Python Package Metadata Management

Basic Databases

by Nguyễn Gia Phong, Nguyễn Quốc Thông,
Nguyễn Văn Tùng and Trần Minh Vương

July 7, 2020

Contents

1 Introduction

1.1 Brief Description

In traditional Unix-like operating systems like GNU/Linux distributions and BSD-based OSes, package managers tries to synchronize the packages metadata (such as available versions and dependencies) with that of central repositories. While this proves to be reliable and efficient, language-specific package managers do not usually have such synchronized databases, since they focus on development libraries which have more flexible contraints.

Within the Python packaging ecosystem, this is the case, where package managers like `pip` needs to fetch metadata of each package to be installed to find out dependencies and other information. This turns out to have heavy performance penalty on the dependency resolution process alone, which is already a NP-hard problem. This project explores ways to store these metadata in an efficient in a database, to be used in practice either locally or in a local/regional network, to avoid Python package managers from having to fetch (and potentially build) entire packages just to find out if it conflicts with others.

1.2 Authors and Credits

The work has been undertaken by group number 8, whose members are listed in the following table.

Full name	Student ID
Nguyễn Gia Phong	BI9-184
Nguyễn Quốc Thông	BI9-214
Nguyễn Văn Tùng	BI9-229
Trần Minh Vương	BI9-239

This report is licensed under a CC BY-SA 4.0 license, while the source code is available on GitHub[*] under AGPLv3+.

We would like to express our special thanks to Dr. Nguyễn Hoàng Hà, whose lectures gave us basic understanding on the key principles of relational databases. In addition, we also recieved a lot of help from the Python packaging community over #pypa on Freenode on understanding the structure of the metadata as well as finding a way to fetch these data from package indices.

[*] `https://github.com/McSinyx/cheese-shop`

2 User Requirements

This project aims to provide a database for metadata queries and Python packages exploration. We try to replicate the PyPI's XML-RPC API [1], which supports queris similar to the following:

- `list_projects()`: Retrieve a list of registered project names.
- `project_releases(project)`: Retrieve a list of releases for the given `project`, ordered by version.
- `project_release_latest()`: Retrieve the latest release of the given `project`.
- `belong_to(name)`: Retrieve a list of projects whose author is `name`.
- `browse(classifier)`: Retrieve a list of (`project`, `version`) of all releases classified with all of the given classifier.
- `release_data(project, version)`: Retrieve the following metadata matching the given release: project, version, homepage, author, author's email, summary, license, keywords, classifiers and dependencies
- `search_name(pattern)`: Retrieve a list of (`project`, `version`, `summary`) where the project name matches the pattern.
- `search_summary(pattern)`: Retrieve a list of (`project`, `version`, `summary`) where the summary matches the pattern.

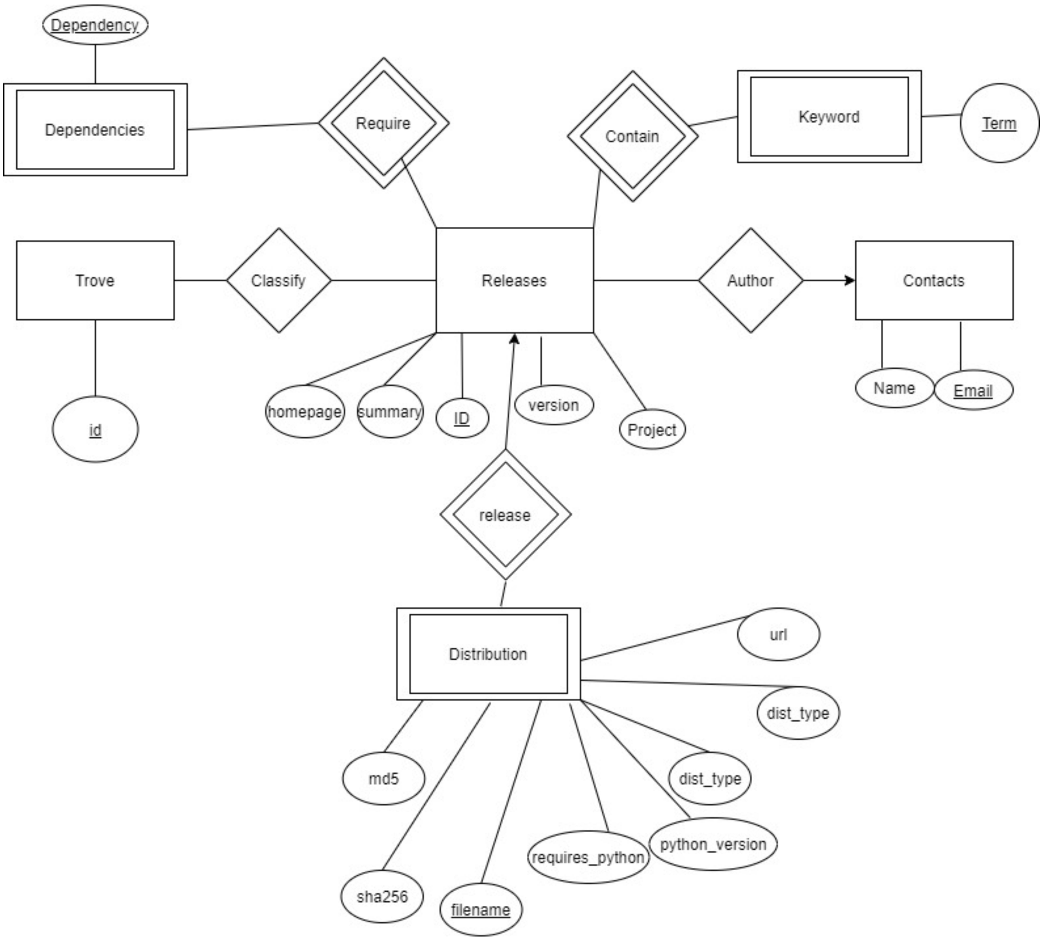
3 Data Definition

3.1 Entity Relationship Diagram

The entity relationship diagram represents the relationship between each of its entity set of data extracted from projects:

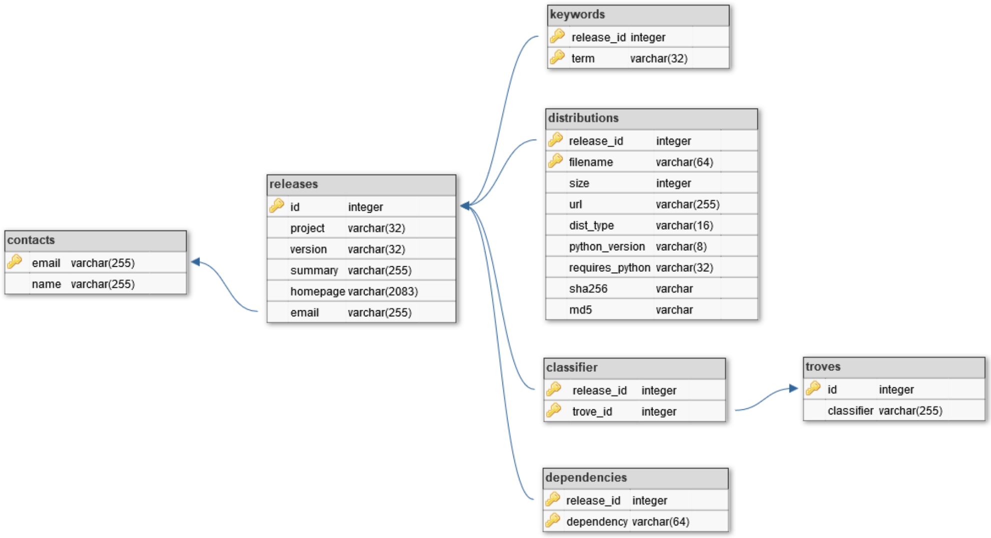
- Author(Releases-Contact: Many-One): Within each release, there could be one author, due to data extraction method doesn't support multi-author. Yet an author could have multiple releases under per name.
- Require(Releases-Dependencies: Many-Many): Every release would require a number of dependencies, and many dependencies can each be used by multiple releases.

- Classify(Releases-Trove: Many-Many): This relationship indicates the relationship between trove classifier and each releases, with many release could be classified under one trove classifier, and a release could be classified by many classifiers.
- Contain(Releases-Keyword: Many-Many): A release has many keywords, and also a keyword can also be in many different releases.
- Release(Releases-Distribution: One-Many): Within each releases, a number of distribution(s) would be released. A distribution could relate to only one releases, but many distributions could be released in the same releases.



3.2 Database Schema

Based on the entity relationship diagram, we worked out a schema complying with the third normal form [2].



contacts(email, name) Contact information of an author, including per email as the primary key and per name.

releases(id, project, version, summary, homepage, email) This relation represents each release of a project, including its name, version, summary, homepage and the email of its author. The ID of each release is the primary key to represent each one of them. This release ID is also the foreign key of many primary key in other entity set.

troves(id, classifier) Valid trove classifiers, identified by their ID.

classifiers(release_id, trove_id) Release ID and corresponding trove classifiers ID the release is classified by.

keywords(release_id, term) Keywords of a specific release. Both the ID of the release and the keyword are set as primary key.

dependencies(release_id, dependency) This relation represents the dependency list of each release, which is a pattern can be matched by a release of another project.

distributions(<u>release_id</u>, <u>filename</u>, size, url, dist_type, python_version, requires_python, sha256, md5) Each distribution (i.e. the file that the package manager can use to install) and the corresponding url, checksums and other auxiliary information.

4 Data Query

4.1 Project Listing

Retrieve a list of the project names registered with the project index.

4.2 Project Releases

Retrieve a list of the releases registered for the given project name, ordered by version.

4.3 Users

Retrieve a list of role, user for a given project name.

4.4 Release URLs

Retrieve a list of download URLs for the given release version.

4.5 Release Data

retrieve metadata describing a specific release version.

4.6 Classifiers

Retrieve a list of name, version of all releases classified with all of the given classifiers, classifiers must be a list of Trove classifier strings.

5 Conclusion

6 References

[1] The Python Packaging Authority. *PyPI's XML-RPC methods*. Warehouse documentation.

[2] Edgar F. Codd. *Further Normalization of the Data Base Relational Model*. IBM Research Report RJ909, August 31, 1971.