UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"RACCOLTA E ANALISI DI DATI TRAMITE WEB SCRAPING. IL CASO DI STUDIO DI UN DIZIONARIO ONLINE."

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Davide Longo

ANNO ACCADEMICO 2021 – 2022

Data di laurea 14/11/2022

Indice

Introduzione

La tecnica del web scraping fornisce a chi la utilizza dei vantaggi non indifferenti, poiché permette di automatizzare operazioni tediose che richiederebbero normalmente considerevoli quantità di tempo per un essere umano. Di conseguenza, è possibile sfruttare proprio questo metodo per riuscire a raccogliere un gran numero di dati in maniera automatica e senza particolare sforzo. L'obiettivo principale della tesi è l'estrazione e il salvataggio dei vocaboli di un dizionario online, contenente termini in spagnolo relativi al commercio medioevale, disponibile al seguente link: https://www.um.es/lexico-comercio-medieval/index.php/. È sottointeso che è necessario trovare un modo per poter raggiungere l'obiettivo senza dover manualmente ricercare e conservare i dati desiderati. Considerato ciò, risulta evidente che un web scraper, ossia uno strumento che si basa sulla tecnica del web scraping, sia la scelta più adatta per questo scopo.

Per conseguire allora tale obiettivo, nella tesi verrà innanzitutto introdotto l'argomento del web scraping in maniera tale che il lettore possa comprendere di cosa esso si tratti. Inoltre, verranno descritti gli strumenti e le tecniche da affiancare, se necessario, ad un web scraper e, in particolare, le fasi che compongono il processo di creazione di un web scraper, da seguire per garantire che il web scraper lavori correttamente. Infine, è bene anche discutere dei rischi legati alla legalità dei web scraper, considerando le azioni da evitare quando si decide di utilizzare questo strumento.

Terminato questo argomento iniziale, ci si concentrerà in seguito alla realizzazione dei due elementi necessari per il raggiungimento dell'obiettivo della tesi. Il primo di questi è il database: a causa dell'attività didattica relativa a questo elaborato, ossia Basi di Dati, è stata scelta come struttura dati il database, che fungerà da contenitore dei vocaboli del dizionario online. Verrà posta particolare attenzione alla progettazione della base di dati, dal punto di vista concettuale, logico e fisico, e alla sua implementazione tramite il RDBMS MySQL. Fatto questo, verrà quindi realizzato il web scraper, il secondo importante elemento, utilizzando come linguaggio di programmazione Python. Il codice sorgente verrà analizzato per una maggior comprensione del comportamento del programma; sarà anche possibile consultare direttamente il codice sorgente completo, privo di spiegazioni.

Concluse le implementazioni di database e web scraper, verranno infine osservati i risultati ottenuti, anche in termini di tempo, e verranno fatte alcune considerazioni finali in merito al web scraping, individuandone anche gli aspetti problematici.

1 Web scraping e i suoi aspetti

1.1 Definizione

Per fornire una definizione appropriata di web scraping è necessario porre una certa attenzione per quanto riguarda le diverse interpretazioni reperibili, in quanto sono presenti discrepanze e imprecisioni. È bene, dunque, cominciare dalla definizione di data scraping: si tratta di una tecnica che ha come obiettivo l'estrazione di dati da un documento per mezzo di un programma. Essendo il web scraping una specializzazione del data scraping è ragionevole pensare che il primo abbia le stesse caratteristiche del secondo, con la sola differenza che il web scraping si effettua esclusivamente su pagine web.

Ora che si dispone di una definizione che, seppur concisa, racchiude il concetto fondamentale del web scraping, è possibile in particolare chiarire alcuni aspetti:

- La semplice copia di certe informazioni da un sito web, per mano di un qualsiasi utente umano, non può essere considerata web scraping. Infatti, non vi è solo una contraddizione della definizione fornita in precedenza, in cui si specifica l'uso di un programma per l'estrazione dei dati, ma bisogna tenere conto della quantità di dati su cui si deve operare. L'esigenza di avere web scraper, ossia di avere strumenti in grado di effettuare web scraping, nasce proprio a causa dell'enorme quantità di informazioni da estrarre che per un utente umano risulta essere una mole di lavoro altrettanto grande. Si può quindi fare affidamento ad un mezzo automatizzato, che in poco tempo può compiere tutte le operazioni necessarie ed ottenere lo stesso risultato che si sarebbe potuto ottenere senza l'uso di un web scraper ma in un tempo largamente maggiore. Per questo motivo l'estrazione di dati avvenuta manualmente sul web non è da ritenersi web scraping, in quanto i dati raccolti sono generalmente sufficientemente esigui e lo sforzo per ottenerli non giustifica l'utilizzo di un web scraper.
- È abbastanza comune trovare delle interpretazioni che includono come parte del processo di web scraping una o più delle fasi che precedono o seguono il web scraping. Tali fasi si riassumono principalmente in web crawling e formattazione dei dati, approfonditi nella sezione 1.2, ed evidentemente è sbagliato considerare anch'essi web scraping poiché risultano completamente diversi rispetto alla definizione di web scraping fornita. Nonostante ciò, rimane comunque un legame stretto tra web scraping e le sue ulteriori fasi, che rimangono di vitale importanza per permettere l'effettiva estrazione dei dati e il corretto impiego di essi.

1.2 Tecniche da affiancare

Affinché l'operato di un web scraper sia utilizzabile, può essere necessario affiancare al web scraper altri strumenti. Il primo fra tutti è un web crawler: esso è definito come un bot in grado di navigare nel web; dunque, si tratta di un programma che in automatico visita siti web tramite gli URL a sua disposizione. A partire da un insieme di URL iniziali, anche detto seed set, il web crawler ispeziona le pagine web, individuando eventuali link ad altri siti. Se tali link sono presenti, gli URL vengono aggiunti alla lista di pagine web ancora da visitare, chiamata crawl frontier. È importante evitare di inserire in tale lista gli URL già visitati, altrimenti si potrebbe potenzialmente generare un loop. Quando il web crawler ha terminato l'ispezione del sito web viene scelto, secondo certi criteri, l'URL successivo da visitare e così via. Generalmente i web crawler sono impiegati dai motori di ricerca per indicizzare pagine web e scaricarne i contenuti; nell'ambito del web scraping invece l'obiettivo di un web crawler è quello di ottenere una lista di URL che il web scraper utilizzerà successivamente per estrarre i dati rilevanti. Perciò è bene che il criterio utilizzato per individuare gli URL di interesse, dai quali verranno estratte le informazioni desiderate tramite il web scraper, sia particolarmente efficiente, così che non siano presenti, alla fine di tutto il processo di web scraping, dati non pertinenti. L'uso di un web crawler per effettuare web scraping è necessario quando si desidera raccogliere dati non solo da una sola lista di URL (identificabile come seed set), ma anche da altri possibili siti rilevanti. Se invece la lista di URL, nota a priori, risulta sufficiente ed unico e solo oggetto di studio, allora non avviene web crawling. Per cui, bisogna valutare se si dispone di un numero adeguato di URL oppure affidarsi ad un web crawler per ottenere ulteriori pagine da sottoporre al web scraper.

Spesso web crawling e web scraping non sono però sufficienti per poter ottenere i dati desiderati, in quanto sono generalmente necessarie ulteriori tecniche da usare. Una di queste non è molto comune, perché spesso incorporata durante il web scraping: si tratta dell'estrazione di dati a partire dal risultato del web scraper. Infatti, potrebbe servire scremare ulteriormente le informazioni ottenute durante il web scraping. In effetti, poiché un web scraper lavora con file HTML, esso si può dedicare esclusivamente, per esempio, all'estrazione del solo testo che compare a schermo nelle pagine web, ignorando tutto quello che ha a che vedere con il linguaggio di markup. Successivamente, si ricavano le informazioni desiderate a partire da un semplice testo ottenuto tramite il web scraper. Detto ciò, quest'ultima scrematura può facilmente essere incorporata all'interno del web scraper; non solo, ma proprio l'analisi della pagina web permette di individuare aspetti che si possono perdere poiché legati, ad esempio, a

come è strutturato il file HTML. Per questo motivo è consigliabile limitare l'estrazione di dati al solo web scraper.

La formattazione dei dati ottenuti risulta, invece, una fase molto importante, che dovrebbe sempre essere presente se si parla di web scraping. La possibilità di organizzare in maniera ordinata e coerente i dati permette di consultare ed eventualmente analizzare l'operato di un web scraper in maniera più semplice. Per realizzare ciò, viene solitamente scelto uno di questi formati:

- Comma Separeted Value (CSV): ideale per dati bidimensionali, quindi organizzabili in righe e colonne. È un formato abbastanza comune e facilmente visualizzabile.
- EXtensive Markup Language (XML): si tratta di un formato che usa un linguaggio di markup, pensato per il trasferimento e la memorizzazione dei dati. Dunque, è sconsigliato se lo si utilizza per visualizzarne il contenuto, poiché spesso molto complesso e conseguentemente difficile da comprendere.
- JavaScript Object Notation (JSON): a differenza del formato CSV, questo permette di lavorare con dati multidimensionali e semistrutturati, cioè che non si conformano alle classiche tabelle di dati bidimensionali. La comprensione dei file JSON è in genere più immediata e tale formato viene utilizzato anche in altri ambiti.
- Database: basato sul linguaggio SQL, Structured Query Language, per conservare i dati in questo formato è necessario che i dati da inserire nelle tabelle e i rispettivi attributi delle tabelle combacino in termini di tipo di dato, per evitare che le istanze non vengano inserite.

La scelta del formato dipende allora dalla tipologia dei dati e dell'uso che si vuole farne. In particolare, se si desidera analizzare i dati in proprio possesso, può essere ottimale la scelta del database come formato. La possibilità di formulare una query tramite il linguaggio SQL è certamente ideale, mentre, se non è in particolare necessaria alcuna analisi, si può optare per un formato CSV o JSON a seconda della complessità dei dati.

1.3 Realizzazione di un web scraper

Prima di analizzare i passaggi da seguire per costruire un web scraper, è estremamente importante tenere a mente alcune problematiche relative al web scraping: il primo ostacolo da superare è rappresentato dalla varietà di pagine web esistenti.

Sebbene possano essere presenti alcuni pattern ripetuti, ciascun sito web è diverso da qualsiasi altro, il che comporta a dover impiegare un web scraper diverso o modificare quello attuale nel caso in cui si decida di ispezionare siti web differenti. La seconda difficoltà invece è la volatilità delle pagine web: si intende di come il contenuto, ma anche la struttura, di una pagina web possa potenzialmente cambiare nel tempo, tanto da rendere inutilizzabile un web scraper e doverlo modificare opportunamente anche quando quest'ultimo risultava funzionante in precedenza. Si evince, quindi, che è necessaria una manutenzione costante del proprio web scraper, in modo che sia sempre possibile estrarre dati dagli stessi siti web e da quelli nuovi.

Queste considerazioni si collegano in particolar modo alla prima fase del web scraping: l'analisi delle pagine web. Con il termine "analisi" si intende l'ispezione dei file HTML delle pagine web in questione, ovvero, concretamente, l'individuazione delle informazioni da estrapolare all'interno dei siti web. Tale operazione è semplificata grazie ad una funzionalità presente in diversi browser, la quale consente di poter selezionare elementi di un sito web e di osservare come questi sono implementati nel codice sorgente della pagina. Questa analisi va effettuata per ciascuna pagina su cui si intende operare e, come specificato in precedenza, potrebbe essere necessario nel tempo compiere ulteriori analisi a causa delle possibili modifiche ai siti e alle loro strutture.

A questo punto è possibile procedere con la seconda fase, ossia la vera e propria estrazione dei dati. Innanzitutto, bisogna scaricare una copia della pagina web in esame, tramite una richiesta HTTP, in modo che il web scraper possa lavorarci direttamente. Il download potrebbe non essere necessario nel caso in cui venga impiegato un web crawler che, scaricando le pagine web, ha già in suo possesso il file HTML desiderato; risulta evidente che ciò è possibile solo nel caso in cui web crawler e web scraper possano comunicare tra loro. Inoltre, si sta presupponendo che la pagina in questione sia statica, ossia che essa è stata ottenuta da un file HTML. In caso contrario, la pagina viene costruita a partire da codice in JavaScript e per questo motivo si parla di pagina dinamica. In entrambi i casi, è consigliato l'uso di librerie per operare più facilmente con file HTML o per convertire da JavaScript a HTML. Nel caso del linguaggio Python, molto utilizzato nell'ambito del web scraping, le librerie principalmente impiegate sono Beautiful Soup, che aiuta lo sviluppatore nel parsing di documenti HTML, e Selenium, che permette di simulare il comportamento di un web browser, ottenere in fase di download file HTML a partire da pagine dinamiche e interagire con tali pagine, per esempio permettendo di cliccare su certi elementi della pagina. Con tali strumenti l'estrazione dei dati risulta più semplice, poiché in ogni caso è possibile lavorare su codice in HTML facilmente navigabile; in particolare, la ricerca di specifici tag, classi o attributi viene facilitata. Quest'ultima nozione è

specialmente importante nell'ambito del web scraping: poiché spesso i dati da estrarre sono incapsulati all'interno di tag che condividono tutti la stessa classe, si può non solo isolare il contenuto di tali tag, ma anche estrapolare solo le informazioni desiderate. Per esempio, se l'obiettivo di un web scraper è quello di ottenere alcune informazioni dei prodotti venduti in un e-shop, è possibile che i dati di ciascun prodotto si trovino in un contenitore definito dal tag:

Allora basterà cercare in tutti i contenitori la cui classe è "product" e, successivamente, estrarre i dati interessati, i quali a loro volta potrebbero essere individuati tramite il nome di un'altra classe.

Infine, l'ultima fase consiste nell'inserimento dei dati ottenuti all'interno di una base di dati o file CSV, XML o JSON, quindi, in genere, in uno dei formati descritti nella sezione 1.2. L'organizzazione dei dati è a discrezione dell'utente, che si adopererà per disporre i dati in maniera adatta alla lettura, nel caso in cui sia questo l'obiettivo, o affinché possano essere analizzati, nel caso, soprattutto, della base di dati. In quest'ultima eventualità, risulta evidente che è necessaria una preparazione previa alla realizzazione del web scraper, in quanto un database deve essere progettato, anche in funzione delle interrogazioni future, ed in seguito creato.

1.4 Legalità

Uno dei temi più discussi riguardo il web scraping, che provoca una certa preoccupazione, è la sua legalità e di conseguenza anche le possibili ripercussioni nei confronti di coloro che utilizzano web scraper. Di fatto l'uso di un web scraper non è in nessun modo illegale, ciò che invece potrebbe esserlo ha a che fare con il tipo di dato che viene estratto e di come esso viene impiegato. Ci si sta riferendo ai dati personali, che possono identificare un individuo sia direttamente che indirettamente.

In molti paesi vigono regolamentazioni in merito al trattamento dei dati personali, che includono, di fatto, anche il web scraping. Quest'ultimo è considerato legale fintanto che non sono estratti dati personali, ma non sempre questo è vero. Per comprendere meglio questo aspetto, verrà considerato come esempio il caso dell'Europa, dove è in vigore il GDPR, acronimo di General Data Protection Regulation. Secondo tale regolamento, qualsiasi tipo di "data processing", cioè di attività svolta su dati personali, può avvenire solo in certi casi, come descritto nell'articolo 6, che elenca sei possibili scenari in cui l'uso di dati personali è legittimo.

Nell'ambito del web scraping, è altamente probabile che solo uno di questi scenari si verifichi: tramite il consenso da parte dei soggetti interessati, l'estrazione di dati personali diventa legale. Rimane pur sempre vero, però, che se il consenso si limita alla sola estrazione ed analisi di dati personali, questi ultimi non potranno essere utilizzati in nessun altro modo che sia diverso da quelli precedentemente citati.

I rischi legati all'uso di web scraper non finiscono qui: numerose richieste ravvicinate ad un web server possono aumentare considerevolmente il traffico di dati. Se ciò dovesse succedere, il proprietario di tale server potrebbe ricorrere a vie legali nel caso in cui il web scraper sia visto come la causa di danni ad infrastrutture e come impedimento a certe operazioni. Questo scenario ricorda il processo del 2000 tra eBay e Bidder's Edge, considerato come uno dei primi processi dovuti al web scraping. eBay fece causa a Bidder's Edge perché il secondo, tramite il proprio web crawler (utilizzato assieme ad un web scraper), accedeva al sito web di eBay all'incirca 100000 volte al giorno. Sebbene il traffico generato a causa di Bidder's Edge non rappresentasse una minaccia, la corte decretò, in particolare, che il comportamento di Bidder's Edge potesse essere imitato da altri e che ciò avrebbe potuto veramente causare dei danni irreparabili. Questa decisione da parte della corte, risalente a più di venti anni fa, è ritenuta secondo certi punti di vista come datata e imprecisa, in quanto un web crawler non è veramente in grado di danneggiare beni fisici e dimostrare il contrario è piuttosto difficile. Rimane comunque vero che eccessive richieste a siti web potrebbero causare l'esclusione all'accesso delle pagine nei confronti di un web crawler o indirizzi IP. Per questo è bene consultare il file robots.txt prima di impiegare un web scraper, collocato nella root di un dominio (ad esempio www.google.com/robots.txt). Oltre a contenere istruzioni per gli user agent in merito agli URL o ai tipi di file di fatto accessibili, robots.txt può contenere anche un parametro "Crawl-delay", che indica i secondi che devono trascorrere tra ciascuna richiesta. Addirittura, il file robots.txt di certi siti web specifica che i bot non possono avere accesso a tali siti, a meno che non sia stato concesso a certi bot il permesso di accedere alle pagine.

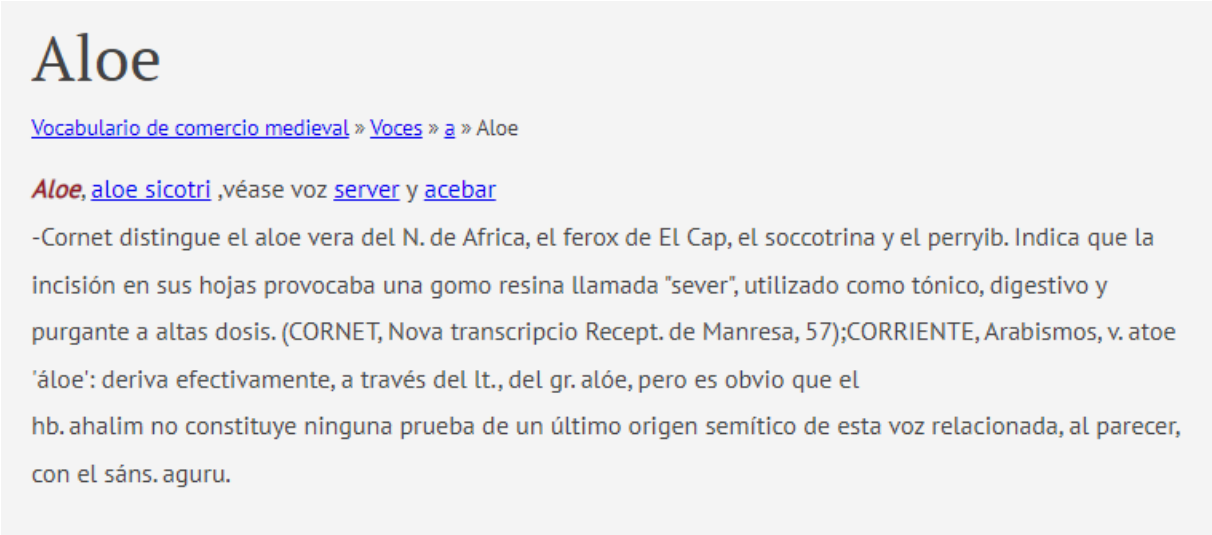
Infine, è bene consultare attentamente le condizioni d'uso e di servizio, poiché è possibile che sia vietato estrarre certi dati o impiegarli in certe maniere; anche in questo caso, può essere proibito ai bot di visitare i siti coinvolti da tale divieto. In generale, di prassi è vietato utilizzare web scraper per realizzare con i dati ottenuti un nuovo sito che offre lo stesso servizio del sito originale; il proprietario del sito che è stato duplicato si può sentire legittimato a fare causa al responsabile di questa duplicazione.

2 Creazione del database

2.1 Individuazione dei dati da estrarre

Dopo aver descritto il web scraping ed in particolare le sue fasi da seguire, è possibile ora iniziare con la realizzazione effettiva del web scraper. Ripercorrendo i passaggi illustrati nella sezione 1.3, la progettazione del database ha la precedenza e questo capitolo tratterà proprio questo aspetto. Detto ciò, è necessario prima di tutto osservare cosa compone ciascun termine del dizionario in esame.

Ciascuna parola ha una definizione ed appartiene ad una categoria (la definizione e la categoria potrebbero non essere presenti), inoltre potrebbero essere presenti alcuni elementi aggiuntivi. Tra di essi vi sono le citazioni, che dispongono di un titolo del documento da cui sono tratte ed un testo e possono essercene più di una oppure nessuna. Anche delle immagini di documenti relativi alla parola considerata potrebbero essere allegate. Infine potrebbe essere presente la bibliografia, che contiene un certo numero di fonti, le quali potrebbero avere un link che conduce al documento online. A seguire, verrano mostrati come esempio i vari elementi che compongono una parola, nel caso del termine “aloe”.



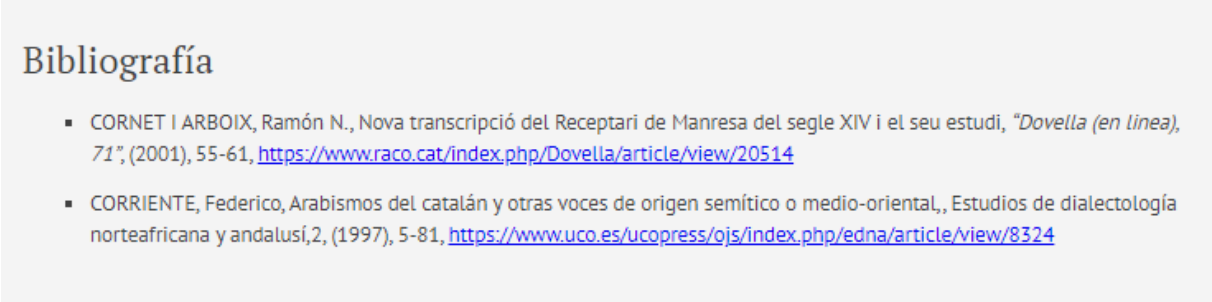
Aloe

Vocabulario de comercio medieval » Voces » a » Aloe

Aloe, aloe sicotri ,véase voz server y acebar

-Cornet distingue el aloe vera del N. de Africa, el ferox de El Cap, el soccotrina y el perryib. Indica que la incisión en sus hojas provocaba una gomo resina llamada "sever", utilizado como tónico, digestivo y purgante a altas dosis. (CORNET, Nova transcripcio Recept. de Manresa, 57);CORRIENTE, Arabismos, v. atoe 'áloe': deriva efectivamente, a través del lt., del gr. alóe, pero es obvio que el hb. ahalim no constituye ninguna prueba de un último origen semítico de esta voz relacionada, al parecer, con el sáns. aguru.

Figura 1: La parola "aloe" e la sua definizione



Bibliografía

- CORNET I ARBOIX, Ramón N., Nova transcripció del Receptari de Manresa del segle XIV i el seu estudi, *"Dovella (en linea), 71"*, (2001), 55-61, https://www.raco.cat/index.php/Dovella/article/view/20514
- CORRIENTE, Federico, Arabismos del catalán y otras voces de origen semítico o medio-oriental,, Estudios de dialectología norteafricana y andalusí,2, (1997), 5-81, https://www.uco.es/ucopress/ojs/index.php/edna/article/view/8324

Figura 2: Bibliografia della parola “aloe”

De: *GUAL CAMARENA, M. Vocabulario del comercio medieval. Colección de aranceles aduaneros de la Corona de Aragón (siglo XIII y XIV) , Tarragona (1968)*

Áloe, lignum (XIV, 8). — Cast. «palo áloe o agáloco» (árbol oriental cuya madera contiene un jugo acre y aromático). El «lignum áloe» es diferente del «áloe epático», de que hablan Capmany, Pegolotti y otras fuentes medievales. El áloe era sustancia amarga, empleada como purgativo. Del lat. «aloe».
Citas. «E aquí debía nascer garingal o gengibre, e pimienta, e cardamomo, e titoal, e girofle, e matis, e nuez moscada, e el madero áloe, e sándalo, e mirra, e encienso, e todas las buenas especias de Paraíso» (Conq. Ultram., edic. Gayangos, 322 b). «Lignum aloes aytal.la conexença (Bibl. Univ. Barc. Ms. núm. 4, fols. 23 v-24). «Lo rey los feu fer una molt bella tomba de lignum aloe, obrada molt artificialment» (Tirant, edic. Aguiló, I, 230).
Bibliografía. Capmanyt 2.a edic., I, 732-733. Bourquelot, Foires Champagne, I,289. Dic. Aguiló y Dic. catvalbal., v. «áloes» y «áloe (lignum)». Roudil, FBaeza; Dic. Histor. 1.a edic. y Dic. Corominas: v. «áloe». Steiger, Versos CBaena, 26. Du Cange, v. «aloa». Gay, Glossaire, v. «aloes» (amplia documentación). Evans, v. «legno aloe».

Tipo: Plantas medicinales

Figura 3: Citazione della parola "aloe", seguita dalla sua categoria

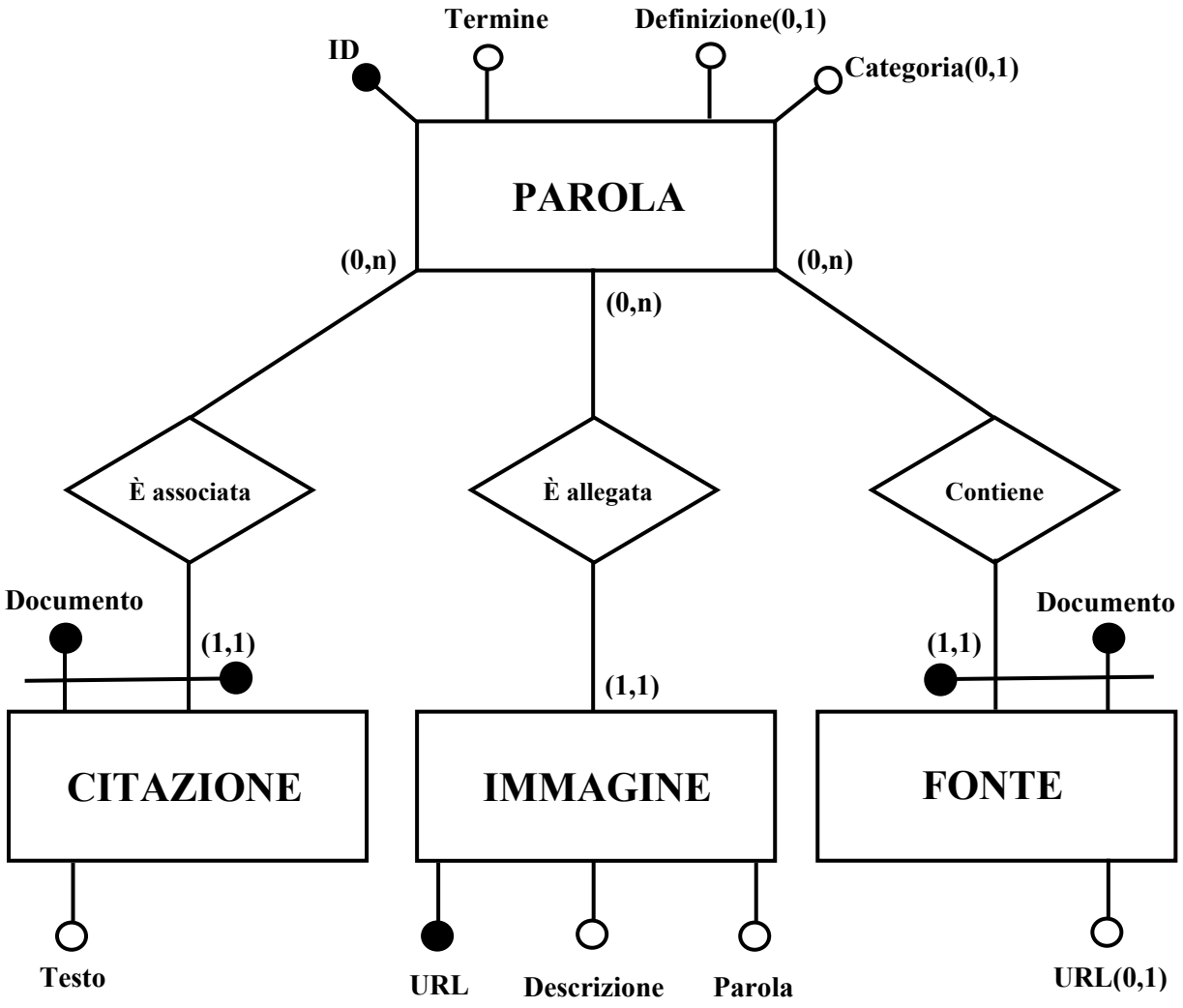
Referencias documentales de «Aloe»

Fichas de la voz «Aloe», extraídas del archivo del profesor Gual

Figura 4: Immagini dei documenti relativi alla parola "aloe"

2.2 Progettazione concettuale

Verrà in seguito mostrato lo schema concettuale della base di dati che si intende realizzare sulla base delle informazioni fornite nella sezione 2.1 e sulla base di alcune osservazioni che verranno esposte successivamente.



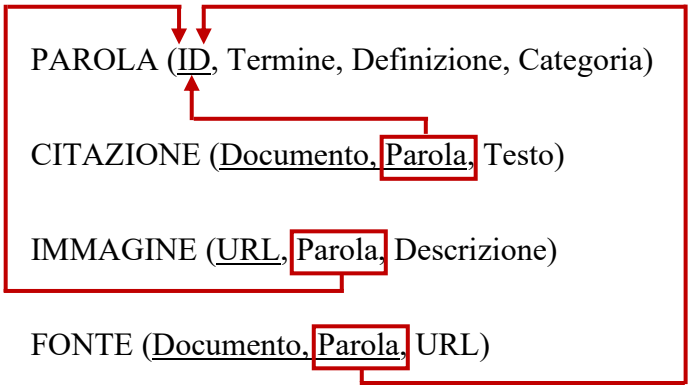
È bene fare alcune osservazioni, per chiarire alcuni aspetti:

- L'entità PAROLA è identificata tramite un ID e non da Termine, il quale rappresenta il semplice vocabolo (come per esempio "Aloe"), in quanto se fossero presenti più parole con lo stesso termine, non verrebbero inserite nel database. Sarebbe stato possibile scegliere come identificatore la coppia di attributi Termine e Definizione, ma la scelta di avere solo un ID come identificatore è più intuitiva ed indicata, soprattutto in previsione dell'inserimento delle tuple e delle interrogazioni.

- Le entità CITAZIONE e FONTE non possono essere identificate con il solo attributo Documento, che indica il documento dal quale sono tratte le citazioni o le fonti, perché se fossero presenti più di una fonte o citazione con lo stesso attributo Documento ma legate a parole diverse, solo una tupla verrebbe, erroneamente, inserita. Identificando queste entità con la coppia di attributi Documento e ID dell'entità PAROLA, si garantisce di non perdere alcuna informazione; per questo motivo CITAZIONE e FONTE sono due entità deboli. Contrariamente al punto precedente, si è scelto di non utilizzare un ID anche per le entità CITAZIONE e FONTE: dopo aver osservato una certa quantità di parole nel dizionario online, si è notata una presenza di fonti abbastanza ridotta, in particolare delle citazioni. Identificare allora queste entità con un ID è risultato allora piuttosto superfluo e si è quindi optato per una soluzione diversa, ossia quella proposta.
- L'entità IMMAGINE utilizza URL come identificatore, in quanto il path dell'URL è unico per ciascuna immagine. Per questo motivo IMMAGINE non è un'entità debole, ma è rimane necessario avere un attributo che specifichi a quale parola ciascuna immagine è associata; l'attributo in questione è Parola.
- L'attributo URL dell'entità FONTE è opzionale perché una fonte non necessariamente ha un link che rimanda ad un documento online. Anche l'attributo Categoria e Definizione dell'entità PAROLA sono opzionali, in quanto sono state individuate nel dizionario online parole non categorizzate e senza definizione.
- Ad ogni parola possono essere associate da 0 a n citazioni, immagini e fonti, perché non è obbligatorio che citazioni, immagini e fonti siano presenti. Nel caso opposto, in quanto entità deboli, ogni citazione e fonte si associano ad una sola parola e similmente anche ogni immagine si associa ad una parola sola, perché l'identificatore di ciascuna immagine, l'URL, contiene un ID che identifica la parola nel dizionario online. Ogni immagine è quindi legata alla parola e tutte le immagini che si associano ad una parola sono distinte l'una dall'altra: per questo motivo ogni immagine si associa ad una sola parola.

2.3 Progettazione logica

Disponendo dello schema concettuale, è possibile realizzare lo schema logico.



Anche in questo caso, si possono considerare alcuni aspetti importanti:

- Le associazioni dello schema concettuale non dispongono di una schema relazionale poiché tutte le associazioni sono binarie e hanno entità che partecipano con cardinalità (0,n) e (1,1). Infatti, in questa situazione gli schemi relazionali che rappresentano le associazioni risulterebbero ridondanti ed è semplicemente necessario aggiungere Parola come chiave esterna allo schema relazionale di IMMAGINE. Per CITAZIONE e FONTE, Parola è già presente perché necessaria per la chiave primaria di tali schemi relazionali; ad ogni modo, Parola è una chiave esterna perché, in particolare, CITAZIONE e FONTE erano in origine entità deboli nello schema concettuale.
- Gli attributi che nello schema concettuale erano opzionali rimangono tali anche nello schema logico. Nello specifico sono opzionali gli attributi Categoria e Definizione dello schema PAROLA e URL dello schema FONTE.

2.4 Progettazione fisica

Dallo schema logico è ora possibile ricavare le istruzioni SQL per la creazione del database.

```
CREATE TABLE Parola (
  ID INTEGER NOT NULL,
  Termine VARCHAR(50),
  Definizione TEXT,
  Categoria VARCHAR(50),
  PRIMARY KEY (ID)
);
```

```
CREATE TABLE Citazione (
  Documento VARCHAR(200) NOT NULL,
  Parola INTEGER NOT NULL,
  Testo TEXT,
  PRIMARY KEY (Documento,Parola),
  FOREIGN KEY (Parola) REFERENCES Parola(ID)
    ON DELETE CASCADE ON UPDATE CASCADE
);



CREATE TABLE Immagine (
  URL VARCHAR(200) NOT NULL,
  Parola INTEGER NOT NULL,
  Descrizione TEXT,
  PRIMARY KEY (URL),
  FOREIGN KEY (Parola) REFERENCES Parola(ID)
    ON DELETE CASCADE ON UPDATE CASCADE
);



CREATE TABLE Fonte (
  Documento VARCHAR(400) NOT NULL,
  Parola INTEGER NOT NULL,
  URL VARCHAR(200),
  PRIMARY KEY (Documento,Parola),
  FOREIGN KEY (Parola) REFERENCES Parola(ID)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

Verranno nuovamente fatte alcune considerazioni in merito alle scelte implementative del database:

- Tutte le chiavi esterne hanno lo stesso comportamento in caso di eliminazione o aggiornamento delle righe della tabella Parola. Sebbene la scelta di avere la cancellazione a cascata di tutte le tuple legate da un vincolo referenziale, a seguito dell'eliminazione di una parola, sia potenzialmente rischiosa, tale decisione è in realtà appropriata, perché altrimenti non verrebbero rispettati i vincoli del modello e inter-relazionali. Infatti, nel caso di `ON DELETE NO ACTION` le chiavi esterne rimarrebbero invariate, ma ciò significa che tali chiavi contengono un valore che non è presente nel tabella Parola come attributo ID. Questo violerebbe il vincolo inter-relazionale:

  $$\exists\, t_1 \in r_1(R_1),\, t_2 \in r_2(R_2) \mid t_1[FK] = t_2[K]$$

  dove R_1 è la tabella referente, r_1 è l'insieme delle istanze di R_1, t_1 è una tupla contenuta in r_1, R_2 è la tabella riferita, r_2 è l'insieme delle istanze di R_2, t_2 è una tupla contenuta in r_2, FK è l'insieme di attributi che costituisce la chiave esterna e K è l'insieme di attributi che costituisce la chiave a cui si riferisce la chiave esterna.

  Se invece si avesse `ON DELETE SET NULL`, le chiavi esterne di Citazione e Fonte diventerebbero nulle, cosa non consentita poiché i loro attributi Parola sono un intero non nullo. In aggiunta, potrebbe non venire rispettato il vincolo di relazione, secondo il quale ogni istanza di relazione possiede tuple tutte distinte: infatti, se Parola fosse nulla, potrebbero poi essere presenti tuple identiche, in quanto le istanze di entrambe le tabelle sono identificate tramite la coppia di attributi Documento, Parola.

  È ora evidente che la soluzione proposta sia l'unica corretta. Risulta anche essere la più sensata: se una parola viene eliminata da un dizionario, anche tutti i possibili contenuti aggiuntivi collegati ad essa dovrebbero essere cancellati, poiché non hanno senso di esistere se la parola a cui sono legati non è presente.
- Gli attributi opzionali specificati nello schema logico sono rimasti tali, in quanto questi attributi possono anche assumere valore nullo.

2.5 Implementazione

Per la creazione effettiva del database, è stato scelto come RDBMS MySQL, poiché utilizza un driver chiamato MySQL Connector/Python, che permetterà di connettersi ad un database MySQL direttamente dal codice sorgente del web scraper, realizzato in Python. Di seguito verranno mostrate le istruzioni per creare un database MySQL basandosi sulle istruzioni SQL presenti nella sezione precedente. Inoltre, si presuppone che il nome del database sia "dizionario".

```
CREATE TABLE `dizionario`.`parola` (
  `id` MEDIUMINT(5) NOT NULL,
  `termine` VARCHAR(50) NULL,
  `definizione` MEDIUMTEXT NULL,
  `categoria` VARCHAR(50) NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `dizionario`.`citazione` (
  `documento` VARCHAR(200) NOT NULL,
  `parola` MEDIUMINT(5) NOT NULL,
  `testo` MEDIUMTEXT NULL,
  PRIMARY KEY (`documento`, `parola`),
  INDEX `parola_idx` (`parola` ASC) VISIBLE,
  CONSTRAINT `cit_parola`
    FOREIGN KEY (`parola`)
    REFERENCES `dizionario`.`parola` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE `dizionario`.`immagine` (
  `URL` VARCHAR(200) NOT NULL,
  `parola` MEDIUMINT(5) NOT NULL,
  `descrizione` TINYTEXT NULL,
  PRIMARY KEY (`URL`),
  INDEX `parola_idx` (`parola` ASC) VISIBLE,
  CONSTRAINT `imm_parola`
    FOREIGN KEY (`parola`)
    REFERENCES `dizionario`.`parola` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE `dizionario`.`fonte` (
  `documento` VARCHAR(400) NOT NULL,
  `parola` MEDIUMINT(5) NOT NULL,
  `URL` VARCHAR(200) NULL,
  PRIMARY KEY (`documento`, `parola`),
  INDEX `parola_idx` (`parola` ASC) VISIBLE,
  CONSTRAINT `fon_parola`
    FOREIGN KEY (`parola`)
    REFERENCES `dizionario`.`parola` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE)
;
```

Sono presenti delle evidenti modifiche rispetto alle istruzioni SQL della sezione precedente, in particolare:

- L'identificatore della tabella "parola" (e di conseguenza anche tutte le chiavi esterne delle altre tabelle) è un MEDIUMINT, che corrisponde ad un numero intero di dimensione massima di 3 byte. Uno SMALLINT (2 byte di dimensione) poteva forse essere sufficiente per conservare l'ID di ciascuna parola, ma si è optato per un MEDIUMINT per motivi che verranno spiegati nella sezione 3.1.
- I TEXT sono stati sostituiti con MEDIUMTEXT o TINYTEXT che hanno dimensione massima rispettivamente di 2^{24} e 2^{8} byte. A seconda dell'attributo sono stati scelti tipi diversi; per esempio, per l'attributo "definizione" della tabella "parola" è stato scelto un MEDIUMTEXT per garantire che sia sufficientemente capiente per contenere la definizione di un vocabolo, che potrebbe essere particolarmente lunga.
- Se una tabella contiene una chiave esterna sarà anche presente un INDEX, che in questo caso permette l'accesso alla chiave a cui fa riferimento una chiave esterna in maniera rapida ed efficiente. Sono inoltre presenti dei CONSTRAINT, che specificano quali dati sono consentiti; aggiungere una chiave esterna implica che quest'ultima può assumere solo certi valori, dunque c'è bisogno di un CONSTRAINT.

3 Creazione del web scraper

3.1 Analisi delle pagine web

Avendo realizzato il database, è ora possibile seguire le tre fasi necessarie per la realizzazione di un web scraper, descritte nella sezione 1.3. La prima di queste è l'analisi delle pagine web, che ha come fine l'individuazione dei dati interessati all'interno dei documenti HTML che costituiscono le pagine web. Gli attributi di ciascuna tabella del database verranno quindi ricavati soprattutto tramite le classi con cui è possibile distinguere i tag uguali di un documento HTML (per esempio, i tag <p> possono essere distinti tramite la loro classe).

Parola:

- ID: per la scelta dell'ID si è deciso di utilizzare lo stesso ID con cui vengono identificate le parole nel sito web. Per esempio, l'ID della parola "Aloe" è 7432 e verrà utilizzato tale valore per identificare le parole nel database. Sarebbe stato ugualmente possibile scegliere un intero autoincrementale, in tal caso uno SMALLINT sarebbe stato sufficiente per contenere tale valore. Nonostante questo, per mantenere una certa coerenza con il dizionario online, sarà scelto l'ID originale come identificatore della parola. Per poter garantire che ogni parola sia inserita, è necessario che l'attributo ID possa anche contenere interi da 3 byte, poiché anche un UNSIGNED SMALLINT potrebbe non essere sufficiente per gli ID che superano il valore 65535.
- Termine: si può individuare all'interno di un tag <h3>, appartenente alla classe "lexema_title".
- Definizione: si può individuare all'interno di un tag <p>, appartenente alla classe "descripcion". È importante tenere in considerazione che anche il testo di una citazione è contenuto nello stesso elemento, ma esso deve essere preceduto da un tag <h4>, appartenente alla classe "complementaria", che rappresenta il documento da cui è tratta la citazione. Poiché una citazione non si trova mai prima della definizione, si può assumere che il primo tag <p class="descripcion"> definisce il paragrafo relativo alla definizione della parola, purché tale definizione sia effettivamente presente.
- Categoria: si può individuare all'interno di un tag , appartenente alla classe "tipo".

Citazione:

- Documento: si può individuare all'interno di un tag <h4>, appartenente alla classe "complementaria", contenuto in un tag .
- Parola: è una chiave esterna che utilizza come valore l'ID di una parola.
- Testo: si può individuare all'interno di un tag <p>, appartenente alla classe "descripcion". Come spiegato in precedenza, deve essere preceduto da un tag <h4>, appartenente alla classe "complementaria", poiché altrimenti si potrebbe erroneamente estrarre la definizione di una parola invece che il testo di una sua citazione.

Immagine:

- URL: si può individuare all'interno di un tag <p>, appartenente alla classe "imagen_lexema", il quale, a sua volta, tramite l'attributo href di un tag <a> specifica l'URL interessato.
- Parola: è una chiave esterna che utilizza come valore l'ID di una parola.
- Descrizione: si può individuare all'interno di un tag <p>, appartenente alla classe "imagen_lexema", il quale, a sua volta, tramite l'attributo alt di un tag <a> specifica la descrizione dell'immagine. Si è scelto di optare per questa soluzione per poter, in particolar modo, individuare l'origine dell'immagine (per esempio, l'immagine può essere fornita dalla fondazione Juan March e la descrizione tiene conto di ciò).

Fonte:

- Documento: si può individuare all'interno di un tag <li>; si tratta di un elemento di una lista definita dal tag <ul> appartenente alla classe "bibliografia". Ciascun documento è poi di fatto composto da più elementi , contenenti nome e cognome dell'autore, nome del documento, le pagine interessate ecc. Per ciascun elemento verranno estratti tutti questi aspetti e riuniti in una sola stringa, ad eccezione dell'elemento appartenente alla classe "coleccion", che rappresenta l'URL del documento.
- Parola: è una chiave esterna che utilizza come valore l'ID di una parola.
- URL: come descritto precedentemente, l'URL di un documento è contenuto nello , il quale si trova in un elemento di una lista <ul class="bibliografia">. All'interno di tale , è presente un tag <a> che specifica l'URL del documento, tramite il suo attributo href.

3.2 Analisi del web scraper

Con le informazioni ottenute nella sezione precedente si può ora procedere con la scrittura del codice sorgente del web scraper, che sarà realizzato in Python, in quanto dispone della libreria Beautiful Soup, che aiuta nel parsing di documenti HTML tramite, in particolare, la ricerca di tag e dei loro contenuti. Delle tre fasi descritte nella sezione 1.3, il web scraper comprende le ultime due fasi della tecnica del web scraping: sarà dunque in grado di estrarre i dati rilevanti e, in questo caso, inserire quanto ottenuto nel database implementato in precedenza. Successivamente, verranno riportate parti del codice sorgente del web scraper e soprattutto descritta come avviene l'estrazione dei dati. Se lo si desidera, è possibile consultare l'intero codice sorgente nella sezione successiva.

Innanzitutto, per ottenere il link per la parola successiva dopo aver raccolto tutte le informazioni necessarie della parola attuale sono principalmente presenti tre modi:

- Il primo consiste nello sfruttare un form per ricercare i vocaboli, tramite l'inserimento di una parola da utilizzare per compiere la ricerca, disponibile a questo link. Possono essere mostrati fino a 1000 risultati, che permettono l'accesso alla pagina web relativa ad una specifica parola. Dopo aver utilizzato questo metodo per la ricerca di parole, si è concluso che esso sia particolarmente inadatto per il web scraping: i risultati forniti dalla ricerca contengono molto spesso vocaboli non attinenti a quanto ricercato. Infatti, si è notato che, affinché una parola venga inclusa nei risultati della ricerca, è sufficiente che la sequenza di lettere inserita nel form sia presente nella definizione di tale parola. Di conseguenza, sono presenti molti risultati indesiderati e sarebbe necessario controllare ogni volta che deve essere inserita nel database una parola se quest'ultima è già presente, rendendo così il web scraper piuttosto inefficiente.
- Il secondo modo si basa invece sulla navigazione di pagine web di cui è possibile modificarne l'URL per accedere ad altre pagine pertinenti. Partendo da questo link, che fornisce l'accesso ad una pagina che contiene le prime 20 parole del dizionario online che cominciano con la lettera A, è possibile manipolare l'URL per spostarsi da una pagina all'altra, acquisendo man mano gli URL alle pagine contenenti i dati da estrarre. Similarmente, è anche possibile spostarsi da una pagina all'altra tramite i link presenti nella pagina attuale: il risultato ottenuto sarebbe comunque lo stesso.
- Il terzo metodo utilizza i link presenti all'interno delle pagine relative alle parole: per ogni parola è possibile accedere alla parola immediatamente precedente o successiva del dizionario online. Le uniche eccezioni sono rappresentate dalla prima e l'ultima

parola del dizionario, che possono accedere rispettivamente solo alla parola immediatamente successiva e immediatamente precedente.

Di questi tre metodi si è scelto di applicare il terzo, poiché permette di ridurre al minimo le richieste da inoltrare al server web e di ridurre di conseguenza il tempo necessario per estrarre tutti i dati. Infatti, il file robots.txt specifica un crawl-delay di 1 secondo, che equivale a dover impiegare più tempo se si sceglie un metodo che prevede più richieste rispetto ad un altro. Poiché il primo metodo è stato scartato e il terzo prevede meno richieste del secondo, il terzo metodo è stato scelto per l'implementazione del web scraper.

Precisato questo aspetto, verrà ora analizzato il codice sorgente: a seguito di ciascuna porzione di codice, verrà fornita una spiegazione sul funzionamento del codice.

```
import time
import requests
import mysql.connector
from bs4 import BeautifulSoup

#Dichiarazione delle variabili per l'inserimento

#Parola:
#id_parola, termine_parola, definizione_parola, categoria_parola

#Citazione:
#documento_citazione, parola_citazione, testo_citazione

#Immagine:
#URL_immagine, parola_immagine, descrizione_immagine

#Fonte:
#documento_fonte, parola_fonte, URL_fonte
```

Oltre alle librerie importate, questa porzione di codice contiene, commentate, le variabili che verranno utilizzate in fase di inserimento dei valori all'interno del database. Sebbene non sia necessario avere tutte queste variabili aggiuntive, considerato il linguaggio di programmazione, è bene in ogni caso fare in modo che il codice sia di più facile comprensione. Per questo motivo sono presenti tante variabili quanti gli attributi delle tabelle del database, ciascuna nominata con il nome dell'attributo, il simbolo "_" e il nome della tabella a cui appartiene.

```
page = requests.get("https://www.um.es/lexico-comercio-medieval/index.php/v/letra/a/")
soup = BeautifulSoup(page.content, "html.parser")
nextURL = soup.find("li", class_="resultado even").find("a")["href"]
```

Per poter iniziare estrazione, è necessario individuare la prima parola del dizionario. Per farlo, viene utilizzato l'URL https://www.um.es/lexico-comercio-medieval/index.php/v/letra/a/, che conduce alle prime 20 parole che cominciano per A del dizionario. Dunque, dopo aver convertito la pagina scaricata in un oggetto navigabile della libreria Beautiful Soup, è possibile cercare all'interno del documento HTML un tag specifico, appartenente ad una classe specifica, tramite il metodo find(), che cerca il primo tag con le caratteristiche specificate. Poiché le parole presenti in questa pagina sono elementi di una lista <ul> e il primo elemento di tale lista è sempre appartenente alla classe "resultado even", è possibile individuare facilmente la prima parola del dizionario ed in particolare il suo link, che conduce alla relativa pagina web. Qualsiasi URL è sempre individuabile tramite l'attributo "href" di un tag <a>.

```
connection = mysql.connector.connect(
  host="localhost",
  user="dbuser",
  password="dbpassword",
  database="dizionario"
)

cursor = connection.cursor()
```

Dopo aver individuato il primo URL, è necessario connettersi al database per poter successivamente inserire i dati estratti nelle tabelle. Tramite la libreria mysql.connector, è possibile utilizzare la funzione connect() per collegarsi al database, in cui si specifica il tipo di host, l'utente, la password dell'utente e il nome del database (nella porzione di codice sono presenti utente e password fittizi). Fatto ciò, si istanzia un oggetto cursor, che servirà per l'inserimento delle tuple.

```
while(nextURL != None):

    time.sleep(1)

    page = requests.get(nextURL)
    soup = BeautifulSoup(page.content, "html.parser")

   #Inserimento dei dati

    if(soup.find("span", class_="siguiente") == None):
        nextURL = None
    else:
        nextURL = soup.find("span", class_="siguiente").find("a")["href"]
```

A questo punto, in un ciclo while, avverrà l'estrazione e l'inserimento dei dati. Ciò è garantito dalle istruzioni presenti alla fine di ogni ciclo (in questo tratto di codice, le istruzioni a cui ci si riferisce si trovano dopo il commento): all'interno di un tag , appartenente alla classe "siguiente", è possibile trovare il link alla parola immediatamente successiva. Se tale tag è presente, significa che c'è ancora almeno un'altra parola, altrimenti, in caso contrario, non vi sono altre pagine web da scaricare e conseguentemente il ciclo while si interrompe. Inoltre, un altro aspetto importante è che in ogni ciclo è presente l'istruzione time.sleep(1), che blocca il processo per un secondo. L'attesa di un secondo è necessaria per poter garantire che il crawl-delay specificato nel file robots.txt sia sempre rispettato.

```
id_parola = nextURL.split("/")[len(nextURL.split("/"))-2]
termine_parola = soup.find("h3", class_="lexema_title").text
if(soup.find("p", class_="descripcion") != None):
    if(soup.find("nav", class_="breadcrumbs").next_sibling.next_sibling["class"] == ["descripcion"]):
        definizione_parola = soup.find("p", class_="descripcion").text
    else:
        definizione_parola = None
else:
    definizione_parola = None
if(soup.find("span", class_="tipo") == None):
    categoria_parola = None
else:
    categoria_parola = soup.find("span", class_="tipo").text
cursor.execute("INSERT INTO parola (id, termine, definizione, categoria) VALUES (%s, %s, %s, %s)",
            (id_parola, termine_parola, definizione_parola, categoria_parola))
```

Per prima cosa, vengono estratti i dati relativi alla tabella “parola”. L’id è ottenuto grazie all’URL della pagina: poiché ogni parola del dizionario ha un ID che è presente nel relativo URL, tale URL viene diviso in più elementi, ciascuno compreso tra i simboli “/”, ossia si individua, di fatto, ciò che compone il path dell’URL. Il penultimo elemento sarà sempre l’ID della parola, trovando quindi l’identificatore della tabella “parola”. In seguito, si ricava il termine che compone la parola, in base a quanto specificato nella sezione 3.1, e così avviene anche per la definizione e la categoria, i quali, essendo attributi opzionali, possono non essere presenti ed avviene di conseguenza un controllo aggiuntivo. Nel caso della definizione, la situazione è più complessa, perché la definizione di una parola condivide lo stesso tag e classe del testo di una citazione. Sebbene la definizione di una parola preceda sempre una citazione, è possibile che una parola sia sprovvista di definizione; dunque, cercare il primo tag <p>, appartenente alla classe “descripcion”, ed assegnare il suo contenuto alla definizione potrebbe essere sbagliato. Per assicurarsi che ciò non avvenga, si controlla se il tag successivo al tag <nav>, appartenente alla classe “breadcrumbs”, appartiene alla classe “descripcion”, in quanto, dopo tale tag <nav>, è possibile trovare la definizione. Se ciò è vero, allora la definizione è presente, se è falso non c’è alcuna definizione. Identificati tutti i valori che compongono una parola, grazie all’istruzione cursor.execute() viene specificata l’istruzione che inserirà i valori specificati nella tabella “parola” nel database MySQL. L’inserimento effettivo non avverrà fino a che non verrà eseguita l’istruzione connection.commit(), che sarà chiamata alla fine

dell'estrazione dei dati di ciascuna parola. Inoltre, se un valore è None, verrà considerato NULL all'interno del database.

```
complementariaIterable = soup.find_all("h4", class_="complementaria")
if(complementariaIterable != None):
    descripcionIterator = iter(soup.find_all("p", class_="descripcion"))
    if(definizione_parola != None):
        next(descripcionIterator)
    for cit in complementariaIterable:
        documento_citazione = cit.find("em").text
        parola_citazione = id_parola
        testo_citazione = next(descripcionIterator).text
        cursor.execute("INSERT INTO citazione (documento, parola, testo) VALUES (%s, %s, %s)",
                (documento_citazione, parola_citazione, testo_citazione))
```

Da questo punto in poi verrà utilizzato anche il metodo find_all(), che, a differenza di find(), trova tutti i tag specificati e restituisce un Iterable. Poiché il nome del documento di una citazione si trova in un tag <h4>, appartenente alla classe "complementaria", si effettua una ricerca in modo che vengano trovati tutti i documenti delle citazioni tramite find_all(). Se non ve ne sono, allora la pagina non contiene citazioni. In caso contrario è possibile sfruttare l'Iterable ottenuto tramite find_all() per realizzare un ciclo for in cui ad ogni ciclo viene estratto il nome del documento, presente all'interno di un tag . Prima di fare ciò, è necessario anche disporre di un Iterator a partire da un Iterable, ottenuto a sua volta da soup.find_all("p", class_="descripcion"). Tale Iterator verrà utilizzato per ricavare i testi delle citazioni, in quanto questi ultimi sono contenuti nei tag <p>, appartenenti alla classe "descripcion". Inoltre, l'Iterator dovrà saltare un elemento nel caso in cui sia presente una definizione, in quanto definizioni di una parola e testi di una citazione condividono lo stesso tag e il primo di questi tag è sempre una definizione, posto che una definizione sia presente. Precisati questi aspetti, i dati ricavati dalla pagina verranno successivamente inseriti nel database e saranno inserite tante citazioni quanti i cicli avvenuti.

```
imagen_lexemaIterable = soup.find_all("p", class_="imagen_lexema")
if(imagen_lexemaIterable != None):
    for img in imagen_lexemaIterable:
```

```
        URL_immagine = img.find("a")["href"]
        parola_immagine = id_parola
        descrizione_immagine = img.find("a")["alt"]
        cursor.execute("INSERT INTO immagine (URL, parola, descrizione) VALUES (%s, %s, %s)",
                (URL_immagine, parola_immagine, descrizione_immagine))
```

Come nel caso delle citazioni, anche le immagini verranno ricercate tramite find_all(); se find_all() restituisce None, non vi sono immagini associate alla parola, altrimenti per ogni elemento dell'Iterable ottenuto tramite find_all() verranno estratti dal tag <a> l'URL e la descrizione dell'immagine. Infine, per ogni immagine vi sarà un inserimento del database quando sarà chiamata connection.commit().

```
bibliografia = soup.find("ul", class_="bibliografia")
if(bibliografia != None):
    bibliografiaIterable = bibliografia.find_all("li")
    for src in bibliografiaIterable:
        documento_fonte = ""
        for elem in src.find_all("span"):
            if(elem.find("a") != None):
                continue
            documento_fonte += elem.text
            documento_fonte += ","
        documento_fonte = documento_fonte[:-1]
        parola_fonte = id_parola
        if(src.find("span", class_="coleccion") == None):
            URL_fonte = None
        else:
            URL_fonte = src.find("span", class_="coleccion").find("a")["href"]
        cursor.execute("INSERT INTO fonte (documento, parola, URL) VALUES (%s, %s, %s)",
                (documento_fonte, parola_fonte, URL_fonte))
```

Come nel caso delle citazioni e delle immagini, è necessario verificare che siano presenti le fonti, nello specifico all'interno di una lista definita dal tag <ul>, appartenente alla classe "bibliografia". In caso positivo, viene creato un Iterable tramite find_all() per estrarre una ad

una le fonti e si ripeterà lo stesso procedimento sugli elementi presenti in ciascuna fonte. Tale operazione servirà per recuperare il nome del documento della fonte e, se presente, l'URL. Con l'Iterable relativo ai tag è possibile estrarre da ogni tag il contenuto, per poi separare ciascun contenuto con una virgola. Ripetendo questa operazione per ciascun tag e assicurandosi anche di non inserire il contenuto del tag relativo all'URL, si ottiene il documento della fonte, al quale verrà eliminato l'ultimo carattere poiché esso corrisponde ad una virgola aggiuntiva. Infine, si cerca, se presente, il tag appartenente alla classe "coleccion", che conterrà il tag <a> che specifica l'URL della fonte, disponendo così dei dati necessari per l'inserimento nel database della fonte. L'operazione si ripeterà per ciascun elemento della bibliografia.

```
    connection.commit()
    print("Inserita la parola:", termine_parola)
```

Terminate le estrazioni dei dati relativi alla parola e alle citazioni, immagini e fonti della parola, verrà effettuato l'inserimento di tutti i dati raccolti tramite cursor.execute(). Poi, sul terminale, verrà mostrata la parola appena inserita (ovviamente il print() non è necessario per il web scraper, ma l'output generato può fornire all'utente un'indicazione per capire a che punto si trova il web scraper nel suo operato). Queste istruzioni verranno eseguite prima della ricerca del prossimo URL da utilizzare per la richiesta al server web.

```
if(connection.is_connected()):
    connection.close()
print("Estrazione terminata")
```

Quando tutte le parole sono state estratte e inserite nel database, la connessione al database viene chiusa e il programma verrà terminato.

3.3 Codice sorgente completo

In questa sezione è possibile consultare il codice sorgente del web scraper. Eventualmente, esso è disponibile anche in una repository online[1], provvista anche delle istruzioni MySQL per la creazione del database.

```
import time
import requests
import mysql.connector
from bs4 import BeautifulSoup

#Dichiarazione delle variabili per l'inserimento

#Parola:
#id_parola, termine_parola, definizione_parola, categoria_parola

#Citazione:
#documento_citazione, parola_citazione, testo_citazione[1]

#Immagine:
#URL_immagine, parola_immagine, descrizione_immagine

#Fonte:
#documento_fonte, parola_fonte, URL_fonte

#Recupero del primo URL
page = requests.get("https://www.um.es/lexico-comercio-medieval/index.php/v/letra/a/")
soup = BeautifulSoup(page.content, "html.parser")
nextURL = soup.find("li", class_="resultado even").find("a")["href"]

#Connessione al database
connection = mysql.connector.connect(
  host="localhost",
  user="dbuser",
```

(1) Davide-Longo/WebScraperDictionary: Web scraper for 'Vocabulario de Comercio Medieval'

```
  password="dbpassword",
  database="dizionario"
)

cursor = connection.cursor()

#Inserimento nel database
while(nextURL != None):

    time.sleep(1)

    page = requests.get(nextURL)
    soup = BeautifulSoup(page.content, "html.parser")

    #Insermento della parola
    id_parola = nextURL.split("/")[len(nextURL.split("/"))-2]
    termine_parola = soup.find("h3", class_="lexema_title").text
    if(soup.find("p", class_="descripcion") != None):
        if(soup.find("nav", class_="breadcrumbs").next_sibling.next_sibling["class"] == ["descripcion"]):
            definizione_parola = soup.find("p", class_="descripcion").text
        else:
            definizione_parola = None
    else:
        definizione_parola = None
    if(soup.find("span", class_="tipo") == None):
        categoria_parola = None
    else:
        categoria_parola = soup.find("span", class_="tipo").text
    cursor.execute("INSERT INTO parola (id, termine, definizione, categoria) VALUES (%s, %s, %s, %s)",
                   (id_parola, termine_parola, definizione_parola, categoria_parola))

    #Inserimento delle citazioni
    complementariaIterable = soup.find_all("h4", class_="complementaria")
    if(complementariaIterable != None):
        descripcionIterator = iter(soup.find_all("p", class_="descripcion"))
        if(definizione_parola != None):
```

```
            next(descripcionIterator)
        for cit in complementariaIterable:
            documento_citazione = cit.find("em").text
            parola_citazione = id_parola
            testo_citazione = next(descripcionIterator).text
            cursor.execute("INSERT INTO citazione (documento, parola, testo) VALUES (%s, %s, %s)",
                    (documento_citazione, parola_citazione, testo_citazione))

    #Inserimento delle immagini
    imagen_lexemaIterable = soup.find_all("p", class_="imagen_lexema")
    if(imagen_lexemaIterable != None):
        for img in imagen_lexemaIterable:
            URL_immagine = img.find("a")["href"]
            parola_immagine = id_parola
            descrizione_immagine = img.find("a")["alt"]
            cursor.execute("INSERT INTO immagine (URL, parola, descrizione) VALUES (%s, %s, %s)",
                    (URL_immagine, parola_immagine, descrizione_immagine))

    #Inserimento delle fonti
    bibliografia = soup.find("ul", class_="bibliografia")
    if(bibliografia != None):
        bibliografiaIterable = bibliografia.find_all("li")
        for src in bibliografiaIterable:
            documento_fonte = ""
            for elem in src.find_all("span"):
                if(elem.find("a") != None):
                    continue
                documento_fonte += elem.text
                documento_fonte += ","
            documento_fonte = documento_fonte[:-1]
            parola_fonte = id_parola
            if(src.find("span", class_="coleccion") == None):
                URL_fonte = None
            else:
                URL_fonte = src.find("span", class_="coleccion").find("a")["href"]
            cursor.execute("INSERT INTO fonte (documento, parola, URL) VALUES (%s, %s, %s)",
```

```
                    (documento_fonte, parola_fonte, URL_fonte))
    connection.commit()
    print("Inserita la parola:", termine_parola)
    if(soup.find("span", class_="siguiente") == None):
        nextURL = None
    else:
        nextURL = soup.find("span", class_="siguiente").find("a")["href"]
if(connection.is_connected()):
    connection.close()
print("Estrazione terminata")
```

4 Risultati e conclusioni

Dopo aver eseguito il codice riportato nella sezione 3.3, il web scraper ha inserito nel database 19091 parole, 2060 citazioni, 51153 immagini e 43093 fonti, per un totale di 115397 istanze. È altamente probabile che rieseguendo il web scraper i risultati non combacerebbero, perché il dizionario online è attualmente in costante revisione e molte parole potrebbero essere aggiunte, modificate od eliminate. Il tempo impiegato per ottenere questo risultato è di circa 7 ore; tale valore può facilmente variare a seconda del dispositivo e per questo non è da considerarsi universale. Disponendo del numero di parole complessive e del tempo che intercorre, come minimo, tra una richiesta di una pagina e l'altra, si è ricavato che sono necessarie almeno 5,3 ore per poter estrarre tutte le informazioni desiderate ed inserirle nel database.

Il web scraper si è fermato ad un certo punto poiché l'URL che doveva condurre alla pagina della parola immediatamente successiva ha invece portato ad un sito web completamente diverso, bloccando così completamente il web scraper. Con il codice attuale, non sarebbe semplice riuscire a gestire un caso particolare come questo, sarebbe necessario optare per un approccio diverso per accedere alle pagine web relative ai vocaboli del dizionario online, che però comporterebbe a prestazioni quasi sicuramente peggiori, soprattutto in termini di tempo.

Da questi risultati è possibile trarre le seguenti conclusioni:

- Come strumento, il web scraper dimostra di essere particolarmente efficiente, purché sia stato realizzato correttamente, nel recuperare grandi quantità di informazioni in un lasso di tempo esiguo. Risulta chiaro che un utente umano non è minimamente in grado di raggiungere le stesse prestazioni di un web scraper e che per questo sia buona cosa affidarsi a tale mezzo per poter risparmiare enormi quantità di tempo.
- Nonostante i risultati soddisfacenti che può ottenere, un web scraper è facilmente soggetto a criticità legate ai contenuti a cui ha accesso e può tendere a commettere errori o a non sapere come comportarsi in certe situazioni. Tali errori derivano, per esempio, dal modo in cui vengono conservati certi dati: un web scraper si aspetta sempre di trovare certi contenuti all'interno di specifici tag. Se invece tali contenuti fossero spostati in altri tag il web scraper non sarebbe in grado, in genere, di recuperare quei dati. Ancora, se al posto di certe informazioni ve ne fossero altre che non risultano essere pertinenti, il web scraper non sarebbe in grado di identificare tale errore (mentre un utente umano sì). È dunque la presenza di pattern che si ripetono che permettono ad un web scraper di lavorare in maniera adatta, ma quando tali pattern scompaiono, il web scraper non può più proseguire correttamente o potrebbe, nel peggiore dei casi, estrarre

dati sbagliati. Tali situazioni richiedendo obbligatoriamente l'intervento umano, che può cercare di risolvere, prevedendoli in anticipo, questi problemi. Purtroppo, anche disponendo delle contromisure adatte, possono sempre sorgere nuove problematiche. Per questo motivo è necessario dover modificare con regolarità i web scraper, sia per ovviare ai problemi appena descritti, sia per riuscire a gestire eventuali modifiche che avvengono alle pagine web (in seguito alla modifica, per esempio, dei documenti HTML utilizzati per le pagine).

Per concludere, il web scraping rappresenta una tecnica che richiede particolare attenzione per quanto riguarda il suo utilizzo, sia perché necessita di costanti modifiche a livello del codice sorgente del programma utilizzato, ma anche perché potrebbero essere estratte informazioni come dati personali e ciò potrebbe essere visto come un comportamento pericoloso e perseguibile legalmente. Nonostante le problematiche, tramite il web scraping sono disponibili al momento numerosi servizi che possono essere d'aiuto per risparmiare grandi quantità di tempo, delegando incarichi ripetitivi ad un web scraper che è in grado di ottenere gli stessi risultati impiegando meno tempo rispetto ad un essere umano. Spesso inconsapevolmente, i web scraper vengono utilizzati ogni giorno; il loro operato ed i loro risultati forniscono vantaggi in termini di tempo e di prestazioni e, come tutti gli strumenti automatizzati, sono ormai necessari ed è impossibile farne a meno.

Sitografia

Data scraping - Wikipedia

What is Data Scraping? - Definition from Techopedia

What is web scraping and internet scraping? No-Coding Ultimate guide

Web Scraping vs. Web Crawling: Understand the Difference

The best file and data formats for Web Scraping services

Beautiful Soup: Build a Web Scraper With Python – Real Python

Web Scraping Basics. How to scrape data from a website | by Songhao Wu | Towards Data Science

What is GDPR, the EU's new data protection law? - GDPR.eu

Watch-outs for Legal and Ethical Web Scraping in 2022

EBay v. Bidder's Edge - Wikipedia

What is a Robots.txt File and how do you create it? - Seobility Wiki

Documentazione consultata

HTML reference - HTML: HyperText Markup Language | MDN

Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation

The Python Standard Library — Python 3.11.0 documentation

PEP 249 – Python Database API Specification v2.0 | peps.python.org

Immagini utilizzate

Figura 1,2,3,4 Aloe - Vocabulario del comercio medieval

Repository del codice sorgente

Davide-Longo/WebScraperDictionary: Web scraper for 'Vocabulario de Comercio Medieval'