UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Progettazione e realizzazione di una base di dati per la raccolta di dati dall'opera "Come vi piace" di Shakespeare"

Relatore: Prof. Di Nunzio Giorgio Maria

Laureando: Anton Pavel Lucian

ANNO ACCADEMICO 2022 – 2023

Data di laurea 16/11/2023

INDICE

1. Introduzione

L'obiettivo di queta tesi di laurea è realizzare una base di dati per facilitare la racolta dei dati, presenti in un foglio elettronico relativi all'opera "Come vi piace" di Shakespeare, e poter in seguito fare un analisi dei dati in modo più veloce ed efficiente e fornire una visualizza deì dati ordinata.

Il lavoro si suddivide in due fasi:

- la prima relativa alla progettazione del database, dove di affronterrano tutti i problemi relativi alla progettazione di un database di tipo relazionale. Si passerà attraverso le fasi di progettazione concettuale, progettazione logica e progettazione fisica attraverso l'uso del linguaggio SQL.
- la seconda relativa allo sviluppo di un applicativo software in Python per verificare la correttezza dei dati, creare il database tramite PostgreSQL, popolare il database e fornire un interfaccia per inserire ulteriore dati da terminale o visualizzare i dati già inseriti, facendo uso delle librerie software Pandas e Psycopg.

2. Lo studio di testi, tramite tecniche di text mining e data visualization

I testi, nel nostro caso le tragedie teatrali, contengono una grande quantità di informazioni rappresentate però in un formato che non consente di visualizzare in modo veloce ed efficace le informazioni più importanti.
Per consentire quindi la fruizione delle informazioni in modo automizzato bisogna fare uso di tecniche di text mining e data visualization. Una prima parte di raccolta di dei dati in una struttura dati semi-struttura è già stata volta, ciò che dovremo fare è organizzare i dati in modo da poter essere inseriti in un database. A questo punto i dati presenti nel database possono essere manipolati a piacimento per poterli visualizzare nel modo più opportuno.
Andiamo quindi a vedere cos'è il text mining.

2.1 Text mining

Il text mining è il processo per ricavare informazioni di alta qualità da risorse scritte di vario tipo (siti Web, libri, e-mail, recensioni, tragedie teatrali), in genere queste risorse sono non strutturate. Prima di poter applicare le tecniche di text mining, è necessaria una fase di pulizia e trasformazione dei dati di testo in un formato utilizzabile, ciò è un aspetto fondamentale dell'elaborazione NLP (Natural Language Processing) e in genere fa uso delle seguenti tecniche: identificazione della lingua, tokenizzazione, suddivisione in blocchi, etichettatura di parti del discorso e analisi della sintassi. Dopo aver completato questa fase, è possibile applicare le tecniche di text mining, alcune di esse sono:

- Il **recupero delle informazioni** (Information retrieval, IR) è l'attività di ricerca informazioni o documenti attraverso delle parole chiavi o composizione di esse (query). Alcune attività secondarie del IR sono: la *tokenizzazione*, processo attraverso cui il testo viene scomposto in frasi e parole dette "token"; la *derivazione*, processo di separazione dei prefissi e dei suffisi per derivare la parola base.
- L'**elaborazione NLP** utilizza metodi di informatica,data science, AI e linguistica per far comprendere il linguaggio umano ai computer. Le attività secondarie sono: riepilogo, questa tecnica fornisce una sintesi del testo; *etichettatura di parti del discorso*, ad ogni token viene assegnati un tag in base a dove si trova nel testo; *categorizzazione del testo*.
- L'**estrazione delle informazioni** (Information extraction, IE) mette in risalto le parti più importanti dei dati durante ricerca delle informzioni. Svolge un'attività di estrazione delle informazioni strutturate dal testo per memorizzarle in una base di dati.

Differenze tra data mining e text mining

Il data mining si basa sulla raccolta e l'analisi automizzata di grandi quantità di fonti di dati al fine di trovare approfondimenti o modelli non conosciuti, in modo da fornire alcune informazioni preziose. Data mining è conosciuto anche come scoperta di conoscenza dai dati. Il text mining è una parte del data mining che cerca di estrarre informazioni utili dalle fonti di dati di testo.

2.2 Data visualization

La visualizzazione dei dati (data visualization) è la rappresentazione dei dati e delle informazioni facendo uso di diagrammi, grafici, mappe a altri strumenti. Se fatta in maniera corretta ci consente di individuare facilmente modelli, tendenze o valori anomali presenti in un set di dati.

La visualizzazione dei dati presenta i dati anche a chi non ha conoscenze su ciò che viene trattato. Alcuni vantaggi della visualizzazione dei dati sono: la facile condivisione delle informazioni; visualizzazione di modelli e relazioni.

3. Progettazione e realizzazione del Database

Nell'ambito delle basi di dati, negli anni, si è stabilita una metodologia di progetto che ha dimostrato di soddisfare pienamente le seguenti proprietà:

- la *generalità* rispetto alle applicazioni e ai sistemi utilizzati (e quindi la possibilità di utilizzo indipendentemente dal problema sottoposto e dagli strumenti a disposizione);
- la *qualità del prodotto* in termini di correttezza, completezza ed efficienza in rapporto alle risorse disponibili;
- la *facilità d'uso* delle strategie e dei modelli utilizzati.

Tale metodologia di progettazione è suddivisa in tre fasi principali da effetuare in cascata e si basa su un principio dell'ingegneria semplice ma molto efficace: dividere in maniera netta la prima fase di decisione relative a "cosa" rappresentare in una base di dati , da quelle successive relative a "come" farlo (seconda e terza fase).
Le fasi sono le seguenti:

- **Progettazione concettuale**
- **Progettazione logica**
- **Progettazione fisica**

Ogni fase rappresenta un diverso livello di astrazione dei dati e delle relazioni tra essi.
A fasi diverse della progettazione corrispondono modelli diversi per la rappresentazione dei dati, ovvero tecniche per rappresentare gli aspetti rilevanti della realtà da modellare, definite da strumenti e vincoli specifici. La rappresentazione prodotta seguendo le regole imposte dal modello viene definita schema.

3.1 Analisi dei requisiti

In questa sezione ci occuperemo della fase preliminare di **raccolta e analisi dei requisiti**.
L'obiettivo di questa fase è quello di raccogliere ed analizzare i requisiti relativi ai fabbisogni informativi degli utenti. I requisiti raccolti permettono di descrivere sia i dati sia le operazioni che si prevedono di eseguire su di essi.
I dati che verrano inseriti nella base di dati provengon dal foglio elettronico "AS_YOU_LIKE_IT_EXCEL_THREE_TYPE_LOGIC_WITH_WORDS.xlsx".
Dal seguente foglio verranno raccolte le informazioni riguardanti i dati dei dialoghi presenti nell'opera "Come vi piace". Per ogni dialogo sono presenti i seguenti dati : atto, scena, speaker, destinatario, logica, adjunct (area semantica del dialogo) e words. Inoltre verrà inserito il nome e l'autore dell'opera.

Frasi per Opera

Ogni opera viene inserita se si ha almeno un dialogo ad essa associata. Per ogni opera viene inserito il titolo (attributo chiave) e l'autore. Viene mantenuta la lista di atti appartenente all'opera.

Frasi per Atto

Ogni atto viene indentificato da un ID (attributo chiave). Si vuole mantenere la descrizione dell'atto e la relativa lista di scene che lo compongono.

Frasi per Scena

Ogni scena viene anch'essa identificata da un ID (attributo chiave) ed ha una descrizione. Per ogni scena so vuole mantenere la lista di dialoghi che la compongono.

Frasi per Logica

La logica è un organizzazione astratta dell'argomento trattato nel dialogo. Ogni logica ha un nome (attributo chiave) e una descrizione.

Frasi per Dialogo

Ogni dialogo appartiene ad una sola scena e ha solo una logica. Viene identificato con l'ID (attributo chiave). Le informazioni che ci interessano di un dialogo sono lo speaker, il destinatario (opzionale), gli adjuncts (aree semantiche del dialogo) e le words (parole relative alle aree semantiche del dialogo).

L'inserimento dei dati avverà ogni qual volta si voglia aggiungere un dialogo che non sia già presente nel database. Le interrogazioni potranno essere effettuate quando si vuole visualizzare i dati o effettuare diverse analisi su di essi.

3.2 Progettazione concettuale

La progettazione concettuale ha lo scopo di rappresentare le specifiche informali della reltà di interesse in termini di una descrizione formale e completa, ma indipendente dai criteri di rappresentazione utilizzati nei sistemi di gestione di basi di dati (DBMS Database Mangement System). Il risultato di questa fase viene detto *schema concettuale* e fa riferimento a un *modello concettuale* dei dati. I modelli concettuali ci consentono di descrivere l'organizzazione dei dati ad un alto livello di astrazione, senza tenere conto degli aspetti implementativi. In questa fase infatti, il progettista deve cercare di rappresentare la descrizione dei requisiti della base di dati, senza preoccuparsi né delle modalità, né dell'efficienza dei programmi che faranno uso di queste informazioni.

I principali vantaggi della progettazione concettuale sono:

- Migliorare la comunicazione tra utenti finali e progettisti, essendo in grado di rappresentare i dati della realtà d'interesse in termini di un modello formale, ad **alto livello.** Poiché si fa uso di un modello grafico (modello ER) di rappresentazione dei requisiti che è di facile comprensione e permette la verifica della corrispondenza dei requisiti espressi dal committente.
- Avere una rappresentazione della base di dati **indipendente** dalla scelta del DBMS utlizzato.

Il modello ER citato in precendenza, chiamato anche modello Entità-Associazione, è un modello concettuale che viene spesso utilizzato per la progettazione delle applicazioni di basi di dati relazioni. Tale modello propone un insieme di costrutti che permetto di descrivere la realtà di interesse attraverso uno schema chiamato diagramma,o schema, ER, che permette di verificare se tutti gli elementi raccolti durante l'analisi dei requsiti sono stati presi in considerazione. I costrutti di base del modello ER sono tre: entità, associazioni e attributi.

Vedremo quindi in questa sezione:

- Lo **schema Entità-Associazione** che rappresenta i requisiti raccolti
- Il **Dizionati dei dati** dove verrano descritte le varie entità, e i loro attributi, e le associazioni
- Le **Regole di vincolo**

3.2.1 Schema ER

In Figura 1 è raffigurato lo schema concettuale costruito sulla base dei requisiti raccolti in precedenza

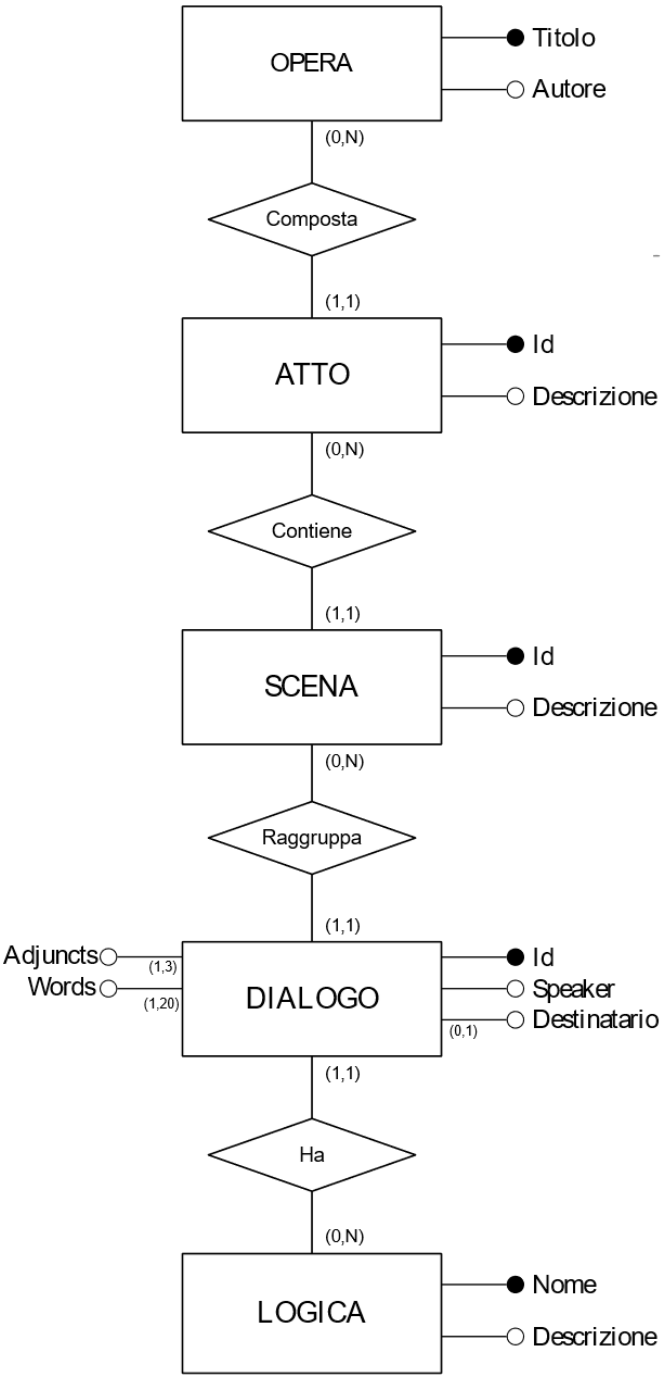

Figura 1

3.2.2 Dizionario dei dati

Entità

Entità	Descrizione	Attributi	Identificatore
Opera	Opera letteraria divisa in atti	Titolo, Autore	Titolo
Atto	Suddivisione principale di un'opera	ID, Descrizione	ID
Scena	Suddivisione di un atto	ID, Descrizione	ID
Dialogo	Ciò che viene detto da un personaggio ad un altro	ID, Speaker, Destinatario, Adjuncts, Words	ID
Logica	Organizzazione astratta dell'argomento trattato nel dialogo	Nome, Descrizione	Nome

Associazioni

Relazione	Descrizione	Entità coinvolte	Attributi
Composta	Associa un'opera ai suoi atti	Opera (0,N), Atto (1,1)	
Contiene	Associa un atto alle sue scene	Atto (0,N), Scena (1,1)	
Ragruppa	Associa una scena ai suoi dialoghi	Scena (0,N), Dialogo (1,1)	
Ha	Associa un dialogo alla sua logica	Dialogo (1,1), Logica (0,N)	

3.3 Progettazione logica

La progettazione logica consiste nella traduzione dello schema concettuale definito nella fase precedente, in termini del modello di rappresentazione dei dati adottato dal DBMS utilizzato. Il risultato di questa fase viene chiamato *schema logico* della base di dati e fa riferimento a un *modello logico* dei dati. Un modello logico ci consente di descrivere i dati secondo una rappresentazione che è ancora indipendente da dettagli fisici, ma molto più pratica perché disponibile nei sistemi di gestione di basi di dati. Le scelte progettuali effettuate in questa fase si basano su criteri di ottimizzazione delle operazioni che verranno effettuate sui dati. In genere si fa uso anche di tecniche formali di verifica della qualità dello schema logico.
La progettazione logica si divide in due fasi:

- **Ristrutturazione dello schema ER**, è indipendente dal modello logico scelto e serve per ottimizzare ulteriormente lo schema e semplificarlo per la fase successiva.
- **Traduzione verso il modello logico**, fa riferimento al specifico modello logico utilizzato, quindi nel nostro caso il modello relazionale.

Nel nostro caso non è necessaria la prima fase.
In questa sezione vedremo:

- lo **schema relazionale** prodotto per la realtà di interesse del progetto.
- un insieme di **regole di vincolo** dello schema relazionale che non sono direttamente deducibili da esso.

3.3.1 Schema relazionale

La traduzione dello schema ER nello schema relazionale è stata effettuata rispettando i seguenti criteri:

- Ogni entità è stata tradotta in una relazione con lo stesso nome, gli stessi attributi e avente come chaive il suo identificatore;
- Ogni associazione uno-a-molti è stata tradotta ed integrata nella relazione che nello schema ER ha cardinalità (1,1), contente gli attibuti della relativa relazione e l'identificatore dell'entità che fa parte dell'associazione.

In Figura 2 lo schema relazione prodotto seguendo i precedenti criteri.

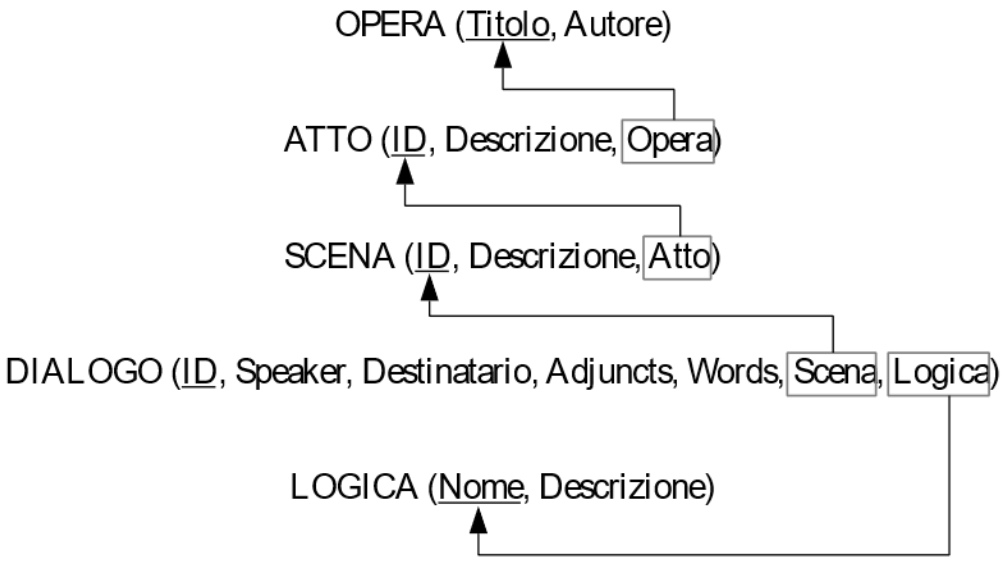

Figura 2

3.3.2 Regole di vincolo

- **RV1:** Gli attributi di Opera devono essere non nulli.
- **RV2:** Gli attributi di Atto devono essere non nulli.
- **RV3:** Gli attributi di Scena devono essere non nulli.
- **RV4:** Gli attributi di Dialogo devono essere non nulli, tranne l'attibuto "Destinatario" che è opzionale.
- **RV5:** Gli attributi di Logica devono essere non nulli

3.4 Progettazione fisica

Nella fase della progettazione fisica lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati(organizzazione dei file e degli indici). Il risultato di questa fase è lo *schema fisico* e fa riferimento a *un modello fisico* dei dati. Tale modello dipende dallo specifico DBMS scelto (nel nostro caso PostgreSQL) e si basa su criteri di organizzazione fisica dei dati in quel sistema.
In questa sezione vedremo la traduzione dello schema logico in schema fisico, utilizzando il linguaggio SQL (Stuctured Query Language).

3.4.1 Schema fisico

```
CREATE TABLE Opera(
    Titolo VARCHAR(124) PRIMARY KEy,
    Autore VARCHAR(50) NOT NULL
);
CREATE TABLE Atto(
    ID SERIAL PRIMARY KEY,
    Descrizione Varchar(50) NOT NULL,
    Opera VARCHAR(124) NOT NULL REFERENCES Opera(Titolo)
);
CREATE TABLE Scena(
    ID SERIAL PRIMARY KEY,
    Descrizione Varchar(50) NOT NULL,
    Atto INTEGER NOT NULL REFERENCES Atto(ID)
);
CREATE TABLE Logica(
    Nome VARCHAR(50) PRIMARY KEY,
    Descrizione VARCHAR(50) NOT NULL
);
CREATE TABLE Dialogo(
    ID SERIAL PRIMARY KEY,
    Speaker VARCHAR(50) NOT NULL,
    Destinatario VARCHAR(50),
    Adjuncts VARCHAR(50) [3],
    Words VARCHAR(100) [20],
    Scena INTEGER NOT NULL REFERENCES Scena(ID),
    Logica VARCHAR(50) NOT NULL REFERENCES Logica(Nome)
);
```

4. Strumenti utilizzati

In questo capitolo vengono elencati i principali strumenti utilizzati per lo sviluppo del progetto

4.1 Python

Python è un linguaggio di programmazione ad alto livello adatto a sviluppare applicazioni distribuite, computazione numerica, system testing e analisi dei dati. Di seguito un elenco delle principali caratteristiche del linguaggio:

- È un linguaggio **interpretato**: i programmi prima di essere eseguiti vengono automaticamente compilati in un formato chiamato *bytecode.*
- È **free**: è completamente gratuito e la sua distribuzione è priva di copyright, nonostante ciò ha una community molto attiva e riceve continui miglioramenti per mantenerlo aggiornato.
- È **multi-paradigma**: supporta sia la programmazione procedurale sia la programmazione ad oggetti.
- È **portabile**: è possibile utilizzarlo su diverse piattaforme (ad es. Linux, Windows, Android, ecc.) a condizione che abbia l'interprete Python installato.
- È ricco di **librerie**: ogni installazione include la Standard Library e tramite il Python Package Index è possibile scaricare e installare moduli aggiuntivi scritti dalla community.
- **Gestice automaticamente la memoria**: fa uso di un meccanismo di *garbage collection* che alloca e rilascia automaticamente la memoria.
- È **integrabile con altri linguaggi**: utlizzando interpreti appositi.

La versione utilizzata è la 3.11.

4.2 PostgreSQL

PostgreSQL è un DBMS (Database Management System) relazionale ad oggetti, open source e gratuito. È stato sviluppato da Michael Stonebraker nel 1986 come evoluzione del prototipo di ricerca Postrgres.

Le principali caratteristiche sono affidabilità, integrità dei dati, estensibilità e scalabilità sia per quanto riguarda i dati che puà memorizzare, sia per la quantità di utenti simultanei che può gestire. PostgreSQL implementa il modello Client\Server: il Client , che può essere anche un'applicazione utente, può eseguire operazioni sulla base di dati, il Server gestisce la base di

dati, accetta le connessioni ed esegue le richieste effettuate dal client sulla base di dati. È in grado di gestire connessioni multiple e concorrenti, inoltre è conforme allo standard SQL.
La versione utilizzata è la 15.

4.3 pgAdmin 4

PgAdmin è un applicazione C++, open source, che fornisce un interfaccia grafica per l'amministrazione di database di PostgreSQL. Le principali funzionialità fornite sono: query tool, visualizzazione di definizione SQL, proprietà e dipendenze degli oggetti presenti nel database; visualizzazione, popolazione e modifica dei dati in modalità tabellare.
Questo tool è stato utilizzato in fase di sviluppo per controllare la corretta esecuzione delle query.
La versione utilizzata è la 7.5

4.4 Psycopg2

Psycopg è l'adapter di database PostgreSQL più utilizzato per il linguaggio Python. Le principali caratteristiche sono l'implementazione della specifica Python DB API 2.0 e la sicurezza dei thread (più thread possono condividere la stessa connessione).
È implementato principalmente in C, inoltre molti tipi di dati di Python sono supportati nativamente e adattati ai tipi di dati PostgreSQL corrispondenti.
La versione utilizzata è la 2.9.7

4.5 Pandas

Pandas è una libreria software di Pyhton, mette a disposizione strutture dati veloci, flessibili ed espressive che permettono di lavorare con dati di tipo sequenziali o tabellari, in modo semplice e intuitivo. Il suo scopo è fornire un elemento di alto livello per eseguire analisi dei dati del mondo reale in Python. Le principali strutture dati in Pandas sono due:

- Series: array unidimensionale con etichette degli assi
- DataFrame: dati tabulari bidimensionali ed eterogeni, con dimensioni variabili. Gli assi sono etichettati(righe e colonne)

Alcune delle funzionalità che Pandas offre sono:

- Gestione dei dati mancanti molto intuitiva, vengono rappresentati con il valore Nan

- Robusti strumenti per l'IO per caricare dati da file flat (delimitati), file Excel, database, ecc.
- Mutabilità delle dimensioni: le colonne possono essere inserite ed eliminate dai DataFrame
- Funzionalità SQL like
- Varietà di funzioni e metodi integrati per l'analisi e la visualizzazione dei dati

La versione utlilizzata è la 2.1.0

4.6 IDLE

IDLE (abbreviazione di Integrated Development and Learning Environment) è un ambiente di sviluppo integrato per Python. Ha le seguenti caratteristiche:

- È scritto esso stesso in Python
- Multipiattaforma: funziona su Windows, Unix e MacOs
- Mette a disposizione una finestra di shell di Python
- Editor di testo multi finestra con colorazione sintattica, indentazione semplificata ,suggerimenti sulle chiamate e completamento automatico
- Debugger: impostazione breakpoint, esecuzione passo passo e visualizzazione di spazi dei nomi globali e locali

La versione utilizzata è la 3.11

5. Sviluppo del software

Il programma completo è disponibile al seguente link Github :

https://github.com/pavellucian/tesi-Pavel-Lucian-Anton/blob/main/progetto_tesi.py

Il programma si compone di 4 funzioni principali. L'utente ad ogni avvio del programma può decidere cosa fare e dopo aver effettuato un operazione potrà scegliere se uscire oppure eseguire un'altra operazione. Le quattro funzioni principali che l'utente può fare sono:

- Creazione del database
- Inserimento dati da foglio excel
- Inserimento dati manuale
- Stampa dei dati

```
if __name__ == "__main__":
    stop=False
    #loop che permette all'utente di eseguire più operazioni ogni volta che esegue
    #il programma
    while not(stop):
        print("Inserire il numero dell'operazione che si vuole sseguire")
        print("1. Creazione database")
        print("2. Inserimento dati da foglio excel")
        print("3. Inserimento manuale dei dati")
        print("4. Stampa dati")
        print("")
        print("5. Per uscire dal programma")
        try:
            op = int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 5")
            continue
        if op==1:
            create_DB()          #creazione DB
        elif op==2:
            insert_from_excel() #inserimento dati da foglio excel
        elif op==3:
            manual_insert()      #inserimento manuale
        elif op==4:
            visualize_data()     #stampa dati
        elif op==5:
            stop=True
        else: print("ATTENZIONE! Inserire un numero compreso tra 1 e 5")
```

Andiamo quindi ad analizzare le funzioni una ad una:

5.1 create_DB

Con questa funzione viene prima creata la connessione a PostgreSQL, in seguito si crea il database e ci si collega ad esso utilizzando la funzione connect_DB(). Vengono quindi create le singole tabelle seguendo la struttura discussa nel paragrafo 3.4.1.

Per la connessione al database e la creazione delle tabelle sono state utilizzate le seguenti funzioni e classi della la libreria psycopg2:

- la funzione **connect()**: crea una nuova sessione di database e restituisce una istanza di *connection*
- la classe **connection** permette di: creare nuove istanze del cursore tramite il metodo *cursor()* per eseguire commandi e query nel database; terminare le transazioni utilizzando i metodi *commit()* e *rollback()*

Nel caso vengano generate eccezzioni non gestite durante la creazione o la connessione al database il programma si chiuderà con un messaggio di errore relativo a dove è sorto il problema.

```
#funzione per la creazione del DB
def create_DB():
    try:
        con = psycopg2.connect(user = "postgres",
                               host = "localhost",
                               password = "admin",
                               port = 5432)
        con.autocommit = True
        cur = con.cursor()
        cur.execute("""CREATE DATABASE terzodb;""")
        print("Creazione Database effettuata")
        cur.close()
        con.close()
    except psycopg2.errors.DuplicateDatabase:
        print("Database già creato")
    except:
        print("Errore creazione DB")
        sys.exit()

    #connesione al DB appena creato
    connect_DB()

    #creazione tabella Opera
    try:
        curr.execute("""CREATE TABLE Opera(
                    Titolo VARCHAR(124) PRIMARY KEy,
                    Autore VARCHAR(50) NOT NULL
                    );
        """)
        conn.commit()
        print("Creata tabella Opera")
    except psycopg2.errors.DuplicateTable:
        print("Tabella Opera già creata")
        curr.execute("ROLLBACK")
        conn.commit()
    except:
        print("Errore creazione tabella Opera")
        sys.exit()
```

```
#creazione tabella Atto
try:
    curr.execute("""CREATE TABLE Atto(
                ID SERIAL PRIMARY KEY,
                Descrizione Varchar(50) NOT NULL,
                Opera VARCHAR(124) NOT NULL REFERENCES Opera(Titolo)
                );
    """)
    conn.commit()
    print("Creata tabella Atto")
except psycopg2.errors.DuplicateTable:
    print("Tabella Atto già creata")
    curr.execute("ROLLBACK")
    conn.commit()
except:
    print("Errore creazione tabella Atto")
    sys.exit()

#creazione tabella Scena
try:
    curr.execute("""CREATE TABLE Scena(
                ID SERIAL PRIMARY KEY,
                Descrizione Varchar(50) NOT NULL,
                Atto INTEGER NOT NULL REFERENCES Atto(ID)
                );
    """)
    conn.commit()
    print("Creata tabella Scena")
except psycopg2.errors.DuplicateTable:
    print("Tabella Scena già creata")
    curr.execute("ROLLBACK")
    conn.commit()
except:
    print("Errore creazione tabella Scena")
    sys.exit()

#creazione tabella Logica
try:
    curr.execute("""CREATE TABLE Logica(
                Nome VARCHAR(50) PRIMARY KEY,
                Descrizione VARCHAR(50) NOT NULL
                );
    """)
    conn.commit()
    print("Creata tabella Logica")
except psycopg2.errors.DuplicateTable:
    print("Tabella Logica già creata")
    curr.execute("ROLLBACK")
    conn.commit()
except:
    print("Errore creazione tabella Logica")
    sys.exit()
```

```
#creazione tabella Dialogo
try:
    curr.execute("""CREATE TABLE Dialogo(
                 ID SERIAL PRIMARY KEY,
                 Speaker VARCHAR(50) NOT NULL,
                 Destinatario VARCHAR(50),
                 Adjuncts VARCHAR(50) [3],
                 Words VARCHAR(100) [20],
                 Scena INTEGER NOT NULL REFERENCES Scena(ID),
                 Logica VARCHAR(50) NOT NULL REFERENCES Logica(Nome)
                 );
    """)
    conn.commit()
    print("Creata tabella Dialogo")
except psycopg2.errors.DuplicateTable:
    print("Tabella Dialogo già creata")
    curr.execute("ROLLBACK")
    conn.commit()
except:
    print("Errore creazione tabella Dialogo")
    sys.exit()
curr.close()
conn.close()
input("Premere INVIO per continuare")
```

Di seguito la funzione connect_DB che verrà utilizzata anche in altre parti del programma

5.1.1 connect_DB

Viene instaurata la connessione al database e viene creato il cursore per interagire con il database.

```
#connesione al DB
def connect_DB():
    try:
        global conn
        conn = psycopg2.connect(database="terzodb",
                                user = "postgres",
                                host = "localhost",
                                password = "admin",
                                port = 5432)
        global curr
        curr = conn.cursor()
        print("Connessione al database effettuata")
    except:
        print("Errore durante la connessione al databse")
        sys.exit()
```

5.2 Helper functions per l'inserimento dei dati

Funzioni utilizzate sia per l'inserimento dei dati da foglio excel sia per l'inserimento manuale dei dati, per migliorare la riusabilità del codice e non ripetere inutilmente parti del codice.
Le funzioni sono le seguenti : *insert_opera*, *insert_atto*, *insert_scena*, *insert_logica*, *insert_dialogo*.
La logica di funzionamento delle funzioni è simile tra di loro:

- le funzioni **insert_opera** e **insert_logica**: verificano se i parametri passati sono stringhe vuote e le sostituiscono con il valore 'NULL', dopodiché provano ad inserire il record nella rispettativa tabella, se il record è già presente con la stessa chiave primaria esso non viene inserito e viene mostrato all'utente un messaggio di errore per avvisarlo di ciò, inoltre vengono rispettati i rispetti vincoli di valori nulli dove non ammessi;
- le funzioni **insert_atto**, **insert_scena** e **insert_dialogo**: anch'esse verificano se i parametri passati sono stringhe vuote e le sostituiscono con il valore 'NULL', in seguito controllano se un record con gli stessi è già presente nel database, se si viene chiesto all'utente se vuole inserire il record con gli stessi dati, viene quindi inserito il record rispettando i vincoli di integrità e i vincoli di valori nulli non ammessi dove presenti.

Di seguito il codice di queste funzioni:

```
#metodo per l'inserimento di un'opera
def insert_opera(titolo,autore):
    if titolo=='': titolo='NULL'
    else: titolo="'"+titolo+"'"
    if autore=='': autore='NULL'
    else: autore="'"+autore+"'"
    try:
        curr.execute("""INSERT INTO Opera VALUES ({0},{1})""".format(titolo,autore))
        conn.commit()
        print("Opera ({0}, {1}) inserita con successo".format(titolo,autore))
    except psycopg2.errors.UniqueViolation:
        print("ERRORE: Opera ({0}, {1}) già presente nel database".format(titolo,autore))
        curr.execute("ROLLBACK")
        conn.commit()
    except psycopg2.errors.NotNullViolation:
        print("ERRORE: Valori null non ammessi")
        curr.execute("ROLLBACK")
        conn.commit()
    input("Premere INVIO per continuare\n")
```

```
#metodo per l'inserimento di un'atto
def insert_atto(descrizione,opera):
    if descrizione=='': descrizione='NULL'
    else: descrizione="'"+descrizione+"'"
    if opera=='': opera='NULL'
    else: opera="'"+opera+"'"
    cont='s'
    curr.execute("""SELECT ID FROM Atto
                    WHERE Descrizione={0} AND Opera={1}""".format(descrizione,opera))
    id=str(curr.fetchall())
    if id!="[]":
        print("ATTENZIONE: atto ({0}) già presente per l'opera ({1}),\
vuoi continuare lo stesso? (s/n)".format(descrizione,opera))
        cont=input()
    if cont=='s':
        try:
            curr.execute("""INSERT INTO Atto(Descrizione,Opera)
                            VALUES ({0},{1})""".format(descrizione,opera))
            conn.commit()
            print("Atto ({0}, {1}) inserito con successo".format(descrizione,opera))
        except psycopg2.errors.ForeignKeyViolation:
            print("ERRORE: Opera ({0}) non presente nel database,\
inserire prima l'opera".format(opera))
            curr.execute("ROLLBACK")
            conn.commit()
        except psycopg2.errors.NotNullViolation:
            print("ERRORE: Valori null non ammessi")
            curr.execute("ROLLBACK")
            conn.commit()
        input("Premere INVIO per continuare\n")

#metodo per l'inserimento di una scena
def insert_scena(descrizione,atto):
    if descrizione=='': descrizione='NULL'
    else: descrizione="'"+descrizione+"'"
    if atto=='': atto='NULL'
    else: atto="'"+atto+"'"
    cont='s'
    curr.execute("""SELECT ID FROM Scena
                    WHERE Descrizione={0} AND Atto={1}""".format(descrizione,atto))
    id=str(curr.fetchall())
    if id!="[]":
        print("ATTENZIONE: scena ({0}) già presente per l'atto con id ({1}),\
vuoi continuare lo stesso? (s/n)".format(descrizione,atto))
        cont=input()

    if cont=='s':
        try:
            curr.execute("""INSERT INTO Scena(Descrizione,Atto)
                            VALUES ({0},{1})""".format(descrizione,atto))
            conn.commit()
            print("Scena ({0}, {1}) inserita con successo".format(descrizione,atto))
        except psycopg2.errors.ForeignKeyViolation:
            print("ERRORE: Atto con id {0} non presente nel database,\
inserire prima l'atto".format(atto))
            curr.execute("ROLLBACK")
            conn.commit()
        except psycopg2.errors.NotNullViolation:
            print("ERRORE: Valori null non ammessi")
            curr.execute("ROLLBACK")
            conn.commit()
        input("Premere INVIO per continuare\n")
```

```
#metodo per l'inserimento di una logica
def insert_logica(nome,descrizione):
    if nome=='': nome='NULL'
    else: nome="'"+nome+"'"
    if descrizione=='': descrizione='NULL'
    else: descrizione="'"+descrizione+"'"
    try:
        curr.execute("""INSERT INTO Logica
                        VALUES ({0},{1})""".format(nome,descrizione))
        conn.commit()
        print("Logica ({0}, {1}) inserita con successo".format(nome,descrizione))
    except psycopg2.errors.UniqueViolation:
        print("ERRORE: Logica ({0},{1}) già presente nel database\
".format(nome,descrizione))
        curr.execute("ROLLBACK")
        conn.commit()
    except psycopg2.errors.NotNullViolation:
        print("ERRORE: Valori null non ammessi")
        curr.execute("ROLLBACK")
        conn.commit()
    input("Premere INVIO per continuare\n")

#metodo per l'inserimento di un dialogo
def insert_dialogo(speaker,destinatario,adjuncts,words,id_scena,logica):
    if speaker=='': speaker='NULL'
    else: speaker="'"+speaker+"'"
    if destinatario=='': destinatario='NULL'
    else: destinatario="'"+destinatario+"'"
    if adjuncts=='': adjuncts='NULL'
    else: adjuncts="'"+adjuncts+"'"
    if words=='': words='NULL'
    else: words="'"+words+"'"
    if id_scena=='': id_scena='NULL'
    else: id_scena="'"+id_scena+"'"
    if logica=='': logica='NULL'
    else: logica="'"+logica+"'"
    cont='s'
    curr.execute("""SELECT ID FROM Dialogo
                    WHERE Speaker={0} AND Destinatario={1} AND Adjuncts={2}
                          AND Words={3} AND Scena={4} AND Logica={5}
                 """.format(speaker,destinatario,adjuncts,words,id_scena,logica))
    id=str(curr.fetchall())
    if id!="[]":
        print("ATTENZIONE: dialogo già presente con questi dati\nSpeaker={0}\n\
Destinatario={1}\nAdjuncts={2}\nWords={3}\nID_Scena={4}\nLogica={5}\n \
vuoi continuare lo stesso? \
(s/n)".format(speaker,destinatario,adjuncts,words,id_scena,logica))
        cont=input()

    if cont=='s':
        try:
            curr.execute("""INSERT INTO Dialogo(Speaker,Destinatario,Adjuncts,Words,Scena,Logica)
                            VALUES ({0},{1},{2},{3},{4},{5})
                         """.format(speaker,destinatario,adjuncts,words,id_scena,logica))
            conn.commit()
            print("Dialogo con questi dati\nSpeaker={0}\nDestinatario={1}\nAdjuncts={2}\n\
Words={3}\nID_Scena={4}\nLogica={5}\ninserito con \
successo".format(speaker,destinatario,adjuncts,words,id_scena,logica))
        except psycopg2.errors.ForeignKeyViolation:
            print("ERRORE: Scena ({0}) o logica ({1}) non presente nel database, \
inserirli prima di inserire il dialogo".format(id_scena,logica))
            curr.execute("ROLLBACK")
            conn.commit()
        except psycopg2.errors.NotNullViolation:
            print("ERRORE: Valori null inseriti dove non ammessi")
            curr.execute("ROLLBACK")
            conn.commit()
        input("Premere INVIO per continuare\n")
```

5.3 insert_from_excel

Questa funzione inanzitutto si collega al database tramite la funzion *connect_DB()* dopodiché viene chiesto all'utente di inserire il nome del file compreso di estensione e il nome del foglio, se il file non viene trovato o il nome del foglio non esiste l'utente dovrà reinserire i dati, se esiste il foglio excel verrà salvato all'interno del programma in un DataFrame.
In seguito l'utente dovrà inserire il titolo e l'autore dell'opera, che verrano inserite nel database tramite la funzione *insert_opera.* Verrà quindi chiesto dove si trovano le logiche nel foglio elettronico, esse devono trovarsi al di sopra dei dialoghi, il numero di riga e colonna richiesti vanno contati non considerando le righe e le colonne completamente vuote. Le logiche devono seguire la seguente struttura "nome_logica: descrizione", deve essere scritto in una sola cella, infatti il nome e la descrizione della logica, letti dal file, vengono in seguito divisi tramite la funzione *split*, vengono tolti gli spazi tramite la funzione *strip* e vengono quindi inserite utilizzando la funzione *insert_logica.*
A questo punto si scorre il DataFrame fino alla sua fine, ad ogni riga vengono fatte le seguenti operazioni:

- se nella **prima colonna** si trova la stringa "ACT ", viene letto il suo valore e inserito nel database assieme al titolo dell'opera tramite la funzione *insert_atto*, successivamente viene richiesto al database l'ID dell'atto appena inserito per poterlo utilizzare in seguito, se si trova la stringa "ACT " nella prima colonna, le successive colonna saranno vuote, i dialoghi relativi a questo atto inizieranno dalla riga successiva;
- se nella **seconda colonna** si trova un valore non nullo esso sarà il nome della scena, che verrà inserito nel database tramite la funzione *insert_scena,* verrà quindi salvato l'ID della scena come per l'atto per poterlo utilizzare successivamente;
- se nella **terza colonna** si trova un valore non nullo significa che in questa riga è presente un dialogo, la sesta colonna che nel file corrisponde a "TOPIC" non viene considerata, vengono quindi letti dal DataFrame i valori dello *speaker* (colonna 3), del *destinatario* (colonna 4), della *logica* (colonna 5), di *adjuncts* (colonna da 7 a 9) e di *words* (colonna da 10 fino alla dimensione massima del DataFrame). Per *adjuncts* e *words* mano a mano che vengono letti i valori vengono salvati con la seguente struttura per permetterne l'inserimento nel database: "{val1, val2, val3, ecc}". Nella stringa *words* vengono sostituiti i le virgolette (') con le doppie virgolette singole ('') per non interferire con la struttura del comando SQL per l'inserimento. Viene quindi utilizzata la funzione insert_dialogo per inserire il dialogo, con tutti i valori letti dal DataFrame, nel database.

Nel caso si vogliano utilizzare altri fogli excel oltre a quello fornito devono rispettare le indicazioni fornite in precedenza, che riassumendo sono:

- logiche sopra ai dialoghi
- i dialoghi devono cominciare dalla prima colonna
- i nomi degli atti devono cominciare con “ACT ”, in modo da poter scrivere nella prima colonna altre informazioni secondarie
- dopo il nome dell’atto i dialoghi vanno inseriti dalla riga successiva

Di seguito il codice della funzione:

```
#inserimento dati da foglio excel
def insert_from_excel():
    #connesione al DB
    connect_DB()

    ok=False
    while not(ok):
        file=input("Inserire il nome del file\n")
        foglio=input("Inserire il nome del foglio all'interno del foglio elettronico\n")
        try:
            df=pd.read_excel(file,sheet_name=foglio)
            ok=True
        except FileNotFoundError:
            print("ERRORE: File non trovato")
        except ValueError:
            print("ERRORE: Foglio non trovato")

    titolo=input("Inserire titolo dell'opera\n")
    autore=input("Inserire autore dell'opera\n")
    insert_opera(titolo,autore)

    print("Inserimento LOGICA")
    riga=int(input("Inserire il numero della prima riga in cui si trovano le logiche,\
non contando le righe totalemente vuote\n"))
    colonna=int(input("Inserire il numero della colonna in cui si trovano le logiche,\
non contando le colonne totalemente vuote\n"))
    n=int(input("Inserire il numero di logiche presenti\n"))
    for i in range(n):
        logic=df.iat[riga+i,colonna]
        tokens=logic.split(":")
        insert_logica(tokens[0].strip(),tokens[1].strip())

    riga=df.shape[0]
    colonna=df.shape[1]
    att=0
    fine=False
    while not(fine):
        if "ACT " in str(df.iat[att+1,0]):
            fine=True
        att+=1

    id_atto=''
    id_scena=''

    while att<riga:
        if "ACT " in str(df.iat[att,0]):
            atto=df.iat[att,0]
            insert_atto(atto,titolo)
            att+=1
            curr.execute("""SELECT ID FROM Atto WHERE Descrizione='{0}' AND Opera='{1}'
                         """.format(atto,titolo))
            id_atto=str(curr.fetchone())
            id_atto=id_atto.strip('(')
            id_atto=id_atto.strip(',)')
```

```
    if str(df.iat[att,1])!='nan':
        insert_scena(df.iat[att,1],id_atto)
        curr.execute("""SELECT ID FROM Scena WHERE Descrizione='{0}' AND Atto='{1}'
                     """.format(df.iat[att,1],id_atto))
        id_scena=str(curr.fetchone())
        id_scena=id_scena.strip('(')
        id_scena=id_scena.strip(',)')
    if str(df.iat[att,2])!='nan':
        speaker=''
        destinatario=''
        adjuncts=''
        words=''
        logica=''
        for col in range(2,colonna):
            if col==2:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    speaker=tp
            elif col==3:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    destinatario=tp
            elif col==4:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    logica=tp.lower()
            elif col==6:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    adjuncts='{'+tp
            elif col==7:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    if adjuncts!='':
                        adjuncts=adjuncts+','+tp
                    else:
                        adjuncts='{'+tp
            elif col==8:
                tp=str(df.iat[att,col])
                if tp!='nan':
                    if adjuncts!='':
                        adjuncts=adjuncts+','+tp
                    else:
                        adjuncts='{'+tp
            elif (col>=9 and col<colonna):
                tp=str(df.iat[att,col])
                if tp!='nan':
                    if words!='':
                        words=words+','+tp
                    else:
                        words='{'+tp
        if adjuncts!='':
            adjuncts=adjuncts+'}'
        if words!='':
            words=words+'}'

        words=words.replace("'","''")
        insert_dialogo(speaker,destinatario,adjuncts,words,id_scena,logica)
    att+=1

curr.close()
conn.close()
```

5.4 manual_insert

Questa funzione si collega al database tramite la funzion *connect_DB()*, successivamente viene chiesto all'utente cosa vuole inserire, se viene inserite testo oppure un numero non compreso nel range 1-6 viene mostrato un messaggio di errore e chiesto all'utente di reinserire un numero.
L'utente può sceglire di inserire:

- un'**opera**, verrà quindi chiamata la funzione *manual_insert_opera*, la quale chiederà all'utente di inserire il titolo e l'autore dell'opera e verrà inserita chiamando la funzione *insert_opera*;
- un **atto**, verrà quindi chiamata la funzione *manual_insert_atto*, verrà chiesto di inserire una descrizione dell'atto e il tiolo dell'opera, l'atto verrà inserito nel database attraverso la funzione *insert_atto*;
- una **scena**, verrà quindi chiamata la funzione *manual_insert_scena*, l'utente dovrà inserire la descrizione della scena e l'ID dell'atto a cui si riferisce, la scena verrà quindi inserita chiamando la funzione *insert_scena*;
- una **logica**, verrà quindi chiamata la funzione *manual_insert_logica*, in modo analogo verrà chiesto all'utente di inserire il nome e la descrizione della logica che verrano inseriti nel database attraverso la funzione *insert_logica*;
- oppure un **dialogo**, verrà quindi chiamata la funzione *manual_insert_dialogo*, l'utente dovrà inserire il speaker, il destinatario, il numero di adjuncts e successivamente gli adjuncts stessi, il numero di words e gli words stessi, l'ID della scena a cui fa riferimento il dialogo e la logica del dialogo. Per permettere il corretto inserimento degli adjuncts e delle words vengono salvate dal programma rispettando la seguente struttura "{val1, val2, ecc.}". Il dialogo verrà quindi inserito nel database chiamando la funzione *insert_dialogo*.

Di seguito il codice delle funzioni:

```
#metodo per gestire l'inserimento manuale dei dati
def manual_insert():
    #connesione al DB
    connect_DB()

    fine=False
    while not(fine):
        print("Inserire il numero di relativo a ciò che si vuole inserire")
        print("1. Opera")
        print("2. Atto")
        print("3. Scena")
        print("4. Logica")
        print("5. Dialogo")
        print("")
        print("6. Per tornare indietro")
        try:
            tab = int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 6")
            continue
        if tab==1: manual_insert_opera()
        elif tab==2: manual_insert_atto()
        elif tab==3: manual_insert_scena()
        elif tab==4: manual_insert_logica()
        elif tab==5: manual_insert_dialogo()
        elif tab==6: fine=True
        else: print("ATTENZIONE! Inserire un numero compreso tra 1 e 6")
    curr.close()
    conn.close()

#metodo per l'inserimento manuale di un dialogo
def manual_insert_dialogo():
    speaker1=input("Inserire il nome dello speaker del dialogo\n")
    destinatario1=input("Inserire il nome del destinatario del dialogo\
(Opzioanale)\n")
    i=0
    while (i not in[1,2,3]):
        try:
            i=int(input("Inserire il numero di adjuncts che si vuole inserire\
(min 1, max 3)\n"))
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 3")
    adjuncts1="{"
    for j in range(i):
        if j>0 : adjuncts1 = adjuncts1+","
        adjuncts1=adjuncts1+input("Inserire adjunct\n")
    adjuncts1=adjuncts1+"}"

    i=0
    while (i not in range(1,21)):
        try:
            i=int(input("Inserire il numero di words che si vuole inserire\
(min 1, max 20)\n"))
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 20")
    words1="{"
    for j in range(i):
        if j>0: words1=words1+","
        words1=words1+input("Inserire word\n")
    words1=words1+"}"

    scena1=input("Inserire ID della scena relativa al dialogo\n")
    logica1=input("Inserire nome della logica del dialogo\n")

    insert_dialogo(speaker1,destinatario1,adjuncts1,words1,scena1,logica1)
```

```
#metodo per l'inserimento manuale di un'opera
def manual_insert_opera():
    titolo1=input("Inserire nome dell'opera\n")
    autore1=input("Inserire autore dell'opera\n")
    insert_opera(titolo1,autore1)

#metodo per l'inserimento manuale di un'atto
def manual_insert_atto():
    descrizione1=input("Inserire descrizione dell'atto\n")
    opera1=input("Inserire il titolo dell'opera a cui si riferisce\n")
    insert_atto(descrizione1,opera1)

#metodo per l'inserimento manuale di una scena
def manual_insert_scena():
    descrizione1=input("Inserire descrizione della scena\n")
    atto1=input("Inserire l'ID dell'atto a cui si riferisce\n")
    insert_scena(descrizione1,atto1)

#metodo per l'inserimento manuale di una logica
def manual_insert_logica():
    nome1=input("Inserire il nome della logica\n")
    descrizione1=input("Inserire la descrizione della logica\n")
    insert_logica(nome1,descrizione1)
```

5.5 visualize_data

La funzione *visualize_data* si connette al database tramite la funzion connect_DB(), successivamente viene chiesto all'utente cosa vuole stampare, se viene inserite testo oppure un numero non compreso nel range 1-6 viene mostrato un messaggio di errore e chiesto all'utente di reinserire un numero.

L'utente può scegliere di stampare le opere, gli atti, le scene, le logiche oppure i dialoghi.

```
#stampa dati
def visualize_data():
    #connesione al DB
    connect_DB()

    stop=False
    while not(stop):
        print("Inserire il numero di ciò che si vuole stampare")
        print("1. Stampa opere")
        print("2. Stampa atti")
        print("3. Stampa scene")
        print("4. Stampa logiche")
        print("5. Stampa dialoghi")
        print("")
        print("6. Per tornare indietro")
        try:
            op=int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 6")
            continue
        if op==1:
            stampa_opere()
        elif op==2:
            stampa_atti()
        elif op==3:
            stampa_scene()
        elif op==4:
            stampa_logiche()
        elif op==5:
            stampa_dialoghi()
        elif op==6:
            stop=True
        else:
            print("ATTENZIONE! Inserire un numero compreso tra 1 e 6")
    curr.close()
    conn.close()
```

Andiamo a vedere una ad una le funzioni: *stampa_opere*, *stampa_atti*, *stampa_scene*, *stampa_logiche*, *stampa_dialoghi*.

5.5.1 stampa_opere

L'utente può scegliere se:

- stampare tutte le opere ragruppate per autore, in questo caso basta fare una proiezione sulla tabella Opera ordinando i risultati in base all'autore;
- stampare le opere di un autore, in questo caso bisognerà fare una selezione sulla tabella Opera delle tuple dove l'autore è quello richiesto.

Verrà inoltre stampato il numero di record ottenuti, per evitare di fare un ulteriore interrogazione al database i record vengono contati man a mano che vengono stampati.

```
#stampa opere
def stampa_opere():
    stop=False
    while not(stop):
        print("Inserire il numero di ciò che si vuole stampare")
        print("1. Stampa tutte le opere raggruppate per autore")
        print("2. Stampa le opere di un solo autore scelto")
        print("")
        print("3. Per tornare indietro")
        try:
            op=int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 3")
        if op==1:
            curr.execute("""SELECT * FROM Opera ORDER BY Autore""")
            print("TITOLO  AUTORE")
            n=0
            for record in curr:
                print(record)
                n=+1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==2:
            print("Inserire il nome dell'autore")
            autore=str(input())
            curr.execute("""SELECT * FROM Opera WHERE Autore='{0}'""".format(autore))
            print("TITOLO  AUTORE")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==3:
            stop=True
        else:
            print("ATTENZIONE! Inserire un numero compreso tra 1 e 3")
```

5.5.2 stampa_atti

L'utente può scegliere se:

- stampare tutte gli atti ragruppati per opera, in questo caso basta fare una proiezione sulla tabella Atto ordinando i risultati in base all'opera;
- stampare gli atti solo di un'opera, l'operazione da fare è una selezione sulla tabella Atto delle tuple dove l'opera è quella scelta.

Verrà inoltre stampato il numero di record, ottenuto con lo stesso metodo utilizzato in precedenza.

```
#stampa atti
def stampa_atti():
    stop=False
    while not(stop):
        print("Inserire il numero di ciò che si vuole stampare")
        print("1. Stampa tutte gli atti raggruppati per opera")
        print("2. Stampa gli atti di una sola opera scelta")
        print("")
        print("3. Per tornare indietro")
        try:
            op=int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 3")
            continue
        if op==1:
            curr.execute("""SELECT * FROM Atto ORDER BY Opera""")
            print("ID  DESCRIZIONE  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==2:
            print("Inserire il nome dell'opera")
            opera=str(input())
            curr.execute("""SELECT * FROM Atto WHERE opera='{0}'""".format(opera))
            print("ID  DESCRIZIONE  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==3:
            stop=True
        else:
            print("ATTENZIONE! Inserire un numero compreso tra 1 e 3")
```

5.5.3 stampa_scene

L'utente può scegliere tra le seguenti operazioni:

- stampare tutte le scene ordinate per opera e a parità di opera per atti, in questo caso viene fatta una proiezione sulla tabella risultante dal LEFT JOIN tra la tabella Scena e la tabella Atto sul attributo Scena.atto=Atto.id, ordinando i risultati in base all'opera e a parità di opera in base all'atto;
- stampare le scene solo di un atto, l'operazione da fare è una proiezione sulla tabella risultante dal LEFT JOIN tra la tabella Scena e la tabella Atto sul attributo Scena.atto=Atto.id, selezionando le tuple dove l'atto è quello scelto dall'utente;
- stampare tutte le scene di un'opera, anche in questo caso l'operazione svolta è la proiezione sulla tabella risultante dal LEFT JOIN tra la tabella Scena e la tabella Atto sul attributo Scena.atto=Atto.id, selezionando solo le tuple dove l'opera è quella inserita dall'utente, i risultati vengono ordinati in base all'atto.

Verrà inoltre stampato il numero di record, ottenuto con lo stesso metodo utilizzato in precedenza.

```
#stampa scene
def stampa_scene():
    stop=False
    while not(stop):
        print("Inserire il numero di ciò che si vuole stampare")
        print("1. Stampa tutte le scene raggruppati per atto,opera")
        print("2. Stampa le scene di un atto")
        print("3. Stampa le scene di un opera")
        print("")
        print("4. Per tornare indietro")
        try:
            op=int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 4")
            continue
        if op==1:
            curr.execute("""SELECT Scena.id,Scena.descrizione,Atto.descrizione,
                                   Atto.opera
                            FROM Scena LEFT JOIN Atto ON Scena.atto=Atto.id
                            ORDER BY Atto.opera,Atto.descrizione""")
            print("ID  DESCRIZIONE_SCENA DESCRIZIONE_ATTO  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
```

```
    elif op==2:
        print("Inserire l'ID dell'atto")
        try:
            id_atto=int(input())
        except:
            print("ERRORE")
            continue
        curr.execute("""SELECT Scena.id,Scena.descrizione,Scena.atto,
                               Atto.descrizione,Atto.opera
                        FROM Scena LEFT JOIN Atto ON Scena.atto=Atto.id
                        WHERE Scena.atto='{0}'""".format(id_atto))
        print("ID  DESCRIZIONE_SCENA ID_ATTO DESCRIZIONE_ATTO  OPERA")
        n=0
        for record in curr:
            print(record)
            n+=1
        conn.commit()
        print("Numero record: {0}".format(n))
        input("Premere INVIO per continuare")
    elif op==3:
        print("Inserire titolo dell'opera")
        opera=str(input())
        curr.execute("""SELECT Scena.id,Scena.descrizione,Atto.descrizione,
                        Atto.opera
                        FROM Scena LEFT JOIN Atto ON Scena.atto=Atto.id
                        WHERE Atto.opera='{0}'
                        ORDER BY Atto.descrizione""".format(opera))
        print("ID  DESCRIZIONE_SCENA ID_ATTO DESCRIZIONE_ATTO  OPERA")
        n=0
        for record in curr:
            print(record)
            n+=1
        conn.commit()
        print("Numero record: {0}".format(n))
        input("Premere INVIO per continuare")
    elif op==4:
        stop=True
    else:
        print("ATTENZIONE! Inserire un numero compreso tra 1 e 4")
```

5.5.4 stampa_logiche

Questa funzione stampa direttamente tutte le logiche, facendo un'operazione di proiezione sulla tabella Logica.

Verrà inoltre stampato il numero di record, ottenuto con lo stesso metodo utilizzato in precedenza.

```
#stampa logiche
def stampa_logiche():
    curr.execute("""SELECT * FROM Logica""")
    print("NOME  DESCRIZIONE")
    n=0
    for record in curr:
        print(record)
        n+=1
    conn.commit()
    print("Numero record: {0}".format(n))
    input("Premere INVIO per continuare")
```

5.5.5 stampa_dialoghi

L'utente può scegliere tra le seguenti operazioni:

- stampare tutti i dialoghi ordinati per opera, atto, scena, in questo caso viene fatta una proiezione sulla tabella risultante dal LEFT JOIN tra la tabella Dialogo e la tabella Scena a cui viene fatto un secondo LEFT JOIN con la tabella Atto, ordinando i risultati in base all'opera, a parità di opera in base all'atto, a parità di atto in base alla scena e a parità di scena in base all'ID del dialogo;
- stampare i dialoghi scegliendo se visualizzare solo quelli relativi a uno speaker, un destinatario, una scena, un atto o un'opera, l'operazione da fare in questo caso è la stessa proiezione fatta nel punto precedente a cui bisogna applicara un operazione di selezione del speaker, del destinatario, della scena, dell'atto o dell'opera inserita dall'utente

Verrà inoltre stampato il numero di record, ottenuto con lo stesso metodo utilizzato in precedenza.

```
#stampa dialoghi
def stampa_dialoghi():
    stop=False
    while not(stop):
        print("Inserire il numero di ciò che si vuole stampare")
        print("1. Stampa tutti i dialoghi raggruppati per scena,atto,opera")
        print("2. Stampa i dialoghi relativi au uno speaker")
        print("3. Stampa i dialoghi relativi ad un destinatario")
        print("4. Stampa i dialoghi relativi ad una scena")
        print("5. Stampa i dialoghi relativi ad un atto")
        print("6. Stampa i dialoghi relativi ad un'opera")
        print("")
        print("7. Per tornare indietro")
        try:
            op=int(input())
        except:
            print("ERRORE: Inserire un numero compreso tra 1 e 7")
            continue
        if op==1:
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.descrizione,Atto.descrizione,Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""")
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA  SCENA  ATTO\
OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
```

```
        elif op==2:
            print("Inserire il nome dello speaker")
            speaker=str(input())
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.descrizione,Atto.descrizione,Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            WHERE Dialogo.speaker='{0}'
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""".format(speaker))
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA  SCENA  ATTO\
OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==3:
            print("Inserire il nome del destinatario")
            destinatario=str(input())
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.descrizione,Atto.descrizione,Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            WHERE Dialogo.destinatario='{0}'
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""".format(destinatario))
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA  SCENA  ATTO\
OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==4:
            print("Inserire l'id della scena")
            try:
                id_scena=int(input())
            except:
                print("ERRORE")
                continue
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.id,Scena.descrizione,Atto.descrizione,
                                   Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            WHERE Scena.id='{0}'
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""".format(id_scena))
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA ID_SCENA\
SCENA  ATTO  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
```

```
        elif op==5:
            print("Inserire l'id dell'atto")
            try:
                id_atto=int(input())
            except:
                print("ERRORE")
                continue
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.descrizione,Atto.id,Atto.descrizione,
                                   Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            WHERE Atto.id='{0}'
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""".format(id_atto))
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA  SCENA\
ID_ATTO  ATTO  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==6:
            print("Inserire il titolo dell'opera")
            opera=str(input())
            curr.execute("""SELECT Dialogo.id,Dialogo.speaker,Dialogo.destinatario,
                                   Dialogo.adjuncts,Dialogo.words,Dialogo.logica,
                                   Scena.descrizione,Atto.descrizione,Atto.opera
                            FROM Dialogo LEFT JOIN Scena on Dialogo.scena=Scena.id
                                         LEFT JOIN Atto on Scena.atto=Atto.id
                            WHERE Atto.opera='{0}'
                            ORDER BY Atto.Opera,Atto.descrizione,Scena.descrizione,
                                     Dialogo.id""".format(opera))
            print("ID  SPEAKER  DESTINATARIO  ADJUNCTS  WORDS  LOGICA  SCENA\
ATTO  OPERA")
            n=0
            for record in curr:
                print(record)
                n+=1
            conn.commit()
            print("Numero record: {0}".format(n))
            input("Premere INVIO per continuare")
        elif op==7:
            stop=True
        else:
            print("ATTENZIONE! Inserire un numero compreso tra 1 e 7")
```

6. Conclusioni

Lo scopo di questa tesi è stato quello di progettare una base di dati per raccogliere i dati provenienti da tragedie teatrali e in seguito sviluppare un software che permetta di creare la base di dati, inserire i dati raccolti dal foglio excel nel database e visualizzare i dati inseriti.
Per realizzare il programma è stata fatta la scelta di utilizzare Python e la libreria Pandas visto che sono strumenti molto potenti e allo stesso abbastanza facile da usare per manipolare grandi quantità di dati.
Il programma realizzato permette inoltre di inserire eventuali dati anche manualmente, nel caso non si voglia più utilizzare un foglio excel per raccogliere i dati, e visualizzare i dati raccolti in vari modi, permettendo così all'utente di visualizzare solo i dati che lo interessano in quel momento.
Un possibile sviluppo futuro del progetto potrebbe essere quello di permettere all'utente di eliminare i dati, facendo atttenzione ai vincoli di integrità. Altri sviluppi potrebbero riguardare l'analisi dei dati presenti nel database con la generazione di varie statistiche.

7. Bibliografia

[1] Paolo Atzeni, Stefano Ceri, Piero Fraternali, Stefano Paraboschi, Riccardo Torlone *Basi di dati* Milano: McGraw-Hill 2018

[2] G.M. Di Nunzio, E. Di Buccio *Basi di dati Progettazione concettuale, logica e sql* Bologna: Esculapio 2017

[3] Python Documentation: https://docs.python.org/3.11/

[4] PostgreSQL Documentation: https://www.postgresql.org/docs/15/index.html

[5] Psycopg2 Documentation: https://www.psycopg.org/docs/

[6] Pandas Documentation: https://pandas.pydata.org/docs/

[7] IDLE Documentation: https://docs.python.org/3/library/idle.html

[8] Text mining: https://www.ibm.com/it-it/topics/text-mining [Data di accesso: 11/2023]

[9] Differenze tra data mining e text mining: https://vitolavecchia.altervista.org/caratteristiche-e-differenza-tra-text-mining-e-data-mining-in-informatica/ [Data di accesso: 11/2023]

[10] Data visualization: https://www.tableau.com/learn/articles/data-visualization [Data di accesso: 11/2023]

[11] Data visualization: https://www.coursera.org/articles/data-visualization [Data di accesso: 11/2023]