Python Package Metadata Management

Basic Databases

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

July 8, 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

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 the given classifier.
- `release_data(project, version)`: Retrieve the following metadata matching the given release: project, version, homepage, author, author's email, summary, 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.

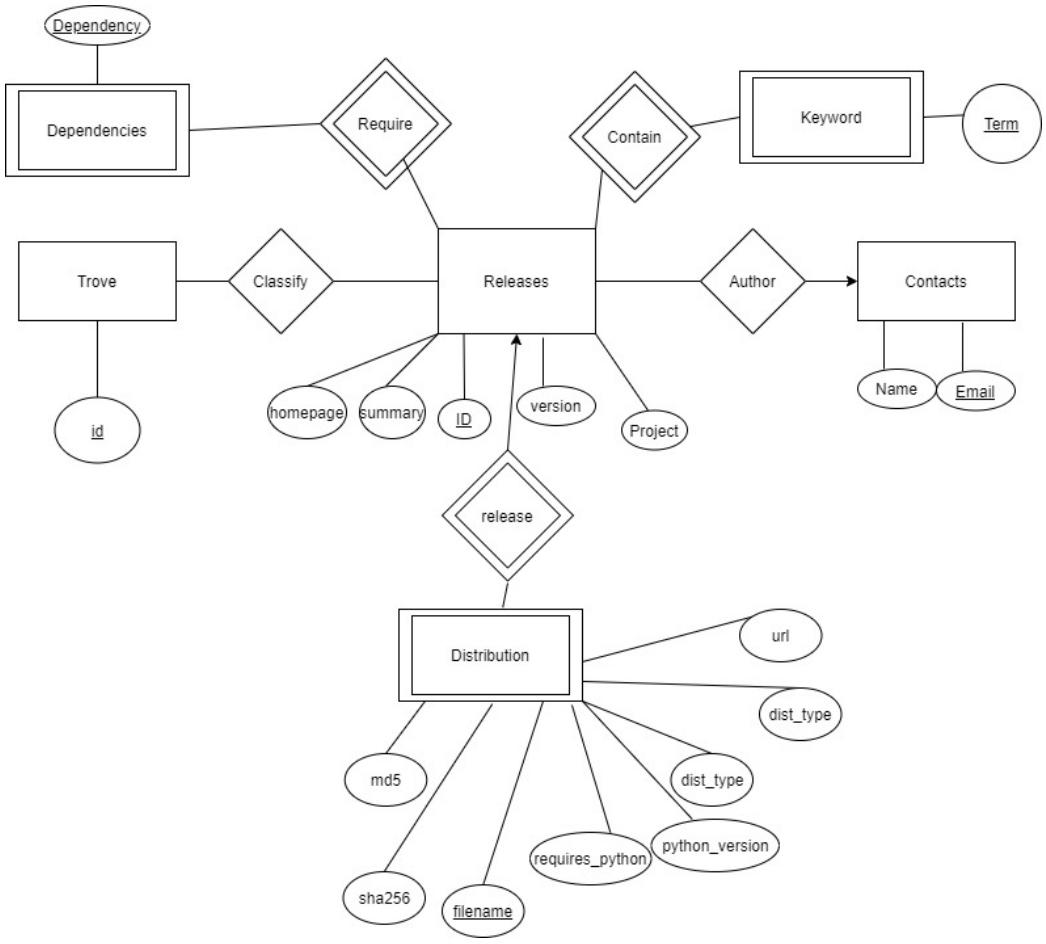
3 Data Definition

3.1 Entity Relationship Diagram

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

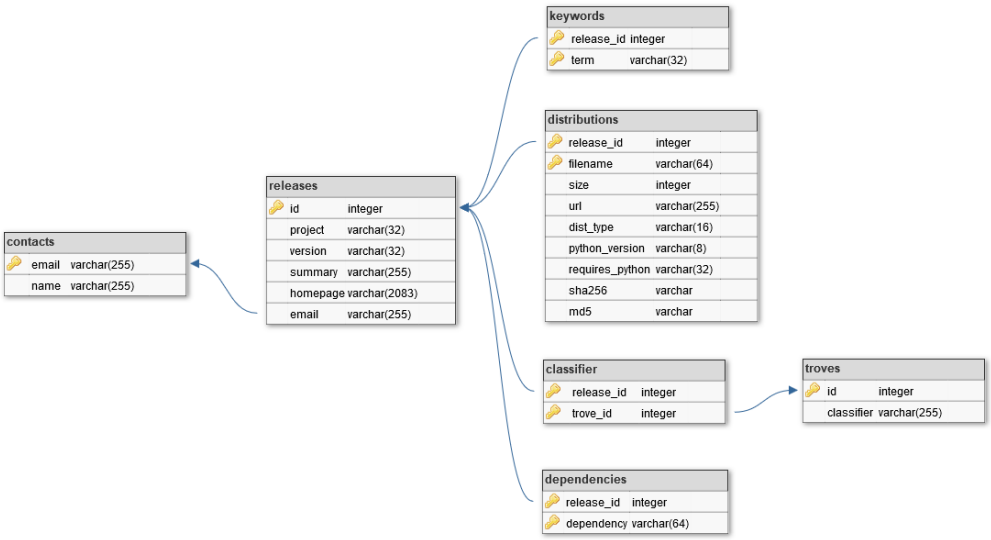
- Author(Releases-Contact: Many-One): Within each release, there could be one author, due to data extraction method doesn't support multi-author. Yet an author could have multiple releases under per name.
- Require(Releases-Dependencies: Many-Many): Every release would require a number of dependencies, 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]. Concrete definitions can be found in `sql/def.sql`.



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(release_id, filename, 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.

With the database defined, we then fetch the metadata of around 100 most popular projects on PyPI. The code used to do this is available in `tools/make-cheeses.py`. More instructions on the setup can be found in the project's README.

4 Data Query

In addition to primary keys and unique attributes which are implicitly [3], to further optimize, we index `contacts.name` and `releases.summary`. Then, the tasks defined in section 2 can be carried out as follows.

4.1 Project Listing

To retrieve a list of project names, we can use the following SQL query

```
SELECT DISTINCT project
FROM releases;
```

4.2 Project Release Listing

To retrieve a list of releases of project `spam`, ordered by version:

```
SELECT version
FROM releases
WHERE project = 'spam'
ORDER BY version;
```

4.3 Project Latest Release

To retrieve the latest version of the project `spam`:

```
SELECT version
FROM releases
```

```
WHERE project = 'spam'
ORDER BY version DESC
LIMIT 1;
```

4.4 Project Listing by Author

For convenience purposes, we create a view of `authorship`:

```
CREATE VIEW authorships
AS SELECT name as author, project
FROM contacts NATURAL JOIN releases
GROUP BY author, project;
```

A list of projects authored by Monty Python is then as easy to obtain as

```
SELECT project
FROM authorships
WHERE author='Monty Python';
```

4.5 Browse by Classifier

Since the query is non-trivial, we define a procedure for convenient use:

```
DELIMITER //
CREATE PROCEDURE browse(class varchar(255))
BEGIN
  SELECT project, version
  FROM releases, classifiers
  WHERE id = release_id AND trove_id = (
    SELECT id
    FROM troves
    WHERE classifier = class);
END//
DELIMITER ;
```

For instance, listing releases classified as `Programming Language :: C` will be as simple as `CALL browse('Programming Language :: C');`

4.6 Release Metadata

We also define a stored procedure for this task, which seems to be even more complex. The procedure can then be used in similar manner, e.g. `CALL release_data('udata', '1.1.1');`

```
DELIMITER //
CREATE PROCEDURE release_data(project varchar(32),
                              version varchar(32))
BEGIN
  DECLARE i smallint;
  SET i = (
    SELECT id
    FROM releases
    WHERE releases.project = project AND releases.version = version);
  SELECT project, version, homepage, name as author, email, summary
  FROM releases NATURAL JOIN contacts
  WHERE id = i;

  SELECT term as keyword
  FROM keywords
  WHERE release_id = i;

  SELECT classifier
  FROM classifiers, troves
  WHERE release_id = i AND trove_id = troves.id;
END//
DELIMITER ;
```

4.7 Project Search by Name

To retrieve project by name matching the SQL pattern `py%`

```
SELECT project, version, summary
FROM releases
WHERE project LIKE 'py%';
```

4.8 Search project name by summary

To retrieve project by summary matching the SQL pattern `%num%`

```
SELECT project, version, summary
FROM releases
WHERE summary LIKE '%num%';
```

5 Conclusion

Working on this project has been a great opportunity for us to and apply our knowledge in relational databases in real-life tasks. During the design and implementation process, we gained a better concept about relational databases, as well as how to build from ground up and use a database in SQL (MySQL in particular), along with abstractions and optimizations.

We also had a chance to take a closer look at the Python Package Index and at how their packages' metadata are used and distibuted. Through this project, we think we have reached closer to the final goal of metadata mirroring and synchronization.

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.

[3] MySQL 8.0 Reference Manual. *Oracle*. Section 8.3.1: *How MySQL Uses Indexes*.