Publications Project

San José State University

Charles W. Davidson College of Engineering

CMPE 138 Database Systems I

Section 01

Instructor: Professor David C. Anastasiu

Alex Richards

Thursday, April 27th, 2017

Publications Project

Publications Parsing

The publications preprocessing happens in three phases. The first phase begins by removing all the *<pub>*, *<authors>*, and *</authors>* tags and splitting the file on *</pub>* tags to form a list of records. The second phase takes as input this list of partially processed publications and builds a list of dictionaries; one dictionary for each record. Finally, the third phase cleans up the broken records that cannot be processed in the previous step. The third phase is necessary because many of the records have artifacts within the title which prevent them from adhering to the expected structure of a record. Generally speaking, subscript and superscript tags form these artifacts. In order to facilitate more efficiency on subsequent executions of 'create.py', the script first checks to see if the serialized file of this finalized list of records, 'pubs.dat', exists in the project folder already and loads it if so.

Database Creation

After creating or loading the intermediary data structure of records, three sqlite3 tables are created: *publication*, *author*, and *written_by*. The latter is a relation that maps many authors to many publications.The *publication* table has a primary key *id* which is an integer, *title* which is a variable character type of length 400, *booktitle* which is a variable character type of length 150, *pages* which is a variable character type of length 50, and *year* which is an integer. The *publication* relation has the following constraints: *title* must be NOT NULL, and the year must be between 1835 and the current year inclusive. I chose 1835 as the lower bound for a publication's year because I thought Charles Babbage's paper "*On the Economy of Machinery and Manufactures*" was an appropriate temporal bookend for publications in computer engineering. The *author* relation has a primary key *id* which is an integer and *name* which is a variable character type of length 50. The *author* relation has a NOT NULL constraint on *name*. *Written_by* is a many-to-many relationship between *publication* and *author* that has no additional attributes. There are ON DELETE CASCADE foreign key constraints on *pub_id* and *author_id*. A simple Entity Relationship diagram is included below to illustrate the relationship between the tables created for the database.

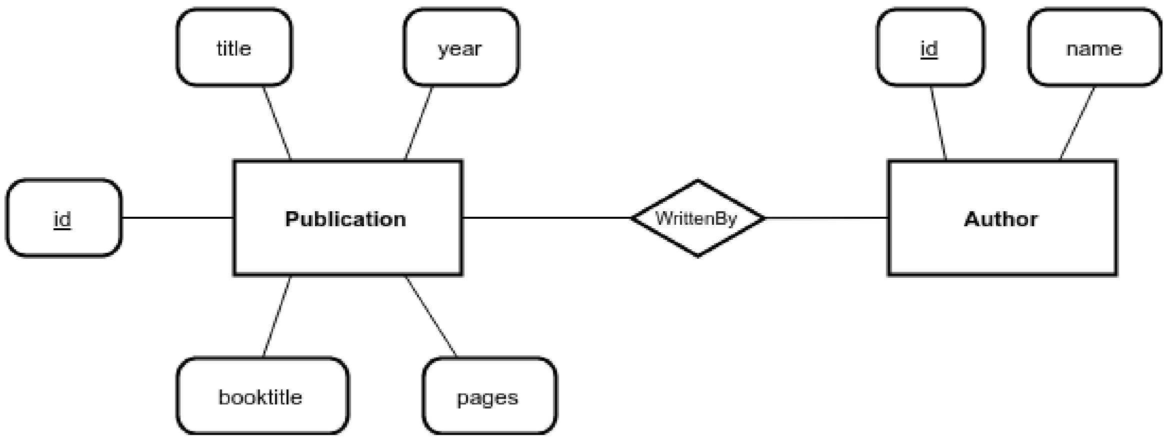

Figure 1: *Entity Relationship diagram of the Publications* application

Inserting records into the three relations is as simple as looping through all the records and creating an insert SQL string from each record's attributes. Because each record may have multiple authors, a loop is used to iterate them. Inside this loop, a dictionary of authors is used to look up if the current record's author has been seen yet. If not, an insert SQL string is created for the *author* relation. At the end of each iteration through the authors, an insert SQL string is created to enter a row into *written_by*. In order to facilitate more efficiency on subsequent executions of 'create.py', the script first checks to see if the database file, 'database.db', exists in the project folder already and notifies the user of this before exiting.

Notes on the Efficiency of create.py

The average time it takes to create the database is 19.54 seconds. When running the script using Python 3, the average time it takes to create the database is 15.97 seconds. This difference is not related to how fast Python 2.7 or Python3 executes Python or SQLite functions -- their elapsed times are approximately the same using either kernel -- but is because of the differences between the two versions' implementation of the cPickle module which the script makes use of. Initially, I had kept track of new authors in a list and used their index as an ID -- this, of course, led to a dramatic inefficiency in the innermost loop of 'create.py'. The average time it took to create the database when storing authors in a list was approximately 14 minutes. The time complexity with this inefficiency was $O(P \times A)$ where P is the number of publications and A is the number of authors. By using a dictionary to store authors and having that author's associated value be an incremented counter representing their *id*, the time complexity of the algorithm is reduced to $O(P)$ since the dictionary allows for an amortized constant time lookup of authors.

Services Provided

The wrapper for the database, 'PublicationAPI.py' is a class that initializes by connecting to 'database.db' with a parameter string to the database file. Included in the wrapper class are functions for inserting a new publication, updating an author's name, updating the title, year, or journal name of a publication, deleting an author by name (exact or fuzzy matching), deleting a publication by title, year, and journal name, and querying publications using author, title, year, or journal.

The class has an instance variable representing the current year so that insert publication operations may check if the submission year is valid (between 1835 and the current year inclusive). The insert publication operation also cleans input of double quotes (") by replacing them with single quotes (') before entering the publication and its authors into the *publication* and *written_by* relations respectively. The insert publication operation also checks if each author of the publication to be inserted has been added to the *author* relation and adds them if they are not present.

Deleting an author simply requires supplying the author's name. The default behavior of this operation is only match authors with the provided name exactly, but setting *exact=False* as a parameter will allow fuzzy matching. This will delete all authors who have the space-delimited substrings of the parameter name inside their own names. For example, *blahDavidblahCblahAnastasiu* would be matched for the input *David C. Anastasiu*. If the parameter name can be split into three substrings (meaning a middle name/initial was provided), then the middle value is mutated into just the first letter of that value. This is done to remove the '.' that may or may not have been included in the parameter in order to match entries that either do not have a '.' or have the entire middle name explicitly entered. Deleting a publication is done by exactly matching parameter values for title, year, and journal name. There is no parameter for author name because the delete operation removes by publication ID which is distinct for publications with multiple authors.

Updating an author's name can be done by simply supplying the name to match exactly and the name that is to replace it. Updating a publication involves two parameter lists, each with values for title, year, and journal. The first list is the matching criteria -- all of which must be supplied and is matched exactly. The second list can be have empty strings in it (or *None* in the case of the year parameter), so that user's may choose to update either the title, year, or journal, or any combination of these attributes.

Querying publications can be done by supplying a list parameter with values for author, title, year, and journal. None of these have to be provided (can use empty strings or *None* in the case of year), but then all publications will be retrieved. There are additional parameters with default values; these include *exact* which allows for fuzzy searching similar to the delete author operation, *output_format* which allows for specifying JSON or XML payloads, *sorted_order* which allows results to be sorted by title, year, journal, or author, *reverse* which will set the sorted order to be in either ascending or descending order, and *queryRange* which sets the offset and query volume. Generally, results are distinct meaning that publications with multiple authors appear on one line with all the authors listed as a single attribute. However, when *sorted_by=name*, the results are a multiset with multiple records for the same publication if that publication has multiple authors. This is done so that every publication and author combination is visible and can be sorted independently.