UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Migrazione e standardizzazione di terminologia multilingue da file Excel a database relazionali: analisi dei dati di Terminology Without Borders"

Relatore: Prof. Giorgio di Nunzio Maria

Laureando: Yihui Zhu

ANNO ACCADEMICO 2022-2023

Data di laurea 25-09-2023

Sommario

Per un progetto come Terminology Without Borders, che si propone di immagazzinare e fornire crescenti quantità di dati strutturati come glossari multilingue, è vantaggioso disporre di un sistema di gestione dati efficiente. Attualmente le organizzazioni aderenti al progetto forniscono nuove risorse sottoforma di fogli di calcolo. Un database relazionale, in questo contesto, potrebbe portare numerosi vantaggi quali eliminare i duplicati, fornire una fonte centralizzata di dati, maggiore scalabilità, garantire consistenza e integrità dei dati e permettere interrogazioni complesse con il linguaggio SQL. Il presente elaborato si propone di illustrare un modo per migrare i dati da fogli di calcoli con schemi eterogenei esistenti verso un database relazionale in maniera automatizzata, applicato ad un sottoprogetto di Yourterm. A causa della struttura disomogenea tra vari fogli di uno stesso file e tra file di progetti diversi, è necessaria una analisi dei campi esistenti per creare uno schema onnicomprensivo, standardizzato, coerente e non ridondante. Infine, l'approccio di questo lavoro potrebbe essere riutilizzato per essere applicato ad altri progetti Yourterm, che presentano caratteristiche e problematiche simili.

Indice

1 Introduzione

Il progetto Terminology Without Borders del Parlamento europeo collabora con università e partner individuali esterni per raccogliere risorse terminologiche in modo da creare glossari e database multilingue che possano essere usati per comunicare con i cittadini in un linguaggio chiaro. I risultati ottenuti vengono inoltre usati per arricchire le risorse del sistema IATE con nuovi termini o aggiungere informazioni dettagliate a termini esistenti, successivamente alla validazione da parte di esperti di terminologia delle Istituzione Europee. I domini che esso copre sono: medicina, ambiente e conservazione, cultura, diritti delle donne, giustizia e legalità, pesca e marittima, educazione, industria del cibo e gastronomia, IT e social media, economia e finanza, rispettivamente rappresentati da 10 progetti. Ciascuno di essi è composto da sotto-progetti. Le risorse sono ospitate sul sito di Yourterm yourterm.eu. Ogni sottoprogetto consiste in un insieme di pagine di fogli di calcolo compilati da vari collaboratori che sono stati accorpati in un unico documento Excel o Google Sheets. L'obiettivo dell'elaborato è mostrare un modo per migrare i dati contenuti in questi documenti verso un database gestito da un DBMS relazionale.

1.1 Vantaggi dei DBMS rispetto ai fogli di calcolo per gestione database

I fogli di calcolo sono utilizzati in statistica, contabilità, finanza ed in innumerevoli altri ambiti per l'analisi di dati o semplicemente per la conservazione di registri in formato tabulare. I programmi disponibili, per esempio Microsoft Excel e Google Sheets, presentano interfacce utenti grafiche intuitive che offrono ad utenti anche non-programmatori funzionalità avanzate come

la formattazione di celle, creazione di grafici, realizzazione di report, calcolo con formule.

I software per fogli elettronici hanno inoltre la capacità di gestire basi di dati relazionali [1]. In Google Sheets, per esempio, è possibile effettuare interrogazioni stile SQL con la funzione *QUERY* [2], simulare *join* di tabelle con la funzione *VLOOKUP* [3], si possono assegnare permessi agli utenti, forzare l'integrità dei dati con lo strumento *Data validation* e altre formule per verificare l'unicità degli attributi *UNIQUE* e *PRIMARY KEY* e garantire l'integrità referenziale per le chiavi esterne. Nonostante ciò non tutte le funzionalità e caratteristiche di un DBMS relazionale sono presenti e spesso molte implementazioni sono forzature inutilmente complesse rispetto ad una soluzione ad-hoc, d'altronde Microsoft stessa offre prodotti dedicati come Access o SQL Server.

Sebbene soluzioni come *Google Apps Script* permettano di creare API con cui realizzare applicazioni che si interfacciano a Google Sheets come data storage, quest'ultimo non è scalabile in quanto ammette fino ad un massimo di 10 milioni di celle [4], inoltre le prestazioni sono inferiori rispetto ad un DBMS come MySQL [5]. Per di più, affidarsi a soluzioni proprietarie così particolari rende difficoltosa la portabilità del database.

1.2 Struttura esistente disomogenea

I fogli elettronici dei vari sotto-progetti presentano una struttura di base comune, contengono sempre due tipi di pagine. Per esempio Climate Change.xlsx è composta da:

- una pagina chiamata *All languages* contenente, per ogni riga della tabella, i termini equivalenti di più lingue, come in tabella 1

EN	RO	ES	...
Oxidation	oxidare	Oxidación	...

Tabella 1: Esempio di record della tabella All languages in Climate Change.xlsx

- una pagina per ogni lingua, in cui ciascuna riga rappresenta un termine con le relative informazioni. In tabella 2 un esempio di record della pagina "IT"

colonna	valore
Source term	benzene
Target term*	benzolo
Term Reference*	https://www.treccani.it/...
Language ID	IT
Notes (Term)	
IATE ID	1566036
Definition*	liquido mobile d'odore etereo...
...	...

Tabella 2: Esempio di record della tabella IT in Climate Change.xlsx

Le regole per la compilazione delle tabelle e i significati delle colonne sono definite nelle linee guida IATE [6], inoltre esiste un documento Excel di riferimento con le istruzioni e contenente una struttura iniziale, di seguito riportati. Nota: i campi con l'asterisco(*) sono obbligatori.

- Source term*: insert the source term
- Language ID*: insert the two-letter code of the ISO 639-1 of the language of the target term, e.g. FR

- Target term*: insert the target term
- Part of speech*: noun, verb, adjective... for the correct value to enter (for each language), refer to the file named "pos_per_language".
- Note (Term): any relevant comments on the target term
- IATE: if available, the IATE ID corresponding to the term
- Term reference *: source in which the target term has been found
- Definition *: short and precise definition of the term, written in the language of the target term
- External cross reference (definition): web link to the source in which the definition has been found
- Source (definition)*: bibliographical citation (title/author/etc.) of the definition source
- Note (definition): any relevant information about the definition
- Synonym: two terms are synonyms if they share the same meaning and the same usage - they have to always be replaceable
- Quasi synonym: if two terms share the same meaning but not the same usage and therefore are not always replaceable, they are quasi-synonyms
- Common name: if the term has a common name, that is a popular variant (e.g. fever for pyrexia).
- Scientific name: if the term has a scientific correspondent (e.g. pyrexia for fever).
- Orthographic variant: any orthographic variant
- Acronym: any acronym

- Abbreviation: any abbreviation
- Domain*: domain of the term (e.g. "medicine") Note: on the FAIR-term WebApp this field is named with the ISO standard label as "subject field"
- Subdomain: specific sub-domain (e.g. oncology)
- Context*: sentence or short paragraph containing the term (it must be different from the definition, but it can be the same as the term reference)
- External cross reference (context): web link to the source in which the context has been found
- Source (context)*: bibliographical citation (title/author/etc.) of the context source
- Project*: name of the Your Term project (e.g. YourTerm MED)
- Subproject*: name of the Your Term subproject (e.g. "bipolar disorders")

Purtroppo tra i vari fogli di sotto-progetti diversi e tra le stesse pagine di uno stesso documento esistono delle inconsistenze. Queste devono essere risolte ridefinendo uno schema ex novo. Inoltre molte colonne sono state compilate in maniera tale da richiedere la loro rielaborazione prima di importarli in un database. I problemi sono:

- schemi delle tabelle diversi: alcune pagine sono prive di certe colonne, altre hanno colonne aggiuntive non presenti nelle istruzioni. Spesso le colonne hanno ordini differenti e addirittura nomi diversi
- assenza di valori in campi obbligatori. A volte l'assenza di valore viene specificata con il simbolo "-"

- i dati sono stati inseriti sotto colonne sbagliate, a volte sistematicamente, a volte sono errori occasionali
- non esatta corrispondenza dei termini in “All languages” con gli equivalenti nelle altre pagine a causa di errori di compilazione, per esempio la capitalizzazione è diversa
- i campi come “Synonym”, “Quasi synonym”, “Abbreviation” e altri possono presentare più valori in una cella, ma sono separati con deliminatori diversi: a volte con il forward slash, a volte con la virgola, altre volte sono enumerati con cifre seguite dal punto
- Talvolta i campi “Target term” e le colonne delle pagine “All languages” presentano più valori che sono separati dai deliminatori in maniera inconsistente

In generale le celle mancano di formule per la verifica dei formati di dato inseriti. Creare un database con schemi uguali a quelli presenti nei fogli di calcolo è quindi impensabile. D’altronde anche considerando le intestazioni delle tabelle Excel come relazioni, queste presentano numerose dipendenze funzionali, perciò è necessario definire una nuova struttura che sia standard e più normalizzata.

1.3 Obiettivi

Il risultato finale della tesi è una base di dati relazionale gestita dal DBMS PostgreSQL adatta per un’applicazione multi utente che deve gestire operazioni di tipo inserimento, modifica ed eliminazione di termini, gestione corrispondenza di stessi termini in più lingue e occasionalmente stampa ed esportazione dei record, per esempio per importarli nel database IATE.

Il lavoro consiste in una fase di progettazione del database, in cui verranno illustrati gli schemi concettuale e relazionali ottenuti e le motivazioni dietro alle scelte progettuali, seguita da una fase di migrazione vera e propria dove si mostreranno le tecniche e gli script usati per estrarre dati dai fogli di calcolo ed inserirli nel database.

Il database e gli script sono stati usando i file appartenenti ai sottoprogetti "Climate change" e "Zoology" del progetto "ENVI" disponibili al link.

2 Design database

Il primo passo della migrazione è stato la progettazione della base di dati. In questa ricerca è stato adottato un metodo di progettazione graduale e metodico, con le seguenti fasi:

1. Nella fase di progettazione concettuale è stato realizzato uno diagramma ER analizzando le strutture tabellari esistenti, per identificare tutti le informazioni necessarie da modellare

2. Lo schema concettuale ottenuto è stato tradotto in uno schema logico, su cui sono state apportate scelte di progetto atte a rendere il database più efficiente in termini di spazio occupato e di velocità di esecuzione di potenziali operazioni frequenti. In assenza di particolari requisiti, è stato deciso di definire degli schemi relazionali altamente normalizzati per garantire qualità al design

3. Nella progettazione fisica è stato prodotto il codice SQL per implementare gli schemi precedenti nel DBMS scelto, PostgreSQL

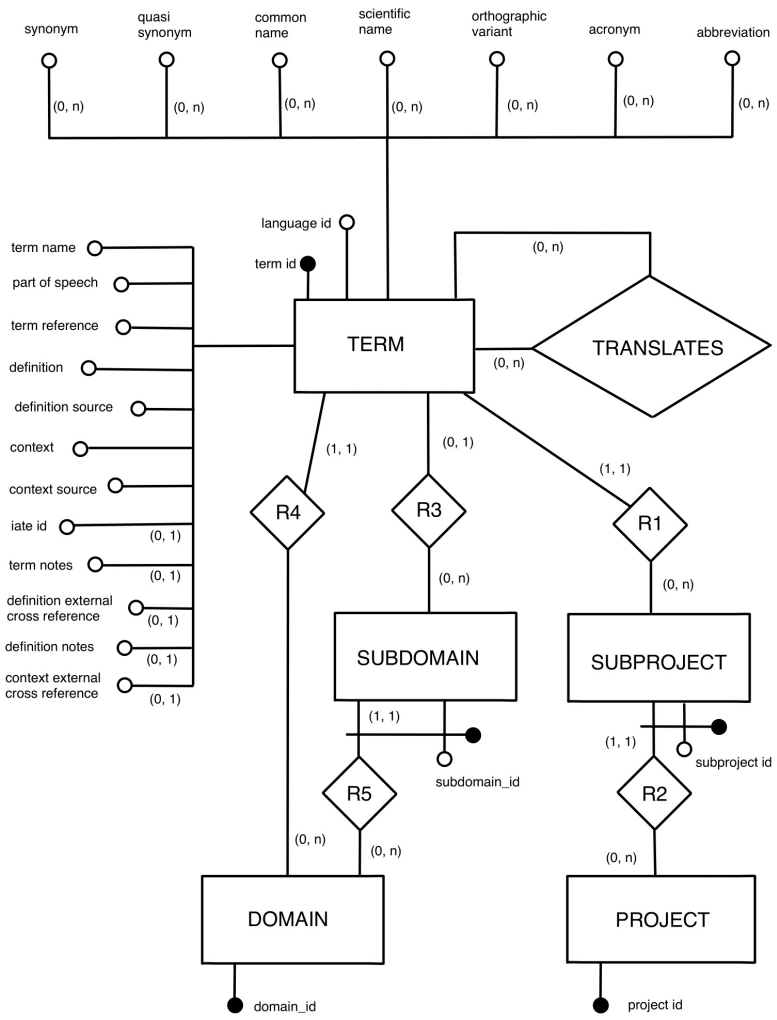

Figura 1: Diagramma ER del database

2.1 Progettazione concettuale

Prendendo come riferimento l'intestazione della tabella di esempio presente in Yourterm toolbox e descritta nella sezione 1.2, si sono rilevate le informazioni da modellare ed è stato ricavato il diagramma ER in fig 1. Eventuali campi aggiuntivi non standard introdotti in certe tabelle sono ignorati(come *Transaction*, *Responsibility*, *Name/Surname*, *Date* e *Animal*), perché sono dati non rilevanti per il progetto aggiunti su iniziativa delle singole univer-

sità e che non sono stati rimossi durante la fase di fusione dei file prima del caricamento sul sito yourterm. Per rendere la nomenclatura più chiara certe espressioni originali sono state rinominate riordinando le parole, sono stati rimossi caratteri speciali come asterischi e parentesi e i tutti caratteri sono stati convertiti in minuscolo: per esempio "External cross reference(definition)" è diventato "definition external cross reference".

L'entità centrale è *TERM*, che rappresenta un termine, ovvero una riga della tabella del foglio di calcolo. Sebbene un termine possa essere identificato univocamente dalla combinazione degli attributi *term name*(*Target term* in origine) e *language id*, è stato deciso di introdurre un identificativo artificiale, *term id*, sottoforma di sequenza di numeri interi. L'approccio con chiave naturale non prevede che un termine possa avere più significati, almeno senza accumularli tutti in un'unica tupla, come in un dizionario, soluzione poco flessibile e che non rispetta la prima forma normale. La soluzione con identificativo artificiale, invece, permette l'esistenza di termini con lo stesso nome che rappresentano più concetti in contesti diversi, ed è anche l'approccio adottato da IATE [7], oltre ad essere potenzialmente più performante nelle operazioni di join in quanto viene usato solo un campo a numeri interi come chiave. I campi contrassegnati dall'asterisco nel file Excel di esempio denotano campi a compilazione obbligatoria, e sono stati resi attributi a cardinalità obbligatoria, e i rimanenti attributi sono stati resi facoltativi.

I campi *Subproject* e *Project* sono stati trasformati in entità separate, *SUBPROJECT* e *PROJECT*, per esprimere la dipendenza del primo dal secondo, cioè un *SUBPROJECT* appartiene ad uno e un solo *PROJECT*; attraverso la partecipazione di cardinalità uno a uno di *TERM* con la associazione *R1* e di *SUBPROJECT* con *R2* si può ottenere a quale *PROJECT*

appartiene un'istanza di TERM. Inoltre sia *PROJECT* che *SUBPROJECT* sono entità con un limitato numero di valori predefiniti ammissibili.

Anche campi *Domain* e *Subdomain* sono state rese entità separate, *DOMAIN* e *SUBDOMAIN*. Nelle regole di compilazione della tabella di riferimento, però, *Domain* è a compilazione obbligatoria mentre *Subdomain* non lo è, quindi è necessaria una ulteriore associazione tra *TERM* e *DOMAIN*.

La associazione *TRANSLATION* esprime le corrispondenze tra termini equivalenti in lingue diverse, specificate nelle tabelle *All languages*.

2.2 Progettazione logica

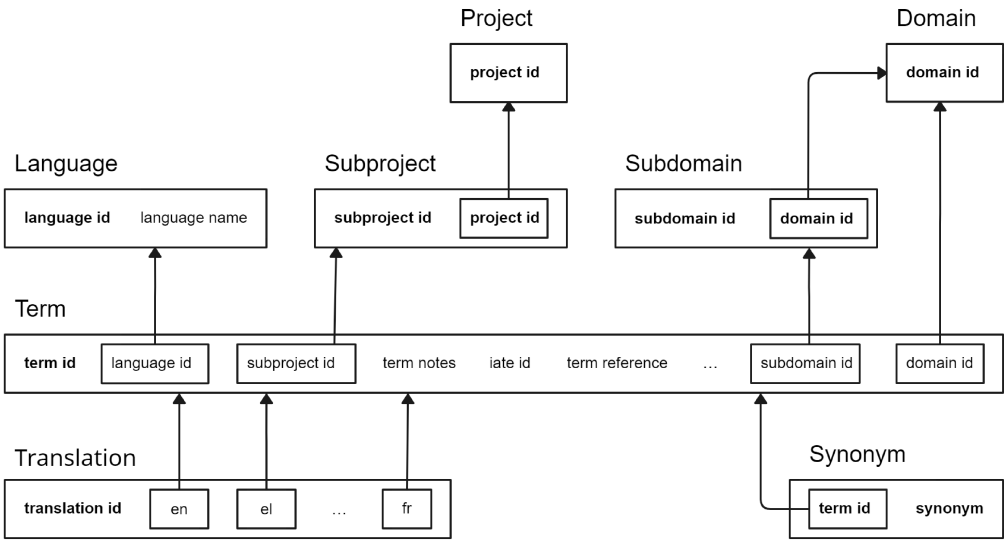

Figura 2: Schema relazionale del database. Le relazioni Acronym, Quasi synonym, Common name, Scientific name, Orthographic variant e Abbreviation sono simili a Synonym, non sono stati inseriti per motivi di spazio.

Traducendo il diagramma ottenuto nella fase precedente è stato ricavato lo schema relazionale in fig 2. Gli attributi multi-valore(*synonym*, *quasi sy-*

nonym, *common name*, *scientific name*, *orthographic variant*, *acronym* e *abbreviation*) sono stati tradotti in relazioni con vincoli di integrità referenziale con la relazione Term.

L'associazione ricorsiva *TRANSLATE* è stata trasformata nella relazione $Translation(\underline{translation\ id},\ en,\ el,\ ...,\ fr)$ che ha come componenti, oltre ad una chiave surrogata *translation id*, una chiave esterna per ogni lingua con vincolo di integrità referenziale con Term.

Una variante più naturale e intuitiva sarebbe potuta essere per esempio $Translation(\underline{source\ term\ id,\ target\ term\ id})$, dove *source term id* e *target term id* sono coppie di termini equivalenti in lingue diverse. Purtroppo tale soluzione risulterebbe piuttosto dispendiosa in termini di storage in quanto presenta ridondanze: il numero di tuple totali per esprimere una traduzione di un termine richiede è pari al numero di disposizioni di n lingue a 2 a 2. Quindi per ogni traduzione, assumendo che il termine sia tradotto in tutte n le lingue disponibile, servirebbero $t = n \cdot (n-1)$ tuple. Nei progetti sono state finora individuate 11 lingue, perciò potenzialmente sarebbero 110 tuple per traduzione.

La soluzione adottata, invece, è più efficiente in termini di spazio occupato in quanto richiede solo una tupla con $n+1$ attributi per ogni traduzione, tranne i casi in cui un campo è composto da più valori separati da delimitatore. In tali situazioni si crea un numero di tuple pari al numero di combinazioni del prodotto cartesiano tra insiemi di termini. La chiave surrogata è necessaria in quanto i restanti campi sono frequentemente nulli a causa dell'assenza di corrispondenze in certe lingue.

Un compromesso sarebbe riservare il campo *source term id* per i soli termini in inglese, dato che nei documenti Excel ogni termine ha sempre il suo corrispondente in inglese. Si avrebbe potenzialmente un grande risparmio

di memoria evitando il grande numero di combinazioni tra ogni coppia di lingue o i numerosi valori nulli. Purtroppo questo approccio è poco flessibile in quanto implica l'obbligo della presenza di un termine in inglese prima di poter stabilire traduzioni.

L'attributo *language id* di *TERM* è stato reso una relazione a se stante. La motivazione è dovuta alla volontà di rendere possibile l'aggiunta di ulteriori attributi non esplicitamente richiesti o assolutamente necessari ma che potrebbero risultare utili, per esempio la denominazione per esteso della lingua *language name* potrebbe essere comoda in fase di realizzazione di un menù di selezione drop-down per la scelta del linguaggio in una ipotetica applicazione.

Lo schema ottenuto è quasi in terza forma normale: presenta solo una dipendenza funzionale, *subdomain id* $\rightarrow$ *domain id*, non normalizzabile a causa del fatto che *subdomain_id* potrebbe essere ignoto per le caratteristiche del problema.

2.3 Progettazione fisica

È stato adottato il naming convention *snake_case*. le chiavi primarie hanno lo stesso nome delle tabelle con suffisso *_id* e campi che fungono da chiavi esterne sono generalmente denominati con i nomi degli attributi a cui si riferiscono.

A causa dell'imprevedibilità della lunghezza in caratteri della maggioranza dei campi, è stato scelto il tipo di dato *TEXT* per tutti gli attributi testuali, ad eccezione di quelli di lunghezza consistente come *project_id*, o quelli su cui è di interessa creare indici, come *term_name*, per cui sono stati scelti tipi *CHAR* o *VARCHAR*.

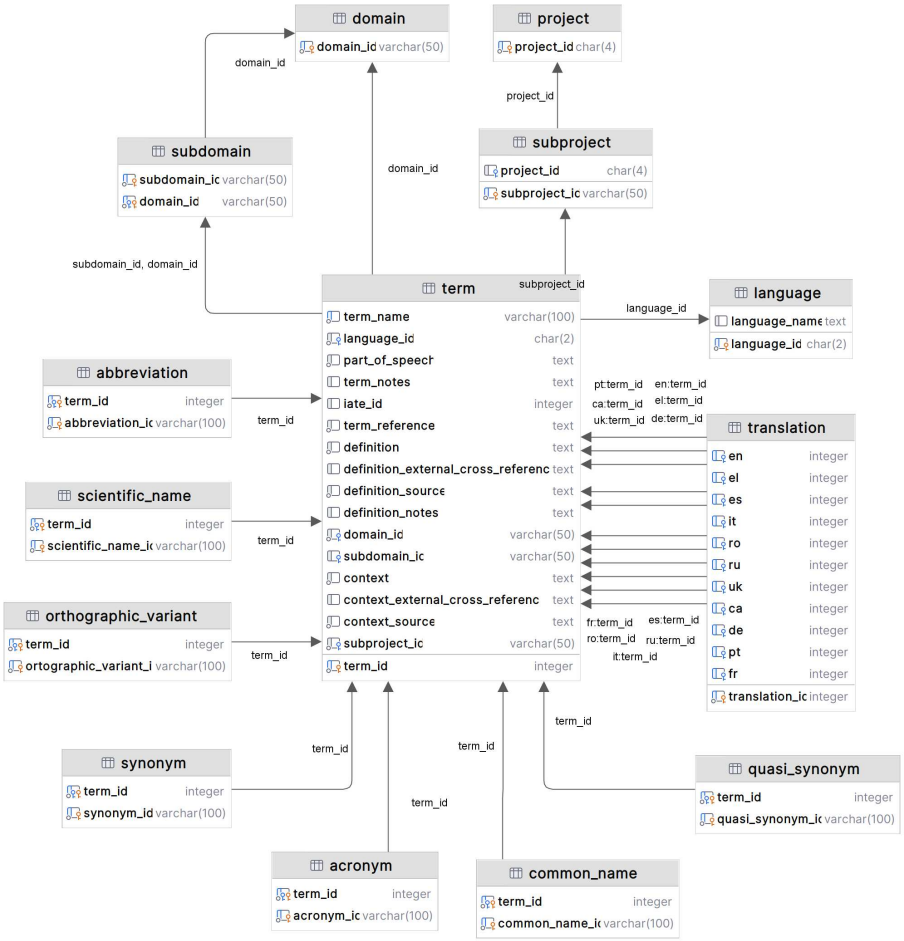

Figura 3: Schema fisico del database.

È stato prodotto il codice SQL per implementare gli schemi del database progettato (9).

Per migliorare le prestazioni delle operazioni di join tra *term* e *translation* sono stati creati indici sulle tutte le chiavi esterne della tabella *translation*. Prevedendo frequenti ricerche sui nomi dei termini è stato inoltre creato un indice sulla colonna *term_name* della tabella *term*.

Prima di effettuare la migrazione è necessario inserire manualmente le tuple relative ai linguaggi, progetti, sotto-progetti, domini e sotto-domini.

La procedura per estrarre rapidamente tali valori univoci usando Excel è:

1. selezionare il menù *Dati*
2. dalla sezione *Ordinaefiltra* cliccare *Avanzate*
3. selezionare la colonna di interesse
4. spuntare il check-box *Copia univoca dei record*

Per imporre l'inserimento corretto di codici lingua definiti dallo standard ISO 639-1 è stato applicato il vincolo di integrità referenziale tra il campo *language_id* di *term* e la tabella *language*, in questo modo solo i valori presenti in quest'ultima sono ammessi.

Alternativamente si sarebbe potuto usare un vincolo di controllo sulla colonna *language_id*:

```
CREATE TABLE term(
    language_id CHAR(2) NOT NULL,
    ...
    CONSTRAINT ADMITTED_LANGUAGES CHECK(language_id IN
  ('en', 'el', 'es', 'it', 'ro', 'ru', 'uk', 'ca', 'de', 'pt', 'fr'))
);
```

Un'ulteriore soluzione sarebbe stata la definizione di un *DOMAIN*, ma il vincolo viene usato solo nella tabella *term* e perciò non è da preferire ad un vincolo di controllo su colonna.

```
CREATE DOMAIN language AS CHAR(2)
CHECK(VALUE IN('en', 'el', 'es', 'it', 'ro', 'ru', 'uk', 'ca', 'de', 'pt', '
    fr'));
```

Con la soluzione adottata risulta più immediato aggiungere nuove lingue e la modifica alle lingue esistenti si propaga in modo automatico sui record della tabella "term", non è necessario infatti eliminare il vecchio *CONSTRAINT/DOMAIN*, crearne un nuovo ed aggiornare manualmente i valori.

3 Migrazione da fogli di calcolo a database

Dopo che il database è stato creato, sono stati trasformati i fogli di calcolo in file interpretabili da uno script di parsing, in questo caso è stato scelto il formato CSV, anche perché facilmente esportabile da Excel o Google Sheets. La codifica dei caratteri è UTF-8, considerato che si stanno importando testi scritti in lingue europee. I due programmi dedicati al parsing e all'inserimento in database sono stati scritti in modo tale da elaborare i documenti CSV ottenuti e risolvere i problemi elencati in sezione 1.2.

3.1 Esportazione documenti Excel in formato CSV

Sono state scaricate le copie dei file *Climate Change.xlsx* e *Zoology.xlsx* da ENVI projects. Usando il software Microsoft Excel per manipolare i documenti scaricati. Sono state eseguite le seguenti modifiche:

1. Si sono eliminate le colonne vuote delle tabelle insieme a quelle non appartenenti allo schema logico

2. Sono state eliminate le righe finali vuote inutili

3. Se sono stati presenti errori di compilazione sistematici, come per esempio nella pagina *DE* di *Zoology*, si è intervenuti manualmente, in questo caso spostando i valori della colonna *Orthographic variant* in *Part of speech*, dove si dovrebbero trovare correttamente

4. Le singole pagine di un file sono state esportate in formato CSV UTF-8

5. le colonne delle tabelle sono state rinominate con i nomi definiti nello schema fisico. In particolare i campi *Synonym*, *Quasi synonym*,

Common name, *Scientific name*, *Orthographic variant*, *Acronym* e *Abbreviation* sono stati rinominati con i nomi delle loro rispettive tabelle, mentre i restanti attributi seguono i nomi degli attributi della tabella *term*. Le pagine *All languages* seguono la denominazione della tabella *translation*.

I file risultanti sono puliti da campi o righe inutili ed condividono intestazioni con nomi standard. Il campo *Source term* non viene mai utilizzato perché l'informazione è già presente nella pagina *All languages*, quindi viene ignorato.

3.2 Parsing

Il parsing dei file ottenuti viene effettuato da due script. Uno si occupa di analizzare le pagine "*All languages*" contenente la traduzioni, l'altro si occupa di tutte le restanti pagine contenenti i termini. Gli script completi sono stati scritti in Python e sono disponibili in appendice 6. Effettuano sia il parsing dei file CSV usando la libreria Pandas che l'inserimento nel database usando l'adapter Python Psycopg2 per PostgreSQL.

```
1  CLASS Term(term_name, language_id, part_of_speech, ...)
2  -- synonym, quasi_synonym, abbreviation, ...
3  CLASS Variant(term_name, variant)
4
5  PARSE_TERMS(filename, language, subproject, domain)
6  dataframe <- read_csv(filename)
7
8  -- output
9  terms <- []
10 variations <- {'synonym': [], 'quasi_synonym': [], ...}
11
12 FOR entry IN dataframe DO
13     -- creazione di un termine per ogni riga di dataframe e assegnargli i
       suoi valori
```

```
    term <- create_empty_term()
    FOR field IN fields(term) DO
        IF field IN {'language_id', 'subproject_id', 'domain_id'} SKIP
        value <- entry[field]
        value <- trim(value)
        IF exists(value) THEN
            IF field == 'iate_id' THEN term.iate_id <- to_integer(value)
            ELSE IF field == 'subdomain_id' THEN term.subdomain_id <-
    lowercase(value)
            ELSE term.field <- value

    -- se record vuoto non inserirlo in database
    IF is_empty(term) THEN CONTINUE

    -- valori costanti assegnati da argomenti e non da file
    term.language_id <- language
    term.subproject_id <- subproject
    term.domain_id <- domain

    -- term name potrebbe avere valori separati
    new_terms <- split_and_trim(term.term_name, ';', ',', '/')
    new_terms <- lowercase(new_terms)
    terms <- [terms, new_term]

    -- associazione di term alle sue varianti
    FOR key IN keys(variant) DO
        value <- entry[key]
        IF exists(value) THEN
            FOR term IN new_terms DO
                FOR variant IN split_and_trim(value, ',', '/', '[0-9].') DO
                    variants[key] <- [variants[key], Variant(term, variant)]

insert_into_db(terms)
insert_into_db(variations)
```

Listing 1: Pseudocodice script di parsing file di termini

```
CLASS Translation(en, el, ... fr)

PARSE_TRANSLATIONS(filename)
dataframe <- read_csv(filename)

```

```
-- output
translations = []

FOR entry IN dataframe DO
    translation <- create_empty_translation()
  FOR field IN fields(translation) DO
    value <- entry[field]
    IF exists(value) THEN
        value <- trim(value)
        value <- lowercase(value)
        translation.field <- value

  -- se i campi contengono valori multipli
  -- sets contiene una lista di liste
  sets <- []
  FOR field IN fields(translation) DO
    set <- split_and_trim(translation.field, ';', ',', '/')
    sets <- [sets, set]
  combinations <- cartesian_product(sets)

  -- creare una traduzione per ogni combinazione di valori
  FOR combination IN combinations DO
    translation <- create_translation(combination)
    translations <- [translations, translation]

insert_into_db(translations)
```

Listing 2: Pseudocodice script di parsing file di traduzioni

L'esecuzione dello script per i termini richiede come parametri da riga di comando, oltre al file da analizzare, il codice del linguaggio, il nome del sottoprogetto e il nome del dominio a cui appartengono i termini del file, che vengono assegnati alle nuove tuple ignorando valori presenti all'interno. Questo per garantire l'inserimento esatto di *language_id* nel formato ISO 639-1 e degli stessi *subproject_id* e *domain_id* presenti già nelle tabelle *subproject* e *domain*; inoltre sono attributi obbligatori costanti in uno stesso file che potrebbero non essere stati compilati, oppure compilati non uniformemente

causando violazioni di vincoli. La colonna *subdomain_id* spesso contiene gli stessi valori ma con capitalizzazione inconsistente, perciò sono stati resi tutti in minuscolo per renderli compatibili con i valori univoci inseriti nella tabella *subdomain*.

```
python termsParser.py csv/ZoologyEN.csv en "Zoology" Environment
```

Listing 3: Esempio di comando per eseguire lo script

Frequentemente i campi obbligatori non sono stati compilati, spesso perché i file CSV mancano interamente di alcune colonne; in questi casi agli attributi *NOT NULL* viene assegnata una stringa vuota, mentre agli attributi senza tale vincolo viene assegnato il valore *NULL* per indicare l'assenza di valori. Per rispettare le linee guida servirebbe l'intervento manuale per la compilazione dei campi mancanti.

A causa dell'inconsistente uso delle lettere maiuscole e delle minuscole nel campo *Target term*, è stato deciso di convertirne tutte le occorrenze in minuscolo. In questo modo si sono evitati i casi in cui un non si riusciva a stabilire la traduzione di un termine perché il valore nella tabella *All languages* di una riga è diverso da quello in *Target term* in termini di capitalizzazione. Talvolta però, questo è dovuto a errori di battitura che sono risolvibili attraverso l'intervento manuale e sono stati ignorati dallo script.

Quando il campo *Target term* presenta più valori separati da un delimitatore come ";", "," o "/", per esempio in numerose righe della pagina *ES* di *Zoology.xlsx*, vengono create più tuple tutte con gli stessi valori.

4 Risultati

4.1 Correttezza migrazione

Completata la migrazione dei due sotto-progetti analizzati, la tabella *term* contiene 1681 righe di cui 72, contati dallo script, sono valori aggiuntivi che nei documenti originali condividono una stessa cella con altri da cui sono separati da deliminatori. Utilizzando la formula *CONTA.VALORI* di Excel è stato contato un numero di righe totale originale pari a 1609, perciò tutte le occorrenze sono state trasferite.

La tabella *translation* contiene 643 righe di cui 123, contati dallo script, sono gli elementi aggiuntivi dovuti alle combinazioni di traduzioni per i casi in cui i campi presentavano più valori separati da deliminatori. Altre 66 righe sono i duplicati dovuti alla presenza di termini con *term_name* e *language_id* uguali, il cui numero è stato trovato con la query (8). Nei documenti originali le righe sono 454, perciò tutte le traduzioni sono state importate.

4.2 Esempi di interrogazioni

È possibile riottenere la rappresentazione completa di tutti termini con tutti gli attributi originari eseguendo la seguente query

```
SELECT term_name AS "term name",
       language_name AS "language name",
       part_of_speech AS "part of speech",
       term_notes AS "note(term)",
       iate_id AS IATE,
       term_reference AS "term reference",
       definition,
       definition_external_cross_reference AS "external cross reference(
    definition)",
       definition_source AS "source(definition)",
       definition_notes AS "note(definition)",
       context,
```

```
        context_external_cross_reference AS "external cross reference(context
    )",
        context_source AS "source(context)",
        synonym_id AS synoynm,
        quasi_synonym_id AS "quasi synonym",
        common_name_id AS "common name",
        scientific_name_id AS "scientific name",
        ortographic_variant_id,
        acronym_id,
        abbreviation_id,
        domain.domain_id,
        subdomain.subdomain_id,
        subproject.subproject_id,
        project.project_id
FROM term
    LEFT JOIN language ON term.language_id = language.language_id
    LEFT JOIN synonym ON term.term_id = synonym.term_id
    LEFT JOIN quasi_synonym ON term.term_id = quasi_synonym.term_id
    LEFT JOIN common_name ON term.term_id = common_name.term_id
    LEFT JOIN scientific_name ON term.term_id = scientific_name.term_id
    LEFT JOIN orthographic_variant ON term.term_id = orthographic_variant.
    term_id
    LEFT JOIN acronym ON term.term_id = acronym.term_id
    LEFT JOIN abbreviation ON term.term_id = abbreviation.term_id
    LEFT JOIN domain ON term.domain_id = domain.domain_id
    LEFT JOIN subdomain ON term.subdomain_id = subdomain.domain_id
    LEFT JOIN subproject ON term.subproject_id = subproject.subproject_id
    LEFT JOIN project ON subproject.project_id = project.project_id;
```

Le traduzioni si ottengono con

```
    SELECT en.term_name, el.term_name, es.term_name,
        it.term_name, ro.term_name, ru.term_name,
        uk.term_name, ca.term_name, de.term_name,
        pt.term_name, fr.term_name
FROM translation
    LEFT JOIN term as en on translation.en = en.term_id
    LEFT JOIN term as el on translation.el = el.term_id
    LEFT JOIN term as es on translation.es = es.term_id
    LEFT JOIN term as it on translation.it = it.term_id
    LEFT JOIN term as ro on translation.ro = ro.term_id
    LEFT JOIN term as ru on translation.ru = ru.term_id
```

```
    LEFT JOIN term as uk on translation.uk = uk.term_id
    LEFT JOIN term as ca on translation.ca = ca.term_id
    LEFT JOIN term as de on translation.de = de.term_id
    LEFT JOIN term as pt on translation.pt = pt.term_id
    LEFT JOIN term as fr on translation.fr = fr.term_id
```

Siccome l'associazione *TRANSLATE* è stata trasformata nella relazione $Translation(\underline{translation\ id},\ en,\ el,\ ...,\ fr)(1)$ e non nella più intuitiva $Translation(\underline{source\ term\ id,\ target\ term\ id})(2)$, le operazioni che coinvolgono la tabella *translation* richiedono più attenzione.

Per esempio, se l'operazione fosse: " *Trovare tutte le traduzioni del termine 'xxx' dalla lingua 'yy'*", la query potrebbe essere:

```
WITH trans AS(
    SELECT en, el, es, it, ro, ru, uk, ca, de, pt, fr
    FROM term JOIN translation ON term.term_id = translation.en
    WHERE term_name = 'younger dryas')
SELECT *
FROM term
WHERE term_id IN (
    SELECT en FROM trans UNION
    SELECT el FROM trans UNION
    SELECT es FROM trans UNION
    SELECT it FROM trans UNION
    SELECT ro FROM trans UNION
    SELECT ru FROM trans UNION
    SELECT uk FROM trans UNION
    SELECT ca FROM trans UNION
    SELECT de FROM trans UNION
    SELECT pt FROM trans UNION
    SELECT fr FROM trans);
```

Listing 4: Query 1

La stessa query, se si fosse adottato lo schema (2), appare più immediata:

```
SELECT term.*
FROM term JOIN translation ON term.term_id = translation.target_term_id
WHERE source_term_id IN (
    SELECT term_id
```

```
    FROM term
    WHERE term_name = 'aditiv' AND language_id = 'ro'
);
```

Listing 5: Query 2

Per comparazione è stato implementato lo schema (2) nella variante con *source_term_id* che contiene solo termini in inglese. Di conseguenza le query che eseguono la stessa operazione si distinguono per i casi in cui il termine da cercare sia inglese o non.

```
WITH source_terms AS(
    SELECT term_id
    FROM term
    WHERE term_name = 'younger dryas' AND language_id = 'en')

SELECT term.*
FROM term JOIN translation ON term.term_id = translation.target_term_id
WHERE source_term_id IN (SELECT * FROM source_terms)
UNION
SELECT term.*
FROM term
WHERE term_id IN (SELECT * FROM source_terms);
```

Listing 6: Query 3. Seleziona le traduzioni in tutte le lingue di un certo termine inglese

```
WITH source_terms AS(
    SELECT source_term_id
    FROM term JOIN translation ON term_id = target_term_id
    WHERE term_name = 'dryas recente' AND language_id = 'it')

SELECT term.*
FROM term JOIN translation ON term.term_id = translation.target_term_id
WHERE source_term_id IN (SELECT * FROM source_terms)
UNION
SELECT term.*
FROM term
```

```
12 WHERE term_id IN (SELECT * FROM source_terms);
```

Listing 7: Query 4. Seleziona le traduzioni in tutte le lingue di un certo termine solo se non inglese

Preponendo il comando $EXPLAINANALYZE$ alle query si ottengono i query plan. Effettuando la stessa query 20 volte in seguito ad una query iniziale di warmup e mediando i risultati del campo *actual time* si sono ottenute informazioni che forniscono indicativamente indizi sulle prestazioni dei diversi schemi e l'importanza degli indici.

	Query 1	Query 1 senza indici	Query 3
startup cost	17.29	181.30	23.72
total cost	92.48	256.50	23.76
actual startup time[ms]	0.05	0.522	0.081
actual total time[ms]	0.062	0.533	0.085

Tabella 3: Comparativa prestazioni query diverse per stessa operazione

Una grande parte del miglioramento del tempo di esecuzione di Query 1 è dovuto alla presenza dell'indice sulla colonna *term_name*, in quanto il costo maggiore è dovuto alla scansione sequenziale sulla tabella "term" per cercare il nome del termine.

Query 1 (4) necessità di un *index scan* su *term* e di un *nested loop join* in meno rispetto a Query 3 (6), ed è quindi più efficiente. Lo spazio disco occupato, però, è superiore. La tabella con schema (1) occupa 304kB e contiene 643 righe, quella con schema (2) occupa 232kB con 1195 record. Il divario aumenta nei casi in cui la maggior parte delle traduzioni coinvolga un numero ridotto di lingue.

5 Conclusioni

Nel seguente elaborato è stato illustrato il processo che ha portato alla definizione di uno schema database adatto a rappresentare le terminologie con le relative traduzioni del progetto Yourterm e la creazione di procedimenti e script per la migrazione automatizzata dei dati preesistenti da fogli di calcolo a un database relazionale gestito da DBMS.

È stato seguito un procedimento di progettazione metodica del database, partendo dallo definire uno schema concettuale, a sua volta tradotto in uno schema logico, fino ad arrivare alla implementazione fisica in linguaggio SQL, adottando scelte progettuali e implementazioni che rendessero il database altamente normalizzato e performante.

Nonostante i fogli di calcolo su cui si sono svolte le analisi fossero compilati in maniera non sempre rigorosa e senza particolari standard, lo script di parsing è stato in grado di estrarre con successo, attraverso elaborazioni, la maggior parte dei dati per rappresentarli in maniera significativa nel database. Purtroppo, i valori del campo *Part of speech* presentano una eccessiva eterogeneità in termini di capitalizzazione, linguaggio, deliminatori e varietà tale che non si è riusciti a importarli in maniera strutturata. Ci si è limitati dunque a copiare i valori sottoforma di testo in un attributo di tipo *TEXT* nella tabella *term*. Sarebbe necessario intervenire manualmente per aggiornare i record con dei valori standard.

Sebbene la progettazione del database e dei due programmi di estrapolazione si sia focalizzata sull'elaborazione di solo due sotto-progetti, *Zoology* e *Climate Change*, del progetto ENVI di Yourterm, il materiale prodotto durante questa ricerca può essere base di ulteriori modifiche e affinamenti futuri per trasferire i restanti progetti, d'altronde tutti i documenti Excel presenti sul sito condividono una struttura comune. In tal caso sarà possibi-

le utilizzare il database come back-end per una applicazione di gestione per le terminologie più centralizzata ed efficace del sistema attuale, e gli script di parsing potranno comunque essere utili per migrare altre collezioni di termini se si volesse continuare ad usare fogli di calcolo come metodo di inserimento dati.

Riferimenti bibliografici

[1] J. Tyszkiewicz, "Spreadsheet as a relational database engine," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 195–206, 2010.

[2] Google, "Query function - google docs editors help," 2023. Last accessed 19 September 2023.

[3] L. Inc., "Workaround for using sql join on google sheets," 2023. Last accessed 19 September 2023.

[4] Google, "Files you can store in google drive - google drive help," 2023. Last accessed 19 September 2023.

[5] L. J. Ekanayake, D. Ihalage, and S. P. Abyesundara, "Performance evaluation of google spreadsheet over rdbms through cloud scripting algorithms," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1–7, IEEE, 2021.

[6] T. C. Unit, "Toolbox," 2023. Last accessed 19 September 2023.

[7] IATE, "Iate faq," 2023. Last build date 22 August 2023 09:39.

6 Appendice

```
WITH duplicates AS
    (SELECT term_name, language_id, COUNT(*) AS duplicate_count
    FROM term
    GROUP BY term_name, language_id
    HAVING COUNT(*) > 1)

SELECT SUM(num) AS "total duplicates", COUNT(*) AS "duplicated rows" FROM
    (SELECT en.term_name, el.term_name, es.term_name,
           it.term_name, ro.term_name, ru.term_name,
           uk.term_name, ca.term_name, de.term_name,
           pt.term_name, fr.term_name, count(*) AS num
    FROM translation
        LEFT JOIN term AS en ON translation.en = en.term_id
        LEFT JOIN term AS el ON translation.el = el.term_id
        LEFT JOIN term AS es ON translation.es = es.term_id
        LEFT JOIN term AS it ON translation.it = it.term_id
        LEFT JOIN term AS ro ON translation.ro = ro.term_id
        LEFT JOIN term AS ru ON translation.ru = ru.term_id
        LEFT JOIN term AS uk ON translation.uk = uk.term_id
        LEFT JOIN term AS ca ON translation.ca = ca.term_id
        LEFT JOIN term AS de ON translation.de = de.term_id
        LEFT JOIN term AS pt ON translation.pt = pt.term_id
        LEFT JOIN term AS fr ON translation.fr = fr.term_id
    WHERE
        (en.term_name, en.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (el.term_name, el.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (es.term_name, es.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (it.term_name, it.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (ro.term_name, ro.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (ru.term_name, ru.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (uk.term_name, uk.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (ca.term_name, ca.language_id) IN (SELECT term_name, language_id
```

```
    FROM duplicates) OR
        (de.term_name, de.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (pt.term_name, pt.language_id) IN (SELECT term_name, language_id
    FROM duplicates) OR
        (fr.term_name, fr.language_id) IN (SELECT term_name, language_id
    FROM duplicates)
    GROUP BY (en.term_name, el.term_name, es.term_name,
           it.term_name, ro.term_name, ru.term_name,
           uk.term_name, ca.term_name, de.term_name,
           pt.term_name, fr.term_name)
    ) AS duplicated_rows
```

Listing 8: Query per conteggio di traduzioni duplicate

```
CREATE TABLE language(
    language_id CHAR(2) PRIMARY KEY,
    language_name TEXT
);

CREATE TABLE project(
    project_id CHAR(4) PRIMARY KEY
);

CREATE TABLE subproject(
    subproject_id VARCHAR(50) PRIMARY KEY,
    project_id    CHAR(4),
    FOREIGN KEY (project_id) REFERENCES project(project_id)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);

CREATE TABLE domain(
    domain_id VARCHAR(50) PRIMARY KEY
);

CREATE TABLE subdomain(
    subdomain_id VARCHAR(50),
    domain_id VARCHAR(50),
    PRIMARY KEY(subdomain_id, domain_id),
    FOREIGN KEY (domain_id) REFERENCES domain(domain_id)
        ON UPDATE CASCADE
```

```
        ON DELETE NO ACTION
);

CREATE TABLE term(
    term_id                              SERIAL PRIMARY KEY,
    term_name                            VARCHAR(100) NOT NULL,
    language_id                          CHAR(2) NOT NULL,
    part_of_speech                       TEXT NOT NULL,
    term_notes                           TEXT,
    iate_id                              INTEGER,
    term_reference                       TEXT NOT NULL,
    definition                           TEXT NOT NULL,
    definition_external_cross_reference  TEXT,
    definition_source                    TEXT NOT NULL,
    definition_notes                     TEXT,
    domain_id                            VARCHAR(50) NOT NULL,
    subdomain_id                         VARCHAR(50),
    context                              TEXT NOT NULL,
    context_external_cross_reference     TEXT,
    context_source                       TEXT NOT NULL,
    subproject_id                        VARCHAR(50) NOT NULL,
    FOREIGN KEY (subproject_id) REFERENCES subproject(subproject_id)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    FOREIGN KEY (language_id) REFERENCES language(language_id)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    FOREIGN KEY (domain_id) REFERENCES domain(domain_id)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    FOREIGN KEY (subdomain_id, domain_id) REFERENCES subdomain(subdomain_id,
     domain_id)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);

CREATE TABLE translation(
    translation_id SERIAL PRIMARY KEY,
    en INTEGER,
    el INTEGER,
    es INTEGER,
```

```
    it INTEGER,
    ro INTEGER,
    ru INTEGER,
    uk INTEGER,
    ca INTEGER,
    de INTEGER,
    pt INTEGER,
    fr INTEGER,
    FOREIGN KEY (en) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (el) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (es) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (it) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (ro) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (ru) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (uk) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (ca) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (de) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (pt) REFERENCES term(term_id)
        ON DELETE SET NULL,
    FOREIGN KEY (fr) REFERENCES term(term_id)
        ON DELETE SET NULL
);

CREATE TABLE synonym(
    term_id    INTEGER,
    synonym_id VARCHAR(100),
    PRIMARY KEY (term_id, synonym_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

```

```
CREATE TABLE quasi_synonym(
    term_id          INTEGER,
    quasi_synonym_id VARCHAR(100),
    PRIMARY KEY (term_id, quasi_synonym_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE common_name(
    term_id        INTEGER,
    common_name_id VARCHAR(100),
    PRIMARY KEY (term_id, common_name_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE scientific_name(
    term_id            INTEGER,
    scientific_name_id VARCHAR(100),
    PRIMARY KEY (term_id, scientific_name_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE orthographic_variant(
    term_id                INTEGER,
    ortographic_variant_id VARCHAR(100),
    PRIMARY KEY (term_id, ortographic_variant_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE acronym(
    term_id    INTEGER,
    acronym_id VARCHAR(100),
    PRIMARY KEY (term_id, acronym_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
```

```
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE abbreviation(
    term_id         INTEGER,
    abbreviation_id VARCHAR(100),
    PRIMARY KEY (term_id, abbreviation_id),
    FOREIGN KEY (term_id) REFERENCES term(term_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE INDEX term_term_name_index
    ON term(term_name);
CREATE INDEX translation_en_index
    ON translation(en);
CREATE INDEX translation_el_index
    ON translation(el);
CREATE INDEX translation_es_index
    ON translation(es);
CREATE INDEX translation_it_index
    ON translation(it);
CREATE INDEX translation_ro_index
    ON translation(ro);
CREATE INDEX translation_ru_index
    ON translation(ru);
CREATE INDEX translation_uk_index
    ON translation(uk);
CREATE INDEX translation_ca_index
    ON translation(ca);
CREATE INDEX translation_de_index
    ON translation(de);
CREATE INDEX translation_pt_index
    ON translation(pt);
CREATE INDEX translation_fr_index
    ON translation(fr);

INSERT INTO language VALUES('en', 'English'),
                        ('el', 'Greek'),
                        ('es', 'Spanish'),
```

```
                        ('it', 'Italian'),
                        ('ro', 'Romanian'),
                        ('ru', 'Russian'),
                        ('uk', 'Ukrainian'),
                        ('ca', 'Catalan'),
                        ('de', 'German'),
                        ('pt', 'Portuguese'),
                        ('fr', 'French');

INSERT INTO project VALUES('ENVI');
INSERT INTO subproject VALUES('Climate change',  'ENVI');
INSERT INTO subproject VALUES('Zoology', 'ENVI');

INSERT INTO domain VALUES('Environment');
INSERT INTO subdomain VALUES('climate change', 'Environment'),
                        ('biochemistry', 'Environment'),
                        ('pollution', 'Environment'),
                        ('chemistry', 'Environment'),
                        ('biology', 'Environment'),
                        ('zoology', 'Environment'),
                        ('sea', 'Environment'),
                        ('green finance', 'Environment'),
                        ('atmosphere', 'Environment'),
                        ('degradation of the environment', 'Environment'),
                        ('international relations', 'Environment'),
                        ('radiation', 'Environment'),
                        ('construction and town planning', 'Environment'),
                        ('finance', 'Environment'),
                        ('meteorology', 'Environment'),
                        ('processes', 'Environment'),
                        ('forestry', 'Environment'),
                        ('natural enviroment', 'Environment'),
                        ('water', 'Environment'),
                        ('water pollution', 'Environment'),
                        ('earth sciences', 'Environment');
```

Listing 9: comandi SQL per creazione database. Alcuni valori di subdomain sono stati omessi causa caratteri particolari non renderizzabili

```
"""
```

```
parses one file at a time. Arguments required: 1. path to the file   2. ISO 639-1 language code  3. subproject  4. domain
For example: python termsParser.py csv/ClimateChangeEN EN "Climate change" Environment
"""

import pandas as pd
from dataclasses import dataclass, fields
import sys
import psycopg2
import re
import copy

# splits concatenated_string on separator and strips the resulting values of whitespaces
def split(concatenated_string, separator):
    split_result = re.split(separator, re.sub(r'\s+', ' ', concatenated_string))
    return [item.strip() for item in split_result if item.strip()]

@dataclass
class Term():
    term_name: str = None
    language_id: str = None
    part_of_speech: str = ''
    term_notes: str = None
    iate_id: str = None
    term_reference: str = ''
    definition: str = ''
    definition_external_cross_reference: str = None
    definition_source: str = ''
    definition_notes: str = None
    domain_id: str = ''
    subdomain_id: str = None
    context: str = ''
    context_external_cross_reference: str = None
    context_source: str = ''
    subproject_id: str = ''

# generic class for synonyms, quasi_synonyms, common_names...
@dataclass
```

```
class Variant():
    term_name: str = None
    variant: str = None

filename = sys.argv[1]
language = sys.argv[2]
subproject = sys.argv[3]
domain = sys.argv[4]

# returns a list of terms and a dictionary of lists containing variants of the terms
def parseTable(dataframe):
    terms = []
    variants = {'synonym': [], 'quasi_synonym': [], 'common_name': [], 'scientific_name': [], 'orthographic_variant': [], 'acronym': [], 'abbreviation': []}

    for index, entry in dataframe.iterrows():
        term = Term()

        # loop through all possible columns of Term
        for field in fields(Term):
            # special values given by program arguments
            if field.name in ['language_id', 'subproject_id', 'domain_id']: continue

            # if dataframe does not have the column then the default value for term remains
            try: value = entry[field.name]
            except KeyError: continue

            # if value is nan or '-' then term retains the default
            if pd.notna(value) and value != '-':
                # remove leading and trailing whitespaces and newlines
                value = re.sub(r'\s+', ' ', str(value)).strip()
                # if column is 'iate_id' then convert first to integer
                if field.name == 'iate_id':
                    setattr(term, field.name, str(int(float(value)))) # cannot convert float string to int directly
                elif field.name == 'subdomain_id':
                    setattr(term, field.name, value.lower())
```

```
                else:
                    setattr(term, field.name, value)

        # if term_name was null then don't even insert it into database
        if pd.isna(term.term_name): continue

        term.language_id = language
        term.subproject_id = subproject
        term.domain_id = domain

        # if term_name were multiple values separated by ';' create multiple terms and append to terms
        new_terms = []
        for item in split(term.term_name, ';|,|/'):
            new_term = copy.copy(term)
            new_term.term_name = item.lower()
            new_terms.append(new_term)
        if len(new_terms) > 1:
            print(f'line {index} has {len(new_terms)} separated values')
        terms.extend(new_terms)

        # parsing variants' fields
        for key in variants.keys():
            try: value = entry[key]
            except KeyError: continue
            if pd.notna(value):
                # split variants separated by commas, forward slashes and digit+period
                for term in new_terms:
                    for variant in split(value, ',|/|[0-9]\.'):
                        variants[key].append(Variant(term.term_name, variant))

    return terms, variants

df = pd.read_csv(filename)

# replaces single quotes with two single quotes so postgresql escapes them
stringify = lambda x: "'" + x.replace("'", "''") + "'"

# insert parsed data into database
```

```
with psycopg2.connect('dbname=yourterm user=postgres password=password') as conn:
    with conn.cursor() as cur:
        terms, variants = parseTable(df)
        for term in terms:
            values = ', '.join(['NULL' if getattr(term, field.name) is None else stringify(getattr(term, field.name)) for field in fields(Term)])
            try:
                cur.execute('INSERT INTO term VALUES(DEFAULT, {})'.format(values))
                conn.commit()
            except psycopg2.Error as err:
                print(err)
                conn.rollback()
                continue
        for key, variant_list in variants.items():
            for variant in variant_list:
                try:
                    cur.execute('SELECT term_id from term WHERE term_name = {} AND language_id = \'{}\''.format(stringify(variant.term_name), language))
                    term_ids = cur.fetchall()
                    for term_id in term_ids:
                        cur.execute('INSERT INTO {} VALUES({}, {})'.format(key, term_id[0], stringify(variant.variant)))
                        conn.commit()
                except psycopg2.Error as err:
                    print(err)
                    conn.rollback()
                    continue
```

Listing 10: Script per parsing file di termini

```
"""
parses 'all languages' csv file. Argument required: path to file.
For example: python translationsParser.py csv/ClimateChangeAllLanguages.csv
"""

import pandas as pd
import sys
from dataclasses import dataclass, fields
import psycopg2
```

```
import re
from itertools import product
from math import prod

@dataclass
class Translation():
    en: str = None
    el: str = None
    es: str = None
    it: str = None
    ro: str = None
    ru: str = None
    uk: str = None
    ca: str = None
    de: str = None
    pt: str = None
    fr: str = None

# splits concatenated_string on separator and strips the resulting values of whitespaces
def split(concatenated_string, separator):
    split_result = re.split(separator, re.sub(r'\s+', ' ', concatenated_string))
    return [item.strip() for item in split_result if item.strip()]

def parseTable(dataframe):
    for index, entry in dataframe.iterrows():
        translation = Translation()
        for field in fields(Translation):
            try: value = entry[field.name]
            except KeyError: continue
            if pd.notna(value):
                value = re.sub(r'\s+', ' ', value).strip().lower()
                setattr(translation, field.name, value)

        # case fields are ; separated values
        values = [split(str(getattr(translation, field.name)), ';|,|/') for field in fields(Translation)]
        combination_number = prod([len(value) for value in values])
        if combination_number > 1: print(f'line {index} has {combination_number} combinations')
```

```
47          for combination in product(*values):
48              yield Translation(*combination)
49
50  # returns the list of all term_id that have the term_name == term and the
        specified language
51  # if there is no corrispondence, [None] is returned
52  def generate_term_ids(cur, term, language):
53      term_ids = []
54      if term is not None:
55          cur.execute('SELECT term_id from term WHERE term_name = {} AND
        language_id = \'{}\''.format(stringify(term), language))
56          for id in cur.fetchall():
57              term_ids.append(id[0])
58      if len(term_ids) == 0:
59          term_ids.append(None)
60      return term_ids
61
62  filename = sys.argv[1]
63
64  df = pd.read_csv(filename)
65
66  # replaces single quotes with two single quotes so postgresql escapes them
67  stringify = lambda x: "'" + x.replace("'", "''") + "'"
68
69  with psycopg2.connect('dbname=yourterm user=postgres password=password') as
        conn:
70      with conn.cursor() as cur:
71          for translation in parseTable(df):
72              try:
73                  # ids contains all the lists corresponding to the term_ids
        of each term in a translation
74                  ids = []
75                  for field in fields(Translation):
76                      ids.append(generate_term_ids(cur, getattr(translation,
        field.name), field.name))
77                  for combination in product(*ids):
78                      if not all([item is None for item in combination]):
79                          values = ', '.join(['NULL' if id is None else str(id
        ) for id in combination])
80                          cur.execute('INSERT INTO translation VALUES(DEFAULT,
         {})'.format(values))
```

```
81                          conn.commit()
82              except psycopg2.Error as err:
83                  conn.rollback()
84                  print(err)
85                  continue
```

Listing 11: Script per parsing file di traduzioni