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 semplificazione del testo: Il progetto SimpleText"

Relatore: Prof. Giorgio Maria Di Nunzio

Laureanda: Francesca Cipolla

Correlatore: Dott. Stefano Marchesin

ANNO ACCADEMICO 2021 – 2022

Data di laurea 19/09/2022

A mia madre,
per la fame di conoscenza
che ha saputo trasmettermi.

Ringraziamenti

Vorrei ringraziare il Professor Giorgio Maria Di Nunzio e il Dottor Stefano Marchesin per avermi seguito in quest'ultima sfida lungo il mio percorso.

Un ringraziamento va anche alle impiegate della Segreteria Didattica del DEI per aver sempre risposto prontamente e con estrema gentilezza ad ogni mia richiesta.

Ringrazio infine tutta la mia famiglia, specie papà Fabrizio e Silvia per avermi sostenuto in tutti questi anni, e Sofia per essermi stata affianco e avermi sopportato lungo tutto il mio percorso, seppur accidentato.

Grazie nonni e zii, e soprattutto grazie mamma, spero che possiate vedere tutto questo e che siate orgogliosi di me.

Indice

1. Introduzione

Al giorno d'oggi avere una solida base scientifica è di fondamentale importanza per pensare in modo critico, prendere decisioni informate sulla propria salute e poter distinguere i fatti dalla finzione. La pandemia da Covid-19 ha impartito una dura lezione su quanto sia importante per le persone comprendere ciò di cui la scienza parla: prevenzione, distanziamento sociale, politiche sanitarie. Tuttavia i testi scientifici, destinati ai professionisti del settore, sono spesso di difficile comprensione per il pubblico generale poiché utilizzano una terminologia molto specifica e presuppongono una preparazione specialistica nel lettore. Per questi motivi i "non-esperti" tendono ad affidarsi a fonti maggiormente comprensibili, come il Web e i Social Network, imbattendosi in tal modo in informazioni spesso incorrette o non veritiere.

In questo contesto si inserisce il progetto SimpleText, un laboratorio organizzato nell'ambito della conferenza CLEF-2022, che mira a ridurre il divario tra le pubblicazioni scientifiche e le persone attraverso la semplificazione del testo in modo automatico. L'obiettivo del lab è di generare riepiloghi semplificati di più testi scientifici sulla base di una determinata query per restituire all'utente un sommario semplificato dell'argomento a cui è interessato, al fine di promuovere l'accesso alla informazione scientifica.

Questo elaborato mira a progettare ed implementare un database che modelli uno dei datasets messi a disposizione dal laboratorio SimpleText. Il lavoro si suddivide in due fasi strettamente interconnesse e spesso sovrapposte: la prima volta all'analisi e all'elaborazione del dataset, la seconda relativa alla progettazione, implementazione e popolamento del database.
L'analisi del dataset inizia con l'osservazione dei dati a cui segue un processo di analisi ed elaborazione sempre più approfondita attraverso degli scripts appositamente sviluppati e che fanno ampio uso delle strutture dati e dei metodi della libreria Pandas.
La progettazione del database viene sviluppata sulla base dell'analisi delle fasi precedenti, mentre le inconsistenze ritrovate all'interno del dataset, dopo un'attenta valutazione, vengono tradotte in determinate scelte progettuali durante l'implementazione del database.
Infine il database viene popolato grazie ad uno script che fa uso di SQLAlchemy.
.

2. Strumenti utilizzati

In questa sezione vengono brevemente elencati i principali strumenti utilizzati per lo sviluppo del progetto.

2.1 Python

Python è un linguaggio di programmazione di alto livello, ampiamente utilizzato nell'ambito dell'analisi dei dati. Di seguito un elenco dei punti di forza del linguaggio:

- È considerato un linguaggio **interpretato**: in realtà un interprete si occupa di analizzare e pre-compilare automaticamente il codice sorgente in bytecode prima di eseguirlo;
- È un linguaggio **multi-paradigma**: supporta diversi paradigmi tra cui la programmazione procedurale e la programmazione ad oggetti;
- È **portabile**: è possibile utilizzarlo su diverse piattaforme purché si abbia l'interprete Python installato;
- È **free**: è completamente gratuito sia nell'utilizzo che nella distribuzione, nonostante questo vanta una community molto attiva che si occupa di migliorarlo e mantenerlo costantemente aggiornato;
- È **ricco di librerie**: oltre alla Standard Library, il Python Package Index permette di installare moduli aggiuntivi creati e mantenuti dalla community;
- **Gestione automatica della memoria**: un sistema di garbage collector si occupa di allocare e rilasciare la memoria automaticamente;
- È f**acile da usare**: grazie alla tipizzazione dinamica forte eseguita a runtime e all'utilizzo dell'indentazione per la sintassi al posto delle parentesi.

La versione di Python utilizzata è la 3.9.12.

2.2 Pandas

Pandas è una libreria software di Python per l'analisi e la manipolazione di dati in formato sequenziale o tabellare altamente performante. Le strutture dati principali in Pandas sono:

- Series: Strutture unidimensionali etichettate con degli index;

- DataFrame: Strutture bidimensionali che contengono Series di dati eterogenei etichettate con degli index (per le righe) e con delle columns (per le colonne).

Panda offre molte funzionalità per l'analisi e l'elaborazione dei dati, tra le quali:

- Gestione dei dati mancanti;
- Lettura e scrittura di file tipo Json, CSV, basi di dati e molto altro;
- Modifica dei DataFrame tramite eliminazione ed inserimento di colonne;
- Possibilità di ordinare i DataFrame;
- Funzionalità SQL like;
- Semplicità nella visualizzazione dei risultati delle operazioni.

La libreria Pandas rappresenta il cuore del progetto essendo stata ampiamente utilizzata negli script per analizzare, manipolare ed esportare i dati del dataset.
La versione di Pandas utilizzata è la 1.4.2.

2.3 Anaconda

Anaconda è distribuzione open source dei linguaggi Python e R per il calcolo scientifico, la sua popolarità è dovuta alla semplicità di configurazione di un ambiente di sviluppo in Python: oltre all'installer di Python prevede anche molti pacchetti già installati e pronti all'uso tra cui: Pandas, Jupiter, Numpy, SQLAlchemy. Per il progetto è stato fatto uso dell'interfaccia utente grafica Anaconda-Navigator.
La versione di Anaconda-Navigator utilizzata è la 2.2.0.

2.4 JupyterLab

JupyterLab è un ambiente di sviluppo web interattivo basato sui Notebook Jupyter, un'applicazione web che permette di creare documenti testuali interattivi contenenti grafici, testo, codice sorgente eseguibile ed equazioni. Il progetto è interamente svolto all'interno di JupyterLab.
La versione di JupiterLab utilizzata è la 3.3.2.

2.5 SQLAlchemy

SQL Alchemy è un toolkit SQL open source e un ORM (Object Relational Mapper) per Python, permette di trasferire i dati archiviati in un database SQL in oggetti Python e viceversa, rende quindi possibile utilizzare codice Python per creare, leggere, aggiornare e cancellare dati senza la necessità di fare esplicitamente uso di codice SQL.
La versione di SQLAlchemy utilizzata è la 1.4.27.

2.6 PostgreSQL

PostgreSQL è un DBMS (DataBase Management System) relazionale ad oggetti open source, conforme allo standard SQL, noto per la sua affidabilità, robustezza, flessibilità e prestazioni elevate, altamente scalabile sia nella quantità di dati che è in grado di memorizzare, sia nella quantità di utenti simultanei che è in grado di ospitare. PostgreSQL utilizza il modello Client/ Server: il processo Server si occupa di gestire i file della base di dati, accettare le connessioni ed eseguire operazioni richieste dal client sulla base di dati, il processo Client può essere un'applicazione che l'utente utilizza per eseguire operazioni sulla base di dati. PostgreSQL può gestire connessioni multiple e concorrenti e Client e Server possono risiedere su host distinti.
La versione di PostgreSQL utilizzata è la 14.4.

2.7 pgAdmin

PgAdmin è un'interfaccia grafica open source che permette di amministrare i database di PostgreSQL. Alcune delle sue funzionalità sono: Query tools; visualizzazione, inserimento e modifica dei dati in modalità tabellare; visualizzazione di proprietà, definizioni SQL e vincoli degli oggetti memorizzati nella base di dati.
La versione di pgAdmin IV utilizzata è la 6.10.

3. Descrizione del dataset

Il dataset utilizzato (dblp1.json) viene fornito previa iscrizione al lab SimpleText. Il dataset è relativo al Citation Network Dataset, è organizzato in un file Json e la sua dimensione è di 10.422.742.820 byte. Il Citation Network Dataset contiene citazioni di pubblicazioni scientifiche estratte da varie fonti: DBLP (sito web che raccoglie risorse bibliografiche nell'ambito informatico), ACM (banca dati per le pubblicazioni dell'Association Computing Machinery) e Microsoft Academic Graph (che consente l'accesso a dati archiviati contenenti records di pubblicazioni scientifiche, relazioni di citazioni, autori, istituzioni, riviste, conferenze e ambiti di studio), sostanzialmente si tratta di un dataset che raccoglie una serie di riferimenti bibliografici e relazioni di citazione relativi alle pubblicazioni scientifiche, siano esse articoli scientifici, libri o atti di conferenze. Lo schema dei dati è fornito al seguente link [1], vengono mostrate solo le proprietà di interesse per la versione del dataset utilizzata (v12):

Field Name	**Field Type**	**Description**	**Example**
id	Long	paper ID	339090091
title	string	paper title	Data mining: concepts and techniques
authors.name	string	author name	Jiawei Han
author.org	string	author affiliation	Department of Computer Science, University of Illinois at Urbana-Champaign
author.id	Long	author ID	2052936527
venue.id	Long	paper venue ID	2595095599
venue.raw	string	paper venue name	Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial
year	int	published year	2000
fos.name	string	paper fields of study	Web mining
fos.w	float	fields of study weight	0.659690857
references	list of strings	paper references	["4909282", "16018031", "16159250", "19838944", ...]
n_citation	int	citation number	40829
page_start	string	page start	11
page_end	string	page end	18
doc_type	string	paper type: journal, book title...	book
publisher	string	publisher	Elsevier
volume	string	volume	10
issue	string	issue	29
doi	string	doi	10.4114/ia.v10i29.873
abstract	string	abstract	Our ability to generate...

Tabella 3.1

Una prima osservazione del file dblp1.json permette di individuarne le proprietà principali e di confrontarle con quelle fornite nello schema precedente.


```
{"_index": "dblp1",
"_type": "_doc",
"_id": "339090091",
"_score": 1,
"_source": {
    "id": 339090091,
    "authors": [
        {
            "name": "Uditha Ratnayake",
            "id": 2439017310
        },
        {
            "name": "TamÃis D. Gedeon",
            "id": 2052936527
        },
        {
            "name": "Nalin Wickramarachchi",
            "id": 2693152212
        }
    ],
    "title": "Document Classification with Recommendation Architecture:
             Extensions for Feature Intensity Recognition and Column Labeling.",
    "year": 2002,
    "n_citation": 0,
    "page_start": "",
    "page_end": "",
    "doc_type": "Conference",
    "publisher": "",
    "volume": "",
    "issue": "",
    "doi": "",
    "fos": [
        {
            "name": "Document classification",
            "w": 0.69468
        },
        {
            "name": "Data mining",
            "w": 0.45555
        },
        {
            "name": "Architecture",
            "w": 0.44832
        },
        {
            "name": "Information retrieval",
            "w": 0.46439
        },
        {
            "name": "Computer science",
```

```
            "w": 0.43309
        }
    ],
    "venue": {
        "raw": "Australasian Document Computing Symposium",
        "id": 1147924868,
        "type": "C"
    },
    "abstract": "",
    "references": "",
    "nb_references": 0,
    "author": {
        "name": "",
        "id": "",
        "org": ""
    }
}
}
```


Il file è costituito da una serie di oggetti Json (JavaScript Object Notation), noto formato per lo scambio di dati, in cui ogni linea costituisce un dictionary di pubblicazioni.
Le proprietà di interesse sono quelle racchiuse all'interno della chiave "_source" che a sua volta contiene un oggetto Json con diverse coppie chiave/valore e quattro livelli annidati, ossia "authors", "fos", "venue" e "author". È possibile riformulare lo schema dei dati nel modo seguente assieme ai tipi di dato dei relativi valori:

- id: codice identificativo della pubblicazione (valore intero);
- authors: questa chiave contiene un array di oggetti con le informazioni sugli autori della pubblicazione, rispettivamente name e id (valori di tipo stringa ed intero);
- title: titolo della pubblicazione (valore di tipo stringa);
- year: anno di pubblicazione dell'articolo (valore intero);
- n_citation: numero di citazioni (valore intero);
- page_start: pagina iniziale (valore di tipo stringa);
- page_end: pagina finale (valore di tipo stringa);
- doc_type: tipologia del documento (valore di tipo stringa);
- publisher: editore (valore di tipo stringa);
- volume: volume in cui è inserita la pubblicazione (valore di tipo stringa);
- issue: edizione (valore di tipo stringa);
- doi: codice identificativo univoco per documenti digitali (valore di tipo stringa);
- fos: questa chiave contiene un array di oggetti con le informazioni sugli ambiti di studio della pubblicazione rispettivamente name e w (valori di tipo stringa e float);

- venue: contiene un oggetto con coppie chiave/valore riguardanti le informazioni circa la sede di competenza della pubblicazione, rispettivamente id, raw e type (valori intero il primo e gli altri di tipo stringa);
- abstract: breve riepilogo della pubblicazione (valore di tipo stringa);
- references: secondo lo schema precedente è un array di valori rappresentanti i codici identificativi degli articoli a cui la pubblicazione fa riferimento (lista di stringhe);
- nb_references: numero di articoli referenziati dalla pubblicazione (valore intero);
- author: contiene un oggetto con coppie chiave/valore riportanti informazioni sull'autore, rispettivamente id, name e org (con valori intero il primo e gli altri di tipo stringa, secondo lo schema precedente).

4. Il progetto: DBLP_dataset_to_DB

Data la dimensione del file si è resa necessaria la realizzazione di un programma ad hoc per estrapolare tutte le proprietà di interesse e poter quindi eseguire un'analisi più approfondita.
La scelta di utilizzare la libreria Pandas è stata dettata dalla sua efficienza in termini di performance nel manipolare big data e dal fatto che i DataFrame, data la loro struttura tabellare, ben si adattano a rappresentare i dati di interesse.
Il progetto DBLP_dataset_to_DB, che prevede l'estrazione dei dati dal dataset, la loro elaborazione e la loro successiva scrittura nelle tabelle del database, è suddiviso in diversi scripts che raccolgono le funzioni comuni, un file di esecuzione principale e un file contenente il codice SQL per creare il database:

- import_dataset.py: script che raccoglie funzioni atte ad importare i dati del file dblp1.json nei diversi DataFrame;
- analize_df.py: script con funzioni che analizzano i DataFrame creati;
- manipulate_df.py; script contenente funzioni che manipolano i DataFrame creati;
- DB_connection: script che si occupa di creare la connessione ad un database esistente, scrivere gli elementi dei DataFrame nelle tabelle del database ed interrogarlo al fine di verificare la corretta importazione di tutti i records;
- config.py: file contenente i parametri di connessione al database sotto forma di dict;
- Json_Parser_DBLP_Dataset: file di esecuzione principale, dove vengono eseguite le funzioni contenute negli scripts, con l'aggiunta di qualche raro utilizzo diretto dei metodi di Pandas;
- DBLP_db: file contenente lo schema del database.

Per brevità non verrà illustrato tutto il codice sorgente, il programma completo può essere trovato al seguente link GitHub.
L'analisi del codice sorgente non segue il flusso del file Json_Parser_DBLP_Dataset, si è reso infatti necessario suddividere l'esecuzione del file in tre parti (una per ogni DataFrame "padre") a causa delle limitate risorse del calcolatore su cui è stato svolto il progetto.

Per prima cosa è necessario capire di quanti dati si sta parlando, path e data rappresentano rispettivamente il percorso al file dblp1.json e la chiave "_source" che ingloba le proprietà di interesse.

```
path='/Users/francescacipolla/Desktop/dblp1.json'
data='_source'

count_lines=sum(1 for line in open(path))
count_lines
```

```
4894063
```

Il file si compone di ben 4.894.063 oggetti Json, rappresentanti le diverse pubblicazioni scientifiche.

4.1 Importazione

In questa parte viene analizzato lo script import_dataset.py contenente le funzioni utilizzate per importare gli elementi del Json nei vari DataFrame, l'operazione di importazione verrà eseguita più volte: per il livello base (quello contenuto all'interno della chiave "_source") e per i livelli annidati del Json contenenti a loro volta oggetti (chiavi "authors" e "fos").

4.1.1 paper_df

Per prima cosa viene inizializzato un DataFrame vuoto.

```
def initialize():
    return pd.DataFrame()
```

Dato un DataFrame, il percorso al file Json, la chiave di interesse ("_source" nello specifico) e una lista di chiavi , la seguente funzione legge il file dblp1.json tramite il metodo Pandas *read_json*, lo legge come un oggetto Json per linea suddividendolo in chunks, per non incorrere nell'errore OutOfMemory. Successivamente itera sui chunks e chiama la funzione read_json_chunk, elimina le colonne corrispondenti alle chiavi passate (parametro columns_list) che, per il momento, non sono necessarie e restituisce il DataFrame finale ottenuto tramite concatenazione in modo iterativo.

```
def read_json (df,path,data,columns_list):
    with open (path) as f:
        chunks=pd.read_json(path_or_buf=f, lines=True, chunksize=10000)
        for chunk in chunks:
            source_df=read_json_chunk(chunk,data)
            source_df.drop(columns=columns_list, inplace=True)
            df=pd.concat(objs=[df,source_df], ignore_index=True)
        return df
```

Dato un oggetto JsonReader (i chunks) e la chiave di interesse (data) è possibile recuperare i dati all'interno di una proprietà dell'oggetto Json tramite il metodo Pandas *json_normalize*, aggiungendo, in modo opzionale, la chiave di un sotto livello (recordpath) per andare più a fondo nella gerarchia e dei metadati (meta) da includere nei risultati.

```
def read_json_chunk (chunk, data, recordpath=None, meta=None, metaprefix=None,
                     recordprefix=None):
      if (recordpath==None) and (meta==None) and (metaprefix==None) and
            (recordprefix==None):
        df=(pd.json_normalize(data=chunk[data]))
    else:
        df= (pd.json_normalize(data=chunk[data], record_path=[recordpath],
                  meta=[meta], meta_prefix=metaprefix,
                  record_prefix=recordprefix))
    return df
```

Di seguito la chiamata della funzione, nel file di esecuzione principale, per creare il DataFrame paper_df, le proprietà “authors” e “fos” dei sotto livelli vengono eliminate in quanto saranno recuperate con metodi diversi.

```
paper_df=read_json(paper_df,path,data,['authors','fos'])
```

4.1.2 author_df

Dopo l’inizializzazione di un DataFrame vuoto, viene chiamata la seguente funzione per importare la proprietà “authors”, si comporta come la funzione read_json esposta precedentemente con la differenza che ora vengono passati , al posto di columns_list, i parametri recordpath (essendo la proprietà “authors” un sotto livello nella gerarchia del file Json) , meta (per mantenere nei risultati la proprietà di un livello più esterno, in questo caso id) ed i relativi parametri (recordprefix e metaprefix) per impostare i prefissi nelle colonne del DataFrame.

```
def read_json_flatten (df,path,data,recordpath,meta,metaprefix,recordprefix):
    with open (path) as f:
        chunks=pd.read_json(path_or_buf=f,lines=True,chunksize=10000)
        for chunk in chunks:
            source_df=read_json_chunk(chunk,data,recordpath,
                        meta,metaprefix,recordprefix)
            df=pd.concat(objs=[df,source_df], ignore_index=True)
        return df
```

Di seguito la chiamata della funzione, nel file di esecuzione principale, per creare il DataFrame author_df.

```
author_df=read_json_flatten(author_df,path,data,'authors','id','paper_','author
                                _')
```

4.1.3 fos_df

Estrarre i dati per il DataFrame fos_df è stato più complicato in quanto il metodo *json_normalize* lavora su liste di oggetti, mentre la chiave "fos", come evidenziato dall'esempio sottostante, in certe parti del dataset rappresenta un oggetto con coppie chiave/valore e non un array di oggetti, portando all'errore TypeError.

```
TypeError: {'id': 343712712, 'authors': [{'name': 'Nadia Mesli', 'id': 2395865256}], 'title': "Bases de donnees et lexiques-grammaires: un exemple d'exploitatio
n linguistique de l'outil informatique.", 'year': 2002, 'n_citation': 0, 'page_start': '5', 'page_end': '22', 'doc_type': '', 'publisher': '', 'volume': '61',
'issue': '', 'doi': '', 'venue': {'raw': 'BIAA'}, 'abstract': '', 'references': '', 'nb_references': 0, 'fos': {'name': '', 'w': ''}, 'author': {'name': '', 'i
d': '', 'org': ''}} has non list value {'name': '', 'w': ''} for path fos. Must be list or null.
```

Figura 4.1

Si è reso quindi necessario l'uso di un'altra funzione, diversa da read_json_flatten, spiegata di seguito. I commenti vengono lasciati per rendere il codice più leggibile.
Inizialmente tale funzione si comporta come read_json, leggendo il Json linea per linea e affettandolo in chunks, itera su ogni chunk e mantiene solo le colonne di interesse (recordpath e meta), converte il DataFrame così creato in un dict tramite il metodo Pandas *to_dict* secondo l'orientamento 'records' , crea un DataFrame temporaneo in cui memorizza gli elementi del dict tramite il metodo Pandas *json_normalize* e restituisce il DataFrame finale ottenuto tramite concatenazione in modo iterativo. Così facendo quegli elementi del Json per cui la chiave "fos" è un oggetto con coppie chiave/valore e non una lista di oggetti non vengono considerati, nel DataFrame source_df infatti vengono mantenute solo le colonne recordpath e meta (chiavi "fos" e "id"), bypassando così l'errore.

```
def read_json_flatten_from_dict
(df,path,data,recordpath,meta,metaprefix,recordprefix):
#legge il json in chunks, itera sui chunk e chiama la funzione read_json_chunk
    with open (path) as f:
        chunks=pd.read_json(f, lines=True, chunksize=10000)
        for chunk in chunks:
            source_df=read_json_chunk(chunk,data)
            #mantiene solo le colonne necessarie
            source_df=source_df[[recordpath,meta]]
            #converte il dataframe in dict
            source_df = source_df.to_dict(orient='records')
            #memorizza gli elementi del dict in un dataframe temporaneo
            df_tmp=pd.json_normalize(source_df,
                        recordpath,meta,metaprefix,recordprefix)
            #concatena il dataframe temporaneo e quello inizialmente vuoto,
            alla fine del loop conterrà tutti gli elementi della chiave fos
            df=pd.concat(objs=[df,df_tmp])
        return df
```

Di seguito la chiamata della funzione, nel file di esecuzione principale, per creare il DataFrame fos_df.

```
fos_df=read_json_flatten_from_dict(fos_df,path,data,'fos','id','paper_','fos_')
```

Questi tre DataFrame “padri” così ricavati: paper_df, author_df, fos_df raccolgono l’intero dataset e sono tutto ciò che serve per le successive fasi di analisi ed elaborazione, da questi verranno ricavati i DataFrame che saranno poi mappati nelle tabelle del database.

4.2 Analisi

In questa parte viene discusso lo script analyze_df.py contenente le funzioni volte ad analizzare i DataFrame ottenuti nella fase di importazione, dopo ciò si avrà una visione più completa del dataset permettendo così una prima bozza della struttura del database.

La seguente funzione stampa un riepilogo completo del DataFrame assieme al conteggio dei valori non nulli.

```
def info(df):
    return df.info(verbose=True,show_counts=True)
```

La seguente funzione verifica se, su un sottoinsieme delle colonne del DataFrame, siano presenti dei duplicati, restituendo un DataFrame di records duplicati ordinati in base alle colonne passate.

```
def check_duplicates (df,columns_list):
        return df[df.duplicated(subset=columns_list,
                  keep=False)].sort_values(by=columns_list)
```

Dal momento che la funzione info restituisce un conteggio dei valori non nulli, dove per 'nullo', in Pandas, si intendono quei records con valore 'NaN' o 'None', con le seguenti funzioni si vuole verificare quanti valori nel DataFrame sono effettivamente assenti, ossia quale percentuale di records contiene il valore 'NaN', 'None' oppure la stringa vuota '' .

```
def miss_percentage(df):
    for columns in df.columns:
        count_miss_value(df,columns)

def count_miss_value (df, column):
    i=0
    j=0
    i=df[column].isna().sum()
    j=df[df[column]==''][column].count() #uso di una maschera booleana per
      contare i valori ''
    return print(f'Percentuale di valori nulli per la colonna {column} =
                 {((i+j)/len(df)*100)}%')
```

La seguente funzione, dato un DataFrame, ricerca i valori massimi e minimi per quelle colonne che contengono un tipo di dato numerico.

```
def check_max_min (df):
    for column in df.columns:
        if df[column].dtype!=object:
            print (f'Colonna {column} : valore massimo {df[column].max()},
                        valore minimo {df[column].min()}')
```

Si procede ora all'applicazione di queste funzioni sui DataFrame ottenuti nella fase di importazione.

4.2.1 paper_df

La chiamata della funzione info su paper_df restituisce il seguente output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4894063 entries, 0 to 4894062
Data columns (total 23 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   id              4894063 non-null  int64
 1   title           4894063 non-null  object
 2   year            4894063 non-null  int64
 3   n_citation      4894063 non-null  int64
 4   page_start      4894063 non-null  object
 5   page_end        4894063 non-null  object
 6   doc_type        4894063 non-null  object
 7   publisher       4894063 non-null  object
 8   volume          4894063 non-null  object
 9   issue           4894063 non-null  object
 10  doi             4894063 non-null  object
 11  abstract        4894063 non-null  object
 12  references      4894063 non-null  object
 13  nb_references   4894063 non-null  int64
 14  venue.raw       4894063 non-null  object
 15  venue.id        4419076 non-null  object
 16  venue.type      4419078 non-null  object
 17  author.name     3839999 non-null  object
 18  author.id       3839999 non-null  object
 19  author.org      3839999 non-null  object
 20  fos.name        16680 non-null    object
 21  fos.w           16680 non-null    object
 22  alias_ids       21408 non-null    object
dtypes: int64(4), object(19)
memory usage: 858.8+ MB
```

Dal riepilogo si può notare che ci sono delle proprietà che non erano state considerate nella prima osservazione del dataset: "fos.name", "fos.w" e "alias_ids".

Dal momento che il metodo Pandas *json_normalize* utilizza come separatore di default il '.' per quei records annidati del tipo {'foo': {'bar': 0}} —> foo.bar , è possibile dedurre che "fos.name", "fos.w" corrispondano ad un oggetto contenente due coppie chiave/valore (come evidenziato dall'errore in Figura 4.1).

Andando ad osservare direttamente il file dblp1.json è possibile notare come la chiave "alias_ids" (evidenziata in grassetto nell'esempio sottostante) contenga un array di valori numerici, non solo, anche la chiave "references" (in grassetto) contiene un array di valori numerici e non di stringhe, come invece indicato dallo schema dei dati in Tabella 3.1.


```
{"_index":"dblp1","_type":"_doc","_id":"332601876","_score":1,"_source":
{"id":332601876,
"alias_ids":[2149364908,2606653434,2911182423],
"authors":[{"name":"Karen Neville","id":2157551517},{"name":"Philip
Powell","id":2274944363},{"name":"Niki Panteli","id":254914480}],
"title":"Knowledge and Security.",
"year":2003,
"n_citation":10,
"page_start":"262",
"page_end":"",
"doc_type":"Conference","publisher":"",
"volume":"",
"issue":"",
"doi":"",
"references":[57388316,1487897302,1997603734,2164813484,2615164812],
"fos":[{"name":"Incentive","w":0.49507},{"name":"Personal knowledge
management","w":0.70274},{"name":"Computer science","w":0.41803},
{"name":"Competitive advantage","w":0.57761},{"name":"Knowledge
management","w":0.47981},{"name":"Knowledge value chain","w":0.72095},
{"name":"Knowledge creation","w":0},{"name":"Organizational
learning","w":0.70171},{"name":"Competitor analysis","w":0.51151}],
"venue":{"raw":"Americas Conference on Information
Systems","id":1171805742,"type":"C"},
"abstract":"While knowledge may be a key organizational resource, it will only
provide sustainable competitive advantage if it is protected. Understanding of
security has expanded to include collaborating with competitors and virtual
relationships, resulting in complex interactions and risks. Individuals form
ties with peers and others to collaborate and create knowledge. However,
collaboration can, if not managed and controlled effectively become a threat to
security. Organizations with the ability to protect valuable knowledge have
many incentives to do so. This research investigates the interactions of
security and knowledge to develop a model to research success and failure
factors. Knowledge creation depends on sharing and protection, but these
interact, as a secure environment is necessary to create knowledge but
knowledge of security is necessary in the provision of a secure environment.
Examples from an on-going case illustrate use of the model. ",
"nb_references":5,
"author":{"name":"","id":"","org":""}}}
```

Poiché molti di questi valori risultano assenti, viene verificata l'effettiva percentuale di valori nulli chiamando la funzione miss_percentage su paper_df.
La chiamata di tale funzione restituisce il seguente output, si noti che alcune colonne del DataFrame sono state rinominate (evidenziate in grassetto):

```
Percentuale di valori nulli per la colonna paper_id = 0.0%
Percentuale di valori nulli per la colonna title = 0.0%
Percentuale di valori nulli per la colonna year = 0.0%
Percentuale di valori nulli per la colonna n_citation = 0.0%
Percentuale di valori nulli per la colonna page_start = 10.989580640870376%
Percentuale di valori nulli per la colonna page_end = 15.97310864204241%
Percentuale di valori nulli per la colonna doc_type = 10.20503822692924%
Percentuale di valori nulli per la colonna publisher = 15.526424567889707%
Percentuale di valori nulli per la colonna volume = 55.391338444151614%
Percentuale di valori nulli per la colonna issue = 66.93458993069767%
Percentuale di valori nulli per la colonna doi = 19.88376528867732%
Percentuale di valori nulli per la colonna abstract = 13.517582425890309%
Percentuale di valori nulli per la colonna references = 22.822673104126366%
Percentuale di valori nulli per la colonna nb_references = 0.0%
Percentuale di valori nulli per la colonna venue_raw = 1.4934625892637672%
Percentuale di valori nulli per la colonna venue_id = 10.667925607005877%
Percentuale di valori nulli per la colonna venue_type = 10.667884741164958%
Percentuale di valori nulli per la colonna author.name = 100.0%
Percentuale di valori nulli per la colonna author.id = 100.0%
Percentuale di valori nulli per la colonna author.org = 100.0%
Percentuale di valori nulli per la colonna fos.name = 100.0%
Percentuale di valori nulli per la colonna fos.w = 100.0%
Percentuale di valori nulli per la colonna alias_ids = 99.56257203881519%
```

Come si può notare le colonne author.name, author.id, author.org, fos.name, fos.w non sono mai valorizzate, di conseguenza potranno essere rimosse nella fase di elaborazione dei DataFrame.

La funzione check_duplicates richiamata sulla colonna paper_id di paper_df restituisce un DataFrame vuoto, significa che tra i valori della colonna paper_id non ci sono duplicati.

```
check_duplicates(paper_df,['paper_id'])
```

paper_id	title	year	n_citation	page_start	page_end	doc_type	publisher	volume	issue	doi	abstract	nb_references	venue_raw	venue_id	venue_type

4.2.2 author_df

La chiamata della funzione info su author_df restituisce il seguente output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14934850 entries, 0 to 14934849
Data columns (total 4 columns):
 #   Column       Non-Null Count     Dtype 
---  ------       --------------     ----- 
 0   author_name  14934850 non-null  object
 1   author_id    14934850 non-null  int64 
 2   author_org   11361508 non-null  object
 3   paper_id     14934850 non-null  object
dtypes: int64(1), object(3)
memory usage: 455.8+ MB
```

Nel riepilogo sopra si può notare la presenza di una proprietà “author_org”, all’interno della chiave “authors”, che non era stato possibile individuare nella prima osservazione del dataset, questa proprietà rappresenta l’affiliazione dell’autore per una determinata pubblicazione.

4.2.3 fos_df

Infine viene richiamata la funzione info su fos_df, restituendo il seguente output:

```
<class ‘pandas.core.frame.DataFrame’>
Int64Index: 45029752 entries, 0 to 26609
Data columns (total 3 columns):
 #   Column    Non-Null Count     Dtype  
---  ------    --------------     -----  
 0   fos_name  45029752 non-null  object 
 1   fos_w     45029752 non-null  float64
 2   paper_id  45029752 non-null  object 
dtypes: float64(1), object(2)
memory usage: 1.3+ GB
```

4.3 Progettazione concettuale: Schema E-R

Ci sono ora tutti gli elementi per progettare una prima bozza dello schema Entity-Relationship che andrà a modellare il database.

Di seguito viene illustrato lo schema E-R costruito sulla base dei dati osservati nella fase precedente, la correttezza di tale schema verrà verificata nella fase di elaborazione dei DataFrame.

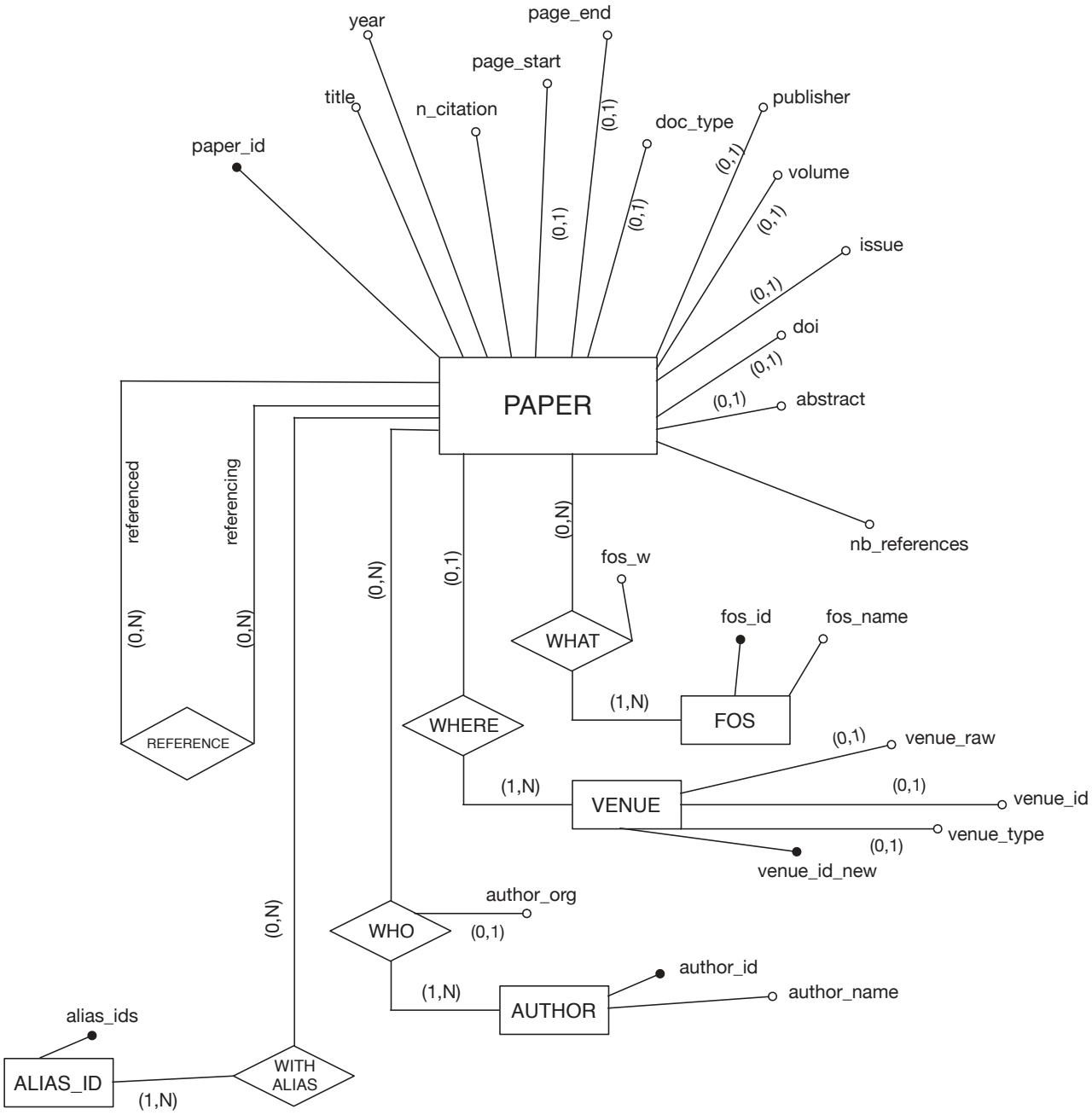

Figura 4.2

La scelta degli attributi opzionali è stata dettata dai riepiloghi ottenuti sui tre DataFrame nella fase precedente.
Sostanzialmente ogni proprietà, all'interno dell'oggetto Json rappresentante un paper, contenente o un array di oggetti/valori o un oggetto con coppie chiave/valore è stata modellata come entità.
Di seguito si riportano le tabelle delle entità e delle associazioni.

4.3.1 Entità

ENTITA'	DESCRIZIONE	ATTRIBUTI	ATTRIBUTI CHIAVE
PAPER	Pubblicazione scientifica	PAPER_ID, TITLE, YEAR, N_CITATION, PAGE_START, PAGE_END, DOC_TYPE, PUBLISHER, VOLUME, ISSUE, DOI, ABSTRACT, NB_REFERENCES	PAPER_ID
VENUE	Sede di competenza della pubblicazione	VENUE_ID, VENUE_RAW, VENUE_TYPE, VENUE_ID_NEW	VENUE_ID_NEW
FOS	Ambito di studio della pubblicazione	FOS_ID, FOS_NAME	FOS_ID
AUTHOR	Autore della pubblicazione	AUTHOR_ID, AUTHOR_NAME	AUTHOR_ID
ALIAS_ID	Codice	ALIAS_IDS	ALIAS_IDS

Tabella 4.1

Per l'entità VENUE è stato introdotto un nuovo attributo (VENUE_ID_NEW) generato appositamente per identificare in modo univoco le occorrenze dell'entità, in quanto l'attributo VENUE_ID, dall'analisi dei dati, non risulta sempre valorizzato, ossia contiene valori nulli, rendendolo non adatto come identificatore.
Per l'entità FOS è stato introdotto un nuovo attributo (FOS_ID) per identificare in modo univoco le occorrenze dell'entità e per velocizzare la ricerca degli ambiti di studio per un determinato PAPER_ID.

4.3.2 Associazioni

ASSOCIAZIONE	DESCRIZIONE	ENTITA' CONIVOLTE	ATTRIBUTI
REFERENCE	Associa ad un paper il riferimento ad un altro paper	PAPER (REFERENCED) (0,N), PAPER (REFERENCING) (0,N)	
WHAT	Associa un ambito di studio ad un paper	PAPER (0,N) FOS (1,N)	FOS_W
WHO	Associa un autore ad un paper	PAPER (0,N), AUTHOR (1,N)	AUTHOR_ORG
WHERE	Associa una sede di competenza ad un paper	PAPER (0,1), VENUE(1,N)	
WITH_ALIAS	Associa un alias_id ad un paper	PAPER (0,N), ALIAS_IDS (1,N)	

Tabella 4.2

Regole di derivazione:

- L'attributo NB_REFERENCES può essere derivato contando le occorrenze di REFERENCE per un determinato PAPER_ID.

4.4 Progettazione Logica: Schema Logico

Si procede alla traduzione dello schema E-R nello schema logico equivalente:

- Ogni entità è stata tradotta in una relazione avente lo stesso nome, i medesimi attributi e per chiave il suo identificatore;
- Ogni associazione N-aria è stata tradotta in una relazione avente lo stesso nome e per attributi gli attributi della relazione (dove presenti) e gli identificatori delle entità coinvolte (questi ultimi formano la chiave della relazione);
- L'unica associazione uno a molti (WHERE) è stata tradotta ed incorporata nella relazione PAPER contenente, oltre agli attributi della relativa entità, l'identificatore dell'entità coinvolta nell'associazione (VENUE) rappresentando così il vincolo di integrità referenziale tra l'attributo VENUE_ID_NEW di PAPER e l'attributo VENUE_ID_NEW di VENUE.

La correttezza di tale schema verrà verificata nella fase di elaborazione dei DataFrame.

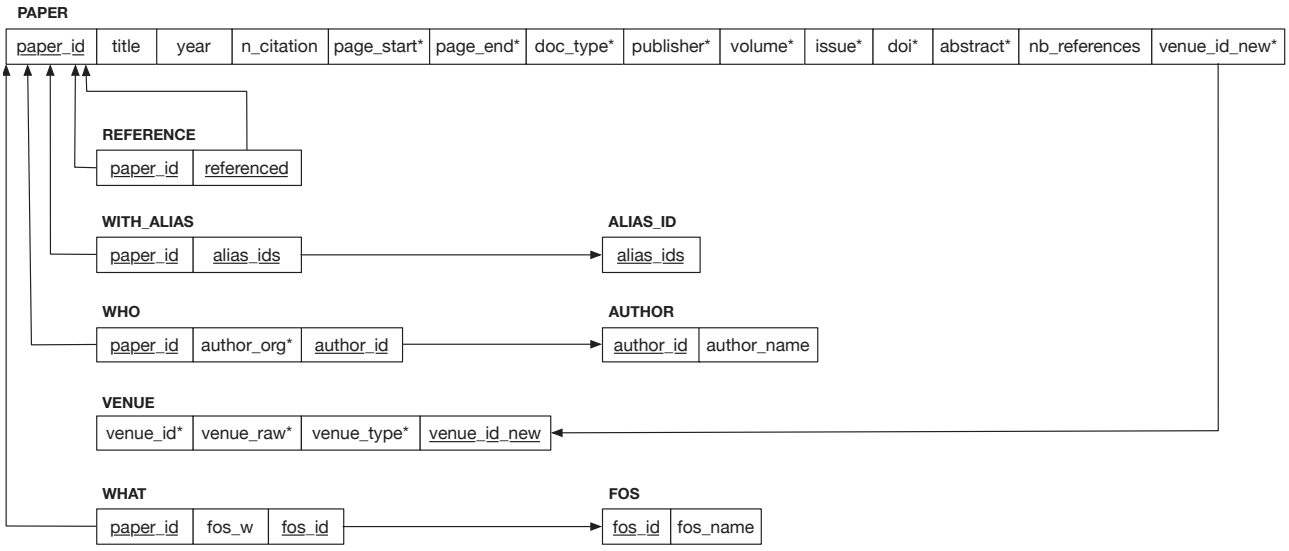

Figura 4.3

È ora possibile procedere all'elaborazione dei DataFrame.

4.5 Elaborazione

In questa parte viene analizzato lo script manipulate_df.py con le funzioni volte a manipolare i DataFrame creati nella fase di importazione attraverso i metodi di Pandas, l'obiettivo è di creare tanti DataFrame quante sono le relazioni presenti nello schema logico, con tante colonne (per ogni DataFrame) quanti sono gli attributi delle omonime relazioni.

Poiché ci sono dati, all'interno dei DataFrame, contenenti la stringa vuota (''), è stata creata una funzione che, dato un DataFrame ed una stringa (nello specifico: string=''), sostituisce i valori contenenti quella stringa con numpy.nan, questo è stato fatto affinché gli elementi contenenti la stringa vuota possano essere correttamente interpretati come valori mancanti all'interno delle tabelle del database. La funzione restituisce un riepilogo completo del DataFrame.

```
def fill_with_nan (df,string):
    df.replace(to_replace=string, value=np.nan, inplace=True)
    return df.info(verbose=True,show_counts=True)
```

La seguente funzione, dato un DataFrame, elimina le righe completamente nulle e successivamente quelle duplicate, restituendo un riepilogo completo del DataFrame.

```
def del_null_duplicates(df):
    df.dropna(axis='index', how='all', inplace=True)
    df.drop_duplicates(inplace=True, ignore_index=True)
    return df.info(verbose=True, show_counts=True)
```

La seguente funzione effettua il cast della colonna di un DataFrame al tipo di dato passato.

```
def cast_to(df,column,dtype):
    df[column]=(df[column]).astype(dtype,copy=False)
```

Con il frammento di codice seguente viene eseguita la funzione *hash_pandas_object* sul DataFrame restituendo un valore di hash per ogni riga del DataFrame. La Series risultante viene convertita al tipo di dato passato ed infine ne viene preso il valore assoluto affinché i suoi elementi siano positivi. È stato deciso di mantenere positivi gli elementi della Series restituita da tale funzione in quanto verrà utilizzata per creare i codici identificativi per i DataFrame venue e fos.

```
def hash_df(df,dtype):
    return abs((pd.util.hash_pandas_object(df,
                    index=False)).astype(dtype,copy=False))
```

Questa funzione serve a trasformare ogni elemento di un array di valori in una riga del DataFrame, verrà utilizzata per i DataFrame with_alias e reference.

```
def expand_column(df,column):
    return df.explode(column,ignore_index=True)
```

Una volta esposte le funzioni principali dello script è ora possibile passare all'elaborazione dei singoli DataFrame.

4.5.1 paper_df

Per prima cosa viene richiamata la funzione fill_with_nan su paper_df per sostituire i valori dei records contenenti la stringa vuota ('') con 'NaN', al fine di un corretto conteggio e rappresentazione dei valori nulli.
Si ottiene il seguente riepilogo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4894063 entries, 0 to 4894062
Data columns (total 18 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   paper_id        4894063 non-null  int64  
 1   title           4894063 non-null  object 
 2   year            4894063 non-null  int64  
 3   n_citation      4894063 non-null  int64  
 4   page_start      4356226 non-null  object 
 5   page_end        4112329 non-null  object 
 6   doc_type        4394622 non-null  object 
 7   publisher       4134190 non-null  object 
 8   volume          2183176 non-null  object 
 9   issue           1618242 non-null  object 
 10  doi             3920939 non-null  object 
 11  abstract        4232504 non-null  object 
 12  references      3777107 non-null  object 
 13  nb_references   4894063 non-null  int64  
 14  venue_raw       4820972 non-null  object 
 15  venue_id        4371968 non-null  float64
 16  venue_type      4371970 non-null  object 
 17  alias_ids       21408 non-null    object 
dtypes: float64(1), int64(4), object(13)
memory usage: 672.1+ MB
```

Pandas utilizza i dtypes di NumPy per le Series e le colonne dei DataFrame, NumPy è una libreria open source di Python che vanta una vasta collezione di funzioni matematiche di alto livello che permettono di operare in maniera efficiente su grandi matrici e array multidimensionali.
Per superare alcune limitazioni della libreria, Pandas definisce un'interfaccia per implementare tipi di dato che estendono quelli di NumPy, in questa fase verrà fatto uso delle estensioni dei tipi di dato 'CategoricalDType' e 'Int64DType'.

Osservando i dati si suppone che le colonne doc_type e venue_type possano contenere un numero limitato di valori, se così fosse potrebbe essere utile convertire tali colonne nel tipo di dato 'CategoricalDType' al fine di ottenere un maggiore risparmio nell'utilizzo della memoria.
Come descritto nella documentazione di Pandas[2]:
"*Categoricals sono un tipo di dati corrispondente a variabili categoriali nelle statistiche. Una variabile categoriale assume un numero limitato, e generalmente fisso, di valori possibili.*"

Di seguito è riportata la parte di codice sorgente riguardante tale verifica e i relativi output.

```
paper_df['doc_type'].unique()
array(['Conference', 'Journal', nan, 'Book', 'Repository', 'Patent',
       'BookChapter'], dtype=object)

paper_df['venue_type'].unique()
array(['C', 'J', nan], dtype=object)
```

I dati osservati suggeriscono anche che le colonne volume ed issue potrebbero contenere dati di tipo numerico, più precisamente interi, si tenta quindi il cast ad 'Int64' (si noti l'uso della maiuscola). 'Int64' è una stringa per il tipo di dato 'Int64DType', un tipo di estensione implementata all'interno di Pandas per poter rappresentare dati di tipo intero con valori mancanti, questo è necessario in quanto 'NaN' è di tipo float e la presenza di valori mancanti per volume ed issue forzerebbe tali colonne al tipo di dato float.

```
cast_to(paper_df,'volume','Int64')
cast_to(paper_df,'issue','Int64')
```

Per ottimizzare ulteriormente l'uso della memoria, vengono ricercati i valori massimi e minimi per quelle colonne di tipo numerico in paper_df, per verificare se esiste la possibilità di effettuare la conversione a qualche altro tipo di intero.

```
check_max_min(paper_df)

Colonna paper_id : valore massimo 3009038462, valore minimo 1091
Colonna year : valore massimo 2020, valore minimo 1800
Colonna n_citation : valore massimo 48327, valore minimo 0
Colonna volume : valore massimo 9783642544323, valore minimo -11
Colonna issue : valore massimo 9789812879356, valore minimo -77
Colonna nb_references : valore massimo 1812, valore minimo 0
Colonna venue_id : valore massimo 2996807011.0, valore minimo 182001.0
```

Dato l'output precedente vengono fatte le seguenti considerazioni:

- Le colonne year, n_citation, nb_references possono essere convertite al tipo di dato 'int16';
- Le colonne volume e issue, data la natura dei dati che rappresentano (volume ed edizione) contengono dati inconsistenti a causa della presenza di valori negativi, verranno comunque mantenuti immutati;
- La colonna venue_id è di tipo 'float64' a causa della presenza di valori nulli ('NaN'), verrà quindi eseguito il cast al tipo 'Int64' (si noti l'uso della maiuscola);
- Le colonne page_start e page_end non possono essere convertite ad intero, in quanto contengono dati inconsistenti ('139-148') data la natura dei dati che rappresentano, difatti il tentativo di conversione restituisce l'errore seguente, viene quindi mantenuto il tipo di dato 'object'.

```
ValueError: invalid literal for int() with base 10: '139-148'
```

Figura 4.4

Nel frammento sottostante si eseguono i cast sulla base delle osservazioni precedenti, viene fornito un riepilogo con il tipo di dati e l'utilizzo della memoria aggiornati.

```
cast_to(paper_df,'doc_type','category')
cast_to(paper_df,'venue_type','category')
cast_to(paper_df,'venue_id','Int64')
cast_to(paper_df,'year','int16')
cast_to(paper_df,'n_citation','int16')
cast_to(paper_df,'nb_references','int16')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4894063 entries, 0 to 4894062
Data columns (total 16 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   paper_id       4894063 non-null  int64
 1   title          4894063 non-null  object
 2   year           4894063 non-null  int16
 3   n_citation     4894063 non-null  int16
 4   page_start     4356226 non-null  object
 5   page_end       4112329 non-null  object
 6   doc_type       4394622 non-null  category
 7   publisher      4134190 non-null  object
 8   volume         2183176 non-null  Int64
 9   issue          1618242 non-null  Int64
 10  doi            3920939 non-null  object
 11  abstract       4232504 non-null  object
 12  nb_references  4894063 non-null  int16
 13  venue_raw      4820972 non-null  object
 14  venue_id       4371968 non-null  Int64
 15  venue_type     4371970 non-null  category
dtypes: Int64(3), category(2), int16(3), int64(1), object(7)
memory usage: 462.1+ MB
```

Da paper_df vengono creati i DataFrame reference, with_alias , alias_ids, venue e paper.

4.5.2 reference

Per il DataFrame reference vengono selezionate le colonne paper_id, references e nb_references (quest'ultima servirà per filtrare velocemente le righe di interesse) e memorizzate in un DataFrame temporaneo (reference_df).

Tramite l'utilizzo di una maschera booleana vengono mantenute solo le righe che hanno almeno una reference e con la chiamata ad expand_column ogni elemento della lista di riferimenti (nella colonna references) viene trasformato in una riga del nuovo DataFrame reference.

```
reference_df=paper_df[['paper_id','nb_references','references']]
reference_df=reference_df[reference_df['nb_references']!=0]
reference=expand_column(reference_df,'references')
```

Seguono il controllo dei duplicati (check_duplicates) , l'eliminazione della colonna nb_references e il cast della colonna references al tipo di dato 'int64', si ottiene il seguente riepilogo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45564149 entries, 0 to 45564148
Data columns (total 2 columns):
 #   Column      Non-Null Count     Dtype
---  ------      --------------     -----
 0   paper_id    45564149 non-null  int64
 1   references  45564149 non-null  int64
dtypes: int64(2)
memory usage: 695.3 MB
```

4.5.3 alias_id e with_alias

Non essendo chiara la natura dei dati contenuti all'interno della colonna alias_ids e per mantenere tutte le informazioni presenti nel dataset è stato deciso di rappresentare tale dato tramite due DataFrame dedicati (come da schemi concettuale e logico).
Per il DataFrame with_alias vengono mantenute le colonne paper_id e alias_ids e memorizzate in un DataFrame temporaneo (alias_id_df).
Tramite l'utilizzo di una maschera booleana vengono selezionate solo quelle righe per cui alias_ids non è nullo e con la chiamata ad expand_column ogni elemento della lista di alias (colonna alias_ids) viene trasformato in una riga del nuovo DataFrame with_alias.

```
alias_id_df=paper_df[['paper_id','alias_ids']]
alias_id_df=alias_id_df[alias_id_df['alias_ids'].notna()]
with_alias=expand_column(alias_id_df,'alias_ids')
alias_id=with_alias['alias_ids']
```

Seguono il controllo dei duplicati (check_duplicates) e il cast della colonna alias_ids di with_alias a 'int64'. Il DataFrame per alias_id è semplicemente una Series ricavata dalla colonna alias_ids di with_alias. Si ottengono i seguenti riepiloghi:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77909 entries, 0 to 77908
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   paper_id   77909 non-null  int64
 1   alias_ids  77909 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.2+ MB
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 77909 entries, 0 to 77908
Series name: alias_ids
Non-Null Count  Dtype
--------------  -----
77909 non-null  int64
dtypes: int64(1)
memory usage: 608.8 KB
```

4.5.4 venue

Per il DataFrame venue vengono mantenute le colonne venue_id, venue_raw, venue_type.

```
venue=paper_df[['venue_id','venue_raw','venue_type']]
```

Seguono il controllo dei duplicati (check_duplicates) e l'eliminazione dei valori nulli e duplicati (del_null_duplicates), ottenendo il seguente riepilogo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49202 entries, 0 to 49201
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   venue_id    10480 non-null  Int64   
 1   venue_raw   49201 non-null  object  
 2   venue_type  10482 non-null  category
dtypes: Int64(1), category(1), object(1)
memory usage: 865.1+ KB
```

Come accennato in precedenza ed osservando il riepilogo, venue non possiede un proprio identificativo in quanto molti dei suoi valori (in particolare quelli della colonna venue_id) sono 'NaN', si procede dunque a creare un codice identificativo che sarà usato come chiave primaria nel database.
Prima di procedere in tal senso è necessario verificare che i dati contenuti in venue non contengano duplicati su un sottoinsieme delle sue colonne.

Analisi di venue

- Il controllo dei duplicati sul sottoinsieme (venue_id,venue_raw) restituisce un DataFrame vuoto, indice che non si cono duplicati;

```
check_duplicates(venue,['venue_id','venue_raw'])
```

venue_id	venue_raw	venue_type

- Il controllo dei duplicati sul sottoinsieme (venue_raw, venue_type) restituisce il seguente output, si tratta di due records omonimi in quanto differiscono nel venue_id, vengono quindi trattati come due venue differenti;

```
check_duplicates(venue,['venue_raw','venue_type'])
```

	venue_id	venue_raw	venue_type
224	2595313807	arXiv: Numerical Analysis	J
8552	2595836090	arXiv: Numerical Analysis	J

- Il controllo dei duplicati sul sottoinsieme (venue_id, venue_type) viene effettuato in due step, prima i duplicati di tale sottoinsieme vengono memorizzati in un DataFrame temporaneo restituendo il seguente riepilogo:

```
venue_null=check_duplicates(venue,['venue_id','venue_type'])
info(venue_null)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38722 entries, 6961 to 49201
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   venue_id    0 non-null      Int64
 1   venue_raw   38721 non-null  object
 2   venue_type  2 non-null      category
dtypes: Int64(1), category(1), object(1)
memory usage: 983.3+ KB
```

Successivamente notando che i venue_id duplicati sono solo quelli nulli (‘NaN’); si procede quindi a verificare quali sono i due valori non nulli duplicati su venue_type e si osserva che si tratta in effetti di due dati diversi;

```
venue_null[venue_null['venue_type'].notna()]
```

	venue_id	venue_raw	venue_type
6961	<NA>	hot topics in software upgrades	C
29004	<NA>	NaN	C

- Il controllo dei duplicati sulla colonna venue_id restituisce lo stesso riepilogo del punto precedente, portando alle medesime conclusioni;

```
info(check_duplicates(venue,['venue_id']))

<class 'pandas.core.frame.DataFrame'>
Int64Index: 38722 entries, 4 to 49201
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   venue_id    0 non-null      Int64   
 1   venue_raw   38721 non-null  object  
 2   venue_type  2 non-null      category
dtypes: Int64(1), category(1), object(1)
memory usage: 983.3+ KB
```

- Infine il controllo dei duplicati sulla colonna venue_raw restituisce un DataFrame con dei dati che vengono considerati come records diversi in quanto alcuni di questi in effetti fanno riferimento alla stessa venue_raw ma con due venue_type differenti e quindi correttamente sono associati a due venue_id diversi (vedi 'Soft Computing) , altri fanno riferimento alla stessa venue_raw ma con il resto delle informazioni assenti, non potendo fare alcuna deduzione sul valore dei campi mancanti, questi dati vengono considerati come referenti a venue diversi.

	venue_id	venue_raw	venue_type
29	2757487807	American Medical Informatics Association Annua...	C
36428	<NA>	American Medical Informatics Association Annua...	NaN
19413	97068678	Aslib Proceedings	J
5526	<NA>	Aslib Proceedings	NaN
34067	49206733	BMC Proceedings	J
16717	<NA>	BMC Proceedings	NaN
45217	56561474	Computational Intelligence	J
198	1120693805	Computational Intelligence	C
2386	1168542559	Data and Knowledge Engineering	C
38136	136993123	Data and Knowledge Engineering	J
2398	11479521	Decision Support Systems	J
696	1182309694	Decision Support Systems	C
19570	<NA>	Frontiers in Robotics and AI	NaN
6189	2595095599	Frontiers in Robotics and AI	J
1133	1201250571	Intelligent Data Analysis	C
13365	2498839158	Intelligent Data Analysis	J
1631	<NA>	Künstliche Intelligenz	NaN
1109	1001063841	Künstliche Intelligenz	J
5878	<NA>	Proceedings of SPIE	NaN
874	183492911	Proceedings of SPIE	J
11470	2996807011	SIAM Journal on Applied Algebra and Geometry	J
7734	<NA>	SIAM Journal on Applied Algebra and Geometry	NaN
9348	<NA>	Social Science Research Network	NaN
8554	2751751161	Social Science Research Network	J
320	1123077274	Soft Computing	C
1215	65753830	Soft Computing	J
2637	2605467207	Workshop on Hot Topics in Operating Systems	C
26377	<NA>	Workshop on Hot Topics in Operating Systems	NaN
224	2595313807	arXiv: Numerical Analysis	J
8552	2595836090	arXiv: Numerical Analysis	J

Non ha senso fare lo stesso genere di considerazioni sulla colonna venue_type in quanto questa contiene solo un numero ristretto di valori distinti: ‘C’,’J’, ‘NaN’.

Elaborazione di venue

Dopo aver verificato l’unicità dei dati all’interno del DataFrame, è ora possibile creare una nuova colonna contenente i codici identificavi per i records di venue.

```
venue['venue_id_new']=hash_df(venue,'int64')
```

Nonostante contenga già dati di tipo intero, la nuova colonna venue_id_new, viene convertita al tipo di dato ‘Int64’ (si noti la maiuscola), per evitare che, quando verrà integrata come colonna di paper_df, la presenza di valori ‘NaN’ la converta automaticamente nel tipo di dato ‘float’.

```
cast_to(venue,’venue_id_new','Int64')
```

Una volta eseguito il cast, viene chiamata la funzione info su venue e si ottiene il seguente riepilogo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49202 entries, 0 to 49201
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype   
---  ------        --------------  -----   
 0   venue_id      10480 non-null  Int64   
 1   venue_raw     49201 non-null  object  
 2   venue_type    10482 non-null  category
 3   venue_id_new  49202 non-null  Int64   
dtypes: Int64(2), category(1), object(1)
memory usage: 1.3+ MB
```

4.5.5 paper

Una volta ottenuti i DataFrame per reference e alias_id le relative colonne vengono eliminate da paper_df.
Al fine di rispettare lo schema logico, è ora necessario aggiungere la colonna venue_id_new (che rappresenterà la chiave esterna) al DataFrame paper, questo viene fatto tramite l'operazione di merge sinistro tra paper_df e venue (simile ad un left outer join) sull'insieme di colonne (venue_id ,venue_raw, venue_type).

```
paper=paper_df.merge(venue, how='left', on=['venue_id', 'venue_raw',
      'venue_type'])
```

Vengono eliminate dal DataFrame paper le colonne venue_id, venue_raw, venue_type in quanto non sono più necessarie.
Si ottiene il seguente riepilogo:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4894063 entries, 0 to 4894062
Data columns (total 14 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   paper_id        4894063 non-null  int64
 1   title           4894063 non-null  object
 2   year            4894063 non-null  int16
 3   n_citation      4894063 non-null  int16
 4   page_start      4356226 non-null  object
 5   page_end        4112329 non-null  object
 6   doc_type        4394622 non-null  category
 7   publisher       4134190 non-null  object
 8   volume          2183176 non-null  Int64
 9   issue           1618242 non-null  Int64
 10  doi             3920939 non-null  object
 11  abstract        4232504 non-null  object
 12  nb_references   4894063 non-null  int16
 13  venue_id_new    4820973 non-null  Int64
dtypes: Int64(3), category(1), int16(3), int64(1), object(6)
memory usage: 457.4+ MB
```

4.5.6 fos_df

Per prima cosa viene eseguito il cast della colonna paper_id di fos_df e viene chiamata la funzione fill_with_nan su questo DataFrame per sostituire i valori contenenti stringhe vuote con 'NaN', al fine di un corretto conteggio e rappresentazione dei valori nulli.

Si prosegue con la verifica dei duplicati sul DataFrame, si può notare che ci sono 4 coppie di duplicati, si procede dunque all'eliminazione dei valori completamente nulli e duplicati tramite la funzione del_null_duplicates.

```
check_duplicates(fos_df,['fos_name','fos_w','paper_id'])
```

	fos_name	fos_w	paper_id
36206	Radius of curvature	0.0	162420740
36207	Radius of curvature	0.0	162420740
87997	Radius of curvature	0.0	1998680315
87998	Radius of curvature	0.0	1998680315
10042	Radius of curvature	0.0	2121529848
10045	Radius of curvature	0.0	2121529848
21819	Radius of curvature	0.0	2131733916
21821	Radius of curvature	0.0	2131733916

```
del_null_duplicates(fos_df)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45029748 entries, 0 to 45029747
Data columns (total 3 columns):
 #   Column    Non-Null Count     Dtype
---  ------    --------------     -----
 0   fos_name  45029748 non-null  object
 1   fos_w     45029748 non-null  float64
 2   paper_id  45029748 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 1.0+ GB
```

Da fos_df vengono creati i DataFrame per fos e what.

4.5.7 fos

Per il DataFrame fos viene mantenuta solo la colonna fos_name e vengono rimossi i records completamente nulli e duplicati.

```
fos=fos_df[['fos_name']]
del_null_duplicates(fos)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132337 entries, 0 to 132336
Data columns (total 1 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   fos_name  132337 non-null  object
dtypes: object(1)
memory usage: 1.0+ MB
```

Trattandosi di una Series con valori univoci, si può procedere all'aggiunta della nuova colonna che conterrà i codici identificativi per i records di fos.

```
fos['fos_id']=hash_df(fos,'int64')
```

Di seguito il riepilogo per il DataFrame fos:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132337 entries, 0 to 132336
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   fos_name  132337 non-null  object
 1   fos_id    132337 non-null  int64
dtypes: int64(1), object(1)
memory usage: 2.0+ MB
```

4.5.8 what

Per rispetto dello schema logico è necessario aggiungere la colonna fos_id al DataFrame what, questo viene fatto tramite un'operazione di merge sinistro tra fos_df e fos (simile ad un left outer join) sulla colonna fos_name.

```
what_df=fos_df.merge(fos, how='left', on=['fos_name'])
```

Per il DataFrame what vengono mantenute le colonne paper_id, fos_id, fos_w.

```
what=what_df[['paper_id', 'fos_id','fos_w']]
```

Ricercando i duplicati sul sottoinsieme (paper_id, fos_id) , che dovrebbe costituire la chiave primaria per la tabella what nel database, si può notare che ci sono diversi valori duplicati, ossia per uno stesso paper_id si trova associato lo stesso fos_id con diversi fos_w, le conseguenti scelte a livello di implementazione verranno illustrate nella parte relativa allo progettazione fisica.
Per brevità e una maggior chiarezza viene mostrato solo l'output relativo ai duplicati nel DataFrame fos_df:

```
check_duplicates(what,['paper_id','fos_id'])
check_duplicates(fos_df,['fos_name','paper_id'])
```

	fos_name	fos_w	paper_id
18135081	Astigmatism	0.66538	204997851
18135082	Astigmatism	0.72689	204997851
11410030	Asymptotic analysis	0.80121	27097632
11410031	Asymptotic analysis	0.76464	27097632
38603528	Asymptotic analysis	0.58493	47018828
...	...	...	...
40650141	UML state machine	0.70119	2401885247
12305969	Ultrafiltration	0.46539	1973855211
12305970	Ultrafiltration	0.46091	1973855211
3408043	Ultrafiltration	0.48233	2065517646
3408044	Ultrafiltration	0.46963	2065517646

84922 rows × 3 columns

Di seguito viene riportato il riepilogo per il DataFrame what:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45029748 entries, 0 to 45029747
Data columns (total 3 columns):
 #   Column    Non-Null Count     Dtype
---  ------    --------------     -----
 0   paper_id  45029748 non-null  int64
 1   fos_id    45029748 non-null  int64
 2   fos_w     45029748 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 1.3 GB
```

4.5.9 author_df

Per prima cosa viene eseguito il cast della colonna paper_id di author_df e viene chiamata la funzione fill_with_nan su questo DataFrame per sostituire i valori dei records contenenti stringhe vuote con il valore 'NaN', al fine di un corretto conteggio e rappresentazione dei valori nulli.
Viene chiamata la funzione check_duplicates su author_df restituendo un DataFrame vuoto, indice che non ci sono duplicati.
Da author_df vengono creati i DataFrame per who e author.

4.5.10 who

Per il DataFrame who vengono selezionate le colonne paper_id, author_id, author_org e viene controllata la presenza di duplicati, il DataFrame restituito da check_duplicates è vuoto, indice che non ci sono duplicati. Di seguito il riepilogo per il DataFrame who:

```
who=author_df[['paper_id','author_id','author_org']]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14934850 entries, 0 to 14934849
Data columns (total 3 columns):
 #   Column      Non-Null Count     Dtype 
---  ------      --------------     ----- 
 0   paper_id    14934850 non-null  int64 
 1   author_id   14934850 non-null  int64 
 2   author_org  11361508 non-null  object
dtypes: int64(2), object(1)
memory usage: 341.8+ MB
```

4.5.11 author e author_extended

Per il DataFrame author si mantengono le colonne author_id, author_name, come da schema logico, e si rimuovono i valori completamente nulli e duplicati (del_null_duplicates).
Eseguendo una verifica dei duplicati sulla colonna author_id è possibile notare che sono presenti diversi aliases per i nomi degli autori.

```
author= author_df[['author_id', 'author_name']]
check_duplicates(author,['author_id'])
```

	author_id	author_name
4136798	32606	Stéphane Debricon
582289	32606	Stephane Debricon
420417	44880	Dario Garcia-Gasulla
1057216	44880	D. Garcia-Gasulla
288324	92734	Brian E. Whitacre
...	...	...
2601655	3008444288	WangTiejian
1523396	3009024956	Alper Ozbilen
4299955	3009024956	Alper OzbIlen
4300751	3009035817	Van der HoevenJoris
1522529	3009035817	Van Der HoevenJoris

1129902 rows × 2 columns

Per non perdere dati significativi e per non complicare troppo il codice, si è deciso di mantenere gli aliases dei nomi degli autori in un DataFrame a sé, chiamato author_extended, segue il riepilogo:

```
author_extended=author[['author_id','author_name']]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5070852 entries, 0 to 5070851
Data columns (total 2 columns):
 #   Column       Non-Null Count    Dtype 
---  ------       --------------    ----- 
 0   author_id    5070852 non-null  int64 
 1   author_name  5070852 non-null  object
dtypes: int64(1), object(1)
memory usage: 77.4+ MB
```

Ora è possibile eliminare da author tutti gli aliases, questo viene fatto rimuovendo i duplicati sul sottoinsieme author_id, per default viene mantenuta la prima occorrenza tra i duplicati, segue il riepilogo:

```
author.drop_duplicates(subset=['author_id'],inplace=True, ignore_index=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4398138 entries, 0 to 4398137
Data columns (total 2 columns):
 #   Column       Non-Null Count    Dtype 
---  ------       --------------    ----- 
 0   author_id    4398138 non-null  int64 
 1   author_name  4398138 non-null  object
dtypes: int64(1), object(1)
memory usage: 67.1+ MB
```

Si noti che i DataFrame ricavati nella fase di importazione (paper_df, fos_df, author_df), necessari per le fasi di analisi ed elaborazione, non saranno mappati in alcuna tabella del database.

4.6 Modifica degli schemi E-R e logico

Una volta ottenuti tutti i DataFrame necessari e date le osservazioni fatte nella fase di elaborazione, si procede alla modifica degli schemi E-R e logico.

Nello schema E-R vengono aggiunte:

- L'entità AUTHOR_EXTENDED con gli attributi AUTHOR_ID, AUTHOR_NAME, questa rappresenta gli autori referenziati in diversi modi, viene identificata dalla coppia di attributi;
- L'associazione uno a molti NAMED che associa un alias del nome dell'autore ad un determinato autore.

A questo punto l'attributo AUTHOR_NAME della relazione AUTHOR diventa ridondante, ma viene mantenuto per velocizzare operazioni del tipo: "Seleziona i nomi degli autori per un determinato paper_id".
Lo scopo della tabella AUTHOR_EXTENDED invece è quello di permettere di trovare i paper scritti da un determinato autore, a prescindere dal modo in cui è referenziato.

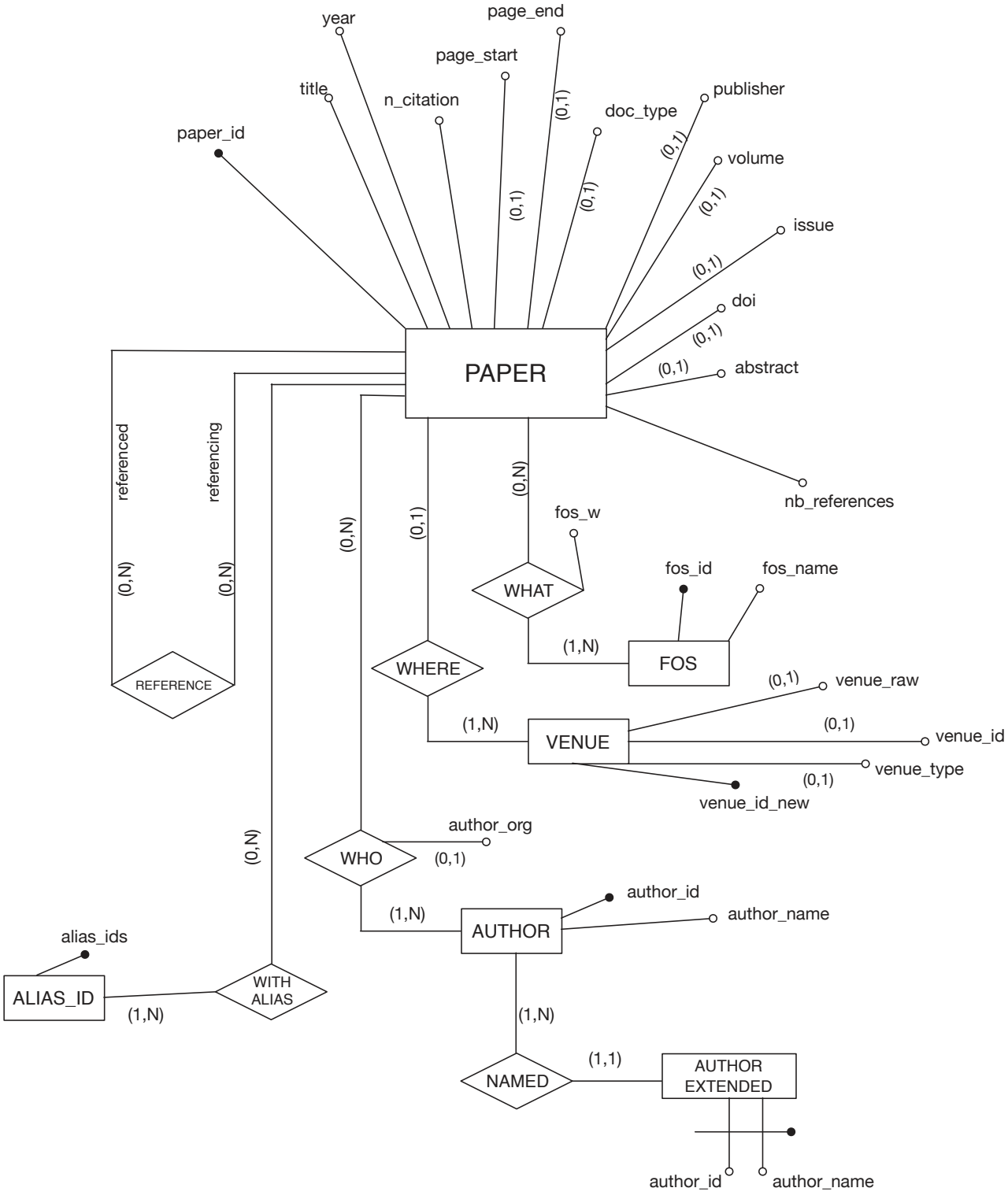
year
page_end
page_start
title
n_citation
(0,1)
doc_type
publisher
paper_id
(0,1)
volume
(0,1)
(0,1)
issue
(0,1)
doi
(0,1)
PAPER
(0,1)
abstract
referenced
referencing
(0,N)
(0,N)
(0,N)
(0,1)
(0,N)
nb_references
fos_w
fos_id
fos_name
WHAT
(1,N)
FOS
WHERE
REFERENCE
(0,1)
venue_raw
(1,N)
VENUE
(0,1)
venue_id
(0,1)
venue_type
venue_id_new
author_org
(0,N)
WHO
(0,1)
author_id
(1,N)
AUTHOR
author_name
alias_ids
ALIAS_ID
WITH
ALIAS
(1,N)
(1,N)
(1,1)
NAMED
AUTHOR
EXTENDED
author_id
author_name

Figura 4.5

Nello schema logico vengono effettuate le seguenti modifiche:

- Viene introdotta la relazione AUTHOR_EXTENDED (tradotta dallo schema E-R) con la coppia di attributi come chiave primaria;
- L'associazione uno a molti NAMED viene tradotta ed inglobata all'interno della relazione AUTHOR_EXTENDED con AUTHOR_ID come chiave esterna che referenzia l'omonimo attributo della tabella AUTHOR;
- Viene modificata la chiave primaria della relazione WHAT in quanto, nella fase di elaborazione, si è visto che il sottoinsieme di colonne (fos_id, paper_id) contiene duplicati, perciò come chiavi primaria viene scelta l'insieme dei tre attributi: PAPER_ID, FOS_ID, FOS_W.

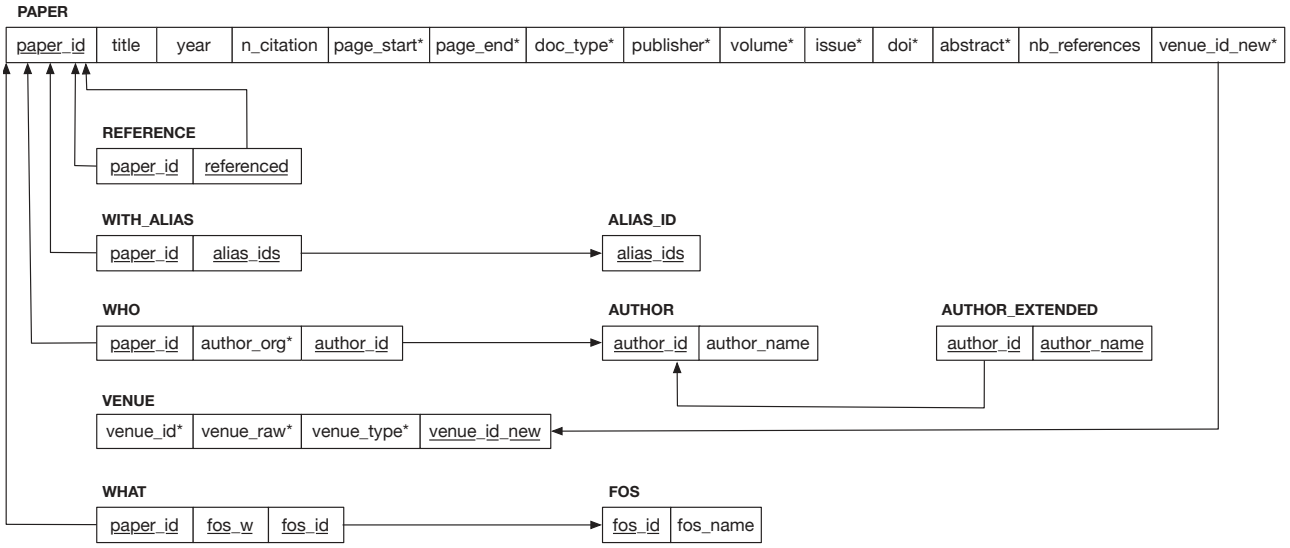

Figura 4.4

4.7 Progettazione fisica

Questa parte riguarda l'implementazione del database in PostgreSQL, ossia la traduzione dello schema logico in linguaggio SQL (Structured Query Language).
Di seguito i punti salienti:

- Ogni relazione (rappresentata lato Pandas da un DataFrame) è stata tradotta in una tabella;
- Per i tipi di dato 'object' è stato utilizzato il tipo di dato 'text';
- I tipi di dati intero sono stati tradotti nei loro equivalenti tipi numerici SQL a 2 ('smallint' per 'int16') e 8 byte ('bigint' per 'int64');
- Per i tipi di dato 'category' (attributi VENUE_TYPE e DOC_TYPE) vengono usati rispettivamente il tipo di dato 'varchar(1)' e 'text';
- Per l'unico dato di tipo 'float' (attributo FOS_W) viene usato il tipo 'decimal(6,5)';
- I valori NOT NULL sono stati scelti sulla base dei riepiloghi dei DataFrame nelle fasi precedenti;
- Coerentemente con quanto osservato nella fase di elaborazione, per gli attributi 'PAGE_START' e 'PAGE_END' viene usato il tipo di dato 'text', contenendo record del tipo '139-148';
- La politica di gestione delle modifiche alle righe delle tabelle a cui si riferiscono le chiavi esterne è di: impedire la cancellazione delle righe contenenti i valori referenziati, e di propagare la modifica delle righe contenenti i valori referenziati;
- Poiché lo scopo dell'elaborato è di modellare tramite un database il dataset dblp1.json senza eseguire query di ricerca complesse e date le inconsistenze ritrovate nei dati per il DataFrame what, tale oggetto viene tradotto in una tabella senza chiave primaria, dato che il linguaggio lo consente.

Il database è stato realizzato tramite il Query Tool di PgAdmin4.
Di seguito il codice SQL per lo schema del database, l'ordine di definizione delle tabelle è significativo avendo già incluso i vincoli di integrità referenziale:

```sql
CREATE TABLE VENUE(
    VENUE_ID BIGINT,
    VENUE_RAW TEXT,
    VENUE_TYPE VARCHAR(1),
    VENUE_ID_NEW BIGINT PRIMARY KEY
);

CREATE TABLE ALIAS_ID(
    ALIAS_IDS BIGINT PRIMARY KEY
);

CREATE TABLE PAPER(
    PAPER_ID BIGINT PRIMARY KEY,
    TITLE TEXT NOT NULL,
    YEAR SMALLINT NOT NULL,
    N_CITATION SMALLINT NOT NULL,
    PAGE_START TEXT,
    PAGE_END TEXT,
    DOC_TYPE TEXT,
    PUBLISHER TEXT,
    VOLUME BIGINT,
    ISSUE BIGINT,
    DOI TEXT,
    ABSTRACT TEXT,
    NB_REFERENCES SMALLINT NOT NULL,
    VENUE_ID_NEW BIGINT,
    FOREIGN KEY (VENUE_ID_NEW) REFERENCES VENUE(VENUE_ID_NEW)
    ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE REFERENCE(
    PAPER_ID BIGINT,
    REFERENCED BIGINT,
    PRIMARY KEY(PAPER_ID,REFERENCED),
    FOREIGN KEY (PAPER_ID) REFERENCES PAPER (PAPER_ID)
    ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (REFERENCED) REFERENCES PAPER (PAPER_ID)
    ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE WITH_ALIAS(
    PAPER_ID BIGINT,
    ALIAS_IDS BIGINT,
    PRIMARY KEY(PAPER_ID,ALIAS_IDS),
    FOREIGN KEY (PAPER_ID) REFERENCES PAPER (PAPER_ID)
    ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (ALIAS_IDS) REFERENCES ALIAS_ID (ALIAS_IDS)
    ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE AUTHOR(
    AUTHOR_ID BIGINT PRIMARY KEY,
    AUTHOR_NAME TEXT NOT NULL
```

```
);

CREATE TABLE AUTHOR_EXTENDED(
   AUTHOR_ID BIGINT,
   AUTHOR_NAME TEXT,
   PRIMARY KEY (AUTHOR_ID,AUTHOR_NAME),
   FOREIGN KEY (AUTHOR_ID) REFERENCES AUTHOR (AUTHOR_ID)
   ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE WHO(
    PAPER_ID BIGINT,
    AUTHOR_ID BIGINT,
    AUTHOR_ORG TEXT,
    PRIMARY KEY (PAPER_ID,AUTHOR_ID),
    FOREIGN KEY (PAPER_ID) REFERENCES PAPER (PAPER_ID)
    ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (AUTHOR_ID) REFERENCES AUTHOR (AUTHOR_ID)
    ON DELETE NO ACTION ON UPDATE CASCADE
);

CREATE TABLE FOS(
   FOS_ID BIGINT PRIMARY KEY,
   FOS_NAME TEXT NOT NULL
);

CREATE TABLE WHAT(
   PAPER_ID BIGINT NOT NULL,
   FOS_ID BIGINT NOT NULL,
   FOS_W DECIMAL (6,5) NOT NULL,
   FOREIGN KEY (PAPER_ID) REFERENCES PAPER (PAPER_ID)
   ON DELETE NO ACTION ON UPDATE CASCADE,
   FOREIGN KEY (FOS_ID) REFERENCES FOS (FOS_ID)
   ON DELETE NO ACTION ON UPDATE CASCADE
);
```

4.8 Connessione al database

In questa parte vengono esposte le funzioni dello script DB_connection.py che si occupano di scrivere i record memorizzati nei DataFrame nelle relative tabelle del database.

Per impostare la connessione al database è necessaria una connection string del formato illustrato nel commento sottostante, i parametri della stringa vengono forniti in un file di configurazione (config.py) sotto forma di dict.

```
import config as cfg

#formato della connection string:
#postgresql://[user[:password]@][host][:port][/dbname]

#recupera i parametri da config.py
user=cfg.postgres['user']
password=cfg.postgres['password']
host=cfg.postgres['host']
port=cfg.postgres['port']
dbname=cfg.postgres['dbname']

conn_str = f"""postgresql://{user}:{password}@{host}:{port}/{dbname}"""
```

La seguente funzione crea l'engine e la connessione al database.

```
def create_conn(conn_str):
    engine = create_engine(conn_str)
    connection = engine.connect()
    return connection
```

La seguente funzione, una volta creata la connessione, scrive i records memorizzati nei DataFrame nelle relative tabelle del database tramite il metodo Pandas *to_sql*. Vengono chiariti qui di seguito alcuni dei parametri utilizzati:

- Poiché le tabelle sono state create in precedenza in PgAdmin, viene passato il parametro if_exists='append' per inserire valori in una tabella esistente;
- Il parametro method='multi' permette di passare più valori (righe) in una singola clausola INSERT;
- Il parametro chunksize=10000 permette di specificare il numero di righe da scrivere in ogni batch alla volta (dimensione del blocco di scrittura).

Alla fine la connessione viene chiusa e vengono restituite il numero di righe scritte nella tabella.

```
def connect_db (df,table_name):
    connection=create_conn(conn_str)
    nb_row=df.to_sql(table_name,
con=connection,if_exists='append',method='multi',chunksize=10000,index=False)
    connection.close()
    return nb_row
```

Di seguito la chiamata alla funzione nel file di esecuzione principale, l'ordine è significativo per non creare errori in fase di inserimento, a causa dei vincoli di integrità referenziale.

```
connect_db(venue,'venue')
49202

connect_db(paper,'paper')
4894063

connect_db(reference,'reference')
45564149

connect_db(alias_id,'alias_id')
77909

connect_db(with_alias,'with_alias')
77909

connect_db(author,'author')
4398138

connect_db(author_extended,'author_extended')
5070852

connect_db(who,'who')
14934850

connect_db(fos,'fos')
132337

connect_db(what,'what')
45029748
```

La seguente funzione interroga il database e stampa il risultato della query SQL, la query in questione conta il numero di righe non nulle per un particolare attributo della tabella.
Il blocco try except è necessario a causa della tabella ALIAS_ID, in quanto, lato Pandas, si tratta di una Series, e non di un DataFrame, conseguentemente l'attributo df.columns non esiste per le Series.
Poiché la query restituisce un'unica riga, per visualizzare il risultato viene chiamato il metodo SQLAlchemy *fetchone* che restituisce un oggetto RowProxy rappresentante una singola riga di risultati e che agisce come una tupla in Python, da questa viene selezionato il primo elemento con results.fetchone [0].

```
def count_row_from_db(df,table_name):
    connection = create_conn(conn_str)
    print (f"""Tabella {table_name}:""")
    try:
        for column in df.columns:
            sql_query=f"""SELECT COUNT({column}) FROM {table_name}"""
            results=connection.execute(sql_query)
            print (f"""Nr righe non nulle per {column} = {results.fetchone()
                        [0]}""")
    except AttributeError:
        sql_query=f"""SELECT COUNT(*) FROM {table_name}"""
        results=connection.execute(sql_query)
        print (f"""Nr righe non nulle = {results.fetchone()[0]}""")
    print()
    connection.close()
```

Di seguito la chiamata della funzione nel file di esecuzione principale, con i relativi output. Confrontando tali risultati con i riepiloghi ottenuti dagli omonimi DataFrame (riportati qui sotto per facilità di consultazione) si può notare che il numero di record passato (con i relativi valori non nulli) corrisponde.

```
count_row_from_db(paper,'paper')
count_row_from_db(venue,'venue')
count_row_from_db(reference,'reference')
count_row_from_db(with_alias,'with_alias')
count_row_from_db(alias_id,'alias_id')

Tabella paper:
Nr righe non nulle per paper_id = 4894063
Nr righe non nulle per title = 4894063
Nr righe non nulle per year = 4894063
Nr righe non nulle per n_citation = 4894063
Nr righe non nulle per page_start = 4356226
Nr righe non nulle per page_end = 4112329
Nr righe non nulle per doc_type = 4394622
Nr righe non nulle per publisher = 4134190
Nr righe non nulle per volume = 2183176
Nr righe non nulle per issue = 1618242
Nr righe non nulle per doi = 3920939
Nr righe non nulle per abstract = 4232504
Nr righe non nulle per nb_references = 4894063
Nr righe non nulle per venue_id_new = 4820973

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4894063 entries, 0 to 4894062
Data columns (total 14 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   paper_id       4894063 non-null  int64
 1   title          4894063 non-null  object
 2   year           4894063 non-null  int16
 3   n_citation     4894063 non-null  int16
```

```
 4   page_start     4356226 non-null  object
 5   page_end       4112329 non-null  object
 6   doc_type       4394622 non-null  category
 7   publisher      4134190 non-null  object
 8   volume         2183176 non-null  Int64
 9   issue          1618242 non-null  Int64
 10  doi            3920939 non-null  object
 11  abstract       4232504 non-null  object
 12  nb_references  4894063 non-null  int16
 13  venue_id_new   4820973 non-null  Int64
dtypes: Int64(3), category(1), int16(3), int64(1), object(6)
memory usage: 457.4+ MB
```

Tabella venue:
Nr righe non nulle per venue_id = 10480
Nr righe non nulle per venue_raw = 49201
Nr righe non nulle per venue_type = 10482
Nr righe non nulle per venue_id_new = 49202

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49202 entries, 0 to 49201
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   venue_id      10480 non-null  Int64
 1   venue_raw     49201 non-null  object
 2   venue_type    10482 non-null  category
 3   venue_id_new  49202 non-null  Int64
dtypes: Int64(2), category(1), object(1)
memory usage: 1.3+ MB
```

Tabella reference:
Nr righe non nulle per paper_id = 45564149
Nr righe non nulle per referenced = 45564149

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45564149 entries, 0 to 45564148
Data columns (total 2 columns):
 #   Column      Non-Null Count     Dtype
---  ------      --------------     -----
 0   paper_id    45564149 non-null  int64
 1   references  45564149 non-null  int64
dtypes: int64(2)
memory usage: 695.3 MB
```

Tabella with_alias:
Nr righe non nulle per paper_id = 77909
Nr righe non nulle per alias_ids = 77909

```
<class ‘pandas.core.frame.DataFrame'>
RangeIndex: 77909 entries, 0 to 77908
Data columns (total 2 columns):
#   Column     Non-Null Count  Dtype
---  ------     --------------  -----
```

```
0   paper_id   77909 non-null  int64
1   alias_ids  77909 non-null  int64
dtypes: int64(1), object(1)
memory usage: 1.2+ MB
```

```
Tabella alias_id:
Nr righe non nulle = 77909
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 77909 entries, 0 to 77908
Series name: alias_ids
Non-Null Count  Dtype
--------------  -----
77909 non-null  int64
dtypes: int64(1)
memory usage: 608.8 KB
```

```
count_row_from_db(author,'author')
count_row_from_db(author_extended,'author_extended')
count_row_from_db(who,'who')
```

```
Tabella author:
Nr righe non nulle per author_id = 4398138
Nr righe non nulle per author_name = 4398138
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4398138 entries, 0 to 4398137
Data columns (total 2 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   author_id    4398138 non-null  int64
 1   author_name  4398138 non-null  object
dtypes: int64(1), object(1)
memory usage: 67.1+ MB
```

```
Tabella author_extended:
Nr righe non nulle per author_id = 5070852
Nr righe non nulle per author_name = 5070852
```

```
<class ‘pandas.core.frame.DataFrame’>
RangeIndex: 5070852 entries, 0 to 5070851
Data columns (total 2 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   author_id    5070852 non-null  int64
 1   author_name  5070852 non-null  object
dtypes: int64(1), object(1)
memory usage: 77.4+ MB
```

```
Tabella who:
Nr righe non nulle per paper_id = 14934850
Nr righe non nulle per author_id = 14934850
```

```
Nr righe non nulle per author_org = 11361508

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14934850 entries, 0 to 14934849
Data columns (total 3 columns):
 #   Column      Non-Null Count     Dtype
---  ------      --------------     -----
 0   paper_id    14934850 non-null  int64
 1   author_id   14934850 non-null  int64
 2   author_org  11361508 non-null  object
dtypes: int64(2), object(1)
memory usage: 341.8+ MB


count_row_from_db(fos,'fos')
count_row_from_db(what,'what')

Tabella fos:
Nr righe non nulle per fos_name = 132337
Nr righe non nulle per fos_id = 132337

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132337 entries, 0 to 132336
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   fos_name  132337 non-null  object
 1   fos_id    132337 non-null  int64
dtypes: int64(1), object(1)
memory usage: 2.0+ MB


Tabella what:
Nr righe non nulle per paper_id = 45029748
Nr righe non nulle per fos_id = 45029748
Nr righe non nulle per fos_w = 45029748

<class 'pandas.core.frame.DataFrame'>
Int64Index: 45029748 entries, 0 to 45029747
Data columns (total 3 columns):
 #   Column    Non-Null Count     Dtype
---  ------    --------------     -----
 0   paper_id  45029748 non-null  int64
 1   fos_id    45029748 non-null  int64
 2   fos_w     45029748 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 1.3 GB
```

5. Conclusioni

Lo scopo di questo elaborato è stato quello di progettare ed implementare un database che modelli il dataset fornito dal progetto SimpleText.
Il tempo speso nella progettazione e nella realizzazione del database è stato molto inferiore rispetto al carico di lavoro svolto per analizzare il dataset ed elaborarlo nelle strutture dati appropriate, soprattutto a causa della mancanza di confidenza con il linguaggio Python e gli strumenti della libreria Pandas.
La scelta di utilizzare Python e Pandas è stata dettata dal fatto di essere considerati gli strumenti principi nel campo della Data Analysis.
Prima di questo lavoro non mi ero mai avvicinata al mondo di Python e delle sue librerie, ma si è rivelato un linguaggio semplice da apprendere e allo stesso tempo complesso e prezioso per le sue enormi potenzialità. La libreria Pandas stessa si è rivelata una vera miniera d'oro per questo progetto soprattutto grazie alla sua ricca documentazione.
Ho avuto modo di apprendere e soprattutto di utilizzare nuovi linguaggi e strumenti, di far fronte alla grande dimensione del dataset e alle limitate capacità del mio calcolatore ottimizzando il codice e di affrontare la problematica della gestione dei dati "sporchi" all'interno del dataset.
Difatti il dataset fornito presenta non poche inconsistenze che sono state gestite tramite diverse scelte progettuali in fase di implementazione del database, tra le quali gli aliases dei nomi degli autori, poiché tale problematica esulava dallo scopo di questo elaborato, si è deciso di mantenere il dato completo attraverso l'uso di una tabella apposita, nonostante questo sprechi notevole spazio.
Un lavoro futuro potrebbe essere quello di normalizzare il nome degli autori, mantenendo comunque in una tabella a parte i suoi aliases, anziché selezionare solo la prima occorrenza tra i duplicati degli id degli autori. Altri potrebbero riguardare l'analisi dell'influenza della pubblicazione o dello stesso autore sulla base del numero di citazioni, o la progettazione di un'interfaccia utente per la consultazione del database stesso, le possibilità sono infinite.
Questa tesi è stata un'esperienza davvero preziosa, sia per la crescita delle competenze individuali, sia per avermi fatto riscoprire la passione per la programmazione.

6. Bibliografia e Sitografia

[1] Citation Network Dataset: https://www.aminer.org/citation

[2] Pandas Documentation: https://pandas.pydata.org/docs/reference/index.html#api

[3] Progetto SimpleText: https://simpletext-project.com/2022/clef/en/

[4] Pubblicazione scientifica: https://it.wikipedia.org/wiki/Pubblicazione_scientifica

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

[6] Giorgio Maria Di Nunzio, Eleonora Di Buccio. *Basi di Dati. Progettazione Concettuale, Logica e SQL.* Bologna: Esculapio, 2017.

[7] Ermakova, L., Bellot, P., Kamps, J., Nurbakova, D., Ovchinnikova, I., SanJuan, E., Mathurin, E., Araújo, S., Hannachi, R., Huet, S., & Poinsu, N. (2022). *Automatic Simplification of Scientific Texts: SimpleText Lab at CLEF-2022.* In M. Hagen, S. Verberne, C. Macdonald, C. Seifert, K. Balog, K. Nørvåg, & V. Setty (Eds.), Advances in Information Retrieval (Vol. 13186, pp. 364–373). Springer International Publishing. https://doi.org/10.1007/978-3-030-99739-7_46

[8] SQLAlchemy Documentation: https://docs.sqlalchemy.org/en/13/index.html

[9] Python Documentation: https://docs.python.org/3.9/

[10] Anaconda Distribution: https://www.anaconda.com/products/distribution

[11] PostgreSQL Documentation: https://www.postgresql.org/docs/14/index.html

[12] Json: https://www.json.org/json-it.html