standard**". Non risulterà alcuna carta chiamata Black Lotus giocabile nel formato standard.
- ➔ Eseguire nuovamente il comando **9** per controllare invece se la carta Black Lotus è legale nel formato vintage. Inserire la Stringa "**Black Lotus**", e successivamente la Stringa "**vintage**". L'utente verrà avvisato che la carta Black Lotus è legalmente giocabile nel formato vintage;
- ➔ Eseguire il comando **4** per rimuovere la carta Black Lotus precedentemente inserita. Inserire la Stringa "**Black Lotus**";
- ➔ Eseguire nuovamente il comando **2** per contare le carte. Ora saranno nuovamente 1419;
- ➔ Eseguire il comando **10** per eliminare il database. Fatto questo, il programma non permetterà di utilizzare le precedenti funzioni per mancanza di un database;
- ➔ Infine, eseguire il comando **0** per terminare correttamente il programma.